



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΕΙΔΙΚΕΥΣΗΣ στην “Μηχανική Λογισμικού για
Διαδικτυακές & Φορητές Εφαρμογές”

Σχεδίαση και ανάπτυξη εφαρμογής λογισμικού Android με
ενσωματωμένη διασύνδεση Back-end μέσω RESTful API

Κουρνούτος Φοίβος - Βασιλειάδης Χρήστος

Επιβλέπων Καθηγητής:
Δρ. Γεώργιος Κακαρότζας
Τμήμα Ψηφιακών Συστημάτων
Πανεπιστήμιο Θεσσαλίας

Λάρισα, Μάρτιος 2024



*Αφιερωμένο στην οικογένεια μου, στο μέλλον και στην ελπίδα που φέρνει αυτή η
εργασία...*



Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι να παρουσιάσει την ανάπτυξη μιας εφαρμογής παρακολούθησης συνηθειών “Habit Tracker” που αποσκοπεί στη διευκόλυνση της καταγραφής, παρακολούθησης και προώθησης θετικών συνηθειών.

Η εφαρμογή Habit Tracker αποσκοπεί στη διαχείριση των ημερήσιων συνηθειών, παρέχοντας μια εύχρηστη πλατφόρμα για τους χρήστες. Οι χρήστες μπορούν να δημιουργήσουν, να παρακολουθήσουν και να διατηρήσουν θετικές συνήθειες στην καθημερινή τους ζωή. Η εφαρμογή επιτρέπει την καταγραφή στόχων, την παρακολούθηση της προόδου, την προσωποποίηση των ειδοποιήσεων και τη διαχείριση των στόχων με αναλυτικές αναφορές. Προσφέρει επίσης επιπλέον λειτουργίες όπως επανεκκίνηση στόχων, προσαρμογή προτεραιοτήτων, τροποποίηση συνηθειών και ενίσχυση της αυτο-πειθαρχίας.

Για την ανάπτυξη της εφαρμογής, χρησιμοποιήθηκαν ποικίλες τεχνολογίες. Στο μέρος του front-end, η εφαρμογή έχει αναπτυχθεί στο Android Studio χρησιμοποιώντας τη γλώσσα προγραμματισμού Java. Η υλοποίηση του back-end έγινε με τη χρήση Django και γλώσσας προγραμματισμού Python. Επιπλέον, για τη φιλοξενία του back-end της εφαρμογής, χρησιμοποιήθηκε η υπηρεσία Amazon EC2 της Amazon Web Services (AWS). Η αποθήκευση δεδομένων πραγματοποιήθηκε σε μια βάση δεδομένων PostgreSQL, ενώ η υιοθέτηση των αρχών του GRASP στον σχεδιασμό συνέβαλε στη δημιουργία ενός σταθερού, ασφαλούς και αποδοτικού συστήματος.

Η εφαρμογή αναπτύχθηκε για να αντιμετωπίσει προκλήσεις στη διαχείριση του χρόνου, την καθημερινή οργάνωση και την υιοθέτηση θετικών συνηθειών, προσφέροντας ένα εύχρηστο εργαλείο για την αυτό-βελτίωση και την επίτευξη προσωπικών στόχων.

Λέξεις – Κλειδιά

Back End, Βάση δεδομένων, Front End, Rest API, πρότυπα ανάθεσης αρμοδιοτήτων σε κλάσεις λογισμικού (GRASP).



Abstract

The purpose of this bachelor's thesis is to present the development of a habit tracking application "Habit Tracker" that aims to facilitate the recording, monitoring and promotion of positive habits.

The Habit Tracker application aims to manage daily habits by providing an easy-to-use platform for users. Users can create, track and maintain positive habits in their daily lives. The app allows users to record goals, track progress, personalize notifications and manage goals with detailed reports. It also offers additional features such as resetting goals, adjusting priorities, modifying habits and enhancing self-discipline.

A variety of technologies were used to develop the application. On the front-end part, the application has been developed in Android Studio using Java programming language. The back-end implementation has been done using Django and Python. In addition, Amazon EC2 of Amazon Web Services (AWS) has been used to host the back-end of the application. Data storage was performed in a PostgreSQL database, and the adoption of GRASP principles in the design contributed to the creation of a stable, secure and efficient system.

The app was developed to address challenges in time management, daily organization and adopting positive habits, offering an easy-to-use tool for self-improvement and achieving personal goals.

Words - Keywords

Back End, Database, Front End, Rest API, software class delegation standards (GRASP) .



Περιεχόμενα

Περίληψη.....	iii
Abstract.....	iv
1 Εισαγωγή.....	1
2 Σχεδιασμός και ανάλυση των απαιτήσεων της εφαρμογής.....	2
2.1 Καταγραφή Συνηθειών.....	3
2.1.1 Ο πυρήνας της εφαρμογής Habit Tracker.....	3
2.1.2 Λεπτομέρειες Συνηθειών και Ορισμός Στόχων.....	4
2.1.3 Ορισμός Στόχων.....	4
2.1.4 Προσαρμογή Συνηθειών.....	5
2.2 Παραμετροποίηση Στόχων.....	5
2.2.1 Ορισμός Συγκεκριμένων Στόχων.....	6
2.2.2 Προσαρμογή Διάρκειας ή Συχνότητας.....	6
2.2.3 Ευελιξία στον Ορισμό Στόχων.....	7
2.3 Παρακολούθηση Προόδου.....	7
2.3.1 Στατιστικά Στοιχεία.....	8
2.3.2 Ειδοποιήσεις και Υπενθυμίσεις.....	9
2.3.3 Συνεχής Βελτίωση.....	10
2.4 Ειδοποιήσεις και Υπενθυμίσεις.....	10
2.4.1 Εξατομικευμένες Ειδοποιήσεις.....	11
2.4.2 Προσαρμοσμένες Ρυθμίσεις Ειδοποιήσεων.....	12
2.4.3 Υπενθυμίσεις Στόχων.....	12
2.4.4 Ευέλικτη Διαχείριση Υπενθυμίσεων.....	13
2.4.5 Ενίσχυση Αυτό-Πειθαρχίας.....	13
2.5 Επεξεργασία Στοιχείων.....	14
2.5.1 Επεξεργασία Συνηθειών.....	14
2.5.2 Τροποποίηση Στόχων.....	15
2.5.3 Αλλαγές στη Ρουτίνα.....	15
2.5.4 Προσαρμογή Προτεραιοτήτων.....	16
3 Σχεδιασμός και ανάλυση του Front End.....	18
3.1 Σχεδιασμός UI (User Interface).....	19
3.1.1 Διάταξη Οθόνης.....	21
3.1.2 Χρωματική Παλέτα.....	22
3.2 Εμπειρία Χρήστη UX (User Experience).....	22
3.2.1 Διαδραστικότητα.....	24
3.2.2 Απλότητα και Κατανόηση.....	24
3.2.3 Πλοήγηση.....	24
3.3 Διαδικασίες Εισόδου.....	25
3.4 Διαδικασίες Εξόδου.....	26
3.5 Αναγνώριση Σφαλμάτων.....	28
3.6 Αρχαιοθετημένες Συνήθειες.....	29
3.7 Αποστολή Email Καλωσορίσματος στον Νέο Χρήστη.....	31
4 Σχεδιασμός και ανάλυση του Back End.....	34
4.1 API διασύνδεση back-end με front-end.....	35
4.2 Βάση δεδομένων PostgreSQL.....	38
4.3 Django Framework.....	40



4.4	Amazon Elastic Compute Cloud (Amazon EC2)	42
4.5	Σχεδιασμός και ανάλυση της εφαρμογής	43
5	GRASP patterns αρχιτεκτονικά και σχεδιαστικά πρότυπα	46
5.1	Σχέση UML και GRASP	47
5.2	Τι είναι τα πρότυπα	47
5.3	Σχεδιαστικά πρότυπα GRASP	48
5.3.1	Δημιουργός (Creator)	48
5.3.2	Πληροφορημένος Ειδικός (Information Expert)	49
5.3.3	Χαμηλή Σύζευξη (Low Coupling)	50
5.3.4	Ελεγκτής (Controller)	51
5.3.5	Υψηλή Συνοχή (High Cohesion)	52
5.3.6	Πολυμορφισμός (Polymorphism)	53
5.3.7	Καθαρή Επινόηση (Pure Fabrication)	55
5.3.8	Πλαγιότητα (Indirection)	56
5.3.9	Προστατευμένες Παραλλαγές (Protected Variations)	58
5.4	Τα πρότυπα GRASP στην εφαρμογή μας (Habit Tracker app)	60
6	Συμπεράσματα και Μελλοντικές Επεκτάσεις	67
	Βιβλιογραφία	69
	Παράρτημα I (Front End)	71
	Παράρτημα II (Back End)	130

Εικόνες

Εικόνα 1	Διεπαφή χρήστη (User Interface)	20
Εικόνα 2	Σελίδα Εισαγωγής (Intro Page)	23
Εικόνα 3	Σύστημα Σύνδεσης (Login System)	25
Εικόνα 4	Σύστημα Εγγραφής Χρηστή (Sign Up)	26
Εικόνα 5	Διαδικασία Αποσύνδεσης (Log Out)	27
Εικόνα 6	Σφάλμα “Λάθος κωδικός πρόσβασης” κτλ.	29
Εικόνα 7	Ενεργοποιημένη συνήθεια για παρακολούθηση	30
Εικόνα 8	Απενεργοποιημένη συνήθεια μη παρακολούθησης	31
Εικόνα 9	Email που έρχεται στον χρήστη μετά την εγγραφή του	32
Εικόνα 10	Σχεδίαση REST API	36
Εικόνα 11	RestApi urls	36
Εικόνα 12	Front-end αιτήσεις (requests) προς Back-end	37
Εικόνα 13	Django αρχιτεκτονικής MTV	40
Εικόνα 14	Αρχιτεκτονική του Amazon EC2	42
Εικόνα 15	UML βάσης δεδομένων	45
Εικόνα 16	Δημιουργός (Creator) Back-end	61
Εικόνα 17	Πληροφορημένος Ειδικός (Information Expert) Back-end	62
Εικόνα 18	Ελεγκτής (Controller) Back-end	64



Πίνακες

Πίνακας 1 Λειτουργίες HTTP μεθόδων	37
--	----



1 Εισαγωγή

Σκοπός της παρούσας εργασίας είναι να παρουσιάσει την διαδικασία ανάπτυξης μιας εφαρμογής η οποία αποσκοπεί στη διευκόλυνση της καταγραφής, παρακολούθησης και προώθησης θετικών συνηθειών . Η εφαρμογή Habit Tracker προσφέρει μια εξαιρετικά χρήσιμη προσέγγιση για την αντιμετώπιση ενός ευρέως διαδεδομένου προβλήματος: την αποτελεσματική οργάνωση και διατήρηση θετικών συνηθειών στην καθημερινή ζωή. Αυτή η εφαρμογή έχει σχεδιαστεί με βάση τη δημιουργία ενός εργαλείου που ενθαρρύνει τη συνειδητοποίηση και παρακολούθηση των συνηθειών που θέλουν να ενσωματώσουν οι χρήστες στην καθημερινότητά τους.

Η βασική αρχή της εφαρμογής Habit Tracker είναι να ενθαρρύνει τη δημιουργία και διατήρηση θετικών συνηθειών μέσω μιας ενιαίας πλατφόρμας που είναι εύχρηστη και ευχάριστη για τους χρήστες. Οι χρήστες έχουν τη δυνατότητα να προσθέτουν διάφορες συνήθειες που επιθυμούν να εντάξουν στην καθημερινότητά τους, όπως άσκηση, ανάγνωση, ύπνο, υγιεινή διατροφή και άλλες, καταγράφοντας την εκτέλεσή τους και προσδιορίζοντας στόχους σε σχέση με τη συχνότητα ή τη διάρκειά τους.

Μια από τις μεγάλες προκλήσεις που αντιμετωπίζει η εφαρμογή είναι η αποτελεσματική οργάνωση των συνηθειών των χρηστών, καθώς και η ενθάρρυνση της τακτικότητας και της συνέπειας στην τήρησή τους. Η εφαρμογή Habit Tracker προσφέρει μια ευέλικτη πλατφόρμα που επιτρέπει στους χρήστες να προσαρμόζουν τους στόχους και τη συχνότητα των συνηθειών σύμφωνα με τις προσωπικές τους ανάγκες.

Για την ανάπτυξη της εφαρμογής, χρησιμοποιήθηκαν ποικίλες τεχνολογίες. Στο μέρος του front-end, η εφαρμογή έχει αναπτυχθεί στο Android Studio χρησιμοποιώντας τη γλώσσα προγραμματισμού Java. Όσον αφορά το back-end, η υλοποίηση έγινε με Django, με γλώσσα προγραμματισμού Python. Το Django προσφέρει μια ολοκληρωμένη πλατφόρμα ανάπτυξης, ενώ η εκτέλεση του σε έναν Amazon EC2 διακομιστή παρέχει την αξιοπιστία και ευελιξία που απαιτούνται για τη λειτουργία της εφαρμογής. Επιπλέον, η αποθήκευση των δεδομένων σε μια βάση PostgreSQL αποτελεί κρίσιμο μέρος του συστήματος, παρέχοντας ασφάλεια και αποδοτική διαχείριση των πληροφοριών. Επίσης η επικοινωνία του back-end με το front-end να γίνεται μέσω API. Επιπλέον, έχουν ενσωματωθεί διάφορα εργαλεία όπως διάφορα πλαίσια εργασίας, βιβλιοθήκες ή εργαλεία ανάπτυξης για τη βελτιστοποίηση της λειτουργικότητας και της απόδοσης της εφαρμογής. Τέλος εφαρμοστήκαν τα GRASP πρότυπα για τον σχεδιασμό του back-end της εφαρμογής Habit Tracker, για οδηγήσει σε μια καλύτερα οργανωμένη, ευέλικτη και εύκολα συντηρήσιμη δομή του κώδικα, βοηθώντας στην αποτελεσματική λειτουργία της εφαρμογής.

Η εφαρμογή Habit Tracker αποτελεί ένα ισχυρό εργαλείο για τη διαχείριση των καθημερινών συνηθειών, προσφέροντας στους χρήστες τη δυνατότητα να αντιμετωπίζουν αποτελεσματικά τις προκλήσεις της οργάνωσης, της παρακολούθησης και της βελτίωσης των καθημερινών τους συνηθειών.



2 Σχεδιασμός και ανάλυση των απαιτήσεων της εφαρμογής

Η ανάλυση και ο σχεδιασμός των απαιτήσεων είναι ένα σημαντικό βήμα για το πώς οι λειτουργίες της εφαρμογής πληρούν τις ανάγκες των χρηστών σε συγκεκριμένο περιβάλλον χρήσης. [[1]] Συνολικά, ο στόχος είναι μια λεπτομερή περιγραφή των λειτουργιών που παρέχει η εφαρμογή, τις ανάγκες των χρηστών και το περιβάλλον της χρήσης.

Ένα πλαίσιο ανάλυσης απαιτήσεων μπορεί να περιλαμβάνει τα παρακάτω:

Περιγραφή Λειτουργιών της Εφαρμογής: Ανάλυση των λειτουργιών που παρέχει η εφαρμογή. [[2]] Για παράδειγμα, καταγράφοντας τις συνήθειες, τη δυνατότητα επεξεργασίας, την παρακολούθηση προόδου, και οποιεσδήποτε άλλες λειτουργίες ή εργαλεία που παρέχονται.

Απαιτήσεις των Χρηστών: Ποιοι είναι οι τύποι των χρηστών που θα χρησιμοποιήσουν την εφαρμογή; Ποιες είναι οι ανάγκες και οι προσδοκίες τους από την εφαρμογή; Ποιες λειτουργίες είναι καίριες για τους χρήστες και τι τους δίνει κίνητρο να χρησιμοποιήσουν την εφαρμογή;

Τεχνολογίες και Εργαλεία: Οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την εφαρμογή μας, τόσο στο front-end, το back-end αλλά επίσης και για την βάση δεδομένων είναι τα εξής:

- **Front End (Android Studio, Java)**
- **Back End (Django, Python)**
- **Αποθήκευση Δεδομένων (PostgreSQL)**
- **Υποδομή (Amazon EC2, AWS)**
- **Επικοινωνία Front End - Back End (API)**
- **Εφαρμογή Αρχών GRASP**

Τεχνολογίες και τα εργαλεία τα οποία θα τα δούμε πιο αναλυτικά σε επόμενα κεφάλαια.

Περιβάλλον Χρήσης: Πώς οι χρήστες θα αλληλεπιδράσουν με την εφαρμογή; Εάν υπάρχουν ιδιαίτερες συνθήκες ή περιβάλλοντα στα οποία οι χρήστες θα χρησιμοποιήσουν την εφαρμογή, αυτό θα πρέπει επίσης να ληφθεί υπόψη.

Οι λειτουργίες της εφαρμογής προσφέρουν ένα πλήρες σύνολο εργαλείων για τη διαχείριση και παρακολούθηση των καθημερινών συνηθειών των χρηστών. Ανάλογα με την ανάγκη και τη λειτουργικότητα της εφαρμογής, [[3]] ορισμένες βασικές λειτουργίες μπορεί να περιλαμβάνουν:

- **Καταγραφή Συνηθειών:** Οι χρήστες μπορούν να προσθέτουν και να καταγράφουν τις συνήθειές τους. Αυτές μπορεί να περιλαμβάνουν την άσκηση, το διάβασμα, τη διατροφή, την ύπνο, ή οποιαδήποτε άλλη συνήθεια θέλουν να παρακολουθήσουν.
- **Παραμετροποίηση Στόχων:** Οι χρήστες μπορούν να ορίσουν συγκεκριμένους στόχους για κάθε συνήθεια, όπως ο αριθμός των φορών που θέλουν να την εκτελέσουν κατά τη διάρκεια μιας εβδομάδας ή ενός μήνα.



- **Παρακολούθηση Προόδου:** Οι χρήστες μπορούν να παρακολουθούν την πρόοδο τους σε σχέση με τους ορισμένους στόχους που έχουν θέσει. Μπορεί να παρέχεται γραφική αναπαράσταση ή στατιστικά στοιχεία που δείχνουν πώς ανταποκρίνονται στις συνήθειές τους.
- **Ειδοποιήσεις και Υπενθυμίσεις:** Η εφαρμογή μπορεί να παρέχει ειδοποιήσεις ή υπενθυμίσεις για τις προσεχείς συνήθειες που πρέπει να εκτελέσουν οι χρήστες.
- **Επεξεργασία Στοιχείων:** Οι χρήστες μπορούν να επεξεργαστούν τις ρυθμίσεις και τους στόχους των συνηθειών τους.

Αυτές είναι μερικές από τις βασικές λειτουργίες που μπορεί να παρέχει μια εφαρμογή Habit Tracker στους χρήστες της. Ανάλογα με την προσαρμογή και την πολυπλοκότητα της εφαρμογής, μπορεί να υπάρχουν και περισσότερες λειτουργίες ή εργαλεία που προσφέρονται.

2.1 Καταγραφή Συνηθειών

Η ανάλυση του συστήματος καταγραφής συνηθειών υλοποιείται μέσω της εφαρμογής Habit Tracker. Η εφαρμογή Habit Tracker αποτελεί ένα χρήσιμο εργαλείο για τη διαχείριση, την παρακολούθηση και τη βελτίωση της καθημερινής ρουτίνας των χρηστών. [4] Πιο κάτω θα παρουσιαστεί μια λεπτομερή ανάλυση των λειτουργιών της εφαρμογής, των δυνατοτήτων που προσφέρει, καθώς και των οφελών που μπορεί να έχει στην καθημερινή ζωή των χρηστών.

2.1.1 Ο πυρήνας της εφαρμογής Habit Tracker

Η καταγραφή των συνηθειών αποτελεί τον πυρήνα της εφαρμογής Habit Tracker. Αυτή η λειτουργία επιτρέπει στους χρήστες να καταγράφουν και να παρακολουθούν τις δραστηριότητές τους, [5] ενώ τους παρέχει τη δυνατότητα να δημιουργούν ένα προσαρμοσμένο χάρτη της καθημερινής τους ζωής.

Μέσω αυτής της εφαρμογής, οι χρήστες μπορούν να παρακολουθούν τις συνήθειές τους και να δημιουργούν ένα είδος προσαρμοσμένου "χάρτη" της καθημερινής τους ζωής. Αυτός ο χάρτης παρέχει μια ολιστική εικόνα των δραστηριοτήτων τους και της προόδου τους προς την επίτευξη των στόχων τους.

Ο πυρήνας αυτός προσφέρει ευελιξία στους χρήστες να προσαρμόσουν τις συνήθειές τους και τους στόχους τους ανάλογα με τις προσωπικές τους ανάγκες και προτιμήσεις. Η καταγραφή των συνηθειών αποτελεί τον κύριο τρόπο με τον οποίο οι χρήστες διαχειρίζονται και παρακολουθούν τις καθημερινές τους δραστηριότητες, παρέχοντας τους ένα εργαλείο για προσωπική ανάπτυξη και βελτίωση.



2.1.2 Λεπτομέρειες Συνηθειών και Ορισμός Στόχων

Η περιγραφή αυτής της λειτουργίας είναι εξαιρετικά σημαντική για τη διαμόρφωση των στόχων και των συνηθειών σε μια εφαρμογή Habit Tracker. Είναι σημαντικό να επιτρέπετε στους χρήστες να προσαρμόζουν την εφαρμογή σύμφωνα με τις δικές τους ανάγκες και προτεραιότητες. [4] Αυτό παρέχει ευελιξία και προσωποποίηση, που είναι κρίσιμη για την επίτευξη των στόχων τους.

Ανάλογα με τη φύση της συνήθειας ή τον στόχο που θέλει να επιτύχει ο χρήστης, οι επιλογές για τον ορισμό των στόχων μπορεί να ποικίλουν. Το να επιτρέπετε στους χρήστες να ορίζουν συγκεκριμένους στόχους για κάθε συνήθεια, είτε με την καθορισμένη συχνότητα ή τη διάρκεια πρακτικής, είναι σημαντικό για την ατομική τους πρόοδο και εμπνέει τη συνέπεια. Η δυνατότητα προσθήκης, διαγραφής και προσαρμογής συνηθειών επιτρέπει τη δημιουργία μιας προσαρμοσμένης εμπειρίας χρήστη. Οι διάφοροι τύποι συνηθειών που αναφέρετε, όπως άσκηση, διάβασμα, ύπνος και διατροφή, προσφέρουν μια ευρεία ποικιλία στόχων και συνηθειών που μπορούν να προσαρμοστούν στις ατομικές ανάγκες και προτιμήσεις των χρηστών.

Συνοψίζοντας, η ευελιξία στην προσαρμογή των στόχων και των συνηθειών είναι ουσιώδης για να ενθαρρύνετε τη συνέπεια και την επίτευξη των στόχων που έχουν θέσει οι χρήστες σας. Επιπλέον, η προσωποποίηση των στόχων τους επιτρέπει να νιώθουν ότι έχουν έλεγχο και συμβάλλει στην αυτο-παρακολούθηση και την επίτευξη προσωπικών επιτυχιών.

2.1.3 Ορισμός Στόχων

Ο ορισμός των στόχων αναφέρεται στη δυνατότητα των χρηστών να ορίζουν συγκεκριμένους στόχους για κάθε συνήθεια που καταγράφουν στην εφαρμογή Habit Tracker. Αυτός ο ορισμός στόχων μπορεί να περιλαμβάνει διάφορα μεγέθη ή μετρήσεις που οι χρήστες θέλουν να επιτύχουν σε σχέση με τη συγκεκριμένη συνήθεια τους. [5]

Μερικά παραδείγματα στόχων που οι χρήστες μπορούν να ορίσουν περιλαμβάνουν:

- **Συχνότητα Εκτέλεσης:** Ο χρήστης μπορεί να ορίσει πόσες φορές ή πόσες ημέρες την εβδομάδα θέλει να αφιερώνει σε μια συγκεκριμένη συνήθεια. Παράδειγμα: Ασκήσεις 2 φορές την εβδομάδα.
- **Διάρκεια Χρόνου:** Ο χρήστης μπορεί να ορίσει τη διάρκεια χρόνου που επιθυμεί να αφιερώσει σε μια συγκεκριμένη συνήθεια. Παράδειγμα: Διάβασμα - 2 ώρα την ημέρα.
- **Επίτευξη Στόχων:** Ο χρήστης ορίζει συγκεκριμένους στόχους που θέλει να επιτύχει σε σχέση με τη συνήθειά του. Παράδειγμα: Απώλεια βάρους - Χάσιμο 5 κιλών τον μήνα.
- **Σταδιακή Αύξηση:** Ο χρήστης θέλει να αυξήσει σταδιακά τη συχνότητα ή τη διάρκεια της συνήθειάς του. Παράδειγμα: Μελέτη - Αύξηση του χρόνου αφιέρωσης κατά 25 λεπτά κάθε εβδομάδα.



Οι στόχοι αυτοί προσδιορίζουν μια κατεύθυνση για τους χρήστες, παρέχοντας μια δομή και μια συγκεκριμένη κατεύθυνση για την επίτευξη των στόχων τους σε σχέση με τις συνήθειές τους. Η δυνατότητα ορισμού στόχων είναι ένα σημαντικό εργαλείο για την παρακίνηση και την αυτο-βελτίωση των χρηστών στο πλαίσιο της εφαρμογής Habit Tracker.

2.1.4 Προσαρμογή Συνηθειών

Η δυνατότητα προσαρμογής των συνηθειών αποτελεί ένα σημαντικό χαρακτηριστικό της εφαρμογής Habit Tracker. Αυτή η ευελιξία που προσφέρεται στους χρήστες επιτρέπει την προσαρμογή των ρυθμίσεων και των στόχων των συνηθειών τους ανάλογα με τις προσωπικές τους ανάγκες και την εξέλιξή τους. [4] Αυτό δημιουργεί μια επικοινωνία που είναι πιο προσαρμοσμένη και εξατομικευμένη στον κάθε χρήστη.

Η προσαρμογή των ρυθμίσεων συμπεριλαμβάνει τη δυνατότητα να αλλάξουν οι χρήστες τους στόχους που έχουν ορίσει για μια συγκεκριμένη συνήθεια. Αυτό μπορεί να σημαίνει αλλαγή στη συχνότητα με την οποία επιδιώκουν τη συνήθεια ή στον χρόνο που επιθυμούν να αφιερώσουν σε αυτήν. Για παράδειγμα, ένας χρήστης που αρχικά έθεσε στόχο να ασχολείται με τη γυμναστική πέντε φορές την εβδομάδα μπορεί να αντιληφθεί ότι αυτό είναι δυσκολότερο από ό,τι περίμενε. [5] Έτσι, μέσω της ευελιξίας της εφαρμογής, μπορεί να μειώσει τον αριθμό των ημερών γυμναστικής σε τρεις την εβδομάδα για να είναι πιο εφικτός ο στόχος του.

Επιπλέον, η δυνατότητα αλλαγής των στόχων συνδέεται με την εξέλιξη του κάθε χρήστη. Καθώς οι προσωπικές ανάγκες, οι στόχοι και οι δεξιότητες τους μπορεί να αλλάζουν με τον χρόνο, η εφαρμογή επιτρέπει στους χρήστες να προσαρμόζουν συνεχώς τις συνήθειές τους, δίνοντας έτσι τη δυνατότητα για μια πιο βιώσιμη και προσαρμοσμένη εμπειρία. Με αυτόν τον τρόπο, η εφαρμογή Habit Tracker υποστηρίζει την προσωπική ανάπτυξη και τη συνεχή βελτίωση, δημιουργώντας ένα περιβάλλον που προσαρμόζεται στις μεταβαλλόμενες ανάγκες των χρηστών της.

2.2 Παραμετροποίηση Στόχων

Η παραμετροποίηση των στόχων στην εφαρμογή Habit Tracker αντικατοπτρίζει μια αξιοσημείωτη προσέγγιση για τη δημιουργία ενός εξατομικευμένου περιβάλλοντος που στοχεύει στην επίτευξη των στόχων. Η δυνατότητα προσαρμογής των στόχων σύμφωνα με τις ατομικές προτιμήσεις και τις ανάγκες του κάθε χρήστη αποτελεί την πυλώνα για την επίτευξη μιας βελτιωμένης εμπειρίας χρήστη. [4] Αυτό διευκολύνει τη χρήση της εφαρμογής και κάνει την επίτευξη των στόχων πιο εφικτή και ευχάριστη.

Μέσω αυτής της δυνατότητας παραμετροποίησης, οι χρήστες μπορούν να καθορίζουν στόχους που ανταποκρίνονται ακριβώς στις ατομικές τους ανάγκες και δυνατότητες. Αυτό δεν μόνο καθιστά πιο εφικτή την επίτευξη των στόχων, αλλά και ενθαρρύνει τη συνέπεια και την αυτο-παρακολούθηση, καθώς οι χρήστες αισθάνονται περισσότερο συμμετέχοντες στη διαδικασία. Η ευελιξία αυτή στην προσαρμογή των στόχων τους



ενδυναμώνει τους χρήστες να ακολουθούν τις συνήθειές τους, [5] αφού οι στόχοι αντανακλούν τις προσωπικές τους προτιμήσεις και δυνατότητες. Αυτή η εξατομικευμένη προσέγγιση μπορεί να ενισχύσει την ενθάρρυνση και την επίτευξη των στόχων, παρέχοντας στους χρήστες ένα πιο ελκυστικό πλαίσιο εργασίας για την προσωπική τους βελτίωση.

Συνολικά, η παραμετροποίηση των στόχων διευκολύνει τη χρήση της εφαρμογής και κάνει την επίτευξη των στόχων πιο πιθανή, καθώς οι στόχοι είναι πιο προσαρμοσμένοι στις δυνατότητες και τις ανάγκες του κάθε χρήστη.

2.2.1 Ορισμός Συγκεκριμένων Στόχων

Η οριοθέτηση συγκεκριμένων στόχων στην εφαρμογή Habit Tracker επιτρέπει στους χρήστες να καθορίσουν συγκεκριμένους αριθμητικούς ή ποσοτικούς στόχους για κάθε συνήθεια που καταγράφουν.

Αυτό σημαίνει ότι οι χρήστες μπορούν να ορίσουν συγκεκριμένα μέτρα επίδοσης ή στόχους που επιθυμούν να επιτύχουν για κάθε συνήθεια που προσθέτουν στην εφαρμογή. Για παράδειγμα, μπορεί να ορίσουν τον αριθμό των ημερών της εβδομάδας που θέλουν να αφιερώσουν σε μια συνήθεια, το χρονικό διάστημα που θέλουν να διατηρήσουν αυτήν τη συνήθεια, τον αριθμό των ενεργειών που θέλουν να ολοκληρώσουν σε μια ημέρα για τη συγκεκριμένη συνήθεια κ.ο.κ.

Με αυτόν τον τρόπο, οι χρήστες μπορούν να ορίσουν σαφείς και μετρήσιμους στόχους που εξυπηρετούν τις προσωπικές τους ανάγκες και τους βοηθούν στην παρακολούθηση και επίτευξη της προόδου τους.

2.2.2 Προσαρμογή Διάρκειας ή Συχνότητας

Η δυνατότητα προσαρμογής της διάρκειας ή της συχνότητας αποτελεί ένα σημαντικό χαρακτηριστικό που προσφέρει ευελιξία στους χρήστες της εφαρμογής Habit Tracker. Αυτό επιτρέπει στους χρήστες να προσαρμόσουν τη διάρκεια ή τη συχνότητα των συνηθειών τους ανάλογα με τις προσωπικές τους ανάγκες, [6] εξασφαλίζοντας μια εφαρμογή που προσαρμόζεται στον κάθε χρήστη.

Οι χρήστες μπορούν να προσαρμόσουν τη διάρκεια μιας συνήθειας, όπως τον αριθμό των ημερών της εβδομάδας ή το χρονικό διάστημα που αφιερώνουν σε αυτήν. Ανάλογα με τον τρόπο με τον οποίο επιθυμούν να διαχειριστούν τις συνήθειές τους, μπορούν να προσαρμόσουν τον χρόνο που επιθυμούν να αφιερώσουν σε κάθε μια από αυτές. Επιπλέον, η προσαρμογή της συχνότητας επιτρέπει τους χρήστες να αλλάξουν τη συχνότητα παρακολούθησης μιας συνήθειας. Μπορεί να πρόκειται για τον αριθμό των φορές της εκτέλεσης μιας συνήθειας σε μια μέρα ή στον αριθμό των ημερών της εβδομάδας που ο χρήστης θέλει να ασχοληθεί με αυτήν.

Αυτή η ευελιξία επιτρέπει στους χρήστες να προσαρμόσουν τις ρυθμίσεις των συνηθειών τους σύμφωνα με την εξέλιξή τους και τις μεταβαλλόμενες ανάγκες τους,



δημιουργώντας έτσι μια εξατομικευμένη εμπειρία που τους βοηθά να διατηρούν και να παρακολουθούν τις συνήθειές τους με πιο ευχάριστο και αποτελεσματικό τρόπο.

2.2.3 Ευελιξία στον Ορισμό Στόχων

Η ευελιξία στον ορισμό των στόχων σε μια εφαρμογή όπως το Habit Tracker είναι κρίσιμη για να προσφέρει στους χρήστες τη δυνατότητα να προσαρμόζουν τους στόχους τους σύμφωνα με τις μεταβαλλόμενες ανάγκες και τις προσωπικές προτιμήσεις τους. Η ευελιξία αυτή αποτελεί σημαντικό χαρακτηριστικό που επιτρέπει την εξατομίκευση και την προσωποποίηση της εμπειρίας τους.

Σε πολλές περιπτώσεις, οι χρήστες έχουν διάφορες ανάγκες και περιορισμούς που επιθυμούν να αντικατοπτρίζονται στους στόχους τους. [5] Είναι σημαντικό η εφαρμογή να παρέχει τη δυνατότητα προσαρμογής των στόχων σε αυτές τις ανάγκες.

Η ευελιξία στον ορισμό των στόχων συχνά περιλαμβάνει:

- **Ποικιλία Στόχων:** Οι χρήστες μπορούν να ορίσουν διαφορετικούς τύπους στόχων για κάθε συνήθειά τους. Για παράδειγμα, μπορεί να πρόκειται για τον αριθμό των ημερών της εβδομάδας που επιθυμούν να ασχολούνται με αυτήν τη συνήθεια ή το χρονικό διάστημα που αφιερώνουν σε αυτήν.
- **Προσαρμογή των Στόχων:** Οι χρήστες έχουν την δυνατότητα να αλλάζουν τους στόχους τους ανάλογα με την πρόοδό τους, τις αλλαγές στο πρόγραμμα τους ή τις προσωπικές τους προτεραιότητες. Αυτό μπορεί να περιλαμβάνει την προσαρμογή των ημερών ή του χρόνου που αφιερώνουν σε μια συγκεκριμένη συνήθεια.
- **Ειδοποιήσεις και Υπενθυμίσεις:** Η εφαρμογή μπορεί να παρέχει υπενθυμίσεις ή ειδοποιήσεις για την επίτευξη των στόχων, βοηθώντας τους χρήστες να διατηρούν συνέπεια και επίτευξη των στόχων τους.

Η δυνατότητα προσαρμογής των στόχων συμβάλλει στην προσωπική προσαρμογή της εφαρμογής στις ανάγκες του κάθε χρήστη, προσφέροντας μια πιο εξατομικευμένη εμπειρία που ενθαρρύνει την επίτευξη των στόχων τους.

2.3 Παρακολούθηση Προόδου

Η παρακολούθηση της προόδου είναι ένα σημαντικό χαρακτηριστικό σε μια εφαρμογή Habit Tracker καθώς επιτρέπει στους χρήστες να παρακολουθούν την εξέλιξη τους σε σχέση με τους ορισμένους στόχους και τις συνήθειες που έχουν θέσει. Αυτό ενισχύει την επίγνωση και τη συνειδητοποίηση των αλλαγών που κάνουν οι χρήστες στην καθημερινή τους ζωή και τις συνήθειες που προσπαθούν να διαμορφώσουν.

Η λειτουργία παρακολούθησης της προόδου σε σχέση με τους ορισμένους στόχους αποτελεί ένα σημαντικό εργαλείο που προσφέρει η εφαρμογή Habit Tracker στους χρήστες της. Αυτή η λειτουργία επιτρέπει στους χρήστες να παρακολουθούν την πρόοδό



τους σε σχέση με τους ορισμένους στόχους που έχουν ορίσει για κάθε συνήθεια. [[4]] Αυτό συνήθως υλοποιείται μέσω γραφικών παραστάσεων, γραφημάτων ή αναφορών που δείχνουν την πρόοδο τους σε σχέση με τους προκαθορισμένους στόχους τους.

Οι χρήστες μπορούν να δουν αναλυτικά πώς εκτελούν τις συνήθειές τους σε σχέση με τους στόχους που έχουν θέσει. [5] Για παράδειγμα, αν ένας χρήστης έχει ορίσει ως στόχο το να γυμνάζεται πέντε φορές την εβδομάδα, μέσω αυτής της λειτουργίας μπορεί να παρακολουθεί πόσες φορές πραγματοποίησε πραγματικά γυμναστική και πώς αυτό σχετίζεται με τον ορισμένο στόχο του.

Η παρακολούθηση της προόδου μπορεί να περιλαμβάνει:

- **Στατιστικά Στοιχεία:** Η εφαρμογή μπορεί να προσφέρει λεπτομερή στατιστικά στοιχεία για την πρόοδο που έχει επιτευχθεί ανά επιλεγμένη περίοδο ή συγκεκριμένη χρονική στιγμή. Αυτά τα στοιχεία μπορούν να περιλαμβάνουν γραφήματα, διαγράμματα ή πίνακες που απεικονίζουν την πρόοδο προς την επίτευξη των στόχων του χρήστη.
- **Ειδοποιήσεις και Υπενθυμίσεις:** Η εφαρμογή προσφέρει ευέλικτη δυνατότητα για ειδοποιήσεις και υπενθυμίσεις για την εκτέλεση των συνηθειών ή την επίτευξη των στόχων. Αυτό ενθαρρύνει τους χρήστες και τους βοηθά να παρακολουθούν καλύτερα την πρόοδό τους.
- **Συνεχής Βελτίωση:** Η εφαρμογή συνεχώς βελτιώνεται μέσω ανανεώσεων και αναβαθμίσεων με στόχο την καλύτερη εξυπηρέτηση των αναγκών και την βελτίωση της εμπειρίας του χρήστη.

Η δυνατότητα αυτής της παρακολούθησης και ανάλυσης της προόδου βοηθά στην ανάπτυξη ευαισθητοποίησης των χρηστών σχετικά με την πρόοδό τους σε σχέση με τους τεθείς στόχους. Με αυτόν τον τρόπο, ενθαρρύνει τους χρήστες να διατηρούν και να βελτιώνουν τις συνήθειές τους, καθώς είναι σε θέση να αναγνωρίσουν ποιες πράξεις αντιστοιχούν στους ορισμένους στόχους τους και πού απαιτείται περαιτέρω προσπάθεια.

2.3.1 Στατιστικά Στοιχεία

Τα στατιστικά στοιχεία αναφέρονται στη συλλογή, ανάλυση και παρουσίαση δεδομένων για την απεικόνιση της προόδου, των τάσεων και των παραμέτρων που σχετίζονται με τις συνήθειες ή τους στόχους των χρηστών [6].

Στο πλαίσιο μιας εφαρμογής Habit Tracker, η παρουσίαση στατιστικών μπορεί να συμπεριλαμβάνει:

- **Συνολική Πρόοδος:** Παρουσίαση του συνολικού ποσοστού ή της προόδου που έχει επιτευχθεί σε σχέση με τους ορισμένους στόχους.
- **Στατιστικά Στοιχεία Συνηθειών:** Ανάλυση των συχνοτήτων ή των χρόνων που αφιερώνονται σε κάθε συνήθεια. Αυτό μπορεί να περιλαμβάνει γραφικές



αναπαραστάσεις για την προβολή της συχνότητας ή των προσπαθειών που καταβάλλονται.

- **Ημερήσια/Εβδομαδιαία/Μηνιαία Πρόοδος:** Παρουσίαση των αλλαγών στην πρόοδο κατά τη διάρκεια διαφόρων χρονικών περιόδων. Αυτό μπορεί να εμφανίζει την εξέλιξη κατά διαστήματα, όπως ημέρες, εβδομάδες ή μήνες.
- **Σύγκριση Στόχων και Πραγματικής Εκτέλεσης:** Παρουσίαση των στόχων που έχουν τεθεί σε σχέση με την πραγματική εκτέλεση τους, επιτρέποντας στους χρήστες να δουν πού βρίσκονται σε σχέση με τις προσδοκίες τους.
- **Σημεία Κορυφής και Υποβάθμισης:** Προβολή των στιγμών όπου η πρόοδος ήταν εξαιρετική ή όπου υπήρξε πτώση στη συνέπεια. Αυτό μπορεί να βοηθήσει τους χρήστες να αναγνωρίσουν τις περιόδους με υψηλή ή χαμηλή απόδοση.

Αυτά τα στατιστικά δεδομένα μπορούν να παρουσιαστούν με γραφήματα, διαγράμματα ή πινακίδες για να δώσουν στους χρήστες μια ολοκληρωμένη εικόνα της προόδου τους στις συνήθειές τους.

2.3.2 Ειδοποιήσεις και Υπενθυμίσεις

Τα συστήματα ειδοποιήσεων και υπενθυμίσεων είναι σημαντικά στην εφαρμογή Habit Tracker καθώς βοηθούν τους χρήστες να παραμένουν πιστοί στις συνήθειές τους και να διατηρούν τη συνέπειά τους. [5] Οι υπενθυμίσεις και οι ειδοποιήσεις μπορούν να λειτουργήσουν ως χρήσιμο εργαλείο για να ενημερώνουν τους χρήστες για τις επικείμενες δραστηριότητές τους.

