ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εξαγωγή Ομαδοποιημένων Συστάσεων με Χρήση Android Εφαρμογής στον Τουριστικό Τομέα

Ελένη Α. Καλογιάννη (Eleni A. Kalogianni)

Επιβλέποντες :
Σταμούλης Γεώργιος, Καθηγητής Πανεπιστημίου Θεσσαλίας
Κολομβάτσος Κωνσταντίνος, Επιστημονικός Σύμβουλος

ΛΑΜΙΑ
ΟΚΤΩΒΡΙΟΣ 2019

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εξαγωγή Ομαδοποιημένων Συστάσεων με Χρήση Android Εφαρμογής στον Τουριστικό Τομέα

Ελένη Α. Καλογιάννη (Eleni A. Kalogianni)
Α.Μ: 2113128

Επιβλέποντες:
Σταμούλης Γεώργιος, Καθηγητής Πανεπιστημίου Θεσσαλίας
Κολομβάτσος Κωνσταντίνος, Επιστημονικός Σύμβουλος

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Όνομα Επώνυμο,** Τίτλος (π.χ. Διδάκτωρ**)**

ΠΕΡΙΛΗΨΗ

Τα συστήματα συστάσεων παρέχουν εξατομικευμένες προτάσεις στους χρήστες για θέματα πιθανού ενδιαφέροντος. Κύρια χρησιμότητά τους, είναι η υποστήριξη των χρηστών στην επιλογή αντικειμένων μιας συγκεκριμένης κατηγορίας (όπως για παράδειγμα, ταινίες, βιβλία, Η/Υ, κ.α.). Τα συστήματα συστάσεων χρησιμοποιούνται ευρέως σε σελίδες ηλεκτρονικού εμπορίου, παρουσιάζοντας λίστες επιμέρους προϊόντων ή υπηρεσιών και επιτρέποντας στον χρήστη να προχωρήσει στην αγορά των επιλεγμένων προϊόντων. Για την αποτελεσματικότητα τους, τα μοντέρνα συστήματα εφαρμόζουν τεχνικές φιλτραρίσματος των δεδομένων, όπως το φιλτράρισμα βάσει τη γνώση (Knowledge-based Filtering) που συστήνει αντικείμενα με βάσει τις πληροφορίες που παρέχονται. Η γνώση αυτή μπορεί να προέρχεται από βάσεις δεδομένων, servers κ.α. Στη παρούσα εργασία, γίνεται χρήση των συστημάτων συστάσεων για την καταχώριση χρηστών σε ταξιδιωτικούς προορισμούς με γνώμονα την ηλικία τους, υπολογίζοντας το συνολικό χρόνο της πιθανής διαδρομής και επιλέγοντας εκείνες που ταιριάζουν καλύτερα στον χρήστη. Οι χρήστες χωρίζονται σε δύο ηλικιακές ομάδες, αυτές των άνω και κάτω των 40 ετών. Έτσι, επιλέγονται διαδρομές με μικρότερο συνολικό χρόνο διαδρομής για τις μεγαλύτερες ηλικίες. Τα αποτελέσματα παρέχονται μέσω ενός συστήματος κανόνων - υποθέσεων (rule - based system) και παρουσιάζονται σε μια εγγενή εφαρμογή (native application) Android για κινητά τηλέφωνα. Η εφαρμογή αυτή, μπορεί να χρησιμοποιηθεί χωρίς σύνδεση στο Διαδίκτυο (offline). Ο σχεδιασμός και η ανάπτυξη της, έγιναν στις πλατφόρμες Android Studio και Eclipse.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ : Συστήματα Συστάσεων
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ : Φιλτράρισμα βάσει τη γνώση, καταχώριση χρηστών, συστάσεις βάσει ηλικίας, σύστημα κανόνων

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ

Κατά τη διάρκεια των τελευταίων δεκαετιών, η απότομη μεγέθυνση του όγκου πληροφοριών, καθώς και η αυξανόμενη χρήση των διαδικτυακών εφαρμογών - online αγορές, κοινωνική δικτύωση, ηλεκτρονικό ταχυδρομείο κ.α. - έκανε απαραίτητη την σωστή εξυπηρέτηση των χρηστών. Τόσο το φαινόμενο της υπερ - πληροφόρησης (information overload), όσο και γενικότερα τα προβλήματα σχετικά με την αποτελεσματικότητα των αναζητήσεων στο Διαδίκτυο, έπρεπε να επιλυθούν. Πολλοί χρήστες έρχονταν αντιμέτωποι με δυσκολίες στην εύκολη και γρήγορη εύρεση αυτού που επιθυμούσαν. Έτσι, με τη χρήση κατάλληλων τεχνικών διαχωρισμού και φιλτραρίσματος, ο χρήστης του Διαδικτύου είναι πλέον σε θέση να λαμβάνει εξατομικευμένες προτάσεις τόσο στις αγορές του, όσο και γενικότερα στις αναζητήσεις του. Τα συστήματα που πραγματοποιούν το φιλτράρισμα της πληροφορίας και στη συνέχεια, παρέχουν ατομικές προτάσεις στους χρήστες ονομάζονται συστήματα συστάσεων (recommender systems). Τα συστήματα αυτά, ως εργαλεία λογισμικού, αποτελούν μια εξελιγμένη μορφή των παλαιότερων συστημάτων πληροφόρησης και έχουν σκοπό την παροχή προτάσεων ανεξάρτητα με το μέγεθος ή/και την πολυπλοκότητα του χώρου πληροφορίας (information space) όπου προέρχονται.

Η τεχνολογία των συστημάτων συστάσεων εμφανίστηκε για πρώτη φορά το 1995, γνωρίζοντας ραγδαία ανάπτυξη, λόγω του εύρους προβλημάτων που επιλύει. Ένας από τους βασικούς της στόχους, ήταν η χρησιμοποίηση των συστημάτων για την πρόβλεψη της χρησιμότητας ενός συγκεκριμένου αντικειμένου [24]. Αυτό μπορεί να επιτευχθεί με τη χρήση αξιολόγησης ενός προϊόντος, μετά από την αναζήτηση και επιλογή του από έναν ηλεκτρονικό κατάλογο. Συνήθως, επιλέγεται το αντικείμενο (ή το προϊόν) που ενδιαφέρει περισσότερο το χρήστη, με αποτέλεσμα η μετέπειτα αξιολόγηση να οδηγεί στη πρόβλεψη χρησιμότητάς του. Ακόμη, χρησιμοποιήθηκαν για να συστήσουν στον χρήστη μια λίστα από στοιχεία. Κάνοντας αναζήτηση, το σύστημα είναι σε θέση να προτείνει μια σειρά από συγκεκριμένα αποτελέσματα, διαχωρίζοντας ποια θεωρεί σημαντικά και ποια όχι. Η εύρεση και ομαδοποίηση των στοιχείων αυτών, μας δίνει τη δυνατότητα καλύτερων προτάσεων και πιο στοχευμένων πληροφοριών σχετικά με το αντικείμενο αναζήτησης.

Τα συστήματα συστάσεων, έγιναν ιδιαίτερα δημοφιλή στο ηλεκτρονικό εμπόριο (E - Commerce), αυξάνοντας τις πωλήσεις στις ιστοσελίδες όπου χρησιμοποιούνταν. Ενσωματώθηκαν σε ηλεκτρονικά καταστήματα, εφαρμόζοντας διάφορες τεχνικές και παρέχοντας σχετικές υπηρεσίες για λόγους καλύτερης εξυπηρέτησης. Το amazon.com, το ebay.com και το e-shop.gr είναι μερικές από τις ιστοσελίδες αυτές, κάνοντας χρήση τεχνικών βασισμένων σε παλαιότερες επιλογές των πελατών. Στα πλαίσια βελτίωσης των συστημάτων συστάσεων γίνεται χρήση εφαρμογών από πεδία, όπως η τεχνητή νοημοσύνη (artificial intelligence), η εξόρυξη δεδομένων (data mining), η μηχανική μάθηση (machine learning) και η ικανοποίηση περιορισμών (constraint satisfaction). Ακόμη, τα συστήματα αυτά οφείλουν την αποτελεσματικότητα τους στο φιλτράρισμα της συμπεριφοράς του χρήστη με τη βοήθεια ποικίλων μεθόδων. Οι μέθοδοι περιλαμβάνουν τόσο το συνεργατικό φιλτράρισμα (collaborative filtering) όσο και το φιλτράρισμα βάσει περιεχομένου (content based filtering), όπου αποτελούν τους πιο γνωστούς τρόπους φιλτραρίσματος.

Στο συνεργατικό φιλτράρισμα (ΣΦ) το σύστημα δημιουργεί το προφίλ του ενεργού χρήστη με το οποίο αλληλεπιδρούν οι υπόλοιποι που έχουν ομοιότητες με αυτόν. Κάποιες από τις μεθόδους που ενσωματώνονται στα συστήματα από το ΣΦ είναι τα μοντέλα πρόβλεψης (predictive models), όπου χρησιμοποιούν στατιστικά στοιχεία για την πρόβλεψη των αποτελεσμάτων και η ευρετική αναζήτηση (heuristic search), στην οποία γίνεται έλεγχος όλων των αλγορίθμων αναζήτησης, αλλά και των δεδομένων που επιστρέφουν σε κάθε βήμα διακλάδωσης πληροφοριών. Η συλλογή δεδομένων (data collection), καθώς και η αλληλεπίδραση με τον χρήστη (user interaction) αποτελούν, επίσης, τεχνικές του ΣΦ. Συγκρίνοντας τα προφίλ των χρηστών, το σύστημα εγκρίνει και συστήνει αντικείμενα που πιθανά ενδιαφέρουν το χρήστη περισσότερο.

Εκτός από τη μέθοδο του ΣΦ, η χρήση φιλτραρίσματος βάσει περιεχομένου (ΦΒΠ) εμφανίζεται ιδιαίτερα συχνά. Η τεχνική αυτή βασίζεται στην ομαδοποίηση (clustering) μεγάλου όγκου παρόμοιων δεδομένων και στην κατηγοριοποίησή (classification) τους, διευκολύνοντας το σύστημα ως προς τη διαχείριση. Τα συστήματα με ΦΒΠ χρησιμοποιούνται για να προτείνουν αντικείμενα ή υπηρεσίες σε ομάδες και όχι σε μεμενωμένους χρήστες, όπως ομάδες πελατών που

επιθυμούν να αγοράσουν κάποιο συγκεκριμένο προϊόν. Τα κριτήρια για τον διαχωρισμό των μελών κάθε ομάδας και στο τέλος τα αποτελέσματα αυτών δίνονται ύστερα από ΦΒΠ του συστήματος.

Τα συστήματα συστάσεων χωρίζονται σε πέντε βασικές κατηγορίες:

- Συστήματα βασισμένα στην συνεργασία (Collaborative filtering systems)
- Συστήματα βασισμένα στο περιεχόμενο (Content - based systems)
- Συστήματα με βάσει τη γνώση (Knowledge - based systems)
- Συστήματα βασισμένα σε δημογραφικά δεδομένα (Demographic - based systems)
- Υβριδικές μέθοδοι συστάσεων (Hybrid recommendation methods)

2. ΣΥΣΤΗΜΑΤΑ ΣΥΣΤΑΣΕΩΝ

Στα πλαίσια διατύπωσης ορισμού, αρκετοί ερευνητές έκαναν τις δικές τους αναφορές. Πρώτα, οι Resnick και Varian (1997), δίνουν έμφαση στην υποστήριξη του συστήματος για τη συνεργασία μεταξύ των χρηστών, παρέχοντας τις συστάσεις ως εισόδους και προωθώντας κατάλληλα στους χρήστες. Στη συνέχεια, ο Burke (2002) δίνει ένα πιο γενικό ορισμό, όπου "σύστημα συστάσεων καθίσταται κάθε σύστημα που παράγει εξατομικευμένες συστάσεις ως έξοδο ή καθοδηγεί το χρήστη με εξατομικευμένο τρόπο σε χρήσιμα ή ενδιαφέροντα αντικείμενα μέσα από ένα μεγάλο χώρο δυνατών επιλογών".

Σημαντική είναι και η διατύπωση του προβλήματος των συστημάτων συστάσεων, από τους Adomavicius και Tuzhilin (2005): " Έστω C το σύνολο όλων των χρηστών και S το σύνολο όλων των πιθανών αντικειμένων που μπορούν να συστηθούν. Έστω u, μια συνάρτηση χρησιμότητας (utility function) που μετράει τη χρησιμότητα του αντικειμένου s για τον χρήστη c, η οποία είναι: u: C x S => R, όπου το R είναι πλήρως διατεταγμένο σύνολο (π.χ. οι πραγματικοί αριθμοί σε ένα συγκεκριμένο διάστημα). Ακολούθως, για κάθε χρήστη c ∈C πρέπει να επιλεγεί αντικείμενο s'∈ S τέτοιο ώστε να μεγιστοποιεί τη χρησιμότητα για το συγκεκριμένο c ". Πιο συγκεκριμένα :

$$\forall c \in C, \quad s'_c = \arg\max_{s \in S} u(c, s) \tag{1}$$

2.1 Τεχνικές Φιλτραρίσματος

Στο κεφάλαιο αυτό, αναλύουμε τις κατηγορίες των συστημάτων συστάσεων βάσει των πληροφοριών που χρησιμοποιούν ως είσοδο.

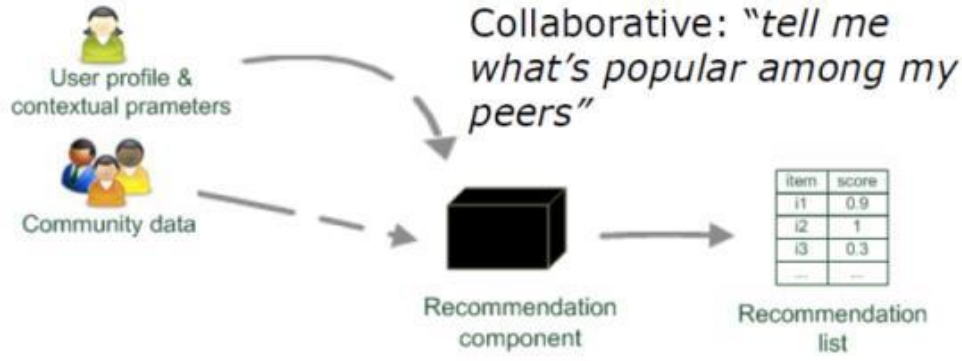
2.1.1 Συνεργατικό φιλτράρισμα (Collaborative Filtering)

Το συνεργατικό φιλτράρισμα (ΣΦ) αναφέρεται για πρώτη φορά στο Tapestry project, ένα πειραματικό σύστημα αλληλογραφίας που αναπτύχθηκε από το Xerox PARC το 1992, καθιστώντας τη μέθοδο μια από της δημοφιλέστερες [23]. Εστιάζει στη δημιουργία εξατομικευμένων προτάσεων βάσει της ομοιότητας άλλων χρηστών με

τον ενεργό χρήστη. Γενικότερα η τεχνική αυτή, αναφέρεται σε ένα σύνολο χρηστών U = {u_1, u_2, . . ., u_m} και ένα σύνολο στοιχείων, I ={i_1, i_2, . . . , i_n}, όπου για κάθε χρήστη $u_i \in U$, διαμορφώνεται ένα προφίλ προτιμήσεων με αντικείμενα που έχει βαθμολογήσει, $I_u \subseteq I$, καθώς και με την αντίστοιχη βαθμολογία κάθε προϊόντος. Το εύρος των αξιολογήσεων κυμαίνεται στο πεδίο των ακεραίων αριθμών, συνήθως με τιμές από 1 εώς 5, όπου στη συνέχεια ορίζεται ο τελικός πίνακας βαθμολογιών κάθε χρήστη. Οι προβλέψεις βασίζονται στην βαθμολογία αυτή. Πιο συγκεκριμένα, η αξιολόγηση στηρίζεται στην απόκληση μεταξύ του μέσου όρου βαθμολογίας ενός αντικειμένου (για παράδειγμα, ενός τραγουδιού) από την νέα ψηφοφορία του ενεργού χρήστη [25].

Η μέθοδος αυτή παρουσιάζει ιδιαίτερη επιτυχία απόδοσης και ακρίβειας, παρά την εξαιρετική της απλότητα σε σχέση με άλλες στρατηγικές. Συγκρίσεις των αποτελεσμάτων φιλτραρίσματος με καλύτερες τεχνικές, οδήγησαν στο συμπέρασμα ότι το ΣΦ παράγει καλύτερα υπολογιστικά αποτελέσματα, καθιστώντας το ιδιαίτερα χρήσιμο για μεγάλο όγκο δεδομένων. Για την αξιολόγηση της ποιότητας της τεχνικής προσδιορίστηκαν τρεις τύποι μετρήσεων, δίνοντας έμφαση στην ακρίβεια των αποτελεσμάτων [22]. Η ακρίβεια προτίμησης, συγκρίνοντας τις μετρήσεις των βαθμολογιών του συστήματος και της πραγματικής βαθμολογίας, η ακρίβεια ταξινόμησης διαχωρίζοντας τα προϊόντα ποιοτικά χωρίς να γίνεται εύρεση του καλύτερου και η ακρίβεια κατάταξης, μετρώντας τη συσχέτιση μεταξύ της πρόβλεψης και της πραγματικής ταξινόμησης (Pearson, Spearman's ρ, και Kendall's τ) αποτελούν τους τύπους αυτούς.

Τα συστήματα ΣΦ, στη διαδικασία εύρεσης γειτονίας (nearest - neighbor) με άλλους χρήστες παρόμοιων προτιμήσεων, κάνουν χρήση διαφόρων αλγορίθμων που τα κατατάσσουν σε δυο κύριες κατηγορίες, ανάλογα με το αν βασίζονται στη μνήμη (memory - based) ή στο μοντέλο (model - based).



Εικόνα 1. Περιγραφή Συνεργατικού Φιλτραρίσματος

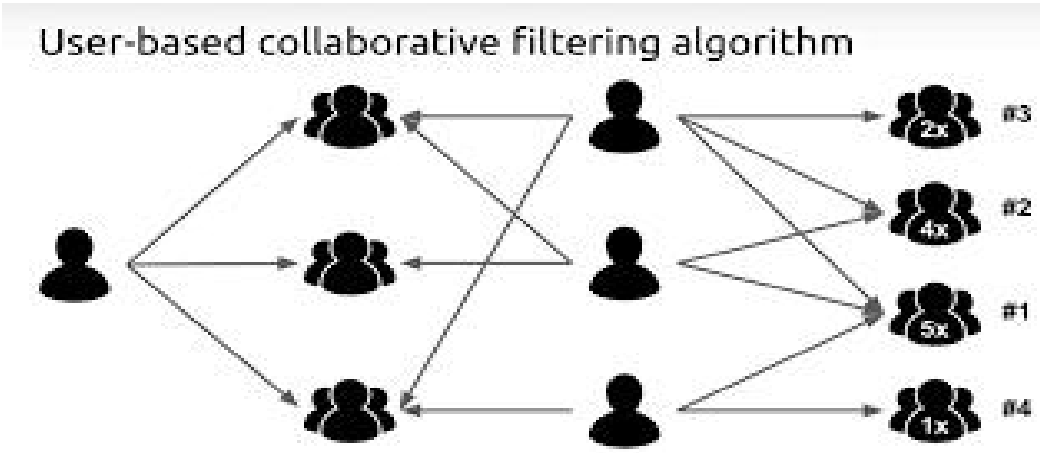
2.1.1.1 Αλγόριθμοι βασισμένοι στη μνήμη (Memory - based algorithms)

Οι αλγόριθμοι βασισμένοι στη μνήμη, προσεγγίζουν το πρόβλημα του φιλτραρίσματος χρησιμοποιώντας όλα τα δεδομένα από τη μνήμη του συστήματος, δίνοντας ιδιαίτερη βαρύτητα στην αξιολόγηση των χρηστών. Η προσέγγιση αυτή διακρίνεται σε δύο υποκατηγορίες αλγορίθμων, ανάλογα με τη διάσταση του πίνακα χρήστη x αντικείμενο (user x item) που χρησιμοποιήθηκε για την εύρεση ομοιοτήτων. Οι user - based αλγόριθμοι βασίζονται στην αναζήτηση χρηστών με παρόμοιες επιλογές αντικειμένων με τον ενεργό χρήστη, ενώ οι αλγόριθμοι βασισμένοι σε αντικείμενα (item - based) εστιάζουν στη αντίστοιχη ταξινόμηση των αντικειμένων από πολλούς χρήστες. Δηλαδή, αν πλήθος χρηστών έχουν επιλέξει κάποιο όμοιο αντικείμενο το σύστημα αντιλαμβάνεται την ομοιότητα αυτή και το προτείνει και σε άλλους. Στους αλγορίθμους με βάσει το χρήστη, λόγω του όγκου πληροφοριών του συστήματος, η αναζήτηση γίνεται πολυπλοκότερη καθώς οι χρήστες αλληλεπιδρούν μεταξύ τους. Και στις δυο περιπτώσεις αλγορίθμων, η καταγραφή των N - πλησιέστερων γειτόνων (N - Nearest Neighbours) από κάποιο στοιχείο - στόχο είναι απαραίτητη. Ο στόχος αυτός μπορεί να είναι ο ίδιος ο χρήστης ή το αντικείμενο που αναζητεί. Η ομοιότητα μεταξύ του χρήστη και των γειτόνων του προκύπτει από την κοινή εκτίμηση τους σε μια ομάδα αντικειμένων. Για τον υπολογισμό της βαθμολογίας του χρήστη σε κάποιο αντικείμενο, ορίζεται η συνάρτηση ομοιότητας χρήστη - γείτονα *sim(u,n)*, όπου u ο χρήστης και n ο γείτονας του. Έτσι, για τον υπολογισμό βαθμολογίας

του χρήστη *u* στο αντικείμενο *i*, ο σταθμισμένος μέσος όρου ορίζεται ως εξής :

$$r_{u,i} = \frac{\sum_{n \in N} sim(u,n) * r_{n,i}}{\sum_{n \in N} sim(u,n)} \quad (2)$$

όπου N είναι το σύνολο των γειτόνων του χρήστη u και $r_{n,i}$ η βαθμολογία των n για το αντικείμενο i.



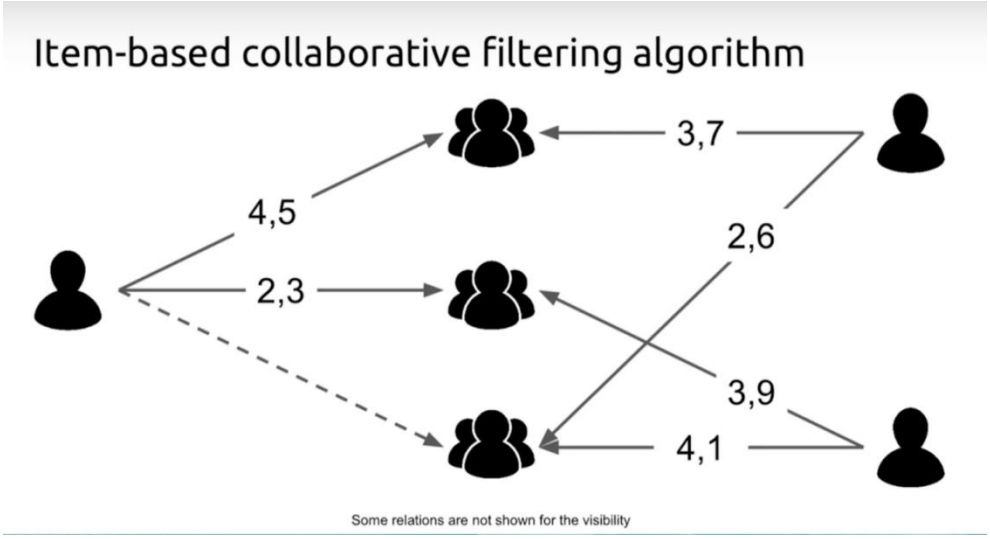
Εικόνα 2. Περιγραφή της user - based προσέγγισης

Η λειτουργία των item - based αλγόριθμων στηρίζεται στη πρόβλεψη της βαθμολογίας του χρήστη u για ένα αντικείμενο i, καθιστώντας βασικό κριτήριο την βαθμολογία του u για ένα σύνολο παρόμοιων στοιχείων. Όπως και στα user - based συστήματα είναι απαραίτητος ο ορισμός της συνάρτησης ομοιότητας *sim(i,i_0),* με τη διαφορά ότι υπολογίζουμε την ομοιότητα μεταξύ δυο αντικειμένων i και i_0. Για τον υπολογισμό της βαθμολογία του χρήστη u στο αντικείμενο i, ο σταθμισμένος μέσος όρος ορίζεται ως εξής :

$$r_{u,i} = \frac{\sum_{i_0 \in I} sim(i,i_0) * r_{u,i_0}}{\sum_{i_0 \in I} sim(i,i_0)} \quad (3)$$

όπου I είναι το σύνολο των όμοιων αντικειμένων με το i και η $r_{u,i}$ η βαθμολογία του u στο αντικείμενο i_0 [4]. Να σημειωθεί, ότι για την

πρόταση ενός προϊόντος σε ένα χρήστη είναι περισσότερη σημαντική η πώληση παρόμοιων αντικειμένων με αυτό στο παρελθόν, παρά η επιλογή του συγκεκριμένου στοιχείου από παρόμοιους χρήστες. Η μέθοδος αυτή, χρησιμοποιείται ευρέως από την Amazon.com [31].



