ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Ανάπτυξη και κατασκευή γενικευμένης υπηρεσίας προ-κρατήσεων

Σάββας Γρηγόριος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέποντες:
Αναγνωστόπουλος Ιωάννης
Αναπληρωτής Καθηγητής
Λουκόπουλος Θανάσης
Επίκουρος Καθηγητής

Λαμία, 2017

.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Ανάπτυξη και κατασκευή γενικευμένης υπηρεσίας προ-κρατήσεων

Σάββας Γρηγόριος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέποντες:
Αναγνωστόπουλος Ιωάννης
Αναπληρωτής Καθηγητής
Λουκόπουλος Θανάσης
Επίκουρος Καθηγητής

Λαμία, 2017

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί* ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί* ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(Υπογραφή)

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

Ανάπτυξη και Κατασκευή Γενικευμένης Υπηρεσίας Προ-κρατήσεων

Σάββας Γρηγόριος

Τριμελής Επιτροπή:

Αναγνωστόπουλος Ιωάννης, Αναπληρωτής Καθηγητής

Κακαρούντας Αθανάσιος, Επίκουρος Καθηγητής

Λουκόπουλος Θανάσης, Επίκουρος Καθηγητής

ΠΕΡΙΛΗΨΗ

Στόχος της πτυχιακής εργασίας είναι η σχεδίαση και η ανάπτυξη ενός Web Service, το οποίο θα μπορεί να χρησιμοποιηθεί για γενικευμένου τύπου κρατήσεις. Το Web Service θα δίνει τη δυνατότητα στους χρήστες του να δημιουργήσουν και να προβάλλουν τις δικές τους διοργανώσεις, καθώς και να πραγματοποιούν κρατήσεις για τις διοργανώσεις αυτές.

Παράλληλα με το Web Service αναπτύχθηκε και μία ιστοσελίδα η οποία εξυπηρετεί τους παραπάνω σκοπούς. Επίσης, στα πλαίσια της εργασίας δημιουργήθηκαν δύο android εφαρμογές, οι οποίες αφορούν την εξυπηρέτηση των χρηστών που θέλουν να κάνουν κρατήσεις, καθώς και των χρηστών που θέλουν να δημιουργήσουν τις δικές τους εκδηλώσεις αντιστοίχως.

Στην εισαγωγή γίνεται γενική αναφορά στις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση των προαναφερθέντων εφαρμογών. Αυτές περιλαμβάνουν το είδος της αρχιτεκτονικής του Web Service και τα πρωτόκολλα επικοινωνίας, τις γλώσσες και τα εργαλεία για την ανάπτυξη ιστοσελίδων, τόσο σε client όσο σε server επίπεδο, καθώς και του συστήματος διαχείρισης της βάσης δεδομένων. Εν συνέχεια γίνεται γενική περιγραφή του συστήματος Android και των δυνατοτήτων του.

Στο δεύτερο κεφάλαιο επικεντρώνουμε στην αρχιτεκτονική του Web Service και στις δυνατότητες που προσφέρει στον χρήστη. Επίσης γίνεται εκτενής αναφορά στις λειτουργίες της ιστοσελίδας και πώς ο χρήστης δύναται να τις αξιοποιήσει.

Στο τρίτο κεφαλαίο αναλύουμε την πλευρά του Server με τον οποίο επικοινωνεί το Web Service, καθώς και τον server-side προγραμματισμό της ιστοσελίδας. Κατόπιν γίνεται περιγραφή της βάσης δεδομένων και το πώς γίνεται η επικοινωνία με τους client.

Στο τέταρτο κεφάλαιο παρουσιάζονται οι android εφαρμογές και οι δυνατότητες που προσφέρουν στους χρήστες τους. Επίσης δίνεται βάση στην επικοινωνία του συστήματος Android με τον Server, καθώς και των επιπρόσθετων λειτουργειών που προσφέρουν οι εφαρμογές αυτές.

Τέλος ακολουθεί ο επίλογος με τα συμπεράσματα της πτυχιακής εργασίας, καθώς και μία αναφορά στις επιπρόσθετες λειτουργίες που μπορούν να αναπτυχθούν για την αναβάθμιση των εφαρμογών που θα αναλυθούν

ΠΕΡΙΕΧΟΜΕΝΑ

1ο ΚΕΦΑΛΑΙΟ

ΕΙΣΑΓΩΓΗ

1.1 ΤΟ ΔΙΑΔΙΚΤΥΟ

1.1.1 ΕΙΣΑΓΩΓΗ

Το **Διαδίκτυο** (αγγλ. Internet) είναι παγκόσμιο σύστημα διασυνδεδεμένων δικτύων υπολογιστών, οι οποίοι χρησιμοποιούν καθιερωμένη ομάδα πρωτοκόλλων, η οποία συχνά αποκαλείται "TCP/IP" (αν και αυτή δεν χρησιμοποιείται από όλες τις υπηρεσίες του Διαδικτύου) για να εξυπηρετεί εκατομμύρια χρήστες καθημερινά σε ολόκληρο τον κόσμο. Οι διασυνδεδεμένοι ηλεκτρονικοί υπολογιστές ανά τον κόσμο, οι οποίοι βρίσκονται σε ένα κοινό δίκτυο επικοινωνίας, ανταλλάσσουν μηνύματα (πακέτα) με τη χρήση διαφόρων πρωτοκόλλων (τυποποιημένοι κανόνες επικοινωνίας), τα οποία υλοποιούνται σε επίπεδο υλικού και λογισμικού. Το κοινό αυτό δίκτυο καλείται Διαδίκτυο.

1.1.2 Η ΤΕΧΝΟΛΟΓΙΑ ΤΟΥ ΔΙΑΔΙΚΤΥΟΥ

Το Διαδίκτυο είναι επικοινωνιακό δίκτυο που επιτρέπει την ανταλλαγή δεδομένων μεταξύ οποιουδήποτε διασυνδεδεμένου υπολογιστή. Η τεχνολογία του είναι κυρίως βασισμένη στην διασύνδεση επιμέρους δικτύων ανά τον κόσμο και σε πολυάριθμα πρωτόκολλα επικοινωνίας. Στην πιο εξειδικευμένη και περισσότερο χρησιμοποιούμενη μορφή του, με τον όρο Διαδίκτυο περιγράφεται το παγκόσμιο πλέγμα διασυνδεδεμένων υπολογιστών και των υπηρεσιών και πληροφοριών που παρέχει στους χρήστες του. Το Διαδίκτυο χρησιμοποιεί μεταγωγή πακέτων και τη στοίβα πρωτοκόλλων. Σήμερα, ο όρος διαδίκτυο κατέληξε στο να αναφέρεται στο παγκόσμιο αυτό δίκτυο. Για να ξεχωρίζει, το παγκόσμιο αυτό δίκτυο γράφεται με κεφαλαίο το αρχικό "Δ". Η τεχνική της διασύνδεσης δικτύων μέσω μεταγωγής πακέτων και της στοίβας πρωτοκόλλων ονομάζεται διαδικτύωση.

Πηγή (*https://el.wikipedia.org/wiki/Διαδίκτυο)*

1.1.3 ΠΑΓΚΟΣΜΙΟΣ ΙΣΤΟΣ (WORLD WIDE WEB)

Κατά τη διάρκεια της εξέλιξης του δικτύου, εμφανίστηκαν και δοκιμάστηκαν πολλά εργαλεία, μέθοδοι και τρόποι αποθήκευσης και μετάδοσης της πληροφορίας, αρκετοί από τους οποίους πέτυχαν και κατάφεραν τελικά να καθιερωθούν. Ωστόσο το αποφασιστικό βήμα για την ολοκλήρωση της αναζήτησης της πληροφορίας στο Internet έγινε με την εισαγωγή μίας νέας υπηρεσίας που ονομάζεται World Wide Web

(WWW) ή αλλιώς WEB. Ο Παγκοσμίως Ευρύς Ιστός, με απλά λόγια το Web, είναι ένα δίκτυο υπολογιστών βασισμένο στο Internet, που επιτρέπει στους χρήστες ενός υπολογιστή να έχουν πρόσβαση σε πληροφορίες που είναι αποθηκευμένες σε κάποιο άλλο υπολογιστή μέσω αυτού του παγκόσμιου δικτύου.