Ορισμένες λειτουργίες που μπορεί να περιλαμβάνονται είναι:

- **Ειδοποιήσεις στόχων:** Ειδοποιήσεις που υπενθυμίζουν στους χρήστες τους στόχους που έχουν ορίσει για κάθε συνήθεια. Αυτές οι ειδοποιήσεις μπορούν να στέλνονται καθημερινά, εβδομαδιαία ή σε οποιοδήποτε άλλο χρονικό διάστημα έτσι ώστε να βοηθούν στη διατήρηση της συνέπειας.
- **Ειδοποιήσεις προς Εκτέλεση Συνήθειας:** Υπενθυμίσεις που ενημερώνουν τους χρήστες όταν έρχεται η ώρα να εκτελέσουν μια συγκεκριμένη συνήθεια. Αυτές οι υπενθυμίσεις μπορεί να στέλνονται σε καθορισμένες ώρες ή κατά την επιλογή του χρήστη.
- **Ειδοποιήσεις Επιτεύξεων:** Ειδοποιήσεις που αποστέλλονται όταν οι χρήστες φτάνουν σε ένα ορισμένο στόχο ή επιτυγχάνουν μια ορισμένη πρόοδο στην εκτέλεση των συνηθειών τους. Αυτή η ενημέρωση μπορεί να λειτουργήσει ως ενθάρρυνση και ανταμοιβή.
- **Προσαρμοζόμενες Ειδοποιήσεις:** Δυνατότητα για τους χρήστες να προσαρμόζουν τις ρυθμίσεις ειδοποιήσεων βάσει των προσωπικών τους προτιμήσεων και προτεραιοτήτων.

Οι ειδοποιήσεις αποτελούν σημαντικό μέρος μιας εφαρμογής Habit Tracker καθώς μπορούν να βοηθήσουν στην κατάκτηση και διατήρηση θετικών συνηθειών μέσα στην καθημερινότητα των χρηστών.



2.3.3 Συνεχής Βελτίωση

Η έννοια της συνεχούς βελτίωσης αναφέρεται στην ανάγκη και τον στόχο να βελτιώνουμε συνεχώς την εμπειρία των χρηστών και τη λειτουργικότητα της εφαρμογής Habit Tracker. Η συνεχής βελτίωση προσφέρει τη δυνατότητα ανταπόκρισης στις ανάγκες των χρηστών, προσφέροντας ακόμα καλύτερες λύσεις και εξυπηρετώντας τις προτιμήσεις τους.

Ένα μέρος αυτής της διαδικασίας είναι η συλλογή ανατροφοδότησης από τους χρήστες. Αυτό μπορεί να πραγματοποιείται μέσω αξιολογήσεων, [[4]] ερευνών ικανοποίησης των χρηστών, ή ακόμη και μέσω αναλύσεων της χρήσης της εφαρμογής. Αναλύοντας αυτή την πληροφορία, η ομάδα ανάπτυξης μπορεί να προσδιορίσει τις περιοχές που χρειάζονται βελτίωση και να καταλήξει σε νέες λειτουργίες ή βελτιώσεις που θα ενισχύσουν την εμπειρία των χρηστών.

Επιπλέον, η συνεχής ανάπτυξη και ενημέρωση της εφαρμογής μπορεί να περιλαμβάνει την εφαρμογή νέων τεχνολογιών ή εργαλείων που μπορούν να βελτιώσουν την απόδοση ή την ασφάλεια της εφαρμογής. Επιπλέον, η ανάπτυξη νέων χαρακτηριστικών ή η βελτίωση των υπαρχόντων μπορεί να συμβάλει στη δημιουργία μιας πιο ολοκληρωμένης εφαρμογής που θα εξυπηρετεί ακόμη καλύτερα τις ανάγκες των χρηστών.

Συνολικά, η συνεχής βελτίωση αποτελεί ένα σημαντικό μέρος της διαδικασίας ανάπτυξης λογισμικού καθώς εξασφαλίζει ότι η εφαρμογή παραμένει σύγχρονη, λειτουργική και προσαρμοσμένη στις ανάγκες των χρηστών.

2.4 Ειδοποιήσεις και Υπενθυμίσεις

Οι ειδοποιήσεις και οι υπενθυμίσεις αποτελούν σημαντικό μέρος της λειτουργικότητας της εφαρμογής Habit Tracker, βοηθώντας τους χρήστες να διατηρούν τις συνήθειές τους και να εκπληρώνουν τους ορισμένους στόχους τους. Αποτελούν χρήσιμα εργαλεία για να διατηρούνται ενημερωμένοι και διατηρούν τη συνειδητοποίηση για την εκτέλεση των καθημερινών τους συνηθειών. [4] Οι υπενθυμίσεις μπορούν να είναι χρονικές ή βασισμένες σε ενέργειες. Για παράδειγμα, ένας χρήστης θα μπορούσε να λάβει υπενθύμιση να εκτελέσει την ημερήσια άσκηση σε συγκεκριμένη ώρα ή να διατηρήσει τη συνήθεια του διαβάσματος κάθε βράδυ σε συγκεκριμένη ημέρα.

Οι ειδοποιήσεις μπορούν να χρησιμοποιηθούν για να ενημερώνουν τους χρήστες για την πρόοδό τους ή για πιθανές αλλαγές στους στόχους τους. [6] Αυτό βοηθάει στη συνειδητοποίηση και στην τήρηση της καθημερινής ρουτίνας.

- **Εξατομικευμένες Ειδοποιήσεις:** Η εφαρμογή προσφέρει τη δυνατότητα δημιουργίας εξατομικευμένων ειδοποιήσεων για κάθε συνήθεια που καταγράφει ο χρήστης. Αυτό επιτρέπει την αυτόματη υπενθύμιση και την προβολή των εργασιών που πρέπει να ολοκληρώσει.



- **Προσαρμοζόμενες Ρυθμίσεις Ειδοποιήσεων:** Οι χρήστες έχουν τη δυνατότητα να προσαρμόσουν τη συχνότητα, την ώρα και τη μορφή των ειδοποιήσεων σύμφωνα με τις προτιμήσεις τους. Αυτό επιτρέπει ευελιξία στον τρόπο που οι χρήστες διαχειρίζονται τις υπενθυμίσεις τους.
- **Υπενθυμίσεις Στόχων:** Οι χρήστες λαμβάνουν υπενθυμίσεις όταν πλησιάζουν ή όταν επιτυγχάνουν τους ορισμένους στόχους που έχουν ορίσει για κάθε συνήθεια. Αυτό ενθαρρύνει και υποστηρίζει τη συνέχιση των προσπαθειών τους.
- **Ευέλικτη Διαχείριση Υπενθυμίσεων:** Οι χρήστες μπορούν να επεξεργαστούν ή να απενεργοποιήσουν υπενθυμίσεις ανάλογα με την κατάσταση ή την ανάγκη τους. Αυτή η δυνατότητα επιτρέπει την εξατομίκευση της εμπειρίας τους με την εφαρμογή.
- **Ενίσχυση Αυτο-Πειθαρχίας:** Οι ειδοποιήσεις και οι υπενθυμίσεις συνεισφέρουν στην ανάπτυξη αυτο-πειθαρχίας, βοηθώντας τους χρήστες να διατηρούν μια συνεπή και οργανωμένη ρουτίνα.

Τόσο οι υπενθυμίσεις όσο και οι ειδοποιήσεις μπορούν να προσαρμοστούν ανάλογα με τις ανάγκες του χρήστη, προσφέροντας ένα προσαρμοσμένο και εξατομικευμένο περιβάλλον για τη διατήρηση των συνηθειών.

2.4.1 Εξατομικευμένες Ειδοποιήσεις

Οι εξατομικευμένες ειδοποιήσεις αποτελούν ένα σημαντικό εργαλείο στην εφαρμογή Habit Tracker. Αυτές οι ειδοποιήσεις είναι σχεδιασμένες να προσαρμόζονται σύμφωνα με τις ανάγκες και τις προτιμήσεις του κάθε χρήστη, προσφέροντας ένα εξατομικευμένο περιβάλλον για τη διαχείριση των συνηθειών τους.

Οι εξατομικευμένες ειδοποιήσεις μπορούν να περιλαμβάνουν:

- **Προσαρμοσμένο Χρονοδιάγραμμα:** Οι χρήστες μπορούν να ορίζουν το πότε θα λαμβάνουν ειδοποιήσεις, ανάλογα με τις προσωπικές τους προτιμήσεις. Αυτό μπορεί να είναι το πρωί για την άσκηση, το απόγευμα για το διάβασμα κ.λπ.
- **Προσαρμογή του Τύπου Ειδοποίησης:** Οι χρήστες μπορούν να επιλέξουν τον τύπο της ειδοποίησης που θέλουν να λαμβάνουν, είτε είναι κείμενο, ήχος, ή άλλος τύπος ειδοποίησης.
- **Εξατομικευμένα Μηνύματα:** Οι ειδοποιήσεις μπορούν να περιλαμβάνουν προσαρμοσμένα μηνύματα που ενθαρρύνουν τον χρήστη ή του υπενθυμίζουν τους στόχους του.
- **Επαναλαμβανόμενες Ειδοποιήσεις:** Οι ειδοποιήσεις μπορούν να ρυθμιστούν ώστε να επαναλαμβάνονται καθημερινά, εβδομαδιαία ή με άλλες προσαρμοσμένες συχνότητες.



Με τη χρήση αυτών των παραμέτρων, οι χρήστες έχουν τη δυνατότητα να προσαρμόσουν τις ειδοποιήσεις ώστε να ταιριάζουν με τις ατομικές τους προτιμήσεις και να βοηθούν στην τήρηση των συνηθειών τους.

2.4.2 Προσαρμοσμένες Ρυθμίσεις Ειδοποιήσεων

Οι προσαρμοζόμενες ρυθμίσεις ειδοποιήσεων αναφέρονται στην δυνατότητα που έχουν οι χρήστες να προσαρμόσουν και να διαμορφώσουν τις ειδοποιήσεις που λαμβάνουν από την εφαρμογή Habit Tracker. [5] Αυτές οι ρυθμίσεις επιτρέπουν στους χρήστες να προσαρμόσουν τον τρόπο, το χρονικό διάστημα και την ποσότητα των ειδοποιήσεων σύμφωνα με τις προσωπικές τους προτιμήσεις και ανάγκες.

Ορισμένες από τις προσαρμοζόμενες ρυθμίσεις ειδοποιήσεων μπορεί να περιλαμβάνουν:

- **Χρονικό Πλαίσιο:** Οι χρήστες μπορούν να ορίσουν το χρονικό πλαίσιο κατά το οποίο θέλουν να λαμβάνουν ειδοποιήσεις, όπως το πρωί, το μεσημέρι ή το βράδυ.
- **Συχνότητα Ειδοποιήσεων:** Οι χρήστες μπορούν να επιλέξουν πόσο συχνά θέλουν να λαμβάνουν ειδοποιήσεις για τις συνήθειές τους, είτε καθημερινά, ή με άλλα προσαρμοσμένα χρονικά διαστήματα.
- **Τύπος Ειδοποίησης:** Οι χρήστες μπορεί να επιλέξουν τον τύπο της ειδοποίησης που προτιμούν να λαμβάνουν, όπως ηχητικές ειδοποιήσεις, κείμενο, ή μηνύματα ενθάρρυνσης.

Η ευελιξία στις ρυθμίσεις των ειδοποιήσεων επιτρέπει στους χρήστες να προσαρμόσουν τον τρόπο με τον οποίο λαμβάνουν πληροφορίες και υπενθυμίσεις από την εφαρμογή, βοηθώντας έτσι στη συνειδητοποίηση και τη συνέπεια στην τήρηση των συνηθειών τους.

2.4.3 Υπενθυμίσεις Στόχων

Οι υπενθυμίσεις στόχων αποτελούν ένα σημαντικό εργαλείο που παρέχει η εφαρμογή Habit Tracker στους χρήστες για την υποστήριξη και την επίτευξη των ορισμένων στόχων που έχουν θέσει [6].

Οι υπενθυμίσεις στόχων μπορεί να περιλαμβάνουν:

- **Ημερήσιες Υπενθυμίσεις:** Οι χρήστες μπορούν να λαμβάνουν υπενθυμίσεις σχετικά με τους στόχους τους καθημερινά, προκειμένου να ενθαρρυνθούν να παραμένουν πιστοί στην εκτέλεση των συνηθειών τους.
- **Εξατομικευμένες Υπενθυμίσεις:** Οι χρήστες μπορούν να ορίσουν εξατομικευμένες ρυθμίσεις για τις υπενθυμίσεις, όπως τον τύπο της



υπενθύμισης, το χρονικό σημείο που θα λαμβάνουν την υπενθύμιση και το περιεχόμενο του μηνύματος.

- **Υπενθυμίσεις Στόχων Προόδου:** Σε περιπτώσεις που οι χρήστες έχουν θέσει συγκεκριμένους στόχους για μια συνήθεια, η εφαρμογή μπορεί να παρέχει υπενθυμίσεις σχετικά με την πρόοδο τους προς την επίτευξη αυτών των στόχων.

Ο σκοπός αυτών των υπενθυμίσεων είναι να υποστηρίξουν τους χρήστες, να τους κρατούν ενήμερους και ενθαρρύνουν τη συνέπεια στην εκτέλεση των συνηθειών και στην προσπάθειά τους για την επίτευξη των στόχων που έχουν θέσει.

2.4.4 Ευέλικτη Διαχείριση Υπενθυμίσεων

Η ευέλικτη διαχείριση των υπενθυμίσεων είναι ένα σημαντικό χαρακτηριστικό που επιτρέπει στους χρήστες να προσαρμόζουν τις ρυθμίσεις των υπενθυμίσεων σύμφωνα με τις προσωπικές τους ανάγκες και προτιμήσεις.

Μερικά στοιχεία της ευέλικτης διαχείρισης των υπενθυμίσεων περιλαμβάνουν:

- **Προσαρμογή Χρονισμού:** Οι χρήστες μπορούν να ορίζουν το χρονικό σημείο που επιθυμούν να λαμβάνουν τις υπενθυμίσεις για τις συνήθειές τους.
- **Επιλογή Συχνότητας:** Οι χρήστες έχουν τη δυνατότητα να επιλέξουν τη συχνότητα με την οποία θα λαμβάνουν τις υπενθυμίσεις, είτε αυτές είναι ημερήσιες, εβδομαδιαίες ή και με βάση προσαρμοσμένα διαστήματα.
- **Εξατομικευμένο Περιεχόμενο:** Οι χρήστες μπορούν να προσαρμόσουν το περιεχόμενο των υπενθυμίσεών τους, προσθέτοντας λεπτομέρειες ή ειδικές οδηγίες που θα τους βοηθήσουν να πραγματοποιήσουν τη συγκεκριμένη συνήθεια.
- **Επιλογή Αποστολής:** Οι χρήστες μπορούν να επιλέξουν τον τρόπο που θα λαμβάνουν τις υπενθυμίσεις, είτε μέσω ειδοποιήσεων εφαρμογής, email ή SMS.

Η δυνατότητα προσαρμογής των υπενθυμίσεων συμβάλλει στην εξατομίκευση της εμπειρίας του χρήστη και την αποτελεσματική διαχείριση των συνηθειών τους, προσφέροντας ένα εξατομικευμένο και ευέλικτο σύστημα υπενθυμίσεων.

2.4.5 Ενίσχυση Αυτό-Πειθαρχίας

Η ενίσχυση της αυτο-πειθαρχίας αναφέρεται στη δυνατότητα των χρηστών να ενισχύουν την ικανότητά τους να τηρούν τις δεσμεύσεις τους ή τις συνήθειές τους μέσω εσωτερικής διακυβέρνησης και αυτο-ελέγχου. [7] Ένα σύστημα που προάγει την αυτο-πειθαρχία σε ένα εργαλείο ή εφαρμογή Habit Tracker μπορεί να συμβάλει με τους παρακάτω τρόπους:



- **Ενίσχυση Συνειδητοποίησης:** Παρέχοντας στους χρήστες τη δυνατότητα να βλέπουν τα δεδομένα και την πρόοδό τους, η εφαρμογή τους υπενθυμίζει τους στόχους τους και τους ενθαρρύνει να παραμένουν προσηλωμένοι σε αυτούς.
- **Διαχείριση Στόχων:** Επιτρέποντας την προσαρμογή των στόχων, οι χρήστες μπορούν να αντιληφθούν την εξέλιξή τους και να προσαρμόζουν τις προσπάθειές τους ανάλογα.
- **Ευέλικτη Εφαρμογή:** Παρέχοντας διάφορες επιλογές προσαρμογής και ειδοποιήσεων, η εφαρμογή ενισχύει την αυτο-πειθαρχία και τη διατήρηση των συνηθειών με βάση τις προτιμήσεις και τις ανάγκες του χρήστη.
- **Αυτο-Εκτίμηση:** Η δυνατότητα να παρακολουθούν την πρόοδό τους βοηθά τους χρήστες να αναπτύξουν μια θετική εικόνα και να αναγνωρίζουν την προσπάθεια που καταβάλλουν.

Η συνδυασμένη χρήση των παραπάνω χαρακτηριστικών μέσα από μια εφαρμογή Habit Tracker μπορεί να ενισχύσει την αυτο-πειθαρχία των ατόμων και να τους βοηθήσει να διατηρήσουν και να εκπληρώσουν τους προσωπικούς τους στόχους.

2.5 Επεξεργασία Στοιχείων

Η λειτουργία "Επεξεργασία Στοιχείων" στην εφαρμογή Habit Tracker είναι κρίσιμη για την εξατομίκευση των στόχων και των συνηθειών κάθε χρήστη. Επιτρέπει την προσαρμογή και την τροποποίηση σημαντικών παραμέτρων που σχετίζονται με την επίτευξη των στόχων και την εκτέλεση των συνηθειών.

Με αυτήν τη λειτουργία, [[4]] οι χρήστες μπορούν να προσαρμόσουν τις λεπτομέρειες κάθε συνήθειας, περιλαμβανομένων των στόχων που έχουν οριστεί γι' αυτήν. Αυτό περιλαμβάνει τη δυνατότητα αλλαγής της συχνότητας που η συνήθεια θα πρέπει να εκτελείται ή της χρονικής διάρκειας που αφιερώνεται γι' αυτήν. [[8],[9]] Επιπλέον, οι χρήστες μπορούν να προσθέσουν, να τροποποιήσουν ή να διαγράψουν παραμέτρους που σχετίζονται με τους καθορισμένους στόχους, όπως ο αριθμός των ημερών ή ο χρόνος που αφιερώνεται σε κάθε συνήθεια.

Αυτή η λειτουργία προσφέρει μεγάλη ευελιξία και επιτρέπει στους χρήστες να προσαρμόσουν τους στόχους και τις συνήθειές τους σε συνάρτηση με την προσωπική τους πρόοδο και τις μεταβαλλόμενες ανάγκες.

2.5.1 Επεξεργασία Συνηθειών

Η λειτουργία "Επεξεργασία Συνηθειών" [4] παρέχει στους χρήστες τη δυνατότητα να τροποποιούν τις υπάρχουσες συνήθειες που έχουν καταχωρήσει στην εφαρμογή Habit Tracker. Αυτή η λειτουργία είναι βασική για την προσαρμογή των συνηθειών σύμφωνα με τις ανάγκες τους.



Ορισμένες δυνατότητες επεξεργασίας συνηθειών περιλαμβάνουν:

- **Αλλαγή Ονόματος ή Περιγραφής:** Οι χρήστες μπορούν να τροποποιήσουν το όνομα ή την περιγραφή κάθε συνήθειας. Αυτό μπορεί να περιλαμβάνει την αλλαγή του τίτλου ή την προσθήκη συγκεκριμένων λεπτομερειών που βοηθούν στην καταγραφή ή την καλύτερη κατανόηση της συνήθειας.
- **Τροποποίηση Ρυθμίσεων:** Οι χρήστες μπορούν να προσαρμόσουν τις ρυθμίσεις κάθε συνήθειας, όπως τους στόχους, τη συχνότητα εκτέλεσης ή τη διάρκεια που επιθυμούν να αφιερώσουν σε αυτήν.
- **Προσθήκη/Αφαίρεση Στοιχείων:** Οι χρήστες μπορούν να προσθέσουν ή να αφαιρέσουν οποιαδήποτε ειδικά στοιχεία ή ρυθμίσεις έχουν ορίσει για τη συγκεκριμένη συνήθεια.

Αυτή η λειτουργία επιτρέπει την ευελιξία και την προσαρμογή των συνηθειών, επιτρέποντας στους χρήστες να προσαρμόζουν τις ρυθμίσεις και τους στόχους τους ανάλογα με την προσωπική τους πρόοδο και προτιμήσεις.

2.5.2 Τροποποίηση Στόχων

Η λειτουργία "Τροποποίηση Στόχων" επιτρέπει στους χρήστες να προσαρμόζουν τους ήδη καθορισμένους στόχους που έχουν ορίσει για κάθε συνήθεια μέσω της εφαρμογής Habit Tracker. Αυτό επιτρέπει στους χρήστες να προσαρμόζουν τους στόχους τους σύμφωνα με τις ανάγκες, την πρόοδό τους ή τις προσωπικές προτιμήσεις τους [9].

Οι δυνατότητες τροποποίησης των στόχων περιλαμβάνουν:

- **Αλλαγή Αριθμού ή Ποσού:** Οι χρήστες μπορούν να αλλάξουν τους αριθμητικούς ή ποσοτικούς στόχους που έχουν ορίσει για κάθε συνήθεια. Αυτό μπορεί να περιλαμβάνει την αλλαγή του αριθμού των ημερών της εβδομάδας που επιθυμούν να αφιερώσουν σε αυτήν τη συνήθεια ή το χρονικό διάστημα που επιθυμούν να ασχοληθούν μαζί της.
- **Αλλαγή Συχνότητας ή Διάρκειας:** Οι χρήστες μπορούν να προσαρμόσουν τη συχνότητα ή τη διάρκεια που έχουν αναθέσει στην εκτέλεση μιας συνήθειας. Αυτό σημαίνει ότι μπορούν να αλλάξουν πόσο συχνά ή για πόση ώρα επιθυμούν να ακολουθούν αυτήν τη συνήθεια.

Αυτή η λειτουργία επιτρέπει στους χρήστες να προσαρμόσουν τους στόχους τους, βοηθώντας στην πιο αποτελεσματική και προσαρμοσμένη προσέγγιση για την εκτέλεση των συνηθειών τους.

2.5.3 Αλλαγές στη Ρουτίνα

Η λειτουργία "Αλλαγές στη Ρουτίνα" επιτρέπει στους χρήστες να προσαρμόζουν την καθημερινή τους ρουτίνα στην εφαρμογή Habit Tracker.



Αυτό περιλαμβάνει:

- **Προσθήκη ή Αφαίρεση Συνηθειών:** Οι χρήστες μπορούν να προσθέσουν νέες συνήθειες στη ρουτίνα τους ή να αφαιρέσουν υπάρχουσες συνήθειες. Αυτό τους επιτρέπει να προσαρμόζουν τις δραστηριότητές τους και να ενσωματώνουν νέες συνήθειες στην καθημερινή τους ρουτίνα.
- **Προσαρμογή του Τρόπου Παρακολούθησης:** Οι χρήστες μπορούν να προσαρμόσουν τον τρόπο με τον οποίο παρακολουθούν τις συνήθειές τους. Αυτό μπορεί να περιλαμβάνει την αλλαγή των ρυθμίσεων για τη συγκεκριμένη συνήθεια ή την προσθήκη νέων παραμέτρων παρακολούθησης.
- **Αλλαγή του Προτύπου της Ρουτίνας:** Οι χρήστες μπορούν να τροποποιήσουν το προτιμώμενο μοτίβο ή τη δομή της ρουτίνας τους. Αυτό μπορεί να σημαίνει την αλλαγή των ωρών, των συνηθειών που ακολουθούνται μαζί ή οποιαδήποτε άλλη αλλαγή που βελτιστοποιεί τον τρόπο με τον οποίο εκτελείται η καθημερινή ρουτίνα τους.

Αυτή η λειτουργία προσφέρει στους χρήστες την ευελιξία να διαμορφώσουν τη ρουτίνα τους σύμφωνα με τις ανάγκες τους, προσφέροντας ένα εξατομικευμένο περιβάλλον που προσαρμόζεται στις προτιμήσεις τους και στην προσωπική τους πρόοδο.

2.5.4 Προσαρμογή Προτεραιοτήτων

Η λειτουργία "Προσαρμογή Προτεραιοτήτων" επιτρέπει στους χρήστες να προσαρμόσουν τη σειρά και τη σημασία των συνηθειών τους στη ρουτίνα τους. Αυτό είναι χρήσιμο για να δίνουν προτεραιότητα σε εκείνες τις συνήθειες που θεωρούν πιο σημαντικές ή επείγουσες κάθε στιγμή.[[9]] Ορισμένοι λόγοι για τους οποίους οι χρήστες ενδέχεται να θέλουν να προσαρμόσουν τις προτεραιότητές τους περιλαμβάνουν:

- **Καθημερινή Καταχώρηση:** Κάποιες συνήθειες ενδέχεται να αλλάζουν στη σημασία τους ανάλογα με την πορεία της ζωής του χρήστη. Η δυνατότητα να προσαρμόζετε τη σειρά τους μπορεί να αντικατοπτρίζει αυτές τις αλλαγές.
- **Επικέντρωση στους Σημαντικούς Στόχους:** Οι χρήστες ενδέχεται να επιθυμούν να δίνουν προτεραιότητα σε συγκεκριμένους στόχους κατά τη διάρκεια ενός χρονικού διαστήματος. Η προσαρμογή των προτεραιοτήτων τους μπορεί να τους βοηθήσει να επικεντρωθούν σε αυτούς τους στόχους.
- **Αντιμετώπιση Επείγουσών Αναγκών:** Ορισμένες συνήθειες μπορεί να είναι επείγουσες και απαιτούν άμεση προσοχή. Οι χρήστες μπορούν να προσαρμόσουν τις προτεραιότητές τους για να δώσουν προτεραιότητα σε αυτές τις συνήθειες.

Αυτή η ευελιξία στον ορισμό των προτεραιοτήτων επιτρέπει στους χρήστες να προσαρμόσουν τη ρουτίνα τους σύμφωνα με τις ανάγκες τους και τις αλλαγές στη ζωή τους.



Front End

Η ενότητα που συγγράφηκε αποκλειστικά από τον Χρήστο Βασιλειάδη.



3 Σχεδιασμός και ανάλυση του Front End

Ένα σύστημα front-end είναι εκείνο το μέρος ενός συστήματος πληροφοριών που έχει άμεση πρόσβαση και αλληλεπιδρά με τους χρήστες για να αποκτήσει ή να χρησιμοποιήσει τις δυνατότητες back-end του συστήματος υποδοχής. [10] Επιτρέπει στους χρήστες να έχουν πρόσβαση και να ζητούν τις λειτουργίες και τις υπηρεσίες του υποκείμενου συστήματος πληροφοριών. Τα συστήματα front-end μπορεί να είναι εφαρμογές λογισμικού ή συνδυασμός υλικού, λογισμικού και πόρων δικτύου.

Τα συστήματα front-end χρησιμοποιούνται κυρίως για την αποστολή ερωτημάτων και αιτημάτων και τη λήψη δεδομένων από συστήματα υποστήριξης ή συστήματα πληροφοριών υποδοχής. Υπηρεσίες ή επιτρέπουν στους χρήστες να αλληλεπιδρούν και να χρησιμοποιούν συστήματα πληροφοριών. Συνήθως, τα συστήματα front-end έχουν πολύ περιορισμένες δυνατότητες επεξεργασίας υπολογιστών ή επιχειρησιακής λογικής και βασίζονται σε δεδομένα και λειτουργίες από κεντρικά συστήματα. Ωστόσο, ορισμένα προηγμένα συστήματα διεπαφής διατηρούν ένα αντίγραφο των δεδομένων, όπως ένα αντίγραφο κάθε συναλλαγής που αποστέλλεται στο σύστημα υποστήριξης.

Ο σχεδιασμός της εφαρμογής Habit Tracker περιλαμβάνει τόσο το front-end όσο και το back-end τμήμα της εφαρμογής. Το front-end αφορά τη διεπαφή του χρήστη, ενώ το back-end αναφέρεται στον server, τη διαχείριση της βάσης δεδομένων και την επικοινωνία μεταξύ front-end και back-end.

Για το front-end αναπτύχθηκε χρησιμοποιώντας το Android Studio ως το κύριο περιβάλλον ανάπτυξης. [18] Αυτό το περιβάλλον παρέχει ένα ενιαίο σύνολο εργαλείων για τη δημιουργία Android εφαρμογών και διεπαφών. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την ανάπτυξη του front-end ήταν η Java, η οποία είναι μια από τις κύριες γλώσσες προγραμματισμού που χρησιμοποιούνται για την ανάπτυξη Android εφαρμογών. Η Java είναι ευρέως αποδεκτή και διαθέτει ένα μεγάλο οικοσύστημα βιβλιοθηκών και πλατφόρμας που βοηθά στη δημιουργία χρήσιμων εφαρμογών με εξαιρετική εμπειρία χρήστη. Το Android Studio προσφέρει πολλές ευκαιρίες για τη δημιουργία εξατομικευμένων και εντυπωσιακών εφαρμογών Android, ενώ η Java υποστηρίζει την εύκολη ανάπτυξη με πολύπλοκα χαρακτηριστικά και δυνατότητες.

Ο σχεδιασμός της βάσης δεδομένων θα πρέπει να επικεντρώνεται στις ανάγκες της εφαρμογής, αποθηκεύοντας πληροφορίες για τους χρήστες, τις συνήθειες και την πρόοδο που καταγράφεται. Η σωστή δομή της βάσης δεδομένων είναι κρίσιμη για τη σταθερή και αποτελεσματική λειτουργία της εφαρμογής Habit Tracker. Ο σχεδιασμός του front-end αναφέρεται στη δημιουργία και την ανάπτυξη του μετώπου μιας εφαρμογής, το οποίο είναι ορατό και διαδραστικό από τους χρήστες. Περιλαμβάνει τον σχεδιασμό και την υλοποίηση των στοιχείων της διεπαφής χρήστη, των γραφικών στοιχείων, της δομής και της λειτουργικότητας που ορίζει το πώς αλληλεπιδρούν οι χρήστες με την εφαρμογή κλπ.

Σχετικά με την αρχιτεκτονική, μια προσέγγιση μπορεί να είναι η διαίρεση σε αρκετά επίπεδα τα οποία θα τα συζητήσουμε παρακάτω. Αυτό διευκολύνει τη διαχείριση και την αποτελεσματικότητα της εφαρμογής.



Έτσι έχουμε τη δημιουργία του front-end και αυτό μπορεί να περιλαμβάνει:

- **Σχεδιασμός UI:** Αυτό περιλαμβάνει τον σχεδιασμό των οθονών, των στοιχείων της διεπαφής χρήστη, των κουμπιών, των φορμών, των μενού και την γενική εμφάνιση της εφαρμογής [11].
- **Εμπειρία Χρήστη (UX):** Η αλληλεπίδραση του χρήστη με τη διεπαφή αποτελεί σημαντικό κομμάτι της εμπειρίας του. Περιλαμβάνει τον τρόπο με τον οποίο τα στοιχεία ανταποκρίνονται στις ενέργειες του χρήστη, όπως η αλλαγή κατάστασης ενός στόχου ή η προσθήκη νέας συνήθειας [11][14].
- **Διαδικασίες Εισόδου :** Η σωστή σχεδίαση της διαδικασίας εισόδου περιλαμβάνει την ευκολία χρήσης, την ευαναγνωσία, την απλότητα και τη σαφήνεια των οδηγιών που προσφέρει η εφαρμογή στον χρήστη. Ο στόχος είναι να διευκολύνει τον χρήστη να πλοηγηθεί με ευκολία και να εκτελέσει τις ενέργειες που επιθυμεί με άνεση και αποτελεσματικότητα [15].
- **Διαδικασίες Εξόδου:** Οι διαδικασίες εξόδου αφορούν τη λειτουργία της εφαρμογής που επιτρέπει στον χρήστη να αποσυνδεθεί ασφαλώς από την εφαρμογή ή να την τερματίσει [16].
- **Αναγνώριση Σφαλμάτων:** Η αναγνώριση σφαλμάτων είναι ένα κρίσιμο στάδιο στην ανάπτυξη εφαρμογών. Αναφέρεται στη διαδικασία εντοπισμού, αναφοράς και αντιμετώπισης σφαλμάτων που μπορεί να προκύψουν κατά τη χρήση της εφαρμογής [14][17].

Ο front-end σχεδιασμός είναι καίριος για την επιτυχία της εφαρμογής, καθώς αποτελεί το πρόσωπο και την εμπειρία του χρήστη. Είναι αυτός που συμβάλει στην διεπαφή του χρήστη και ορίζει το πώς αλληλεπιδρά με την εφαρμογή. [11] Ένας προσεκτικά σχεδιασμένος front-end προσφέρει ευχάριστη και ελκυστική εμπειρία στον χρήστη. Η χρήση σωστών χρωμάτων, διατάξεων και γενικά ενός καθαρού σχεδιασμού οδηγεί σε ευχάριστη εμπειρία χρήσης. [13] Η ευχρηστία αφορά την ευκολία πλοήγησης και την κατανόηση της διεπαφής. Ένας καλός σχεδιασμός front-end βοηθά στη σωστή οργάνωση και παρουσίαση των πληροφοριών που είναι εύκολες στην κατανόηση. Είναι σημαντικός ο front-end σχεδιασμός να είναι βελτιστοποιημένος για την απόδοση και την ταχύτητα. Αυτό περιλαμβάνει την βελτιστοποίηση για γρήγορη φόρτωση και αποτελεσματική αλληλεπίδραση με τον χρήστη. Μια καλή σχεδίαση front-end δημιουργεί μια συνεπή εμπειρία για τον χρήστη σε διάφορες συσκευές και περιβάλλοντα.

Συνολικά, το front-end έχει την δυνατότητα να δημιουργήσει μια θετική εμπειρία για τον χρήστη, η οποία είναι κρίσιμη για την επιτυχία μιας εφαρμογής ή μιας ιστοσελίδας.

3.1 Σχεδιασμός UI (User Interface)

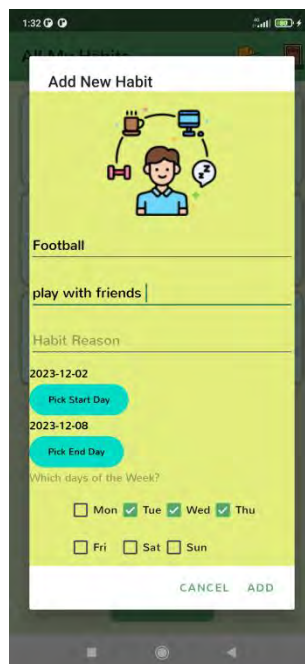
Τα στοιχεία του User Interface (UI) είναι κρίσιμα για την αποτελεσματικότητα και την ευχρηστία μιας εφαρμογής. Τα στοιχεία αυτά αναφέρονται στα στοιχεία που συνθέτουν τη διεπαφή του χρήστη σε μια εφαρμογή ή σε κάποιο λογισμικό.

Περιλαμβάνουν διάφορα στοιχεία που επιτρέπουν στον χρήστη να αλληλεπιδρά με την εφαρμογή, κάνοντας τη χρήση της ευκολότερη και πιο ευχάριστη [11].

Μια εφαρμογή Habit Tracker μπορεί να έχει τα εξής UI στοιχεία:

- **Κουμπιά (Buttons):** Τα κουμπιά μπορεί να σχεδιάζονται με εμφανή και εύκολα γραφικά, όπως πλήρως χρωματισμένα ή με έντονα χρώματα για να ξεχωρίζουν από το υπόλοιπο περιβάλλον. Μπορεί να χρησιμοποιούνται για ενέργειες όπως "Προσθήκη Συνήθειας", "Αποθήκευση" ή "Εναρξη Καταγραφής"(add , save , delete , cancel).
- **Φόρμες (Forms):** Οι φόρμες μπορεί να περιλαμβάνουν πεδία εισαγωγής κειμένου για να προσθέσετε μια νέα συνήθεια ή να επεξεργαστείτε τις υπάρχουσες. Η διάταξη και το στυλ των φορμών πρέπει να είναι χρήσιμα και φιλικά προς τον χρήστη.
- **Λίστες (Lists):** Οι λίστες χρησιμοποιούνται για την εμφάνιση των συνηθειών που έχετε δημιουργήσει. Μπορείτε να τις ταξινομήσετε αλφαβητικά, κατά κατηγορία ή με βάση την πρόοδο. Η εμφάνισή τους θα πρέπει να είναι καθαρή και ευανάγνωστη.
- **Μενού (Menus):** Τα μενού μπορεί να περιλαμβάνουν επιλογές πλοήγησης και ρυθμίσεων, όπως την αλλαγή της γλώσσας, την αλλαγή προτεραιότητας συνήθειας, ή τη διαγραφή συνήθειας.

Η σχεδίαση των στοιχείων της διεπαφής χρήστη (UI) είναι κρίσιμη για την ευκολία χρήσης και την ευχρηστία μιας εφαρμογής όπως φαίνεται στην Εικόνα 1. Είναι σημαντικό να λαμβάνονται υπόψη πολλοί παράγοντες για να δημιουργηθεί μια διεπαφή που θα παρέχει μια ευχάριστη και λειτουργική εμπειρία στον χρήστη.



Εικόνα 1 Διεπαφή χρήστη (User Interface)



Οι παράγοντες αυτοί μπορεί να περιλαμβάνουν τη διεπαφή να είναι ευανάγνωστη, να έχει λογική δομή και οργάνωση, και να παρέχει εύκολη πρόσβαση σε κρίσιμες λειτουργίες. Επίσης, τα χρώματα, οι γραμματοσειρές και ο τρόπος που τα στοιχεία της διεπαφής ανταποκρίνονται στις ενέργειες του χρήστη.

3.1.1 Διάταξη Οθόνης

Οι διάφορες διατάξεις οθόνης στην ανάπτυξη εφαρμογών σχεδιάζονται για να παρέχουν διαφορετικές εμπειρίες χρήστη και διαφορετικούς τρόπους πλοήγησης και αλληλεπίδρασης. Οι διατάξεις οθόνης σε μια εφαρμογή συνήθως σχεδιάζονται για να διευκολύνουν τη χρήση και την επικοινωνία μεταξύ του χρήστη και της εφαρμογής. Κάθε διάταξη οθόνης μπορεί να προσφέρει διαφορετικό τρόπο παρουσίασης πληροφοριών και επιτρέπει διαφορετικές εμπειρίες χρήστη [12].

Ας εξετάσουμε κάθε μοντέλο μεγαλύτερης διεπαφής χρήστη (UI):

1. Tabs (Καρτέλες):

- Περιγραφή: Οι καρτέλες εμφανίζουν διαφορετικές κατηγορίες ή οθόνες και επιτρέπουν στους χρήστες να μετακινούνται μεταξύ τους με ένα άγγιγμα. Κάθε καρτέλα εμφανίζει περιεχόμενο που σχετίζεται με μια συγκεκριμένη κατηγορία.
- Χρήση: Κυρίως χρησιμοποιούνται για να ομαδοποιούν παρόμοια περιεχόμενα ή να παρέχουν πλοήγηση σε διαφορετικές κατηγορίες.

2. Navigation drawers (Συρτάρια πλοήγησης):

- Περιγραφή: Το navigation drawer είναι ένα συρτάρι πλοήγησης που συνήθως εμφανίζεται από την αριστερή πλευρά της οθόνης (σε συσκευές με λειτουργικό Android) και περιλαμβάνει διάφορες επιλογές πλοήγησης.
- Χρήση: Χρησιμοποιείται για την κατανομή πλοήγησης σε μεγάλα σύνολα κατηγοριών ή για να παρέχει πρόσβαση σε πλούσιο περιεχόμενο.

3. Bottom navigation (Πλοήγηση στο κάτω μέρος):

- Περιγραφή: Η πλοήγηση στο κάτω μέρος είναι ένα σύνολο κουμπιών που συνήθως εμφανίζονται στο κάτω μέρος της οθόνης.
- Χρήση: Χρησιμοποιείται για γρήγορη πλοήγηση μεταξύ κύριων τμημάτων της εφαρμογής και για άμεση πρόσβαση σε βασικές λειτουργίες.

Κάθε διάταξη οθόνης παίζει έναν σημαντικό ρόλο στη βέλτιστη αλληλεπίδραση με τον τελικό χρήστη, παρέχοντας ευκολία χρήσης, επιτυχημένη παρουσίαση πληροφοριών και εξασφαλίζοντας ότι ο χρήστης μπορεί να εκτελέσει τις επιθυμητές λειτουργίες με ευκολία και άνεση. Ο σχεδιασμός του UI εξαρτάται από τις ανάγκες της εφαρμογής και την ευκολία χρήσης που επιθυμεί να παρέχει στους χρήστες.

3.1.2 Χρωματική Παλέτα

Η χρωματική παλέτα αποτελεί ένα σημαντικό στοιχείο στον σχεδιασμό της διεπαφής χρήστη (UI/UX). Τα χρώματα επηρεάζουν την αισθητική της εφαρμογής και τον τρόπο με τον οποίο οι χρήστες αλληλεπιδρούν με αυτή. Επιλέγοντας τη σωστή χρωματική παλέτα, μπορεί να δημιουργηθεί μια εμπειρία που είναι ευχάριστη στο μάτι, να ενισχύει την αναγνωσιμότητα και την ιεραρχία των πληροφοριών και να προκαλεί συγκεκριμένα συναισθήματα ή αντιδράσεις στον χρήστη.

Οι χρωματικές παλέτες [13] συχνά περιλαμβάνουν έναν συνδυασμό βασικών χρωμάτων, συμπληρωματικών χρωμάτων και αποχρώσεων που χρησιμοποιούνται για να δώσουν συνοχή στον σχεδιασμό της εφαρμογής. Σε συνδυασμό με τη σωστή χρήση τους, τα χρώματα μπορούν να βελτιώσουν την κατανόηση των πληροφοριών, να καθορίσουν την προτεραιότητα και τη σημασία των στοιχείων, και να ενισχύσουν την ευχρηστία της διεπαφής. Η ανάλυση της χρησιμοποιούμενης χρωματικής παλέτας συνήθως συμπεριλαμβάνει:

1. Επιλογή Χρωμάτων:

- Οι λόγοι πίσω από την επιλογή των συγκεκριμένων χρωμάτων: Ποια συναισθήματα ή συμβολισμοί προσπαθούν να μεταφέρουν τα χρώματα;
- Συχνά τα χρώματα επιλέγονται βάσει της συμβολικής τους σημασίας. Για παράδειγμα, το μπλε μπορεί να αντιπροσωπεύει ηρεμία και εμπιστοσύνη, το κόκκινο μπορεί να συμβολίζει πάθος ή επείγουσα δράση, ενώ το πράσινο συνήθως συνδέεται με φυσική ειρήνη ή ανανέωση.