Εικόνα 3. Περιγραφή της item - based προσέγγισης

2.1.1.2 Αλγόριθμοι βασισμένοι στο μοντέλο (Model - based algorithms)

Οι αλγόριθμοι που βασίζονται σε μοντέλα στηρίζονται στη χρήση του πίνακα user x item, χρησιμοποιώντας πληροφορίες, δημιουργώντας μοντέλα πρόβλεψης και εκπαιδεύοντας αλγόριθμους μηχανικής μάθησης. Έτσι, υπολογίζουμε την βαθμολογία για ένα άγνωστο στοιχείο βάσει το προφίλ του χρήστη. Ένα σημαντικό χαρακτηριστικό της προσέγγισης αυτής, είναι η ικανότητα οικοδόμησης μοντέλων σε κατάσταση εκτός σύνδεσης (offline). Σε αντίθεση με την " σε σύνδεση " κατάσταση (online) του συστήματος, η πολυπλοκότητα του αλγορίθμου είναι μικρότερη, αλλά η διαδικασία πιο χρονοβόρα [1]. Κάποιες από της τεχνικές των model - based αλγορίθμων στηρίζονται στη συσταδοποίηση (clustering) [26], τους πιθανοτικούς αλγόριθμους (probabilistic algorithms) [27] και τη δημιουργία μοντέλων λανθάνοντα παράγοντα [28][29] ή Παραγοντοποίηση πινάκων (latent factor model - Matrix Factorization) [30].

- **Συσταδοποίηση χρηστών (Clustering)**

Τα συστήματα που βασίζονται στην συσταδοποίηση, παρέχουν προτάσεις δημιουργώντας ομάδες χρηστών παρόμοιων προτιμήσεων ή αντικειμένων. Η προσέγγιση αυτή, παρουσιάζει καλύτερα αποτελέσμα-

τα στα user - based συστήματα κατηγοριοποιώντας όλους τους χρήστες σε ανάλογα συμπλέγματα, καθώς χρήστες που συμφώνησαν στο παρελθόν για την επιλογή ενός αντικειμένου τείνουν να συμφωνούν και στο μέλλον [2]. Στη συνέχεια, το σύστημα επεξεργάζεται τις πληροφορίες των χρηστών και παρέχει προτάσεις για κάθε ομάδα ξεχωριστά, κάνοντας τα αντικείμενα να συνιστώνται στις ομάδες ανάλογα. Έτσι, με τη διάσπαση του όγκου δεδομένων, ένας συνεργατικός αλγόριθμος αναλαμβάνει την επεξεργασία φιλτραρίσματος των ομάδων (clusters) υπολογίζοντας την ομοιότητα τους. Ο υπολογισμός γίνεται με τη χρήση της Ευκλείδειας απόστασης d μεταξύ δυο σημείων χρηστών. Το αποτέλεσμα όμως, δεν είναι πάντα έγκυρο, καθώς η απόσταση δυο χρηστών είναι αντιστρόφως ανάλογη με την ομοιότητα τους. Επομένως, όσο πιο μικρή η τιμή της απόστασης τόσο περισσότερο “μοιάζουν” δυο χρήστες [3]. Πιο συγκεκριμένα, η Ευκλείδεια απόσταση (Euclidean distance) υπολογίζει την τετραγωνική ρίζα του αθροίσματος της διαφοράς τετραγώνου μεταξύ δυο σημείων (μεταβλητών). Ο παρακάτω αποτελεί το μαθηματικό τύπο:

$$similarity(X,Y) = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2} \quad (4)$$

όπου n είναι ο αριθμός συνιστωσών των διανυσμάτων, ενώ X_i και Y_i είναι οι τιμές της i - οστής συνιστώσας.

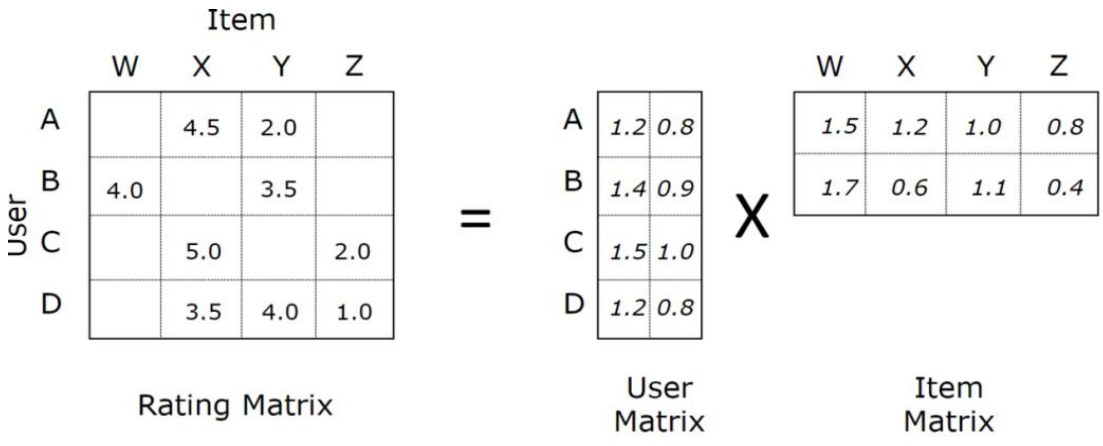
- **Πιθανοτικοί Αλγόριθμοι (Probabilistic Algorithms)**

Οι πιθανοτικοί αλγόριθμοι αναπτύχθηκαν για να παράγουν τυχαίες επιλογές, καθώς τα αποτελέσματα προτάσεων δεν μπορούν να είναι απόλυτα λόγω της υποκειμενικότητας των χρηστών αλλά και του όγκου των δεδομένων. Σε περιπτώσεις περιορισμένης μνήμης ή μερικής γνώσης η τεχνική αυτή, καθίσταται ως η μοναδική αποδοτική λύση εξαιτίας της απλότητας του τρόπου λειτουργίας της. Αν και οι πιθανοτικοί αλγόριθμοι είναι ταχύτεροι από τους ντετερμινιστικούς, συχνά παρουσιάζουν μικρές πιθανότητες λάθους, ασταθή χρόνο εκτέλεσης και εξαιρετικά δύσκολη αποσφαλμάτωση (debugging). Για την εύρυθμη και αποτελεσματική λειτουργία τους, τα πιθανοτικά μοντέλα κάνουν χρήση τόσο τεχνικών συλλογής δεδομένων όσο και “εργαλείων” των μαθηματικών. Κάποιες από τις βασικές μεθόδους, αποτελούν η τυχαία δειγματοληψία αντιπροσωπευτικού περιεχομένου

(π.χ. clusters), ο έλεγχος πιστοποιητικών (property testing), η τυχαία κατανομή των εργασιών, αλλά και το ελάχιστο κόστος, η προσομοίωση διαδικασιών, η γρήγορη ανάμιξη (rapid mixing) κ.α. [5]

- **Μοντέλο λανθάνοντα παράγοντα - Παραγοντοποίηση Πίνακα (Latent factor model - Matrix Factorization)**

Εξαιτίας της υπερ - πληροφόρησης των συστημάτων, έγινε αρκετά δύσκολη η διαδικασία εύρεσης των κατάλληλων παραγόντων για την λήψη έγκυρων αποτελεσμάτων. Έτσι, η ανάλυση παραγόντων με τη χρήση στατιστικών μοντέλων καθίσταται ιδιαίτερα σημαντική για την επίτευξη του συνεργατικού φιλτραρίσματος. Ο αλγόριθμος του λανθάνοντα παράγοντα βασίζεται στο ιστορικό του χρήστη, βρίσκοντας όλα τα στοιχεία και τις περιγραφές που είναι χρήσιμες. Για την λήψη συστάσεων δεν απαιτείται ο ορισμός άλλων παραγόντων. Τα μοντέλα LF ή Matrix Factorization μοντέλα, χρησιμοποιούνται ευρέως για την καταγραφή σημαντικών λανθανουσών συνδέσεων, μεταξύ χρηστών u και αντικειμένων v, μέσα από τις αξιολογήσεις που προκύπτουν. Στη τεχνική αυτή, υποθέτουμε ότι κάθε χρήστης είναι συνδεδεμένος με ένα στοιχείο μέσω διανυσμάτων χαρακτηριστικών χρήστη $u_i \in R^d$ και στοιχείων $v_j \in R^d$, αντίστοιχα, καθώς υποθέτουμε ότι d είναι η διάσταση των λανθανόντων μεταβλητών [13]. Στην περίπτωση όπου οι επιλογές του χρήστη u_i ταιριάζουν με τα χαρακτηριστικά του αντικειμένου v_j, τότε το στοιχείο j είναι πιθανότερο να λάβει υψηλότερες αξιολογήσεις. Συνεπώς, οι βαθμολογίες σχηματίζουν μοντέλα βάσει των εσωτερικών επιλογών προϊόντων των u_i^T και v_j, δημιουργώντας μια ποσοτική συσχέτιση μεταξύ χρηστών και αντικειμένων [6].



Εικόνα 4. Παραγοντοποίηση πίνακα ως μέθοδο δημιουργίας μοντέλων αξιολογήσεων.

2.1.1.3 Σύγκριση μεταξύ Memory - based και Model - based Προσέγγισης

Οι memory - based τεχνικές στα συστήματα συστάσεων ΣΦ χρησιμοποιούν δεδομένα όπως οι αξιολογήσεις, για την εύρεση ομοιοτήτων μεταξύ χρηστών και αντικειμένων. Η διαδικασία πραγματοποιείται ξεχωρίζοντας τους χρήστες ή τα αντικείμενα ανάλογα με την απόσταση από κάποιο σημείο - στόχο u. Τα model-based συστήματα, με την εκπαίδευση αλγορίθμων μηχανικής μάθησης αλλά και την προεπισκόπηση του προφίλ των χρηστών, είναι σε θέση να παρέχουν μοντέλα αξιολογήσεων.

Η memory - based προσέγγιση, έγινε ιδιαίτερα δημοφιλή λόγω της ακρίβειας και της απλότητας στη λειτουργία της, αλλά και εξαιτίας της ικανότητας παρακολούθησης των γειτονικών πινάκων βαθμολόγησης και προτιμήσεων από το χρήστη. Ακόμη, η μέθοδος αυτή παρέχει σταθερότητα, επηρεάζοντας ελάχιστα τα αποτελέσματα από την προσθήκη δεδομένων στο σύστημα. Έτσι, γίνεται εφικτή η διαχείριση μεγάλου όγκου πληροφοριών. Από την άλλη μεριά, η model - based προσέγγιση δεν επιλέγεται τόσο συχνά για την οικοδόμηση συστημάτων συστάσεων, όμως αποτελεί λύση στο πρόβλημα κλιμάκωσης των παραδοσιακών memory - based τεχνικών. Συνεπώς, η memory - based προσέγγιση είναι καλύτερη της model - based [4].

2.1.1.4 Προβλήματα Συνεργατικού Φιλτραρίσματος

Κατά την λειτουργία των συστημάτων συνεργατικού φιλτραρίσματος, δημιουργούνται αρκετά προβλήματα εκ των οποίων κάποια περιγράφονται παρακάτω [6]:

- **Πρόβλημα αραιών δεδομένων (Sparsity Problem)**

Ένα συχνό πρόβλημα του συνεργατικού φιλτραρίσματος, αποτελεί ο μεγάλος όγκος προτάσεων. Σε περιπτώσεις χρηστών με πολλές αξιολογήσεις, η αναζήτηση δεν θα παρέχει αξιόπιστα αποτελέσματα, καθώς το φιλτράρισμα των δεδομένων δεν θα λαμβάνει ως παράμετρο τις επιλογές των γειτονικών χρηστών. Επομένως, ο χρήστης δεν θα είναι σε θέση να ψάξει όλα τα προβαλλόμενα αποτελέσματα λόγω του πλήθους τους και ενδέχεται να μη βρει αυτό που αρχικά έψαχνε.

Η λύση του προβλήματος βασίστηκε στην προσθήκη στοιχείων για τη δημιουργία ενός πιο ολοκληρωμένου προφίλ του χρήστη. Έτσι, δίνεται

η δυνατότητα περισσότερων συνδυασμών και τελικά πιο αξιόπιστων δεδομένων [7]. Μια άλλη τεχνική είναι η χρήση του αλγορίθμου SVD (Singular Value Decomposition). Ο SVD χρησιμοποιείται για την μείωση των διαθέσιμων αποτελεσμάτων, εντοπίζοντας συσχετίσεις μεταξύ των προτάσεων και των χρηστών και παρέχοντας καλύτερες συστάσεις.

- **Πρόβλημα ψυχρής εκκίνησης (Cold Start Problem)**

Το πρόβλημα αυτό, παρατηρείται κατά την εισαγωγή ενός νέου χρήστη ή αντικειμένου και χωρίζεται σε δυο περιπτώσεις: το πρόβλημα του νέου χρήστη (New User Cold Start Problem), και του νέου αντικειμένου (New Item Cold Start Problem). Το πρώτο πρόβλημα (New User Cold Start Problem), οφείλεται στην αδυναμία διαχείρισης του νέου χρήστη, καθώς το σύστημα δεν γνωρίζει τα ενδιαφέροντα και τις αξιολογήσεις του. Ακόμη και αν ο χρήστης εισήγαγε στοιχεία για τη δημιουργία προφίλ, το σύστημα δεν θα ήταν σε θέση να παρέχει σωστά αποτελέσματα, μέσω της αδυναμίας εύρεσης όμοιων χρηστών. Η λύση στηρίζεται στη χρήση δημογραφικών δεδομένων που αποτελεί συνδυασμό των τεχνικών του ΣΦ και του ΦΒΠ, συμπληρώνοντας το προφίλ του χρήστη με πιο γρήγορες τεχνικές από την παραδοσιακή αναμονή για την αξιολόγηση των στοιχείων. Η συμπλήρωση γίνεται με στοιχεία αυξημένης δημοτικότητας κ.α.

Το δεύτερο πρόβλημα (New Item Cold Start Problem), δημιουργείται με την εισαγωγή ενός καινούριου αντικειμένου στο σύστημα. Η έλλειψη παλαιότερων αξιολογήσεων θα οδηγήσει στην αδυναμία σύστασης του σε χρήστες και συνεπώς στον αποκλεισμό του από τα αποτελέσματα αναζητήσεων. Η λύση του προβλήματος βασίζεται στον συνδυασμό των τεχνικών του ΣΦ και του ΦΒΠ.

- **Πρόβλημα των συνωνύμων (Synonymy)**

Το σύστημα στηρίζει τη διαδικασία εύρεσης αποτελεσμάτων σε λέξεις κλειδιά χωρίς όμως, να περιλαμβάνονται λέξεις όμοιες ή συνώνυμες. Έτσι, σε περίπτωση χρήσης παρόμοιων λέξεων για την αναζήτηση ενός αντικειμένου, τα αποτελέσματα θα είναι ελλιπή και ανακριβή.

- **Πρόβλημα μη αναμενόμενης προτίμησης των χρηστών**

Το πρόβλημα αυτό, δημιουργείται σε περιπτώσεις που ο χρήστης επιλέγει ένα στοιχείο μικρής δημοτικότητας, δηλαδή ελάχιστων αξιολογήσεων. Το σύστημα δεν είναι σε θέση να συσχετίσει κατάλληλα

τις αξιολογήσεις και να παράγει αξιόπιστα αποτελέσματα, καθώς όσοι περισσότεροι χρήστες επιλέγουν ένα αντικείμενο, τόσο καλύτερη προσέγγιση συστάσεων έχουμε.

- **Πρόβλημα αλλαγής προτιμήσεων των χρηστών**

Στο πέρασμα του χρόνου, οι επιλογές των χρηστών μπορεί να αλλάξουν και κάποια αντικείμενα που προτιμούσαν στο παρελθόν τώρα να μην τα προτιμούν. Στην περίπτωση αυτή, οι αξιολογήσεις συνεχίζουν να υπάρχουν στο σύστημα, επηρεάζοντας την ποιότητα των αποτελεσμάτων. Η αντιμετώπιση του προβλήματος δεν αποτελεί εύκολη διαδικασία, καθώς δεν μπορεί να προβλεφθεί η προτίμηση των χρηστών σε σχέση με την υποκειμενικότητα και το χρόνο. Υπάρχουν και άλλοι περιορισμοί στη λειτουργία του ΣΦ, όπως το στατιστικό πρόβλημα, το πρόβλημα πολλών αντικειμένων, το πρόβλημα της κλίμακας (scalability problem) κ.α.

2.1.2 Φιλτράρισμα βασισμένο στο Περιεχόμενο (Content - Based Filtering)

Το φιλτράρισμα βασισμένο στο περιεχόμενο (ΦΒΠ), έκανε την εμφάνιση του πριν από το συνεργατικό φιλτράρισμα, καθιστώντας τη τεχνική αυτή από τις πρώτες στην οικοδόμηση συστημάτων συστάσεων. Συνδυάζοντας τις πληροφορίες για ένα στοιχείο με τις προτιμήσεις του χρήστη, το σύστημα βρίσκεται σε θέση να παρέχει προτάσεις για τα κατάλληλα αντικείμενα. Λόγω της εξάρτησης των αποτελεσμάτων με τα χαρακτηριστικά περιεχομένου των αντικειμένων, η μέθοδος αυτή έγινε γνωστή ως “συστάσεις βάσει περιεχομένου” (content - based recommendation) [8]. Τα συστήματα ΦΒΠ προέρχονται από τα επιστημονικά πεδία της Ανάκτηση Πληροφορίας (Information Retrieval) και της Τεχνητής Νοημοσύνης (Artificial Intelligence) και χρησιμοποιούνται σε πολλούς τομείς, ανάμεσα εκ των οποίων βρίσκονται προτάσεις για ιστοσελίδες, πωλήσεις αντικειμένων, διαφημίσεις κ.α. Αν και οι λεπτομέρειες σε κάθε τομέα διαφέρουν, η βασική δομή των συστημάτων συστάσεων ΦΒΠ για την ανάλυση του περιεχομένου παραμένει ίδια.

Το ΦΒΠ στηρίζεται στις ατομικές προτιμήσεις του χρήστη, δημιουργώντας μικρότερη ποικιλομορφία στις αντίστοιχες συστάσεις. Αν θεωρήσουμε ότι ένας χρήστης δεν έχει επιλέξει ποτέ του μια

κατηγορία αντικειμένων, τότε το σύστημα από μόνο του δεν θα του προτείνει κάτι διαφορετικό που ενδεχομένως να αρέσει σε άλλους χρήστες. Για την κατασκευή ενός τέτοιου συστήματος, αρχικά είναι απαραίτητη η ανάλυση των χαρακτηριστικών κάθε στοιχείου. Τα δεδομένα της ανάλυσης αποτελούν την είσοδο για την εκμάθηση του προφίλ του χρήστη, έτσι ώστε να πραγματοποιηθεί συνδυασμός με το περιεχόμενο των αντικειμένων. Στη συνέχεια, το σύστημα διαλέγει ποιο αντικείμενο αποτελεί κατάλληλη σύσταση. Το προφίλ του χρήστη απαρτίζεται από τις προτιμήσεις του και από κάποια στοιχεία για τα διάφορα αντικείμενα της αρεσκείας του. Έτσι, βάσει τα δεδομένα από το προφίλ του χρήστη τα αντικείμενα ενημερώνονται εκ νέου. Τεχνικές όπως η αξιολόγηση (rating), τα σχόλια με τη μορφή κειμένου (text comments) και η κατηγοριοποίηση αρέσω /δεν αρέσω (like/dislike), βοηθούν στην ενημέρωση αυτή. Η χρησιμότητα ενός στοιχείου s για ένα χρήστη c, στηρίζεται στην αξιολόγηση του χρήστη για τη χρησιμότητα $u(c,s_i)$ των στοιχείων $s_i \in S$ που παρουσιάζουν κοινά χαρακτηριστικά με το s. Η συσχέτιση της προσέγγισης με την επιβλεπόμενη μηχανική μάθηση, μπορεί να μετατρέψει το πρόβλημα εκμάθησης σε ομάδες ταξινομητών, όπου οι κατηγορίες διακρίνονται σε “χρήσιμο για τον χρήστη X” και “όχι χρήσιμο για τον χρήστη X” [8]. Για την εκτίμηση της βαθμολογίας ενός χρήστη u προς ένα αντικείμενο i, υπολογίζεται η συνάρτηση χρησιμότητας *Utility(u,i).* Ένα τρόπος εύρεσης είναι η χρήση της συνάρτησης ομοιότητας συνημιτόνου *similarity* που είναι η εξής :

$$similarity = cos(\theta) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2}\sqrt{\sum_{i=1}^{n} v_i^2}} \quad (5)$$

όπου u είναι ο χρήστης και v το αντικείμενο.

Και η συνάρτηση χρησιμότητας η παρακάτω :

$$Utility(u,i) = \frac{w_u \cdot w_i}{\|w_u\|_2 \times \|w_i\|_2} \quad (6)$$

όπου u ο χρήστης, i το αντικείμενο, $w_u = \{w_{u1},\ldots,w_{un}\}$ το διάνυσμα για το προφίλ του χρήστη και $w_i = \{w_{i1},\ldots,w_{in}\}$ το διάνυσμα για τα χαρακτηριστικά του αντικειμένου.

2.1.2.1 Προβλήματα Φιλτραρίσματος Βάσει Περιεχομένου

Τα συστήματα ΦΒΠ, συχνά αντιμετωπίζουν προβλήματα στη λειτουργία τους κάποια εκ των οποίων είναι τα εξής:

- **Περιγραφή περιεχομένου (Content description)**

Σε κάποιες περιπτώσεις αντικειμένων, η παρουσίαση αποτελεί αρκετά δύσκολη διαδικασία. Για παράδειγμα, η περιγραφή μουσικών επιλογών ή βίντεο δεν είναι πάντα εφικτή.

- **Αρκετά μεγάλη εξειδίκευση (Over-specialization)**

Με την τεχνική ΦΒΠ το σύστημα δεν είναι σε θέση να επιλέξει κάτι για το οποίο δεν γνωρίζει, καθώς δεν έχει ξανά επιλεχθεί στο παρελθόν από τον χρήστη και δεν υπάρχουν πληροφορίες για αυτό. Γίνεται, επομένως, απαραίτητη η διαδικασία προσθήκης μεθόδων για να συμπεριληφθούν συστάσεις εκτός του πεδίου εφαρμογής που χρησιμοποιείται το ΦΒΠ.

- **Υποκειμενικό πρόβλημα τομέα (Subjective domain problem)**

Το πρόβλημα βασίζεται στην αντίληψη, πως η υποκειμενικότητα του χρήστη μπορεί να αποτελέσει εμπόδιο στην ποιότητα των συστάσεων.
Στις περιπτώσεις, όπου χρήστες είχαν “ιδιαίτερες” απόψεις ή έκαναν χιούμορ, το σύστημα δεν διέθετε την ικανότητα σύγκρισης και διάκρισης των πληροφοριών αυτών.