Κάθε δίκτυο-δομική μονάδα του διαδικτύου αποτελείται από συνδεδεμένους υπολογιστές σε τοπικό επίπεδο, για παράδειγμα το δίκτυο υπολογιστών των κεντρικών γραφείων μιας εταιρίας. Αυτά τα δίκτυα με τη σειρά τους συνδέονται σε ευρύτερα δίκτυα, όπως εθνικά και υπερεθνικά. Το ευρύτερο δίκτυο στον κόσμο λέγεται παγκόσμιος ιστός το οποίο είναι μοναδικό (δηλαδή δεν υπάρχουν παραπάνω από ένα δίκτυα υπολογιστών παγκόσμιας κλίμακας), και συμπεριλαμβάνεται τόσο τα γήινα δίκτυα, όσο και τα δίκτυα των δορυφόρων της και άλλων διαστημικών συσκευών που είναι συνδεδεμένα σε αυτό. Η τεχνολογία του ιστού καθιστά δυνατή την δημιουργία "υπερκειμένων", μία διασύνδεση δηλαδή πάρα πολλών μη ιεραρχημένων στοιχείων που παλαιότερα ήταν απομονωμένα. Τα στοιχεία αυτά μπορούν να πάρουν και άλλες μορφές πέραν της μορφής του γραπτού κειμένου, όπως εικόνας και ήχου. Η τεχνολογία του ιστού δημιουργήθηκε το 1989 από τον Βρετανό Τιμ Μπέρνερς Λη, που εκείνη την εποχή εργαζόταν στον Ευρωπαϊκό Οργανισμό Πυρηνικών Ερευνών (CERN) στην Γενεύη της Ελβετίας. Το όνομα που έδωσε στην εφεύρεσή του ο ίδιος ο Lee είναι World Wide Web, όρος γνωστός στους περισσότερους από το "www".

Ο Web είναι δομημένος σε μία αρχιτεκτονική client / server. Σε μία απλουστευμένη προσέγγιση, ο server περιέχει τα λεγόμενα Web Documents, μέσα στα οποία βρίσκονται οι διάφοροι σύνδεσμοι και “διοχετεύει” τις πληροφορίες προς τον client, χειριζόμενος ταυτόχρονα και τις αιτήσεις για επικοινωνία και διασύνδεση με άλλους απομακρυσμένους υπολογιστές (hosts). Στο Web ο client αποκαλείται Web Browser. Ένας Web browser (φυλλομετρητής ιστοσελίδων, πλοηγός Web, πρόγραμμα περιήγησης Web ή περιηγητής Ιστού) είναι ένα λογισμικό που επιτρέπει στον χρήστη του να προβάλλει και να αλληλοεπιδρά με όλα τα Web Documents, που είναι συνήθως αναρτημένα σε μια ιστοσελίδα ενός ιστότοπου στον Παγκόσμιο Ιστό ή σε ένα τοπικό δίκτυο. Το πρωτόκολλο που χρησιμοποιείται από τον Web για την εκτέλεση όλων αυτών των διεργασιών είναι το HTTP (Πρωτόκολλο Μεταφοράς Υπερκειμένου), ενώ η HTML ((Γλώσσα Επισήμανσης Υπερκειμένου) αποτελεί την γλώσσα κατασκευής των Web Documents, της πληροφορίας που ρέει στο Web.

Πηγή
(*http://www.medialab.ntua.gr/education/MultimediaTechnology/MultimediaTechnologyNotes/chap3a_4.htm)*

1.2 WEB SERVICES

1.2.1 ΟΡΙΣΜΟΣ

Ως Web Service μπορούμε να χαρακτηρίσουμε μία προσφερόμενη, από μία ηλεκτρονική συσκευή, υπηρεσία προς μία άλλη ηλεκτρονική συσκευή, οι οποίες επικοινωνούν μεταξύ τους μέσω του παγκόσμιου ιστού. Σε ένα web service

τεχνολογίες διαδικτύου, οι οποίες αρχικά σχεδιάστηκαν για επικοινωνία ανθρώπου με υπολογιστή, χρησιμοποιούνται για επικοινωνία υπολογιστή με υπολογιστή, πιο συγκεκριμένα για τη μεταφορά αρχείων τύπου XML (eXtensible Markup Language) και JSON (JavaScript Object Notation). Στην πράξη, ένα web service συνήθως παρέχει μία αντικειμενοστραφή, βασισμένη στο διαδίκτυο, διεπαφή ενός διακομιστή μίας βάσης δεδομένων, η οποία μπορεί να χρησιμοποιηθεί π.χ. από έναν άλλο διακομιστή ή μία εφαρμογή για κινητά, τα οποία παρέχουν μία γραφική διεπαφή προς τον χρήστη. Πολλές εταιρίες χρησιμοποιούν διάφορα συστήματα για τη διαχείριση την δεδομένων τους. Τα συστήματα αυτά συχνά χρειάζονται να ανταλλάξουν πληροφορίες μεταξύ τους, έτσι ένα web service επιτρέπει την επικοινωνία μεταξύ των διαφορετικών συστημάτων μέσω του διαδικτύου. Το σύστημα που ζητάει δεδομένα ονομάζεται Service Requester, ενώ το σύστημα που τα επεξεργάζεται Service Provider.

Τα web services λοιπόν αποτελούν μία αρχιτεκτονική κατανεμημένων συστημάτων κατασκευασμένη από πολλά διαφορετικά υπολογιστικά συστήματα τα οποία επικοινωνούν μέσω του δικτύου ώστε να δημιουργήσουν ένα σύστημα. Αποτελούνται από ένα σύνολο από πρότυπα τα οποία επιτρέπουν στους υπεύθυνους για την ανάπτυξη (προγραμματιστές - developers) να υλοποιήσουν κατανεμημένες εφαρμογές (χρησιμοποιώντας διαφορετικά εργαλεία από διαφορετικούς προμηθευτές) ώστε να κατασκευάσουν εφαρμογές που χρησιμοποιούν ένα συνδυασμό από ενότητες λογισμικού (software modules) οι οποίες καλούνται από συστήματα που ανήκουν σε διαφορετικά τμήματα ενός οργανισμού ή σε διαφορετικούς οργανισμούς. **Πηγή (*Paul Deitel, Harvey Deitel 2011, Προγραμματισμός Ίντερνετ & World Wide Web 4η Έκδοση*)**

1.2.2 ΤΕΧΝΙΚΗ ΣΚΟΠΙΑ

Ενώ τα web services προσφέρουν όλα αυτά τα δυναμικά χαρακτηριστικά ώστε να συνδυάσουμε υπηρεσίες σε εφαρμογές, πρέπει πρώτα να χτίσουμε αυτές τις υπηρεσίες. Οι γλώσσες προγραμματισμού στην πληροφορική συνεχώς εξελίσσονται. Ξεκινήσαμε δεκαετίες πριν με την ιδέα της function στην οποία παρέχουμε μερικές παραμέτρους, εκτελεί μια λειτουργία με αυτές τις παραμέτρους, και επιστρέφει μια τιμή βασισμένη στους υπολογισμούς που έγιναν. Τελικά, αυτή η πρώτη έννοια εξελίχθηκε σε αντικείμενο (object) όπου κάθε αντικείμενο είχε όχι απλώς έναν αριθμό από λειτουργίες (functions) που μπορεί να εκτελέσει αλλά και τις δικές του ιδιωτικές μεταβλητές (private data variables), αντί να στηρίζεται σε εξωτερικές μεταβλητές του συστήματος που προηγουμένως έκαναν πιο περίπλοκη την ανάπτυξη εφαρμογών. Δεδομένου ότι οι εφαρμογές άρχισαν να επικοινωνούν μεταξύ τους, η έννοια του καθορισμού καθολικών διεπαφών (universal interfaces) για αντικείμενα έγινε σημαντική, επιτρέποντας αντικείμενα σε διαφορετικές πλατφόρμες να επικοινωνούν ακόμη και αν είχαν αναπτυχθεί σε διαφορετικές γλώσσες προγραμματισμού και εκτελούνταν σε διαφορετικά λειτουργικά συστήματα.