2. Χρήση Χρωμάτων:

- Τον τρόπο χρήσης των χρωμάτων για να αντιπροσωπεύουν διάφορες καταστάσεις ή ενέργειες: Για παράδειγμα, το κόκκινο μπορεί να χρησιμοποιείται για να επισημάνει σφάλματα ή κάτι σημαντικό, το πράσινο για επιβεβαίωση ή επιτυχία, και το γκρι για ανενεργή ή ανενεργή κατάσταση.

Σημαντικό είναι να επικεντρωθεί η επιλογή των χρωμάτων στη συνέπεια και την αποδοτικότητα για το κοινό του συγκεκριμένου προϊόντος ή υπηρεσίας. Αυτό διασφαλίζει ότι τα χρώματα δεν αντιβαίνουν στην εμπειρία του χρήστη και βοηθούν στην επίτευξη των στόχων τους.

3.2 Εμπειρία Χρήστη UX (User Experience)

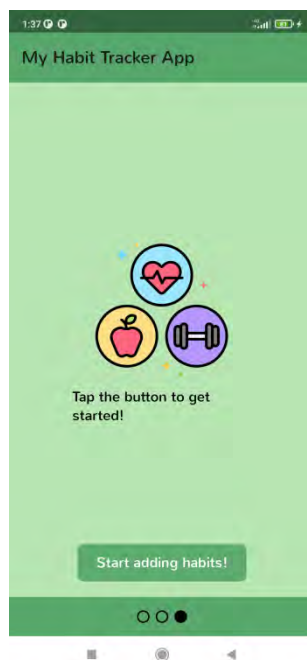
Στοιχεία της εμπειρίας χρήστη (UX) αναφέρονται στον τρόπο με τον οποίο αλληλεπιδρά ο χρήστης με την εφαρμογή και το πώς αυτή αντιλαμβάνεται ο χρήστης της. [[11]] Οι πτυχές του UX συνδέονται στενά με την αποτελεσματικότητα, την ευχρηστία και τη γενική αίσθηση που παρέχει μια εφαρμογή.

Το UX συνδέεται με τον τρόπο με τον οποίο η εφαρμογή ανταποκρίνεται στις ανάγκες του χρήστη. Περιλαμβάνει την απλότητα χρήσης, την ευκολία πλοήγησης στην εφαρμογή όπως φαίνεται στην Εικόνα 2, τη σαφήνεια των οδηγιών και των μηνυμάτων που εμφανίζονται, καθώς και την απόκρυψη πιθανών τεχνικών περιπλοκοτήτων από τον χρήστη.

Το καλό σχεδιασμό UX συχνά συνδέεται με την ανάπτυξη ενός εύχρηστου και ελκυστικού περιβάλλοντος που προσφέρει μια ευχάριστη εμπειρία στον χρήστη. [14] Οι διαδικασίες όπως η μελέτη του χρήστη, οι προτιμήσεις του και ο τρόπος αλληλεπίδρασης του με την εφαρμογή είναι ουσιώδεις για τη βελτίωση του σχεδιασμού UX.

Ορισμένα στοιχεία της εμπειρίας χρήστη περιλαμβάνουν:

- **Διαδραστικότητα:** Ανάλυση του τρόπου με τον οποίο οι χρήστες αλληλεπιδρούν με την εφαρμογή, περιγράφοντας τις κινήσεις, τις αντιδράσεις και τις μεταβάσεις μεταξύ των οθονών.
- **Απλότητα και Κατανόηση:** Πώς ο σχεδιασμός βοηθά στην εύκολη κατανόηση των λειτουργιών και πώς η απλότητα βελτιώνει την εμπειρία του χρήστη.
- **Πλοήγηση:** Περιγραφή των μοντέλων πλοήγησης και του τρόπου που επιτρέπουν στον χρήστη να βρει εύκολα τις λειτουργίες που χρειάζεται.



Εικόνα 2 Σελίδα Εισαγωγής (Intro Page)

Η συνολική συνεισφορά αυτών των πτυχών οδηγεί σε μια ολοκληρωμένη και ικανοποιητική εμπειρία για τον χρήστη.



3.2.1 Διαδραστικότητα

Η διαδραστικότητα μιας εφαρμογής είναι ένα σημαντικό στοιχείο που επηρεάζει την εμπειρία του χρήστη. Αναλυτικότερα, αυτή η πτυχή αναφέρεται στην αλληλεπίδραση του χρήστη με το περιβάλλον της εφαρμογής. Αναλύει τον τρόπο που ο χρήστης εκτελεί κινήσεις, αλληλεπιδρά με τα διάφορα στοιχεία και πώς η εφαρμογή ανταποκρίνεται σε αυτές τις ενέργειες.

Συγκεκριμένα, στον αναλυτικό επίπεδο, αναφέρει την δομή και την συμπεριφορά των στοιχείων UI (διεπαφή χρήστη) και πώς αλληλεπιδρούν με τις δράσεις του χρήστη. Περιγράφει την ροή των διαφόρων καταστάσεων μεταξύ των οθονών, τις μεταβάσεις, τα εφέ και την συνολική ανταπόκριση της εφαρμογής στις ενέργειες του χρήστη. Ένα καλά σχεδιασμένο σύστημα διαδραστικότητας ενθαρρύνει τους χρήστες να αλληλεπιδρούν εύκολα, γρήγορα και αποτελεσματικά με την εφαρμογή, βελτιώνοντας την εμπειρία χρήστη και την αποτελεσματικότητα της εφαρμογής.

3.2.2 Απλότητα και Κατανόηση

Η απλότητα στον σχεδιασμό είναι ένα σημαντικό στοιχείο που συμβάλλει στην εύκολη κατανόηση των λειτουργιών μιας εφαρμογής. Ο σχεδιασμός που βασίζεται στην απλότητα βοηθά τον χρήστη να αντιληφθεί εύκολα τη χρήση και την πλοήγηση στην εφαρμογή.

Ο εντοπισμός των λειτουργιών και η πρόσβαση σε αυτές γίνονται πιο απλοί και γρήγοροι, μειώνοντας τον χρόνο που χρειάζεται ο χρήστης για να εκτελέσει διάφορες δράσεις. Ο απλός σχεδιασμός βοηθά στη δημιουργία μιας ομαλής και φυσικής διαδικασίας για τον χρήστη, προσφέροντας μια απολαυστική και ομαλή εμπειρία.

Ένας καλά σχεδιασμένος χρήσιμος χώρος προσφέρει απλές, κατανοητές, και σαφείς διεπαφές που καθιστούν ευκολότερη την πλοήγηση και τη χρήση των λειτουργιών. Αυτό συνεπάγεται μια πιο ευχάριστη και ενδιαφέρουσα εμπειρία χρήστη, ενθαρρύνοντας τη συνεχή χρήση της εφαρμογής.

3.2.3 Πλοήγηση

Η πλοήγηση είναι αναπόσπαστο τμήμα του σχεδιασμού μιας εφαρμογής, καθώς διευκολύνει τον χρήστη να εντοπίσει και να αλληλεπιδράσει με τις λειτουργίες που χρειάζεται.

Μοντέλα πλοήγησης όπως τα Tabs, Navigation drawers, Bottom navigation και άλλα, διευκολύνουν την ανταπόκριση των χρηστών στο UI (διεπαφή χρήστη), επιτρέποντάς τους να μετακινούνται μεταξύ των διαφόρων οθονών ή λειτουργιών με άνεση. Οι λειτουργίες πρέπει να είναι διαθέσιμες και προσβάσιμες σε συγκεκριμένα μέρη της εφαρμογής ώστε να παρέχεται μια συνεκτική εμπειρία χρήστη.

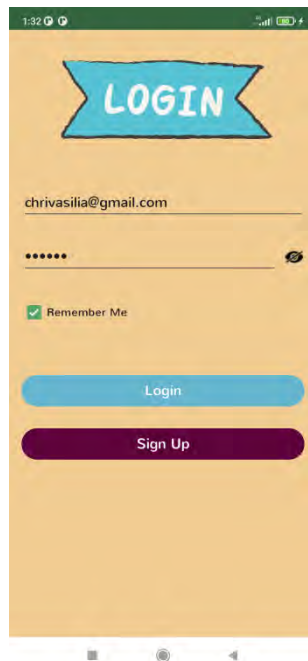
Ένα καλό μοντέλο πλοήγησης πρέπει να είναι εύκολο στη χρήση, να προσφέρει γρήγορη πρόσβαση στις κύριες λειτουργίες και να είναι ενσωματωμένο στη φυσική ροή της εφαρμογής.

3.3 Διαδικασίες Εισόδου

Οι διαδικασίες εισόδου αναφέρονται στα στάδια όπου οι χρήστες συνδέονται ή εισέρχονται στην εφαρμογή. [15] Αυτό μπορεί να είναι μέσω διαφόρων μεθόδων εισόδου, όπως σύνδεση με λογαριασμό (email/username/password), χρήση αναγνώρισης δακτυλικών αποτυπωμάτων, κωδικών PIN, ή ακόμη και μέσω αυτόματης εισόδου (αν ο χρήστης έχει επιλέξει αυτή την επιλογή). Συχνά, η διαδικασία εισόδου αποτελεί κρίσιμο σημείο για την εμπειρία του χρήστη, καθώς η απλότητα και η ασφάλεια είναι κρίσιμες. Ο σχεδιασμός της διαδικασίας εισόδου πρέπει να είναι φιλικός προς τον χρήστη, εξασφαλίζοντας την ασφάλεια των προσωπικών του δεδομένων. Μια επιτυχημένη διαδικασία εισόδου είναι αυτή που επιτρέπει στον χρήστη να αποκτήσει πρόσβαση στην εφαρμογή με ελάχιστο κόπο, αλλά υπάρχει πάντα η επιβεβαίωση της ταυτότητάς του για λόγους ασφαλείας.

1. Σύστημα Σύνδεσης (Login System):

- Περιγραφή της διαδικασίας σύνδεσης χρήστη, συμπεριλαμβανομένων των διαφόρων μεθόδων εισόδου (email /username/password , OAuth, δακτυλικό αποτύπωμα, κ.λπ.).
- Πιθανά στάδια επαλήθευσης, όπως ελέγχους ασφαλείας ή απαιτούμενες πληροφορίες εισόδου.



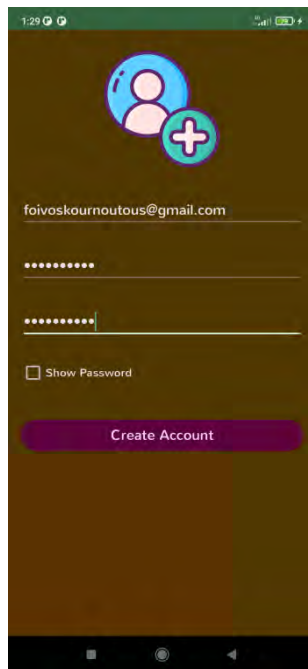
Εικόνα 3 Σύστημα Σύνδεσης (Login System)

Η σελίδα σύνδεσης που βλέπει ο χρήστης όταν επισκέπτεται την εφαρμογή (Login page) είναι η σελίδα σύνδεσης, όπως φαίνεται στην Εικόνα 3. Εδώ ζητείται από τον χρήστη το

email και ο κωδικός του, όπου συμπληρώνοντας τα και πατώντας το κουμπί "Login", αν είναι σωστά συνδέεται στην εφαρμογή. Αν κάποιο από τα δύο είναι λάθος ή μη συμπληρωμένο εμφανίζονται αντίστοιχα μηνύματα για λάθος συνδυασμό ονόματος χρήστη (email) και κωδικού (password) ή μη συμπλήρωσης πεδίου.

2. Διαδικασίες Εγγραφής (Registration Procedures):

- Δημιουργία νέου λογαριασμού εν αυτός δεν υπάρχει.
- Απαραίτητες πληροφορίες που απαιτούνται κατά τη διαδικασία εγγραφής.
- Σε περίπτωση επιτυχούς εγγραφής, ο χρήστης ενημερώνεται με μήνυμα επιτυχίας.



Εικόνα 4 Σύστημα Εγγραφής Χρηστή (Sign Up)

Η σελίδα εγγραφής που βλέπει ο χρήστης όταν επιλέγει το "Sign Up" όπως φαίνεται στην Εικόνα 4, είναι παρόμοια με τη σελίδα σύνδεσης προσφέροντας πεδία για την εισαγωγή των απαραίτητων πληροφοριών. Αφού συμπληρώσει τα πεδία και πατήσει το κουμπί "Create Account" (Δημιουργία Λογαριασμού), εκτελείται η διαδικασία εγγραφής με αντίστοιχα μηνύματα επιτυχίας ή σφάλματος, ανάλογα με την επιτυχία της διαδικασίας.

3.4 Διαδικασίες Εξόδου

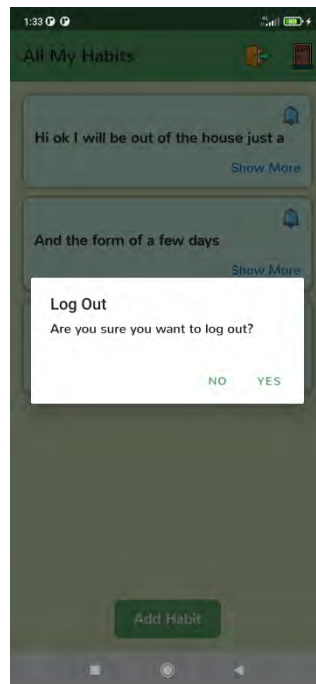
Οι διαδικασίες εξόδου αναφέρονται στις διάφορες διαδικασίες που επιτρέπουν σε έναν χρήστη να αποσυνδεθεί από μια εφαρμογή ή ένα σύστημα. [16] Αυτές

συμπεριλαμβάνουν την αποσύνδεση από τον λογαριασμό ή την αποσύνδεση από το λογισμικό ή το σύστημα.

Μερικές διαδικασίες εξόδου περιλαμβάνουν:

- **Αποσύνδεση (Logout):** Η διαδικασία αποσύνδεσης συνήθως αφορά τον τρόπο ασφαλούς αποσύνδεσης από έναν λογαριασμό.
- **Τερματισμός (Exit):** Η λειτουργία που κλείνει την εφαρμογή ή το λογισμικό. Αυτός ο τερματισμός μπορεί να οδηγήσει στο κλείσιμο όλων των διεργασιών που σχετίζονται με την εφαρμογή.

Η διαδικασία αποσύνδεσης (logout) σε μια εφαρμογή αναφέρεται στον τρόπο αποσύνδεσης ενός χρήστη από τον λογαριασμό του. Η διαδικασία αποσύνδεσης πρέπει να εξασφαλίζει ότι ο χρήστης μπορεί να αποσυνδεθεί από τον λογαριασμό του με ασφάλεια και να μην έχει πρόσβαση σε ευαίσθητα δεδομένα ή λειτουργίες της εφαρμογής. Συνήθως, η διαδικασία αποσύνδεσης περιλαμβάνει ένα κουμπί ή μια επιλογή εντός της εφαρμογής όπου ο χρήστης μπορεί να πατήσει για να αποσυνδεθεί όπως φαίνεται στην Εικόνα 5. Κατά την αποσύνδεση, ο χρήστης θα μεταφερθεί σε μια σελίδα επιβεβαίωσης ή σε μια σελίδα εισόδου (login page) για να ξανασυνδεθεί.



Εικόνα 5 Διαδικασία Αποσύνδεσης (Log Out)

Παράλληλα, σε ορισμένες περιπτώσεις, μπορεί να περιλαμβάνει επιπρόσθετα μέτρα ασφαλείας για την επιβεβαίωση ταυτότητας του χρήστη, όπως η επαλήθευση του κωδικού πρόσβασης ή η χρήση δεύτερου παράγοντα αυθεντικοποίησης (π.χ. κωδικός επιβεβαίωσης που αποστέλλεται στο κινητό του χρήστη). Αυτό είναι ένας πρόσθετος τρόπος επιβεβαίωσης ότι ο χρήστης έχει την εξουσιοδότηση να αποσυνδεθεί από τον λογαριασμό του.



Ο τερματισμός (Exit) αναφέρεται στη διαδικασία κλείσιματος της εφαρμογής ή του λογισμικού. Όταν ο χρήστης επιλέγει να τερματίσει την εφαρμογή, η διαδικασία τερματισμού οδηγεί στο κλείσιμο των διαφόρων διεργασιών που σχετίζονται με τη λειτουργία της εφαρμογής. Ο τερματισμός μπορεί να επιτευχθεί μέσω διαφόρων μεθόδων, όπως το κλείσιμο του παράθυρου της εφαρμογής ή μέσω της χρήσης κουμπιών ή επιλογών μέσα στην εφαρμογή που οδηγούν στον τερματισμό. Κατά τον τερματισμό, η εφαρμογή αποθηκεύει τυχόν μη αποθηκευμένες αλλαγές και δεδομένα, κλείνει τυχόν δρομολογημένες διεργασίες και απελευθερώνει τους πόρους που χρησιμοποιούνται από την εφαρμογή. Ο σκοπός του τερματισμού είναι να ολοκληρώσει με ασφάλεια και να διακόψει τη λειτουργία της εφαρμογής.

Ο σχεδιασμός αυτών των διαδικασιών έχει σημαντική σημασία για τη χρήση και την ασφάλεια του συστήματος, ενώ η εξοικείωση του χρήστη με αυτές τις διαδικασίες συμβάλλει στην καλύτερη εμπειρία χρήστη.

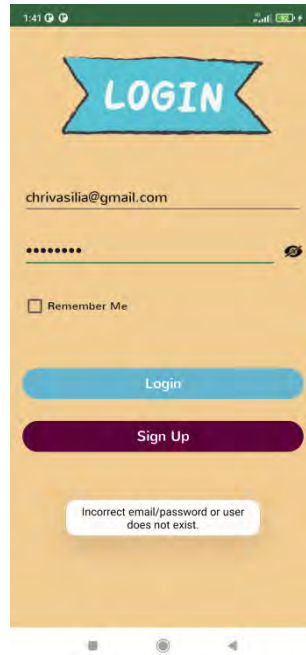
3.5 Αναγνώριση Σφαλμάτων

Η διαδικασία αναγνώρισης σφαλμάτων στην εφαρμογή είναι ζωτικής σημασίας για τη βελτίωση της εμπειρίας του χρήστη. Αποτελεί μια διαδικασία στη σχεδίαση και την ανάπτυξη λογισμικού που στοχεύει στον εντοπισμό, την ανάλυση και τη διόρθωση σφαλμάτων που μπορεί να προκύψουν. [14] Οι διαδικασίες αναγνώρισης σφαλμάτων συχνά περιλαμβάνουν την χρήση συστημάτων ανίχνευσης σφαλμάτων, τα οποία καταγράφουν πληροφορίες σχετικά με τυχόν σφάλματα στο λογισμικό. Συνήθως περιλαμβάνουν τη συλλογή σφαλμάτων από τους χρήστες, την ανάλυση των σφαλμάτων και την εφαρμογή διορθώσεων.

Η καλή αναγνώριση και διόρθωση σφαλμάτων συμβάλλει στη βελτίωση της ποιότητας της εφαρμογής, στην αύξηση της αξιοπιστίας του λογισμικού και στη βελτίωση της εμπειρίας των χρηστών [17].

Ας δούμε τις επιμέρους πτυχές:

- **Αναγνώριση Χρήστη:** Πρόκειται για τον τρόπο με τον οποίο η εφαρμογή αναγνωρίζει και επιβεβαιώνει την ταυτότητα του χρήστη. Αυτό μπορεί να γίνει μέσω συνδρομής σε λογαριασμό με κωδικό πρόσβασης, αναγνώριση μέσω email ή κινητού τηλεφώνου, biometrics (όπως αναγνώριση δακτυλικών αποτυπωμάτων ή αναγνώριση προσώπου), κ.λπ.
- **Ενδείξεις Σφάλματος:** Πρόκειται για τον τρόπο με τον οποίο η εφαρμογή ειδοποιεί τον χρήστη όταν παρουσιάζεται ένα σφάλμα. Αυτό μπορεί να είναι με μηνύματα σφάλματος (όπως "Λάθος κωδικός πρόσβασης", "Σφάλμα σύνδεσης" κλπ) όπως φαίνεται στην Εικόνα 6 αλλά και χρωματικές ενδείξεις όπως κόκκινα ή πορτοκαλί χρώματα για να δείξουν κάτι που δεν λειτουργεί σωστά.



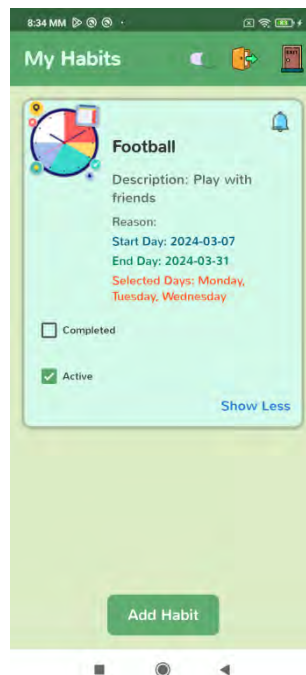
Εικόνα 6 Σφάλμα “Λάθος κωδικός πρόσβασης” κτλ

Ο έμπειρος σχεδιασμός στοιχείων που βελτιώνουν την εμπειρία χρήστη (UX) μπορεί να συμβάλει στην ενίσχυση της κατανόησης του χρήστη και τη βελτίωση της αποτελεσματικότητας στην αντιμετώπιση τυχόν σφαλμάτων ή προβλημάτων. Με τη χρήση καλώς σχεδιασμένων στοιχείων UI/UX, οι χρήστες είναι πιο πιθανό να κατανοήσουν γρήγορα τη λειτουργία της εφαρμογής, να αποφεύγουν λάθη και να τα αντιμετωπίζουν εύκολα όταν προκύπτουν. Ο σωστός σχεδιασμός μπορεί να ενισχύσει την προσιτότητα και την ευκολία χρήσης, ενισχύοντας τη συνολική απόκριση του χρήστη σε περιπτώσεις προβλημάτων.

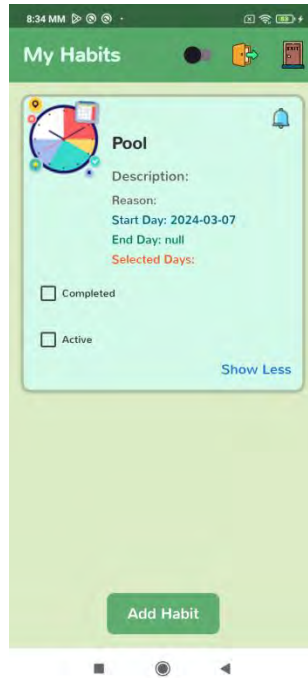
3.6 Αρχαιοθετημένες Συνήθειες

Η λειτουργία των αρχαιοθετημένων συνηθειών επιτρέπει στους χρήστες να διατηρούν μια ιστορική εγγραφή των συνηθειών που έχουν ολοκληρώσει ή ακόμη και των συνηθειών που έχουν εγκαταλείψει. Αυτό είναι χρήσιμο για την ανάλυση της προόδου τους με την πάροδο του χρόνου. Οι χρήστες μπορούν να επισκέπτονται το ιστορικό των συνηθειών τους για να δουν την εξέλιξη τους, να αξιολογήσουν τη συνεπή τους προσπάθεια και να εμπνευστούν να συνεχίσουν την προσπάθειά τους για την αυτοβελτίωση. Η λειτουργία αυτή προσφέρει στους χρήστες τη δυνατότητα να επιλέξουν συνηθειες από μια προηγούμενη λίστα ή να ανακαλύψουν συνηθειες που μπορεί να έχουν αγνοήσει στο παρελθόν. Αυτό μπορεί να τους βοηθήσει να επιλέξουν συνηθειες που είναι πιο πιθανό να ενσωματώσουν στην καθημερινή τους ρουτίνα, καθώς έχουν ήδη αναγνωρίσει τη σημασία ή την αποτελεσματικότητά τους από προηγούμενες προσπάθειες ή παρατηρήσεις. Επιπλέον, μπορεί να προσφέρει ένα ευρύτερο φάσμα επιλογών στους χρήστες, καθώς ορισμένες συνηθειες που προτείνονται μπορεί να μην είναι στην αρχική τους λίστα σκέψης.

Η χρήση ενός switch button στις αρχειοθετημένες συνήθειες μπορεί να παρέχει μια εύκολη και γρήγορη μέθοδο για τον χειρισμό τους. Με τη χρήση ενός switch button, ο χρήστης μπορεί να ενεργοποιεί ή να απενεργοποιεί μια συνήθεια με ένα απλό άγγιγμα. Όταν ο χρήστης ενεργοποιεί το switch button για μια συνήθεια και έχει επιλέξει στην αντίστοιχη συνήθεια το πλαίσιο active, όπως φαίνεται στην Εικόνα 7 αυτή η συνήθεια προστίθεται στις ενεργές συνήθειες και ξεκινά να παρακολουθείται. Αντίστοιχα, όταν ο χρήστης απενεργοποιεί το switch button και το πλαίσιο active, όπως φαίνεται στην Εικόνα 8 η συνήθεια αφαιρείται από τις ενεργές συνήθειες και παύει να παρακολουθείται.



Εικόνα 7 Ενεργοποιημένη συνήθεια για παρακολούθηση



Εικόνα 8 Απενεργοποιημένη συνήθεια μη παρακολούθησης

Με αυτόν τον τρόπο, οι χρήστες μπορούν εύκολα να ελέγχουν τις συνήθειες που θέλουν να αρχειοθετήσουν ή να επανενεργοποιήσουν, προσφέροντας μια απλή και ευέλικτη διαδικασία διαχείρισης των συνηθειών τους.

3.7 Αποστολή Email Καλωσορίσματος στον Νέο Χρήστη

Η λειτουργία αποστολής καλωσορίσματος μέσω email στον νέο χρήστη είναι σημαντική για να καλωσορίσεις το νέο μέλος στην κοινότητα της εφαρμογής σου και να τον ενθαρρύνεις να αρχίσει να χρησιμοποιεί την υπηρεσία σου, όπως φαίνεται στην Εικόνα 9. Αυτό μπορεί να δημιουργήσει μια θετική εντύπωση και να βοηθήσει στην καλή εκκίνηση της χρήσης της εφαρμογής.

- **Καλωσόρισμα στην κοινότητα:** Το email καλωσορίσματος δίνει την αίσθηση της υποδοχής σε μια νέα κοινότητα. Αναγνωρίζει το νέο μέλος ως σημαντικό μέρος της κοινότητας της εφαρμογής.
- **Ενθάρρυνση για χρήση:** Το καλωσόρισμα μπορεί να περιλαμβάνει κίνητρα και ενθαρρυντικά μηνύματα για να κινητοποιήσει τον νέο χρήστη να ξεκινήσει να χρησιμοποιεί την εφαρμογή.
- **Παροχή πληροφοριών:** Συνήθως, τα email καλωσορίσματος περιλαμβάνουν σημαντικές πληροφορίες για τη λειτουργία και τις δυνατότητες της εφαρμογής. Αυτό βοηθά το νέο μέλος να εξοικειωθεί γρήγορα με την εφαρμογή και τις διαθέσιμες λειτουργίες.
- **Πρώτη εντύπωση:** Το email καλωσορίσματος δημιουργεί μια πρώτη εντύπωση για την εφαρμογή και την ομάδα που την υποστηρίζει. Μια θετική πρώτη εντύπωση μπορεί να ενθαρρύνει τον χρήστη να παραμείνει πιστός στην εφαρμογή και να εξερευνήσει τις δυνατότητές της.



Εικόνα 9 Email που έρχεται στον χρήστη μετά την εγγραφή του

Συνολικά, το καλωσόρισμα μέσω email είναι ένα σημαντικό μέσο για να καλωσορίσεις τους νέους χρήστες, να τους ενθαρρύνεις να αρχίσουν να χρησιμοποιούν την εφαρμογή και να τους προσφέρεις την υποστήριξη που χρειάζονται για να εξερευνήσουν τα πλεονεκτήματα και τις δυνατότητες της.



Back End

Η ενότητα που συγγράφηκε αποκλειστικά από τον Φοίβο Κορνούτο.

4 Σχεδιασμός και ανάλυση του Back End

Το back-end αναφέρεται στο μέρος μιας ιστοσελίδας ή μιας εφαρμογής που δεν είναι ορατό στον τελικό χρήστη, αλλά παίζει κρίσιμο ρόλο στη λειτουργία της. [19] Το back-end αναλαμβάνει τη διαχείριση των δεδομένων, την επικοινωνία με τη βάση δεδομένων, την επεξεργασία των αιτημάτων των χρηστών και πολλές άλλες λειτουργίες που είναι απαραίτητες για τη σωστή λειτουργία της εφαρμογής και συγκεκριμένα της εφαρμογής μας Habit Tracker.

Οι τεχνολογίες που χρησιμοποιούνται συχνά για την υλοποίηση του back-end περιλαμβάνουν γλώσσες προγραμματισμού όπως η Python, η Java, η Ruby, η PHP, πλαίσια εργασίας (frameworks) όπως το Django, το Ruby on Rails, το Laravel και συστήματα διαχείρισης βάσεων δεδομένων όπως η PostgreSQL, η MySQL, η MongoDB κ.ά.

Η διασύνδεση (ή αλληλεπίδραση) μεταξύ του back-end και του front-end είναι κρίσιμη για τη σωστή λειτουργία μιας εφαρμογής. Η εφαρμογή μας Habit Tracker χρησιμοποιεί ένα back-end που διαχειρίζεται τα δεδομένα της και ένα front-end που παρουσιάζει αυτά τα δεδομένα στους χρήστες χρησιμοποιώντας [21] ένα API για τη διασύνδεση μεταξύ τους. Το API επιτρέπει στο front-end να επικοινωνεί με το back-end. Οι αιτήσεις για δεδομένα ή οποιαδήποτε ενέργεια που πρέπει να εκτελεστεί στα δεδομένα αποστέλλονται μέσω του API στο back-end.

Το back-end είναι υπεύθυνο για την εξυπηρέτηση των αιτημάτων των χρηστών και τη διαχείριση των δεδομένων που απαιτούνται για την ορθή λειτουργία της εφαρμογής. Είναι σημαντικό να είναι αξιόπιστο, ασφαλές και να παρέχει καλή απόδοση για να εξασφαλίσει μια ομαλή εμπειρία για τους χρήστες. Αντιπροσωπεύει τον υπολογιστικό πυρήνα μιας εφαρμογής που λειτουργεί στον διακομιστή και χειρίζεται τον λογικό προγραμματισμό, τη διαχείριση της βάσης δεδομένων και την αλληλεπίδραση με τον χρήστη (συνήθως μέσω του front-end). [20] Ανακεφαλαιωτικά, παρέχει τις εξής λειτουργίες:

- **Διαχείριση Βάσης Δεδομένων:** Το back-end είναι υπεύθυνο για τη διαχείριση των δεδομένων που αποθηκεύονται στη βάση δεδομένων. Αυτό περιλαμβάνει την ανάγνωση, εγγραφή, ενημέρωση και διαγραφή δεδομένων.
- **Επιχειρησιακή Λογική:** Το back-end περιέχει την επιχειρησιακή λογική που καθορίζει πώς λειτουργεί η εφαρμογή. Αυτό περιλαμβάνει τον έλεγχο της επικοινωνίας με τη βάση δεδομένων, τον υπολογισμό των αποτελεσμάτων και την ανταπόκριση στα αιτήματα του χρήστη.
- **Ασφάλεια:** Η ασφάλεια είναι ένας σημαντικός παράγοντας του back-end. Πρέπει να παρέχει μέτρα ασφαλείας για την προστασία των δεδομένων και την πρόληψη από τυχόν επιθέσεις.
- **Απόδοση:** Το back-end πρέπει να είναι αποδοτικός και να υποστηρίζει τις ανάγκες του χρήστη, παρέχοντας γρήγορη απόκριση στα αιτήματα.



- **Επικοινωνία με το Front End:** Το back-end αλληλεπιδρά με το front-end μέσω διαφόρων τεχνολογιών (όπως HTTP, WebSockets) για να παρέχει δεδομένα στους χρήστες.
- **Επεκτασιμότητα:** Το back-end πρέπει να είναι σχεδιασμένο για να υποστηρίζει την αυξανόμενη κίνηση και την προσθήκη νέων λειτουργιών όπως αναπτύσσεται η εφαρμογή.

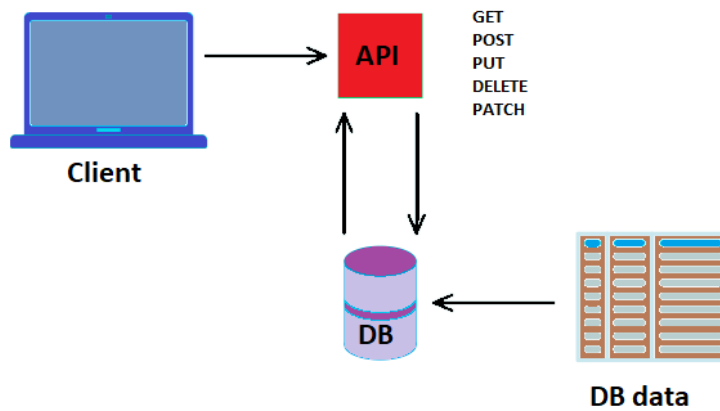
Συνοπτικά, το back-end αποτελεί τον "εγκέφαλο" της εφαρμογής σας, όπου γίνονται όλοι οι υπολογισμοί και διαχειρίζονται τα δεδομένα. Είναι απαραίτητο να είναι σταθερό, ασφαλές και αποδοτικό για να εξασφαλίσει τη σωστή λειτουργία της εφαρμογής σας. Το back-end αποτελεί ένα σημαντικό μέρος μιας εφαρμογής και συγκεκριμένα της εφαρμογής μας Habit Tracker, η χρήση PostgreSQL, Django και το AWS EC2 είναι μια ισχυρή συνδυασμένη λύση για τη δημιουργία ενός ασφαλούς και αποτελεσματικού back-end τα οποία θα δούμε στην συνέχεια πιο αναλυτικά.

4.1 API διασύνδεση back-end με front-end

Το API [21] είναι το μέσο μέσω του οποίου το back-end συνδέεται με το front-end σε μια εφαρμογή. Παρέχει έναν τρόπο επικοινωνίας μεταξύ τους χρησιμοποιώντας καθορισμένους κανόνες και πρωτόκολλα. Ένα API που συνδέει το back-end με το front-end επιτρέπει στο front-end να στέλνει αιτήσεις στο back-end για δεδομένα και να λαμβάνει τις απαντήσεις που απαιτούνται για την εμφάνιση των πληροφοριών στον χρήστη. Το back-end εκτελεί την αναγκαία επεξεργασία και επιστρέφει τα αποτελέσματα στο front-end.

Από την άλλη πλευρά το REST (Representational State Transfer) αποτελεί ένα αρχιτεκτονικό στυλ που καθορίζει πώς πρέπει να σχεδιάζονται και να λειτουργούν δικτυακές εφαρμογές. Τα RESTful API ακολουθούν αυτές τις αρχές, χρησιμοποιώντας τυπικές μεθόδους HTTP για την ανταλλαγή δεδομένων. Κάθε αίτημα από έναν χρήστη πρέπει να περιέχει όλες τις πληροφορίες που απαιτούνται για την εκτέλεση του αιτήματος, ο εξυπηρετητής δεν πρέπει να διατηρεί κατάσταση χρήστη μεταξύ αιτημάτων. Οι πόροι είναι αντικείμενα ή υπηρεσίες που μπορούν να προσδιοριστούν με μοναδικούς αναγνωριστικούς (URLs). Οι διάφορες συσκευές πρέπει να επικοινωνούν με τον ίδιο τρόπο και αυτό προωθεί την απλότητα, την επεκτασιμότητα και την ανεξαρτησία των επιμέρους συστημάτων. Τέλος η πληροφορία για την εκτέλεση του αιτήματος βρίσκεται εξ ολοκλήρου στο αίτημα, και όχι στον εξυπηρετητή.

Σχεδίαση REST API



Εικόνα 10 Σχεδίαση REST API

Η εικόνα 8 περιέχει τα URLs που έχουν υλοποιηθεί από την μεριά του back-end για την σωστή επικοινωνία με οποιοδήποτε front-end όπως web page, smartphone ή tablet. Στην εικόνα 9 βλέπουμε τις αιτήσεις από το front-end προς το back-end και συγκεκριμένα για τη λίστα των habits ενός συγκεκριμένου χρήστη. Το front-end στέλνει το αίτημα μέσω ενός συγκεκριμένου URL (endpoint) που έχει οριστεί στο back-end. Το back-end λαμβάνει αυτό το αίτημα μέσω του REST API και αναζητά τα δεδομένα σχετικά με τα habits του συγκεκριμένου χρήστη από τη βάση δεδομένων. Θα επιστρέφει τα αποτελέσματα πίσω στο front-end εάν αυτά υπάρχουν, στοχεύοντας την παρουσίαση αυτών των δεδομένων στον χρήστη.

```
urlpatterns = [  
    path('api/v1/categories/', views.Categories.as_view(), name='categories_by_user'),  
    path('api/v1/categories/<int:pk>', views.CategoryDetails.as_view(), name='category_detail_by_user'),  
    path('api/v1/habits/', views.Habits.as_view(), name="habits_by_user"),  
    path('api/v1/habits/<int:pk>', views.HabitDetails.as_view(), name="habit_detail_by_user"),  
    path('signup/', views.UserCreate.as_view(), name='signup'),  
]
```

Εικόνα 11 RestApi urls

```

import java.util.List;
import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.DELETE;
import retrofit2.http.GET;
import retrofit2.http.Header;
import retrofit2.http.POST;
import retrofit2.http.PUT;
import retrofit2.http.Path;

public interface ApiService {
    // Endpoint to authenticate user and retrieve JWT token
    @POST("/api/token/")
    Call<JwtResponse> authenticateUser(@Body UserLogin userLogin);

    // Endpoint to sign up a new user
    @POST("/signup/")
    Call<Void> registerUser(@Body SignUpRequest signUpRequest);

    // Endpoint to get the list of habits for the user
    @GET("/api/v1/habits/")
    Call<List<Habit>> getUserHabits(@Header("Authorization") String authToken);

    // Endpoint to post a new habit
    @POST("/api/v1/habits/")
    Call<Habit> createHabit(@Header("Authorization") String authToken, @Body Habit
    habit);

    // Endpoint to update an existing habit
    @PUT("/api/v1/habits/{id}")
    Call<Habit> updateHabit(@Path("id") int habitId, @Header("Authorization") String
    authToken, @Body Habit habit);

    // Endpoint to delete a habit
    @DELETE("/api/v1/habits/{id}")
    Call<Void> deleteHabit(@Path("id") int habitId, @Header("Authorization") String
    authToken);
}

```

Εικόνα 12 Front-end αιτήσεις (requests) προς Back-end

Συνοπτικά, το REST API [22] στέλνει αιτήσεις και λαμβάνει απαντήσεις μεταξύ του back-end και του front-end, επιτρέποντας στις δύο πλευρές να αλληλεπιδρούν, εμφανίζοντας τα δεδομένα στους χρήστες μέσω του front-end με βάση τις ενέργειες που πραγματοποιούνται στο back-end. Το REST API καθορίζει τη δομή των αιτημάτων και των αποδόσεων, συμπεριλαμβανομένων των μεθόδων (GET, POST, PUT, DELETE κτλ) που χρησιμοποιούνται για την αλληλεπίδραση.

Μέθοδος	Συνήθης Χρήση
GET	Ανάγνωση πόρων
POST	Δημιουργία πόρων
PUT	Ενημέρωση/Αντικατάσταση πόρων
DELETE	Διαγραφή πόρων
PATCH	Ενημέρωση/Αντικατάσταση πόρων
HEAD	Ανάγνωση μεταδεδομένων για τους πόρους
OPTIONS	Ανάγνωση πληροφοριών για περαιτέρω επικοινωνία με την υπηρεσία

Πίνακας 1 Λειτουργίες HTTP μεθόδων



Η χρήση ενός καλά σχεδιασμένου REST API επιτρέπει την αποδοτική και ασφαλή επικοινωνία μεταξύ του back-end και του front-end σε μια εφαρμογή και συνήθως βασίζονται σε τυπικές μορφές δεδομένων όπως JSON ή XML. Είναι ένας σημαντικός τρόπος για τη δημιουργία εφαρμογών για να επικοινωνούν μεταξύ τους.

4.2 Βάση δεδομένων PostgreSQL

Η PostgreSQL, που αναλύεται στο συγκεκριμένο κεφάλαιο διότι όπως αναφέρθηκε παραπάνω είναι η βάση δεδομένων που επιλέχθηκε για την συγκεκριμένη εφαρμογή είναι ένα σύστημα διαχείρισης βάσεων δεδομένων [23] (DBMS) ανοιχτού κώδικα,. Αυτό σημαίνει ότι ο κώδικας του PostgreSQL είναι διαθέσιμος για το κοινό να τον εξετάσει, να τον τροποποιήσει και να τον χρησιμοποιήσει ελεύθερα. Επειδή η PostgreSQL είναι ανοιχτού κώδικα, μπορείτε να το χρησιμοποιήσετε δωρεάν χωρίς την ανάγκη να πληρώσετε για άδεια χρήσης της. Αυτό το καθιστά προσιτό για μικρές επιχειρήσεις, ανεξάρτητους προγραμματιστές και άλλους που θέλουν να χρησιμοποιήσουν μια ισχυρή βάση δεδομένων χωρίς περιορισμούς στην άδεια χρήσης.