2.1.3 Χρήση Δημογραφικών Δεδομένων (Demographic - Based Filtering)

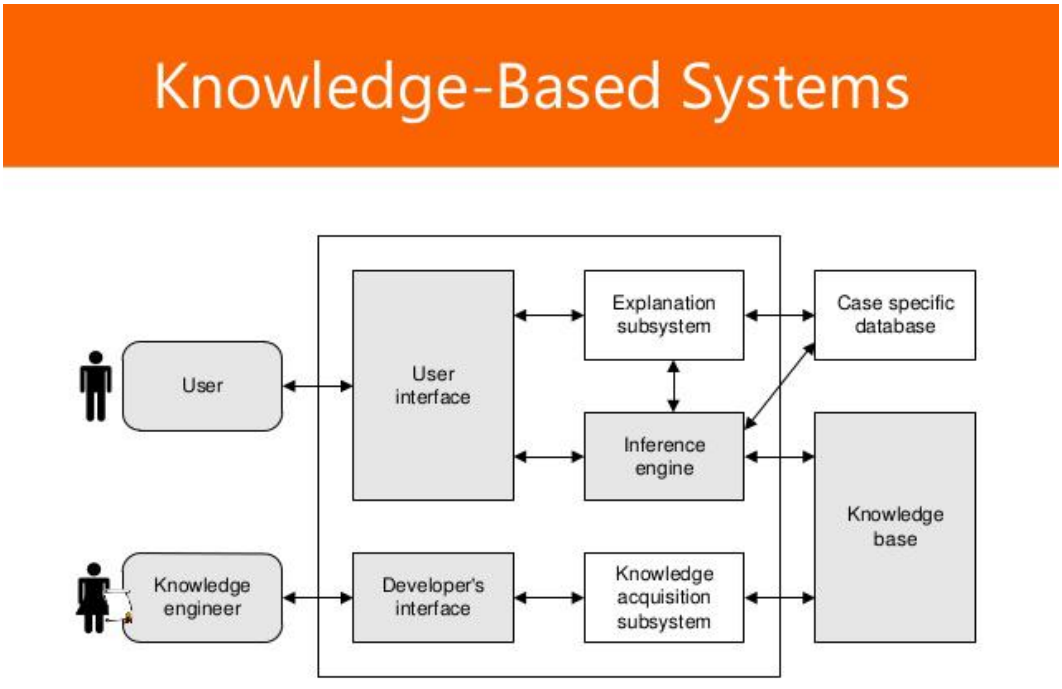
Με τον όρο δημογραφικά δεδομένα, αναφερόμαστε στα στοιχεία που πλαισιώνουν το προφίλ του χρήστη επιτρέποντας την αποδοτικότερη συσχέτιση του με τα αντικείμενα. Η χρήση δημογραφικών δεδομένων, γίνεται με την αντίληψη πως χρήστες κοινών χαρακτηριστικών θα κάνουν όμοιες επιλογές. Έτσι, η αξιολόγηση ενός χρήστη u για το στοιχείο i καθορίζεται από την αξιολόγηση χρηστών με τα ίδια δημογραφικά δεδομένα. Ένα βασικό πρόβλημα της τεχνικής αυτής, αποτελεί η ελλιπής πληροφόρηση του συστήματος λόγω της ανεπαρκής συμπλήρωσης του προφίλ του χρήστη. Οι χρήστες, δεν δίνουν όλα τα στοιχεία τους με αποτέλεσμα να μην γίνεται σωστή παροχή αποτελεσμάτων. Στη προσπάθεια πληροφόρησης του

συστήματος, αποσκοπούν και τα ερωτηματολόγια ενδιαφέροντος, όμως δεν αποτελούν ιδιαίτερη λύση. Το ποσοστό των ατόμων που συμπληρώνει διαδικτυακά ερωτηματολόγια είναι αρκετά μικρό. Η λύση στο πρόβλημα, δίνεται από τεχνικές που βασίζονται στην ταυτοποίηση στοιχείων μέσα από κάποια άλλη προσωπική σελίδα. Έτσι, το προφίλ του χρήστη εμπλουτίζεται, αντλώντας πληροφορίες πραγματικής ανταπόκρισης για τα ενδιαφέροντα και τα χαρακτηριστικά του χρήστη. Το INTRIGUE [9] αποτελεί ένα σύστημα παροχής πληροφοριών για τους επισκέπτες στη πόλη του Τορίνου της Ιταλίας, προτείνοντας προορισμούς ανάλογα με τα δημογραφικά τους δεδομένα. Για την αποτελεσματικότητα των συστάσεων, ο αλγόριθμος διαχωρίζει τους χρήστες σε διαφορετικές ομάδες επισκεπτών, όπως οι οικογένειες και οι ηλικιωμένοι.

2.1.4 Φιλτράρισμα βάσει Γνώσης (Knowledge-Based Filtering)

Τα συστήματα συστάσεων βασισμένα στη γνώση (ΦΒΓ) αποτελούν ένα ιδιαίτερο τρόπο φιλτραρίσματος, καθώς εφαρμόζεται σε περιπτώσεις όπου άλλες εναλλακτικές προσεγγίσεις, όπως το ΣΦ και το ΦΒΠ, δεν μπορούν. Το ΦΒΓ προτείνει αντικείμενα σύμφωνα με το μοντέλο του χρήστη. Το μοντέλο του χρήστη, μπορεί να αποτελείται από οποιαδήποτε δομή γνώσης επιτρέποντας τη διαχείριση σύνθετων τομέων για αντικείμενα που δεν επιλέγονται συχνά. Επιπλέον, λόγω της έλλειψης αξιολογήσεων, η εκτίμηση χρησιμότητας και ομοιότητας σύμφωνα με τις βαθμολογίες των χρηστών δεν καθίσταται αξιόπιστος τρόπος φιλτραρίσματος [10]. Έτσι, σε περίπλοκες περιπτώσεις είναι απαραίτητη η λήψη περιορισμών. Για παράδειγμα, μόνο οι τουριστικοί προορισμοί που υποστηρίζουν την ταξιδιωτική περίοδο που όρισε ο πελάτης θα πρέπει να συνιστώνται. Η προσέγγιση των περιορισμών (Constraint - based systems) βασίζεται σε ένα λεπτομερή σύστημα κανόνων σύστασης, όπου ο χρήστης πρέπει να πληρεί τους κανόνες του συστήματος. Ένας άλλος τρόπος ΦΒΓ, σχετίζεται με υποθέσεις (Case - based systems) ανάλογες των μέτρων ομοιότητας, επιλέγοντας στοιχεία που μοιάζουν με συγκεκριμένες προτάσεις. Και στις δυο περιπτώσεις, οι χρήστες δίνουν ως είσοδο στο σύστημα τις απαιτήσεις τους και το σύστημα εντοπίζει τις πιθανές προτάσεις (λύσεις). Αν δεν βρεθούν λύσεις, οι χρήστες καλούνται να αλλάξουν τις απαιτήσεις τους. Ακόμη, τα συστήματα αυτά παρουσιάζουν αρκετά πλεονεκτήματα. Η ποιότητα των αποτελεσμάτων είναι ένα από αυτά, καθώς το προφίλ του χρήστη

παρέχει όλες τις απαραίτητες πληροφορίες για την παροχή αξιόπιστων προτάσεων. Η ικανότητα προσομοίωσης διαλόγων μεταξύ πωλητή και πελάτη (σε περιπτώσεις ηλεκτρονικού εμπορίου), επίσης είναι σημαντικό χαρακτηριστικό.



Εικόνα 5. Μηχανισμός λειτουργίας του συστήματος συστάσεων βάσει γνώσεων.

2.1.4.1 Συστήματα Περιορισμών (Constraint - based Systems)

Τα συστήματα περιορισμών ενημερώνονται από μια βάση γνώσης (knowledge base), αποτελούμενη από δεδομένα του χρήστη και από χαρακτηριστικά των αντικειμένων. Η είσοδος στα συστήματα αυτά, προέρχεται συνήθως από τοπικές βάσεις δεδομένων συνδέοντας το προφίλ του χρήστη και των αντικειμένων με τους περιορισμούς. Οι κανόνες που χρησιμοποιούνται από το σύστημα είναι της μορφής “if,then,else” και η σύνδεση εκτελείται με ερωτήματα (queries) προς την βάση δεδομένων. Τα αποτελέσματα, εξαρτώνται από το είδος των κανόνων, ανάλογα με την ύπαρξη χαλαρών ή αυστηρών περιορισμών. Για την εφαρμογή του αλγορίθμου, είναι απαραίτητη η ύπαρξη χρηστών και αντικειμένων που να πληρούν τον ίδιο αριθμό κανόνων. Ένα σημαντικό πρόβλημα της προσέγγισης είναι η πολυπλοκότητα για την οικοδόμηση του συστήματος περιορισμών, καθώς πρέπει να μελετώνται όλοι οι παράμετροι. Ακόμη και αν το προφίλ και οι απαιτήσεις του χρήστη είναι σαφής, λάθη στη διατύπωση των κανόνων

θα οδηγούσαν σε μη πραγματικά αποτελέσματα. Συνεπώς, η στατικότητα των συστημάτων μπορεί να οδηγήσει σε αδιέξοδο. Χρήσιμο είναι να σημειωθεί, πως η ικανότητα των συστημάτων να αλλάζουν βάσει των αλληλεπιδράσεων μεταξύ του χρήστη με άλλους χρήστες και αντικείμενα, αποτελεί βασικό χαρακτηριστικό απόδοσης. Στα συστήματα περιορισμών η παραπάνω ικανότητα δεν ανταποκρίνεται.

2.1.4.2 Συστήματα βασισμένα σε Υποθέσεις (Case - Based Systems)

Στα συστήματα αυτά, οι συστάσεις λαμβάνονται με την χρήση μέτρων ομοιότητας, καθώς οι χρήστες και τα αντικείμενα αλληλεπιδρούν. Οι χρήστες δεν πραγματοποιούν συγκεκριμένες αναζητήσεις και το σύστημα στηρίζει τα αποτελέσματα των προτάσεων βάσει των κριτικών άλλων χρηστών. Επιπλέον, δίνεται η δυνατότητα καθορισμού ελαφρών περιορισμών, όπως το επιθυμητό εύρος τιμών στις περιπτώσεις αναζήτησης προϊόντων. Για τον υπολογισμό της ομοιότητας μεταξύ των αντικειμένων, χρησιμοποιείται η συνάρτηση sim(p,r), η οποία εκφράζει για κάθε αντικείμενο την απόσταση του από την απαίτηση του χρήστη r ∈ REQ. Ο μαθηματικός τύπος *similarity(p,REQ)* είναι ο εξής *:*

$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r} \quad (7)$$

όπου w_r είναι το βάρος σημαντικότητας για την απαίτηση r.

2.1.5 Υβριδικά Συστήματα (Hybrid Systems)

Στην προσπάθεια βελτίωσης της ποιότητας των συστάσεων, καθώς και του περιορισμό των προβλημάτων, τόσο του ΣΦ όσο και του ΦΒΠ, οι επιστήμονες δημιούργησαν υβριδικά συστήματα (ΥΣ), συνδυάζοντας τις δύο τεχνικές. Η προσέγγιση αυτή, μπορεί να υλοποιηθεί με διάφορους τρόπους όπως :

- η ξεχωριστή εκτέλεση των τεχνικών ΣΦ και ΦΒΠ και στη συνέχεια ο συνδυασμός των αποτελεσμάτων τους,

- η προσθήκη εργαλείων του ΦΒΠ στη μέθοδο του ΣΦ και αντίστροφα και
- η δημιουργία γενικού συνδυαστικού μοντέλου και των δύο τεχνικών.

Με τις παραπάνω υβριδικές προσεγγίσεις, τα συστήματα συστάσεων παρέχουν πιο αξιόπιστα και ακριβή αποτελέσματα, σε σχέση με τις τεχνικές που μελετήθηκαν παραπάνω (Collaborative, Content - Based, Demographic - Based, Knowledge - Based Filtering). Οι τεχνικές παράλληλων υβριδικών συστημάτων είναι οι ακόλουθες :

Σταθμισμένη (Weighted)
Η τεχνική αυτή χρησιμοποιείται για την σταθμισμένη σύνδεση των αποτελεσμάτων των διαθέσιμων μεθόδων σύστασης, καθώς και για τον υπολογισμό του αθροίσματος της συνολικής βαθμολογίας ενός συγκεκριμένου στοιχείου των αποτελεσμάτων.

Μεταβατική (Switching)
Με την τεχνική αυτή, αποφασίζεται η μέθοδος σύστασης που θα χρησιμοποιηθεί βάσει των παραγόμενων αποτελεσμάτων, των χαρακτηριστικών του χρήστη και των συναφών παραμέτρων, όπως τα ενδιαφέροντα και οι πεποιθήσεις του χρήστη.

Ανεξάρτητος συνδυασμός (Mixed)
Με τον τρόπο αυτό, οι προτάσεις των διαθέσιμων συστημάτων συστάσεων συνδυάζονται και παρουσιάζονται ως κοινό αποτέλεσμα.

Συνδυασμός χαρακτηριστικών (Feature combination)
Η τεχνική βασίζεται στη προσθήκη πληροφοριών του ΣΦ στη μέθοδο του ΦΒΠ, αλλά και στην δημιουργία ενός ενιαίου αλγορίθμου βάσει τα δεδομένα - χαρακτηριστικά των συστημάτων του ΣΦ και του ΦΒΠ.

Μέθοδος καταρράκτης (Cascade)
Η μέθοδος του καταρράκτη διαφέρει από τις παραπάνω τεχνικές, καθώς η κατασκευή του υβριδικού συστήματος γίνεται σταδιακά. Στην αρχή, η μια μέθοδος συστάσεων που χρησιμοποιείται, παρέχει ένα σύνολο προτεινόμενων στοιχείων. Στη συνέχεια, μια δεύτερη μέθοδος φτάνει τη διαδικασία στο τέλος, μειώνοντας τις προβαλλόμενες συστάσεις και δίνοντας ποιοτικότερα αποτελέσματα.

Επαύξηση χαρακτηριστικών (Feature augmentation)
Η επαύξηση χαρακτηριστικών χρησιμοποιείται για τον υπολογισμό της βαθμολογίας ή την κατηγοριοποίηση ενός αντικειμένου. Τα αποτελέσματα, χρησιμοποιούνται ως είσοδος για την επόμενη διαθέσιμη μέθοδο σύστασης του συστήματος.

Μετά - Επίπεδο (Meta-level):
Η τεχνική παρουσιάζει ομοιότητες με την επαύξηση χαρακτηριστικών, καθώς και στις δυο τεχνικές οι παραγόμενες πληροφορίες από την μια μέθοδο εισάγονται στην άλλη. Η διαφορά τους, βασίζεται στην εισαγωγή των δεδομένων, αφού στην μετα-επίπεδο τεχνική επιλέγεται ολόκληρο το παραγόμενο μοντέλο, ενώ στην επαύξηση χαρακτηριστικών μόνο οι πληροφορίες της προηγούμενης μεθόδου.

Τα υβριδικά συστήματα συναντώνται συχνά, σε σελίδες του διαδικτύου και όχι μόνο, λόγω της ανάγκης ποιοτικότερων και πιο αξιόπιστων συστάσεων, σύμφωνα με το προφίλ του χρήστη. Ένα παράδειγμα, αποτελεί το Netflix, μια υπηρεσία συνεχούς ροής (streaming service) που δίνει την δυνατότητα παρακολούθησης ταινιών, τηλεοπτικών εκπομπών, ντοκιμαντέρ κ.α., σε συνδεδεμένες στο διαδίκτυο συσκευές [11]. Για την επίτευξη ακρίβειας στα αποτελέσματα αναζήτησης, χρησιμοποιήθηκε μια μέθοδος που αποτελείται από 107 διαφορετικές αλγοριθμικές προσεγγίσεις σε μια κοινή πρόβλεψη [12]. Ένα άλλο παράδειγμα, αποτελεί το σύστημα συστάσεων εστιατορίων Entree, στο οποίο γίνεται συσχέτιση αλγορίθμων ΣΦ, ΦΒΠ καθώς και ανάκτησης δεδομένων βάσει γνώσης. Η απόδοση του υβριδικού συστήματος στηρίχθηκε στον υπολογισμό της μέσης βαθμολόγησης των σωστών συστάσεων (average rank of the correct recommendations - ARC).

2.2 Αξιολόγηση των Συστημάτων Συστάσεων

Η αξιολόγηση των συστημάτων συστάσεων και των αλγορίθμων που τους απαρτίζουν, καθίσταται μια εξαιρετικά δύσκολη διαδικασία για διάφορους λόγους [22]:

- Κάποιοι αλγόριθμοι παρουσιάζουν καλύτερα ή χειρότερα αποτελέσματα, ανάλογα με το σύνολο των δεδομένων που δέχονται ως είσοδο.
- Η χρήση αλγορίθμων για ειδικές περιπτώσεις συνόλων. Για παράδειγμα, ο αλγόριθμος για την εξυπηρέτηση μεγάλου αριθμού

χρηστών που υπερτερούν τον αριθμό των αντικειμένων. Σε περίπτωση, που τα δεδομένα αλλάξουν και υπερτερούν τα αντικείμενα έναντι των χρηστών η απόδοση του συστήματος δεν θα είναι η βέλτιστη.

- Όταν οι στόχοι αξιολόγησης ενός συστήματος διαφέρουν.
- Όταν δεν αξιολογούνται όλοι οι παράγοντες. Τα περισσότερα συστήματα επικεντρώνονται στην ποιότητα των συστάσεων και όχι σε επιμέρους παράγοντες, όπως τις πραγματικές προτιμήσεις αγορών του χρήστη.
- Η αλλαγή προτιμήσεων των χρηστών και η διαφορετική εκτίμηση των αντικειμένων στο πέρασμα του χρόνου.
- Το πρόβλημα εύρεσης μονάδας μέτρησης αξιολόγησης των αλγορίθμων.

Αν και παρουσιάζονται διάφορες δυσκολίες στην προσπάθεια αξιολόγησης των συστημάτων, έχουν αναπτυχθεί αποτελεσματικοί μέθοδοι για την επίτευξη του στόχου. Κάποιοι από αυτούς είναι [22] :

Μέσο απόλυτο σφάλμα (Mean absolute error)

Ο υπολογισμός του μέσου απόλυτου σφάλματος (MAE), σχετίζεται με τη απόκλιση (deviation) της πραγματικής βαθμολογίας του χρήστη με αυτή που δόθηκε από το σύστημα. Η εξίσωση είναι η ακόλουθη:

$$MAE = \frac{\sum_{i=1}^{N} |p_i - r_i|}{N} \quad (8)$$

Όπου MAE είναι το μέσο απόλυτο σφάλμα, N το σύνολο των αντικειμένων για το οποίο γίνεται η πρόβλεψη, p_i η βαθμολογία του συστήματος για ένα αντικείμενο i και r_i η βαθμολογία του χρήστη για ένα αντικείμενο i.

Ρίζα του μέσου τετραγωνικού σφάλματος (Root Squared Mean Error - RMSE)

Η ρίζα του μέσου τετραγωνικού σφάλματος είναι παρόμοια της MAE και διαφοροποιείται στην περίπτωση μεγάλης απόκλισης. Και στις δύο μονάδες αξιολόγησεις, MAE και RMSE, οι μικρότερες μετρήσεις προβλέπουν μεγαλύτερη απόδοση. Η εξίσωση είναι η ακόλουθη:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}} \quad (9)$$

Όπου RMSE είναι η ρίζα του μέσου τετραγωνικού σφάλματος, N το σύνολο των αντικειμένων για το οποίο γίνεται η πρόβλεψη, $Predicted_i$ η βαθμολογία του συστήματος για ένα αντικείμενο i και $Actual_i$ η βαθμολογία του χρήστη για ένα αντικείμενο i.

Ακρίβεια (Precision)

Η μέτρηση αυτή υπολογίζει τη σχέση των σωστών συστάσεων που προτάθηκαν από τον αλγόριθμο και του συνόλου των προβαλλόμενων συστάσεων. Ο τύπος της ακρίβειας είναι :

$$P = \frac{N_{rs}}{N_s} \quad (10)$$

Όπου N_{rs} είναι οι συστάσεις που προτάθηκαν από τον αλγόριθμο και N_s το σύνολο των συστάσεων που έχουν επιλεγεί.. Όσο μεγαλύτερη υπολογίζεται η τιμή του P, τόσο καλύτερη είναι η προσέγγιση.

Ανάκληση (Recall)

Η μέτρηση της ανάκλησης υπολογίζει τη σχέση των σωστών συστάσεων που προτάθηκαν από τον αλγόριθμο και των συνολικών συστάσεων που βρίσκονται στο σύστημα. Όσο μεγαλύτερη υπολογίζεται η τιμή της ανάκλησης, τόσο καλύτερη προσέγγιση έχουμε.

Ο τύπος της ανάκλησης είναι:

$$R = \frac{N_{rs}}{N_r} \quad (11)$$

Όπου N_{rs} είναι οι συστάσεις που σωστά προτάθηκαν από τον αλγόριθμο και N_r το σύνολο των συστάσεων που υπάρχουν.

3. ΕΦΑΡΜΟΓΕΣ ΓΙΑ ΚΙΝΗΤΑ ΤΗΛΕΦΩΝΑ (MOBILE APPLICATIONS)

Οι εφαρμογές για κινητά (mobile applications) αποτελούν εφαρμογές λογισμικού που έχουν σχεδιαστεί για να λειτουργούν σε κινητές συσκευές, όπως τηλέφωνα και tablets. Παρουσιάζουν αρκετές ομοιότητες με τις υπηρεσίες υπολογιστών έχοντας, όμως, μικρότερο μέγεθος και περιορισμένη λειτουργία. Βασικός τους στόχος είναι η επέκταση της λειτουργικότητας των συσκευών για την καλύτερη εξυπηρέτηση των χρηστών. Τα λειτουργικά συστήματα που χρησιμοποιούνται, συχνότερα, για την κατασκευή μιας κινητής εφαρμογής είναι τα Android (Google), iOS (Apple Inc.) και Windows (Microsoft). Οι εφαρμογές τύπου Android, γίνονται διαθέσιμες στους χρήστες μέσω του Google Play, ενός διαδικτυακού καταστήματος λογισμικού με δισεκατομμύρια πελάτες, ενώ οι χρήστες της Apple Inc. λαμβάνουν εφαρμογές και υπηρεσίες από το αντίστοιχο διαδικτυακό κατάστημα App Store. Το App Store διαθέτει εφαρμογές τύπου iOS και αποτέλεσε επανάσταση στον τομέα λήψης και μεταφόρτωσης (download and upload) κινητών εφαρμογών με υπέρογκο αριθμό χρηστών. Για τις εφαρμογές των συσκευών Windows, χρησιμοποιείται το Microsoft Store ή Windows Store. Οι εφαρμογές τέτοιου τύπου κατασκευάζονται κυρίως για tablets και συσκευές αφής (touch - based devices), αλλά και γενικότερα για συσκευές φορητών και επιτραπέζιων υπολογιστών. Εκτός από τα διαδικτυακά καταστήματα κατανομής εφαρμογών που αναφέρθηκαν, υπάρχουν και άλλα όπως το το Amazon Appstore, ένα εναλλακτικό κατάστημα εφαρμογών για το λειτουργικό σύστημα Android, το Opera Mobile Store για εφαρμογές iOS, Android, Java, BlackBerry OS, Symbian και Windows Mobile, το Samsung Apps της Samsung για εφαρμογές κυρίως Android κ.α. [10].

3.1 Τύποι Εφαρμογών

Οι εφαρμογές για κινητά διακρίνονται σε τρεις τύπους, τους οποίους θα μελετήσουμε στο κεφάλαιο αυτό.

3.1.1 Εγγενείς Εφαρμογές (Native Applications)

Οι εγγενείς εφαρμογές αποτελούν προγράμματα σχεδιασμένα για μια συγκεκριμένη πλατφόρμα ή συσκευή, χρησιμοποιώντας συμβατές

γλώσσες και τεχνικές προγραμματισμού. Καθίστανται οι πιο αποδοτικές και αξιόπιστες από τα είδη εφαρμογών, έχοντας πολλαπλή πρόσβαση σε διάφορες λειτουργίες της συσκευής και παρέχοντας ασφάλεια και προστασία προς το κατάστημα εφαρμογών (App Store) που θα μεταφορτωθούν (upload). Οι εφαρμογές τέτοιου τύπου είναι ιδανικές για παιχνίδια και εφαρμογές υψηλής απόδοσης. Σημαντική είναι και η λειτουργία τους σε κατάσταση εκτός σύνδεσης (offline). Για την κατασκευή εγγενών εφαρμογών iOS, γίνεται χρήση των εφαρμογών Objective - C και Swift, ενώ για εφαρμογές τύπου Android οι γλώσσες προγραμματισμού Java ή Kotlin. Για τις εφαρμογές με Windows χρησιμοποιείται η C++ ή C#. Τα μειονεκτήματα των εγγενών εφαρμογών είναι κυρίως η αδυναμία υποστήριξης χρηστών διαφορετικών συσκευών και λειτουργικών συστημάτων που ενδέχεται να χρησιμοποιούν άλλες εκδόσεις της εφαρμογής, αλλά και η δαπανηρή και χρονοβόρα διαδικασία ανάπτυξής τους σε περιπτώσεις που δεν διαθέτει κάποιος τις κατάλληλες γνώσεις. Παράδειγμα εγγενούς εφαρμογής αποτελεί η εφαρμογή για κινητά Facebook, όπου αρχικά κατασκευάστηκε ώστε να είναι συμβατή με iOS, Android και τον ιστό για κινητά, με τον ίδιο κώδικα (HTML5). Η αδυναμία γρήγορης απόδοσης για τους χρήστες iOS, οδήγησε στην δημιουργία νέας έκδοσης της εφαρμογής για κάθε λειτουργικό σύστημα, iOS και Android, ξεχωριστά. Άλλα παραδείγματα εγγενών εφαρμογών είναι οι Google Maps, Uber, LinkedIn για iOS και Android, αντίστοιχα [13].
Στην παρούσα πτυχιακή εργασία, γίνεται κατασκευή της εγγενούς εφαρμογής Trevino, όπου ο τρόπος κατασκευής και λειτουργίας περιγράφεται παρακάτω.