Τα web services προχώρησαν μπροστά με την έννοια των διεπαφών και επικοινωνιών καθορισμένων με XML, ενώνοντας τελικά κάθε είδους εφαρμογή με οποιαδήποτε άλλη, όπως και παρέχοντας την ελευθερία στις εφαρμογές αν αλλάξουν

και να εξελιχθούν με το χρόνο, αρκεί να είναι σχεδιασμένες σύμφωνα με την κατάλληλη διεπαφή. Η μεταβλητότητα της XML είναι αυτό που κάνει τα web services διαφορετικά από τεχνολογίες προηγούμενων γενεών. Επιτρέπει το διαχωρισμό της γραμματικής δομής (syntax) και της γραμματικής έννοιας (semantics), και του πώς αυτά υποβάλλονται σε επεξεργασία και κατανοούνται από μία υπηρεσία και το περιβάλλον μέσα στο οποίο υπάρχει. Έτσι λοιπόν τώρα, τα αντικείμενα μπορούν να καθοριστούν σαν υπηρεσίες, οι οποίες επικοινωνούν με άλλες υπηρεσίες σε γραμματική καθορισμένη σε XML, με την οποία κάθε υπηρεσία μεταφράζει και αναλύει το μήνυμα σύμφωνα με μην τοπική της υλοποίησης και το περιβάλλον της. Κατά συνέπεια μια διαδικτυακή εφαρμογή μπορεί πραγματικά να συντεθεί από πολλαπλές οντότητες διαφόρων υλοποιήσεων και σχεδιασμών εφόσον προσαρμόζονται στους κανόνες που καθορίζονται από την προσανατολισμένη στις υπηρεσίες αρχιτεκτονική τους.

Κατά συνέπεια, με αυτά στο μυαλό, τα web services μας επιτρέπουν :

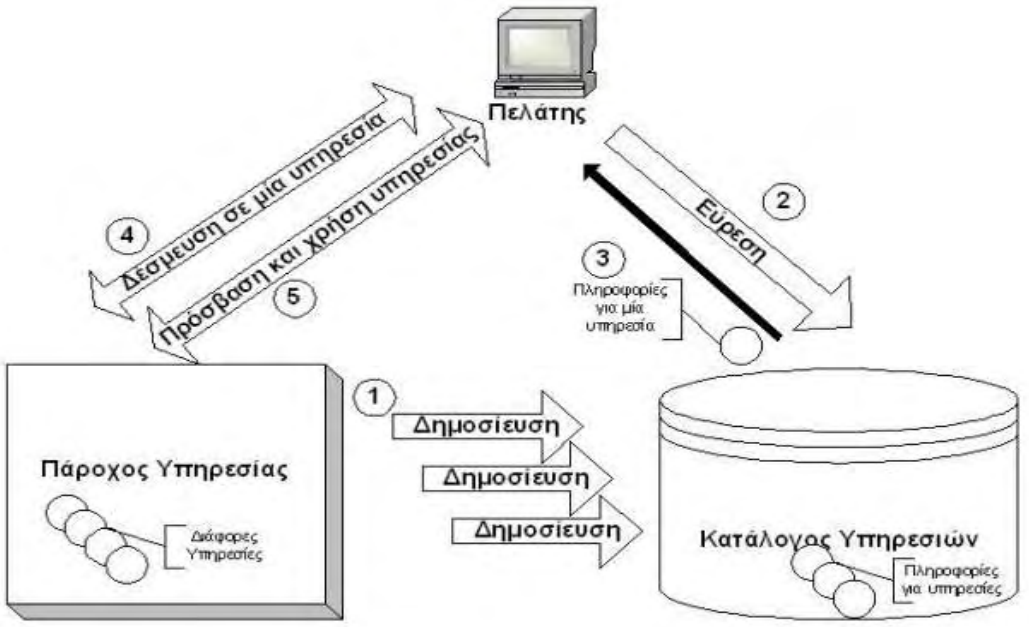
- Την αλληλεπίδραση μεταξύ υπηρεσιών σε οποιαδήποτε πλατφόρμα, γραμμένες σε οποιαδήποτε γλώσσα προγραμματισμού.
- Να αντιληφθούμε λειτουργίες εφαρμογών ως εργασίες, οδηγούμενοι σε ανάπτυξη και ροές εργασιών προσανατολισμένες σε εργασίες. Αυτό επιτρέπει μια υψηλότερη αφαίρεση του λογισμικού το οποίο μπορεί να υιοθετηθεί από λιγότερο τεχνικά καταρτισμένους χρήστες.
- Τη χαλαρή συνδεσιμότητα μεταξύ εφαρμογών, πράγμα που σημαίνει ότι αλληλεπιδράσεις μεταξύ υπηρεσιών δε θα χαλάνε κάθε φορά που υπάρχει κάποια αλλαγή το πώς μία ή περισσότερες υπηρεσίες σχεδιάζονται ή υλοποιούνται.
- Την προσαρμογή ήδη υπάρχων εφαρμογών στις μεταβαλλόμενες επιχειρησιακές συνθήκες και ανάγκες των πελατών.
- Να παρέχουμε υπάρχουσες εφαρμογές λογισμικού με διεπαφές υπηρεσιών χωρίς να αλλάξουμε τις αρχικές εφαρμογές, επιτρέποντάς τους να λειτουργούν πλήρως στο περιβάλλον των υπηρεσιών.
- Να εισάγουμε άλλες διοικητικές λειτουργίες ή λειτουργίες διαχείρισης διαδικασιών όπως η αξιοπιστία, υπευθυνότητα, ασφάλεια, κ.λπ., ανεξάρτητες της αρχικής λειτουργίας μιας εφαρμογής, αυξάνοντας κατά συνέπεια τη μεταβλητότητα και τη χρησιμότητά της στο επιχειρησιακό περιβάλλον.

Πηγή *(pdplab.it.uom.gr/project/soap/Theory/introduction.html*)

1.2.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ WEB SERVICES (SOA)

Το μοντέλο των web services ακολουθεί το παράδειγμα δημοσίευση (publish), εύρεση(find) και σύνδεση(bind). Στο πρώτο βήμα, ο προμηθευτής της υπηρεσίας δημοσιεύει την υπηρεσία σε ένα κατάλογο υπηρεσιών. Στο δεύτερο βήμα, ο πελάτης ο οποίος ψάχνει για μία υπηρεσία η οποία να καλύπτει τις απαιτήσεις του την αναζητεί στον κατάλογο. Αφού επιτυχημένα βρει πολλαπλές υπηρεσίες επιλέγει μία βάσει των προτιμήσεών του. Τότε μεταφορτώνει την περιγραφή της υπηρεσίας και

συνδέεται (δεσμεύεται) με αυτήν ώστε να μπορέσει να καλέσει και να εκτελέσει την υπηρεσία.