Η PostgreSQL προσφέρει ευρεία υποστήριξη για stored procedures και επιτρέπει την εκτέλεση τους σε πολλές γλώσσες προγραμματισμού, συμπεριλαμβανομένων των Python, Java, Ruby, C/C++, PL/pgSQL η οποία είναι παρόμοια με την PL/SQL της Oracle και παρέχει προηγμένες δυνατότητες για τη δημιουργία stored procedures και triggers. Η PostgreSQL παρέχει εκατοντάδες built-in συναρτήσεις που καλύπτουν ένα ευρύ φάσμα λειτουργιών, συμπεριλαμβανομένων μαθηματικών, διαχείριση συμβολοσειρών, κρυπτογραφίας, και συμβατότητας με την Oracle. Τα triggers και τα stored procedures μπορούν να γραφούν σε γλώσσα C και να φορτωθούν στη βάση δεδομένων ως βιβλιοθήκη, παρέχοντας μεγάλη ευελιξία στην επέκταση των δυνατοτήτων της βάσης. Η PostgreSQL περιλαμβάνει ένα πλαίσιο που επιτρέπει τον ορισμό και τη δημιουργία custom data types, καθώς και βοηθητικές συναρτήσεις και τελεστές που περιγράφουν τη λειτουργία τους.

Είναι διαθέσιμη για όλα τα κύρια λειτουργικά συστήματα, περιλαμβανομένων του Linux, UNIX και Windows. Αυτό την καθιστά εξαιρετικά ευέλικτη και κατάλληλη για χρήση σε διάφορα περιβάλλοντα ανάπτυξης και παραγωγής. Οι χρήστες μπορούν να εγκαταστήσουν και να χρησιμοποιήσουν την PostgreSQL στο λειτουργικό σύστημα της επιλογής τους ανάλογα με τις ανάγκες τους. Η διαθεσιμότητα σε πολλά λειτουργικά συστήματα συμβάλλει στη διασφάλιση της ευρείας διάδοσης και υποστήριξης της PostgreSQL στην κοινότητα των προγραμματιστών και στον κόσμο των επιχειρήσεων.

Υπάρχει μια ενεργή κοινότητα προγραμματιστών και επαναδραστηριοποίησης για τη PostgreSQL που διασφαλίζει ότι ο κώδικας είναι συνεχώς ενημερωμένος και υποστηρίζεται. Αυτό σημαίνει ότι όταν αντιμετωπίζετε προβλήματα ή χρειάζεστε βοήθεια, μπορείτε να βασιστείτε σε μια ευρεία κοινότητα για υποστήριξη. Μπορείτε να προσαρμόσετε τη PostgreSQL στις ανάγκες σας επεκτείνοντας τον κώδικα, προσθέτοντας επεκτάσεις ή τροποποιώντας τον τρόπο λειτουργίας του. [24] Αυτό σας δίνει μεγάλη ευελιξία για να δημιουργήσετε προσαρμοσμένες λύσεις. Δεν είστε



εξαρτημένοι από έναν συγκεκριμένο προμηθευτή λογισμικού. Μπορείτε να αναπτύξετε και να διαχειριστείτε τη βάση δεδομένων σας ανεξάρτητα.

Η PostgreSQL είναι γνωστή για την ικανότητά της να διαχειρίζεται εύκολα μεγάλους αριθμούς ταυτόχρονων χρηστών και μεγάλου όγκου δεδομένων. Αυτό συνεισφέρει στο να την καθιστά κατάλληλη για εφαρμογές που απαιτούν υψηλή απόδοση και κλιμακωσιμότητα. Υποστηρίζει πολυνηματικότητα, επιτρέποντας πολλαπλές επικοινωνίες με τη βάση δεδομένων ταυτόχρονα. Αυτό είναι σημαντικό για εφαρμογές με πολλούς χρήστες που κάνουν αιτήσεις ταυτόχρονα. Ακόμα υποστηρίζει συναλλαγές, οι οποίες επιτρέπουν στους χρήστες να εκτελούν πολλαπλές εντολές SQL ως ένα ατομικό και συνεπές "έργο". Αυτό είναι κρίσιμο για τη σωστή διαχείριση δεδομένων σε περιβάλλοντα με πολλούς χρήστες. Επίσης, υποστηρίζει διάφορους τύπους δεδομένων και ευέλικτες δυνατότητες σχεδίασης πίνακα, επιτρέποντας την αποτελεσματική αποθήκευση και ανάκτηση πληροφοριών. Τέλος, η PostgreSQL διαθέτει ισχυρούς βελτιστοποιητές ερωτημάτων που μπορούν να βελτιώσουν την απόδοση των ερωτημάτων SQL, ακόμη και σε περιπτώσεις μεγάλων όγκων δεδομένων.

Η PostgreSQL είναι ένα πολύ ισχυρό και αξιόπιστο σύστημα διαχείρισης βάσεων δεδομένων (DBMS) που χρησιμοποιείται ευρέως σε πολλές εφαρμογές λόγω των πολλών του δυνατοτήτων.

Ορισμένα από τα βασικά χαρακτηριστικά του PostgreSQL περιλαμβάνουν:

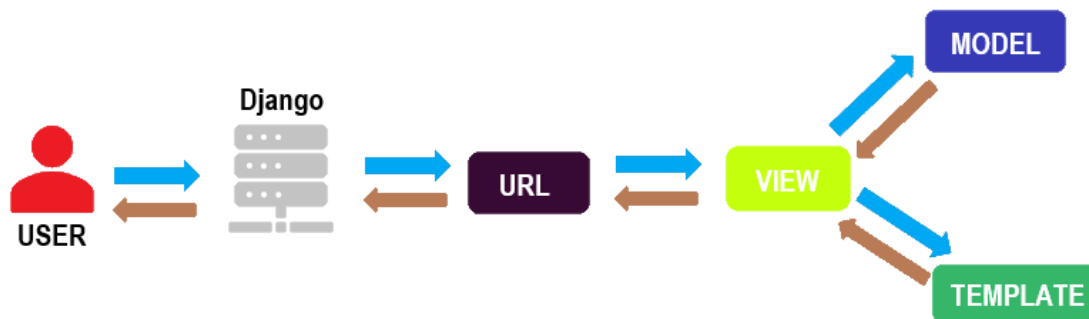
- **Σταθερότητα και Αξιοπιστία:** Η PostgreSQL είναι γνωστή για τη σταθερότητά του και την αξιοπιστία των δεδομένων που διαχειρίζεται. Παρέχει μηχανισμούς για να διασφαλίσει ότι τα δεδομένα είναι ασφαλή και προστατευμένα από απώλειες.
- **Ανοιχτός Κώδικας:** Η PostgreSQL είναι ένα σύστημα διαχείρισης βάσεων δεδομένων ανοιχτού κώδικα, που σημαίνει ότι είναι δωρεάν για χρήση και υποστηρίζεται από μια ενεργή κοινότητα.
- **Υποστήριξη Πολλαπλών Τύπων Δεδομένων:** Η PostgreSQL υποστηρίζει πολλαπλούς τύπους δεδομένων, συμπεριλαμβανομένων των γεωγραφικών, JSON και XML δεδομένων, παρέχοντας ευελιξία στην αποθήκευση ποικίλων μορφών πληροφοριών.
- **Πλούσιες Λειτουργίες:** Η PostgreSQL προσφέρει ποικιλία λειτουργιών όπως υποστήριξη για πολλαπλές γλώσσες προγραμματισμού, triggers, περίπλοκες ερωτήσεις SQL, αλληλεπίδραση με πληθώρα άλλων συστημάτων κ.ά.
- **Διαχείριση Συναλλαγών:** Υποστηρίζει συναλλαγές σε πολλαπλές επιθυμητές βάσεις δεδομένων, επιτρέποντας την ασφαλή διαχείριση αλλαγών και ενημερώσεων.

Συνολικά, η PostgreSQL προσφέρει μια ισχυρή λύση για τη διαχείριση των δεδομένων σε μια ευρεία γκάμα εφαρμογών και περιβαλλόντων. Με τις πολλές λειτουργίες και την ευελιξία που προσφέρει, η PostgreSQL αποτελεί επιλογή προτίμησης για πολλούς προγραμματιστές και επιχειρήσεις που αναζητούν μια σταθερή, ασφαλή και επεκτάσιμη βάση δεδομένων.

4.3 Django Framework

Το Django είναι ένα δημοφιλές ανοικτού κώδικα web framework βασισμένο σε Python, [25] υψηλού επιπέδου πλαίσιο ανάπτυξης λογισμικού για δημιουργία ιστού και εφαρμογών. Έχει σχεδιαστεί για να κάνει την ανάπτυξη της ιστοσελίδας πιο εύκολη και γρήγορη, παρέχοντας πολλά εργαλεία και βιβλιοθήκες που επιτρέπουν τη δημιουργία πλούσιων και ασφαλών εφαρμογών.

Επιτρέπει στους προγραμματιστές να εστιάσουν στη λογική της εφαρμογής, ακολουθεί το πρότυπο αρχιτεκτονικής MTV (Model-Template-View), το οποίο είναι μια ελαφρώς παραλλαγμένη μορφή του πιο γνωστού προτύπου MVC (Model-View-Controller) και χρησιμοποιεί τη γλώσσα προγραμματισμού Python. Τόσο στο MVC όσο και στο MTV, το μοντέλο (Model) αντιπροσωπεύει τα δεδομένα και τη λογική της εφαρμογής. Καθορίζει πώς δομούνται, αποθηκεύονται και προσπελούνται τα δεδομένα. Στο MTV, το πρότυπο (Template) αντιστοιχεί στην όψη (View) στο MVC. Τα πρότυπα χειρίζονται τη λογική παρουσίασης και καθορίζουν πώς θα πρέπει να παρουσιαστούν τα δεδομένα. Είναι υπεύθυνα για τη δημιουργία του HTML που στέλνεται στον χρήστη. Στο MTV, η όψη (View) αντιστοιχεί στον ελεγκτή (Controller) στο MVC. Οι όψεις διαχειρίζονται την αλληλεπίδραση μεταξύ του μοντέλου και του προτύπου. Χειρίζονται την είσοδο του χρήστη, επεξεργάζονται δεδομένα και καθορίζουν ποιο πρότυπο θα χρησιμοποιηθεί για την απεικόνιση. Επομένως, ενώ το MTV του Django μπορεί να έχει διαφορετικά ονόματα για ορισμένα στοιχεία σε σύγκριση με το παραδοσιακό MVC, οι υποκείμενες έννοιες είναι παρόμοιες. Η κύρια διάκριση βρίσκεται στη ονομασία και στις συγκεκριμένες ευθύνες που ανατίθενται σε κάθε στοιχείο εντός του πλαισίου.



Εικόνα 13 Django αρχιτεκτονικής MTV

Είναι δημοφιλές για την ευελιξία του, την ασφάλεια, την επίτευξη στόχων σε απλότητα και την εκτενή τεκμηρίωση. Περιστρέφεται γύρω από τις λειτουργίες CRUD (Create, Read, Update και Delete). Το CRUD μπορεί να εξηγηθεί καλύτερα ως μια προσέγγιση για τη δημιουργία μιας διαδικτυακής εφαρμογής Django. Σε γενικές γραμμές CRUD σημαίνει εκτέλεση λειτουργιών δημιουργίας, ανάκτησης, ενημέρωσης και διαγραφής σε έναν πίνακα σε μια βάση δεδομένων.

Με τη διαχείριση της βάσης δεδομένων, τη διαχείριση των URLs, και το διαχειριστικό περιβάλλον, [26] το Django καλύπτει πολλούς τομείς της ανάπτυξης και παρέχει μια ολοκληρωμένη λύση για τη δημιουργία ασφαλών και προηγμένων εφαρμογών.



Ορισμένα από τα χαρακτηριστικά του Django περιλαμβάνουν:

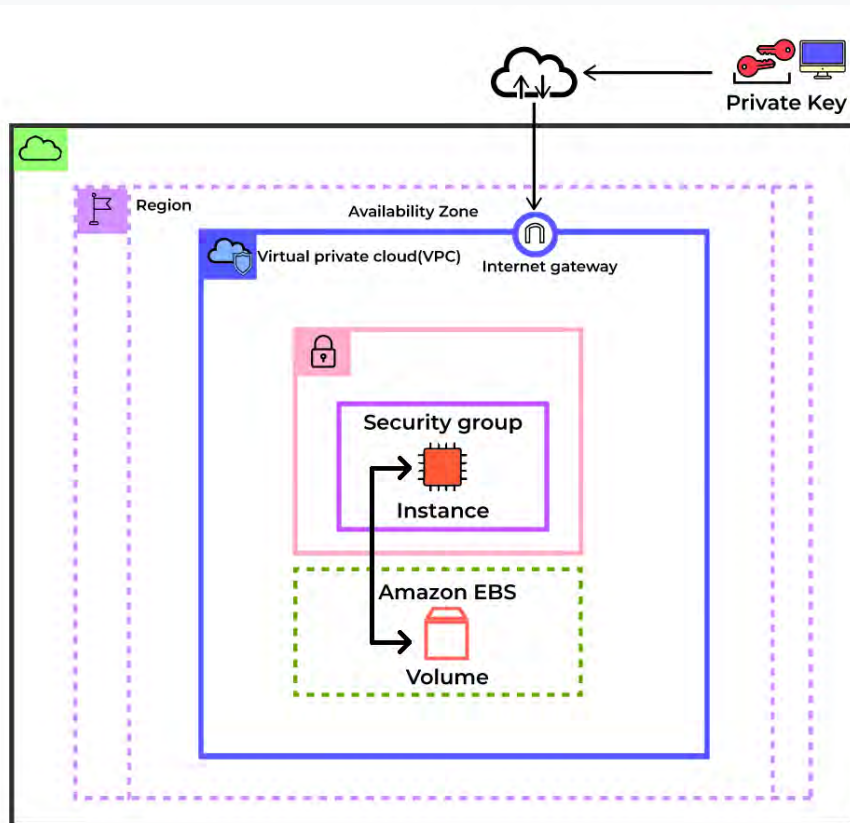
- **Ευελιξία:** Μπορείτε να χρησιμοποιήσετε το Django για τη δημιουργία ιστοσελίδων και εφαρμογών από απλές ιστοσελίδες μέχρι πολύπλοκες εφαρμογές. Το Django προσφέρει ένα πλήρες σύνολο λειτουργιών για την ανάπτυξη web εφαρμογών, περιλαμβανομένων εργαλείων για τη διαχείριση βάσεων δεδομένων, τη διαχείριση χρηστών, και τη διαχείριση URL.
- **Διαχείριση Βάσης Δεδομένων:** Το Django παρέχει ενσωματωμένη υποστήριξη για πολλούς τύπους βάσεων δεδομένων. Απλοποιεί τη δημιουργία, την ενημέρωση και τον έλεγχο της βάσης δεδομένων. Χρησιμοποιεί συχνά σχεσιακές βάσεις δεδομένων όπως το PostgreSQL, MySQL, SQLite, κ.λπ.
- **Διαχείριση URL:** Το Django χρησιμοποιεί ένα σύστημα διαχείρισης URL που καθορίζει πώς οι URLs προσομοιώνονται σε προβολές (views).
- **Διασύνδεση διαχειριστή (Admin Interface):** Το Django παρέχει ένα ενσωματωμένο σύστημα διαχείρισης για τη διαχείριση των δεδομένων σας, γνωστό ως Django Admin Interface.
- **Αυτοματοποιημένος διαχωρισμός HTML:** Το Django χρησιμοποιεί μοτίβα διαχωρισμού HTML για να επιτρέψει την ανεξαρτησία μεταξύ του λογισμικού και του σχεδιασμού.
- **Ασφάλεια:** Το Django ενσωματώνει αυτόματα πολλές βασικές ασφάλειας, όπως προστασία από SQL επιθέσεις, Cross-Site Scripting (XSS), και Cross-Site Request Forgery (CSRF).
- **Κοινότητα:** Ένα ισχυρό πλεονέκτημα του Django είναι η ενεργή κοινότητα που παρέχει υποστήριξη, εγκατάσταση πακέτων και ανταλλαγή ιδεών. Η κοινότητα του Django συμβάλλει στην εξέλιξη και βελτίωση του framework.

Μερικά από τα παραπάνω ήταν και οι βασικοί λόγοι για την επιλογή του Django ως back-end framework στην εφαρμογή Habit Tracker. Το Django σε συνδυασμό με την Python προσφέρει ένα πολύ ισχυρό πλαίσιο για την ανάπτυξη ασφαλών, αποδοτικών και εύκολων στη συντήρηση εφαρμογών ιστού. Η Python είναι μια γλώσσα προγραμματισμού που έχει σχεδιαστεί για να είναι ευανάγνωστη και εύκολη στη χρήση. Η συντακτική της δομή επιτρέπει στους προγραμματιστές να εκφράζουν ιδέες με λιγότερο κώδικα σε σύγκριση με άλλες γλώσσες, προσφέροντας έτσι ευκολία στην ανάπτυξη και συντήρηση κώδικα. Η Python παρέχει την ευκολία και την ευχρηστία, ενώ το Django φέρνει ένα εύρος λειτουργιών.

Συνεπώς, η συνδυασμένη χρήση Python και Django παρέχει τόσο την ευελιξία και ευχρηστία της γλώσσας προγραμματισμού όσο και τις λειτουργίες που επιτρέπουν τη γρήγορη ανάπτυξη εφαρμογών.

4.4 Amazon Elastic Compute Cloud (Amazon EC2)

Το Amazon Elastic Compute Cloud (Amazon EC2) [28] παρέχει κατ' απαίτηση, κλιμακούμενη υπολογιστική ικανότητα στο Amazon Web Services (AWS) Cloud, είναι μια υπηρεσία υπολογιστικού νέφους παρέχει ευελιξία όσον αφορά τον χρόνο λειτουργίας, το μέγεθος, τον τύπο και το λειτουργικό σύστημα των εικονικών server instances που χρησιμοποιούνται για τη φιλοξενία του back-end γενικά και συγκεκριμένα της εφαρμογής μας Habit Tracker.



Εικόνα 14 Αρχιτεκτονική του Amazon EC2

Η χρήση του Amazon EC2 μειώνει το κόστος υλικού, ώστε να μπορείτε να αναπτύσσετε και να αναπτύσσετε εφαρμογές πιο γρήγορα. Μπορείτε να χρησιμοποιήσετε το Amazon EC2 για να εκκινήσετε πολλούς ή λίγους εικονικούς διακομιστές που χρειάζεστε, μπορείτε να επιλέξετε τον τύπο των εικονικών διακομιστών (server instances) που ταιριάζει καλύτερα με τις απαιτήσεις της εφαρμογής σας και να διαχειριστείτε τους πόρους σας αποτελεσματικά, να διαμορφώσετε την ασφάλεια και τη δικτύωση και να διαχειριστείτε τον χώρο αποθήκευσης. Μπορείτε να προσθέσετε χωρητικότητα (αναβάθμιση) για να χειριστείτε εργασίες βαρέως υπολογισμού, όπως μηνιαίες ή ετήσιες διαδικασίες ή αυξήσεις στην επισκεψιμότητα ενός ιστότοπου. Όταν η χρήση μειώνεται, μπορείτε να μειώσετε ξανά τη χωρητικότητα (μείωση κλίμακας), αυτό προσφέρει τη δυνατότητα εύκολης αναμόρφωσης των resources, ανάλογα με τις ανάγκες της

εφαρμογής και εξασφαλίζει ένα ευέλικτο περιβάλλον για τη λειτουργία του back-end σε ένα αξιόπιστο cloud περιβάλλον υπολογισμού. Είναι ένα από τα βασικά στοιχεία του υπολογιστικού νέφους της Amazon και παρέχει μεγάλη ελευθερία και ευελιξία για τη δημιουργία και λειτουργία εφαρμογών σε κλίμακα. [27] Το Amazon Elastic Compute Cloud (Amazon EC2) προσφέρει πολλά χαρακτηριστικά και επιλογές για τη φιλοξενία εικονικών server instances στο AWS.

Ορισμένα από τα κύρια χαρακτηριστικά περιλαμβάνουν:

- **Ευελιξία των server instances:** Μπορείτε να επιλέξετε από μια ποικιλία διαθέσιμων τύπων server instances, καθένα από τα οποία προσφέρει διαφορετικές ποσότητες CPU, μνήμης RAM και αποθηκευτικού χώρου, επιτρέποντάς σας να επιλέξετε τον κατάλληλο τύπο για τις ανάγκες της εφαρμογής σας.
- **Λειτουργικά συστήματα:** Το Amazon EC2 υποστηρίζει διάφορα λειτουργικά συστήματα, όπως Linux, Windows, macOS και πολλά άλλα.
- **Αποθηκευτικός χώρος:** Μπορείτε να επιλέξετε τον τύπο αποθηκευτικού χώρου που χρειάζεστε, συμπεριλαμβανομένων των ενιαίων σκληρών δίσκων (EBS), του αποθηκευτικού χώρου SSD, και άλλων.
- **Ασφάλεια:** Το Amazon EC2 προσφέρει διάφορα μέσα ασφαλείας, συμπεριλαμβανομένων των Virtual Private Clouds (VPCs), δυνατοτήτων διαχείρισης πρόσβασης και ελέγχου δικαιωμάτων.
- **Διαχείριση:** Το Amazon EC2 παρέχει διάφορα εργαλεία για τη διαχείριση των server instances, συμπεριλαμβανομένης της δυνατότητας δημιουργίας εικόνων (images) του server instance σας για αντιγραφή ή επαναχρησιμοποίηση.
- **Ανθεκτικότητα:** Οι server instances του Amazon EC2 είναι σχεδιασμένες για ανθεκτικότητα, και μπορούν να ανακάμψουν από αποτυχίες υλικού.
- **Καθυστερήση (Latency):** Το Amazon EC2 προσφέρει επιλογές για την τοποθέτηση των server instances σας σε διάφορες γεωγραφικές περιοχές, επιτρέποντάς σας να μειώσετε την καθυστέρηση για τους χρήστες σας.

Αυτά είναι μόνο ορισμένα από τα χαρακτηριστικά του Amazon EC2. Ανάλογα με τις ανάγκες και τις απαιτήσεις της εφαρμογής σας, μπορείτε να επιλέξετε και να διαμορφώσετε τα server instances προς το όφελός σας.

4.5 Σχεδιασμός και ανάλυση της εφαρμογής

Σε αυτό το κεφάλαιο έγινε αναφορά των διαφόρων επιλογών για την υλοποίηση του backend και της βάσης δεδομένων της εφαρμογής. Καθώς ολοκληρώθηκε η ανάλυση των απαιτήσεων και κατά τον σχεδιασμό ήταν απαραίτητη η επιλογή των κατάλληλων εργαλείων για την ανάπτυξη της εφαρμογής. Στο backend απαραίτητη ήταν η επιλογή της κατάλληλης γλώσσας με το κατάλληλο framework που θα έχουν σαν αποτέλεσμα την ασφάλεια των δεδομένων, την επεκτασιμότητα και την ευκολία στην συντήρηση. Η



python είναι μία γλώσσα με πλούσιες βιβλιοθήκες και σε συνδυασμό με το Django ήταν η κατάλληλη επιλογή για το συγκεκριμένο project. Για την βάση δεδομένων επιλέχθηκε η PostgreSQL διότι ως σύστημα διαχείρισης δεδομένων είναι ολοκληρωμένο και έχει υψηλή απόδοση και ανθεκτικότητα. Ακόμα η Django έχει εξαιρετική υποστήριξη για την PostgreSQL, η οποία παρέχει την δυνατότητα κλιμάκωσης και επεκτασιμότητας τόσο σε κάθετο όσο και σε οριζόντιο επίπεδο. Ένα επιπλέον χαρακτηριστικό είναι πως υποστηρίζει πλήρεις συναλλαγές ACID (Atomicity, Consistency, Isolation, Durability) που είναι κρίσιμες για την διασφάλιση της ακεραιότητας της βάσης δεδομένων ειδικά στο περιβάλλον της συγκεκριμένης εφαρμογή που περιλαμβάνει πολλαπλότητα χρηστών.

Στην ανάλυση των απαιτήσεων για την διασφάλιση της σωστής λειτουργίας της εφαρμογής κρίθηκε απαραίτητο να δοθεί προσοχή δύο κύριους πυλώνες τον χρήστη και τα δεδομένα. Στην πρώτη περίπτωση η εφαρμογή έπρεπε να αναπτυχθεί με τέτοιο τρόπο ώστε να είναι εύκολη η τροποποίηση του συστήματος ανάλογα με τους χρήστες που την χρησιμοποιούν και γι' αυτό το λόγο επιλέχθηκε το AWS της Amazon ως χώρος φιλοξενίας του backend συστήματος και της βάσης δεδομένων. Η δυνατότητα που παρέχει το Amazon EC2 χαρακτηρίζεται από εξαιρετική ευελιξία και υψηλό βαθμό προσαρμοστικότητας που είναι απαραίτητο σε καινούριες εφαρμογές.

Στην εικόνα 12 δίνεται το διάγραμμα UML (Unified Modeling Language) το οποίο απεικονίζει τη σχεδίαση της βάσης δεδομένων για την εφαρμογή διαχείρισης συνηθειών. Το διάγραμμα δείχνει τέσσερις κλάσεις: Habit, Category, HabitRecord, και User, μαζί με τις σχέσεις μεταξύ τους. Η κλάση Habit περιέχει πληροφορίες για μια συνήθεια, με τα ακόλουθα χαρακτηριστικά (attributes): ένα ακεραίο αναγνωριστικό (id), το όνομα (name), περιγραφή (description), λόγος για την έναρξη της συνήθειας (reason), τη μέρα της εβδομάδας (day_of_the_week), ημερομηνία έναρξης (start_date), ημερομηνία λήξης (end_date), καθώς και boolean τιμές για το αν είναι ενεργή (is_active), για το αν έχει ολοκληρωθεί (is_complete), τον χρόνο δημιουργίας (created_at), τον χρόνο τελευταίας ενημέρωσης (updated_at) και τον χρόνο της ημέρας (time_of_the_day). Η κλάση Category απεικονίζει την κατηγορία στην οποία μπορεί να ανήκει μια συνήθεια. Διαθέτει τα εξής χαρακτηριστικά: id, name, description, created_at, updated_at, is_active, και color. Η κλάση HabitRecord καταγράφει πότε μια συνήθεια έχει εκτελεστεί. Περιλαμβάνει ένα id, μια ημερομηνία (date), και μια boolean τιμή (performed) που δείχνει αν η συνήθεια εκτελέστηκε.

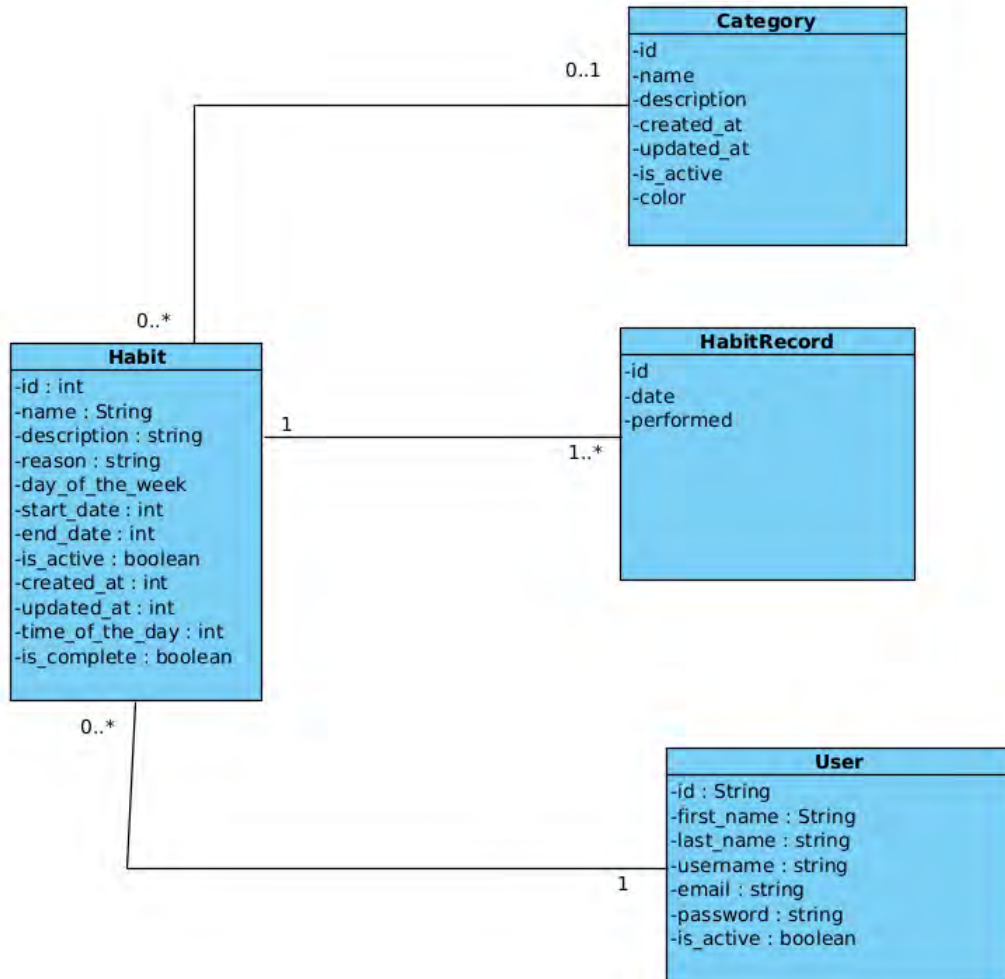
Τέλος, η κλάση User περιγράφει έναν χρήστη της εφαρμογής με τα εξής χαρακτηριστικά: id, first_name, last_name, username, email, password, και is_active.

Οι σχέσεις μεταξύ των κλάσεων είναι:

- Ένα Habit μπορεί να ανήκει σε καμία ή μία Category (0..1).
- Ένα Habit μπορεί να έχει πολλά HabitRecord (1..*).
- Ένας User έχει να έχει από 0 μέχρι * Habit.

Η κανονικοποίηση που ακολουθεί η βάση δεδομένων είναι ή τρίτη κανονική μορφή (3NF), καθώς κάθε χαρακτηριστικό εξαρτάται μόνο από το κλειδί της κλάσης στην

οποία ανήκει, δεν υπάρχουν μεταβατικές εξαρτήσεις, και τα δεδομένα είναι διαχωρισμένα με τρόπο που μειώνει την επανάληψη και την πιθανότητα ασυνέπιας.



Εικόνα 15 UML βάσης δεδομένων



5 GRASP patterns αρχιτεκτονικά και σχεδιαστικά πρότυπα

Τα GRASP (General Responsibility Assignment Software Patterns) είναι μια συλλογή από πρότυπα σχεδιασμού λογισμικού που χρησιμοποιούνται για την ανάθεση αρμοδιοτήτων στις κλάσεις στη διαδικασία ανάπτυξης λογισμικού. [29] Τα GRASP patterns αποτελούν ένα μέρος της μεθοδολογίας ανάπτυξης λογισμικού που συνδυάζει τη χρήση της Unified Modeling Language (UML) με αρχές αντικειμενοστραφούς ανάπτυξης. Επίσης παρέχουν κατευθυντήριες γραμμές για την ανάθεση αρμοδιοτήτων στις κλάσεις, βοηθώντας στον σωστό σχεδιασμό και την κατανόηση των σχέσεων μεταξύ τους. Είναι σημαντικό να σημειωθεί ότι οι αρχές των GRASP patterns είναι συντακτικά ανεξάρτητες και δεν εξαρτώνται από κάποια συγκεκριμένη γλώσσα προγραμματισμού. Αυτά τα πρότυπα βοηθούν στην οργάνωση του κώδικα και τη δομή του λογισμικού. [34] Ένας αρχιτέκτονας λογισμικού μπορεί να εφαρμόσει τα GRASP patterns και άλλα σχεδιαστικά πρότυπα για την αποτελεσματική ανάπτυξη και υλοποίηση της εφαρμογής.

Κατά την εφαρμογή των GRASP patterns, ένας αρχιτέκτονας λογισμικού μπορεί να εφαρμόσει διάφορα πρότυπα για την ανάθεση αρμοδιοτήτων στις κλάσεις, τα οποία είναι τα εξής :

1. Δημιουργός (Creator)
2. Πληροφορημένος Ειδικός (Information Expert)
3. Χαμηλή Σύζευξη (Low Coupling)
4. Ελεγκτής (Controller)
5. Υψηλή Συνοχή (High Cohesion)
6. Πολυμορφισμός (Polymorphism)
7. Καθαρή Επινόηση (Pure Fabrication)
8. Πλαγιότητα (Indirection)
9. Προστατευμένες Παραλλαγές (Protected Variations)

Κάθε πρότυπο εστιάζει σε διαφορετικές πτυχές της ανάθεσης αρμοδιοτήτων, όπως η δημιουργία αντικειμένων, ο έλεγχος ροής, η οργάνωση των δεδομένων και η αποφυγή περίπλοκων σχέσεων μεταξύ κλάσεων.

Από αρχιτεκτονική άποψη, συγκεκριμένα πρότυπα σχεδιασμού, όπως το Model-View-Controller (MVC), το Model-View-ViewModel (MVVM) και άλλα, [30] μπορούν να εφαρμοστούν για την οργάνωση και τη διαχείριση των διαφόρων στρωμάτων της εφαρμογής. Κάθε πρότυπο σχεδιασμού έχει τα δικά του πλεονεκτήματα και χρησιμοποιείται σε συγκεκριμένα σενάρια ανάπτυξης, επικεντρώνοντας σε συγκεκριμένες αρχές και αξίες που ορίζουν τη δομή και τη λειτουργία του λογισμικού.

Καθώς επεκτείναμε την εφαρμογή μας, θα μπορούμε να εξετάσουμε τα διάφορα GRASP patterns και πρότυπα σχεδιασμού, προσαρμόζοντάς τα στις ανάγκες και την εξέλιξη της εφαρμογής μας.

5.1 Σχέση UML και GRASP

Το UML (Unified Modeling Language) και τα GRASP (General Responsibility Assignment Software Patterns) [29] είναι σχετικές αλλά διαφορετικές έννοιες στον χώρο του λογισμικού. Το UML είναι ένα γλώσσα μοντελοποίησης που χρησιμοποιείται για να απεικονίσει, να περιγράψει και να σχεδιάσει το λογισμικό, ενώ τα GRASP αποτελούν ένα σύνολο αρχών που χρησιμοποιούνται για να αντιστοιχίσουν ευθύνες σε αντικείμενα στο σχεδιασμό λογισμικού.

Το UML μπορεί να χρησιμοποιηθεί για να απεικονίσει διαγράμματα κλάσεων, διαγράμματα δραστηριοτήτων, διαγράμματα ακολουθίας, διαγράμματα καταστάσεων και άλλα, παρέχοντας ένα οπτικό εργαλείο για τον σχεδιασμό και την κατανόηση ενός συστήματος λογισμικού. [31] Από την άλλη πλευρά, τα GRASP αντιπροσωπεύουν μια σειρά από αρχές σχεδιασμού που βοηθούν στην ανάθεση των ευθυνών στα αντικείμενα του λογισμικού. Καθορίζουν τον τρόπο με τον οποίο τα αντικείμενα θα πρέπει να αντιδρούν σε ερωτήματα και εντολές, βάσει των αρχών της ευθύνης και της πληροφόρησης.

Συνοψίζοντας, το UML χρησιμοποιείται για την αναπαράσταση και τον σχεδιασμό του συστήματος, ενώ τα GRASP αφορούν τον τρόπο ανάθεσης ευθυνών σε αντικείμενα και τις αρχές σχεδιασμού που πρέπει να ακολουθηθούν για να επιτευχθεί αυτό. Συχνά, το UML μπορεί να χρησιμοποιηθεί για να απεικονίσει την εφαρμογή των αρχών των GRASP σε ένα σχέδιο λογισμικού.

5.2 Τι είναι τα πρότυπα

Τα πρότυπα, στον τομέα της τεχνολογίας και του σχεδιασμού, αναφέρονται σε καθολικές οδηγίες, μοτίβα ή αρχές που καθορίζουν τον τρόπο με τον οποίο πρέπει να λειτουργεί ή να αναπτύσσεται ένα σύστημα, ένα προϊόν ή μια διαδικασία. [29] Τα πρότυπα μπορούν να είναι αρχές που ισχύουν για συγκεκριμένους τομείς ή μοντέλα που έχουν αποδειχθεί αποτελεσματικά για την ανάπτυξη λογισμικού, τη διαχείριση έργων, την αρχιτεκτονική ή άλλους τομείς. Αυτά τα πρότυπα προσφέρουν μια κοινή γλώσσα και καθοδήγηση προς την επίλυση συγκεκριμένων προβλημάτων ή για τη βελτίωση της ποιότητας ή της απόδοσης των προϊόντων. Αυτά τα πρότυπα είναι συχνά το αποτέλεσμα βέλτιστων πρακτικών και εμπειριών από ειδικούς του συγκεκριμένου τομέα. Μπορούν να παρέχουν ένα πλαίσιο αναφοράς για τον σχεδιασμό, την ανάπτυξη ή τη διαχείριση, βοηθώντας να εξασφαλίσουν την αποτελεσματικότητα, την αξιοπιστία και τη συνοχή στην υλοποίηση έργων.

Τα σχεδιαστικά πρότυπα (ή design patterns) αναφέρονται σε καταγεγραμμένες λύσεις για επαναλαμβανόμενα σχεδιαστικά προβλήματα στον τομέα της ανάπτυξης λογισμικού. [32] Τα πρότυπα αυτά δημιουργήθηκαν για να βοηθήσουν τους προγραμματιστές να αντιμετωπίσουν ή να αντιληφθούν προβλήματα που εμφανίζονται συχνά και να προσφέρουν γνωστές και δοκιμασμένες λύσεις. [33] Κάθε πρότυπο περιγράφει ένα πρόβλημα που συναντάται συχνά στην ανάπτυξη λογισμικού και παρέχει μια γενική δομή που μπορεί να χρησιμοποιηθεί για τη λύση του. Κάθε πρότυπο



έχει ονομασίες, περιγραφές του προβλήματος και της λύσης, καθώς και παραδείγματα εφαρμογής του.

Αυτά τα πρότυπα προέρχονται από τη συσσώρευση εμπειριών προγραμματιστών και αρχιτεκτόνων λογισμικού και αποτελούν ένα είδος βιβλιοθήκης γνώσεων. Εφαρμόζονται για να βοηθήσουν στην αποτελεσματική ανάπτυξη λογισμικού, προσφέροντας επαληθευμένες λύσεις για γνωστά προβλήματα σχεδιασμού.

5.3 Σχεδιαστικά πρότυπα GRASP

Τα συγκεκριμένα πρότυπα διευκολύνουν τον προγραμματιστή στον καθορισμό των ευθυνών, των σχέσεων και των αλληλεπιδράσεων μεταξύ των διαφόρων στοιχείων του λογισμικού. [29] Αυτά τα πρότυπα παρέχουν έναν οδηγό για τη σωστή οργάνωση του κώδικα, επιτρέποντας την ανάπτυξη εύελικτου, [31] ανταποκρίνονται στις αλλαγές και εύκολου στη συντήρηση λογισμικού.

Μερικά από τα βασικά GRASP που έχουν περιγραφεί είναι:

5.3.1 Δημιουργός (Creator)

Το πρότυπο σχεδιασμού "Δημιουργός" (Creator) ανήκει στα GRASP (General Responsibility Assignment Software Patterns) και επικεντρώνεται στη διάθεση των αρμοδιοτήτων για τη δημιουργία νέων αντικειμένων ή κλάσεων σε μια σχεδίαση λογισμικού. Το πρότυπο "Δημιουργός" βοηθά στον καθορισμό του ποιού αντικειμένου πρέπει να είναι υπεύθυνη για τη δημιουργία άλλων αντικειμένων.

Οι κύριοι στόχοι του προτύπου "Δημιουργός" είναι:

- **Αποφυγής Πλεονάσματος Ευθύνης (Avoiding Excessive Responsibility):** Ορισμένα αντικείμενα ή κλάσεις έχουν τη φυσική ή λογική αρμοδιότητα να δημιουργούν άλλα αντικείμενα ή κλάσεις. Το πρότυπο "Δημιουργός" βοηθά στο να τοποθετηθεί η δημιουργία στο κατάλληλο μέρος.
- **Προώθηση της Επαναχρησιμοποίησης (Promoting Reusability):** Η επιλογή του σωστού αντικειμένου για τη δημιουργία άλλων αντικειμένων βοηθά στη δημιουργία εύελικτου κώδικα που μπορεί εύκολα να επαναχρησιμοποιηθεί.
- **Προαγωγή της Συντήρησης (Aiding Maintainability):** Ορισμένα αντικείμενα είναι περισσότερο κατάλληλα ή πιο ενδεδειγμένα για τη δημιουργία άλλων. Ο καθορισμός του αντικειμένου "δημιουργού" μπορεί να βελτιώσει τη συντηρησιμότητά του κώδικα.

Ο στόχος του "Δημιουργού" είναι να καθορίσει ποιο αντικείμενο θα πρέπει να είναι υπεύθυνο για τη δημιουργία νέων αντικειμένων ή κλάσεων, βασιζόμενο στη φύση του προβλήματος και των σχέσεών τους. Η σωστή ανάθεση της ευθύνης της δημιουργίας μπορεί να βελτιώσει τη δομή του κώδικα, την ευελιξία και τη συντηρησιμότητά του.

Πρόβλημα: Στον σχεδιασμό λογισμικού, υπάρχει συχνά η ανάγκη για δημιουργία νέων αντικειμένων ή κλάσεων. Το πρόβλημα είναι πού πρέπει να τοποθετηθεί η αρμοδιότητα



για τη δημιουργία αυτών των αντικειμένων, ώστε να εξασφαλιστεί η σωστή δομή και ο έλεγχος του κώδικα.

Λύση (Πρότυπο "Δημιουργός"): Η λύση που προτείνεται από το πρότυπο "Δημιουργός" είναι να αναθέσουμε την ευθύνη για τη δημιουργία νέων αντικειμένων σε κατάλληλα αντικείμενα ή κλάσεις.

Ένα παράδειγμα μπορεί να είναι ένα σύστημα κατασκευής αυτοκινήτων. Έστω ότι έχουμε μια κλάση "Κατασκευαστής Αυτοκινήτου" που αναλαμβάνει τη δημιουργία αντικειμένων της κλάσης "Αυτοκίνητο". Η κλάση "Κατασκευαστής Αυτοκινήτου" μπορεί να περιλαμβάνει μεθόδους για την ανάθεση των χαρακτηριστικών του αυτοκινήτου, όπως μάρκα, μοντέλο, χρώμα, κινητήρας, κ.λπ. Κατά τη δημιουργία ενός νέου αυτοκινήτου, η κλάση "Κατασκευαστής Αυτοκινήτου" είναι υπεύθυνη για τη δημιουργία και αρχικοποίηση του αντικειμένου της κλάσης "Αυτοκίνητο".