Εικόνα 6.Περιγραφή της αλληλεπίδρασης των λειτουργιών της συσκευής με τις εγγενούς εφαρμογές.

3.1.2 Εφαρμογές Ιστού (Mobile Web Applications)

Οι εφαρμογές ιστού είναι ιστότοποι, όπου με διάφορες τεχνικές παρουσιάζονται στους χρήστες ως εγγενείς εφαρμογές, χωρίς να διαθέτουν τα ίδια χαρακτηριστικά με αυτές. Καθώς είναι διαδικτυακές εφαρμογές γράφονται σε HTML5, CSS και JavaScript. Με τις εφαρμογές ιστού δίνεται η δυνατότητα σύνδεσης σε οποιαδήποτε ιστοσελίδα και στη συνέχεια, η δημιουργία σελιδοδεικτών με τις προβαλλόμενες ιστοσελίδες που άρεσαν στους χρήστες. Με άλλα λόγια, οι εφαρμογές ιστού διευκολύνουν την πρόσβαση σε ιστοσελίδες, μέσω μιας κινητής εφαρμογής. Η οικοδόμησή τους γίνεται εύκολα, καθώς δίνεται η δυνατότητα χρήσης παραπάνω από μιας πλατφόρμας και η διαχείριση του κώδικα είναι απλή. Επίσης, είναι συμβατές με όλα τα λειτουργικά συστήματα και είναι απαραίτητη η σύνδεση των συσκευών στον ιστό (online) για την πλήρη λειτουργία της εφαρμογής. Σε περίπτωση που μια συσκευή δεν έχει πρόσβαση στον ιστό, ο χρήστης θα μπορεί να δει μόνο τα τελευταία στιγμιότυπα που φορτώθηκαν όταν βρισκόταν συνδεδεμένος, χωρίς τη δυνατότητα ανανέωσης. Οι εφαρμογές ιστού, όμως, δεν είναι τόσο αποδοτικές όσο οι εγγενείς και σε πολλές περιπτώσεις μη λειτουργικές σε κάποιες συσκευές. Ακόμη, παρουσιάζουν περιορισμένη εμπειρία χρήστη και διαδραστική χρησιμότητα. Παραδείγματα εφαρμογών ιστού για κινητά αποτελούν οι εφαρμογές ηλεκτρονικών καταστημάτων όπως το AliExpress, εφημερίδων και κοινωνικής δικτύωσης όπως το Tweeter Lite.

3.1.3 Υβριδικές Εφαρμογές (Hybrid Applications)

Οι υβριδικές εφαρμογές είναι εφαρμογές ιστού σε εγγενές πρόγραμμα περιήγησης, δημιουργώντας ένα πλαίσιο σύνδεσης και των δυο κατηγοριών εφαρμογών. Αναπτύσσονται με τη χρήση HTML5, CSS και Javascript - χαρακτηριστικό ανάπτυξης εφαρμογής ιστού - και στη συνέχεια συνδέονται σε μία εγγενή εφαρμογή. Η σύνδεση της εγγενούς εφαρμογής με την εφαρμογή ιστού γίνεται μέσω κάποιας πλατφόρμας όπως η Cordova, η οποία διαθέτει διάφορες επεκτάσεις (plugins) απαραίτητες για την εύρυθμη λειτουργία της υβριδικής εφαρμογής. Τέτοιες επεκτάσεις αποτελούν η κάμερα και το υλικό του τηλεφώνου (μικρόφωνο, οθόνη αφής κ.α.). Η υλοποίηση και σχεδίαση των εφαρμογών γίνεται εύκολα, γρήγορα και διατίθενται απλές τεχνικές για την αλλαγή πλατφόρμας όποτε χρειάζεται. Ένα σημαντικό πλεονέκτημα

των υβριδικών εφαρμογών απευθύνεται στους προγραμματιστές που θέλουν να ενημερώνουν συχνά την εφαρμογή τους. Σε περίπτωση που δεν είναι απαραίτητη η ενημέρωση ολόκληρου του εγγενούς κώδικα της εφαρμογής, δεν χρειάζεται η εκ νέου υποβολή νέας έκδοσης. Οι υβριδικές εφαρμογές επιτρέπουν την επαναχρησιμοποίηση του κώδικα τους εξοικονομώντας πόρους, χρόνο και χρήματα. Όσον αφορά τη λειτουργία τους, είναι πιο αργές σε σχέση με τις άλλες κατηγορίες εφαρμογών. Παραδείγματα υβριδικών εφαρμογών είναι το Instagram, όπου είναι δυνατή η υποστήριξη δεδομένων εκτός σύνδεσης και η πρόσβαση σε μέσα αναπαραγωγής όταν υπάρχει σύνδεση, το Pacifica, το Evernote κ.α.

3.2 Λειτουργικά Συστήματα Κινητών Τηλεφώνων (Mobile Operating Systems - Mobile OS)

Τα λειτουργικά συστήματα κινητών συσκευών αποτελούν συστήματα παρόμοια με αυτά των προσωπικών υπολογιστών, παρουσιάζοντας διαφορές σε λειτουργίες που δεν υποστηρίζονται ή από τους επιτραπέζιους υπολογιστές ή από τις κινητές συσκευές. Αν και οι φορητοί υπολογιστές αποτελούν ένα είδος κινητών συσκευών, τα λειτουργικά συστήματα για κινητά δεν απευθύνονται στη κατηγορία αυτή. Πιο συγκεκριμένα, αναφερόμαστε σε μια πλατφόρμα λογισμικού πάνω στην οποία γίνεται δυνατή η διαχείριση και η εκτέλεση των διαθέσιμων εφαρμογών της συσκευής (π.χ. συγχρονισμός εφαρμογών, πληκτρολόγιο αφής, ηλεκτρονικό ταχυδρομείο, κ.α.). Τρεις βασικές κατηγορίες λειτουργικών συστημάτων, είναι τα συστήματα Android, iOS και Windows Phone, καθώς και διάφορες υποκατηγορίες των συστημάτων αυτών που σχεδιάστηκαν για τη χρήση σε συγκεκριμένες κινητές συσκευές. Τα λειτουργικά συστήματα χωρίζονται σε δυο κατηγορίες, στα συστήματα ανοιχτού κώδικα (open source) και κλειστού κώδικα (closed source).

Τα συστήματα με ανοιχτό κώδικα, δίνουν την άδεια χρήσης του πηγαίου κώδικα και των περιεχομένων ενός προϊόντος, ενθαρρύνοντας την ανάπτυξη εφαρμογών και την ανοιχτή συνεργασία. Η ανάγκη για μοντέλα ανοιχτού κώδικα, ξεκίνησε για την αντιμετώπιση περιορισμών που έθεταν οι αποκλειστικοί κώδικες προϊόντων, καθώς δεν ήταν διαθέσιμοι στο κοινό για τυχόν αλλαγές και τροποποιήσεις. Έτσι, μπόρεσε να εκτιμηθεί καλύτερα η ποιότητα του λογισμικού και να

διορθωθούν σφάλματα που πιθανόν να μην γίνονταν ορατά από τους κατασκευαστές. Το μοντέλο αυτό, χρησιμοποιείται και στο λειτουργικό σύστημα Android. Από την άλλη μεριά, τα συστήματα κλειστού κώδικα αφορούν προγράμματα ή προϊόντα, όπου ο πηγαίος κώδικας τους δεν είναι διαθέσιμος στο κοινό. Το μοντέλο αυτό δεν μπορεί να κοινοποιηθεί αλλά ούτε και να τροποποιηθεί. Συνήθως, τα μοντέλα ανοιχτού κώδικα είναι ελεύθερα και δωρεάν. Αυτό δεν συμβαίνει στην περίπτωση του κλειστού κώδικα, όπου ακόμη και αν διατίθεται δωρεάν, ο κώδικας του προγράμματος δεν είναι ίδιος με τον πηγαίο του κώδικα και δεν μπορεί να μεταποιηθεί [10]. Χρήση του μοντέλου κλειστού κώδικα γίνεται από το λειτουργικό σύστημα iOS.

3.2.1 Τύποι Λειτουργικών Συστημάτων

Στο κεφάλαιο αυτό, γίνεται περιγραφή των διαφόρων τύπων λειτουργικών συστημάτων για κινητές συσκευές.

- **Android OS (Google Inc.)**

Το λειτουργικό σύστημα Android, σχεδιάστηκε αρχικά από την εταιρία Silicon Valley με την επωνυμία Android Inc. και αποτελεί το πιο γνωστό λειτουργικό σύστημα στο χώρο των πωλήσεων. Η συνεργασία με την Google μέσω του ανοικτού και ελεύθερου λογισμικού της, έδωσε την δυνατότητα στο Android να γίνει ένα ολοκληρωμένο λειτουργικό σύστημα, με κύριο και μεσαίο λογισμικό καθώς και πλήθος προεπιλεγμένων εφαρμογών, όπως ηλεκτρονικό ταχυδρομείο (Gmail), χάρτες (Google Maps), εφαρμογή τηλεφώνου κ.α. Ακόμη, το Android θεωρείται μοντέλο ανοιχτού κώδικα και έχει σχεδιαστεί βάσει το πυρήνα του Linux. Το δομικό μοντέλο του Android με τη μορφή μιας στοίβας λογισμικού αποτελείται από εφαρμογές, λειτουργικό σύστημα, περιβάλλον χρόνου εκτέλεσης, μεσαίο λογισμικό υπηρεσίες και βιβλιοθήκες. Κάθε επίπεδο, αλληλεπιδρά με τα υπόλοιπα ώστε να γίνεται ανάπτυξη και εκτέλεση εφαρμογών. Εφαρμογές Android μπορούν να αναπτυχθούν εξίσου σε διαφορετικά περιβάλλοντα, όπως τα Windows, Linux και Mac, με τη βοήθεια κατάλληλων εργαλείων (Android SDK - Software Development Kit). Στις εφαρμογές Android (Android Apps), χρησιμοποιείται κυρίως η γλώσσα προγραμματισμού Java. Για τις ενημερώσεις του λειτουργικού συστήματος, γίνεται ανάπτυξη εκδόσεων ανοιχτού κώδικα, λαμβάνοντας τις ονομασίες αλφαβητικά.

Code name	Version number	Linux kernel version[1]	Initial release date	API level
(No codename)	1.0	?	September 23, 2008	1
Petit Four	1.1	2.6	February 9, 2009	2
Cupcake	1.5	2.6.27	April 27, 2009	3
Donut	1.6	2.6.29	September 15, 2009	4
Eclair	2.0 – 2.1	2.6.29	October 26, 2009	5 – 7
Froyo	2.2 – 2.2.3	2.6.32	May 20, 2010	8
Gingerbread	2.3 – 2.3.7	2.6.35	December 6, 2010	9 – 10
Honeycomb	3.0 – 3.2.6	2.6.36	February 22, 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.4	3.0.1	October 18, 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	3.0.31 to 3.4.39	July 9, 2012	16 – 18
KitKat	4.4 – 4.4.4	3.10	October 31, 2013	19 – 20
Lollipop	5.0 – 5.1.1	3.16	November 12, 2014	21 – 22
Marshmallow	6.0 – 6.0.1	3.18	October 5, 2015	23
Nougat	7.0 – 7.1.2	4.4	August 22, 2016	24 – 25
Oreo	8.0 – 8.1	4.10	August 21, 2017	26 – 27
Pie	**9.0**	4.4.107, 4.9.84, and 4.14.42	August 6, 2018	28
Android Q	10.0			29
Legend: Old version	Older version, still supported	**Latest version**	Latest preview ve	

Εικόνα 7. Πίνακας με όλες τις εκδόσεις του Android [10].

- **iOS (Apple Inc.)**

Το λειτουργικό σύστημα iOS, αναπτύχθηκε και σχεδιάστηκε από την Apple Inc. και έγινε ιδιαίτερα γνωστό για την αλληλεπίδραση του υλικού με τον χρήστη, εξυπηρετώντας καλύτερα τις ενέργειες των δακτύλων (multi - touch) σε σχέση με τη συσκευή. Για την επίτευξη αυτού του στόχου, οι ενέργειες εκτελούνται σε οθόνες αφής, υψηλής χωρητικότητας και γρήγορης απόκρισης. Το iOS ανήκει στην κατηγορία λειτουργικών συστημάτων κλειστού κώδικα (closed source), αποτελεί μια νεότερη έκδοση του Mac OS X και μοιάζει με το λειτουργικό σύστημα UNIX. Χωρίζεται σε τέσσερα επίπεδα και διαθέτει αρκετές προεπιλεγμένες εφαρμογές, όπως το ηλεκτρονικό ταχυδρομείο, το Safari Web κ.α. Τα επίπεδα στα οποία χωρίζεται είναι:

Βασικό επίπεδο λειτουργικού συστήματος (Core OS Layer)
Αποτελεί το χαμηλότερο επίπεδο παρέχοντας πληροφορίες και περιορισμούς για προστασία και αλληλεπίδραση με το υλικό (hardware).

Επίπεδο βασικών πληροφοριών (Core Services Layer)
Παρέχει τις υπηρεσίες που χρειάζονται τα ανώτερα επίπεδα.

Επίπεδο Μέσων Αναπαραγωγής (Media Layer)
Παρέχει μεθόδους για την αναπαραγωγή γραφικών, ήχου και βίντεο.

Επίπεδο αφής (Cocoa Touch Layer)
Παρέχονται τα πλαίσια (frameworks), στα οποία βασίζεται η δημιουργία εφαρμογών.

Για την ανάπτυξη εφαρμογών iOS υπάρχουν διαθέσιμα εργαλεία ανάπτυξης λογισμικού - iOS SDK (Software Development Kit) που περιλαμβάνουν τις κατάλληλες διεπαφές ανάπτυξης, εγκατάστασης και εκτέλεσης. Στις εγγενείς εφαρμογές iOS (native iOS apps) χρησιμοποιείται κυρίως η γλώσσα προγραμματισμού Objective - C. Οι εφαρμογές τέτοιου τύπου είναι συμβατές μόνο με κινητές συσκευές της Apple (iPhone, iPad, iPod Touch).

Version	Build	Processor support	Application support	Kernel	Release date	Device end-of-life		
						iPad	iPhone	iPod Touch
3.1.3	7E18	32-bit ARM			Feb 2, 2010	N/A	1st gen	1
4.2.1	8C148				Nov 22, 2010		3G	2
5.1.1	9B206				May 7, 2012	1st gen	N/A	3
6.1.6	10B500				Feb 21, 2014	N/A	3GS	4
7.1.2	11D257	32/64-bit ARM[1][2]			Jun 30, 2014		4	N/A
9.3.5	13G36				Aug 25, 2016	2, 3, Mini 1	4S	5
10.3.3	14G60				Jul 19, 2017	4	5, 5C	N/A
12.2	16E227	64-bit ARM[3]			Mar 25, 2019	N/A		
12.3 Beta 1	16F5117h				Mar 27, 2019			

Legend: Discontinued **Current** Beta

Εικόνα 8. Πίνακας παρουσίασης εκδόσεων iOS κατά χρονολογική σειρά.

- **Windows Phone και Windows 10 Mobile (Microsoft)**

Το Windows Phone είναι ένα λειτουργικό σύστημα κλειστού κώδικα σχεδιασμένο από την Microsoft. Στόχος της εταιρίας, είναι η επίτευξη της καλύτερης δυνατής εμπειρίας χρήστη, μέσω της αυστηρής αδειοδότησης χρήσης του λογισμικού από τρίτους κατασκευαστές. Το Windows Phone αντικατέστησε το Windows Mobile και Zune, παλαιότερες εκδόσεις του λειτουργικού. Το 2011 η εταιρία της Nokia επέλεξε την έκδοση της Windows Phone, ως λειτουργικό σύστημα σε όλες τις κινητές συσκευές της (smartphones) παρέχοντας σταθερή υποστήριξη. Αργότερα, η Windows Phone αντικαταστάθηκε από την νεότερη έκδοση της Windows 10 Mobile. Τα λειτουργικά συστήματα της Microsoft, αποτελούν προσομοίωση των αντίστοιχων λειτουργικών της για σταθερούς και φορητούς υπολογιστές, ανάλογα προσαρμοσμένων ώστε να εξυπηρετούν τις ανάγκες των κινητών συσκευών (smartphones και tablets). Η Windows 10 Mobile έχει σχεδιαστεί έτσι, ώστε να εκτελείται σε συσκευές με οθόνη μικρότερη των εννέα ιντσών (inch screen) με σκοπό την ενοποίηση των

πολλαπλών κατηγοριών των Windows [14]. Σημαντική, αποτελεί και η ανάπτυξη εργαλείων για προγραμματιστές, προκειμένου να γίνεται εύκολα η μεταφορά εφαρμογών iOS με ελάχιστες τροποποιήσεις. Παρόλα αυτά, το Windows 10 Mobile δεν ξεπέρασε σε πωλήσεις τις συσκευές με λειτουργικά συστήματα Android και iOS. Οι ενημερώσεις των εκδόσεων Windows Phone και Windows 10 Mobile παρουσιάζονται παρακάτω :

Windows Phone 7 : Φεβρουάριος, 2010
Windows Phone 7.5 Mango : Μάιος, 2011
Windows Phone 7.5 Tango : 2012
Windows Phone 7.8 : 2013
Windows Phone 8 : Απρίλιος, 2014
Windows 10 : Version 1507 "10.0.10240.x", αρχική έκδοση
Windows 10 : Version 1511 "10.0.10586.x"
Windows 10 : Version 1607 "10.0.14393.x"
Windows 10: Version 1703 "10.0.15063.x"
Windows 10: Version 1709 "10.0.16299.x"
Windows 10: Version 1803 "10.0.17134.x"

Πέρα από τις βασικές πλατφόρμες λογισμικού έχουν δημιουργηθεί και διάφορα συνεργαζόμενα λειτουργικά συστήματα για την εξυπηρέτηση των αναγκών εταιριών κινητής τηλεφωνίας.

- **BlackBerry Secure (BlackBerry)**

Το BlackBerry Secure είναι ένα λειτουργικό σύστημα της BlackBerry που στηρίχθηκε στο μοντέλο ανοιχτού κώδικα της Android (Android Open Source Project - AOSP). Το λειτουργικό αυτό χρησιμοποιήθηκε σε συσκευές με διεπαφές αφής. Πριν από την συνεργασία της BlackBerry με την Android, το BlackBerry Secure χρησιμοποιούταν σε συσκευές όπως οι BlackBerry Priv, DTEK 50/60 and BlackBerry KEYone. Η BlackBerry έδωσε έμφαση στην ασφάλεια των δεδομένων, χρησιμοποιώντας πυρήνα Linux για να το εξασφαλίσει, καθώς και διάφορα συστήματα κρυπτογράφησης. Το λειτουργικό σύστημα της εταιρίας, την καθιστά την ασφαλέστερη επιλογή για την προστασία δεδομένων και επιλέγεται συνήθως από επιχειρήσεις. Η έκδοση του BlackBerry Secure σε συνεργασία με την Android είναι η BlackBerry Secure version 1.x (βασισμένο στην έκδοση Android "Marshmallow" 6.x and "Nougat" 7.x).

	Android	iOS	Blackberry	Windows
Vendor	Open Handset Alliance	Apple	Blackberry Ltd.	Microsoft
Symbol				
Official Site	www.android.com\	www.apple.com\ios\	Us.blackberry.com/apps-software/blackberry7/	www.windowsphone.com\
Developed in programming Language	C, C++, Java	C, C++, Objective-C, Swift	C, C++, HTML5,Javascript,CSS, ActionScript,Java	C#, VB.NET, F#, C++, Jscript
License	Open source	Proprietary	Proprietary	Proprietary
App Store	Google Play	App Store	BlackBerry World	Windows Phone Store
No. of App	1.3 million	1.2 million	234,500	500,000
Side loading	Available	Done by installing Xcode7	----	Available with windows phone 10, not in earlier version
Battery Demand	Highest	Less	Moderate	Least
Customizability	Highest	Provide few option (allows a few selected widgets to be applied on the notification panel)	User can change the options in the notification panel, and can add a few more shortcuts in it	Allow re-sizable live tiles, various colour schemes can be chosen in addition to background images
Security	Softest to crack	Hard to crack	Hard to crack	Hardest to crack
Voice Assistance	Google now	Siri	Blackberry voice assistance	Cortana

Εικόνα 9. Πίνακας σύγκρισης λειτουργικών συστημάτων.

3.3 Ανάπτυξη Android Εφαρμογής για Κινητά Τηλέφωνα

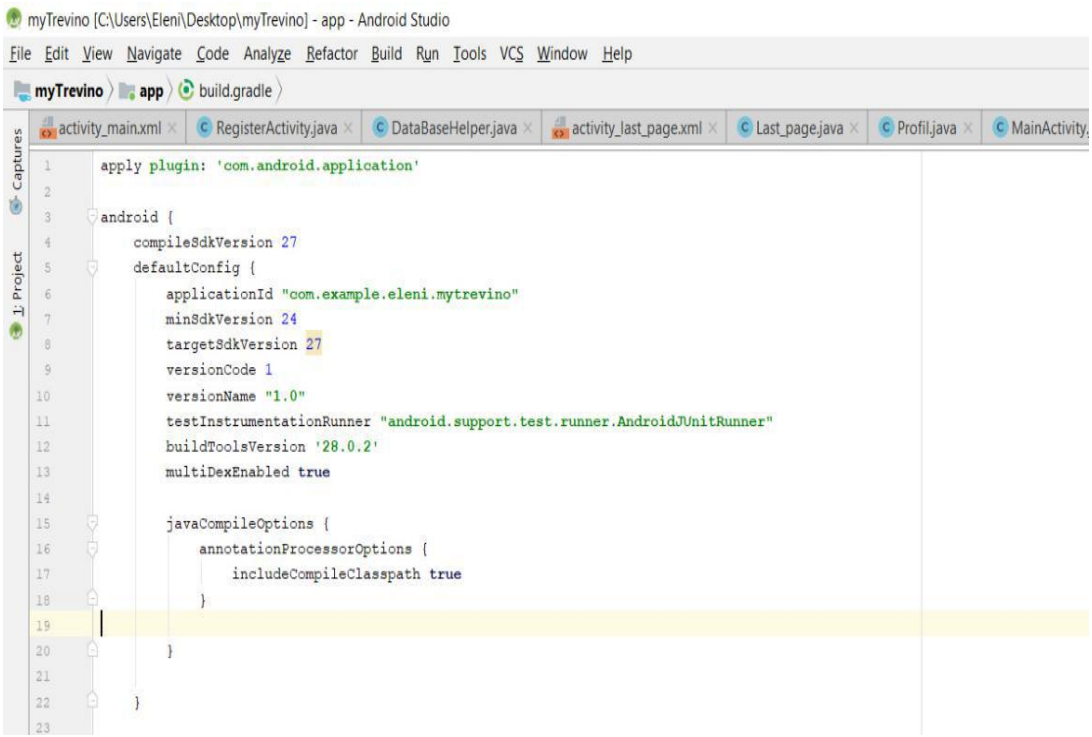
Για την κατασκευή εφαρμογών τύπου Android χρησιμοποιούνται πλατφόρμες, όπως το Android Studio, η Eclipse, η ShoutEm, η AppMakr, η AppMachine κ.α. Το Android Studio αποτελεί περιβάλλον ανάπτυξης αποκλειστικά για εφαρμογές Android, βασίζεται στο λογισμικό IntelliJ IDEA της JetBrains και καταστάθηκε από την Google ως η επίσημη πλατφόρμα κατασκευής Android εφαρμογών. Η Eclipse χρησιμοποιώντας τα εργαλεία Android Development Tools (ADT), υποστηρίζει επίσης την ανάπτυξη Android εφαρμογών με χρήση της γλώσσας προγραμματισμού Java. Η ShoutEm αποτελεί μια πλήρη λύση για την ανάπτυξη εφαρμογών με τη δυνατότητα ενσωμάτωσης της νέας εφαρμογής σε υπάρχουσες. Οι εφαρμογές, μέσω της πλατφόρμας μπορούν να δημοσιευθούν στο Google Play store, καθώς και να ενημερώνονται σε πραγματικό χρόνο αυτόματα. Η AppMakr παρουσιάζει μια εναλλακτική μορφή δημιουργίας εφαρμογών, όπου δεν απαιτείται η γνώση προγραμματισμού (coding) και αποτελεί μια από τις μεγαλύτερες πλατφόρμες εκδόσεων DIY παγκοσμίως. Ακόμη, η AppMachine χρησιμοποιείται για την κατασκευή επαγγελματικών εγγενών εφαρμογών με δυνατότητες μεταφοράς περιεχομένου

ιστοτόπων, επιλογής τρόπου σχεδιασμού και προώθησης στο Google Play [15].