ΕΙΚΟΝΑ 1.1 – ΥΠΗΡΕΣΙΟΣΤΡΑΦΗΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗ

Το παραπάνω μοντέλο αποτελείται από τις 3 εξής οντότητες:

- **Service Provider (Πάροχος Υπηρεσίας):** Αποτελεί την οντότητα, η οποία συντάσσει την αντίστοιχη υπηρεσία στο Διαδίκτυο. Είναι υπεύθυνος για την επιλογή των εφαρμογών που θα παρέχονται στους χρήστες της συγκεκριμένης web service, το επίπεδο ασφάλειας και προσβασιμότητας καθώς και τον τρόπο κοστολόγησης. Η πιο σημαντική αρμοδιότητα του Παρόχου Υπηρεσίας ωστόσο είναι να δημιουργήσει τις κατάλληλες προϋποθέσεις ώστε η συγκεκριμένη υπηρεσία να είναι ορατή σε όλους τους χρήστες του Διαδικτύου. Για την συγκεκριμένη αρμοδιότητα απαιτείται η συνεργασία του με μια άλλη δομική μονάδα, την Service Registry (Υπηρεσία Καταλόγου). Ο Πάροχος υπηρεσίας περιγράφει τη υπηρεσία του ενώ παρέχει και κάποιες πληροφορίες πρόσβασης σε αυτή, στον κατάλογο υπηρεσίας. Ταυτόχρονα κατηγοριοποιεί την υπηρεσία μέσα στον κατάλογο και ορίζει τι είδους συμφωνίες απαιτούνται για να μπορεί κανείς να προσπελάσει την συγκεκριμένη λειτουργία.
- **Service Registry(Υπηρεσία Καταλόγου):** Η υπηρεσία καταλόγου αποτελεί μια σύγχρονη βιβλιοθήκη. Αρμοδιότητα της είναι η καταχώρηση μαζί με τις αντίστοιχες περιγραφές όλων των υπαρχόντων υπηρεσιών Διαδικτύου. Ο υπεύθυνος του κάθε καταλόγου αποφασίζει για το φάσμα χρηστών στους οποίους θα είναι ορατός ο κατάλογος, τη ποσότητα των καταχωρήσεων που θα υποστηρίζει καθώς και το επίπεδο ασφαλείας που θα παρέχει. Υπάρχουν δημόσιοι αλλά και ιδιωτικοί κατάλογοι, μερικοί καλύπτουν ένα ευρύ φάσμα υπηρεσιών ενώ άλλοι ειδικεύονται σε καταχωρήσεις συγκεκριμένων κλάδων.

Η συγκεκριμένη μονάδα αποτελεί στην ουσία τον συνδετικό κρίκο ανάμεσα στον Πάροχο και στον Χρήστη Υπηρεσίας. Εδώ δημοσιοποιείται η Περιγραφή της εκάστοτε Υπηρεσίας από τον Πάροχο, η οποία ταυτόχρονα αποτελεί το αποτέλεσμα της αναζήτησης του χρήστη στον κατάλογο.

- **Service Requestor(Αιτούμενος Υπηρεσίας):** Αποτελεί τον αιτούμενο της κάθε υπηρεσίας. Αρχικά αναζητά καταχωρήσεις μέσα από έναν αριθμό καταλόγων αντλώντας την περιγραφή της κάθε υπηρεσίας και καταλήγει στον εντοπισμό της ζητούμενης καταχώρησης καθώς και της διεύθυνσης της. Εδώ λήγει και ο ρόλος της ενδιάμεσης δομικής μονάδας. Ο χρήστης συνδέεται με τον πάροχο και καλεί το web service που εκείνος δημοσιεύει.

Πηγή (*pdplab.it.uom.gr/project/soap/Theory/architecture.html*)

1.2.4 SOAP WEB SERVICES

Το πρωτόκολλο SOAP (Simple Object Access Protocol) είναι ένα ανεξάρτητο πλατφόρμας πρωτόκολλο, που χρησιμοποιεί XML για να διευκολύνει τις κλήσεις απομακρυσμένων διαδικασιών, συνήθως επί HTTP. Το SOAP είναι ένα συνηθισμένο πρωτόκολλο για μεταβίβαση πληροφοριών ανάμεσα σε πελάτες υπηρεσιών web και σε υπηρεσίες web.

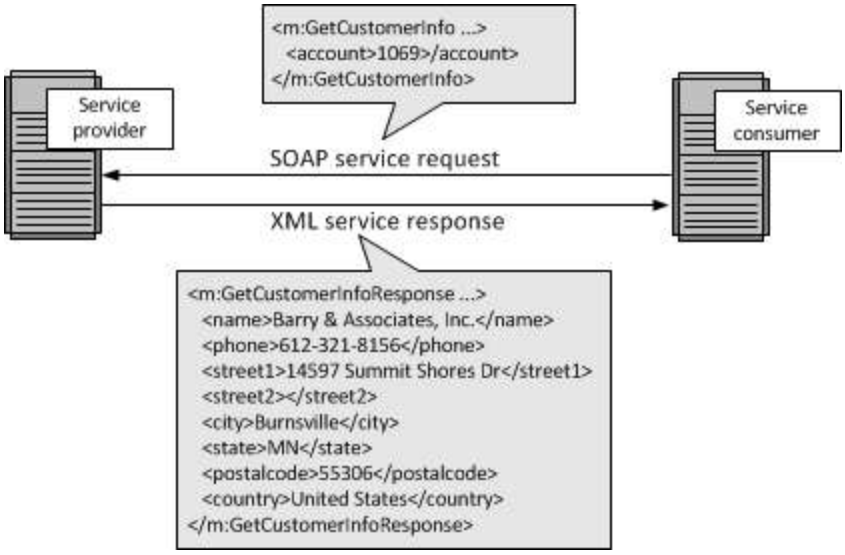
Κάθε αίτηση και απόκριση συσκευάζεται μέσα σ' ένα μήνυμα SOAP (επίσης γνωστό ως **φάκελος SOAP**) – μία "συσκευασία" XML που περιέχει τις πληροφορίες που απαιτεί μία υπηρεσία web για να επεξεργαστεί το μήνυμα. Τα μηνύματα SOAP γράφονται σε XML, έτσι ώστε να είναι ανεξάρτητα πλατφόρμας. Πολλά firewalls – τείχη προστασίας, που περιορίζουν την επικοινωνία ανάμεσα σε δίκτυα – διαμορφώνονται έτσι, ώστε να επιτρέπουν να περνά διαμέσου τους κίνηση HTTP. Έτσι η XML και το HTTP επιτρέπουν σε υπολογιστές, που λειτουργούν σε διαφορετικές πλατφόρμες να στέλνουν και να δέχονται μηνύματα SOAP με λίγους περιορισμούς.

Όταν ένα πρόγραμμα καλεί μία μέθοδο web, η αίτηση και όλες οι σχετικές πληροφορίες συσκευάζονται μέσα σε ένα μήνυμα SOAP και στέλνονται στο διακομιστή, στον οποίο βρίσκεται η υπηρεσία web. Η υπηρεσία web επεξεργάζεται τα περιεχόμενα του μηνύματος SOAP (που περιέχονται μέσα στο φάκελο SOAP), το οποίο καθορίζει τη μέθοδο που θέλει να καλέσει ο πελάτης και τα ορίσματα της μεθόδου. Αυτή η διαδικασία ερμηνείας των περιεχομένων ενός μηνύματος SOAP είναι γνωστή σαν ανάλυση ενός μηνύματος SOAP. Αφού η υπηρεσία web δεχθεί και αναλώσει μία αίτηση, καλείται η σωστή μέθοδος με τα καθορισμένα ορίσματα και η απόκριση στέλνεται πίσω στον πελάτη μέσα σε ένα άλλο μήνυμα SOAP. Ο πληρεξούσιος στην πλευρά του πελάτη αναλύει την απόκριση, η οποία περιέχει το αποτέλεσμα της κλίσης της μεθόδου και επιστρέφει το αποτέλεσμα στην εφαρμογή πελάτη.

ΕΙΚΟΝΑ 1.2 – ΦΑΚΕΛΟΣ SOAP

Τα βασικά χαρακτηριστικά του πρωτόκολλου SOAP είναι τα εξής :

1. **ΑΠΛΟΤΗΤΑ:** Εξαιτίας αυτού του χαρακτηριστικού του, το κόστος αλλά και η πολυπλοκότητα υλοποίησης του πρωτόκολλου SOAP μειώνονται αισθητά.
2. **ΕΥΕΛΙΞΙΑ:** Το συγκεκριμένο πρωτόκολλο κάνει χρήση προτύπων πρωτοκόλλων όπως το HTTP ως μέσα μεταφοράς. Συνεπώς είναι αρκετά ευέλικτο καθώς μπορεί να χρησιμοποιηθεί στο Διαδίκτυο χωρίς συμβιβασμούς στην ασφάλεια της υποδομής μιας επιχείρησης.
3. **ΑΝΕΞΑΡΤΗΣΙΑ:** Είναι ανεξάρτητο από πλατφόρμα και γλώσσα προγραμματισμού οπότε μπορεί να χρησιμοποιηθεί για την επικοινωνία μεταξύ εφαρμογών γραμμένων σε διαφορετικές γλώσσες προγραμματισμού και για διαφορετικές πλατφόρμες.
4. **ΕΠΕΚΤΑΣΙΜΟ:** Αποτελεί το πιο σημαντικό του χαρακτηριστικό καθώς έτσι μπορούν να βασιστούν επάνω του πολλές αναπτυσσόμενες τεχνολογίες υπηρεσιών παγκοσμίου ιστού προσφέροντας αξιοπιστία και ασφάλεια.
5.