Κάποια αντικείμενα ή κλάσεις μπορεί να είναι πιο φυσικά ή λογικά κατάλληλα για τη δημιουργία νέων αντικειμένων. Επομένως, σύμφωνα με το πρότυπο "Δημιουργός", ορισμένα αντικείμενα ή κλάσεις πρέπει να έχουν την ευθύνη για τη δημιουργία άλλων. Αυτό βελτιώνει τη συντηρησιμότητά του κώδικα και την επαναχρησιμοποίηση, καθώς καθιστά τη δομή του πιο ευέλικτη και καλύτερα οργανωμένη. Οπότε, προς αποφυγή πλεονάσματος ευθύνης και για τη βελτιστοποίηση της δομής του λογισμικού, το πρότυπο "Δημιουργός" καθορίζει ποια αντικείμενα πρέπει να έχουν την αρμοδιότητα για τη δημιουργία νέων αντικειμένων ή κλάσεων.

5.3.2 Πληροφορημένος Ειδικός (Information Expert)

Το πρότυπο "Πληροφορημένος Ειδικός" (Information Expert) είναι ένα από τα βασικά GRASP (General Responsibility Assignment Software Patterns) στον τομέα του σχεδιασμού λογισμικού. Αποσκοπεί στη σωστή ανάθεση των ευθυνών και των λειτουργιών στα αντικείμενα ή τις κλάσεις που διαθέτουν την κατάλληλη γνώση και πληροφορίες.

Οι κύριοι στόχοι του προτύπου "Πληροφορημένος Ειδικός" είναι:

- **Ανάθεση Ευθυνών βασισμένη στη Γνώση:** Το πρότυπο αυτό προσδιορίζει ποιο αντικείμενο ή ποια κλάση θα πρέπει να αναλάβει μια ευθύνη ή να εκτελέσει μια λειτουργία βασισμένο στη γνώση που διαθέτει. Αυτό εξασφαλίζει ότι οι λειτουργίες ή ευθύνες ανατίθενται σε αντικείμενα που έχουν την καλύτερη γνώση και την πληροφόρηση που απαιτούνται για να εκτελέσουν επιτυχώς τη συγκεκριμένη εργασία.
- **Βελτιστοποίηση της Συντήρησης:** Η ανάθεση των ευθυνών σε αντικείμενα με την καλύτερη γνώση σχετικά με συγκεκριμένες λειτουργίες βοηθά στη διαχείριση και συντήρηση του κώδικα. Οι αντικειμενοστραφείς ευθύνες μπορούν να διευκολύνουν την αντιμετώπιση αλλαγών και επεκτάσεων στο λογισμικό.
- **Βελτίωση της Απόδοσης:** Η αντικειμενοστραφής προσέγγιση επιτρέπει στο λογισμικό να λειτουργεί με μεγαλύτερη αποδοτικότητα, καθώς οι ευθύνες



ανατίθενται σε αντικείμενα που διαθέτουν τη σχετική γνώση για την εκτέλεση συγκεκριμένων λειτουργιών.

Ο στόχος είναι να βελτιστοποιηθεί η διαχείριση των ευθυνών και να εξασφαλιστεί ότι οι εργασίες ανατίθενται σε αντικείμενα ή κλάσεις που διαθέτουν την κατάλληλη γνώση, βελτιώνοντας έτσι τη συνολική απόδοση και τη συντηρησιμότητα του λογισμικού.

Πρόβλημα: Κατά τον σχεδιασμό μιας εφαρμογής, συχνά υπάρχουν πολλά αντικείμενα ή κλάσεις που χρειάζονται πληροφορίες ή πρέπει να εκτελέσουν συγκεκριμένες λειτουργίες. Το πρόβλημα είναι ποιο αντικείμενο πρέπει να έχει την κατάλληλη γνώση και πληροφόρηση για να εκτελέσει αυτές τις λειτουργίες.

Λύση (Πρότυπο "Πληροφορημένος Ειδικός"): Η λύση που προτείνεται από το πρότυπο "Πληροφορημένος Ειδικός" είναι να αναθέσουμε την αρμοδιότητα εκεί όπου υπάρχει η καλύτερη γνώση και πληροφόρηση που απαιτείται για να εκτελεστούν συγκεκριμένες λειτουργίες.

Ένα παράδειγμα μπορεί να είναι ένα σύστημα διαχείρισης παραγγελιών. Έστω ότι έχουμε μια κλάση "Παραγγελία" που περιέχει τα δεδομένα όπως προϊόντα, ποσότητες και τιμές. Αν θέλουμε να υπολογίσουμε το συνολικό κόστος της παραγγελίας, θα ανατίθεται η ευθύνη στην κλάση "Παραγγελία" καθώς διαθέτει τα απαραίτητα δεδομένα για να εκτελέσει αυτήν τη λειτουργία.

Το πρότυπο "Πληροφορημένος Ειδικός" προτείνει ότι η ευθύνη πρέπει να ανατίθεται σε εκείνο το αντικείμενο ή την κλάση που διαθέτει την απαραίτητη γνώση, πληροφόρηση και δεξιότητες για να εκτελέσει συγκεκριμένες ενέργειες. Αυτό βελτιώνει τη συντηρησιμότητα του κώδικα και την αποτελεσματικότητα της εφαρμογής, καθώς οι ευθύνες ανατίθενται σε εκείνα τα αντικείμενα που έχουν τη μεγαλύτερη γνώση για να εκτελέσουν τις συγκεκριμένες εργασίες.

5.3.3 Χαμηλή Σύζευξη (Low Coupling)

Το πρότυπο "Χαμηλή Σύζευξη" (Low Coupling) είναι ένα από τα βασικά μοτίβα σχεδιασμού λογισμικού που ανήκει στη συλλογή των GRASP (General Responsibility Assignment Software Patterns). Το "Χαμηλή Σύζευξη" επικεντρώνεται στον τρόπο με τον οποίο οι κλάσεις και τα αντικείμενα σε ένα σύστημα λογισμικού συνδέονται ή εξαρτώνται μεταξύ τους.

Οι κύριοι στόχοι του προτύπου "Χαμηλή Σύζευξη" είναι:

- **Μείωση των Εξαρτήσεων:** Στοχεύει στη μείωση των συνδέσεων μεταξύ κλάσεων και αντικειμένων. Αυτό επιτρέπει τη μείωση της εξάρτησης μιας κλάσης από άλλες, επιτρέποντας την ανεξάρτητη τροποποίηση και εξέλιξη των κλάσεων χωρίς να επηρεάζονται υπερβολικά άλλα μέρη του συστήματος.
- **Αύξηση της Ευελιξίας και της Ανακατασκευής:** Με τη μείωση των συνδέσεων μεταξύ των κλάσεων, ο κώδικας γίνεται πιο εύλικτος και πιο εύκολος στην ανακατασκευή ή την αναδιοργάνωση. Αυτό επιτρέπει στο λογισμικό να προσαρμόζεται πιο εύκολα σε νέες απαιτήσεις.



- **Βελτιστοποίηση της Συντήρησης και της Διαχείρισης του Κώδικα:** Με τη μείωση των συνδέσεων μεταξύ των κλάσεων, η συντήρηση και η διαχείριση του κώδικα γίνονται πιο εύκολες. Αυτό επιτρέπει την αποφυγή περιττών εξαρτήσεων μεταξύ των τμημάτων του κώδικα, επιτρέποντας έτσι την αποδοτική συντήρηση και επέκταση του λογισμικού.

Η "Χαμηλή Σύζευξη" επιδιώκει τη δημιουργία μιας δομής λογισμικού όπου οι διάφορες κλάσεις είναι λιγότερο συσχετισμένες μεταξύ τους, ενθαρρύνοντας την επαναχρησιμοποίηση του κώδικα, την ευκολία συντήρησης και τη διαχείριση μεγάλων συστημάτων λογισμικού.

Πρόβλημα: Όταν στο λογισμικό υπάρχει υψηλό επίπεδο σύζευξης, δηλαδή οι διάφορες συνιστώσες ή κλάσεις είναι έντονα συνδεδεμένες, αλλαγές σε μία συνιστώσα μπορεί να επηρεάσουν σημαντικά άλλες, δυσκολεύοντας τη συντήρηση, την ανάπτυξη και την επαναχρησιμοποίηση του κώδικα.

Λύση (Πρότυπο "Χαμηλή Σύζευξη"): Η χαμηλή σύζευξη επιτυγχάνεται με τη διάκριση λειτουργιών και καθηκόντων σε ανεξάρτητες και απομονωμένες κλάσεις ή συνιστώσες. Αυτό γίνεται μέσω της χρήσης διεπαφών, αφήνοντας κάθε κλάση να εκτελεί συγκεκριμένες λειτουργίες χωρίς να γνωρίζει τις εσωτερικές λεπτομέρειες των υπόλοιπων. Αυτό επιτρέπει την αλλαγή μίας συνιστώσας χωρίς να επηρεάζει τις υπόλοιπες, επιτρέποντας την ευκολότερη συντήρηση και επέκταση του λογισμικού. Επίσης, μειώνει τον κίνδυνο σφαλμάτων και διευκολύνει την επαναχρησιμοποίηση των συνιστωσών ή των κλάσεων.

Ένα παράδειγμα μπορεί να είναι ένα σύστημα διαχείρισης βιβλιοθήκης. Έστω ότι έχουμε δύο κλάσεις, την "Κλάση Βιβλίο" και την "Κλάση Βιβλιοθήκη". Η "Κλάση Βιβλιοθήκη" χειρίζεται την προσθήκη και την αφαίρεση βιβλίων. Αν η "Κλάση Βιβλιοθήκη" χρησιμοποιεί μια διεπαφή (interface) για να αλληλεπιδρά με την "Κλάση Βιβλίο" αντί να εξαρτάται απευθείας από την υλοποίηση της "Κλάσης Βιβλίο", τότε υπάρχει χαμηλή σύζευξη. Αν σε κάποιο σημείο αλλάξουμε την υλοποίηση της "Κλάσης Βιβλίο", η "Κλάση Βιβλιοθήκη" δεν θα επηρεαστεί εάν διατηρήσει τη σύζευξή της με τη διεπαφή αντί να εξαρτάται απευθείας από την υλοποίηση της.

Η χαμηλή σύζευξη βοηθά στη διατήρηση και επέκταση του κώδικα, καθώς καθιστά τις αλλαγές πιο τοπικές και ελαφρές, χωρίς να επηρεάζουν υπερβολικά τις υπόλοιπες συνιστώσες του συστήματος.

5.3.4 Ελεγκτής (Controller)

Το πρότυπο "Ελεγκτής" (Controller) είναι ένα από τα βασικά μοτίβα σχεδιασμού που ανήκουν στη συλλογή των GRASP (General Responsibility Assignment Software Patterns). Στο πλαίσιο του λογισμικού, ο "Ελεγκτής" χρησιμοποιείται για τον διαχωρισμό της λογικής της εφαρμογής από τη διεπαφή χρήστη, αναλαμβάνοντας τον έλεγχο της αλληλεπίδρασης με τον χρήστη ή εξωτερικά συστήματα.

Οι κύριοι στόχοι του προτύπου "Ελεγκτής" είναι:



- **Διαχωρισμός της Λογικής της Εφαρμογής:** Ο ελεγκτής επιτρέπει τη διάκριση της επιχειρησιακής λογικής (ή λογικής επεξεργασίας) της εφαρμογής από τη λογική της διεπαφής χρήστη. Αυτός ο διαχωρισμός βοηθά στη διατήρηση της σαφούς δομής του λογισμικού.
- **Κεντρικός Σημείο Ελέγχου:** Ο ελεγκτής είναι υπεύθυνος για την καθοδήγηση της ροής της εφαρμογής, ελέγχοντας την είσοδο και την έξοδο από τη διεπαφή χρήστη προς τις υπόλοιπες κλάσεις του συστήματος.
- **Εξυπηρέτηση των Αιτημάτων του Χρήστη:** Ο ελεγκτής ανταποκρίνεται στις εντολές ή τα αιτήματα του χρήστη, ενεργώντας ως ενδιάμεσος μεταξύ της διεπαφής χρήστη και των υπόλοιπων στοιχείων της εφαρμογής.

Ο στόχος είναι να διασφαλιστεί η οργάνωση και η αποτελεσματική λειτουργία του συστήματος, διαχωρίζοντας τη λογική της εφαρμογής από τις λειτουργίες της διεπαφής, προσφέροντας ένα σημείο ελέγχου για την αλληλεπίδραση με τον χρήστη.

Πρόβλημα: Κατά την ανάπτυξη ενός λογισμικού, πολλές φορές αντιμετωπίζουμε την ανάγκη να διαχειριστούμε τη ροή των δεδομένων και των εντολών που έρχονται από τον χρήστη ή το σύστημα. Το πρόβλημα αφορά τον τρόπο με τον οποίο αντιμετωπίζουμε αυτές τις εισόδους ώστε να εξυπηρετήσουμε τις ανάγκες του συστήματος χωρίς να προκύψουν πολύπλοκες ή μη διαχειρίσιμες δομές.

Λύση (Πρότυπο "Ελεγκτής"): Το πρότυπο "Ελεγκτής" παρέχει λύση σε αυτό το πρόβλημα με τον καθορισμό ενός κεντρικού στοιχείου, γνωστού ως ελεγκτής (Controller), που αναλαμβάνει τη διαχείριση των εισερχόμενων αιτημάτων και την κατεύθυνσή τους στα αντίστοιχα στοιχεία του συστήματος για εκτέλεση.

Ένα παράδειγμα μπορεί να είναι ένα σύστημα διαχείρισης χρηστών. Έστω ότι έχουμε μια κλάση "Κλάση Χρήστη" που περιέχει τα δεδομένα ενός χρήστη όπως όνομα, ηλεκτρονική διεύθυνση, κωδικός πρόσβασης κ.λπ. Η κλάση "Κλάση Ελεγκτή" θα μπορούσε να είναι υπεύθυνη για τον έλεγχο των ενεργειών που σχετίζονται με τη διαχείριση χρηστών, όπως τη δημιουργία, την ενημέρωση, και τη διαγραφή χρηστών. Ο έλεγχος της ροής εργασίας (π.χ., εάν ένας χρήστης έχει το σωστό δικαίωμα να πραγματοποιήσει μια ενέργεια) είναι υποκείμενο στην κλάση ελεγκτή, ενώ οι λεπτομέρειες του χρήστη παραμένουν στην κλάση χρήστη.

Μέσω αυτού του μοντέλου, η διαχείριση των αιτημάτων γίνεται πιο αποτελεσματική, επιτρέποντας την ευκολότερη διαχείριση του κώδικα και την ανάπτυξη δομημένων λύσεων που εξυπηρετούν τις ανάγκες του συστήματος.

5.3.5 Υψηλή Συνοχή (High Cohesion)

Το πρότυπο "Υψηλή Συνοχή" (High Cohesion) αποτελεί ένα από τα βασικά στοιχεία σχεδιασμού λογισμικού και ανήκει στη συλλογή των GRASP (General Responsibility Assignment Software Patterns). Στο πλαίσιο της αρχιτεκτονικής λογισμικού, η υψηλή συνοχή αφορά τη συγκέντρωση λειτουργιών ή ευθυνών σε ένα μέρος του κώδικα, ενθαρρύνοντας την οργάνωση των συστατικών (κλάσεις, μέθοδοι, αντικείμενα κτλ) του συστήματος.



Οι κύριοι στόχοι του προτύπου "Υψηλή Συνοχή" είναι:

- **Συγκέντρωση και Ομαδοποίηση Συναφών Λειτουργιών:** Η υψηλή συνοχή αναζητά τη συγκέντρωση συναφών λειτουργιών ή καθηκόντων σε μια μονάδα κώδικα. Αυτό ενισχύει τη συνοχή και τη σαφήνεια του κώδικα, καθώς συναφείς λειτουργίες ή ευθύνες θα βρίσκονται κοντά η μία στην άλλη.
- **Ενίσχυση της Ανάγνωσης και Κατανόησης:** Η ομαδοποίηση συναφών λειτουργιών σε ένα σημείο του κώδικα κάνει τον κώδικα πιο ευανάγνωστο και κατανοητό, καθώς συναφείς ενέργειες βρίσκονται μαζί, βοηθώντας στην εύκολη παρακολούθηση του κώδικα.
- **Ευκολία Συντήρησης και Επέκτασης:** Ένα σύστημα με υψηλή συνοχή είναι πιο εύκολο να συντηρηθεί και να επεκταθεί. Η συγκέντρωση συναφών λειτουργιών καθιστά πιο εύκολη την ενημέρωση ή την προσθήκη νέων λειτουργιών χωρίς να επηρεάζονται περισσότερα τμήματα του συστήματος.

Συνολικά, η υψηλή συνοχή βοηθά στη βελτίωση της δομής και της οργάνωσης του λογισμικού, επιτρέποντας την ευκολότερη κατανόηση, συντήρηση και επέκταση του κώδικα.

Πρόβλημα: Κατά τη διαδικασία σχεδιασμού λογισμικού, συχνά αντιμετωπίζουμε το πρόβλημα της οργάνωσης των διάφορων μεθόδων ή λειτουργιών σε κλάσεις και αντικείμενα με τρόπο που εξασφαλίζει μια συνεκτική και αποτελεσματική δομή.

Λύση (Πρότυπο "Υψηλή Συνοχή"): Το πρότυπο "Υψηλή Συνοχή" αποσκοπεί στη δημιουργία κλάσεων ή αντικειμένων όπου οι μέθοδοι τους συνδέονται με έναν τρόπο που εξυπηρετεί μια κοινή λειτουργικότητα ή στόχο. Βασικά, η "υψηλή συνοχή" δείχνει το πόσο στενά σχετίζονται τα διάφορα στοιχεία εντός μιας κλάσης ή ενός αντικειμένου.

Ένα παράδειγμα μπορεί να είναι μια κλάση "Διαχείριση Παραγγελιών" σε ένα σύστημα ηλεκτρονικού εμπορίου. Αυτή η κλάση μπορεί να περιλαμβάνει μεθόδους για τη δημιουργία, επεξεργασία, ακύρωση και παρακολούθηση παραγγελιών. Οι ευθύνες της κλάσης είναι συγκεντρωμένες γύρω από τη διαχείριση των παραγγελιών, παρέχοντας υψηλή συνοχή. Αν η κλάση ξεκίναγε να αναλαμβάνει και άλλες ευθύνες που δεν σχετίζονται με τη διαχείριση παραγγελιών, όπως τη διαχείριση χρηστών ή τον υπολογισμό τιμών προϊόντων, θα χαμήλωνε το επίπεδο συνοχής. Συνεπώς, η διατήρηση υψηλής συνοχής στην υλοποίηση των ευθυνών μιας κλάσης είναι σημαντική για την ευανάγνωση, τη συντηρησιμότητα και την επαναχρησιμοποίηση του κώδικα.

Με την εφαρμογή αυτού του προτύπου, επιτυγχάνεται ένας καλύτερα οργανωμένος κώδικας, με μια δομή που εξυπηρετεί μια συγκεκριμένη λειτουργικότητα. Αυτό βελτιώνει την κατανόηση του κώδικα και τη διαχείριση της συντήρησης, καθώς οι μέθοδοι παραμένουν συγκροτημένες γύρω από ένα κοινό θέμα ή λειτουργία.

5.3.6 Πολυμορφισμός (Polymorphism)

Ο πολυμορφισμός είναι ένα βασικό έννοια στην αντικειμενοστραφή προγραμματισμό, και αναφέρεται στη δυνατότητα ενός αντικειμένου να παρουσιάζει διαφορετική συμπεριφορά με βάση τον τύπο ή τη θέση από την οποία καλείται. Αυτό επιτρέπει σε



διαφορετικά αντικείμενα να υλοποιούν μια κοινή διεπαφή ή μια κοινή συμπεριφορά, αλλά με διαφορετικούς τρόπους.

Ο πολυμορφισμός έχει τρεις κύριες μορφές:

- **Υπερφόρτωση Μεθόδων (Method Overloading):** Αφορά τη δυνατότητα μιας κλάσης να έχει πολλαπλές μεθόδους με το ίδιο όνομα, αλλά διαφορετικές παραμέτρους.
- **Υποκατάσταση (Subtyping):** Αναφέρεται στη δυνατότητα μιας υποκλάσης να αντικαταστήσει μια μητρική κλάση και να χρησιμοποιήσει μια μέθοδο που έχει ήδη οριστεί στη μητρική κλάση, παρέχοντας τη δική της υλοποίηση.
- **Ανώτερη Υποκλάση (Supertype Polymorphism):** Αφορά την αδυναμία του κώδικα να διακρίνει τον πραγματικό τύπο του αντικειμένου, επιτρέποντας σε διαφορετικά αντικείμενα να χρησιμοποιούν μια κοινή διεπαφή και να αντιδρούν διαφορετικά στις ίδιες κλήσεις μεθόδων.

Ο πολυμορφισμός βοηθά στην αποφυγή επαναλαμβανόμενου κώδικα και στην οργάνωση του λογισμικού, επιτρέποντας την ανακάλυψη κοινών συμπεριφορών σε διαφορετικά αντικείμενα και κλάσεις.

Πρόβλημα: Κατά τη διαδικασία ανάπτυξης λογισμικού, συχνά αντιμετωπίζουμε το πρόβλημα της ανάγκης για εύκολη διαχείριση διαφορετικών τύπων αντικειμένων ή συμπεριφορών, ιδανικά χωρίς την ανάγκη για υπερβολικά πολλαπλές δομές ελέγχου.

Λύση (Πρότυπο "Πολυμορφισμός"): Ο πολυμορφισμός αναφέρεται στη δυνατότητα ενός αντικειμένου να παρουσιάζει διαφορετικές συμπεριφορές ανάλογα με τον τύπο των δεδομένων ή το πλαίσιο όπου χρησιμοποιείται. Στο πλαίσιο της αντικειμενοστραφούς προγραμματισμού, ο πολυμορφισμός επιτρέπει σε αντικείμενα με κοινές διαδικασίες ή στοιχεία να προσφέρουν διαφορετική συμπεριφορά, ανάλογα με την πραγματική υλοποίησή τους.

Ένα παράδειγμα μπορεί να είναι ένα σύστημα αποθήκευσης αρχείων, όπου έχουμε μια κλάση "Αρχείο" και υποκλάσεις όπως "Κείμενο Αρχείου" και "Εικόνα". Κάθε υποκλάση υλοποιεί μια μέθοδο "άνοιγμα()" που αντιπροσωπεύει τον τρόπο με τον οποίο ανοίγει το συγκεκριμένο είδος αρχείου.

```
class Αρχείο:
```

```
    def ανοιγμα(self):  
        pass
```

```
class ΚείμενοΑρχείου(Αρχείο):
```

```
    def άνοιγμα(self):  
        print("Ανοίγει το κείμενο του αρχείου")
```

```
class Εικόνα(Αρχείο):
```

```
    def άνοιγμα(self):
```



```
print("Ανοίγει η εικόνα")
```

Εδώ, ο πολυμορφισμός επιτρέπει στις υποκλάσεις “Κείμενο Αρχείου” και “Εικόνα” να υλοποιήσουν τη μέθοδο "άνοιγμα()" κατά τρόπο που είναι συγκεκριμένος για το καθένα από αυτά. Αυτό επιτρέπει την ευελιξία χρήσης της ίδιας διεπαφής για διαφορετικά είδη αρχείων.

Καθώς εφαρμόζετε ο πολυμορφισμός στον σχεδιασμό λογισμικού, επωφελείστε από την ευελιξία που προσφέρει στον τρόπο που τα αντικείμενα αντιδρούν και αλληλεπιδρούν με τον κώδικα σας.

5.3.7 Καθαρή Επινόηση (Pure Fabrication)

Το πρότυπο "Καθαρή Επινόηση" (Pure Fabrication) ανήκει στα μοτίβα σχεδιασμού λογισμικού της συλλογής GRASP (General Responsibility Assignment Software Patterns). Στην ουσία, αναφέρεται στη δημιουργία ενός τεχνητού αντικειμένου ή ενός δομικού στοιχείου στην αρχιτεκτονική ενός συστήματος που δεν αντιστοιχεί άμεσα σε έναν υπαρκτό μηχανισμό ή δεν ανήκει στο πραγματικό περιβάλλον της εφαρμογής.

Οι βασικοί στόχοι του προτύπου "Καθαρής Επινόησης" είναι:

- **Αποφυγή Υπερφόρτωσης Κλάσεων:** Αντί να υπερφορτώνουμε υπάρχουσες κλάσεις με ευθύνες, δημιουργούμε νέα, τεχνητά στοιχεία ή κλάσεις για να αναλάβουν ορισμένες λειτουργίες, αποφορτίζοντας έτσι τις υπάρχουσες κλάσεις.
- **Βελτιστοποίηση της Δομής του Συστήματος:** Η καθαρή επινόηση βοηθά στη διατήρηση της καθαρότητας και της οργάνωσης του κώδικα με τη δημιουργία νέων κλάσεων ή αντικειμένων που έχουν συγκεκριμένες λειτουργίες ή ευθύνες.
- **Μείωση της Πολυπλοκότητας:** Η δημιουργία καθαρών επινοήσεων μπορεί να μειώσει την πολυπλοκότητα του συστήματος, καθώς ορισμένες λειτουργίες μπορούν να ανατεθούν σε νέα, εικονικά ή τεχνητά στοιχεία.

Ο στόχος της καθαρής επινόησης είναι να βελτιώσει την οργάνωση και τη διαχείριση του κώδικα, δημιουργώντας νέες κλάσεις ή αντικείμενα που αναλαμβάνουν συγκεκριμένες λειτουργίες, βελτιώνοντας έτσι την αναγνωσιμότητα, τη συντήρηση και την επεκτασιμότητα του κώδικα.

Πρόβλημα: Ένα πρόβλημα στο οποίο μπορεί να εφαρμοστεί το πρότυπο "Καθαρή Επινόηση" είναι όταν υπάρχουν λειτουργίες ή ευθύνες που δεν ανήκουν αυστηρά σε κάποια υπάρχουσα κλάση ή αντικείμενο. Το πρόβλημα είναι η ανάγκη δημιουργίας μιας νέας δομής που να μην προστίθεται στην υπάρχουσα ιεραρχία ή δομή του λογισμικού, αλλά να αναλαμβάνει αυτές τις λειτουργίες.

Λύση (Πρότυπο "Καθαρή Επινόηση"): Η λύση που προτείνει το πρότυπο "Καθαρή Επινόηση" είναι η δημιουργία μιας νέας κλάσης ή ενός αντικειμένου για τη διαχείριση αυτών των ευθυνών που δε σχετίζονται άμεσα με τις υπάρχουσες κλάσεις. Αυτή η νέα δομή επιτρέπει την αποδέσμευση λειτουργιών που δεν ανήκουν στις ήδη υπάρχουσες κλάσεις, βελτιώνοντας τη συνολική οργάνωση του κώδικα και αποφεύγοντας την υπερφόρτωση με πολλές ευθύνες σε μια κλάση. Αντί να επιβαρύνουμε μία υπάρχουσα



κλάση με ακόμα περισσότερες ευθύνες, δημιουργούμε μια νέα κλάση που αναλαμβάνει αυτές τις λειτουργίες. Με αυτόν τον τρόπο, καταφέρνουμε να διατηρήσουμε τη σαφήνεια στον κώδικα, αποφεύγοντας την υπερφόρτωση των υπεύθυνων κλάσεων και βελτιώνοντας τη συνολική δομή και συντηρησιμότητα του λογισμικού.

Ένα παράδειγμα Καθαρής Επινόησης μπορεί να είναι η χρήση μιας εικονικής κλάσης για τη διαχείριση ρυθμίσεων. Ας υποθέσουμε ότι έχουμε ένα σύστημα όπου χρειαζόμαστε να αποθηκεύουμε και να διαχειριζόμαστε διάφορες ρυθμίσεις της εφαρμογής. Μπορούμε να δημιουργήσουμε μια κλάση "Διαχείριση Ρυθμίσεων" που θα παρέχει μεθόδους για την ανάκτηση και την ενημέρωση των ρυθμίσεων. Αυτή η κλάση θα είναι ένα είδος "καθαρής επινόησης", καθώς δημιουργείται για να εξυπηρετήσει ένα συγκεκριμένο σκοπό (τη διαχείριση ρυθμίσεων) χωρίς να αντιστοιχεί σε πραγματικό αντικείμενο του προβλήματος.

```
class ΔιαχείρισηΡυθμίσεων:
```

```
    def __init__(self):
```

```
        self.ρυθμίσεις = {}
```

```
    def πάρε_ρύθμιση(self, κλειδί):
```

```
        return self.ρυθμίσεις.get(κλειδί, None)
```

```
    def ορίστε_ρύθμιση(self, κλειδί, τιμή):
```

```
        self.ρυθμίσεις[κλειδί] = τιμή
```

```
# Παράδειγμα χρήσης
```

```
διαχείριση_ρυθμίσεων = ΔιαχείρισηΡυθμίσεων()
```

```
διαχείριση_ρυθμίσεων.ορίστε_ρύθμιση("γλώσσα", "Ελληνικά")
```

```
γλώσσα = διαχείριση_ρυθμίσεων.πάρε_ρύθμιση("γλώσσα")
```

```
print(γλώσσα) # Εκτύπωση: Ελληνικά
```

Σε αυτό το παράδειγμα, η κλάση "Διαχείριση Ρυθμίσεων" δημιουργείται με σκοπό τη διαχείριση ρυθμίσεων, αλλά δεν αντιστοιχεί σε κάτι φυσικό ή πραγματικό στον πραγματικό κόσμο.

Προσφέρει μια λύση όταν οι ευθύνες δεν προσδιορίζονται φυσικά στο πλαίσιο των υπάρχων κλάσεων, αλλά πρέπει να διαχειριστούν ως ένα ξεχωριστό μονάδα λειτουργιών.

5.3.8 Πλαγιότητα (Indirection)

Η "Πλαγιότητα" (Indirection) είναι ένα από τα βασικά μοτίβα σχεδιασμού λογισμικού που ανήκουν στη συλλογή των GRASP (General Responsibility Assignment Software Patterns). Αφορά τη χρήση ενός ενδιάμεσου επιπέδου, ενός είδους ενδομεταφοράς, για



την αντιμετώπιση ενός προβλήματος ή την επίλυση μιας ανάγκης, αντί να γίνεται απευθείας από τον προορισμό που την αφορά.

Οι βασικοί στόχοι της πλαγιότητας είναι:

- **Δημιουργία Ενδιάμεσων Επιπέδων:** Αντί να έχουμε άμεση αλληλεπίδραση μεταξύ διαφορετικών στοιχείων του λογισμικού, χρησιμοποιούμε ένα ενδιάμεσο επίπεδο ή δομή, παρέχοντας μια πιο ευέλικτη, ελαφριά και αποδοτική λύση.
- **Αποσύνδεση Κλάσεων ή Αντικειμένων:** Η πλαγιότητα βοηθά στην αποσύνδεση ή απομάκρυνση των στοιχείων του λογισμικού, επιτρέποντας την αλληλεπίδραση μέσω ενός ενδιάμεσου, αντί να συμβαίνει απευθείας μεταξύ τους.
- **Επίλυση Πολυπλοκότητας:** Η πλαγιότητα βοηθά στη μείωση της πολυπλοκότητας του λογισμικού, καθώς επιτρέπει τη χρήση μιας ενδιάμεσης δομής για την επίλυση προβλημάτων ή την παροχή υπηρεσιών, αποσυνδεδεμένη από τις λεπτομέρειες της υλοποίησης.

Οι αρχές της πλαγιότητας επιτρέπουν μεγαλύτερη ευελιξία, καθαρότητα και διαχείριση σε ένα σύστημα λογισμικού, μειώνοντας την εξάρτηση και την πολυπλοκότητα μεταξύ των στοιχείων του συστήματος.

Πρόβλημα: Συχνά, στον σχεδιασμό του λογισμικού, υπάρχει η τάση να έχουμε άμεσες συσχετίσεις μεταξύ διάφορων τμημάτων του κώδικα. Αυτό μπορεί να οδηγήσει σε περίπλοκο, δύσκολο στη συντήρηση και ευάλωτο κώδικα, καθώς αλλαγές σε ένα μέρος του κώδικα μπορεί να έχουν ανεπιθύμητες επιπτώσεις σε άλλα μέρη.

Λύση (Πρότυπο "Πλαγιότητα"): Η λύση που προτείνει το πρότυπο "Πλαγιότητα" είναι να εισαγάγουμε ένα επίπεδο ενδιάμεσης ανακατεύθυνσης μεταξύ των συσχετιζόμενων τμημάτων του κώδικα. Αυτή η ενδιάμεση ανακατεύθυνση, γνωστή ως "πλαγιότητα," δημιουργεί ένα επίπεδο αφαιρετικότητας που διαχωρίζει τα διάφορα τμήματα του κώδικα από το ένα άλλο. Αυτή η διάκριση επιτρέπει την ευελιξία και την αντικαταστασιμότητα τμημάτων του κώδικα χωρίς να απαιτείται συχνή αλλαγή σε άλλα τμήματα. Ο κώδικας γίνεται πιο ευανάγνωστος και συντηρήσιμος, καθώς οι αλλαγές εφαρμόζονται σε πιο περιορισμένες περιοχές.

Ένα παράδειγμα μπορεί να είναι ένας διαχειριστής αρχείων, όπου χρησιμοποιούμε έναν ενδιάμεσο διαχειριστή για να αποκρύψουμε τις λεπτομέρειες της φυσικής αποθήκευσης από το υπόλοιπο σύστημα.

```
class ΔιαχειριστήςΑρχείων:
```

```
    def __init__(self, φυσική_αποθήκευση):
```

```
        self.φυσική_αποθήκευση = φυσική_αποθήκευση
```

```
    def ανοιγμα_αρχείου(self, όνομα_αρχείου):
```

```
        return self.φυσική_αποθήκευση.ανοιγμα(όνομα_αρχείου)
```

```
    def ανάγνωση_αρχείου(self, αρχείο):
```



```
return self.φυσική_αποθήκευση.ανάγνωση(αρχείο)

def κλείσιμο_αρχείου(self, αρχείο):
    return self.φυσική_αποθήκευση.κλείσιμο(αρχείο)

class ΦυσικήΑποθήκευση:
    def ανοιγμα(self, όνομα_αρχείου):
        # Λογική για το άνοιγμα του αρχείου

    def ανάγνωση(self, αρχείο):
        # Λογική για την ανάγνωση του αρχείου

    def κλείσιμο(self, αρχείο):
        # Λογική για το κλείσιμο του αρχείου
```

Σε αυτό το παράδειγμα, ο διαχειριστής αρχείων (Διαχειριστής Αρχείων) χρησιμοποιεί έναν ενδιάμεσο διαχειριστή (Φυσική Αποθήκευση) για τις φυσικές ενέργειες όπως το άνοιγμα, η ανάγνωση και το κλείσιμο των αρχείων. Αυτή η προσέγγιση επιτρέπει την αντικατάσταση ή την επέκταση της φυσικής αποθήκευσης χωρίς να επηρεαστεί ο διαχειριστής αρχείων.

Συνοψίζοντας, το πρότυπο "Πλαγιότητα" αντιμετωπίζει το πρόβλημα της υψηλής συσχέτισης μεταξύ κομματιών του κώδικα προσθέτοντας ένα επίπεδο ενδιάμεσης ανακατεύθυνσης για να διαχωρίσει τα διάφορα τμήματα και να βελτιώσει τη συντηρησιμότητα και την αποδοτικότητα του λογισμικού.

5.3.9 Προστατευμένες Παραλλαγές (Protected Variations)

Το πρότυπο "Προστατευμένες Παραλλαγές" (Protected Variations) αποτελεί ένα από τα βασικά μοτίβα σχεδιασμού που περιλαμβάνονται στη συλλογή GRASP (General Responsibility Assignment Software Patterns). Αυτό το πρότυπο εστιάζει στην ανάπτυξη συστημάτων που είναι ευέλικτα και προστατεύουν τα εσωτερικά στοιχεία τους από εξωτερικές αλλαγές. Κεντρικό σημείο είναι η δημιουργία διεπαφών μεταξύ στοιχείων του συστήματος, επιτρέποντας την αντικατάσταση ή την επέκταση των συστατικών με άλλα χωρίς να επηρεαστεί η λειτουργία του συστήματος.

Οι βασικοί στόχοι του προτύπου "Προστατευμένες Παραλλαγές" είναι:

- **Διαχωρισμός των Μερών του Συστήματος:** Χρησιμοποιεί διεπαφές για την απομόνωση των συστατικών μερών ενός συστήματος, επιτρέποντας την αντικατάσταση με άλλα στοιχεία που ακολουθούν την ίδια δομή.
- **Ελαχιστοποίηση των Επιπτώσεων Αλλαγών:** Δομή που επιτρέπει αλλαγές χωρίς να επηρεάζεται η συνολική λειτουργικότητα του συστήματος.
- **Προστασία από Εξωτερικές Αλλαγές:** Προστατεύει το σύνολο του συστήματος από εξωτερικές αλλαγές, επιτρέποντας την αντικατάσταση ενός στοιχείου με άλλο που προσφέρει την ίδια διεπαφή.



Αυτό το πρότυπο επιτρέπει την ανάπτυξη συστημάτων που είναι ανθεκτικά στις αλλαγές, παρέχοντας δομές που επιτρέπουν την εύκολη αντικατάσταση ή επέκταση στοιχείων χωρίς να δημιουργείται δυσλειτουργία στο σύστημα.

Πρόβλημα: Κατά την ανάπτυξη λογισμικού, είναι συχνό φαινόμενο να υπάρχουν εξαρτήσεις από εξωτερικά συστήματα, τρίτους παρόχους, ή διεπαφές. Αλλαγές σε αυτούς τους εξωτερικούς παράγοντες μπορεί να επηρεάσουν το λογισμικό και να δημιουργήσουν προβλήματα και αστάθεια. Επιπλέον, η ανάπτυξη λογισμικού μπορεί να απαιτεί την προσαρμογή σε διάφορες πλατφόρμες ή συσκευές, κάτι που μπορεί να καταστήσει το λογισμικό ευάλωτο σε αλλαγές στο περιβάλλον.

Λύση (Πρότυπο "Προστατευμένες Παραλλαγές"): Το πρότυπο "Προστατευμένες Παραλλαγές" προτείνει τη χρήση ενός ενδιάμεσου επιπέδου ή προστατευτικών επιπέδων μεταξύ του λογισμικού και των εξωτερικών παραγόντων ή διεπαφών. Αυτό το επίπεδο δρα ως "αποθήκη" που προστατεύει το υπόλοιπο λογισμικό από τις αλλαγές στους εξωτερικούς παράγοντες. Ανάλογα με τις συγκεκριμένες ανάγκες, αυτό το επίπεδο μπορεί να παρέχει διάφορες λειτουργίες, όπως μετατροπή δεδομένων, διαχείριση συσκευών, επαναδρομολόγηση, και άλλες. Η ιδέα είναι να μειώσετε τις εξαρτήσεις του βασικού λογισμικού από τους εξωτερικούς παράγοντες, κάνοντας το λογισμικό πιο ανεξάρτητο από τις αλλαγές σε αυτούς τους παράγοντες.

Ένα παράδειγμα μπορεί να είναι ένα σύστημα καταγραφής δεδομένων αισθητήρων. Έστω ότι έχουμε έναν αισθητήρα θερμοκρασίας και έναν αισθητήρα υγρασίας. Ο προστατευόμενος κώδικας, ο οποίος καταγράφει τις μετρήσεις των αισθητήρων, θα μπορούσε να έχει την ακόλουθη δομή:

```
# Προστατευόμενος κώδικας
```

```
class Αισθητήρας:
```

```
    def μέτρηση(self):
```

```
        raise NotImplementedError("Υλοποιήστε αυτήν τη μέθοδο στις υποκείμενες κλάσεις")
```

```
class ΘερμοκρασίαςΑισθητήρας(Αισθητήρας):
```

```
    def μέτρηση(self):
```

```
        # Υλοποίηση για τη μέτρηση θερμοκρασίας
```

```
class ΥγρασίαςΑισθητήρας(Αισθητήρας):
```

```
    def μέτρηση(self):
```

```
        # Υλοποίηση για τη μέτρηση υγρασίας
```

```
class Καταγραφέας:
```

```
    def καταγραφή_μέτρησης(self, αισθητήρας):
```

```
        μέτρηση = αισθητήρας.μέτρηση()
```

```
        # Λογική για την καταγραφή της μέτρησης
```



Σε αυτό το παράδειγμα, ο προστατευόμενος κώδικας χρησιμοποιεί την αφαιρετική κλάση "Αισθητήρας" για να αποφύγει την άμεση εξάρτηση από τους αισθητήρες. Αυτό επιτρέπει την αντικατάσταση των αισθητήρων χωρίς να χρειάζεται να τροποποιηθεί ο προστατευόμενος κώδικας.

Με τον τρόπο αυτό, το πρότυπο "Προστατευμένες Παραλλαγές" βοηθά στη δημιουργία ενός πιο ευέλικτου και εύλατου λογισμικού που μπορεί να ανταποκριθεί σε αλλαγές στο περιβάλλον ή στους εξωτερικούς παράγοντες χωρίς να προκαλεί σοβαρά προβλήματα στο λογισμικό.

5.4 Τα πρότυπα GRASP στην εφαρμογή μας (Habit Tracker app)

Τα GRASP (General Responsibility Assignment Software Patterns) είναι ένα σύνολο προτύπων σχεδιασμού λογισμικού που προσφέρουν κατευθυντήριες αρχές για τον καλύτερο δυνατό σχεδιασμό αντικειμένων και τον καλύτερο δυνατό καθορισμό των υπευθυνότητων τους. Στην εφαρμογή μας Habit Tracker, έχουν τηρηθεί αρκετά από αυτά τα πρότυπα προκειμένου να επιτευχθεί ένας σχεδιασμός που είναι ευανάγνωστος, επεκτάσιμος και συντηρήσιμος.