Η εφαρμογή Trevino αναπτύχθηκε στα πλαίσια της εργασίας με τη χρήση των Android Studio και Eclipse.

3.3.1 Περιγραφή και Ανάπτυξη Εφαρμογής Trevino στο Android Studio

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε το Android Studio Version 3.2.1 και το JRE - Java Version - 1.8.0. Ο σχεδιασμός της εφαρμογής, έγινε σε διεπαφή προγραμματισμού εφαρμογών (API) 28 και η προσομοίωση σε συσκευή Nexus 4. Η εκτέλεση της εφαρμογής γίνεται σε συσκευή Samsung Galaxy A5 2017 με Android έκδοσης 8.0.0.



Εικόνα 10. build.gradle της εφαρμογής Trevino

Η εφαρμογή Trevino παρέχει προτάσεις διαδρομών, ανάλογα με τον αρχικό και τον τελικό προορισμό του χρήστη. Το μονοπάτι των διαδρομών αποτελείται από δρομολόγια αεροπλάνων, τρένων και του συνδυασμού τους. Με την εγγραφή ενός χρήστη στην εφαρμογή καλείται να συμπληρώσει μεταξύ άλλων και την ηλικία του, που παίζει πρωταρχικό ρόλο στο είδος των προτάσεων που θα λάβει.

Για την υλοποίηση του στόχου αυτού, κατασκευάστηκε μια **βάση δεδομένων (database)** ονόματος Trevino, που αποτελείται από πέντε πίνακες (tables) τους User, User_Travel_info, Destinations, Aeroplane, Train που περιγράφονται παρακάτω:

- **User Table**

Ο πίνακας αυτός περιέχει πέντε (5) στήλες (columns) τις ID, Username, Email, Password, Age. Το ID αριθμείται αυτόματα και είναι μοναδικό για κάθε χρήστη. Ακόμη, στη στήλη Username αποθηκεύεται το όνομα χρήστη και στις στήλες email και password, η διεύθυνση ηλεκτρονικού ταχυδρομείου και ο κωδικός για την είσοδο κάθε χρήστη, αντίστοιχα. Στο Age αποθηκεύεται η ηλικία του χρήστη.

- **User_Travel_info Table**

Ο πίνακας αυτός αποτελείται από έξι (6) στήλες τις ID, Start_Destination, Final_Destination, Date, Trip_Title και ID_User. Το ID αριθμεί το νούμερο του χρήστη που είναι εγγεγραμμένος στην εφαρμογή και η αρίθμηση γίνεται αυτόματα. Στις στήλες Start_Destination και Final_Destination αποθηκεύονται αντίστοιχα, ο αρχικός και ο τελικός προορισμός που επιθυμεί ο χρήστης. Στις Date και Trip_Title η ημερομηνία του ταξιδιού και το όνομα που επιθυμεί να δώσει στο ταξίδι που θα πραγματοποιήσει. Στη στήλη ID_User αποθηκεύεται το ID του χρήστη που δόθηκε κατά την εγγραφή του (Registration) στην εφαρμογή, δηλαδή το ID που τον αντιπροσωπεύει στο User πίνακα. Έτσι, γίνεται ορατός ο διαχωρισμός των διαδρομών για κάθε χρήστη στη βάση δεδομένων.

- **Destinations Table**

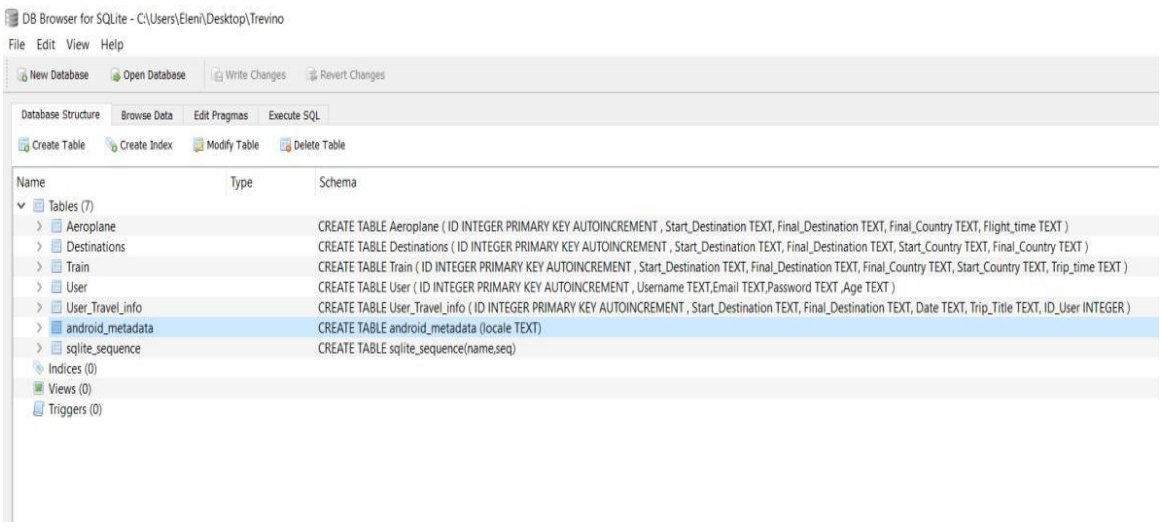
Στον πίνακα υπάρχουν όλοι οι πιθανοί προορισμοί που μπορεί να ταξιδέψει ο χρήστης. Αποτελείται από πέντε (5) στήλες τις ID, Start_Destination, Final_Destination, Start_Country και Final_Country. Στις στήλες αυτές, αποθηκεύεται το ID όπου αποθηκεύεται αυτόματα και είναι μοναδικό για κάθε προορισμό, καθώς και ο αρχικός και τελικός προορισμός στο Start_Destination και Final_Destination. Στις στήλες Start_Country και Final_Country αποθηκεύνται οι χώρες αρχικού και τελικού προορισμού. Οι δυο τελευταίες στήλες δεν χρησιμοποιούνται έμπρακτα στην λειτουργία της εφαρμογής, αλλά παρέχουν μια πιο ολοκληρωμένη εικόνα των προορισμών στη βάση δεδομένων. Ο πίνακας αυτός περιέχει εβδομήντα δύο (72) εγγραφές πιθανών προορισμών.

- **Aeroplane Table**

Εδώ περιέχονται όλα τα δρομολόγια με αεροπλάνο. Ο πίνακας αποτελείται από τέσσερις (4) στήλες τις ID, Start_Destination, Final_Destination και Flight_time. Στις στήλες ID, Start_Destination και Final_Destination αποθηκεύονται το ID κάθε διαδρομής όπου είναι μοναδικό,καθώς και ο αρχικός και τελικός προορισμός, αντίστοιχα. Στο Flight_time αποθηκεύεται ο χρόνος πτήσης κάθε διαδρομής. Στον πίνακα έχουν καταχωρηθεί πενήντα (50) εγγραφές διαδρομών με αεροπλάνο.

- **Train Table**

Στον πίνακα αυτό, περιέχονται τα δρομολόγια των τρένων και αποτελείται από τέσσερις (4) στήλες. Τις ID, Start_Destination και Final_Destination όπου αποθηκεύονται το ID κάθε διαδρομής όπου είναι μοναδικό, καθώς και ο αρχικός και ο τελικός προορισμός, αντίστοιχα. Στη στήλη Trip_time αποθηκεύεται ο χρόνος που διαρκεί η διαδρομή με τρένο. Στον πίνακα έχουν καταχωρηθεί τριάντα τέσσερις (34) εγγραφές διαδρομών. Η δημιουργία της βάσης δεδομένων και των πινάκων γίνεται στη κλάση DataBaseHelper. Αν και η βάση δεδομένων ενημερώνεται στο Android Studio η επεξεργασία των πληροφοριών και η παροχή προτάσεων γίνεται στη πλατφόρμα της Eclipse. Η διαδικασία αυτή περιγράφεται στο κεφάλαιο 3.3.2. Σημαντικό είναι επίσης να αναφερθεί, ότι η βάση δεδομένων αποθηκεύεται απευθείας στη συσκευή. Για την καταχώρηση των στοιχείων στη βάση δεδομένων χρησιμοποιήθηκαν οι κλασεις User (Εικόνα 13), UserInfo, Aeroplane, Train και Destination.



Εικόνα 11. Πίνακες βάσης δεδομένων

Εικόνα 12. Δημιουργία πινάκων στη βάση δεδομένων.

```java
public class User {
    public String id;
    public String email, username;
    public String password;
    public String age;


    public String getId() { return id; }

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

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }

    public String getPassword() { return password; }

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

    public String getUsername() { return username; }

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

    public String getAge() { return age; }

    public void setAge(String age) { this.age = age; }
}
```

Εικόνα 13. Η κλάση User.

Από τον σύνολο των πινάκων της βάσης δεδομένων, μόνοι οι User και User_Travel_info αποθηκεύουν στοιχεία που παρέχει ο χρήστης.

myTrevino [C:\Users\Eleni\Desktop\myTrevino] - ...\app\src\main\java\com\example\eleni\mytrevino\sql\DataBaseHelper.java [app] - Android Studio

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

myTrevino > app > src > main > java > com > example > eleni > mytrevino > sql > DataBaseHelper

activity_main.xml | MainActivity.java | DataBaseHelper.java | Aeroplane.java | Profil.java

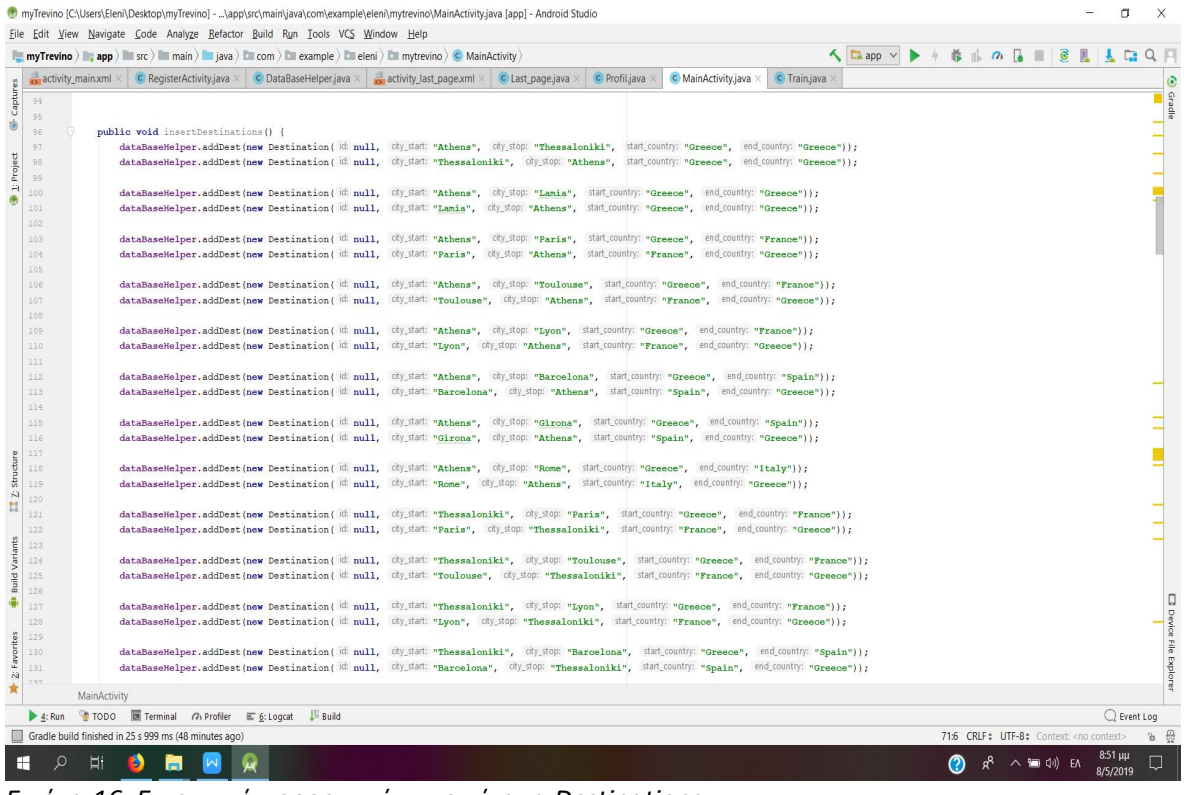
```java
public boolean add(User user)
{
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();

    values.put(COLUMN_USER_USERNAME, user.username);
    values.put(COLUMN_USER_EMAIL, user.email);
    values.put(COLUMN_USER_PASSWORD, user.password);
    values.put(COLUMN_USER_AGE, user.age);
    long todo = db.insert(TABLE_USER, nullColumnHack: null, values);
    if (todo == -1) return false;
    else return true;

}
public boolean addInfo(UserInfo userInfo ) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();

        values.put(COLUMN_INFO_START, userInfo.city_start);
        values.put(COLUMN_INFO_END, userInfo.city_stop);
        values.put(COLUMN_INFO_DATE, userInfo.date);
        values.put(COLUMN_INFO_TITLE, userInfo.trip_name);
        values.put(USER_ID,userInfo.id_user);

        long todo = db.insert(TABLE_USER_INFO, nullColumnHack: null, values);
        if (todo == -1) return false;
        else return true;
}
```

Εικόνα 14. Συναρτήσεις προσθήκης χρηστών και προσθήκης λεπτομερειών προορισμού.

```
public boolean addPlane (Aeroplane plane)
{
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();

    values.put(COLUMN_PLANE_START, plane.getPlane_city_start());
    values.put(COLUMN_PLANE_END, plane.getPlane_city_stop());
    values.put(COLUMN_PLANE_FLIGHT_TIME, plane.getFlight_time());

    long todo = db.insert(TABLE_PLANE,  nullColumnHack: null, values);
    if (todo == -1) return false;
    else return true;
}


public boolean addTrain (Train train)
{
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();

    values.put(COLUMN_TRAIN_START, train.getTrain_city_start());
    values.put(COLUMN_TRAIN_END, train.getTrain_city_stop());
    values.put(COLUMN_TRAIN_TRIP_TIME, train.getTrip_time());

    long todo = db.insert(TABLE_TRAIN,  nullColumnHack: null, values);
    if (todo == -1) return false;
    else return true;
}
```

Εικόνα 15. Συναρτήσεις προσθήκης αεροπλάνων και τρένων στη βάση δεδομένων.

Στην κλάση MainActivity ενημερώνονται όλα τα δρομολόγια του πίνακα Destinations.



Εικόνα 16. Εισαγωγή προορισμών στο πίνακα Destinations.

Παρόμοια, γίνεται και η εισαγωγή στοιχείων και για τους πίνακες Aeroplane και Train της βάσης δεδομένων.

3.3.1.1 Σύνδεση και Εγγραφή στην Εφαρμογή (Login - Register)

Στη πρώτη σελίδα έχουμε την σύνδεση του χρήστη, αφού πρώτα έχει προηγηθεί η εγγραφή του στην εφαρμογή. Κατά την εγγραφή ο χρήστης δεν μπορεί να καταχωρήσει email που έχει ήδη καταχωρηθεί παλαιότερα στην βάση δεδομένων, έτσι ώστε να μπορεί να συνδεθεί αξιόπιστα. Για την υλοποίηση της σύνδεσης χρησιμοποιήθηκε η κλάση MainActivity και για τον σχεδιασμό της σελίδας η activity_main.xml. Ακόμη, για την εγγραφή χρησιμοποιήθηκε η κλάση RegisterActivity, καθώς και για τον σχεδιασμό της σελίδας εγγραφής η activity_register.xml. Στην σελίδα της σύνδεσης, ο χρήστης καλείται να δώσει το email και τον κωδικό του ώστε να περάσει στην επόμενη σελίδα. Στις παρακάτω εικόνες απεικονίζεται ο κώδικας για την σάρωση της βάσης δεδομένων με σκοπό την αντιστοιχία μεταξύ του email και του κωδικού που δόθηκε. Αν βρεθεί, η σύνδεση πετυχαίνει. Σε αντίθετη περίπτωση, εμφανίζεται μήνυμα λάθους, ώστε ο χρήστης να ξαναπροσπαθήσει ή να εγγραφεί.

```
public Boolean Authenticate(String email1,String password1) {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery( sql: "select * from User where password=?  and email=?", new String[]{password1, email1});

    if (cursor.getCount() > 0) {
        return true;
    } else {
        return false;
    }
}
public boolean isEmailExists(String email) {
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.rawQuery( sql: "Select * from User where email=?", new String[]{email});

    if (cursor.getCount() > 0) return false;
    else {
        return true;
    }
}
```

Εικόνα 17. Εύρεση email και κωδικού στη βάση δεδομένων για την επιβεβαίωση του χρήστη και έλεγχος ύπαρξης email νέου χρήστη κατά την εγγραφή - DataBaseHelper.

```
button_login.setOnClickListener((view) → {

        email1 = email.getText().toString();
        password1 = password.getText().toString();

        boolean auth = dataBaseHelper.Authenticate(email1, password1);
        if (auth == true) {
            Toast.makeText(getApplicationContext(), text: "Successfully Logged in!", Toast.LENGTH_LONG).show();
            Intent intent = new Intent( packageContext: MainActivity.this, HomePageActivity.class);
            startActivity(intent);
        } else {
            Toast.makeText(getApplicationContext(), text: "Failed to log in , please try again", Toast.LENGTH_LONG).show();

        }
});
```

Εικόνα 18. Έλεγχος σύνδεσης χρήστη - MainActivity.

Για τον σχεδιασμό της πρώτης σελίδας Login και Register χρησιμοποιήθηκε ως επιφάνεια εργασίας εικόνα 1.280x868 JPEG (24 - bit - color). Για τη σελίδα εγγραφής, χρησιμοποιήθηκαν δύο EditTexts (email1, password), ένα κουμπί σύνδεσης (button_login), καθώς και ένα TextView που προτείνει στο χρήστη να εγγραφεί, στη περίπτωση που δεν έχει ακόμη εγγραφεί.

```
private void initCreateAccountTextView() {
    TextView createAccount = (TextView) findViewById(R.id.createAccount);
    createAccount.setText(fromHtml( source: "<font color='#ffffff'>I don't have account yet. </font><font color='#99ccff'>create one</font>"));
    createAccount.setOnClickListener((view) → {
            Intent intent = new Intent( packageContext: MainActivity.this, RegisterActivity.class);
            startActivity(intent);
    });
}

//this method is used to connect XML views to its Objects
private void initViews() {
    email = (EditText) findViewById(R.id.email);
    password = (EditText) findViewById(R.id.password);
    emailLayout = (TextInputLayout) findViewById(R.id.startLayout);
    passwordLayout = (TextInputLayout) findViewById(R.id.passwordLayout);
    button_login = (Button) findViewById(R.id.button_login);
}
```

Εικόνα 19. Συναρτήσεις σύνδεσης κλάσης με το αρχείο xml.

Η κλάση RegisterActivity χρησιμοποιείται για τη καταχώρηση του χρήστη κατά την εγγραφή του στην εφαρμογή. Στην περίπτωση, που δεν συμπληρωθούν όλα τα πλαίσια ή ο χρήστης υπάρχει ήδη, δεν είναι εφικτή η εγγραφή του.

```
register_button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String username1 = username.getText().toString();
        String email1 = email.getText().toString();
        String password1 = password.getText().toString();
        String age1 = age.getText().toString();
        if (username1.equals("") || email1.equals("") || password1.equals("") || !android.util.Patterns.EMAIL_ADDRESS.matcher(email1).matches()) {
            Toast.makeText(getApplicationContext(), text: "No valid Fields", Toast.LENGTH_LONG).show();
        } else {
            if (dataBaseHelper.isEmailExists(email1) == true) {
                User user=new User();
                user.setId(null);
                user.setUsername(username1);
                user.setPassword(password1);
                user.setEmail(email1);
                user.setAge(age1);
                boolean add = dataBaseHelper.add(user);

                if (add == true) {

                    Toast.makeText(getApplicationContext(), text: "User created successfully! Please Login ", Toast.LENGTH_LONG).show();

                    new Handler().postDelayed(() -> {
                            finish();
                        }, Toast.LENGTH_LONG);
                } else {

                    //Email exists with email input provided so show error user already exist
                    Toast.makeText(getApplicationContext(), text: "User already exists with same email ", Toast.LENGTH_LONG).show();
                }
            }
        }
    }
});
```

Εικόνα 20. Εισαγωγή νέου χρήστη.

```
<TextView
    android:id="@+id/register"
    android:layout_width="167dp"
    android:layout_height="45dp"
    android:layout_marginStart="7dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="250dp"
    android:layout_marginBottom="467dp"
    android:text="Register"
    android:textColor="@color/colorPrimary"
    android:textSize="30sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.0" />

<android.support.design.widget.TextInputLayout
    android:id="@+id/usernameLayout"
    android:layout_width="235dp"
    android:layout_height="55dp"
    android:layout_marginStart="10dp"
    android:layout_marginTop="150dp"
    android:layout_marginEnd="151dp"
    android:ems="10"
    android:textColorHint="@color/colorPrimary"
    app:layout_constraintBottom_toTopOf="@+id/register"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.42"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/register"
    app:layout_constraintVertical_bias="0.49">
```

```
        <android.support.design.widget.TextInputEditText
            android:id="@+id/username"
            android:layout_width="206dp"
            android:layout_height="wrap_content"
            android:hint="Username"
            android:inputType="textPersonName"
            android:textAppearance="@style/TextAppearance.AppCompat"
            android:textColor="@color/colorPrimary"
            android:textSize="18sp"
            android:textStyle="bold" />
    </android.support.design.widget.TextInputLayout>

    <android.support.design.widget.TextInputLayout
        android:id="@+id/startLayout"
        android:layout_width="235dp"
        android:layout_height="55dp"
        android:layout_marginStart="20dp"
        android:layout_marginEnd="151dp"
        android:layout_marginBottom="301dp"
        android:ems="10"
        android:textColorHint="@color/colorPrimary"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/usernameLayout">

        <android.support.design.widget.TextInputEditText
            android:id="@+id/email"
            android:layout_width="206dp"
            android:layout_height="wrap_content"
            android:hint="Email"
            android:inputType="textEmailAddress"
            android:textColor="@color/colorPrimary"
            android:textSize="18sp"
            android:textStyle="bold" />
    </android.support.design.widget.TextInputLayout>
```

Εικόνα 21. activity_register.xml

3.3.1.2 Επιλογή Προορισμού

Στη σελίδα αυτή, ο χρήστης καλείται να δηλώσει τον αρχικό και τελικό προορισμό, την ημερομηνία που θα ταξιδέψει, αλλά και προαιρετικά ένα όνομα για το ταξίδι. Οι πληροφορίες αυτές, αποθηκεύονται στο User_Travel_info table μέσω της κλάσης UserInfo. Ο μηχανισμός λειτουργίας της σελίδας, περιγράφεται στη κλάση MainActivity.HomePageActivity και ο σχεδιασμός της στο αρχείο XML activity_home_page.xml. Αν ο αρχικός και ο τελικός προορισμός, είναι ανόμοιοι τότε μπορεί ο χρήστης να συνεχίσει στην επόμενη σελίδα. Στη συνέχεια, εμφανίζεται μήνυμα επιβεβαίωσης ή μήνυμα λάθους για επιτυχημένη ή αποτυχημένη πρόσβαση.

```
add_button.setOnClickListener((view) → {


        t_name1 = trip_name.getText().toString();
        city_start1 = String.valueOf(spinner1.getSelectedItem());
        city_stop1 = String.valueOf(spinner2.getSelectedItem());
        date1 = date.getText().toString();

        if (city_start1 == city_stop1) {
            Toast.makeText(getApplicationContext(), text: "No valid destinations", Toast.LENGTH_LONG).show();
        } else {
            String id_user = dataBaseHelper.user_id(email1, password1);
            // String id_plane=dataBaseHelper.id_plane(city_start1,countryStop);
            boolean addk = dataBaseHelper.addInfo(new UserInfo( id: null, city_start1, city_stop1, date1, t_name1, id_user));

            if (addk == true) {
                Toast.makeText(getApplicationContext(), text: "Add a destination!", Toast.LENGTH_LONG).show();

                Intent intent = new Intent( packageContext: HomePageActivity.this, Profil.class);
                startActivity(intent);
            } else {
                Toast.makeText(getApplicationContext(), text: "Failed!", Toast.LENGTH_LONG).show();
            }

        }
});
```

Εικόνα 22. MainActivity.HomePageActivity - Καταχώρηση στοιχείων προορισμού.