ΕΙΚΟΝΑ 1.3 – ΠΑΡΑΔΕΙΓΜΑ SOAP WEB SERVICE

Κάθε μήνυμα SOAP ταυτόχρονα με όλες τις παραπάνω λειτουργίες του, υλοποιεί κλήσεις απομακρυσμένης διαδικασίας RPC(Remote Procedure Call). Πιο συγκεκριμένα για την κλήση μιας μεθόδου ενός απομακρυσμένου αντικειμένου μιας Web Service έχει καθοριστεί μια συγκεκριμένη αναπαράσταση μηνύματος SOAP, η οποία περιλαμβάνει στο σώμα του το όνομα της μεθόδου, τις επιτρεπόμενες παραμέτρους και το URL της. Το μοντέλο που ακολουθείται είναι κυρίως της μορφής αίτησης – απάντησης ενώ η όλη διαδικασία πραγματοποιείται μεταξύ του πελάτη και του διακομιστή.

Τα web service που ακολουθούν το πρωτόκολλο SOAP είθισται να επεξηγούν τις λειτουργίες που προσφέρουν μέσω μίας περιγραφικής γλώσσας, της WSDL (Web Services Description Language). Το W3C ορίζει ως WSDL: "Η WSDL είναι ένα σχήμα XML για την περιγραφή δικτυακών υπηρεσιών σαν ένα σύνολο από τελικά σημεία που λειτουργούν σε μηνύματα τα οποία περιέχουν πληροφορία είτε προσανατολισμένη στα έγγραφα είτε προσανατολισμένη στις διαδικασίες. Οι λειτουργίες και τα μηνύματα περιγράφονται περιληπτικά, και τότε δένονται σε ένα συγκεκριμένο πρωτόκολλο δικτύων και μορφή μηνυμάτων για να καθορίζουν ένα τελικό σημείο. Πολλά σχετικά τελικά σημεία συνδυάζονται σε υπηρεσίες.".

Με απλά λόγια η WSDL μας βοηθάει να περιγράψουμε ένα σύνολο από μηνύματα και τον τρόπο με τον οποίο αυτά τα μηνύματα ανταλλάσσονται. Η WSDL βασίζεται στην XML για την περιγραφή των δημόσιων διεπαφών (public interfaces) μιας υπηρεσίας παγκοσμίου ιστού. Η WSDL είναι ανεξάρτητη ως προς την πλατφόρμα και την γλώσσα προγραμματισμού. Αυτό το γεγονός την κάνει κατάλληλη για να περιγράψει διεπαφές web services, οι οποίες είναι προσβάσιμες από μια μεγάλη ποικιλία πλατφόρμων και γλωσσών προγραμματισμού. Επιπρόσθετα εκτός του ότι περιγράφει τα περιεχόμενα των μηνυμάτων, ορίζει που είναι διαθέσιμη μια υπηρεσία και ποια πρωτόκολλα επικοινωνίας χρησιμοποιούνται για να επικοινωνήσουμε με αυτήν. Επομένως κάθε αρχείο WSDL ορίζει όλα όσα χρειάζονται για να γράψουμε ένα πρόγραμμα το οποίο να λειτουργεί με ένα Web Service.

Πηγές (*Paul Deitel, Harvey Deitel 2011, Προγραμματισμός Ίντερνετ & World Wide Web 4η Έκδοση*), (*https://www.w3.org/TR/wsdl*)

1.2.5 REST-FULL WEB SERVICES

Το REST (REpresentative State Transfer) αρχικά προτάθηκε στη διδακτορική διατριβή του Roy Thomas το 2001. Το REST αναφέρεται σε ένα αρχιτεκτονικό στυλ, το οποίο αποτελεί έναν εναλλακτικό τρόπο υλοποίησης υπηρεσιών web από τη χρησιμοποίηση του πρωτοκόλλου SOAP. Αν και το REST δεν είναι ένα πρωτόκολλο (όπως το SOAP), οι υπηρεσίες web με δυνατότητες REST υλοποιούνται χρησιμοποιώντας πρωτόκολλα web, ως τα HTTP, XML και JSON. Κάθε λειτουργία μέσα σε μία υπηρεσία web με δυνατότητες REST αναγνωρίζεται εύκολα με ένα μοναδικό URL(Unique Recourse Locator). Έτσι όταν ο διακομιστής δέχεται μία αίτηση, γνωρίζει αμέσως ποια λειτουργία να εκτελέσει. Τέτοιες υπηρεσίες web

μπορούν να καλούνται από ένα πρόγραμμα ή απευθείας από ένα πρόγραμμα περιήγησης, εισάγοντας το URL μέσα στο πεδίο διεύθυνσης του προγράμματος περιήγησης. Σε ορισμένες περιπτώσεις, τα αποτελέσματα μίας συγκεκριμένης πράξης μπορούν να αποθηκευτούν (cached) τοπικά από το πρόγραμμα περιήγησης. Αυτό μπορεί να κάνει περισσότερες αιτήσεις για την ίδια λειτουργία ταχύτερες, φορτώνοντας το αποτέλεσμα κατευθείαν από την cache του προγράμματος περιήγησης. Πολλές υπηρεσίας του web βασίζονται στην αρχιτεκτονική REST.

Αρχικά ως διαδικτυακοί πόροι ορίζονταν μόνο έγγραφα και αρχεία που αναγνωρίζονταν από το URL τους, αλλά σήμερα ο όρος έχει γενικευτεί και περιλαμβάνει οτιδήποτε μπορεί να αναγνωριστεί, ονομαστεί και να διαχειριστεί με οποιοδήποτε τρόπο στο διαδίκτυο. Έτσι όταν γίνεται μία αίτηση σε ένα URI (Unique Resource Identifier) ενός πόρου, η απάντηση μπορεί να είναι τύπου XML, HTML, JSON ή κάποιου άλλου προκαθορισμένου προτύπου. Η απάντηση αυτή μπορεί να φέρει πληροφορίες σχετικά με τον αναφερόμενο πόρο ή να επιβεβαιώνει ότι έγινε κάποια αλλαγή σε αυτόν. Με τη χρήση του προτύπου HTTP, ως το πιο σύνηθες, τα είδη των λειτουργιών που επιτρέπονται περιλαμβάνουν τις προκαθορισμένες μεθόδους του προτύπου: GET, POST, PUT, DELETE κ.ο.κ. Χρησιμοποιώντας, λοιπόν, ένα stateless πρωτόκολλο και προκαθορισμένες μεθόδους, τα συστήματα REST στοχεύουν σε πιο γρήγορη λειτουργία, αξιοπιστία, και περιθώρια ανάπτυξης, επαναχρησιμοποιώντας υλικά που μπορούν να διαχειριστούν χωρίς να επηρεάζεται ολόκληρο το σύστημα, ακόμα και αν βρίσκεται εν λειτουργία.

ΕΙΚΟΝΑ 1.4 – ΠΑΡΑΔΕΙΓΜΑ RESTFULL WEB SERVICE

Όπως αναφέρθηκε το REST αποτελεί είδος αρχιτεκτονικής και όχι πρωτοκόλλου. Αυτό σημαίνει ότι, αν και όχι απόλυτα, για να θεωρηθεί μία υπηρεσία web ως REST πρέπει να ακολουθεί κάποιους κανόνες/περιορισμούς. Αυτοί οι περιορισμοί είναι οι ακόλουθοι:

- **ΠΕΛΑΤΗΣ-ΔΙΑΚΟΜΙΣΤΗΣ:** Οι υπηρεσίες web που βασίζονται σε REST αρχιτεκτονική ακολουθούν το μοντέλο του πελάτη-διακομιστή.