Δημιουργός (Creator): Στο front-end η κλάση RetrofitClient λειτουργεί ως Δημιουργός (Creator) στο πλαίσιο του GRASP. Αναλαμβάνει τον ρόλο της δημιουργίας του αντικείμενου Retrofit, παρέχοντας μια μέθοδο (getClient()) που δημιουργεί και επιστρέφει ένα αντικείμενο της κλάσης Retrofit. Η χρήση ενός δημιουργού βοηθάει στην αποφυγή σκληρής σύνδεσης (hard coupling) με τις λεπτομέρειες υλοποίησης του Retrofit και παρέχει ένα ενδιάμεσο επίπεδο που μπορεί να αλλάξει χωρίς να επηρεάσει την υπόλοιπη εφαρμογή.

- Η κλάση RetrofitClient λειτουργεί ως δημιουργός, καθώς δημιουργεί ένα αντικείμενο Retrofit και παρέχει μια μέθοδο getClient() για τη δημιουργία του.
- Η κλάση RetrofitClient είναι υπεύθυνη για τη δημιουργία αντικειμένων της κλάσης Retrofit.
- Παρέχει τη μέθοδο getClient(), η οποία δημιουργεί ένα αντικείμενο Retrofit και το επιστρέφει.

Αυτό βοηθάει στην αποφυγή σκληρής σύνδεσης με τις λεπτομέρειες υλοποίησης του Retrofit, καθώς οποιεσδήποτε αλλαγές στην υλοποίηση του Retrofit μπορούν να γίνουν μόνο σε αυτήν την κλάση, χωρίς να επηρεάζουν άλλα μέρη της εφαρμογής.

Ακολουθώντας στο back-end της εφαρμογής το πρότυπο "Δημιουργός" χρησιμοποιείται κατά την δημιουργία των αντικειμένων από άλλα αντικείμενα και συγκεκριμένα ή μέθοδος create του JsonSerializer που είναι υπεύθυνη για να δημιουργεί τα αντικείμενα των Users για να εγγυάται την επαναχρησιμοποίηση του κώδικα που βελτιώνει την συντηρησιμότητά. Παρακάτω στην εικόνα εμφανίζεται αυτό το πρότυπο.

```
class UserSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True)

    def create(self, validated_data):
        user = User.objects.create(
            username=validated_data['username'],
        )
        user.set_password(validated_data['password'])
        user.save()
        return user
```

Εικόνα 16 Δημιουργός (Creator) Back-end

Πληροφορημένος Ειδικός (Information Expert): Η κλάση `UserAuthActivity` που υλοποιήθηκε για την ανάπτυξη της android εφαρμογής είναι πιθανό να είναι ο πληροφορημένος ειδικός για τον έλεγχο του συστήματος σύνδεσης χρήστη. Ας αναλύσουμε τη χρήση του προτύπου "Πληροφορημένος Ειδικός (Information Expert)" στην κλάση `UserAuthActivity`. Αυτό το πρότυπο λέει ότι μια κλάση πρέπει να είναι υπεύθυνη για τον εκάστοτε καθήκον ή λειτουργία που αφορά τα δεδομένα. Δηλαδή, πρέπει να είναι η κλάση που έχει τις απαραίτητες πληροφορίες για να εκτελέσει ένα συγκεκριμένο καθήκον. Η `UserAuthActivity` είναι υπεύθυνη για το σύστημα σύνδεσης χρήστη. Έχει πρόσβαση στα στοιχεία εισόδου του χρήστη (όπως email και password), ελέγχει την εγκυρότητά τους και αλληλεπιδρά με το API μέσω του Retrofit για τον έλεγχο ταυτότητας.

Συγκεκριμένα Παραδείγματα:

1. Έλεγχος Στοιχείων Εισόδου: Η `UserAuthActivity` ελέγχει αν τα στοιχεία εισόδου (email, password) είναι έγκυρα και εμφανίζει ανάλογα μηνύματα σφάλματος αν δεν είναι.
2. Κλήση του API για Έλεγχο Ταυτότητας: Με τη χρήση του Retrofit, η κλάση καλεί το API για να ελέγξει τα διαπιστευτήρια του χρήστη και λαμβάνει την απάντηση.
3. Ενημέρωση UI: Αφού προελέγξει και αλληλεπιδρά με το API, ενημερώνει το UI με ανάλογα μηνύματα, είτε για επιτυχημένη σύνδεση είτε για σφάλματα.

Η κλάση `UserAuthActivity` λοιπόν, φαίνεται να είναι ο πληροφορημένος ειδικός για τον έλεγχο του συστήματος σύνδεσης χρήστη. Είναι υπεύθυνη για την εξέταση των στοιχείων εισόδου του χρήστη, τον έλεγχο της εγκυρότητας τους, καθώς και για την αλληλεπίδραση με το API για τον έλεγχο της ταυτότητας του χρήστη. Ο ρόλος της κλάσης συνάδει με την έννοια του "Πληροφορημένου Ειδικού", όπου η κλάση που έχει τις απαραίτητες πληροφορίες και είναι εξειδικευμένη στον τομέα του εκάστοτε καθήκοντος αναλαμβάνει τον έλεγχο. Έτσι, η `UserAuthActivity` λειτουργεί ως



πληροφορημένος ειδικός καθώς διαχειρίζεται τα δεδομένα και τις λειτουργίες που σχετίζονται με το σύστημα σύνδεσης χρήστη.

Στο “πίσω” μέρος της εφαρμογής το πρότυπο "Πληροφορημένος Ειδικός" χρησιμοποιείται σε όλα τα μοντέλα που χρησιμοποιούνται για την αποθήκευση της πληροφορίας όπως το **Habit** και το **Category**, καθώς περιέχουν και διαχειρίζονται πληροφορίες σχετικές με τους τομείς τους. Χαρακτηριστικά η εικόνα 14 περιέχει τον κώδικα του μοντέλου

```
class Habit(models.Model):
    name = models.CharField(max_length=250)
    description = models.CharField(max_length=500, blank=True)
    reason = models.CharField(max_length=400, blank=True)
    days_of_the_week = models.JSONField(default=default_days)
    start_date = models.DateField(default=datetime.date.today)
    end_date = models.DateField(null=True, blank=True)
    is_active = models.BooleanField(default=True)
    is_complete = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    category = models.ForeignKey(Category, on_delete=models.SET_NULL, related_name='category_habits', null=True,
                                blank=True)
    preferred_time = models.TimeField(null=True, blank=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='user_habits')
```

Εικόνα 17 Πληροφορημένος Ειδικός (Information Expert) Back-end

Χαμηλή Σύζευξη (Low Coupling): Οι κλάσεις UserAuthActivity και ApiService φαίνεται να έχουν χαμηλή σύζευξη, καθώς η UserAuthActivity χρησιμοποιεί το ApiService για τις ανακοινώσεις και δεν είναι άμεσα συνδεδεμένη με λεπτομέρειες υλοποίησης. Το πρότυπο "Χαμηλή Σύζευξη (Low Coupling)" εφαρμόζεται μεταξύ των κλάσεων UserAuthActivity και ApiService. Αναφέρεται στον βαθμό στον οποίο δύο κλάσεις είναι ανεξάρτητες μεταξύ τους. Έχουν χαμηλή σύζευξη όταν μια αλλαγή σε μία κλάση δεν επηρεάζει αναγκαστικά την άλλη.

Πώς εφαρμόζεται στην UserAuthActivity και ApiService:

- **UserAuthActivity:** Δεν γνωρίζει τις λεπτομέρειες υλοποίησης των κλήσεων του API. Χρησιμοποιεί το ApiService ως έναν "σέρβις" για την επικοινωνία με το API χωρίς να γνωρίζει πώς ακριβώς υλοποιούνται αυτές οι κλήσεις.
- **ApiService:** Παρέχει τις απαραίτητες μεθόδους για την επικοινωνία με το API, αλλά δεν ξέρει πώς χρησιμοποιούνται αυτές οι μέθοδοι. Δεν γνωρίζει τον τρόπο που η UserAuthActivity χρησιμοποιεί τις μεθόδους αυτές.

Τα οφέλη της Χαμηλής Σύζευξης:

1. **Καλύτερη συντηρησιμότητα:** Οι αλλαγές σε μία κλάση δεν επηρεάζουν απαραίτητα τη λειτουργία της άλλης.



2. **Ευελιξία:** Η αντικατάσταση ή επέκταση μιας κλάσης γίνεται πιο εύκολα χωρίς να επηρεάζονται οι υπόλοιπες.

Συνεπώς, η `UserAuthActivity` μπορεί να επικοινωνεί με το API μέσω του `ApiService` χωρίς να γνωρίζει τις λεπτομέρειες της υλοποίησης, προσφέροντας χαμηλή σύζευξη.

Για την ανάδειξη της αρχή του Low Coupling του back-end της εφαρμογής παρατίθενται παρακάτω τα αντιπροσωπευτικά τμήματα:

1. **Διαχωρισμός Μοντέλων και Views:** Τα μοντέλα (**Category**, **Habit**, **HabitRecord**) και τα views (**Categories**, **CategoryDetails**, **Habits**, **HabitDetails**, **UserCreate**) είναι σχεδιασμένα να λειτουργούν ανεξάρτητα. Τα μοντέλα είναι υπεύθυνα για την αναπαράσταση των δεδομένων και την επιχειρησιακή λογική, ενώ τα views διαχειρίζονται τα εισερχόμενα αιτήματα και τις αποκρίσεις.
2. **Ανεξαρτησία των Serializers:** Οι σειριοποιητές (serializers), όπως ο **HabitSerializer** και ο **CategorySerializer**, είναι σχεδιασμένοι να αντιμετωπίζουν μόνο τη σειριοποίηση και την αποσειριοποίηση δεδομένων, χωρίς να επιβαρύνουν τα μοντέλα ή τα views με πρόσθετες ευθύνες.
3. **Μεθοδολογία API Views:** Κάθε κλάση στο `views.py`, όπως οι **Categories** ή **Habits**, είναι αυτόνομη και επικεντρώνεται σε μια συγκεκριμένη λειτουργική περιοχή όπως διαχείριση κατηγοριών ή συνηθειών. Αυτό συμβάλλει στην χαμηλή σύζευξη, καθώς κάθε κλάση είναι ανεξάρτητη και μπορεί να τροποποιηθεί χωρίς σημαντικές επιπτώσεις σε άλλα τμήματα του συστήματος.
4. **Χρήση Django ORM:** Η χρήση του Django ORM (Object-Relational Mapping) για την αναπαράσταση των μοντέλων δεδομένων ως αντικειμένων Python επιτρέπει τον απομονωμένο χειρισμό των δεδομένων, μειώνοντας την ανάγκη για σύνθετες εξαρτήσεις και διασυνδέσεις.

Αυτές οι πρακτικές και σχεδιαστικές επιλογές που έχουν γίνει κατά την υλοποίηση του κώδικα έχουν συμβάλει στη διατήρηση της αρχιτεκτονικής με χαμηλή σύζευξη, πράγμα που καθιστά το σύστημα πιο ευέλικτο, ευκολότερο στη συντήρηση και πιο ανθεκτικό σε αλλαγές

Ελεγκτής (Controller): Στο front-end η κλάση `UserAuthActivity` λειτουργεί ως ελεγκτής στο πλαίσιο του μοντέλου MVC, διαχειρίζοντας τις αλληλεπιδράσεις του χρήστη και την επικοινωνία με το μοντέλο (`ApiService`, `Habit`, κλπ.). Ας δούμε πώς η κλάση `UserAuthActivity` λειτουργεί ως ελεγκτής (Controller) στο πλαίσιο του μοντέλου MVC (Model-View-Controller). Στο μοντέλο MVC, ο ελεγκτής είναι υπεύθυνος για τον έλεγχο της ροής των δεδομένων και των αλληλεπιδράσεων μεταξύ της Προβολής (View) και του Μοντέλου (Model). Διαχειρίζεται τις ενέργειες του χρήστη και επικοινωνεί με το μοντέλο για την ανάκτηση ή ενημέρωση δεδομένων.

Η `UserAuthActivity` αναλαμβάνει τον έλεγχο των αλληλεπιδράσεων του χρήστη, όπως το κλικ σε κουμπιά, την εισαγωγή στοιχείων, και άλλες δραστηριότητες που αφορούν



την ταυτοποίηση του χρήστη. Χρησιμοποιεί το ApiService για την επικοινωνία με το API για λειτουργίες όπως η σύνδεση χρήστη, η εγγραφή, κλπ. Εκτελεί λειτουργίες που αφορούν το μοντέλο της εφαρμογής, όπως η διαχείριση των συνηθισμένων του χρήστη. Ο ελεγκτής διαχωρίζει τις υποχρεώσεις της εφαρμογής μεταξύ της λογικής χρήστη και της επικοινωνίας με το μοντέλο, επιτρέποντας έτσι την ευελιξία και την ανακατανομή των καθηκόντων. Ο κώδικας που αφορά τη λογική του χρήστη μπορεί να επαναχρησιμοποιηθεί ευκολότερα, καθώς δεν είναι συνδεδεμένος στις λεπτομέρειες της υλοποίησης του μοντέλου.

Συνεπώς, η UserAuthActivity είναι υπεύθυνη για τον έλεγχο της ροής των δεδομένων και των αλληλεπιδράσεων με το μοντέλο, και ως εκ τούτου, λειτουργεί ως ελεγκτής στο πλαίσιο του μοντέλου MVC.

Ακόμα στο back-end τα views όπως τα **Categories**, **CategoryDetails**, **Habits**, **HabitDetails**, και **UserCreate** λειτουργούν ως Controllers. Αυτά τα στοιχεία είναι υπεύθυνα για την επεξεργασία των εισερχόμενων HTTP αιτημάτων, την απόφαση της λογικής που θα εφαρμοστεί και την επιστροφή της κατάλληλης απόκρισης στον χρήστη. Στην παρακάτω εικόνα εμφανίζεται η υλοποίηση της κλάσης Habits που χειρίζεται τις λειτουργίες API για τις συνήθειες του χρήστη. Διαχειρίζεται τα GET αιτήματα για την ανάκτηση όλων των συνηθειών ενός χρήστη και τα POST αιτήματα για τη δημιουργία νέων συνηθειών. Επιβεβαιώνει επίσης ότι μόνο επιβεβαιωμένοι χρήστες έχουν πρόσβαση σε αυτές τις λειτουργίες.

```
class Habits(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        habits = Habit.objects.filter(user=request.user)
        serializer = HabitSerializer(habits, many=True)
        return Response(serializer.data)

    def post(self, request):
        serializer = HabitSerializer(data=request.data)
        if serializer.is_valid():
            try:
                serializer.save(user=request.user)
                return Response(serializer.data, status=status.HTTP_201_CREATED)
            except IntegrityError:
                return Response({
                    "detail": "A habit with this name already exists for this user."
                },
                    status=status.HTTP_400_BAD_REQUEST
                )
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Εικόνα 18 Ελεγκτής (Controller) Back-end



Υψηλή Συνοχή (High Cohesion): Στην υλοποίηση της εφαρμογής του back-end η αρχή της "Υψηλής Συνοχής" (High Cohesion) έπαιξε κεντρικό ρόλο, εξασφαλίζοντας ότι κάθε κλάση και σειριοποιητής (serializer) είναι επικεντρωμένοι σε ειδικές και καθορισμένες λειτουργίες. Αυτό επιτυγχάνεται μέσω της σαφούς διαχωρισμού των ευθυνών μεταξύ των διάφορων στοιχείων του κώδικα. Συγκεκριμένα ο **HabitSerializer** είναι αποκλειστικά αφιερωμένος στην επεξεργασία και τη σειριοποίηση των δεδομένων που αφορούν τις συνήθειες. Με αυτόν τον τρόπο, κάθε κλάση και σειριοποιητής είναι αυτοτελής και εξειδικευμένος σε συγκεκριμένη λειτουργία, βελτιστοποιώντας τη διαχείριση και την συντηρησιμότητα του κώδικα.

Ακόμα στο front-end η κλάση ApiService φαίνεται να έχει υψηλή συνοχή, καθώς περιλαμβάνει μόνο λειτουργίες που σχετίζονται με την αλληλεπίδραση με το API. Ας δούμε πώς η κλάση ApiService πληροί την αρχή της Υψηλής Συνοχής (High Cohesion). Η υψηλή συνοχή αναφέρεται στον βαθμό που οι λειτουργίες ενός module ή μιας κλάσης σχετίζονται λογικά μεταξύ τους. Σε μια κλάση με υψηλή συνοχή, τα μέλη συνδέονται σε μια συγκεκριμένη λογική οντότητα. Η ApiService παρέχει λειτουργίες που σχετίζονται αποκλειστικά με την αλληλεπίδραση με το API. Περιλαμβάνει μεθόδους που αφορούν τον έλεγχο του χρήστη (authenticateUser και registerUser) καθώς και λειτουργίες για τα συνήθη των συνηθισμένων (getUserHabits, createHabit, updateHabit, deleteHabit). Όλες οι μέθοδοι της ApiService έχουν σαφή συσχέτιση με τη διαχείριση δεδομένων χρήστη και την επικοινωνία με τον διακομιστή.

Οφέλη υψηλής συνοχής:

1. **Κατανοητότητα:** Ο κώδικας γίνεται πιο ευανάγνωστος και κατανοητός, καθώς οι συσχετισμένες λειτουργίες βρίσκονται μαζί.
2. **Συντηρησιμότητα:** Η συντήρηση γίνεται πιο εύκολη, καθώς οι αλλαγές που αφορούν την αλληλεπίδραση με το API συγκεντρώνονται σε μια κλάση.
3. **Επαναχρησιμοποίηση:** Η κλάση μπορεί να είναι επαναχρησιμοποιήσιμη, καθώς περιλαμβάνει συνδεδεμένες λειτουργίες.

Επομένως, η ApiService φαίνεται να έχει υψηλή συνοχή, καθώς οι μέθοδοί της σχετίζονται συνολικά με την αλληλεπίδραση με το API.

Πλαγιότητα (Indirection): Η κλάση RetrofitClient παρέχει ένα επίπεδο πλαγιότητας, καθώς παρέχει ένα ενδιάμεσο επίπεδο για τη δημιουργία του Retrofit χωρίς να αποκαλύπτει λεπτομέρειες υλοποίησης. Διανέμει την υπεύθυνη για τη δημιουργία του Retrofit, παρέχοντας ένα καθαρό API μέσω της μεθόδου getClient(). Αυτό επιτρέπει την απομόνωση των λεπτομερειών υλοποίησης και παρέχει ένα επίπεδο πλαγιότητας μεταξύ των κλάσεων.

Στον κώδικά, χρησιμοποιείται για την δημιουργία του Retrofit αντικειμένου, το οποίο χρησιμοποιείται για δικτυακές κλήσεις. Αντί να έχουμε τον κώδικα δημιουργίας του Retrofit απευθείας στην UserAuthActivity, χρησιμοποιείτε το RetrofitClient ως ενδιάμεσο, προσφέροντας έτσι μια πιο καθαρή δομή και επιτρέποντας μεγαλύτερη ευελιξία στο μέλλον για αλλαγές στη διαμόρφωση του Retrofit ή της υπηρεσίας. Αυτό σημαίνει ότι αν, για παράδειγμα, θέλουμε να αλλάξουμε τον τρόπο με τον οποίο ο



Retrofit δημιουργείται ή να προσθέσουμε περισσότερες παραμέτρους ρύθμισης, θα χρειαστεί να κάνουμε αλλαγές μόνο στην RetrofitClient, χωρίς να επηρεάζετε τις υπόλοιπες κλάσεις που χρησιμοποιούν το Retrofit.

Προστατευμένες Παραλλαγές (Protected Variations): Η χρήση του Retrofit ως εξωτερικής βιβλιοθήκης προστατεύει την εσωτερική λειτουργία της υπηρεσίας από τον υπόλοιπο κώδικα της εφαρμογής. Οι λεπτομέρειες υλοποίησης της αλληλεπίδρασης με το δίκτυο, τον χειρισμό των αιτημάτων HTTP και τη διαχείριση των απαντήσεων είναι απομειωμένες και προστατεύονται από τη βιβλιοθήκη. Αυτό επιτρέπει στην εφαρμογή να επωφεληθεί από την λειτουργικότητα του Retrofit χωρίς να χρειάζεται να γνωρίζει λεπτομέρειες υλοποίησης, και επιτρέπει μεγαλύτερη ευελιξία για μελλοντικές αλλαγές στη διαμόρφωση ή αντικατάσταση του Retrofit. Στην περίπτωση της εξωτερικής βιβλιοθήκης, όπως το Retrofit, έχουμε μια περίπτωση προστατευμένης παραλλαγής γιατί:

- Προστατεύει την εσωτερική υλοποίηση: Ο κώδικας της εφαρμογής μας δεν χρειάζεται να ξέρει πώς λειτουργεί το Retrofit ακριβώς. Αυτό το προστατεύει από τις λεπτομέρειες της υλοποίησης του Retrofit.
- Ευελιξία και Αλλαγές: Εάν αποφασίσουμε να αλλάξουμε τη βιβλιοθήκη που χρησιμοποιείται για τις δικτυακές κλήσεις, δεν χρειάζεται να αλλάξουμε τον κώδικα σε πολλά μέρη της εφαρμογής μας. Το Retrofit αποτελεί μια παραλλαγή που προστατεύει το υπόλοιπο της εφαρμογής από τις αλλαγές που μπορεί να γίνουν στην εσωτερική υλοποίησή του.

Αυτή η αρχή συνηθίζεται σε συστήματα που χρησιμοποιούν εξωτερικές βιβλιοθήκες, APIs ή άλλα εσωτερικά στοιχεία του συστήματος, επιτρέποντας στο υπόλοιπο του συστήματος να λειτουργεί χωρίς να εξαρτάται άμεσα από την εσωτερική υλοποίηση αυτών των εξωτερικών στοιχείων.



6 Συμπεράσματα και Μελλοντικές Επεκτάσεις

Η εργασία έχει προσφέρει μια σφαιρική και ολοκληρωμένη ματιά στη διαδικασία του σχεδιασμού λογισμικού και της ανάλυσης απαιτήσεων, αφήνοντας περιθώριο για περαιτέρω ανάπτυξη και βελτίωση. Η προσεκτική ανάλυση και ο σχεδιασμός των απαιτήσεων της εφαρμογής αποτέλεσαν τη βάση για την ανάπτυξη του front-end και του back-end, καθώς και της βάσης δεδομένων. Η κατανόηση των αναγκών του χρήστη ήταν καθοριστική για τη σχεδίαση μιας λειτουργικής και αποτελεσματικής εφαρμογής.

Ο σχεδιασμός του front-end βασίστηκε σε σύγχρονες μεθόδους και τεχνολογίες, επιτρέποντας μια φιλική προς τον χρήστη διεπαφή. Τα στοιχεία του UI/UX σχεδιασμού συνέβαλαν στην εμπειρία του χρήστη, επιτρέποντάς του να επικοινωνεί και να χρησιμοποιεί την εφαρμογή με άνεση. Το back-end που χρησιμοποιείται στην εφαρμογή, Habit Tracker είναι υπεύθυνο για την εξυπηρέτηση των αιτημάτων των χρηστών, τη διαχείριση των δεδομένων και τη διασύνδεση με το front-end μέσω RESTful API. Επισημαίνεται η σημασία της αξιοπιστίας, της ασφάλειας και της απόδοσης για τη διασφάλιση μιας ομαλής εμπειρίας χρήστη. Τόσο στο front-end αλλά και στο back-end, η αξιοπιστία και η αποτελεσματικότητα των λειτουργιών βελτιώθηκε μέσω του GRASP patterns. Η χρήση αρχιτεκτονικών και σχεδιαστικών προτύπων συνέβαλε στην αξιόπιστη λειτουργία του συστήματος. Η σωστή λειτουργία και συνεργασία μεταξύ front-end, back-end και της βάσης δεδομένων οδήγησε στην επιτυχή υλοποίηση της εφαρμογής. Οι μέθοδοι αναγνώρισης σφαλμάτων και οι διαδικασίες εξόδου που υιοθετήθηκαν συνέβαλαν στη βελτίωση της ποιότητας του λογισμικού και στη βελτίωση της εμπειρίας των χρηστών.

Τα αποτελέσματα και οι διαδικασίες που ακολουθήθηκαν σε αυτήν την εργασία επιβεβαιώνουν τη σημασία της σωστής ανάλυσης, σχεδιασμού και υλοποίησης λογισμικού για την επίτευξη μιας αποτελεσματικής και ικανοποιητικής εμπειρίας χρήστη. Συνολικά, η εργασία παρέχει μια εποικοδομητική άποψη στον τομέα του σχεδιασμού λογισμικού και της ανάλυσης απαιτήσεων.

Οι μελλοντικές επεκτάσεις της εργασίας μπορούν να επικεντρωθούν σε πολλούς τομείς, λαμβάνοντας υπόψη τις ανάγκες του χρήστη, την ταχύτητα εξελισσόμενη τεχνολογία και τις βέλτιστες πρακτικές του χώρου του λογισμικού. Ορισμένες πιθανές μελλοντικές επεκτάσεις περιλαμβάνουν:

- **Βελτίωση της Ασφάλειας:** Οι μελλοντικές επεκτάσεις μπορούν να επικεντρωθούν στη βελτίωση της ασφάλειας της εφαρμογής. Αυτό περιλαμβάνει την εφαρμογή προηγμένων μεθόδων ασφαλείας, όπως (κρυπτογράφηση δεδομένων, πιστοποιητικά και πρόσβαση, αυτόματο backup και recovery κλπ).
- **Βελτίωση της Απόδοσης:** Ο σχεδιασμός του back-end μπορεί να εξελιχθεί για να επιτύχει βελτιωμένη απόδοση της εφαρμογής, προσφέροντας πιο γρήγορους χρόνους απόκρισης και αποδοτική χρήση των πόρων.
- **Ανάπτυξη για Διαφορετικές Πλατφόρμες:** Εφόσον η εφαρμογή αναπτύχθηκε αρχικά για μια συγκεκριμένη πλατφόρμα, μελλοντικές επεκτάσεις μπορούν να συμπεριλαμβάνουν τη διαθεσιμότητά της για διάφορες πλατφόρμες (π.χ. web, desktop κλπ).



- **Προσαρμογή σε Ανάγκες Χρηστών:** Η εξέλιξη της εφαρμογής μπορεί να γίνει με βάση τις ανάγκες των χρηστών και τις ανατροφοδοτήσεις που λαμβάνετε από αυτούς.
- **Κοινοποίηση Προόδου:** Προσθέστε μια λειτουργία που επιτρέπει στους χρήστες να κοινοποιούν την πρόδο τους με άλλους. Αυτό θα δημιουργήσει μια κοινότητα στήριξης και ανταλλαγής εμπειριών μεταξύ των χρηστών.
- **Διαχείριση Εκδόσεων:** Η διαχείριση εκδόσεων και η επικοινωνία με τους χρήστες για τυχόν αναβαθμίσεις και αλλαγές στην εφαρμογή είναι σημαντικές πτυχές των μελλοντικών επεκτάσεων.
- **Επέκταση της Εφαρμογής:** Οι μελλοντικές επεκτάσεις μπορούν να περιλαμβάνουν την προσθήκη νέων χαρακτηριστικών και λειτουργιών που να ανταποκρίνονται στις αναγκαίες επεκτάσεις της εφαρμογής.

Μερικές επεκτάσεις της εφαρμογής Habit Tracker μπορεί να είναι οι εξής:

- Ανταμοιβές και Κίνητρα: Δημιουργήστε ένα σύστημα ανταμοιβών και κινήτρων που θα προσφέρει στους χρήστες επιβραβεύσεις για την επίτευξη των στόχων τους. Αυτό μπορεί να συμπεριλαμβάνει ειδικά badges, εκπτώσεις ή ακόμη και διαγωνισμούς.
- Στατιστικά Στοιχεία: Προσφέρετε λεπτομερή στατιστικά στοιχεία για την πρόοδο των χρηστών, συμπεριλαμβάνοντας γραφήματα, διαγράμματα και πίνακες. Αυτά τα στοιχεία μπορούν να παρέχουν επιπλέον κίνητρο στους χρήστες να επιδιώξουν τους στόχους τους.
- Αναλυτικές Αναφορές: Προσφέρετε στους χρήστες αναλυτικές αναφορές που παρέχουν πληροφορίες για τις συνήθειές τους και την εξέλιξή τους στον χρόνο. Αυτό μπορεί να βοηθήσει τους χρήστες να παρακολουθούν την πρόδο τους και να ενισχύσει την αυτογνωσία τους.
- Επιλογή Αποστολής: Παροχή επιλογών για προσαρμογή του τρόπου αποστολής υπενθυμίσεων σύμφωνα με το χρονοδιάγραμμα και τις προτιμήσεις του κάθε χρήστη. Η επιλογή αποστολής υπενθυμίσεων μέσω email ή μηνύματος στο κινητό τηλέφωνο μπορεί να βελτιώσει σημαντικά την προσαρμογή στις προτιμήσεις και το προσωπικό χρονοδιάγραμμα κάθε χρήστη.

Οι μελλοντικές επεκτάσεις θα πρέπει να στοχεύουν στην βελτίωση της εφαρμογής σε όλους τους τομείς, από την ασφάλεια και την απόδοση μέχρι την εμπειρία των χρηστών και τη δυνατότητα εξάλειψης σφαλμάτων. Αυτές οι επεκτάσεις θα δώσουν στους χρήστες περισσότερους λόγους να χρησιμοποιούν την εφαρμογή και να επιδιώκουν τους στόχους τους. Επιπλέον, θα δημιουργήσουν ένα πιο διασκεδαστικό και κοινωνικό περιβάλλον που θα ενθαρρύνει τους χρήστες να παραμείνουν αφοσιωμένοι στη χρήση της εφαρμογής, ενισχύοντας την αίσθηση ελέγχου και ικανοποίησης που προσφέρει η εφαρμογή Habit Tracker.

Τέλος, η διαρκής ανάγκη για βελτίωση και ανάπτυξη στον τομέα της τεχνολογίας είναι προφανής, υπογραμμίζοντας τη σημασία της συνεχούς εξέλιξης και προόδου στον ψηφιακό κόσμο.



Βιβλιογραφία

- [1] Karl Wieggers, Joy Beatty, (August 2013). Software Requirements, 3rd Edition.
- [2] Suzanne Robertson, James Robertson, (2012). Mastering the Requirements Process, Publisher Addison-Wesley.
- [3] Van Lamsweerde, Axel, (2009). Requirements Engineering: From System Goals To Uml Models To Software Specifications, Published By John Wiley Sons.
- [4] Clear James, (2018). Atomic Habits: An Easy & Proven Way to Build Good Habits & Break Bad Ones.
- [5] Charles Duhigg (2012). The Power of Habit: Why We Do What We Do in Life and Business is a book.
- [6] Fogg, Bj (2019). Tiny Habits: The Small Changes That Change Everything.
- [7] Gretchen Rubin (2015). Better Than Before: What I Learned About Making and Breaking Habits.
- [8] James O. Prochaska, John Norcross, Carlo DiClemente, (1994). Changing for Good: A Revolutionary Six-Stage Program for Overcoming Bad Habits and Moving Your Life Positively Forward.
- [9] Greg McKeown, (2011). Essentialism: The Disciplined Pursuit of Less.
- [10] Chris Aquino, Todd Gandee, Publisher Big Nerd Ranch, (2016). Front-End Web Development: The Big Nerd Ranch Guide.
- [11] Jenny Preece, Yvonne Rogers, Helen Sharp 4th edition (May, 2015). Interaction Design: Beyond Human-Computer Interaction.
- [12] Jenifer Tidwell, December (2010). Designing Interfaces, 2nd Edition.
- [13] Terry Lee Stone, Sean Adams, Noreen Morioka, (2006). Color Design Workbook: A Real-World Guide to Using Color in Graphic Design.
- [14] Donald A. Norman, (November 2013). The Design of Everyday Things.
- [15] Richard E. Smith, (2002). Authentication: From Passwords To Public Keys .
- [16] Ross J. Anderson, (2008). Security Engineering A Guide to Building Dependable Distributed Systems, 2nd Edition.
- [17] Brenda Laurel, S. Joy Mountford, (1990). The Art of Human-Computer Interface Design.
- [18] Bill Phillips, Chris Stewart, Brian Hardy, Kristin Marsicano (2015). Android Programming: The Big Nerd Ranch Guide, 2nd edition.
- [19] Robert C. Martin, September (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design.
- [20] David Thomas, Andrew Hunt, (September 2019). The Pragmatic Programmer: Your journey to mastery, 20th Anniversary Edition, 2nd Edition.
- [21] Leonard Richardson, Mike Amundsen, Sam Ruby, (September 2013). RESTful Web APIs.
- [22] Mark Masse, (October 2011). REST API Design Rulebook.
- [23] Regina O. Obe, Leo S. Hsu, December, (2014) PostgreSQL: Up and Running, 2rd Edition.
- [24] Korry Douglas, Douglas Susan, (2003). PostgreSQL: The Comprehensive Guide To Building, Programming and Administering PostgreSQL Databases.



- [25] William S. Vincent, (2018). Django for APIs: Build web APIs with Python & Django.
- [26] Daniel Roy Greenfeld, Audrey Roy Greenfeld, (2015). Two Scoops of Django: Best Practices for Django 1.8, 3rd Edition.
- [27] Andreas Wittig, Michael Wittig, (2015). Amazon Web Services in Action Publisher: Manning Publications.
- [28] AWS Documentation Amazon EC2 [διαδίκτυο] από : https://docs.aws.amazon.com/ec2/?icmpid=docs_homepage_featuredsvcs
- [29] Larman, Craig, (2001). Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and the Unified Process 2nd Edition.
- [30] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, (1995). Design Patterns: Elements of Reusable Object-Oriented Software.
- [31] Fowler Martin, (2003). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.
- [32] Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, (2000). Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2.
- [33] Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, (2004). Head First Design Patterns: A Brain-Friendly Guide.
- [34] GRASP (object-oriented design) - Wikipedia, the free encyclopedia, (2011). [διαδίκτυο] από : [http://en.wikipedia.org/wiki/GRASP_\(object-oriented_design\)](http://en.wikipedia.org/wiki/GRASP_(object-oriented_design))



Παράρτημα I (Front End)

IntroActivity

```
package com.example.myapplicationforhabits;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.viewpager2.widget.ViewPager2;

import com.tbunomo.viewpagerdotsindicator.WormDotsIndicator;

import java.util.ArrayList;
import java.util.List;

public class IntroActivity extends AppCompatActivity {

    private List<IntroView> introViewList;
    private Button btnStartApp;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_intro);

        // Check if it's the user's first time
        SharedPreferences sharedPreferences =
        getSharedPreferences("UserPrefs", MODE_PRIVATE);
        boolean hasSeenIntro =
        sharedPreferences.getBoolean("hasSeenIntro", false);

        if (hasSeenIntro) {
            // User has seen the intro, navigate to UserAuthActivity or
            MainActivity
            navigateToUserAuthActivity();
        } else {
            // User hasn't seen the intro, show the intro screen
            initViewPager();
            initStartButton();
            // Set the flag to indicate that the user has seen the
            intro
            SharedPreferences.Editor editor = sharedPreferences.edit();
            editor.putBoolean("hasSeenIntro", true);
            editor.apply();
        }
    }

    private void initViewPager() {
        ViewPager2 viewPager2 = findViewById(R.id.viewPager2);
        WormDotsIndicator wormDotsIndicator =
```



```
findViewById(R.id.wormDotsIndicator);

    addToIntroView();

    ViewPagerIntroAdapter viewPagerIntroAdapter = new
ViewPagerIntroAdapter(introViewList);
    viewPager2.setAdapter(viewPagerIntroAdapter);
    viewPager2.setOrientation(ViewPager2.ORIENTATION_HORIZONTAL);

    wormDotsIndicator.setViewPager2(viewPager2);

    viewPager2.registerOnPageChangeCallback(new
ViewPager2.OnPageChangeCallback() {
        @Override
        public void onPageScrolled(int position, float
positionOffset, int positionOffsetPixels) {
            if (position == 2) {
                animationButton();
            }
            super.onPageScrolled(position, positionOffset,
positionOffsetPixels);
        }
    });
}

private void initStartButton() {
    btnStartApp = findViewById(R.id.btn_start_app);
    btnStartApp.setVisibility(View.INVISIBLE);
}

private void animationButton() {
    btnStartApp.setVisibility(View.VISIBLE);

    btnStartApp.animate().setDuration(1500).alpha(1f);

    btnStartApp.setOnClickListener(v -> {
        navigateToUserAuthActivity();
    });
}

private void navigateToUserAuthActivity() {
    Intent intent = new Intent(getApplicationContext(),
UserAuthActivity.class);
    startActivity(intent);
    finish();
}

private void addToIntroView() {
    // Create some items that you want to add to your viewpager
    introViewList = new ArrayList<>();
    introViewList.add(new IntroView("Welcome to My Habit Tracker
App!", R.drawable.ic_workout));
    introViewList.add(new IntroView("Here you can track your
habits!", R.drawable.ic_vitamin));
    introViewList.add(new IntroView("Tap the button to get
started!", R.drawable.ic_habit2));
}
}
```



IntroView

```
package com.example.myapplicationforhabits;

public class IntroView {
    private final String description;
    private final int image;

    public IntroView(String description, int image) {
        this.description = description;
        this.image = image;
    }

    public String getDescription() {
        return description;
    }

    public int getImage() {
        return image;
    }
}
```

ViewPagerIntroAdapter

```
package com.example.myapplicationforhabits;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import java.util.List;

public class ViewPagerIntroAdapter extends
RecyclerView.Adapter<ViewPagerIntroAdapter.IntroViewHolder> {

    private final List<IntroView> introViewList;

    public ViewPagerIntroAdapter(List<IntroView> introViewList) {
        this.introViewList = introViewList;
    }

    @NonNull
    @Override
    public IntroViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
        View view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.intro_item_pa
ge, parent, false);
        return new IntroViewHolder(view);
    }
}
```



```
@Override
public void onBindViewHolder(@NonNull IntroViewHolder holder, int
position) {
    IntroView introView = introViewList.get(position);
    holder.introImage.setImageResource(introView.getImage());
    holder.introText.setText(introView.getDescription());
}

@Override
public int getItemCount() {
    return introViewList.size();
}

public static class IntroViewHolder extends RecyclerView.ViewHolder
{
    ImageView introImage;
    TextView introText;

    public IntroViewHolder(@NonNull View itemView) {
        super(itemView);
        introImage = itemView.findViewById(R.id.iv_image_intro);
        introText =
itemView.findViewById(R.id.tv_description_intro);
    }
}
}
```

MainActivity

```
package com.example.myapplicationforhabits;

import android.app.AlertDialog;
import android.app.DatePickerDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.SwitchCompat;
import androidx.appcompat.widget.Toolbar;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
```



```
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Locale;
import java.util.Map;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class MainActivity extends AppCompatActivity {
    private HabitAdapter habitAdapter;
    RecyclerView habitRecyclerView;
    private SharedPreferences sharedPreferences;
    private ApiService apiService;
    private boolean menuIsActive = true;

    private Menu menu; // Define menu as a class-level variable
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Set up the Toolbar
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        // Initialize RecyclerView and Adapter
        habitRecyclerView = findViewById(R.id.habitRecyclerView);
        habitRecyclerView.setLayoutManager(new
LinearLayoutManager(this));
        apiService =
RetrofitClient.getClient().create(ApiService.class);
        habitAdapter = new HabitAdapter(this, apiService);
        habitRecyclerView.setAdapter(habitAdapter);

        // Initialize SharedPreferences and fetch auth token
        sharedPreferences = getSharedPreferences("UserPrefs",
MODE_PRIVATE);
        String accessToken = sharedPreferences.getString("authToken",
"");

        // Check if accessToken is not empty and fetch habits
        if (!accessToken.isEmpty()) {
            Log.d("MainActivity", "Token is saved: " + accessToken);
            fetchUserHabits(accessToken, false); // Pass false as the
default switcher state
        } else {
            Log.d("MainActivity", "Token Not Found");
            // Redirect to Login or perform other action
        }

        // Other UI components like AddHabitButton
        Button addHabitButton = findViewById(R.id.addHabitButton);
    }
}
```



```
        addHabitButton.setOnClickListener(v ->
showEditHabitDialog(null));
    }

    void showEditHabitDialog(final Habit habit) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle(habit == null ? "Add New Habit" : "Edit
Habit");

        String accessToken = sharedPreferences.getString("authToken",
        "");
        if (TextUtils.isEmpty(accessToken)) {
            Toast.makeText(MainActivity.this, "No access token
available", Toast.LENGTH_SHORT).show();
            return;
        }

        View view =
getLayoutInflater().inflate(R.layout.dialog_add_habit, null);
        builder.setView(view);

        final EditText habitNameEditText =
view.findViewById(R.id.habitNameEditText);
        final EditText habitDescriptionEditText =
view.findViewById(R.id.habitDescriptionEditText);
        final EditText reasonEditText =
view.findViewById(R.id.reasonEditText);
        final TextView startDayTextView =
view.findViewById(R.id.startDayTextView);
        final TextView endDayTextView =
view.findViewById(R.id.endDayTextView);

        // Extract selected days from the habit and pre-select
checkboxes
        List<String> selectedDays = new ArrayList<>();
        if (habit != null) {
            JSONObject daysOfWeekObject = habit.getDaysOfWeek();
            if (daysOfWeekObject != null) {
                Iterator<String> keys = daysOfWeekObject.keys();
                while (keys.hasNext()) {
                    String day = keys.next();
                    try {
                        boolean isSelected =
daysOfWeekObject.getBoolean(day);
                        if (isSelected) {
                            selectedDays.add(day);
                        }
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
        Log.d("EditHabitDialog", "Selected Days: " +
selectedDays.toString());

        habitNameEditText.setText(habit.getName());
        habitDescriptionEditText.setText(habit.getDescription());
        reasonEditText.setText(habit.getReason());
    }
}
```



```
startDayTextView.setText(habit.getStartDay());
endDayTextView.setText(habit.getEndDay());