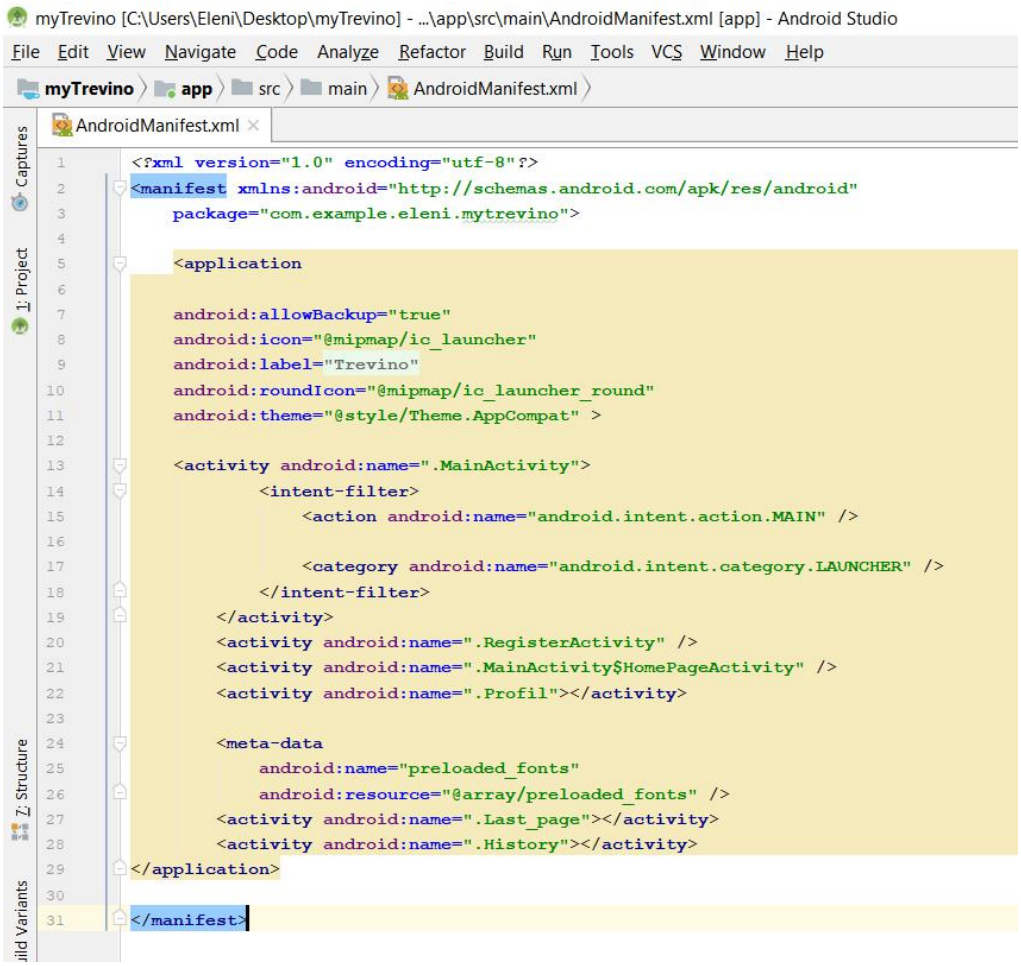
Για τον σχεδιασμό της σελίδας χρησιμοποιήθηκε εικόνα 346x146 JPEG(24-bit-color), δύο spinners που περιέχουν τους αρχικούς και τελικούς προορισμούς, δύο TextViews που εμφανίζονται ως ετικέτες για τον αρχικό και τελικό προορισμό (Origin, Destination), καθώς και EditTexts που καταγράφουν την ημερομηνία (Date) και το όνομα του ταξιδιού(Trip Name), αντίστοιχα.

```
public void spinnerList() {
    final List<String> spinnerList = new ArrayList<>();
    spinnerList.add("Athens");
    spinnerList.add("Thessaloniki");
    spinnerList.add("Lamia");
    spinnerList.add("Toulouse");
    spinnerList.add("Paris");
    spinnerList.add("Lyon");
    spinnerList.add("Barcelona");
    spinnerList.add("Girona");
    spinnerList.add("Rome");


    ArrayAdapter<String> myAdapter = new ArrayAdapter<>( context: HomePageActivity.this, R.layout.support_simple_spinner_dropdown_item, spinnerList);
    myAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

    spinner1.setAdapter(myAdapter);
    spinner2.setAdapter(myAdapter);
}


}
```

Εικόνα 23. Λίστα προρισμών των spinners.

```
public void calendar() {
    date = (EditText) findViewById(R.id.date);
    date.setOnClickListener((v) → {
            Calendar mcurrentDate = Calendar.getInstance();
            year = mcurrentDate.get(Calendar.YEAR);
            month = mcurrentDate.get(Calendar.MONTH);
            day = mcurrentDate.get(Calendar.DAY_OF_MONTH);
            DatePickerDialog mDatePicker = new DatePickerDialog( context: HomePageActivity.this, (datepicker, selectedyear, selectedmonth, selectedday) → {
                    Calendar myCalendar = Calendar.getInstance();
                    myCalendar.set(Calendar.YEAR, selectedyear);
                    myCalendar.set(Calendar.MONTH, selectedmonth);
                    myCalendar.set(Calendar.DAY_OF_MONTH, selectedday);
                    String myFormat = "dd/MM/yy"; //Change as you need
                    SimpleDateFormat sdf = new SimpleDateFormat(myFormat, Locale.ENGLISH);
                    date.setText(sdf.format(myCalendar.getTime()));

                    day = selectedday;
                    month = selectedmonth;
                    year = selectedyear;
            }, year, month, day);
            mDatePicker.setTitle("Select date");
            mDatePicker.show();
    });
```

Εικόνα 24. Τρόπος λειτουργίας του ημερολογίου και η εμφάνιση της ημερομηνίας που επιλέχθηκε στο EditText date.



Εικόνα 25. AndroidManifest.xml.

Μετά την εισαγωγή όλων των στοιχείων στη βάση δεδομένων αποθηκεύουμε ένα αντίγραφο της στην επιφάνεια εργασίας. Το αντίγραφο αυτό θα χρησιμοποιηθεί για την επεξεργασία των πληροφοριών στην Eclipse.

3.3.2 Επεξεργασία Πληροφοριών στην Eclipse

Για την επεξεργασία των πληροφοριών και τη δημιουργία συστάσεων χρησιμοποιήθηκε η πλατφόρμα της Eclipse, έκδοσης 2018-09 (4.9.0), καθώς και ένα σύστημα δημιουργίας και διαχείρισης κανόνων Drools έκδοσης 7.17.0. Το σύστημα αυτό, ενημερώνεται με το αντίγραφο της βάσης δεδομένων που αποθηκεύσαμε νωρίτερα και αποτελεί το μέσο παροχής συστάσεων σύμφωνα με τους κανόνες (rules) που θέσαμε. Οι κανόνες, αφορούν όλους τους πιθανούς συνδυασμούς μέσω μεταφορών που υπάρχουν στη βάση δεδομένων (αεροπλάνα και τρένα). Η επεξεργασία των πληροφοριών δεν γίνεται σε πραγματικό χρόνο και στοχεύει στην αποθήκευση των συστάσεων για όλους τους πιθανούς προορισμούς σε δυο αρχεία ανάλογα με την ηλικία του χρήστη. Έτσι στη συνέχεια, η επιλογή της κατάλληλης διαδρομής από το Android Studio γίνεται με την ανάγνωση του ανάλογου αρχείου και την εύρεση της σωστής πρότασης.

Για την διαχείριση των κανόνων χρειάστηκαν οι κλάσεις Plane (Εικόνα 27) και Train. Ακόμη, χρειάστηκαν η κλάση Queries στην οποία διαχειρίζονται όλα τα ερωτήματα (queries) προς τη βάση, αλλά και οι κλάσεις DestDrools και Results που είναι υπεύθυνες για την ενημέρωση του συστήματος και για την αποθήκευση των αποτελεσμάτων (συστάσεων).

Για το σκοπό αυτό, θέσαμε τέσσερα (4) ArrayLists τύπου Plane και Train, αντίστοιχα.

```
public static final ArrayList<Plane> finalDesPlane = new ArrayList<Plane>();
public static final ArrayList<Plane> nextDesPlane = new ArrayList<Plane>();
public static final  ArrayList<Train> finalDesTrain = new ArrayList<Train>();
public static final ArrayList<Train> nextDesTrain= new ArrayList<Train>();
```

Η σύνδεση της πλατφόρμας με την βάση δεδομένων περιγράφεται στη παρακάτω εικόνα:

```
public static final void main(String[] args) throws SQLException,ParseException {
    try {

         Class.forName("org.sqlite.JDBC");
           connection = DriverManager.getConnection("jdbc:sqlite:C:\\Users\\Eleni\\Desktop\\Trevino");

    } catch (Throwable t) {
        t.printStackTrace();
    }
```

Εικόνα 26. DestDrools - Συνάρτηση σύνδεσης με τη βάση δεδομένων.

```
public class Plane {

    String plane_city_start,plane_city_stop,stop1,flight_time,total_flight_time,stop2,stop3;
    Boolean directExists,many_stops,planes_trains;

    public String getStop2() {  return stop2;}

    public void setStop2(String stop2) {this.stop2 = stop2;}

    public String getStop3() {return stop3;}

    public Boolean getPlanes_trains() {return planes_trains;}

    public void setPlanes_trains(Boolean planes_trains) {this.planes_trains = planes_trains;}

    public void setStop3(String stop3) {this.stop3 = stop3;}

    public Boolean getMany_stops() {return many_stops;}

    public void setMany_stops(Boolean many_stops) {this.many_stops = many_stops;}

    public String getFlight_time() {return flight_time; }

    public void setFlight_time(String flight_time) {this.flight_time = flight_time;}

    public String getTotal_flight_time() {return total_flight_time;}

    public void setTotal_flight_time(String total_flight_time) {this.total_flight_time = total_flight_time;}

    public Boolean getDirectExists() {return directExists;}

    public void setDirectExists(Boolean directExists) {this.directExists = directExists;}

    public String getPlane_city_start() {return plane_city_start;}

    public void setPlane_city_start(String plane_city_start) {this.plane_city_start = plane_city_start;}

    public  String getPlane_city_stop() {return plane_city_stop;}

    public void setPlane_city_stop(String plane_city_stop) {this.plane_city_stop = plane_city_stop;}
```

Εικόνα 27. Plane - Κλάση διαχείρισης πίνακα Aeroplane.

Οι μεταβλητές stop1, stop2, stop3, many_stops και planes_trains εξυπηρετούν την ομαλή λειτουργία των κανόνων. Η εύρεση του αρχικού και του τελικού προορισμού γίνεται με τα παρακάτω queries, βάσει το ID του προορισμού :

```
public static void start_destination(Connection connection,int num) throws SQLException
{
    try{
             String num1=Integer.toString(num);
             query = connection.createStatement();
             String sql="select Start_Destination from Destinations where ID=?";
             pr_query = connection.prepareStatement(sql);
             pr_query.setString(1,num1);
            rs= pr_query.executeQuery();

            while(rs.next())
            {
               start_d = rs.getString("Start_Destination");
               System.out.println(start_d);
            }
          }catch(SQLException e) {
            System.out.println(e.getMessage());
          }
    }

    public static void final_destination(Connection connection,int num) throws SQLException
    {
          try{
              String num1=Integer.toString(num);

            query = connection.createStatement();
            String sql="select Final_Destination from Destinations where ID=?";
             pr_query = connection.prepareStatement(sql);
             pr_query.setString(1,num1);
             rs= pr_query.executeQuery();

            while(rs.next())
            {
               final_d= rs.getString("Final_Destination");
               System.out.println(final_d);
            }
           }catch(SQLException e) {
            System.out.println(e.getMessage());
        }
    }
```

Εικόνα 28. Στις μεταβλητές start_d και final_d αποθηκεύονται ο αρχικός και ο τελικός προορισμός.

Ο σχεδιασμός του κώδικα έχει γίνει, ώστε το ο αρχικός και τελικός προορισμός να εξαρτάται από το ID που αντιπροσωπεύει η διαδρομή στον πίνακα Destinations. Στην περίπτωση που υπάρχει εγγραφή στο πίνακα Aeroplane με αρχικό και τελικό προορισμό ίδιο με αυτό που βρέθηκε, τότε θεωρούμε πως έχουμε απευθείας πτήση.

```
public static boolean directFlight(Connection connection) throws SQLException
{
    Plane p=new Plane();
    String sql="SELECT flight_time FROM Aeroplane WHERE Start_Destination=?  AND Final_Destination=?";
    try{
        pr_query = connection.prepareStatement(sql);
        pr_query.setString(1,start_d);
        pr_query.setString(2,final_d);
        rs= pr_query.executeQuery();

        if(rs.next())
        {
         ff=rs.getString(1);
          directExistsPlane=true;
          p.setDirectExists(directExistsPlane);
          p.setFlight_time(ff);
        }
        else
          {
             directExistsPlane= false;
             p.setDirectExists(directExistsPlane);
          }

          }catch(SQLException e) {
             System.out.println(e.getMessage());
         }

    return  p.getDirectExists();

}
```

Εικόνα 29. Συνάρτηση απευθείας πτήσης.

Με τον ίδιο τρόπο λειτουργεί και η συνάρτηση directTrain() με την οποία γίνεται η εύρεση απευθείας διαδρομής με τρένο.

```
public static void finalDestList(Connection connection) throws SQLException, ParseException
{
    String sql="SELECT Final_Destination,Flight_time  FROM Aeroplane WHERE Start_Destination=? AND Final_Destination!=?";
    String sql1="SELECT Final_Destination ,Trip_time FROM Train WHERE Start_Destination=? AND Final_Destination!=? ";
    try{
        pr_query = connection.prepareStatement(sql);
        pr_query.setString(1,start_d);
        pr_query.setString(2,final_d);
        rs= pr_query.executeQuery();


         while(rs.next())
         {
           Plane p=new Plane();

           DateFormat time=new SimpleDateFormat("HH:mm");

           end=rs.getString(1);
           p.setPlane_city_stop(end);
           String k2=rs.getString(2);
           flight_time=time.parse(k2);
           String ff=time.format(flight_time);

           p.setFlight_time(ff);
           p.setPlane_city_start(start_d);
           start1=p.getPlane_city_start();
           p.setPlane_city_start(start1);
           p.setDirectExists(false);
           p.setMany_stops(null);
           finalDesPlane.add(p);

         }

        }catch(SQLException e) {
            System.out.println(e.getMessage());
         }
```

Εικόνα 30. Εύρεση ενδιάμεσων προορισμών με αεροπλάνο και αποθήκευση σε finalDesPlane.

Για την εύρεση του πρώτου ενδιάμεσου προορισμού χρησιμοποιείται η συνάρτηση finalDesList με queries για τον πίνακα Aeroplane και Train. Η αναζήτηση γίνεται μεταξύ των εγγραφών εκ των οποίων ο αρχικός προορισμός είναι ίδιος με αυτόν του χρήστη, αλλά ο τελικός όχι. Έτσι αποθηκεύονται στο arrayList finalDesPlane οι διαδρομές με αεροπλάνο με τα παραπάνω χαρακτηριστικά. Αντίστοιχα, συμβαίνει το ίδιο και για τις εγγραφές στον πίνακα Train όπου αποθηκεύονται στο arrayList finalDesTrain. Στην παρακάτω εικόνα απεικονίζεται η συνάρτηση matchPlane(), όπου θέτουμε ως αρχικό προορισμό τον τελικό προορισμό plane_city_stop του finalDesPlane. Στις μεταβλητές start και stop1 αποθηκεύονται ο τελικός και ο αρχικός προορισμός των διαδρομών του finalDesPlane. Ακόμη, απαραίτητο είναι ο τελικός προορισμός να είναι διάφορος του αρχικού. Τα αποτελέσματα του query (αρχικός και τελικός προορισμός, χρόνος πτήσης) αποθηκεύονται στο arrayList nextDesPlane.

```
public static void matchPlane(Connection connection) throws SQLException,ParseException
{
    String sql="SELECT Final_Destination,Flight_time FROM Aeroplane WHERE Start_Destination=? AND Final_Destination!=?";

    try{

        for(Plane p: finalDesPlane)
        {

            DateFormat time = new SimpleDateFormat("hh:mm");

          pr_query = connection.prepareStatement(sql);
          pr_query.setString(1,p.getPlane_city_stop());
          pr_query.setString(2,start_d);
          rs= pr_query.executeQuery();
          start= p.getPlane_city_stop();
          stop1=p.getPlane_city_start();
          p.setStop1(stop1);
          String stop2=stop1;

         while(rs.next())
         {
            Plane p1=new Plane();

            end=rs.getString(1);
            p1.setPlane_city_stop(end);
            String k2=rs.getString(2); //flight_time
            flight_time=time.parse(k2);
            ff=time.format(flight_time);
            p1.setFlight_time(ff);
            p1.setPlane_city_start(start);
            p1.setStop1(stop2);

            nextDesPlane.add(p1);
        }
        }
    } catch(SQLException e) {
            System.out.println(e.getMessage());
 }
}
```

Εικόνα 31. Συνάρτηση matchPlane().

Το ίδιο συμβαίνει και για την εύρεση τρένων στην συνάρτηση *matchTrain()*. Οι συναρτήσεις *queryPlane()* και *queryTrain()* λειτουργούν με την ίδια φιλοσοφία, με τη διαφορά ότι ο αρχικός προορισμός λαμβάνεται από την μεταβλητή plane_city_stop του nextDesPlane και train_city_stop του nextDesTrain, που "γέμισαν" στην προηγούμενη συνάρτηση. Επίσης, στην Εικόνα 33 περιγράφεται η συνάρτηση *train_plane_one_stop().* Εδώ, γίνεται εύρεση διαδρομών όπου ο αρχικός προορισμός διαδέχεται τον επόμενο και ο χρήστης πρέπει να ταξιδέψει με τρένο. Από τον προορισμό αυτό, θα χρειαστεί αεροπλάνο για να φτάσει στον τελικό προορισμό. Το ίδιο συμβαίνει και στην περίπτωση *plane_train_one_stop(), ό*που ο χρήστης για να μεταφερθεί από τον αρχικό προορισμό στον επόμενο θα χρησιμοποιήσει αεροπλάνο και στη συνέχεια τρένο για να φτάσει εκεί που θέλει. Τα αποτελέσματα του ανάλογου query, σύμφωνα με τις πληροφορίες που λαμβάνει από το finalDesPlane (θέτω ως αρχικό προορισμό τον τελικό προορισμό που είναι αποθηκευμένο στο arrayList), αποθηκεύονται στο nextDesTrain. Επίσης, στις συναρτήσεις plane_train_plane(), train_plane_train(), many_planes_one_train() και many_trains_one_plane(), καλούνται queries για την δημιουργία διαδρομών συνδυασμού αεροπλάνων και τρένων με την ίδια λογική "αλυσίδας".

```
}
    public static void queryTrain(Connection connection) throws SQLException,ParseException
    {
        String sql1="SELECT Final_Destination,Trip_time  FROM Train WHERE Start_Destination=? AND Final_Destination!=?";

    try {

        for(Train tr:nextDesTrain)
         {
           if(tr.getTrain_city_start()!=null)
           {
               DateFormat time = new SimpleDateFormat("hh:mm");

           pr_query = connection.prepareStatement(sql1);
           pr_query.setString(1,tr.getTrain_city_stop());
           pr_query.setString(2,start_d);
           rs= pr_query.executeQuery();
           stop1= tr.getTrain_city_start();
           start= tr.getTrain_city_stop();
           String stop2=stop1;

           while(rs.next())
           {
             Train tr1=new Train();
             end=rs.getString(1);
             tr1.setTrain_city_stop(end);
             trip_t=rs.getString(2);
             tr1.setTrain_city_start(start);
             tr1.setStop1(stop2);
             trip_time=time.parse(trip_t);
             tr1.setTrip_time(trip_t);

             finalDesTrain.add(tr1);

        }
      }
      }
    }catch(SQLException e) {
         System.out.println(e.getMessage());
    }
   }
```

Εικόνα 32. Συνάρτηση queryTrain().Τα αποτελέσματα αποθηκεύονται στο finalDesTrain.

```
public static void train_plane_one_stop(Connection connection) throws SQLException,ParseException
{

    String sql1="SELECT Final_Destination,Flight_time FROM Aeroplane WHERE Start_Destination=? AND Final_Destination!=? ";

     try{

        for(Train tr: finalDesTrain)
        {
          DateFormat time = new SimpleDateFormat("hh:mm");

           pr_query = connection.prepareStatement(sql1);
           pr_query.setString(1,tr.getTrain_city_stop());
           pr_query.setString(2,start_d);
           rs= pr_query.executeQuery();
           start= tr.getTrain_city_stop();
           stop1=tr.getTrain_city_start();
           tr.setStop1(stop1);
           String stop2=stop1;

           while(rs.next())
           {
              Plane p1=new Plane();
              end=rs.getString(1);
              p1.setPlane_city_stop(end);
              String k2=rs.getString(2); //flight_time
              flight_time=time.parse(k2);
              ff=time.format(flight_time);
              p1.setFlight_time(ff);
              p1.setPlane_city_start(start);
              p1.setStop1(stop2);

              nextDesPlane.add(p1);
            }
        }
      }catch(SQLException e) {
             System.out.println(e.getMessage());
        }
}
```

Εικόνα 33. Συνάρτηση train_plane_one_stop(). Τα αποτελέσματα αποθηκεύονται στο nextDesPlane.

3.3.2.1 Δημιουργία και Ενημέρωση Συστήματος Κανόνων

Για την αποθήκευση των αποτελεσμάτων των κανόνων χρησιμοποιήθηκε η κλάση Results, η οποία περιέχει την boolean μεταβλητή status, αλλά και το arrayList `ArrayList <String> msg_Arr=new ArrayList<String>();` στο οποίο θα αποθηκεύεται το σύνολο μηνυμάτων του κάθε κανόνα. Για τη δημιουργία, ακόμη, κάθε σύστασης πρέπει να τηρούνται οι προϋποθέσεις του εκάστοτε κανόνα, οι οποίοι περιέχονται στο αρχείο test.drl.

- **Απευθείας Πτήση**

```
1  package com.sample
2
3  import com.sample.Train;
4  import com.sample.Plane;
5  import com.sample.Queries;
6
7  rule "Direct Flight"
8   no-loop
9      when
10     $plane:Plane($direct:directExists, $direct==true,$flight_time:flight_time)
11     $res:Results($status:status,$status==true)
12     then
13       $res.setMessage("Direct Flight, Total Time : "+ $flight_time);
14
15 end
16
```

Εικόνα 34. Κανόνας για απευθείας πτήση.

Στην παραπάνω περίπτωση, όταν η boolean μεταβλητή $direct (Εικόνα 29) γίνεται αληθής (Plane) και η μεταβλητή status γίνει και εκείνη αληθής (Results) τότε η μεταβλητή `String message;` ενημερώνεται με το μήνυμα της εικόνας. Στη κλάση DestDrools ενημερώνονται οι κανόνες.

```
private static void direct_Plane()
{
    try {
         KieServices kieServices = KieServices.Factory.get();
         KieContainer kContainer = kieServices.getKieClasspathContainer();
         KieSession kSession = kContainer.newKieSession("ksession-rules");

         Plane p= new Plane();

         p.setDirectExists(Queries.directExistsPlane);
         p.setFlight_time(Queries.ff);
         Results res=new Results();
         res.setStatus(true);

         kSession.insert(p);

         kSession.insert(res);
         kSession.fireAllRules();

         if(res.getMessage()!=null)
         {
             under40Results.add(res.getMessage());
             plus40Results.add(res.getMessage());
         }



    }catch(Throwable t) {
         t.printStackTrace();
     }
}
```

Εικόνα 35. Ενημέρωση κανόνων.

Οι εντολές kSession.insert() και kSession.fireAllRules() χρησιμοποιούνται, ώστε οι κανόνες να ενημερωθούν με τα αντικείμενα (Objects) και στη συνέχεια, οι ενημερώσεις να περάσουν στο σύστημα. Όπως φαίνεται και στη παραπάνω εικόνα χρησιμοποιούνται επίσης, δύο arrayList :

```
ArrayList <String> under40Results=new ArrayList<String>();
ArrayList <String> plus40Results=new ArrayList<String>();
```

στα οποία ανάλογα με την ηλικία του χρήστη αποθηκεύονται τα μηνύματα των κανόνων.
Με τον ίδιο τρόπο λειτουργεί και ενημερώνεται ο κανόνας για απευθείας διαδρομή με τρένο. Στην Εικόνα 36, η μεταβλητή trip_time αναφέρεται στον χρόνο διάρκειας της διαδρομής.

- **Απευθείας Τρένο**

```
rule "Direct Train"
no-loop
    when
    $train:Train($direct:directExists, $direct==true,$trip_time:trip_time)
     $res:Results($status:status,$status==true)
     then
          $res.setMessage("Direct Train, Total Time: " + $trip_time);
end
```

Εικόνα 36. Κανόνας για απευθείας διαδρομή με τρένο.