Διαχωρίζοντας την ανάπτυξη/σχεδίαση της διεπαφής του χρήστη από την ανάπτυξη/σχεδίαση του χώρου και τρόπου αποθήκευσης των δεδομένων, βελτιώνεται η δυνατότητα μεταφοράς της διεπαφής αυτής σε πολλαπλές πλατφόρμες, καθώς και η επεκτασιμότητα του διακομιστή αφού απλοποιούνται οι λειτουργίες αυτού. Όσον αφορά το διαδίκτυο, πιο σημαντικό είναι ότι η διαχώριση αυτή επιτρέπει την ατομική εξέλιξη των δύο αυτών στοιχείων (πελάτη – διακομιστή).

- **STATELESS:** Η επικοινωνία μεταξύ πελάτη και διακομιστή δεν πρέπει να επιτρέπει την αποθήκευση στοιχείων του πελάτη στον διακομιστή. Κάθε αίτηση ενός πελάτη περιέχει όλες τις πληροφορίες που είναι απαραίτητες ώστε να πραγματοποιηθεί η αίτηση αυτή. Οποιαδήποτε πληροφορία που χρειάζεται να γνωρίζει ο διακομιστής για τον πελάτη, πρέπει να αποθηκεύεται σε αυτόν και να αποστέλλεται με την αίτηση του. Τα στοιχεία αυτά που χρειάζεται να ξέρει ο διακομιστής για τον πελάτη, μπορούν να μεταφερθούν σε μία βάση δεδομένων, ώστε να διατηρηθεί η σύνδεση και να επιτρέπει πχ την ταυτοποίηση του χρήστη.
- **CACHEABLE:** Η απαντήσεις πρέπει να προσδιορίζουν τη δυνατότητα τους να μπορούν ή όχι να αποθηκευτούν στην κρυφή μνήμη (cache) του πελάτη, ώστε να αποτρέπουν αυτόν από το να επαναχρησιμοποιεί παλιότερα δεδομένα σε μελλοντικές αιτήσεις. Η σωστή χρήση της κρυφής μνήμης αποτρέπει την άσκοπη αποστολή αιτήσεων στον διακομιστή, βελτιώνοντας έτσι την απόδοση του.
- **ΠΟΛΥΕΠΙΠΕΔΟ ΣΥΣΤΗΜΑ:** Ο πελάτης δεν ξέρει αν συνδέεται άμεσα με το διακομιστή, ή σε κάποιον ενδιάμεσο. Οι ενδιάμεσοι διακομιστές μπορούν να βελτιώσουν την επεκτασιμότητα του συστήματος, καθώς επιτρέπουν την διαμοίραση των δεδομένων και παρέχουν κοινές κρυφές μνήμες. Μπορούν επίσης να επιβάλλουν πολιτικές ασφαλείας.
- **ΚΩΔΙΚΑΣ ΚΑΤΑ ΠΑΡΑΓΓΕΛΙΑ (προαιρετικό):** Ο διακομιστής μπορεί προσωρινά να επεκτείνει τις λειτουργίες του πελάτη, επιτρέποντας τη μεταφορά εκτελέσιμου κώδικα σε αυτόν, όπως πχ ενός JAVA Applet ή κώδικα JavaScript.
- **ΟΜΟΙΟΜΟΡΦΗ ΔΙΕΠΑΦΗ:** Η ομοιομορφία στη διεπαφή της υπηρεσίας web είναι και ο πιο θεμελιώδης περιορισμός της REST αρχιτεκτονικής. Απλοποιεί την αρχιτεκτονική και επιτρέπει κάθε τμήμα της να εξελίσσεται ξεχωριστά. Οι 4 αρχές αυτής της διεπαφής είναι:

  1. Αναγνώριση των πόρων: Οι επιμέρους πόροι προσδιορίζονται στην αίτηση, για παράδειγμα με τη χρήση URI σε υπηρεσίες web. Οι πόροι που στέλνονται στον πελάτη εννοιολογικά διαφέρουν από τις αναπαραστάσεις τους στον διακομιστή. Για παράδειγμα ο διακομιστής μπορεί να στείλει τα δεδομένα που αντιπροσωπεύουν ένα URI και που βρίσκονται αποθηκευμένα στη βάση του, ως HTML, XML ή JSON μορφή, κανένα από τα οποία δεν αντιπροσωπεύει την αναπαράσταση των δεδομένων αυτών όπως είναι αποθηκευμένα στη βάση αυτή.

2. Διαχείριση των πόρων μέσω του αναγνωριστικού τους: Όταν ο πελάτης γνωρίζει την αναπαράσταση ενός πόρου (πχ το URI), τότε έχει επαρκείς πληροφορίες ώστε να τον επεξεργαστεί ή να το διαγράψει.

3. Αυτό-προσδιοριζόμενα μηνύματα: Κάθε μήνυμα εμπεριέχει αρκετές πληροφορίες ώστε να περιγράφει το πώς θα το επεξεργαστεί ο πελάτης.

4. HATEOAS (Hypermedia As The Engine Of Application State): Όταν ένας πελάτης μίας REST υπηρεσίας αποκτήσει πρόσβαση σε ένα URI, τότε θα πρέπει να βρίσκεται σε θέση να ανακαλύψει δυναμικά όλες τις διαθέσιμες λειτουργίες που μπορεί να ακολουθήσει από εκεί που βρίσκεται. Αυτό πραγματοποιείται από το διακομιστή, ο οποίος στην απάντηση του προς τον πελάτη θα πρέπει να περιέχει συνδέσμους για τις επικείμενες λειτουργίες που διατίθενται.

Πηγές (*Paul Deitel, Harvey Deitel 2011, Προγραμματισμός Ίντερνετ & World Wide Web 4η Έκδοση),*
(https://en.wikipedia.org/wiki/Representational_state_transfer)

1.3 ΤΟ ΛΕΙΤΟΥΡΓΙΚΟ ΣΥΣΤΗΜΑ ANDROID

1.3.1 ΕΙΣΑΓΩΓΗ

Το Android είναι ένα λειτουργικό σύστημα ανοιχτού κώδικα, που είναι βασισμένο στο Linux σχεδιασμένο για κινητές συσκευές με οθόνες αφής όπως τα έξυπνα κινητά και τα tablets. Αρχικά σχεδιάστηκε για συσκευές οι οποίες έχουν οθόνη αφής και αργότερα χρησιμοποιήθηκε σε άλλες συσκευές όπως τηλεοράσεις και κονσόλες. Αναπτύχθηκε από την Google και αργότερα από την Open Handset Alliance. Η πρώτη παρουσίαση της πλατφόρμας Android έγινε στις 5 Νοεμβρίου 2007, παράλληλα με την ανακοίνωση της ίδρυσης του οργανισμού Open Handset Alliance

Χάρη στην Google και το Android ο προγραμματισμός σε smartphone αλλάζει τα δεδομένα στο χώρο. Ο κύριος λόγος πίσω από αυτή τη ραγδαία επιτυχία των Android εφαρμογών είναι η πλατφόρμα ανοικτού κώδικα, που εξαλείφει τους περιορισμούς που είναι κοινοί σε άλλες πλατφόρμες. Αυτό καθιστά την ανάπτυξη εφαρμογών σε Android πολύ πιο διασκεδαστική ακόμα και για αρχάριους χρήστες που έχουν προσχωρήσει σε αυτό το πεδίο. Οι συσκευές Android δίνουν στους χρήστες την ευχέρεια να διαχειρίζονται το λειτουργικό σύστημα και να το τροποποιούν ανάλογα με τις απαιτήσεις τους. Αυτό παρέχει πολλές ευκαιρίες για τους χρήστες του Android να έχουν ένα λειτουργικό σύστημα στα τηλέφωνά τους, που μπορεί να φιλοξενήσει κάθε είδους εφαρμογή σε Android, ανάλογα με τι επιθυμεί ο καθένα τους.