CheckBox mondayCheckboxOnShow =
view.findViewById(R.id.checkboxMonday);
CheckBox tuesdayCheckboxOnShow =
view.findViewById(R.id.checkboxTuesday);
CheckBox wednesdayCheckboxOnShow =
view.findViewById(R.id.checkboxWednesday);
CheckBox thursdayCheckboxOnShow =
view.findViewById(R.id.checkboxThursday);
CheckBox fridayCheckboxOnShow =
view.findViewById(R.id.checkboxFriday);
CheckBox saturdayCheckboxOnShow =
view.findViewById(R.id.checkboxSaturday);
CheckBox sundayCheckboxOnShow =
view.findViewById(R.id.checkboxSunday);
//Switch switcher =
dialog.findViewById(R.id.action_switcher);

// Perform null checks before setting checkbox states
if (mondayCheckboxOnShow != null && tuesdayCheckboxOnShow
!= null && wednesdayCheckboxOnShow != null
&& thursdayCheckboxOnShow != null &&
fridayCheckboxOnShow != null && saturdayCheckboxOnShow != null
&& sundayCheckboxOnShow != null) {
mondayCheckboxOnShow.setChecked(false);
tuesdayCheckboxOnShow.setChecked(false);
wednesdayCheckboxOnShow.setChecked(false);
thursdayCheckboxOnShow.setChecked(false);
fridayCheckboxOnShow.setChecked(false);
saturdayCheckboxOnShow.setChecked(false);
sundayCheckboxOnShow.setChecked(false);

for (String day : selectedDays) {
switch (day) {
case "Mon":
mondayCheckboxOnShow.setChecked(true);
break;
case "Tue":
tuesdayCheckboxOnShow.setChecked(true);
break;
case "Wed":
wednesdayCheckboxOnShow.setChecked(true);
break;
case "Thu":
thursdayCheckboxOnShow.setChecked(true);
break;
case "Fri":
fridayCheckboxOnShow.setChecked(true);
break;
case "Sat":
saturdayCheckboxOnShow.setChecked(true);
break;
case "Sun":
sundayCheckboxOnShow.setChecked(true);
break;
}
}
}
```



```
    }  
    }  
  
    final Calendar calendar = Calendar.getInstance();  
    final int currentYear = calendar.get(Calendar.YEAR);  
    final int currentMonth = calendar.get(Calendar.MONTH);  
    final int currentDayOfMonth =  
calendar.get(Calendar.DAY_OF_MONTH);  
  
    final DatePickerDialog.OnDateSetListener  
startDatePickerListener = new DatePickerDialog.OnDateSetListener() {  
        @Override  
        public void onDateSet(DatePicker view, int year, int  
monthOfYear, int dayOfMonth) {  
            String selectedDate =  
String.format(Locale.getDefault(), "%d-%02d-%02d", year, monthOfYear +  
1, dayOfMonth);  
            startDayTextView.setText(selectedDate);  
        }  
    };  
  
    final DatePickerDialog.OnDateSetListener endDatePickerListener  
= new DatePickerDialog.OnDateSetListener() {  
        @Override  
        public void onDateSet(DatePicker view, int year, int  
monthOfYear, int dayOfMonth) {  
            String selectedDate =  
String.format(Locale.getDefault(), "%d-%02d-%02d", year, monthOfYear +  
1, dayOfMonth);  
            endDayTextView.setText(selectedDate);  
        }  
    };  
  
    Button dayStartButton = view.findViewById(R.id.dayStartButton);  
    dayStartButton.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            DatePickerDialog datePickerDialog = new  
DatePickerDialog(MainActivity.this,  
                startDatePickerListener, currentYear,  
currentMonth, currentDayOfMonth);  
            datePickerDialog.show();  
        }  
    });  
  
    Button dayEndButton = view.findViewById(R.id.dayEndButton);  
    dayEndButton.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            DatePickerDialog datePickerDialog = new  
DatePickerDialog(MainActivity.this,  
                endDatePickerListener, currentYear,  
currentMonth, currentDayOfMonth);  
            datePickerDialog.show();  
        }  
    });  
  
    // Add the "Delete" button only if habit is not null (i.e.,  
when editing)  
    if (habit != null) {
```



```
        builder.setNeutralButton("Delete", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which)
        {
            AlertDialog.Builder deleteConfirmationBuilder = new
AlertDialog.Builder(MainActivity.this);
            deleteConfirmationBuilder.setTitle("Confirm
Deletion");
            deleteConfirmationBuilder.setMessage("Are you sure
you want to delete this habit?");

            deleteConfirmationBuilder.setPositiveButton("Yes",
new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int
which) {
                    // Trigger the habit deletion action here
                    int habitId = (int) habit.getId();
                    Log.e("Delete Habit",
String.valueOf(habitId));
                    apiService.deleteHabit(habitId, "Bearer " +
accessToken).enqueue(new Callback<Void>() {
                        @Override
                        public void onResponse(Call<Void> call,
Response<Void> response) {
                            if (response.isSuccessful()) {
                                // Successfully deleted the
habit from the server
                                Toast.makeText(MainActivity.this, "Habit deleted successfully",
Toast.LENGTH_SHORT).show();
                                // Call the method to fetch
updated habits and update the UI
                                MenuItem switchMenuItem =
menu.findItem(R.id.action_view_archived);
                                SwitchCompat switcher =
(SwitchCompat)
switchMenuItem.getActionView().findViewById(R.id.action_switcher);
                                fetchUserHabits(accessToken,
switcher.isChecked());
                                // No need to update local data
here
                            } else {
                                // Failed to delete the habit
on the server
                                Toast.makeText(MainActivity.this, "Failed to delete habit",
Toast.LENGTH_SHORT).show();
                            }
                        }
                    });
                }
            });
        }
    }

    @Override
    public void onFailure(Call<Void> call,
Throwable t) {
        // Handle network or other errors
        Toast.makeText(MainActivity.this,
"Error deleting habit", Toast.LENGTH_SHORT).show();
    }
}
```



```
        });
        dialog.dismiss();
    }
});

deleteConfirmationBuilder.setNegativeButton("No",
new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int
which) {
        dialog.dismiss();
    }
});
deleteConfirmationBuilder.create().show();
}
});
}

AlertDialog dialog = builder
    .setPositiveButton(habit == null ? "Add" : "Save",
null)
    .setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int
which) {
        dialog.dismiss();
    }
})
    .create();

dialog.setOnShowListener(new DialogInterface.OnShowListener() {
    @Override
    public void onShow(DialogInterface dialogInterface) {
        Button saveButton =
dialog.getButton(AlertDialog.BUTTON_POSITIVE);
        saveButton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String updatedName =
habitNameEditText.getText().toString();

                // Check if the updated name is empty
                if (updatedName.isEmpty()) {
                    Toast.makeText(MainActivity.this, "Habit
must have title", Toast.LENGTH_SHORT).show();
                    return; // Exit the method early if the
title is empty
                }

                String updatedDescription =
habitDescriptionEditText.getText().toString();
                String updatedReason =
reasonEditText.getText().toString();
                String updatedStartDay =
startDayTextView.getText().toString();
                String updatedEndDay =
endDayTextView.getText().toString();
                CheckBox checkboxMonday =
```



```
dialog.findViewById(R.id.checkboxMonday);
    CheckBox checkboxTuesday =
dialog.findViewById(R.id.checkboxTuesday);
    CheckBox checkboxWednesday =
dialog.findViewById(R.id.checkboxWednesday);
    CheckBox checkboxThursday =
dialog.findViewById(R.id.checkboxThursday);
    CheckBox checkboxFriday =
dialog.findViewById(R.id.checkboxFriday);
    CheckBox checkboxSaturday =
dialog.findViewById(R.id.checkboxSaturday);
    CheckBox checkboxSunday =
dialog.findViewById(R.id.checkboxSunday);
    SwitchCompat switcher =
dialog.findViewById(R.id.action_switcher);

        // Check if updatedStartDay is in yyyy-MM-dd
format, if not, set it to null
        if (!updatedStartDay.matches("^\\d{4}-\\d{2}-
\\d{2}$")) {
            updatedStartDay = null;
        }

        // Check if updatedEndDay is in yyyy-MM-dd
format, if not, set it to null
        if (!updatedEndDay.matches("^\\d{4}-\\d{2}-
\\d{2}$")) {
            updatedEndDay = null;
        }

        // Check the state of each checkbox and update
the days_of_the_week map
        Map<String, Boolean> daysOfWeekMap = new
HashMap<>();
        daysOfWeekMap.put ("Monday",
checkboxMonday.isChecked());
        daysOfWeekMap.put ("Tuesday",
checkboxTuesday.isChecked());
        daysOfWeekMap.put ("Wednesday",
checkboxWednesday.isChecked());
        daysOfWeekMap.put ("Thursday",
checkboxThursday.isChecked());
        daysOfWeekMap.put ("Friday",
checkboxFriday.isChecked());
        daysOfWeekMap.put ("Saturday",
checkboxSaturday.isChecked());
        daysOfWeekMap.put ("Sunday",
checkboxSunday.isChecked());

        Habit updatedHabit;
        Habit createHabit;

        if (habit != null) {
            // If habit is not null, update the
existing habit
            updatedHabit = new Habit(
                habit.getId(),
                updatedName,
```



```
        updatedDescription,
        updatedReason,
        habit.getCategory(),
        daysOfWeekMap,
        updatedStartDay,
        updatedEndDay,
        habit.isActive()
    );

    apiService.updateHabit((int) habit.getId(),
"Bearer " + accessToken, updatedHabit).enqueue(new Callback<Habit>() {
        @Override
        public void onResponse(Call<Habit>
call, Response<Habit> response) {
            if (response.isSuccessful()) {

Toast.makeText(MainActivity.this, "Habit updated successfully",
Toast.LENGTH_SHORT).show();

                if (switcher != null) { // Null
check added here

fetchUserHabits(accessToken, switcher.isChecked()); // Refresh the list
of habits
                    }
                } else {

Toast.makeText(MainActivity.this, "Failed to update habit",
Toast.LENGTH_SHORT).show();

                    }
                }

        @Override
        public void onFailure(Call<Habit> call,
Throwable t) {
            Toast.makeText(MainActivity.this,
"Error updating habit", Toast.LENGTH_SHORT).show();
        }
    });

    } else {
        // If habit is null, create a new habit
        createHabit = new Habit(
            updatedName,
            updatedDescription,
            updatedReason,
            daysOfWeekMap,
            updatedStartDay,
            updatedEndDay

        );

        apiService.createHabit("Bearer " +
accessToken, createHabit).enqueue(new Callback<Habit>() {
            @Override
            public void onResponse(Call<Habit>
call, Response<Habit> response) {
                if (response.isSuccessful()) {

Toast.makeText(MainActivity.this, "Habit created successfully",
```



```
Toast.LENGTH_SHORT).show();

// Check if switcher is not
null before using it
if (switcher != null) {
fetchUserHabits(accessToken, switcher.isChecked()); // Refresh the list
of habits
} else {
fetchUserHabits(accessToken, menuIsActive); // Default to menuIsActive
if switcher is null
}
} else {

Toast.makeText(MainActivity.this, "Failed to create habit",
Toast.LENGTH_SHORT).show();
}
}

@Override
public void onFailure(Call<Habit> call,
Throwable t) {
Toast.makeText(MainActivity.this,
"Error creating habit", Toast.LENGTH_SHORT).show();
});
}
dialog.dismiss();
}
});
}
dialog.show();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
getMenuInflater().inflate(R.menu.menu_main, menu);
this.menu = menu; // Initialize menu

// Find the switcher item
MenuItem switchMenuItem =
menu.findItem(R.id.action_view_archived);
SwitchCompat switcher = (SwitchCompat)
switchMenuItem.getActionView().findViewById(R.id.action_switcher);

// Set the initial state of the switcher
switcher.setChecked(menuIsActive);

// Fetch habits based on initial switch state
String accessToken = sharedPreferences.getString("authToken",
"");
fetchUserHabits(accessToken, switcher.isChecked());

// Handle switcher state change
switcher.setOnCheckedChangeListener(new
```



```
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
        // Fetch habits based on switcher state
        fetchUserHabits(accessToken, isChecked);
    }
});

return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    String accessToken = sharedPreferences.getString("authToken",
    "");

    if (id == R.id.nav_exit) {
        // Show the confirmation dialog for exiting the app
        showExitConfirmationDialog();
        return true;
    } else if (id == R.id.action_logout) {
        // Show the confirmation dialog for logging out
        showLogoutConfirmationDialog();
        return true;
    } else if (id == R.id.action_view_archived) {
        // Fetch habits based on switcher state
        SwitchCompat switcher =
item.getActionView().findViewById(R.id.action_switcher);
        fetchUserHabits(accessToken, switcher.isChecked());
        return true;
    }

    return super.onOptionsItemSelected(item);
}

private void showExitConfirmationDialog() {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Exit App");
    builder.setMessage("Are you sure you want to exit the app?");
    builder.setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Log out the user by clearing the saved user ID from
SharedPreferences
            clearUserIdFromSharedPreferences();

            // Finish the entire activity stack
            finishAffinity();
        }
    });
    builder.setNegativeButton("No", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
        }
    });
}
```



```
    }
    });
    builder.show();
}

private void showLogoutConfirmationDialog() {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Log Out");
    builder.setMessage("Are you sure you want to log out?");
    builder.setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            // Log out the user by clearing the saved user ID from
SharedPreferences
            clearUserIdFromSharedPreferences();

            // Redirect the user to the login or sign-up screen
startActivity(new Intent(MainActivity.this,
UserAuthActivity.class));
            finish();
        }
    });
    builder.setNegativeButton("No", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
        }
    });
    builder.show();
}

private void clearUserIdFromSharedPreferences() {
    // Clear the user ID from SharedPreferences
    SharedPreferences sharedPreferences =
getSharedPreferences("UserPrefs", MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.remove("userId");
    editor.apply();
}

@Override
protected void onResume() {
    super.onResume();
    // Load data if needed
//    loadHabits(); // Load the habits when the activity resumes
}

@Override
protected void onPause() {
    super.onPause();
    // Save data if needed
}

private void fetchUserHabits(String accessToken, boolean isActive)
{
```



```
String authHeader = "Bearer " + accessToken;
apiService.getUserHabits(authHeader).enqueue(new
Callback<List<Habit>>() {
    @Override
    public void onResponse(Call<List<Habit>> call,
Response<List<Habit>> response) {
        if (response.isSuccessful() && response.body() != null)
        {
            List<Habit> allHabits = response.body();
            List<Habit> filteredHabits =
filterHabitsByActivity(allHabits, isActive);

            for (Habit habit : filteredHabits) {
                Log.d("MAINACTIVITY", "Habit: " +
habit.toString());
            }

            // Update the adapter with the filtered habits
            habitAdapter.setHabits(filteredHabits);
            habitAdapter.notifyDataSetChanged();
        } else {
            Toast.makeText(MainActivity.this, "Failed to fetch
habits: " + response.code(), Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onFailure(Call<List<Habit>> call, Throwable t)
    {
        Toast.makeText(MainActivity.this, "Network error while
fetching habits: " + t.getMessage(), Toast.LENGTH_SHORT).show();
    }
});
}

private List<Habit> filterHabitsByActivity(List<Habit> habits,
boolean isActive) {
    List<Habit> filteredHabits = new ArrayList<>();

    for (Habit habit : habits) {
        // Check if the habit's activity status matches the filter
        if (habit.isActive() == isActive) {
            filteredHabits.add(habit);
        }
    }

    return filteredHabits;
}
}
```

HabitAdapter

```
package com.example.myapplicationforhabits;

import android.app.AlarmManager;
import android.app.AlertDialog;
import android.app.PendingIntent;
import android.app.TimePickerDialog;
```



```
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Build;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.TimePicker;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class HabitAdapter extends
RecyclerView.Adapter<HabitAdapter.ViewHolder> {
    private List<Habit> habitList;
    private ApiService apiService;
    private Context context;
    private SharedPreferences sharedPreferences;

    // Constructor
    public HabitAdapter(Context context, ApiService apiService) {
        this.context = context;
        this.apiService = apiService;
        this.habitList = new ArrayList<>();
        sharedPreferences = context.getSharedPreferences("UserPrefs",
Context.MODE_PRIVATE);
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
        View view =
LayoutInflater.from(context).inflate(R.layout.item_habit, parent,
```



```
false);
    return new ViewHolder(view);
}

@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    Habit habit = habitList.get(position);

    holder.habitNameItemTextView.setText(habit.getName());
    holder.habitDescriptionItemTextView.setText("Description: " +
habit.getDescription());
    holder.reasonItemTextView.setText("Reason: " +
habit.getReason());
    holder.startDayItemTextView.setText("Start Day: " +
habit.getStartDay());
    holder.endDayItemTextView.setText("End Day: " +
habit.getEndDay());
    holder.selectedDaysTextView.setText("Selected Days: " +
habit.getDaysAsString());

    // Set the Archive checkbox state without triggering the
listener
    holder.checkboxActive.setOnCheckedChangeListener(null);
    holder.checkboxActive.setChecked(habit.isActive());
    holder.checkboxActive.setOnCheckedChangeListener((buttonView,
isChecked) -> {
        // Here you will call the method to update the habit
archiving status using the apiService
        updateHabitArchiving(habit, isChecked);
    });

    // Set the checkbox state without triggering the listener
    holder.checkboxCompleted.setOnCheckedChangeListener(null);
    holder.checkboxCompleted.setChecked(habit.isCompleted());

holder.checkboxCompleted.setOnCheckedChangeListener((buttonView,
isChecked) -> {
        // Here you will call the method to update the habit
completion
status using the apiService
        updateHabitCompletion(habit, isChecked);
    });

    // Check if the habit has reminders
    if (habit.hasReminders()) {
        // If reminders exist, set the bell icon to the image for
having reminders
        holder.bellIcon.setImageResource(R.drawable.ic_bell3);
    } else {
        // If no reminders, set the bell icon to the image for no
reminders
        holder.bellIcon.setImageResource(R.drawable.ic_bell1);
    }

    // Handling expand/collapse for additional details
    holder.expandCollapseTextView.setOnClickListener(v -> {
        if (holder.additionalDetailsLayout.getVisibility() ==
View.VISIBLE) {
```



```
holder.additionalDetailsLayout.setVisibility(View.GONE);
        holder.expandCollapseTextView.setText("Show More");
    } else {

holder.additionalDetailsLayout.setVisibility(View.VISIBLE);
        holder.expandCollapseTextView.setText("Show Less");
    }
});
// Set a click listener for the habit item
holder.itemView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Call the showEditHabitDialog method in MainActivity
to edit the habit
        if (apiService != null) {
            ((MainActivity)
context).showEditHabitDialog(habit);
        }
    }
});
}

private void updateHabitArchiving(Habit habit, boolean newIsActive)
{
    String accessToken = sharedPreferences.getString("authToken",
    "");
    if (!accessToken.isEmpty()) {
        Log.d("MainActivity", "Token is saved: " + accessToken);
    } else {
        Log.d("MainActivity", "Token Not Found");
        // Redirect to Login or perform other action
    }

    habit = new Habit(
        habit.getId(),
        habit.getName(),
        habit.getDaysOfWeekMap(),
        habit.isCompleted(),
        newIsActive
    );

    ApiService apiService =
RetrofitClient.getClient().create(ApiService.class);
    apiService.updateHabit((int) habit.getId(), "Bearer " +
accessToken, habit).enqueue(new Callback<Habit>() {
        @Override
        public void onResponse(Call<Habit> call, Response<Habit>
response) {
            if (response.isSuccessful()) {
                Log.d("HabitAdapter", "Habit archiving updated
successfully on server");
            } else {
                Log.e("HabitAdapter", "Response body is null");
                int responseCode = response.code();
                String errorBody = response.errorBody() != null ?
response.errorBody().toString() : "Unknown error";
                Log.e("HabitAdapter", "Error updating habit.
Response code: " + responseCode + ", Error: " + errorBody);
            }
        }
    });
}
```



```
        // You can display an error message to the user
        based on the response code
    }
}

@Override
public void onFailure(Call<Habit> call, Throwable t) {
    Log.e("HabitAdapter", "Failed to update habit
archiving", t);
}
});
}

private void updateHabitCompletion(Habit habit, boolean
newIsCompleted) {
    String accessToken = sharedPreferences.getString("authToken",
    "");
    // Check if accessToken is not empty and fetch habits
    if (!accessToken.isEmpty()) {
        Log.d("MainActivity", "Token is saved: " + accessToken);
    } else {
        Log.d("MainActivity", "Token Not Found");
        // Redirect to Login or perform other action
    }

    habit = new Habit(
        habit.getId(),
        habit.getName(),
        habit.getDaysOfWeekMap(),
        newIsCompleted
    );

    // Get an instance of ApiService using Retrofit
    ApiService apiService =
RetrofitClient.getClient().create(ApiService.class);
    // Make the call to update the habit. Assume 'updateHabit' is a
method in your ApiService interface.
    apiService.updateHabit((int) habit.getId(), "Bearer " +
accessToken, habit).enqueue(new Callback<Habit>() {
        @Override
        public void onResponse(Call<Habit> call, Response<Habit>
response) {
            if (response.isSuccessful()) {
                Log.d("HabitAdapter", "Habit completion updated
successfully on server");
                // Refresh the list of habits
            } else {
                Log.e("HabitAdapter", "Response body is null");
                int responseCode = response.code();
                String errorBody = response.errorBody() != null ?
response.errorBody().toString() : "Unknown error";
                Log.e("HabitAdapter", "Error updating habit.
Response code: " + responseCode + ", Error: " + errorBody);
                // You can display an error message to the user
                based on the response code
            }
        }
    });
}

@Override
```

```
        public void onFailure(Call<Habit> call, Throwable t) {
            Log.e("HabitAdapter", "Failed to update habit
completion", t);
        }
    });
}

// updates the list inside the adapter
public void setHabits(List<Habit> newHabits) {
    this.habitList = newHabits;
    notifyDataSetChanged();
}

@Override
public int getItemCount() {
    return habitList != null ? habitList.size() : 0;
}

public class ViewHolder extends RecyclerView.ViewHolder {
    TextView habitNameItemTextView;
    TextView habitDescriptionItemTextView;
    TextView reasonItemTextView;
    TextView startDayItemTextView;
    TextView endDayItemTextView;
    TextView selectedDaysTextView;
    CheckBox checkBoxCompleted;
    ImageButton bellIcon;

    CheckBox checkBoxActive;

    LinearLayout additionalDetailsLayout;
    TextView expandCollapseTextView;

    public ViewHolder(@NonNull View itemView) {
        super(itemView);

        habitNameItemTextView =
itemView.findViewById(R.id.habitNameItemTextView);
        habitDescriptionItemTextView =
itemView.findViewById(R.id.habitDescriptionItemTextView);
        reasonItemTextView =
itemView.findViewById(R.id.reasonItemTextView);
        startDayItemTextView =
itemView.findViewById(R.id.startDayItemTextView);
        endDayItemTextView =
itemView.findViewById(R.id.endDayItemTextView);
        selectedDaysTextView =
itemView.findViewById(R.id.selectedDaysTextView);
        checkBoxCompleted =
itemView.findViewById(R.id.checkBoxCompleted);
        bellIcon = itemView.findViewById(R.id.bellIcon);
        additionalDetailsLayout =
itemView.findViewById(R.id.additionalDetailsLayout);
        expandCollapseTextView =
itemView.findViewById(R.id.expandCollapseTextView);
        checkBoxActive =
itemView.findViewById(R.id.checkBoxActive);

        checkBoxActive.setOnCheckedChangeListener((buttonView,
isChecked) -> {
```



```
        // Call the method to update the habit archiving status
        using the apiService

        updateHabitArchiving(habitList.get(getAdapterPosition()), isChecked);
    });

    // Set click listener for expand/collapse view
    expandCollapseTextView.setOnClickListener(v -> {
        if (additionalDetailsLayout.getVisibility() ==
View.VISIBLE) {
            additionalDetailsLayout.setVisibility(View.GONE);
            expandCollapseTextView.setText("Show More");
        } else {
            additionalDetailsLayout.setVisibility(View.VISIBLE);
            expandCollapseTextView.setText("Show Less");
        }
    });

    // Set click listener for the bell icon
    bellIcon.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Call the method to show the reminder dialog
            showAddReminderDialog(habitList.get(getAdapterPosition()),
            getAdapterPosition());
        }
    });
}

private void showAddReminderDialog(final Habit habit, int position)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(context);

    // Set the dialog title based on whether a reminder exists
    if (habit.hasReminders()) {
        builder.setTitle("Update Reminder");
    } else {
        builder.setTitle("Add Reminder");
    }

    View dialogView =
LayoutInflater.from(context).inflate(R.layout.dialog_add_reminder,
null);
    builder.setView(dialogView);

    EditText reminderMessageEditText =
dialogView.findViewById(R.id.reminderMessageEditText);
    TextView reminderTimeTextView =
dialogView.findViewById(R.id.reminderTimeTextView);
    Button selectTimeButton =
dialogView.findViewById(R.id.selectTimeButton);
}
```



```
        if (habit.hasReminders()) {
            // Pre-populate the dialog with the existing reminder
            details
            Reminder existingReminder = habit.getReminders().get(0); //
            Assuming one reminder per habit

            // Log the existing reminder details
            Log.d("ReminderDialog", "Existing Reminder Message: " +
            existingReminder.getReminderMessage());
            Log.d("ReminderDialog", "Existing Reminder Time: " +
            existingReminder.getReminderTime());

            reminderMessageEditText.setText(existingReminder.getReminderMessage());
            reminderTimeTextView.setText(existingReminder.getReminderTime());
        }

        selectTimeButton.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v) {
                showTimePicker(reminderTimeTextView);
            }
        });

        builder.setPositiveButton("Save", new
        DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                String reminderMessage =
            reminderMessageEditText.getText().toString().trim();
                String reminderTime =
            reminderTimeTextView.getText().toString().trim();

                if (reminderMessage.isEmpty() &&
            reminderTime.isEmpty()) {
                    // Both message and time are empty, show a toast
                    message
                    Toast.makeText(context, "Please fill in either
            message or time field", Toast.LENGTH_SHORT).show();
                    return;
                }

                if (reminderMessage.isEmpty() ||
            reminderTime.isEmpty()) {
                    // Either message or time is empty, show a toast
                    message
                    Toast.makeText(context, "Please fill in both
            message and time fields", Toast.LENGTH_SHORT).show();
                    return;
                }

                if (habit.hasReminders()) {
                    // Update an existing reminder
                    Reminder existingReminder =
            habit.getReminders().get(0); // Assuming one reminder per habit
                    existingReminder.setReminderMessage(reminderMessage);
                }
            }
        });
    }
}
```



```
        existingReminder.setReminderTime(reminderTime);
    } else {
        // Habit doesn't have reminders, add a new one
        Reminder newReminder = new
Reminder(System.currentTimeMillis(), reminderTime, reminderMessage);
        habit.addReminder(newReminder);
    }

    // Schedule or update alarms for all reminders
    for (Reminder reminder : habit.getReminders()) {
        scheduleAlarm(context, habit.getId(),
reminder.getReminderTime(), reminder.getReminderMessage());
    }

    // Log the reminder details
    Log.d("ReminderDialog", "Reminder Message: " +
reminderMessage);
    Log.d("ReminderDialog", "Reminder Time: " +
reminderTime);

    // Show a different toast message if both message and
time are filled
    if (!habit.hasReminders()) {
        Toast.makeText(context, "Reminder added",
Toast.LENGTH_SHORT).show();
        Log.d("ReminderDialog", "Reminder added");
    }

    notifyDataSetChanged();
    dialog.dismiss();
});

    if (habit.hasReminders()) {
        builder.setNegativeButton("Remove", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which)
{
                // Show a confirmation dialog for removing the
reminder
                AlertDialog.Builder confirmBuilder = new
AlertDialog.Builder(context);
                confirmBuilder.setTitle("Confirm Removal");
                confirmBuilder.setMessage("Are you sure you want to
remove this reminder?");

                confirmBuilder.setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int
which) {
                        // Get the reminder to remove (assuming
only one reminder per habit)
                        Reminder reminderToRemove =
habit.getReminders().get(0);

                        // Remove the reminder from the habit
```



```
habit.getReminders().remove(reminderToRemove);

// Cancel the scheduled alarm for the
removed reminder
cancelScheduledAlarm(context,
habit.getId(), reminderToRemove.getReminderMessage());

// Log the removal
Log.d("ReminderDialog", "Removed Reminder:
" + reminderToRemove.getReminderMessage());

notifyDataSetChanged();
dialog.dismiss();
}
});

confirmBuilder.setNegativeButton("No", new
DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int
which) {
dialog.dismiss();
}
});

confirmBuilder.create().show();
});
}
builder.setNeutralButton("Cancel", new
DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
dialog.dismiss();
}
});

builder.create().show();
}

public void cancelScheduledAlarm(Context context, long habitId,
String reminderMessage) {
AlarmManager alarmManager = (AlarmManager)
context.getSystemService(Context.ALARM_SERVICE);
Intent alarmIntent = new Intent(context, AlarmReceiver.class);
alarmIntent.putExtra("habitId", habitId);
alarmIntent.putExtra("message", reminderMessage);
PendingIntent pendingIntent =
PendingIntent.getBroadcast(context, (int) habitId, alarmIntent,
PendingIntent.FLAG_UPDATE_CURRENT);

if (alarmManager != null) {
alarmManager.cancel(pendingIntent);
}
}
}
```



```
private void showTimePicker(final TextView reminderTimeTextView) {
    TimePickerDialog timePickerDialog = new TimePickerDialog(
        context,
        new TimePickerDialog.OnTimeSetListener() {
            @Override
            public void onTimeSet(TimePicker view, int
hourOfDay, int minute) {
                String selectedTime =
String.format("%02d:%02d", hourOfDay, minute);
                reminderTimeTextView.setText(selectedTime);
            }
        },
        Calendar.getInstance().get(Calendar.HOUR_OF_DAY),
        Calendar.getInstance().get(Calendar.MINUTE),
        true
    );

    timePickerDialog.show();
}

private void scheduleAlarm(Context context, long habitId, String
reminderTime, String reminderMessage) {
    AlarmManager alarmManager = (AlarmManager)
context.getSystemService(Context.ALARM_SERVICE);
    Intent alarmIntent = new Intent(context, AlarmReceiver.class);
    alarmIntent.putExtra("habitId", habitId);
    alarmIntent.putExtra("message", reminderMessage);
    PendingIntent pendingIntent =
PendingIntent.getBroadcast(context, (int) habitId, alarmIntent,
PendingIntent.FLAG_UPDATE_CURRENT);

    if (!reminderTime.isEmpty()) {
        String[] timeParts = reminderTime.split(":");
        if (timeParts.length == 2) {
            try {
                int hourOfDay = Integer.parseInt(timeParts[0]);
                int minute = Integer.parseInt(timeParts[1]);

                if (hourOfDay >= 0 && hourOfDay < 24 && minute >= 0
&& minute < 60) {
                    Calendar calendar = Calendar.getInstance();

calendar.setTimeInMillis(System.currentTimeMillis());
                    calendar.set(Calendar.HOUR_OF_DAY, hourOfDay);
                    calendar.set(Calendar.MINUTE, minute);
                    calendar.set(Calendar.SECOND, 0);

                    if (calendar.getTimeInMillis() <=
System.currentTimeMillis()) {
                        calendar.add(Calendar.DAY_OF_YEAR, 1);
                    }

                    if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.M) {
                        alarmManager.setExactAndAllowWhileIdle(AlarmManager.RTC_WAKEUP,
calendar.getTimeInMillis(), pendingIntent);
                    } else if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.KITKAT) {
```



```
alarmManager.setExact(AlarmManager.RTC_WAKEUP,
calendar.getTimeInMillis(), pendingIntent);
    } else {
        alarmManager.set(AlarmManager.RTC_WAKEUP,
calendar.getTimeInMillis(), pendingIntent);
    }
    } else {
        showToast(context, "Please select a valid time
between 00:00 and 23:59");
    }
    } catch (NumberFormatException e) {
        e.printStackTrace();
        Toast.makeText(context, "Error parsing time",
Toast.LENGTH_SHORT).show();
    }
    } else {
        showToast(context, "Invalid time format");
    }
    } else {
        showToast(context, "Please select a time for the
reminder");
    }
}

private void showToast(Context context, String message) {
    Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
}

// Inside the method saveReminders
public void saveReminders(Context context, long habitId,
List<Reminder> reminders) {
    SharedPreferences sharedPreferences =
context.getSharedPreferences("HabitPrefs_" + habitId,
Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();

    // Create a JSONArray to store reminders
    JSONArray remindersArray = new JSONArray();

    // Add each reminder to the JSONArray
    for (Reminder reminder : reminders) {
        JSONObject reminderJson = new JSONObject();
        try {
            reminderJson.put("id", reminder.getId());
            reminderJson.put("reminderTime",
reminder.getReminderTime());
            reminderJson.put("reminderMessage",
reminder.getReminderMessage());
            // Add other properties if needed
            remindersArray.put(reminderJson);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    // Save reminders as a JSON array in SharedPreferences
    editor.putString("reminders", remindersArray.toString());
}
```



```
        editor.apply();
    }

    // Inside the method loadReminders
    public List<Reminder> loadReminders(Context context, long habitId)
    {
        SharedPreferences sharedPreferences =
context.getSharedPreferences("HabitPrefs_" + habitId,
Context.MODE_PRIVATE);

        // Load reminders as a JSON array from SharedPreferences
        String remindersJsonArray =
sharedPreferences.getString("reminders", "");
        Gson gson = new Gson();
        Type type = new TypeToken<List<Reminder>>() {}.getType();

        // Parse JSON array to a list of Reminder objects
        List<Reminder> remindersList =
gson.fromJson(remindersJsonArray, type);

        return remindersList != null ? remindersList : new
ArrayList<>(); // Return an empty list if null
    }
}
```

Habit

```
package com.example.myapplicationforhabits;

import android.text.TextUtils;

import com.google.gson.annotations.SerializedName;

import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

public class Habit {
    @SerializedName("id")
    private long id;
    @SerializedName("name")
    private String name;
    @SerializedName("description")
    private String description;
    @SerializedName("reason")
    private String reason;
    @SerializedName("category")
    private String category; // If category is an object with more
fields, create a Category class
    @SerializedName("days_of_the_week")
    private Map<String, Boolean> days_of_the_week = new HashMap<>(); //
```



```
This will automatically map the days and their boolean values
@SerializedName("start_date")
private String startDay;
@SerializedName("end_date")
private String endDay;
@SerializedName("is_active")
private boolean active;
private List<String> days;
@SerializedName("is_complete")
private boolean completed;

private List<Reminder> reminders;

@SerializedName("is_archived")
private boolean isArchived;

public Habit(long id, String name, String description, String
reason, String category, Map<String, Boolean> days_of_the_week, String
startDay, String endDay, boolean active) {
    this.id = id;
    this.name = name;
    this.description = description;
    this.reason = reason;
    this.category = category;
    this.days_of_the_week = days_of_the_week;
    this.startDay = startDay;
    this.endDay = endDay;
    this.active = active;
    this.reminders = new ArrayList<>();
    this.completed = false;
}

public Habit(long id, String name, Map<String, Boolean>
days_of_the_week, boolean completed) {
    this.id = id;
    this.name = name;
    this.days_of_the_week = days_of_the_week;
    this.completed = completed;
}

public Habit(long id, String name, Map<String, Boolean>
days_of_the_week, boolean completed, boolean active) {
    this.id = id;
    this.name = name;
    this.days_of_the_week = days_of_the_week;
    this.completed = completed;
    this.active = active;
}

public Habit(String name, String description, String reason,
Map<String, Boolean> days_of_the_week, String startDay, String endDay)
{
    this.name = name;
    this.description = description;
    this.reason = reason;
    this.days_of_the_week = days_of_the_week;
    this.startDay = startDay;
```



```
        this.endDay = endDay;
        this.active = true;
    }

    public boolean isArchived() {
        return isArchived;
    }

    public void setReminders(List<Reminder> reminders) {
        this.reminders = reminders;
    }

    public List<Reminder> getReminders() {
        return reminders;
    }

    public void addReminder(Reminder reminder) {
        if (reminders == null) {
            reminders = new ArrayList<>();
        }
        reminders.add(reminder);
    }

    public boolean hasReminders() {
        return reminders != null && !reminders.isEmpty();
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getReason() {
        return reason;
    }

    public void setReason(String reason) {
        this.reason = reason;
    }
}
```



```
}

public String getCategory() {
    return category;
}

public void setCategory(String category) {
    this.category = category;
}

public Map<String, Boolean> days_of_the_week() {
    return days_of_the_week;
}

public void days_of_the_week(Map<String, Boolean> daysOfTheWeek) {
    this.days_of_the_week = daysOfTheWeek;
}

public String getStartDay() {
    return startDay;
}

public void setStartDay(String startDay) {
    this.startDay = startDay;
}

public String getEndDay() {
    return endDay;
}

public void setEndDay(String endDay) {
    this.endDay = endDay;
}

public boolean isActive() {
    return active;
}

public void setActive(boolean active) {
    this.active = active;
}

public boolean isCompleted() {
    return completed;
}

public void setCompleted(boolean completed) {
    this.completed = completed;
}

public List<String> getDays() {
    return days;
}

public void setDays(List<String> days) {
    this.days = days;
}

public void setDayChecked(String day, boolean isChecked) {
    if (this.days_of_the_week == null) {
```



```
        this.days_of_the_week = new HashMap<>(); // Initialize if
null
    }
    this.days_of_the_week.put(day, isChecked); // Set the day's
checked status
    }

    @Override
    public String toString() {
        return "Habit{" +
            "id=" + id +
            ", name='" + name + '\'' +
            ", description='" + description + '\'' +
            ", reason='" + reason + '\'' +
            ", category='" + category + '\'' +
            ", days_of_the_week=" + days_of_the_week +
            ", startDay='" + startDay + '\'' +
            ", endDay='" + endDay + '\'' +
            ", isActive=" + active +
            ", isArchived=" + isArchived +
            "reminders=" + reminders +
            '}';
    }

    public String getDaysAsString() {
        // Check if 'days_of_the_week' is not null before proceeding
        if (this.days_of_the_week == null) {
            return "No days set"; // or handle it accordingly
        } else {
            // Filter and collect active days into a list
            List<String> activeDays = new ArrayList<>();
            for (Map.Entry<String, Boolean> entry :
this.days_of_the_week.entrySet()) {
                if (entry.getValue()) { // if the day is marked as
true/active, add it to the list
                    activeDays.add(entry.getKey());
                }
            }
            // Join active days into a single string separated by
commas
            return TextUtils.join(", ", activeDays);
        }
    }

    public void setDaysOfWeek(Map<String, Boolean> days_of_the_week) {
        // Create a JSONObject to store the days_of_the_week
        JSONObject daysOfWeekJson = new JSONObject();

        // Iterate through the map and add each entry to the JSONObject
        for (Map.Entry<String, Boolean> entry :
days_of_the_week.entrySet()) {
            try {
                daysOfWeekJson.put(entry.getKey(), entry.getValue());
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }

    public JSONObject getDaysOfWeek() {
```



```
// Create a JSONObject to store the days_of_the_week
JSONObject daysOfWeekJson = new JSONObject();

// Iterate through the map and add each entry to the JSONObject
for (Map.Entry<String, Boolean> entry :
this.days_of_the_week.entrySet()) {
    try {
        daysOfWeekJson.put(entry.getKey(), entry.getValue());
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

return daysOfWeekJson;
}

Map<String, Boolean> jsonObjectToMap(JSONObject jsonObject) {
    Map<String, Boolean> map = new HashMap<>();
    Iterator<String> keys = jsonObject.keys();
    while (keys.hasNext()) {
        String key = keys.next();
        try {
            boolean value = jsonObject.getBoolean(key);
            map.put(key, value);
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
    return map;
}

public Map<String, Boolean> getDaysOfWeekMap() {
    return days_of_the_week;
}
}
```

UserAuthActivity

```
package com.example.myapplicationforhabits;

import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.text.method.HideReturnsTransformationMethod;
import android.text.method.PasswordTransformationMethod;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
```



```
public class UserAuthActivity extends AppCompatActivity {

    private EditText emailEditText;
    private EditText passwordEditText;
    private Button signUpButton;
    private Button signInButton;
    private CheckBox rememberMeCheckBox;
    private ImageView showPasswordImageView;

    private ApiService apiService;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_auth);

        // Initialize the ApiService
        apiService =
RetrofitClient.getClient().create(ApiService.class);

        emailEditText = findViewById(R.id.emailEditText);
        passwordEditText = findViewById(R.id.passwordEditText);
        signUpButton = findViewById(R.id.signUpButton);
        signInButton = findViewById(R.id.signInButton);
        rememberMeCheckBox = findViewById(R.id.rememberMeCheckBox);

        // Set the text of the sign-in button
        signInButton.setText("Login");

        // Retrieve remembered credentials and fill the corresponding
EditText fields
        retrieveRememberedCredentials();

        // Add an OnCheckedChangeListener to the "Remember Me" CheckBox
rememberMeCheckBox.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
                handleRememberMeCheckedChanged(isChecked);
            }
        });

        signUpButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                signUpUser();
            }
        });

        signInButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                signInUser();
            }
        });

        showPasswordImageView =
findViewById(R.id.showPasswordImageView);
    }
}
```



```
        showPasswordImageView.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                togglePasswordVisibility();
            }
        });
    }

    private void togglePasswordVisibility() {
        // Toggle the password visibility based on the current state
        if (passwordEditText.getTransformationMethod() ==
PasswordTransformationMethod.getInstance()) {

passwordEditText.setTransformationMethod(HideReturnsTransformationMetho
d.getInstance());

showPasswordImageView.setImageResource(R.drawable.ic_eye_open);
        } else {

passwordEditText.setTransformationMethod(PasswordTransformationMethod.g
etInstance());

showPasswordImageView.setImageResource(R.drawable.ic_eye_closed);
        }

        // Move the cursor to the end of the text
passwordEditText.setSelection(passwordEditText.getText().length());
    }

    private void signUpUser() {
        // Navigate to the sign-up XML (activity)
        Intent intent = new Intent(UserAuthActivity.this,
SignUpActivity.class);
        startActivity(intent);
    }

    private void signInUser() {
        String email = emailEditText.getText().toString().trim();
        String password = passwordEditText.getText().toString().trim();

        if (email.isEmpty() || password.isEmpty()) {
            showToast("Please fill in all fields");
            return;
        }

        if (!isEmailValid(email)) {
            showToast("Invalid email format. Email must contain '@' and
'.' characters");
            return;
        }