- **Πτήση με μια στάση**

```
rule "One stop with Aeroplane"
 no-loop
  when

    $plane:Plane($stop2:stop2,$stop2==null,$start:plane_city_start, $stop:plane_city_stop  ,
                 $total:total_flight_time, $direct:directExists, $direct==false,
                 $stop==Queries.final_d,$many_stops:many_stops, $many_stops==false, $total!=null)

    $res:Results($status:status,$status==true,$arraylist:msg_Arr)
    then
     $res.setMessage(Queries.start_d +"-> " + $start + "-> " + $stop + " (Aeroplane) Total Time: "+ $total);
      $arraylist.add($res.getMessage());
end
```

Εικόνα 37. Κανόνας για πτήση με μια στάση.

Στις μεταβλητές direct ελέγχεται αν υπάρχει απευθείας πτήση. Έτσι αν η μεταβλητή $direct==true, τότε δεν θα πληρούσε τις προϋποθέσεις και δεν θα μπορούσε να χρησιμοποιηθεί ο κανόνας. Η μεταβλητή

many_stops ελέγχει αν έχουμε πολλές στάσεις και στην προκειμένη περίπτωση είναι ψευδής (false). Η total αναφέρεται στον συνολικό χρόνο διαδρομής και υπολογίζεται με τη πρόσθεση των δύο μεταβλητών flight_time των arrayLists finalDesPlane και nextDesPlane (Εικόνα 38). Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, η εισαγωγή των στοιχείων στα arrayLists έγινε με τα ανάλογα queries για την δημιουργία ροής πιθανών διαδρομών. Στην παρακάτω εικόνα, δηλώνεται η ηλικία απευθείας και δεν ενημερώνεται σε πραγματικό χρόνο από την βάση δεδομένων. Ακόμη, γίνεται έλεγχος ύπαρξης αποτελεσμάτων (μηνυμάτων) από τον ανάλογο κανόνα και στη συνέχεια αποθήκευση του arrayList msg_Arr (βρίσκονται όλα τα μηνύματα που παράγει ο κανόνας) στο εκάστοτε arrayList (plus40Results ή under40Results). Ο διαχωρισμός, γίνεται ανάλογα με τον αν η ηλικία του χρήστη είναι μικρότερη ή μεγαλύτερη των 40 ετών.

```
17  private static void printPlanesOneStop()
18  {
19        try {
20            KieServices kieServices = KieServices.Factory.get();
21            KieContainer kContainer = kieServices.getKieClasspathContainer();
22            KieSession kSession = kContainer.newKieSession("ksession-rules");
23
24            Results res=new Results();
25            res.setStatus(true);
26
27         for(Plane p:Queries.finalDesPlane)
28          {
29             for(Plane p1:Queries.nextDesPlane)
30             {
31                 if(p.getPlane_city_stop()==p1.getPlane_city_start() && p.getFlight_time()!=null)
32                  {
33                     DateFormat df = new SimpleDateFormat("hh:mm");
34                     df.setTimeZone(TimeZone.getTimeZone("UTC"));
35
36                     String t=p.getFlight_time();
37                     String t1=p1.getFlight_time();
38                     Date date1 = df.parse(t);
39                     Date  date2 = df.parse(t1);
40
41                     long sum=date1.getTime() + date2.getTime();      // count total time of trip
42                     Date dat=new Date(sum);
43                     String date=df.format(new Date (sum));
44
45                     String d="05:00";
46                     String d1="06:40";
47                     Date date3=df.parse(d);
48                     Date date4=df.parse(d1);
49
50                     Queries.age=20;
51
52                 if(dat.before(date3) && Queries.age>40)
53                     {
54                         p1.setTotal_flight_time(date);
55                         p1.setMany_stops(false);
56                         p1.setDirectExists(false);
57                         p1.setStop2(null);
58                     }
59
```

```
                if(dat.before(date4) &&  Queries.age<=40)
                    {
                       p1.setTotal_flight_time(date);
                       p1.setMany_stops(false);
                       p1.setDirectExists(false);
                        p1.setStop2(null);

                    }

                  kSession.insert(p1);
                  kSession.insert(res);
                }
            }
         }
          kSession.fireAllRules();

        if(res.getMessage()!=null)
        {
         for(String p:res.msg_Arr)
           {
                if(Queries.age>40)
               {

                  plus40Results.add(p);

               }
               else
               {
                   under40Results.add(p);

               }
           }
         }

       }catch(Throwable t) {
                t.printStackTrace();
   }
 }
```

Εικόνα 38. Ενημέρωση κανόνα για πτήση με μια στάση.

- **Πτήση με πολλές στάσεις**

```
rule "More stops with Aeroplane"
 no-loop
  when

    $plane:Plane($start:plane_city_start,$stop1:stop1,$stop:plane_city_stop,$stop1!=$start,$start!=$stop,$stop==Queries.final_d,$total:total_flight_time,
    $total!=null,$many_stops:many_stops,$many_stops==true,$planes_tr:planes_trains,$planes_tr==false)
    $res:Results($status:status,$status==true,$arraylist:msg_Arr)
     then
       $res.setMessage(Queries.start_d +"-> "+$stop1 + "-> " +$start + "-> " + $stop + " (Aeroplane) " +" ,Total Time : "+$total );
       $arraylist.add($res.getMessage());


 end
```

Εικόνα 39. Κανόνας πολλαπλών πτήσεων.

Ο κανόνας αυτός, παράγει συστάσεις αν η διαδρομή που επιλέχθηκε μπορεί να πραγματοποιηθεί με πολλές διαδοχικές πτήσεις. Αν πληρούνται οι προϋποθέσεις, τότε θα εμφανιστεί μήνυμα ανάλογο του message. Η μεταβλητή planes_trains γίνεται αληθής όταν έχουμε συνεργασία μέσων μεταφοράς (διαδρομές με αεροπλάνα και τρένα). Ο έλεγχος one_time που φαίνεται στην Εικόνα 40, χρησιμοποιείται για

την αποφυγή εισόδου των πληροφοριών με ίδιο όνομα. Πιο συγκεκριμένα, επειδή χρησιμοποιούμε διαδοχικά μόνο δύο arrayLists όπου τα αδειάζουμε και τα ξαναχρησιμοποιούμε είναι πιθανή η δημιουργία λάθους. Οι κανόνες λειτουργούν ασύγχρονα και η ενημέρωση τους πρέπει να είναι συγκεκριμένη. Η ενημέρωση των κανόνων για τρένο με μια ή περισσότερες στάσεις γίνεται με τον ίδιο τρόπο.

```
direct_Plane();
direct_Train();
Queries.finalDestList(connection);        //one stop with plane
Queries.matchPlane(connection);
one_time=true;
printPlanesOneStop();
one_time=false;
Queries.finalDesPlane.clear();
one_time=true;
Queries.queryPlane(connection);   //more stops with plane
printPlanesMoreStops();
one_time=false;

 Queries.nextDesTrain.clear();
 Queries.matchTrain(connection);          // one stop with train
 one_time=true;
 printTrainsOneStop();
 one_time=false;
 Queries.finalDesTrain.clear();
 one_time=true;
 Queries.queryTrain(connection);         // more stops with Train
 printTrainsMoreStops();
 one_time=false;
```

Εικόνα 40. Ενημέρωση του one_time και κάλεσμα των συναρτήσεων ενημημέρωσης των arrayLists και των κανόνων.

```
}
private static void printPlanesMoreStops()
 {

      try {
          KieServices kieServices = KieServices.Factory.get();
          KieContainer kContainer = kieServices.getKieClasspathContainer();
          KieSession kSession = kContainer.newKieSession("ksession-rules");

          Results res=new Results();
          res.setStatus(true);

          for(Plane p1:Queries.nextDesPlane)
          {
           for(Plane p2:Queries.finalDesPlane)
             {
              if(one_time==true && p1.getPlane_city_stop()==p2.getPlane_city_start() && p1.getTotal_flight_time()!=null)
               {
                DateFormat df = new SimpleDateFormat("HH:mm");
                df.setTimeZone(TimeZone.getTimeZone("UTC"));

                String t=p1.getTotal_flight_time();
                String t1=p2.getFlight_time();

                Date date1 = df.parse(t);
                Date  date2 = df.parse(t1);
                long sum=date1.getTime() + date2.getTime();
                Date dat=new Date(sum);

                String date=df.format(new Date (sum));

                String d="05:30";
                String d1="07:00";
                Date date3=df.parse(d);             //old
                Date date4=df.parse(d1);
                Queries.age=20;

            if(dat.before(date3) && Queries.age>40)
               {
                       p2.setTotal_flight_time(date);
                       p2.setMany_stops(true);
                       p2.setPlanes_trains(false);
               }

            if(dat.before(date4) &&  Queries.age<=40)
                {
                       p2.setTotal_flight_time(date);
                       p2.setMany_stops(true);
                       p2.setPlanes_trains(false);
                }

                   kSession.insert(p2);
               }
              }
           }
             kSession.insert(res);
          kSession.fireAllRules();

          if(res.getMessage()!=null)
          {
           for(String p:res.msg_Arr)
             {
                  if(Queries.age>40)
                 {

                    plus40Results.add(p);

                 }
                 else
                 {
                     under40Results.add(p);

                 }
           }
          }
         }catch(Throwable t) {
                 t.printStackTrace();

          }
 }
```

Εικόνα 41. Ενημέρωση κανόνα πολλαπλών στάσεων με αεροπλάνο.

- **Διαδρομή με τρένο και αεροπλάνο**

```
79
80 rule "Train,Plane"
81
82 no-loop
83  when
84
85     $train:Train($stop2:stop2,$stop2!=null,$planes_trains:planes_trains,$planes_trains==false,
86                  $many_stops:many_stops,$many_stops==true)
87     $plane:Plane($stop2_plane:stop2,$stop2_plane!=null,$start_Plane:plane_city_start,
88                  $stop_Plane:plane_city_stop, $stop_Plane==Queries.final_d,$total:total_flight_time,$total!=null)
89     $res:Results($status:status,$status==true,$arraylist:msg_Arr)
90
91  then
92
93      $res.setMessage(Queries.start_d +"->(Train) ->"+ $start_Plane + "-> (Aeroplane) ->"  + $stop_Plane + ", Total Time: " + $total );
94     $arraylist.add($res.getMessage());
95
96 end
97
```

```
0  private static void train_Plane_one_stop()
1  {
2      try {
3             KieServices kieServices = KieServices.Factory.get();
4             KieContainer kContainer = kieServices.getKieClasspathContainer();
5             KieSession kSession = kContainer.newKieSession("ksession-rules");
6
7             Results res=new Results();
8             res.setStatus(true);
9
0              for(Train tr:Queries.finalDesTrain)
1              {
2                 for(Plane p:Queries.nextDesPlane)
3                 {
4                     if(tr.getTrain_city_stop()==p.getPlane_city_start() && tr.getTrip_time()!=null)
5                     {
6
7                         DateFormat df = new SimpleDateFormat("HH:mm");
8                         df.setTimeZone(TimeZone.getTimeZone("UTC"));
9
0                         String t=tr.getTrip_time();
1                         String t1=p.getFlight_time();
2
3                         Date date1 = df.parse(t);
4                         Date date2 =df.parse(t1);
5                         long sum=date1.getTime() + date2.getTime();
6                         Date dat=new Date(sum);
7                         String date=df.format(dat);
8                         String d="07:00";
9                         String d1="08:10";
0
1                         Date date3=df.parse(d);
2                         Date date4=df.parse(d1);
3
4                        Queries.age=20;
5                         if(dat.before(date3) &&  Queries.age>40)
6                         {
7                             p.setTotal_flight_time(date);
8                             tr.setStop2("go");
9                             p.setStop2("go");
0                             tr.setPlanes_trains(false);
1                             tr.setMany_stops(true);
2                         }
```

```
                    if(dat.before(date4) &&  Queries.age<=40)
                    {
                            p.setTotal_flight_time(date);
                            tr.setStop2("go");
                            p.setStop2("go");
                            tr.setPlanes_trains(false);
                            tr.setMany_stops(true);

                    }

                  kSession.insert(tr);
                  kSession.insert(p);
                }
              }
            }

              kSession.insert(res);
              kSession.fireAllRules();

              if(res.getMessage()!=null)
               {
                for(String p:res.msg_Arr)
                  {
                       if(Queries.age>40)
                      {

                         plus40Results.add(p);

                      }
                      else
                      {
                          under40Results.add(p);

                      }
                  }
                }

             }catch(Throwable t) {
            t.printStackTrace();
          }
}
```

Εικόνα 42. Δημιουργία και ενημέρωση κανόνα για διαδρομή με τρένο και αεροπλάνο.

Στον παραπάνω κανόνα, έχουμε την περίπτωση όπου η διαδρομή από τον αρχικό στον τελικό προορισμό μπορεί να πραγματοποιηθεί πρώτα με τρένο και έπειτα με αεροπλάνο. Στη συνάρτηση train_Plane_one_stop() υπολογίζεται ο συνολικός χρόνος ταξιδιού. Η μεταβλητή d, η οποία μετατρέπεται σε μεταβλητή τύπου Date, αποτελεί το συνολικό χρονικό όριο που μπορεί να ταξιδέψει ένας χρήστης με ηλικία μεγαλύτερη των 40 ετών και στην μεταβλητή d1, όπου επίσης μετατρέπεται σε τύπου Date αποθηκεύεται το ανάλογο χρονικό όριο για τους χρήστες κάτω των 40 ετών. Η μεταβλητή stop2 χρησιμοποιείται ως περαιτέρω έλεγχος για την είσοδο των πληροφοριών στο σύστημα και όχι για την αποθήκευση των στάσεων. Σε όλες τις συναρτήσεις ενημέρωσης του συστήματος δίνεται χρονικό όριο συνολικού χρόνου διαδρομής, με σκοπό την καλύτερη παροχή συστάσεων.

- **Διαδρομή με δύο τρένα και ένα αεροπλάνο**

```
rule "Many Trains & One Plane"
 no-loop
  when

    $train:Train($stop_Train:train_city_stop, $start_Train:train_city_start,$many_stops1:many_stops,$many_stops1==true)
    $plane:Plane($start_Plane:plane_city_start,$stop_Plane:plane_city_stop,$stop_Plane==Queries.final_d,
                $start_Train!=Queries.start_d,$many_stops:many_stops,$many_stops==false,
                $total:total_flight_time,$total!=null)
    $res:Results($status:status,$status==true,$arraylist:msg_Arr)

   then

    $res.setMessage(Queries.start_d+"->(Train) ->"+$start_Train +"->(Train) ->" + $stop_Train + "-> (Aeroplane) ->"  + $stop_Plane +" Total Time: "+ $total);
    $arraylist.add($res.getMessage());
end
```

Εικόνα 43. Κανόνας για διαδρομή δύο τρένων και αεροπλάνων.

Ο υπολογισμός του συνολικού χρόνου διαδρομής και για την παραπάνω περίπτωση, γίνεται όπως και στους υπόλοιπους κανόνες. Έτσι, για την μαζική ενημέρωση του συστήματος καλούνται όλες οι παρακάτω συναρτήσεις σε συνδυασμό με την διαδοχική χρησιμοποίηση των arrayLists.

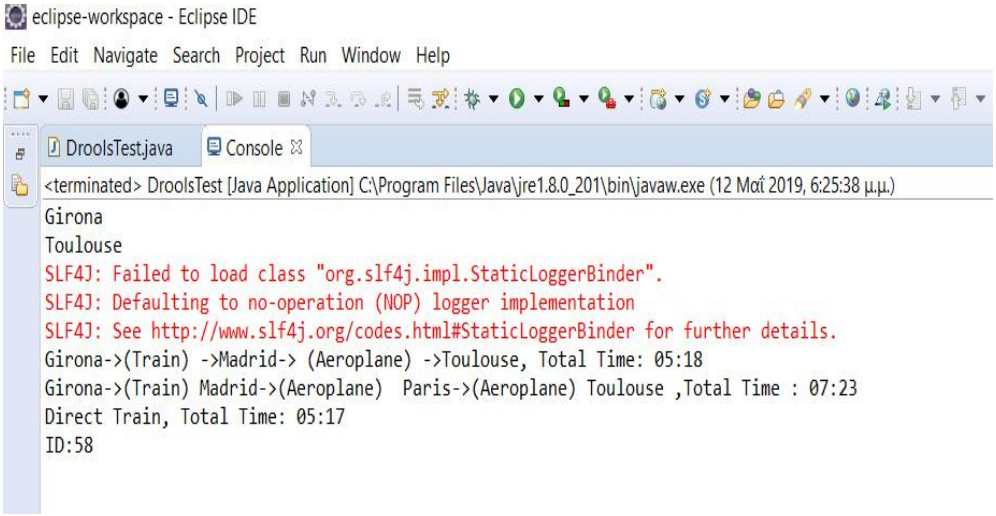
```
Queries.finalDesPlane.clear();
Queries.finalDesTrain.clear();
Queries.finalDestList(connection);
Queries.nextDesPlane.clear();
Queries.train_plane_one_stop(connection);  // train -> plane
train_Plane_one_stop();
Queries.nextDesTrain.clear();
Queries.train_plane_train(connection);     //train->plane->train
train_plane_train();

Queries.finalDesPlane.clear();
Queries.finalDesTrain.clear();
Queries.finalDestList(connection);
Queries.nextDesTrain.clear();
Queries.matchTrain(connection);
Queries.finalDesPlane.clear();
one_time=true;
Queries.many_trains_one_plane(connection);   // train-> train-> plane
many_trains_Plane_one_stop();
one_time=false;

Queries.finalDesPlane.clear();
Queries.finalDesTrain.clear();
Queries.finalDestList(connection);
Queries.nextDesTrain.clear();
Queries.plane_train_one_stop(connection);    // plane->train
one_time=true;
plane_Train_one_stop();
one_time=false;
Queries.nextDesPlane.clear();
Queries.plane_train_plane(connection);      //plane->train->plane
one_time=true;
plane_train_plane();
one_time=false;

Queries.finalDesTrain.clear();
one_time=true;
Queries.queryTrain(connection);
plane_train_train();                        //plane->train->train
one_time=false;
```

```
Queries.finalDesPlane.clear();
Queries.finalDesTrain.clear();
Queries.finalDestList(connection);
Queries.nextDesPlane.clear();
Queries.train_plane_one_stop(connection);
train_Plane_one_stop();
Queries.finalDesPlane.clear();
Queries.queryPlane(connection);
one_time=true;
train_plane_plane();              //train->plane->plane
one_time=false;

 Queries.finalDesPlane.clear();
 Queries.finalDestList(connection);
 Queries.nextDesPlane.clear();
 Queries.matchPlane(connection);
 printPlanesOneStop();

 Queries.finalDesTrain.clear();
 Queries.many_planes_one_train(connection);    //plane-> plane -> train
 one_time=true;
 many_planes_Train_one_stop();
 one_time=false;

Queries.nextDesTrain.clear();
Queries.matchTrain(connection);       //plane->plane->train->train
one_time=true;
many_planes_many_trains();
one_time=false;
```

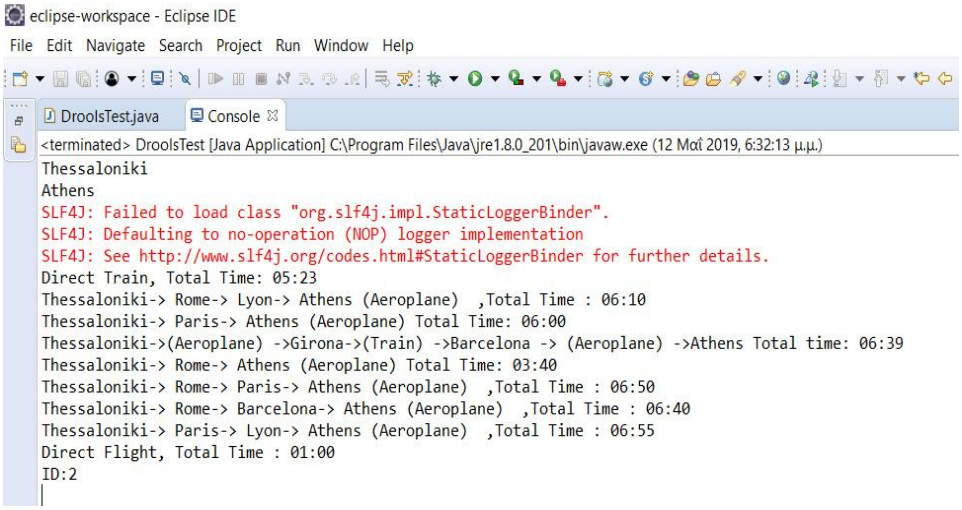
Εικόνα 44. Συναρτήσεις ενημέρωσης συστήματος - DestDrools.

3.3.2.2 Αποτελέσματα Συστήματος Κανόνων

Μετά την δημιουργία των κανόνων και την ενημέρωση τους με τα στοιχεία της βάσης δεδομένων, παράχθηκαν αποτελέσματα ανάλογα των διαδρομών. Η αποθήκευση των αποτελεσμάτων, όπως αναφέρθηκε και σε παραπάνω κεφάλαιο, έγινε σε arrayLists τύπου String και στη συνέχεια, σε αρχεία τύπου txt ανάλογα με την ηλικία του χρήστη. Στην ηλικιακή ομάδα κάτω των 40 ετών χρησιμοποιήθηκε το αρχείο results_Under_40.txt, καθώς το αρχείο results_Plus_40.txt για τα αποτελέσματα των χρηστών με ηλικία μεγαλύτερη των 40 ετών. Για την διευκόλυνση αποθήκευσης των αποτελεσμάτων, η εύρεση των δρομολογίων γίνεται με τον αριθμό ID των διαδρομών του πίνακα Destinations, ο οποίος δηλώνεται από τον διαχειριστή του κώδικα. Παρακάτω, αναφέρονται παραδείγματα συστάσεων για την **ηλικιακή ομάδα κάτω των 40 ετών** :

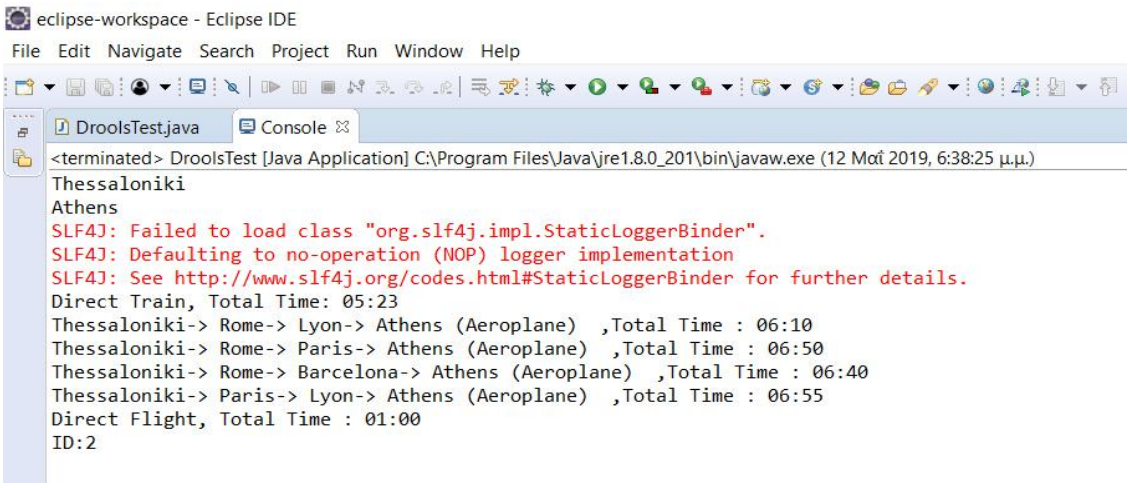


Εικόνα 45. Συστάσεις για την διαδρομή Girona -> Toulouse.



Εικόνα 46. Συστάσεις για την διαδρομή Thessaloniki->Athens.

Και άνω των 40 ετών:



Εικόνα 47. Συστάσεις για την διαδρομή Thessaloniki->Athens.

Καθώς το χρονικό όριο διαδρομής της ηλικιακής ομάδας άνω των 40 είναι μικρότερο, οι παραγόμενες συστάσεις είναι λιγότερες. Αυτό γίνεται φανερό από την σύγκριση των εικόνων 48 και 49. Αν και είναι συχνό φαινόμενο, δεν αποτελεί κανόνα. Αρκετές φορές ο αριθμός των συστάσεων είναι ίδιος άσχετα με την ηλικία του χρήστη. Αρκεί να πληρούνται οι προϋποθέσεις των κανόνων. Ακόμη, συχνή είναι και η εμφάνιση διπλότυπων αποτελεσμάτων, καθώς κάποιες συναρτήσεις καλούνται παραπάνω από μια φορά.