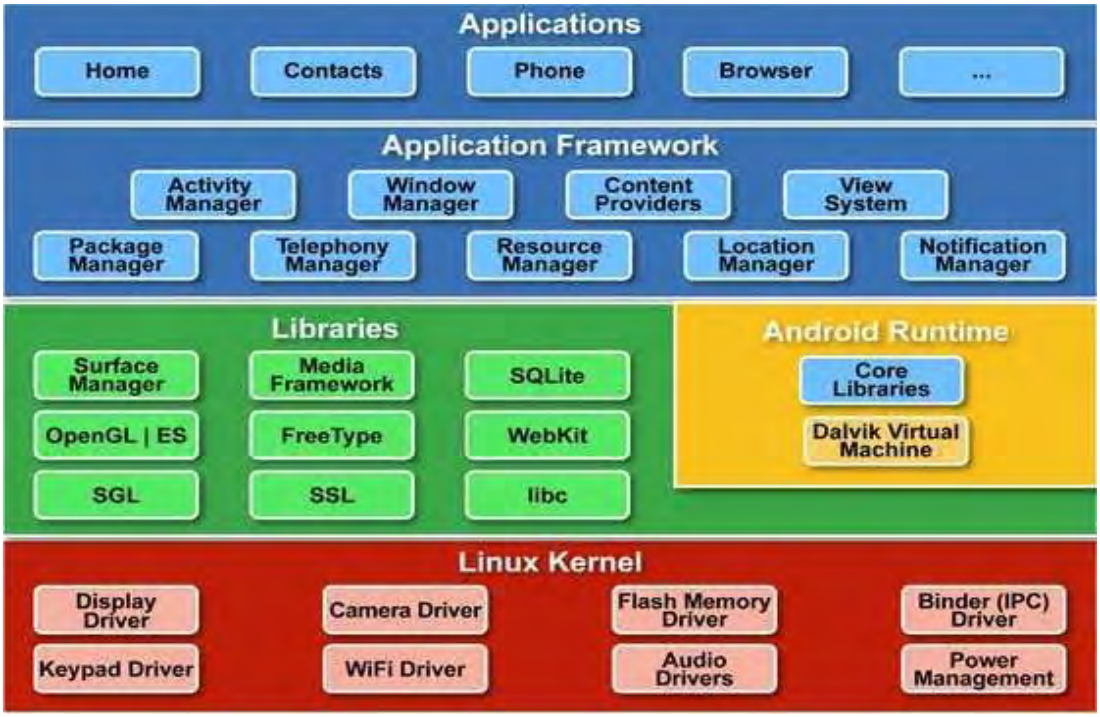
Το Android παρέχει ένα πιο ευέλικτο περιβάλλον για να σχεδιάσουμε και να δημιουργήσουμε μια εφαρμογή και είναι ένας πιο ασφαλής τρόπος για να εξασκηθεί κάνεις στο app design από τι σε άλλα λειτουργικά συστήματα.
Πηγή (*Butler Margaret 2011, Pervasive Computing, IEEE [Volume 10, Issue 1]*)

1.3.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ANDROID

Το λειτουργικό σύστημα Android είναι μια στοίβα λογισμικού η οποία αποτελείται από 5 βασικά κομμάτια:

Το λειτουργικό σύστημα Linux

- τις βιβλιοθήκες (Libraries)
- το χρόνο εκτέλεσης (Android Runtime)
- το πλαίσιο εφαρμογής (Application Framework)
- τις εφαρμογές Android (applications)



ΕΙΚΟΝΑ 1.5: ΑΡΧΙΤΕΚΤΟΝΙΚΗ ANDROID

1) Πυρήνας Linux (Linux Kernel)

Η βασική στρώση της στοίβας του Android είναι ο πυρήνας Linux. Από τον Απρίλιο του 2014, οι συσκευές Android χρησιμοποιούν κυρίως τις εκδόσεις 3.4, 3.10 ή 3.18 του πυρήνα αυτού. ο όποιος παρέχει βασικές υπηρεσίες χαμηλού επιπέδου όπως η διαχείριση μνήμης, η διαχείριση διεργασιών και η διαχείριση ενέργειας όπως επίσης να παρέχει μια στοίβα δικτύου και διάφορους οδηγούς υλικού (hardware drivers) για την οθόνη, το Wi-Fi και τον ήχο.

2) Βιβλιοθήκες (Libraries)

Το επόμενο επίπεδο της στοίβας περιέχει τις εγγενείς βιβλιοθήκες (Native Libraries). Μπορούμε να σκεφτούμε τις βιβλιοθήκες ως ένα σύνολο οδηγιών που λένε στη συσκευή πώς να διαχειριστεί διαφορετικούς τύπους δεδομένων. Οι βιβλιοθήκες αυτές είναι προγραμματισμένες σε γλώσσα C ή C++ και έχουν μεταγλωττιστεί έτσι ώστε να χρησιμοποιηθούν από το λειτουργικό σύστημα.

3) Android Runtime

Το Android Runtime περιλαμβάνει τις κύριες βιβλιοθήκες και το ART (Android Runtime, αντικατέστησε το Dalvik Virtual Machine από την έκδοση 5.0). Εδώ βρίσκουμε το κοινό σημείο επαφής μεταξύ των δυνατοτήτων που παρέχουν οι βιβλιοθήκες και του ART, τις λειτουργίες του οποίου περιγράφονται παρακάτω.

- Οι κύριες βιβλιοθήκες (Core Libraries) είναι προγραμματισμένες σε Java και για την ανάπτυξη των εφαρμογών. Ο χρήστης μέσων αυτών των βιβλιοθηκών έχει πρόσβαση σε διάφορα API.

- Σχεδόν όλα τα API του Android είναι βασισμένα στη γλώσσα προγραμματισμού java. Το ART είναι η εικονική μηχανή η οποία διαβάζει και εκτελεί το κώδικα των εφαρμογών στην Android συσκευή.

4) Πλαίσιο Εφαρμογής (Application Framework)

Το επίπεδο αυτό παρέχει εργαλεία (API) τα οποία μπορούμε να τα χρησιμοποιήσουμε για τη κατασκευή εφαρμογών. Τα προγράμματα αυτά διαχειρίζονται τις βασικές λειτουργείες του τηλεφώνου όπως τη διαχείριση των πόρων της συσκευής και τη διαχείριση των κλήσεων ομιλίας.

5) Εφαρμογές Android (Applications)

Στο υψηλότερο επίπεδο της αρχιτεκτονικής του Android βρίσκονται οι εφαρμογές οι οποίες χρησιμοποιούνται από το τελικό χρήστη. Με την εγκατάσταση διαφόρων εφαρμογών ο χρήστης μπορεί να αναβαθμίσει το κινητό του προσθέτοντας σε αυτό περισσότερες λειτουργίες.

Πηγή *(Yao Yumin, Liu Weiguo 2008. Study of Android's Architecture and its Application Development, Compute Systems & Applications*)

Ο καθένας μπορεί να κατεβάσει εφαρμογές από το Google Play, τη πλατφόρμα που έχει δημιουργήσει η Google για τη διαμοίραση των εφαρμογών Android. Το Google Play μέχρι το Φεβρουάριο του 2017 αριθμεί περισσότερες από 2.7 δισεκατομμύρια εφαρμογές και πάνω από 1.4 δισεκατομμύρια συσκευές σε χρήση, πράγμα που τη κάνει την πιο πετυχημένη σε πωλήσεις πλατφόρμα στο κόσμο για συσκευές Android.