        // This UserLogin model should match what your server expects
for authentication
        UserLogin loginRequest = new UserLogin(email, password);

        // Retrofit network call to authenticate the user
```



```
        ApiService.authenticateUser(loginRequest).enqueue(new
Callback<JwtResponse>() {
    @Override
    public void onResponse(Call<JwtResponse> call,
Response<JwtResponse> response) {
        if (response.isSuccessful() && response.body() != null)
    {
        JwtResponse jwtResponse = response.body();
        String accessToken = jwtResponse.getAccess();

        // Save the token
        saveAuthToken(accessToken);

        // Save the user's credentials if "Remember Me" is
checked
        saveRememberedCredentials(email, password);

        showToast("Login successful!");
        // Fetch the habits
        navigateToMainActivity();
    } else {
        if (response.code() == 401) {
            showToast("Incorrect email/password or user
does not exist.");
        } else {
            showToast("Authentication failed: " +
response.code());
        }
    }
}

    @Override
    public void onFailure(Call<JwtResponse> call, Throwable t)
    {
        showToast("Network error: " + t.getMessage());
    }
});
}

private boolean isValidEmail(String email) {
    String emailPattern = "[A-Za-z0-9+.-]+@(.+)$";
    return email.matches(emailPattern);
}

private void showToast(String message) {
    Toast.makeText(UserAuthActivity.this, message,
Toast.LENGTH_SHORT).show();
}

private void saveAuthToken(String token) {
    SharedPreferences sharedPreferences =
getSharedPreferences("UserPrefs", MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString("accessToken", token);
    editor.apply();
}

private void saveRememberedCredentials(String email, String
password) {
    if (rememberMeCheckBox.isChecked()) {
```



```
        SharedPreferences sharedPreferences =
getSharedPreferences("UserPrefs", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString("email", email);
        editor.putString("password", password);
        editor.apply();
    }
}

private void retrieveRememberedCredentials() {
    SharedPreferences sharedPreferences =
getSharedPreferences("UserPrefs", MODE_PRIVATE);
    String rememberedEmail = sharedPreferences.getString("email",
    "");
    String rememberedPassword =
sharedPreferences.getString("password", "");
    boolean isRememberedChecked = !rememberedEmail.isEmpty() &&
!rememberedPassword.isEmpty();

    // Fill the EditText fields with remembered credentials
    emailEditText.setText(rememberedEmail);
    passwordEditText.setText(rememberedPassword);

    // Set the state of rememberMeCheckBox
    rememberMeCheckBox.setChecked(isRememberedChecked);
}

private void handleRememberMeCheckedChanged(boolean isChecked) {
    if (!isChecked) {
        // If "Remember Me" is unchecked, clear the stored
credentials
        clearRememberedCredentials();
    }
}

private void clearRememberedCredentials() {
    SharedPreferences sharedPreferences =
getSharedPreferences("UserPrefs", MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.remove("email");
    editor.remove("password");
    editor.apply();
}

private void navigateToMainActivity() {
    Intent intent = new Intent(UserAuthActivity.this,
MainActivity.class);
    startActivity(intent);
    finish();
}
}
```

SignUpActivity

```
package com.example.myapplicationforhabits;

import android.content.Intent;
import android.os.Bundle;
```



```
import android.text.method.HideReturnsTransformationMethod;
import android.text.method.PasswordTransformationMethod;
import android.text.method.TransformationMethod;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class SignUpActivity extends AppCompatActivity {

    private EditText emailEditText;
    private EditText passwordEditText;
    private EditText confirmPasswordEditText;
    private CheckBox showPasswordCheckBox;
    private Button signUpButton;
    private ProgressBar progressBar;

    private static final int MIN_PASSWORD_LENGTH = 6;

    private ApiService apiService;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.sign_up);

        emailEditText = findViewById(R.id.emailEditText);
        passwordEditText = findViewById(R.id.passwordEditText);
        confirmPasswordEditText =
findViewById(R.id.confirmPasswordEditText);
        showPasswordCheckBox = findViewById(R.id.showPasswordCheckBox);
        signUpButton = findViewById(R.id.signUpButton);
        progressBar = findViewById(R.id.progressBar);

        // Initialize the ApiService
        apiService =
RetrofitClient.getClient().create(ApiService.class);

        signUpButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Get user input
                String email = emailEditText.getText().toString();
                String password =
passwordEditText.getText().toString();
                String confirmPassword =
confirmPasswordEditText.getText().toString();

                // Validate input
                if (email.isEmpty() || password.isEmpty() ||
confirmPassword.isEmpty()) {
                    showToast("Please fill in all fields.");
                }
            }
        });
    }
}
```



```
        return;
    }

    if (!isEmailValid(email)) {
        showToast("Invalid email format. Email must contain
'@' and '.' characters.");
        return;
    }

    if (password.length() < MIN_PASSWORD_LENGTH) {
        showToast("Password must be at least 6 characters
long.");
        return;
    }

    if (!password.equals(confirmPassword)) {
        showToast("Passwords do not match.");
        return;
    }

    // If all checks pass, you can proceed with the sign-up
logic

    // Show progress bar
    progressBar.setVisibility(View.VISIBLE);

    // Create a UserRegistration model with email and
password
    SignUpRequest userRegistration = new
SignUpRequest(email, password);

    // Make a POST request to your API for user
registration
    apiService.registerUser(userRegistration).enqueue(new
Callback<Void>() {
        @Override
        public void onResponse(Call<Void> call,
Response<Void> response) {
            progressBar.setVisibility(View.GONE);
            if (response.code() == 201) {
                // Handle successful sign-up when the HTTP
status code is 201 Created
                showToast("Sign up successful!");

                // Navigate to the sign-up XML (activity)
after successful sign-up
                Intent intent = new
Intent(SignUpActivity.this, UserAuthActivity.class);
                startActivity(intent);
            } else {
                showToast("Sign up failed: Account already
exists, " + response.code());
            }
        }

        @Override
        public void onFailure(Call<Void> call, Throwable t)
{
            progressBar.setVisibility(View.GONE);
            showToast("Network error: " + t.getMessage());
        }
    });
}
```



```
        });
    }
});

// Add an OnCheckedChangeListener to the "Show Password"
CheckBox
showPasswordCheckBox.setOnCheckedChangeListener((buttonView,
isChecked) -> {
    // Toggle the password visibility based on the checkbox
state
    TransformationMethod transformationMethod = isChecked ?
        HideReturnsTransformationMethod.getInstance() :
        PasswordTransformationMethod.getInstance();

passwordEditText.setTransformationMethod(transformationMethod);

confirmPasswordEditText.setTransformationMethod(transformationMethod);

    // Move the cursor to the end of the text

passwordEditText.setSelection(passwordEditText.getText().length());

confirmPasswordEditText.setSelection(confirmPasswordEditText.getText().
length());
});

}

// Define the showToast method
private void showToast(String message) {
    Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
}

// Define the isValidEmail method
private boolean isValidEmail(String email) {
    String emailPattern = "[A-Za-z0-9+_.-]+@[A-Za-z0-9.-
]+\\. (com|org|net|edu|gov|mil|biz|info|name|museum|coop|aero|asia|cat|j
obs|mobi|tel|travel|arpa|int|eu|us|uk|gr|fr|de|jp|cn|in|br|ru|au|ca|it)
$";
    return email.matches(emailPattern);
}

}
```

Reminder

```
package com.example.myapplicationforhabits;

public class Reminder {
    private long id;
    private String reminderTime;
    private String reminderMessage;

    public Reminder(long id, String reminderTime, String
reminderMessage) {
        this.id = id;
        this.reminderTime = reminderTime;
        this.reminderMessage = reminderMessage;
    }
}
```



```
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getReminderTime() {
        return reminderTime;
    }

    public void setReminderTime(String reminderTime) {
        this.reminderTime = reminderTime;
    }

    public String getReminderMessage() {
        return reminderMessage;
    }

    public void setReminderMessage(String reminderMessage) {
        this.reminderMessage = reminderMessage;
    }

    @Override
    public String toString() {
        return "Reminder{" +
            "id=" + id +
            ", reminderTime='" + reminderTime + '\'' +
            ", reminderMessage='" + reminderMessage + '\'' +
            '}';
    }
}
```

AlarmReceiver

```
package com.example.myapplicationforhabits;

import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.Build;
import android.widget.Toast;

import androidx.core.app.NotificationCompat;

public class AlarmReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        // Get the reminder message and habit ID from the intent
        String reminderMessage = intent.getStringExtra("message");
        long habitId = intent.getLongExtra("habitId", -1);
    }
}
```



```
// Create a notification channel (for Android 8.0 and higher)
createNotificationChannel(context);

// Create an intent to open the app when the notification is
clicked
Intent openAppIntent = new Intent(context, MainActivity.class);
PendingIntent pendingIntent =
PendingIntent.getActivity(context, 0, openAppIntent,
PendingIntent.FLAG_IMMUTABLE);

// Create a vibration pattern (you can adjust this pattern as
needed)
long[] vibrationPattern = {0, 500, 1000, 500};

// Build the notification
NotificationCompat.Builder builder = new
NotificationCompat.Builder(context, "habit_reminders")
    .setSmallIcon(R.mipmap.ic_launcher) // Change this to
your app's icon
    .setContentTitle("Habit Reminder")
    .setContentText(reminderMessage)
    .setContentIntent(pendingIntent)
    .setAutoCancel(true)
    .setVibrate(vibrationPattern) // Corrected method for
setting vibration
    .setSound(getNotificationSound()) // Set notification
sound
    .setPriority(NotificationCompat.PRIORITY_DEFAULT);

// Display the notification
NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify((int) habitId, builder.build());

// You can also show a Toast message as a simple way to verify
that the alarm went off
Toast.makeText(context, "Reminder: " + reminderMessage,
Toast.LENGTH_SHORT).show();
}

// Create a notification channel (for Android 8.0 and higher)
private void createNotificationChannel(Context context) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        CharSequence name = "HabitReminders";
        String description = "Habit reminder notifications";
        int importance = NotificationManager.IMPORTANCE_DEFAULT;
        NotificationChannel channel = new
NotificationChannel("habit_reminders", name, importance);
        channel.setDescription(description);

        NotificationManager notificationManager =
context.getSystemService(NotificationManager.class);
        notificationManager.createNotificationChannel(channel);
    }
}

// Get the default notification sound
private Uri getNotificationSound() {
    return
```



```
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);  
    }  
}
```

RetrofitClient

```
package com.example.myapplicationforhabits;  
  
import okhttp3.OkHttpClient;  
import okhttp3.logging.HttpLoggingInterceptor;  
import retrofit2.Retrofit;  
import retrofit2.converter.gson.GsonConverterFactory;  
  
public class RetrofitClient {  
  
    private static Retrofit retrofit = null;  
    private static final String BASE_URL =  
"http://tracker.foivoskournoutos.com";  
  
    public static Retrofit getClient() {  
        if (retrofit == null) {  
            // Create a logging interceptor  
            HttpLoggingInterceptor loggingInterceptor = new  
HttpLoggingInterceptor();  
  
loggingInterceptor.setLevel(HttpLoggingInterceptor.Level.BODY);  
  
            // Build the OkHttpClient with the logging interceptor  
            OkHttpClient okHttpClient = new OkHttpClient.Builder()  
                .addInterceptor(loggingInterceptor)  
                .build();  
  
            // Initialize the Retrofit instance with the OkHttpClient  
            retrofit = new Retrofit.Builder()  
                .baseUrl(BASE_URL)  
                .addConverterFactory(GsonConverterFactory.create())  
                .client(okHttpClient)  
                .build();  
        }  
        return retrofit;  
    }  
}
```

ApiService

```
package com.example.myapplicationforhabits;  
  
import java.util.List;  
  
import retrofit2.Call;  
import retrofit2.http.Body;  
import retrofit2.http.DELETE;  
import retrofit2.http.GET;  
import retrofit2.http.Header;  
import retrofit2.http.POST;  
import retrofit2.http.PUT;  
import retrofit2.http.Path;
```



```
public interface ApiService {
    // Endpoint to authenticate user and retrieve JWT token
    @POST("/api/token/")
    Call<JwtResponse> authenticateUser(@Body UserLogin userLogin);

    // Endpoint to sign up a new user
    @POST("/signup/")
    Call<Void> registerUser(@Body SignUpRequest signUpRequest);

    // Endpoint to get the list of habits for the user
    @GET("/api/v1/habits/")
    Call<List<Habit>> getUserHabits(@Header("Authorization") String
authToken);

    // Endpoint to post a new habit
    @POST("/api/v1/habits/")
    Call<Habit> createHabit(@Header("Authorization") String authToken,
@Body Habit habit);

    // Endpoint to update an existing habit
    @PUT("/api/v1/habits/{id}")
    Call<Habit> updateHabit(@Path("id") int habitId,
@Header("Authorization") String authToken, @Body Habit habit);

    // Endpoint to delete a habit
    @DELETE("/api/v1/habits/{id}")
    Call<Void> deleteHabit(@Path("id") int habitId,
@Header("Authorization") String authToken);
}

// Model used to post login details
class UserLogin {
    private String username;
    private String password;

    // Constructor
    public UserLogin(String username, String password) {
        this.username = username;
        this.password = password;
    }

    // Getters
    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    // Setters
    public void setUsername(String username) {
        this.username = username;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

```
}

// Model for JWT Response
class JwtResponse {
    private String access;
    private String refresh;

    // Constructor
    public JwtResponse(String access, String refresh) {
        this.access = access;
        this.refresh = refresh;
    }

    // Getters
    public String getAccess() {
        return access;
    }

    public String getRefresh() {
        return refresh;
    }

    // Setters (if necessary for your implementation)
    public void setAccess(String access) {
        this.access = access;
    }

    public void setRefresh(String refresh) {
        this.refresh = refresh;
    }
}

class SignUpRequest {
    private String username;
    private String password;

    public SignUpRequest(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    // Setters
    public void setUsername(String username) {
        this.username = username;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}

class ApiResponse {
```



```
private boolean success;
private String message;

public boolean isSuccess() {
    return success;
}

public String getMessage() {
    return message;
}
}
```

activity_intro.xml

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#b7e4b2"
    tools:context=".IntroActivity">

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/Base.Theme.MyApplicationForHabits">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:background="@color/colorPrimary"
            android:layout_height="?attr/actionBarSize"
            app:title="@string/my_habit_tracker_app" />

    </com.google.android.material.appbar.AppBarLayout>

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/viewPager2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_above="@+id/horizontal_bar" />

    <!-- FrameLayout to contain horizontal bar and WormDotsIndicator -->
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true">

        <!-- Horizontal bar at the bottom -->
        <View
            android:id="@+id/horizontal_bar"
            android:layout_width="match_parent"
            android:layout_height="50dp"
            android:background="@color/colorPrimary" />
    </FrameLayout>
</RelativeLayout>
```



```
<!-- WormDotsIndicator inside horizontal bar -->
<com.tbunomo.viewpagerdotsindicator.WormDotsIndicator
    android:id="@+id/wormDotsIndicator"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_above="@id/horizontal_bar"
    app:dotsColor="@color/black"
    app:dotsStrokeColor="@color/black"
    app:dotsStrokeWidth="2dp" />
</FrameLayout>

<!-- Button above horizontal bar -->
<Button
    android:id="@+id/btn_start_app"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:background="@drawable/round_button"
    android:text="@string/start_adding_habits"
    android:textSize="18sp"
    android:backgroundTint="#58A76B"
    android:alpha="0"
    android:visibility="gone"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="70dp" />
</RelativeLayout>
```

activity_mail.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#DCEBC5"
    tools:context=".MainActivity">

    <!-- Add a Toolbar or ActionBar -->
    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="@color/colorPrimary"
        app:title="My Habits"
        app:titleTextColor="#dbfaee"
        app:titleTextAppearance="@style/ToolbarTitle"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/habitRecyclerView"
```



```
android:layout_width="match_parent"
android:layout_height="0dp"
app:layout_constraintBottom_toTopOf="@+id/addHabitButton"
app:layout_constraintTop_toBottomOf="@+id/toolbar"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintEnd_toEndOf="parent"
android:scrollbars="vertical"
android:padding="10dp" />
```

<Button

```
android:id="@+id/addHabitButton"
android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:text="Add Habit"
android:textSize="18sp"
android:background="@drawable/round_button"
android:backgroundTint="#58A76B"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
android:layout_marginBottom="16dp" />
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_user_auth.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:background="#f0cc92"
    tools:context=".UserAuthActivity">

    <ImageView
        android:id="@+id/appLogoImageView"
        android:layout_width="300dp"
        android:layout_height="150dp"
        android:src="@drawable/ic_user"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="16dp" />

    <EditText
        android:id="@+id/emailEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Email"
        android:inputType="textEmailAddress"
        android:layout_below="@+id/appLogoImageView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="16dp" />

    <LinearLayout
        android:id="@+id/passwordContainer"
        android:layout_width="match_parent"
```

```
android:layout_height="wrap_content"
android:orientation="horizontal"
android:layout_below="@+id/emailEditText"
android:layout_centerHorizontal="true"
android:layout_marginTop="8dp"
android:layout_marginBottom="8dp">

<EditText
    android:id="@+id/passwordEditText"
    android:layout_width="0dp"
    android:layout_weight="1"
    android:layout_height="wrap_content"
    android:hint="Password"
    android:inputType="textPassword"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp" />

<ImageView
    android:id="@+id/showPasswordImageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_eye_closed"
    android:layout_gravity="center_vertical|end"
    android:clickable="true" />
</LinearLayout>

<CheckBox
    android:id="@+id/rememberMeCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/passwordContainer"
    android:text="Remember Me"
    android:layout_marginTop="16dp"/>

<Button
    android:id="@+id/signInButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Log In"
    android:backgroundTint="#66b5ce"
    android:textSize="18sp"
    android:layout_below="@+id/rememberMeCheckBox"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="52dp" />

<Button
    android:id="@+id/signUpButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:text="Sign Up"
    android:backgroundTint="#5e033c"
    android:layout_below="@+id/signInButton"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp" />

<ProgressBar
    android:id="@+id/progressBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```



```
android:layout_centerInParent="true"  
android:visibility="gone" />
```

```
</RelativeLayout>
```

dialog_add_habit.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    <LinearLayout  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:orientation="vertical"  
        android:background="#e1ed7c"  
        android:gravity="center">  
  
        <!-- Habit Image -->  
        <ImageView  
            android:id="@+id/habitNameImageView"  
            android:layout_width="160dp"  
            android:layout_height="140dp"  
            android:layout_marginTop="20dp"  
  
            android:contentDescription="@string/create_habit_select_image"  
            android:src="@drawable/ic_habit1" />  
  
        <!-- Habit Name EditText -->  
        <EditText  
            android:id="@+id/habitNameEditText"  
            android:layout_width="match_parent"  
            android:layout_height="wrap_content"  
            android:layout_marginTop="16dp"  
            android:hint="@string/create_habit_title_hint"  
            android:inputType="text"  
            android:textColor="@android:color/black" />  
  
        <!-- Habit Description EditText -->  
        <EditText  
            android:id="@+id/habitDescriptionEditText"  
            android:layout_width="match_parent"  
            android:layout_height="wrap_content"  
            android:layout_marginTop="16dp"  
            android:hint="@string/create_habit_description_hint"  
            android:inputType="text"  
            android:textColor="@android:color/black" />  
  
        <!-- Habit Reason EditText -->  
        <EditText  
            android:id="@+id/reasonEditText"  
            android:layout_width="match_parent"  
            android:layout_height="wrap_content"  
            android:layout_marginTop="16dp"  
            android:hint="@string/create_habit_reason_hint"  
            android:inputType="text"  
            android:textColor="@android:color/black" />
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_marginTop="10dp">

    <!-- Start Day TextView -->
    <TextView
        android:id="@+id/startDayTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/create_habit_start_day"
        android:textColor="@android:color/black"
        android:textSize="14sp" />

    <!-- Start Day Button -->
    <Button
        android:id="@+id/dayStartButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="#FF03DAC5"
        android:text="@string/create_habit_pick_start_day"
        android:textSize="12sp"
        android:textColor="@android:color/black" />

    <!-- End Day EditText -->
    <TextView
        android:id="@+id/endDayTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="@string/create_habit_end_day"
        android:textColor="@android:color/black"
        android:textSize="14sp" />

    <!-- End Day Button -->
    <Button
        android:id="@+id/dayEndButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="#FF03DAC5"
        android:text="@string/create_habit_pick_end_day"
        android:textSize="12sp"
        android:textColor="@android:color/black" />

</LinearLayout>
<TextView
    android:id="@+id/daysTextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:hint="@string/dayOfWeek"
    android:textColor="@android:color/black"
    android:textSize="14sp" />

<GridLayout
    android:layout_width="wrap_content"
```



```
android:layout_height="wrap_content"
android:layout_gravity="center"
android:columnCount="4"
android:rowCount="2"
android:layout_marginTop="8dp">

<!-- Checkbox for Monday -->
<CheckBox
    android:id="@+id/checkboxMonday"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@android:color/black"
    android:textSize="16sp"
    android:buttonTint="@color/red_orange"
    android:text="Mon" />

<!-- Checkbox for Tuesday -->
<CheckBox
    android:id="@+id/checkboxTuesday"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@android:color/black"
    android:textSize="16sp"
    android:buttonTint="@color/red_orange"
    android:text="Tue" />

<!-- Checkbox for Wednesday -->
<CheckBox
    android:id="@+id/checkboxWednesday"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@android:color/black"
    android:textSize="16sp"
    android:buttonTint="@color/red_orange"
    android:text="Wed" />

<!-- Checkbox for Thursday -->
<CheckBox
    android:id="@+id/checkboxThursday"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@android:color/black"
    android:textSize="16sp"
    android:buttonTint="@color/red_orange"
    android:text="Thu" />

<!-- Checkbox for Friday -->
<CheckBox
    android:id="@+id/checkboxFriday"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@android:color/black"
    android:textSize="16sp"
    android:buttonTint="@color/red_orange"
    android:text="Fri" />

<!-- Checkbox for Saturday -->
<CheckBox
    android:id="@+id/checkboxSaturday"
    android:layout_width="wrap_content"
```



```
        android:layout_height="wrap_content"
        android:textColor="@android:color/black"
        android:textSize="16sp"
        android:buttonTint="@color/red_orange"
        android:text="Sat" />

    <!-- Checkbox for Sunday -->
    <CheckBox
        android:id="@+id/checkboxSunday"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@android:color/black"
        android:textSize="16sp"
        android:buttonTint="@color/red_orange"
        android:text="Sun" />
</GridLayout>

</LinearLayout>
</ScrollView>
```

dialog_add_reminder.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:background="#ffcfaf"
    android:padding="16dp">

    <!-- TextView for displaying the selected time -->
    <TextView
        android:id="@+id/reminderTimeTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Selected Time: "
        android:textSize="18sp"
        android:textColor="#000000"
        android:layout_marginBottom="16dp" />

    <EditText
        android:id="@+id/reminderMessageEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Reminder Message"
        android:inputType="text"
        android:layout_marginBottom="16dp" />

    <!-- Button for selecting time -->
    <Button
        android:id="@+id/selectTimeButton"
        android:layout_width="200dp"
        android:layout_height="45dp"
        android:text="Select Time"
        android:background="@drawable/custom_button_background"
        android:textColor="#FFFFFF"
        android:textSize="18sp"
```



```
        android:layout_marginBottom="16dp"
        android:layout_gravity="center_horizontal" />
```

```
</LinearLayout>
```

intro_item_page.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="230dp"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:gravity="center"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/iv_image_intro"
            android:layout_width="170dp"
            android:layout_height="170dp"
            android:layout_centerInParent="true"
            android:contentDescription="Intro Image"
            android:src="@drawable/ic_habits" />

        <TextView
            android:id="@+id/tv_description_intro"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/iv_image_intro"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="14dp"
            android:text="My Habit Tracker App"
            android:textColor="@android:color/black"
            android:textSize="18sp" />

    </LinearLayout>
</RelativeLayout>
```

item_habit.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="4dp"
    android:layout_marginBottom="4dp"
    android:layout_marginStart="4dp"
    android:layout_marginEnd="4dp"
    android:foreground="?android:attr/selectableItemBackground"
```

```
android:elevation="4dp"
app:cardCornerRadius="12dp">

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:background="@drawable/rounded_background"
    android:padding="16dp">

    <!-- Habit Name TextView -->
    <ImageButton
        android:id="@+id/bellIcon"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end"
        android:background="#d3f9ea"
        android:layout_marginBottom="4dp"
        android:contentDescription="Add Reminder"
        android:src="@drawable/ic_bell1" />

    <!-- Habit Name TextView -->
    <TextView
        android:id="@+id/habitNameItemTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="14dp"
        android:text="Title:"
        android:textColor="#333333"
        android:layout_marginLeft="90dp"
        android:textSize="20sp"
        android:textStyle="bold" />

    <!-- Additional details initially hidden -->
    <LinearLayout
        android:id="@+id/additionalDetailsLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:visibility="gone">

        <!-- Habit Description TextView -->
        <TextView
            android:id="@+id/habitDescriptionItemTextView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Description:"
            android:textSize="16sp"
            android:textColor="#666666"
            android:layout_marginLeft="90dp"
            android:layout_marginBottom="4dp" />

        <!-- Reason TextView -->
        <TextView
            android:id="@+id/reasonItemTextView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="14sp"
            android:text="Reason:"
```



```
        android:textColor="#666666"
        android:layout_marginLeft="90dp"
        android:layout_marginTop="4dp" />

<!-- Start Day TextView -->
<TextView
    android:id="@+id/startDayItemTextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="14sp"
    android:text="Start Day:"
    android:textColor="#05627f"
    android:layout_marginLeft="90dp"
    android:layout_marginTop="4dp" />

<!-- End Day TextView -->
<TextView
    android:id="@+id/endDayItemTextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="14sp"
    android:text="End Day:"
    android:textColor="#057F5F"
    android:layout_marginLeft="90dp"
    android:layout_marginTop="4dp" />

<!-- Selected Days TextView -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginLeft="90dp"
    android:layout_marginTop="4dp">

    <TextView
        android:id="@+id/selectedDaysTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Days of the Week:"
        android:textSize="14sp"
        android:visibility="visible"
        android:textColor="#FF5722" />
</LinearLayout>

<!-- CheckBox for marking habits as completed -->
<CheckBox
    android:id="@+id/checkboxCompleted"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Completed"
    android:textSize="12sp"
    android:textColor="#333333"
    android:layout_marginTop="6dp" />

<CheckBox
    android:id="@+id/checkboxActive"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Active"
```



```
        android:textSize="12sp"
        android:textColor="#333333"
        android:layout_marginTop="6dp"/>

    </LinearLayout>

    <!-- Clickable area to toggle visibility of additional details -->
-->
    <RelativeLayout
        android:id="@+id/clickableLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <!-- Text and icon to indicate expand/collapse -->
        <TextView
            android:id="@+id/expandCollapseTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Show More"
            android:textColor="#1976D2"
            android:textSize="16sp"
            android:layout_alignParentEnd="true" />

    </RelativeLayout>

</LinearLayout>
</androidx.cardview.widget.CardView>
```

menu_switch_layout.xml

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <androidx.appcompat.widget.SwitchCompat
        android:id="@+id/action_switcher"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:checked="true"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

sign_up.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">
```

```
android:background="#f0cc92"
tools:context=".SignUpActivity">

<ImageView
    android:id="@+id/appLogoImageView"
    android:layout_width="300dp"
    android:layout_height="150dp"
    android:src="@drawable/ic_signup"
    android:layout_marginTop="10dp"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="16dp" />

<EditText
    android:id="@+id/emailEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Email"
    android:inputType="textEmailAddress"
    android:layout_below="@+id/appLogoImageView"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="16dp"
    android:layout_marginBottom="16dp" />

<EditText
    android:id="@+id/passwordEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Password"
    android:inputType="textPassword"
    android:layout_below="@+id/emailEditText"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dp"
    android:layout_marginBottom="16dp" />

<EditText
    android:id="@+id/confirmPasswordEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Confirm Password"
    android:inputType="textPassword"
    android:layout_below="@+id/passwordEditText"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dp"
    android:layout_marginBottom="16dp" />

<CheckBox
    android:id="@+id/showPasswordCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Show Password"
    android:layout_below="@+id/confirmPasswordEditText"
    android:layout_marginTop="2dp" />

<Button
    android:id="@+id/signUpButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:text="Create Account"
    android:backgroundTint="#5e033c"
```



```
        android:layout_below="@+id/showPasswordCheckBox"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp" />

        <ProgressBar
            android:id="@+id/progressBar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:visibility="gone" />

    </RelativeLayout>
```

menu_main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <item
        android:id="@+id/action_view_archived"
        android:title="Switch"
        app:actionLayout="@layout/menu_switch_layout"
        app:showAsAction="ifRoom|withText" />

    <item
        android:id="@+id/action_logout"
        android:icon="@drawable/ic_logout"
        android:title="Log Out"
        app:showAsAction="ifRoom|withText" />

    <item
        android:id="@+id/nav_exit"
        android:icon="@drawable/ic_exit"
        android:title="Exit"
        app:showAsAction="ifRoom|withText" />

</menu>
```



Παράρτημα II (Back End)

requirements.txt

```
asgiref==3.7.2
Django==4.2
sqlparse==0.4.4
typing_extensions==4.8.0
psycopy2-binary==2.9.6
django-environ==0.11.2
django-debug-toolbar==4.2.0
djangorestframework==3.14.0
djangorestframework-simplejwt==5.3.0
```

admin.py

```
from django.contrib import admin
from .models import HabitRecord, Habit, Category
from django.contrib.auth.admin import UserAdmin
from django.contrib.auth.models import User

admin.site.register(Habit)
admin.site.register(Category)
admin.site.register(HabitRecord)
admin.site.unregister(User)

class CustomUserAdmin(UserAdmin):
    list_display = ('id', 'username', 'email' )

admin.site.register(User, CustomUserAdmin)
```

models.py

```
import datetime
from django.db import models
from django.contrib.auth.models import User

class Category(models.Model):
    name = models.CharField(max_length=250)
    description = models.CharField(max_length=500, blank=True)
    is_active = models.BooleanField(default=True)
    color = models.CharField(max_length=7, default="#FFFFFF") #
    Default color set to white
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='user_categories')

    class Meta:
        unique_together = ['user', 'name']
        ordering = ['created at']
```



```
verbose_name_plural = 'Categories'

def __str__(self):
    return f"{self.id} - {self.user.username} - {self.name}"

def default_days():
    return {
        "Monday": True,
        "Tuesday": True,
        "Wednesday": True,
        "Thursday": True,
        "Friday": True,
        "Saturday": True,
        "Sunday": True,
    }

class Habit(models.Model):
    name = models.CharField(max_length=250)
    description = models.CharField(max_length=500, blank=True)
    reason = models.CharField(max_length=400, blank=True)
    days_of_the_week = models.JSONField(default=default_days)
    start_date = models.DateField(default=datetime.date.today)
    end_date = models.DateField(null=True, blank=True)
    is_active = models.BooleanField(default=True)
    is_complete = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    category = models.ForeignKey(Category, on_delete=models.SET_NULL,
related_name='category_habits', null=True,
                                blank=True)

    preferred_time = models.TimeField(null=True, blank=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='user_habits')

    def __str__(self):
        if self.category:
            return f"{self.id} - {self.user.username:20} -
{self.category.name:30} - {self.name:40}"
        else:
            return f"{self.id} - {self.user.username:20} -
{self.name:40}"

    class Meta:
        unique_together = ['user', 'name']
        ordering = ['created_at']

class HabitRecord(models.Model):
    STATUS_OPTION = [
        ('DONE', "Done"),
        ('FAIL', 'Fail'),
        ('SKIP', 'Skip'),
        ('DELETE', 'Delete'),
    ]

    date = models.DateField(auto_now_add=True)
    performed = models.CharField(max_length=10, choices=STATUS_OPTION)
    habit = models.ForeignKey(Habit, on_delete=models.CASCADE,
```



```
related_name='habit_records')

    def __str__(self):
        return f"{self.habit.name} - {self.date} - {self.performed}"
```

serializers.py

```
from rest_framework import serializers, status
from django.contrib.auth.models import User

from .models import HabitRecord, Category, Habit

class HabitSerializer(serializers.ModelSerializer):
    # category_name = serializers.CharField(source='category.name',
    read_only=True)
    class Meta:
        model = Habit
        fields = (
            'id', 'name', 'description', 'reason', 'category',
            'days_of_the_week', 'start_date', 'end_date', 'is_active',
            'is_complete',
            'created_at', 'updated_at'
        )
        read_only_fields = ('id', 'created_at', 'updated_at')

class CategorySerializer(serializers.ModelSerializer):
    user = serializers.PrimaryKeyRelatedField(read_only=True)

    class Meta:
        model = Category
        fields = (
            'id', 'name', 'description', 'is_active',
            'color', 'created_at', 'updated_at', 'user'
        )
        read_only_fields = ('id', 'created_at', 'updated_at', 'user')

class HabitRecordSerializer(serializers.ModelSerializer):
    class Meta:
        model = HabitRecord
        field = "__all__"

class UserSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True)

    def create(self, validated_data):
        user = User.objects.create(
            username=validated_data['username'],
        )
        user.set_password(validated_data['password'])
        user.save()
        return user

    class Meta:
```



```
model = User
fields = ('username', 'password')
```

urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('api/v1/categories/',
views.Categories.as_view(), name='categories_by_user'),
    path('api/v1/categories/<int:pk>', views.CategoryDetails.as_view(),
name='category_detail_by_user'),
    path('api/v1/habits/', views.Habits.as_view(),
name="habits_by_user"),
    path('api/v1/habits/<int:pk>', views.HabitDetails.as_view(),
name="habit_detail_by_user"),
    path('signup/', views.UserCreate.as_view(), name='signup'),
]
```

views.py

```
from rest_framework.response import Response
from rest_framework import status, generics
from rest_framework.views import APIView
from rest_framework.permissions import IsAuthenticated
from django.core.mail import send_mail
from django.conf import settings
from django.template.loader import render_to_string

from .serializers import HabitSerializer, CategorySerializer,
HabitRecord, UserSerializer
from .models import Habit, HabitRecord, Category

from django.db.utils import IntegrityError

class Categories(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        user_categories = Category.objects.filter(user=request.user)
        serializer = CategorySerializer(user_categories, many=True)
        return Response(serializer.data)

    def post(self, request):
        serializer = CategorySerializer(data=request.data)
        if serializer.is_valid():
            try:
                serializer.save(user=request.user)
                return Response(serializer.data,
status=status.HTTP_201_CREATED)
            except IntegrityError:
                return Response({
```



```
        "detail": "A CATEGORY with this name already exists
for this user."
    },
    status=status.HTTP_400_BAD_REQUEST
)
return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

class CategoryDetails(APIView):
    def get_object(self, pk, user):
        try:
            return Category.objects.get(id=pk, user=user)
        except Category.DoesNotExist:
            return None

    def get(self, request, pk):
        category = self.get_object(pk, request.user)
        if not category:
            return Response({"detail": "CATEGORY not found"},
status=status.HTTP_404_NOT_FOUND)
        serializer = CategorySerializer(category)
        return Response(serializer.data)

    def put(self, request, pk):
        category = self.get_object(pk, request.user)
        serializer = CategorySerializer(category, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, pk):
        category = self.get_object(pk, request.user)
        if not category:
            return Response({"detail": "CATEGORY not found"},
status=status.HTTP_404_NOT_FOUND)
        category.delete()
        return Response({"detail": "CATEGORY Successfully"},
status=status.HTTP_204_NO_CONTENT)

class Habits(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        habits = Habit.objects.filter(user=request.user)
        serializer = HabitSerializer(habits, many=True)
        return Response(serializer.data)

    def post(self, request):
        serializer = HabitSerializer(data=request.data)
        if serializer.is_valid():
            try:
                serializer.save(user=request.user)
                return Response(serializer.data,
status=status.HTTP_201_CREATED)
            except IntegrityError:
                return Response({
```



```
        "detail": "A habit with this name already exists
for this user."
    },
    status=status.HTTP_400_BAD_REQUEST
)
return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

class HabitDetails(APIView):
    # permission_classes = [IsAuthenticated]

    def get_object(self, pk, user):
        try:
            return Habit.objects.get(id=pk, user=user)
        except Habit.DoesNotExist:
            return None

    def get(self, request, pk):
        habit = self.get_object(pk, request.user)
        if not habit:
            return Response({"detail": "Habit not found"},
status=status.HTTP_404_NOT_FOUND)
        serializer = HabitSerializer(habit)
        return Response(serializer.data)

    def put(self, request, pk):
        habit = self.get_object(pk, request.user)
        serializer = HabitSerializer(habit, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, pk):
        habit = self.get_object(pk, request.user)
        if not habit:
            return Response({"detail": "Habit not found"},
status=status.HTTP_404_NOT_FOUND)
        habit.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)

class UserCreate(APIView):
    """
    Creates the user.
    """

    def post(self, request, format='json'):
        serializer = UserSerializer(data=request.data)
        if serializer.is_valid():
            user = serializer.save()
            if user:
                # Load and render the email template
                subject = 'Welcome to Our Website'
                template = 'welcome_page.html' # Assuming the template
is in the root of the 'templates' directory
                context = {'username': user.username}
```



```
        message = render_to_string(template, context)

        # Send email to the user
        email_from = settings.EMAIL_HOST_USER
        recipient_list = [user.username] # Assuming username
is the email address
        send_mail(subject, message, email_from, recipient_list,
html_message=message)

        # Return response
        json = serializer.data
        return Response(json, status=status.HTTP_201_CREATED)
    return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
```

settings.py

```
"""
Django settings for habitTracking_djangoApi project.

Generated by 'django-admin startproject' using Django 4.2.5.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""
import logging
import os
from datetime import timedelta
from pathlib import Path

import environ

env = environ.Env(
    # set casting, default value
    DEBUG=(bool, False)
)

# Get an instance of a logger
logger = logging.getLogger(__name__)

# Set the project base directory
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Take environment variables from .env file
environ.Env.read_env(os.path.join(BASE_DIR, '.env'))

# False if not in os.environ because of casting above
DEBUG = env.bool('DEBUG', default=False) # Default to True if DEBUG is
not set in .env

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = env('SECRET_KEY')
```



```
ALLOWED_HOSTS = env.list('DJANGO_ALLOWED_HOSTS', default=['localhost',
'127.0.0.1', '[:,1]'])

# Application definition

INSTALLED_APPS = [
    'tracking_service',
    'rest_framework',
    'debug_toolbar',
    'rest_framework.authentication',
    'rest_framework_simplejwt',

    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'tracking_service.management',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    "debug_toolbar.middleware.DebugToolbarMiddleware",

]

ROOT_URLCONF = 'habitTracking_djangoApi.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
    },
    {
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'habitTracking_djangoApi.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases
```



```
DATABASES = {
    'default': env.db() # Reads the DATABASE_URL environment variable
}

# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
}

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(days=60),
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': False,
    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'VERIFYING_KEY': None,
    'AUTH_HEADER_TYPES': ('Bearer',),
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',
    'AUTH_TOKEN_CLASSES':
('rest_framework_simplejwt.tokens.AccessToken',),
    'TOKEN_TYPE_CLAIM': 'token_type',
}

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '{levelname} {asctime} {module} {message}',
            'style': '{',
        },
        'simple': {
            'format': '{levelname} {message}',

```



```
        'style': '{}',
    },
},
'handlers': {
    'console': {
        'class': 'logging.StreamHandler',
        'formatter': 'verbose',
    },
},
'loggers': {
    'django': {
        'handlers': ['console'],
        'level': 'WARNING',
        'propagate': True,
    },
    'django.utils.autoreload': {
        'handlers': ['console'],
        'level': 'WARNING',
        'propagate': False,
    },
    'tracking_service': { # Replace 'myapp' with the name of your
app or any other logger you want to configure
        'handlers': ['console'],
        'level': 'DEBUG',
        'propagate': False,
    },
    'habitTracking_djangoApi.settings': { # Add this logger
        'handlers': ['console'],
        'level': 'DEBUG',
    },
},
'root': {
    'handlers': ['console'],
    'level': 'WARNING',
},
}

# logging.basicConfig(level=logging.DEBUG, format='%(levelname)s:
%(message)s')

# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/
LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/
STATIC_URL = 'code/static/'
```



```
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

# SMTP Configuration
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com' # Your SMTP server address
EMAIL_PORT = 587 # Port for SMTP (587 is commonly used for TLS)
EMAIL_USE_TLS = True # Whether to use TLS encryption
EMAIL_HOST_USER = 'habit.tracker.new.app@gmail.com' # Your email
address for sending emails
EMAIL_HOST_PASSWORD = 'joct ikmc qmmr krvg' # Your email password or
app password if using Gmail

INTERNAL_IPS = [
    "127.0.0.1",
]

if DEBUG:
    def show_toolbar(request):
        return True

    DEBUG_TOOLBAR_CONFIG = {
        "SHOW_TOOLBAR_CALLBACK": show_toolbar,
    }
# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-
field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Dockerfile

```
# Use a base image with Python and ping installed
FROM python:3.11-slim-bookworm

# Set environment variables
ENV PIP_DISABLE_PIP_VERSION_CHECK 1
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# Set work directory
WORKDIR /code

# Install dependencies
COPY ./requirements.txt .
RUN pip install -r requirements.txt

# Copy project
COPY . .

EXPOSE 8000
```



docker-compose.yml

```
version: "3"
services:
  app:
    image: tracking_habit_app_image
    container_name: tracking_app_container
    build:
      context: .
      dockerfile: Dockerfile
    command: >
      sh -c "python /code/manage.py collectstatic --noinput
      && python /code/manage.py migrate --noinput
      && python /code/manage.py runserver 0.0.0.0:8000"
    volumes:
      - ./code
      - static:/code/static
    ports:
      - 8000:8000
    depends_on:
      - db
  #   entrypoint: ["python", "/code/management/commands/init_admin.py"]
  db:
    image: postgres:16-alpine
    container_name: tracking_db_container
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - "POSTGRES_HOST_AUTH_METHOD=trust"
      - "POSTGRES_USER=uth_user"
      - "POSTGRES_PASSWORD=cangetin"
      - "POSTGRES_DB=habits_db"

volumes:
  postgres_data:
  media:
  static:
```