3.3.3 Σύνδεση Συστάσεων με το Android Studio

Για την σύνδεση των συστάσεων με την πλατφόρμα του Android Studio γίνεται χρήση των αρχείων που τις αποθηκεύσαμε νωρίτερα. Για το σκοπό αυτό δημιουργήθηκαν δύο κλάσεις οι Profil και Last_page. Στην κλάση Profil διαβάζεται το κατάλληλο αρχείο ανάλογα με την ηλικία του χρήστη και με βάσει το ID που αντιπροσωπεύει τον προορισμό του, διανέμονται οι συστάσεις σε κουμπιά (Buttons). Στη συνέχεια, ο χρήστης επιλέγει την διαδρομή της αρεσκείας του.

```
public void read_file() {

    int age1 = dataBaseHelper.age1();
    String fileName="";
    if (age1 <= 40) {
        fileName = "results_Under_40.txt";
    }
    if(age1>40)
    {
        fileName = "results_Plus_40.txt";
    }
    String line="";

    String k = dataBaseHelper.findID(start_d, final_d);
    int counter = Integer.valueOf(k);

    int counter1 = counter + 1;
    String k1 = "ID:" + counter;
    String line1 = "ID:" + counter1;
    String last="ID:72";
    String end="END";
    try {
        InputStream is = getAssets().open(fileName);
        Scanner scanner = new Scanner(is);
     if(counter!=72)
     {
        while (scanner.hasNextLine()) {
            line = scanner.nextLine();
            if (line.equals(k1)) {
                while (!line1.equals(line)) {
                    line = scanner.nextLine();
                    array.add(line + "\n");
                }

            }
```

Εικόνα 48. Επιλογή αρχείου και κατάλληλων συστάσεων.

Για την διαχείριση της τελευταίας σελίδας της εφαρμογής χρησιμοποιήθηκε η κλάση Last_Page, όπου εμφανίζονται οι πληροφορίες του χρήστη σχετικά με την διαδρομή που επιθυμεί να πραγματοποιήσει. Για το σκοπό αυτό γίνεται εμφάνιση του αρχικού και τελικού προορισμού, η ημερομηνία του ταξιδιού, το όνομα που έθεσε καθώς οι λεπτομέρειες της διαδρομής. Στο TextView Trip Details αναγράφεται η σύσταση που επιλέχθηκε από τον χρήστη στην προηγούμενη σελίδα. Ακόμη, προστέθηκαν δύο κουμπιά (Buttons) δίνοντας τις επιλογές ο χρήστης είτε να δει τους παλαιότερους προορισμούς του, είτε να αποχωρήσει από την εφαρμογή.

```
btn_history.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view)
    {
        Intent intent = new Intent( packageContext: Last_page.this, History.class);
        startActivity(intent);
    }
});

btn_exit.setOnClickListener(new View.OnClickListener() {
    final AlertDialog.Builder builder = new AlertDialog.Builder( context: Last_page.this);

    @Override
    public void onClick(View view) {

        builder.setTitle("Exit");
        builder.setMessage("Do you want to exit ??");
        builder.setNegativeButton( text: "Not now", (dialogInterface, i) → {
                dialogInterface.dismiss();
        });

        builder.setPositiveButton( text: "Yes. Exit now!", (dialogInterface, i) → {
                Intent startMain = new Intent(Intent.ACTION_MAIN);
                startMain.addCategory(Intent.CATEGORY_HOME);
                startMain.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(startMain);
                finish();
        });
        AlertDialog dialog = builder.create();
        dialog.show();
    }
});
```

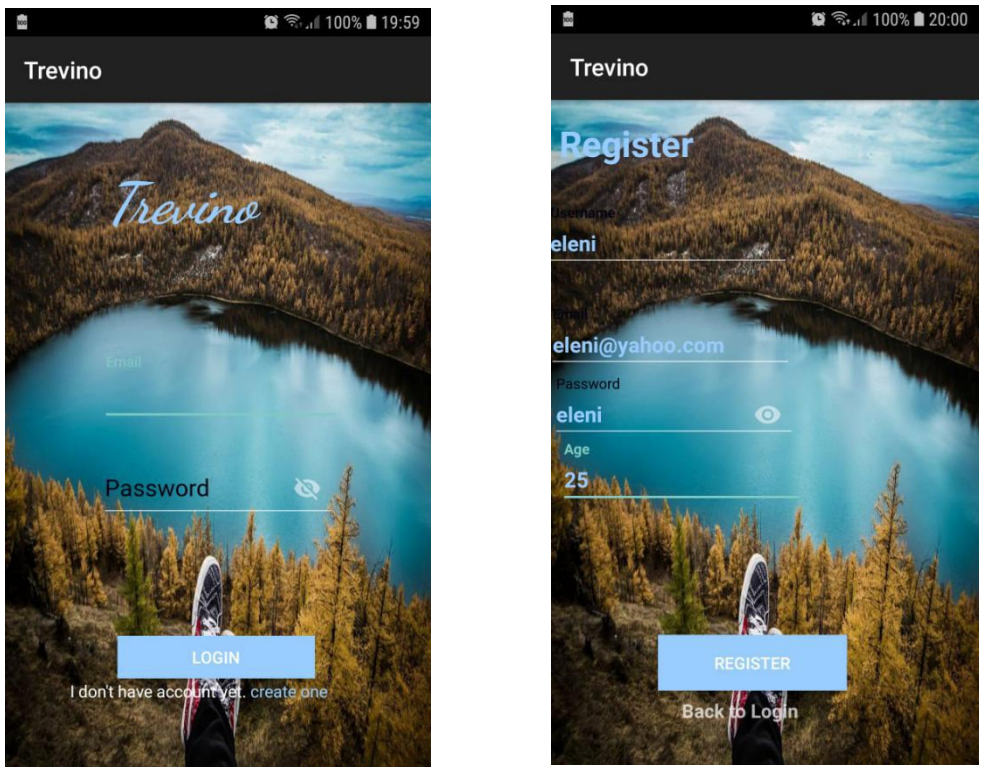
Εικόνα 49. Δραστηριότητα κουμπιών.

```
public void initViews() {
    dateText=(TextView)findViewById(R.id.dateText);
    tripDetails=(TextView)findViewById(R.id.tripText);
   details=(TextView)findViewById(R.id.details);
   trip_name = (TextView) findViewById(R.id.trip_name);
   date1 = (TextView) findViewById(R.id.date1);
   text1 = (TextView) findViewById(R.id.start);
   text2 = (TextView) findViewById(R.id.stop);
   textView=(TextView) findViewById(R.id.textView);
   origin=(TextView)findViewById(R.id.origin);
   destination=(TextView)findViewById(R.id.destination);
   btn_history = (Button) findViewById(R.id.btn_history);
   btn_exit = (Button) findViewById(R.id.btn_exit);
}
```

Εικόνα 50. Δήλωση γραφικών στοιχείων σελίδας.

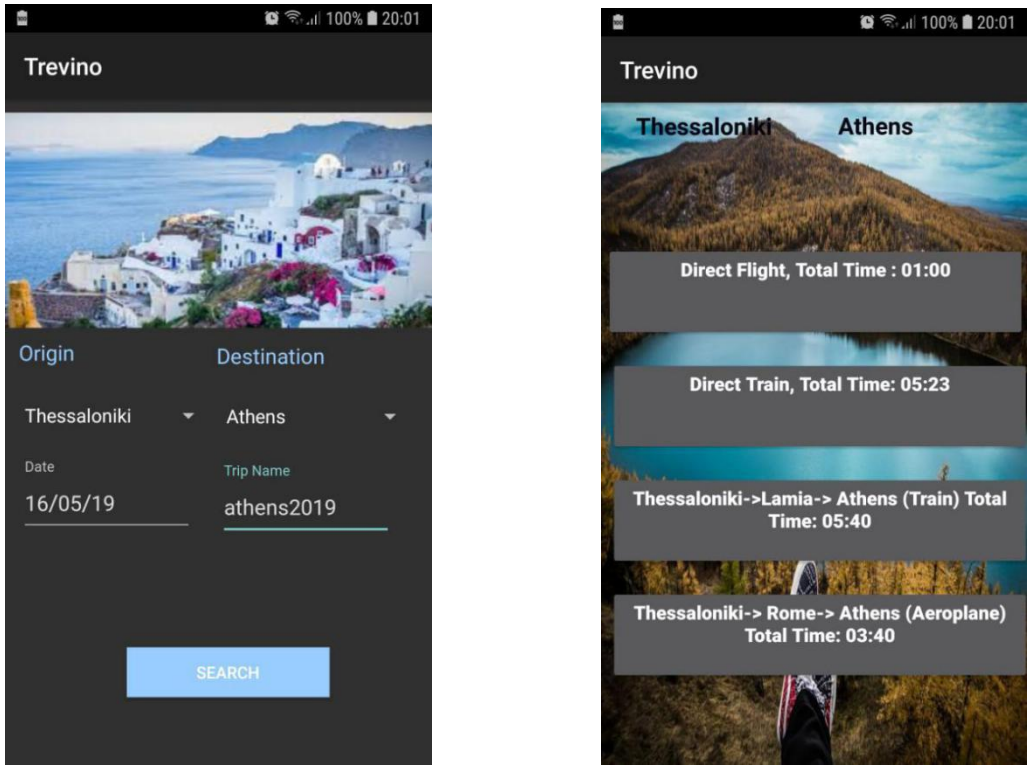
3.3.4 Εκτέλεση Εφαρμογής Trevino στη Κινητή Συσκευή

Για την ενημέρωση της βάσης δεδομένων, αλλά και γενικότερα της εφαρμογής “τρέχουμε” την εφαρμογή από το Android Studio και η εκτέλεση πραγματοποιείται στη κινητή συσκευή. Αυτό επιτυγχάνεται με τη σύνδεση της συσκευής μέσω USB καλωδίου. Τα αποτελέσματα για κάθε σελίδα παρουσιάζονται παρακάτω :



Εικόνα 51. Σελίδα σύνδεσης και εγγραφής (Login - Register).

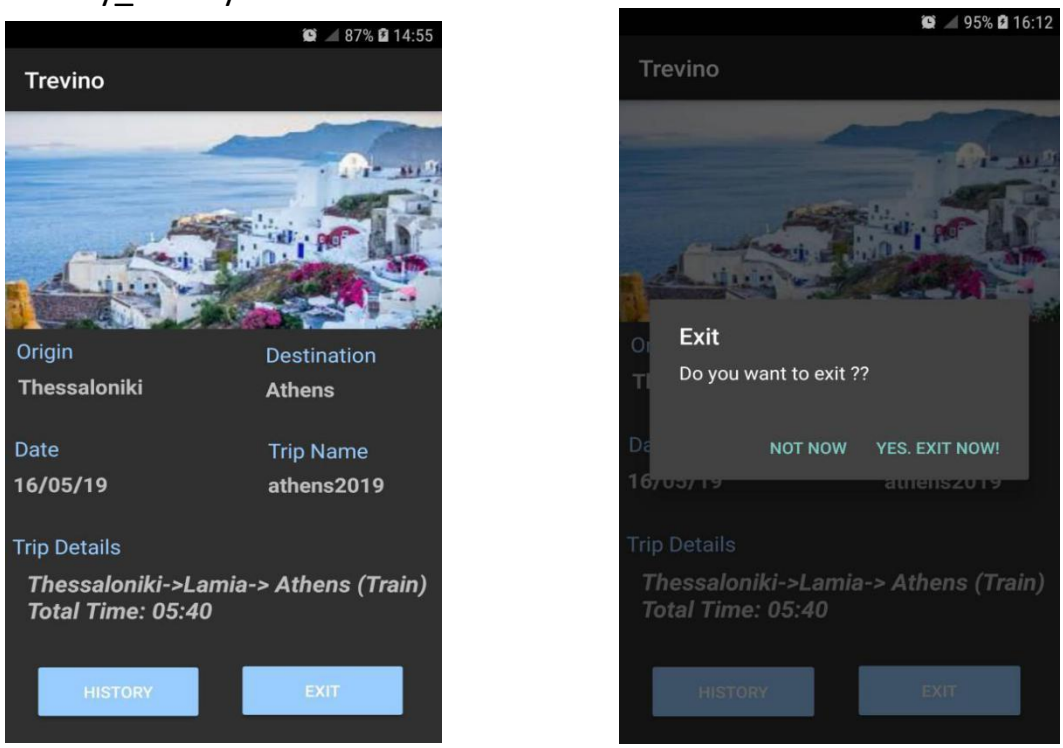
Στη συνέχεια, για την εισαγωγή προορισμού και λεπτομερειών μεταφερόμαστε στην επόμενη σελίδα και στη έπειτα, αφού γίνει η εισαγωγή προορισμού μεταφερόμαστε στη σελίδα παρουσίασης των συστάσεων :



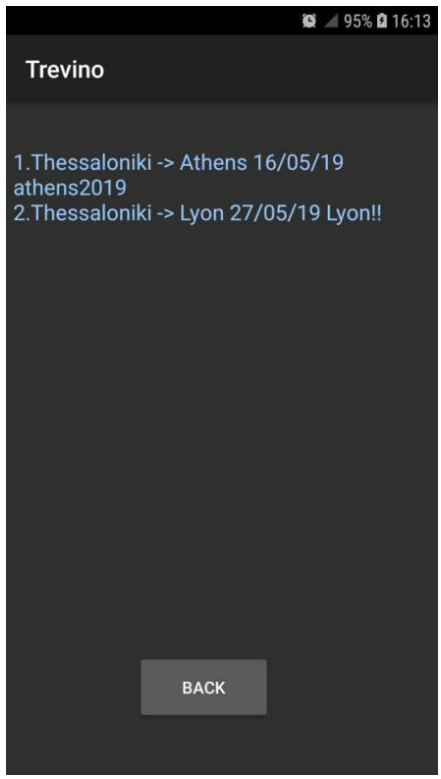
Εικόνα 52. Σελίδα εισαγωγής στοιχείων προορισμού και εμφάνιση συστάσεων.

Μετά την παρουσίαση των συστάσεων σε κουμπιά, ο χρήστης καλείται να επιλέξει ένα μονοπάτι διαδρομής. Το επιλεγμένο μονοπάτι, καθώς και οι λεπτομέρειες που εισήγαγε σχετικά με το πότε θα ταξιδέψει, τον αρχικό και τελικό προορισμό, αλλά και το όνομα του ταξιδιού του, εμφανίζονται στην επόμενη σελίδα. Έτσι, ο χρήστης μπορεί να έχει μια πιο ολοκληρωμένη εικόνα για την επιλογή του. Ακόμη, όπως φαίνεται και στην Εικόνα 53 δίνονται δύο επιλογές. Η μια αφορά την έξοδο από την εφαρμογή και η άλλη την εμφάνιση του ιστορικού επιλογών του χρήστη. Επιλέγοντας την έξοδο από την εφαρμογή, ο χρήστης μπορεί να διαλέξει αν θέλει σίγουρα να αποχωρήσει η όχι. Για την εμφάνιση του ιστορικού, ο χρήστης πρέπει να πατήσει το κουμπί “HISTORY”. Στην Εικόνα 54, με αριθμητική σειρά επιλογών από την παλαιότερη στην νεότερη, εμφανίζονται οι προορισμοί και οι λεπτομέρειες των ταξιδιών που έχει πραγματοποιήσει. Με το κουμπί “BACK”, ο χρήστης μεταφέρεται στη πρώτη σελίδα της Εικόνας 53. Για την εμφάνιση του

ιστορικού, χρησιμοποιήθηκε η κλάση History και το αρχείο activity_history.xml.



Εικόνα 53. Σελίδα εμφάνισης στοιχείων προορισμού, ιστορικού χρήστη και επιλογή εξόδου.



Εικόνα 54. Σελίδα ιστορικού χρήστη.

Όπως αναφέρθηκε και σε προηγούμενη ενότητα, στον πίνακα User_Travel_info αποθηκεύεται το ID_User, δηλαδή το ID του χρήστη

από τον πίνακα User. Έτσι, είναι εφικτή η εμφάνιση των προορισμών που καταχώρησε ο εκάστοτε χρήστης. Αρχικά, πρέπει να βρεθεί το ID του χρήστη από τον πίνακα User και μετά να γίνει η σύγκριση με το ID_User του πίνακα User_Travel_info. Για την εύρεση του ID γίνεται αναζήτηση με βάσει το email που χρησιμοποιεί για την σύνδεση στην εφαρμογή.

```
}
    public String find_User_ID(String email)
    {
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor;
        String sql="select ID from User where Email=?";
        cursor=db.rawQuery(sql,new String[]{email});
        cursor.moveToNext();
        user_id=cursor.getString(cursor.getColumnIndex( s: "ID"));

        return user_id;
    }
```

Εικόνα 55. Εύρεση του ID με βάση το email του χρήστη.

Για την καταχώριση και εμφάνιση των αποτελεσμάτων του ιστορικού χρησιμοποιείται η παρακάτω συνάρτηση:

```
public void history_array(String id)
{
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor;
    int counter=1;
    cursor=db.rawQuery( sql: "select Start_Destination,Final_Destination,Date,Trip_Title from User_Travel_info where ID_User=?",new String[]{id});

   if(cursor!=null && cursor.getCount() > 0) {
       if (cursor.moveToFirst()) {
           do {
               String start_dest = cursor.getString( i: 0);
               String stop_dest = cursor.getString( i: 1);
               String date = cursor.getString( i: 2);
               String title = cursor.getString( i: 3);
               sb.append(counter);
               sb.append(".");
               sb.append(start_dest);
               sb.append(" -> ");
               sb.append(stop_dest);
               sb.append(" ");
               sb.append(date);
               sb.append(" ");
               sb.append(title);
               sb.append(" ");
               sb.append("\n");
               counter++;
           } while (cursor.moveToNext());
       }
   }
}
```

Εικόνα 56. Εύρεση του ιστορικού του χρήστη και κατάταξη αποτελεσμάτων στην σελίδα.

Έπειτα από τα παραπάνω καταλήγουμε στο αποτέλεσμα της Εικόνας 54.

4. ΣΥΜΠΕΡΑΣΜΑ

Η παρούσα διπλωματική, πραγματεύεται την δημιουργία και παροχή συστάσεων για ταξιδιωτικούς προορισμούς σε δύο διαφορετικές ηλικιακές ομάδες. Η εμφάνιση των συστάσεων γίνεται με τη δημιουργία εγγενούς εφαρμογής τύπου Android. Στόχος της εργασίας είναι η δημιουργία διαδρομών με τη χρήση τόσο αεροπλάνων όσο και τρένων, ανάλογα με τον συνολικό χρόνο διαδρομής. Για την δημιουργία των συστάσεων χρησιμοποιήθηκε σύστημα κανόνων Drools σε Eclipse και για το σχεδιασμό των γραφικών και την δημιουργία της βάσης δεδομένων η πλατφόρμα του Android Studio.

ΑΝΑΦΟΡΕΣ

[1] Comparison of Collaborative Filtering Algorithms: Limitations of Current Techniques and Proposals for Scalable, High-Performance Recommender Systems**,** FIDEL CACHEDA, V´ICTOR CARNEIRO, DIEGO FERNA´NDEZ, and VREIXO FORMOSO, University of A Coruña.
[2] “Towards service composition based on mash up”, in Proc. of IEEE Congress on Services, X. Liu, Y. Hui, W. Sun, et al.
[3] G. Bianchi, “Performance analysis of the IEEE 802.11 distributed coordination function”, IEEE J. Sel. Areas Communication., Vol. 18, No. 3, PP. 535–547, Mar. 200
[4] “Αυτόματη Κατηγοριοποίηση Χρηστών ως Μέθοδος Επίλυσης του Προβλήματος Ψυχρής Εκκίνησης”, Μπλερίνα Π. Λίκα, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, Σχολή Θετικών Επιστημών, Τμήμα Πληροφορικής, Διπλωματική Εργασία, Ιούλιος 2013.
[5] “Αλγόριθμοι και Πολυπλοκότητα, Πιθανοτικοί Αλγόριθμοι”, Δισάσκοντες Σ. Ζαχος, Δ. Φωτάκης, Εθνικό Μετσόβειο Πολυτεχνείο, Χειμώνας 2010.
[6] IEEE/CAA JOURNAL OF AUTOMATICA SINICA, VOL. 6, NO. 1, JANUARY 2019 131, "Randomized Latent Factor Model for High - dimensional and Sparse Matrices from Industrial Applications", Mingsheng Shang, Xin Luo, Senior Member, IEEE, Zhigang Liu, Jia Chen, Ye Yuan,and Meng Chu Zhou, Fellow, IEEE.
[7] "ΓΡΑΦΟΘΕΩΡΗΤΙΚΕΣ ΜΕΘΟΔΟΛΟΓΙΕΣ ΓΙΑ ΤΗΝ ΔΗΜΙΟΥΡΓΊΑ ΣΥΣΤΗΜΑΤΩΝ ΣΥΣΤΑΣΗΣ", Μπαλτζή Βασιλική, Πανεπιστήμιο Πειραιώς, Σχολή Θετικών Επιστημών, Τμήμα Πληροφορικής, Διπλωματική Εργασία, Απρίλιος 2015.
[8] Pazzani, A Framework For Collaborative, Content ‐ Based And Demographic Filtering, 1999.
[9] Recommender Systems: An Overview, Robin Burke, Alexander Felfernig, and Mehmet H. Göker.
[10] L. Adrissono, A. Goy, G. Petrone, M. Segnan, P. Torasso, “ Intrigue: Personalized recommendation of tourist attractions for desktop and handheld devices”, Applied Artificial Intelligence, 2003, pp. 687 ‐ 714.
[11] Wikipidia: www.wikipidia.com.
[12] Netflix : https://help.netflix.com/en/node/412
[13] R. Bell; Y. Koren; C. Volinsky (2007). "The BellKor solution to the Netflix Prize”.
[14] Quora : www.Quora.com.

[15] "Microsoft will unify Windows, Windows Phone, and Xbox into 'one converged operating system'". ExtremeTech. Condé Nast. Retrieved October 31, 2015.
[16] https://mashable.com/article/build-mobile-apps/?europe=true
[17] "Ομαδικές συστάσεις βάσει περίπτωσης για διαμορφώσιμα προϊόντα με χρήση πολυδιάστατης ομαδοποίησης ", Παπαδημητρίου Α. Χαρίκλεια, Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, Σχολή Θετικών Επιστημών, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Διπλωματική Εργασία, Νοέμβριος 2016.
[18] G. Adomavicius, and A. Tuzhilin, Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions, IEEE Transactions on Knowledge and Data Engineering, vol. 17(6), pp. 734–749.
[19] P. Resnick, and H. R. Varian, Recommender Systems. Communications of the ACM, vol. 40(3), 1997, pp. 56–58.
[20] R. Burke, Hybrid Recommender Systems: Survey and Experiments, User Modeling and User-Adapted Interaction, vol. 12(4), 2002, pp. 331–370.
[21] R. Burke, A. Felfernig, and M. Goeker. Recommender Systems: An Overview, AI Magazine, AAAI, vol.32(3), 2011, pp. 13-18.
[22] Herlocker, J. (2004). "Evaluating Collaborative Filtering Recommender Systems." Oregon State University and JOSEPH A. KONSTAN, LOREN G. TERVEEN, and JOHN T. RIEDL - University of Minnesota.
[23] Goldberg, David; Oki, Brian; Nichols, David; Terry, Douglas B.; ``Using Collaborative Filtering to Weave an Information Tapestry,'' Communications of the ACM, December 1992, Vol 35, No 12, pp. 61-70.
[24] U. Shardanand, and P. Maes. Social information filtering: Algorithms for automating 'word of mouth'. In Proc. of the Conf. on Human Factors in Computing Systems, 1995.
[25] B. N. Miller, J. A. Konstan, and J. Riedl, "PocketLens: towarda personal recommender system," ACM Transactions on Information Systems, vol. 22, no. 3, pp. 437–476, 2004.
[26] Ungar, L. and Foster, D. (1998). Clustering methods for collaborative filtering. In Proceedings of the Workshop on Recommendation Systems. AAAI Press, Menlo Park California.
[27] Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI98), pages 43–52

[28] Luo Si and Rong Jin. Flexible mixture model for collaborative filtering. In Proc. of ICML, volume 3, pages 704–711, 2003.
[29] Thomas Hofmann. Latent semantic models for collaborative filtering. ACM Transactions on Information Systems (TOIS), 22(1):89–115, 2004.
[30] Chris Ding, Tao Li, Wei Peng, and Hae-sun Park. Orthogonal non negative matrix t-factorizations for clustering. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 126–135. ACM, 2006.
[31] Linden, G., Smith, B., York, J: Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing, 2003, pp. 76–80.
[32] Jennings, N., & Higuchi, H. (1993). A User Model Neural Network for a Personal News Service. Kluwer Academic.