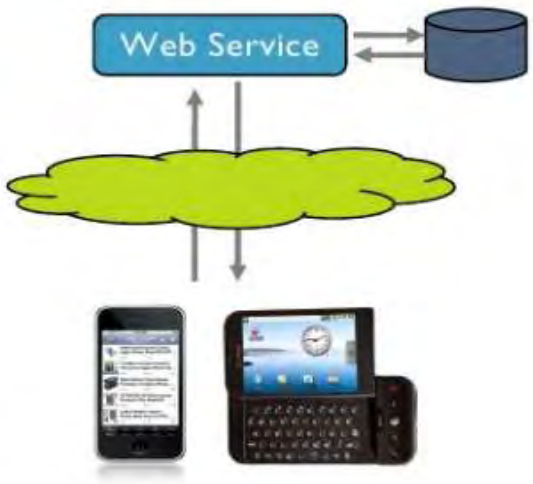
Πηγή (*http://en.wikipedia.org/wiki/Google_Play*)

1.3.3 ANDROID ΚΑΙ WEB SERVICES

Καθώς οι τεχνολογίες εξελίσσονται, οι εταιρίες προσπαθούν να βρουν τον πιο οικονομικό τρόπο ώστε να παρέχουν τις υπηρεσίες web τους σε εφαρμογές κινητών.

Το δίλλημα που τίθεται είναι η επιλογή μεταξύ SOAP ή REST. Η πιο διαδεδομένη επιλογή είναι αυτής της αρχιτεκτονικής REST, αλλά υπάρχουν και μεγάλες επιχειρήσεις που χρησιμοποιούν το πρωτόκολλο SOAP ή κάποιο άλλο σύστημα που δεν υποστηρίζει REST ή JSON. Για να καταλήξουμε στο πια επιλογή είναι η πιο συμφέρουσα, πρέπει να ληφθούν υπόψη οι περιορισμοί που έρχονται με τη χρήση των κινητών συσκευών, όπως το περιορισμένο bandwidth, η μικρότερη υπολογιστική δύναμη, περιορισμένη χωρητικότητα κ.ά. Η υπηρεσίες web που βασίζονται σε REST υπερέχουν σε κάποια σημεία από αυτές που βασίζονται σε SOAP.

- **ΟΓΚΟΣ ΔΕΔΟΜΕΝΩΝ:** Το μέγεθος των δεδομένων που διακινείται μέσω REST είναι αισθητά μικρότερος από το SOAP, συνεπώς επιτυγχάνεται και μικρότερη κίνηση στο δίκτυο, το οποίο είναι σημαντικό για κινητές συσκευές, αν αναλογιστεί κάποιος το περιορισμένο bandwidth.
- **ΑΠΟΔΟΤΙΚΟΤΗΤΑ:** Η αρχιτεκτονική REST παρέχει έναν πιο γρήγορο και οικονομικό τρόπο να αναλυθούν τα δεδομένα που στέλνει ο server με τη χρήση του προτύπου JSON, το οποίο είναι πιο ελαφρύ από την εξαγωγή δεδομένων που είναι «συσκευασμένα» σε ένα φάκελο SOAP.
- **CACHING:** Η υποστήριξη του caching από τις υπηρεσίες REST παρέχει ταχύτερες αποκρίσεις από το διακομιστή καθώς και πιο ευέλικτη χρήση του δικτύου.



ΕΙΚΟΝΑ 1.6 – ΕΠΙΚΟΙΝΩΝΙΑ ΚΙΝΗΤΟΥ ΜΕ WEB SERVICE

Τέλος, το Android παρέχει τρόπους που καθιστούν δυνατή την κατανάλωση μίας υπηρεσίας web. Συνήθως προτιμούνται οι REST υπηρεσίες, καθώς το λειτουργικό παρέχει τα μέσα για να γίνει η κλίση μιας τέτοιας υπηρεσίας, σε αντίθεση με το SOAP, που συνήθως χρησιμοποιούνται εξωτερικές βιβλιοθήκες. ***Πηγή (Incorporating Web Services in Mobile Applications - Web 2.0 2009)***

2□ □□□ □□□ □□

CLIENT SIDE ΑΝΑΛΥΣΗ ΤΟΥ WEB SERVICE ΚΑΙ WEBSITE

2.1 ΕΙΣΑΓΩΓΗ

Σε αυτό το κεφάλαιο γίνεται η ανάλυση της ιστοσελίδας, καθώς και του web service από την πλευρά του χρήστη. Αρχικά γίνεται μία γενική αναφορά στην δομή της ιστοσελίδας και στις υπηρεσίες που παρέχει προς τον χρήστη. Κατόπιν αναλύονται εκτενώς οι υπηρεσίας αυτές μαζί με τα αντίστοιχα interface, δίνοντας ιδιαίτερη βάση στην υλοποίηση τους και στο πως μπορεί να τις χρησιμοποιήσει ο κάθε χρήστης. Ακολουθεί η περιγραφή και η ανάλυση του web service που ανταποκρίνεται στους ίδιους σκοπούς με την ιστοσελίδα, και πιο συγκεκριμένα στους τρόπους με τους οποίους μπορεί ο χρήστης να καταναλώσει τις προσφερόμενες υπηρεσίες, καθώς και ποιες είναι οι υπηρεσίες αυτές.

2.2 ΑΝΑΛΥΣΗ ΤΗΣ ΙΣΤΟΣΕΛΙΔΑΣ

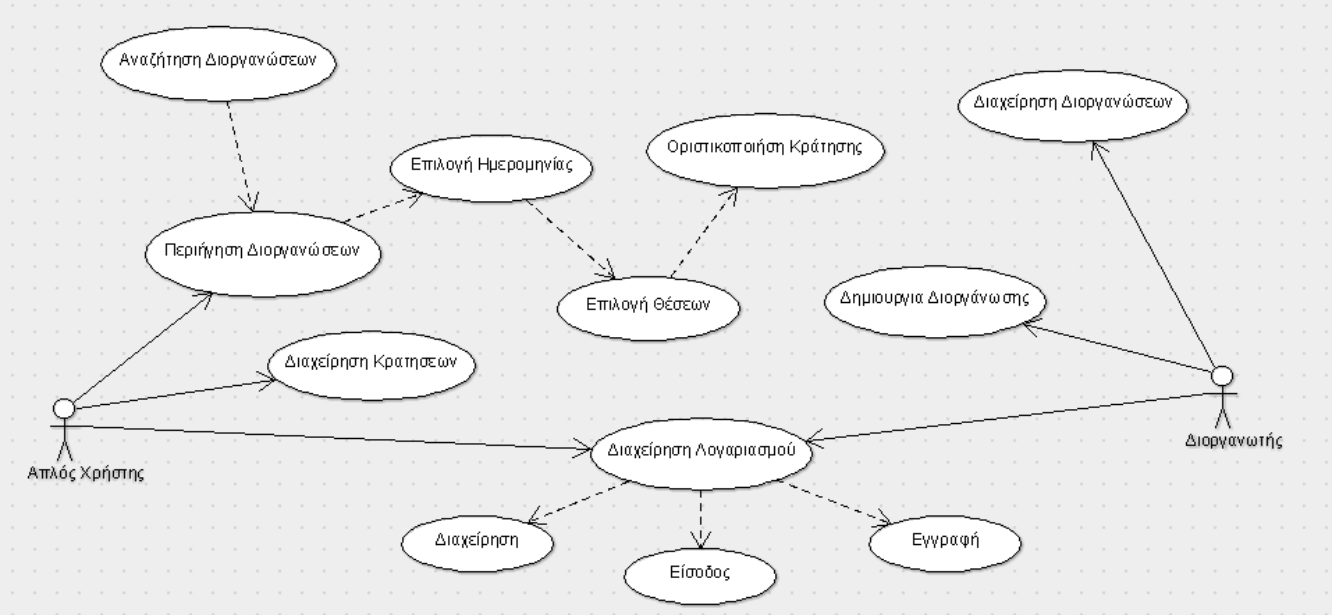
2.2.1 ΣΤΟΧΟΣ ΚΑΙ ΛΕΙΤΟΥΡΓΙΕΣ

Ο στόχος αυτής της ιστοσελίδας είναι να παρέχει στους χρήστες της μία πλατφόρμα, μέσω της οποίας θα μπορούν να δημοσιεύουν οποιαδήποτε εκδήλωση ή γεγονός διοργανώνουν και να προσφέρουν τις θέσεις ή τα εισιτήρια αυτών των διοργανώσεων προς κράτηση.

Οι χρήστες έχουν τη δυνατότητα να εγγραφούν στην ιστοσελίδα επιλέγοντας μία από τις δύο ιδιότητες που παρέχονται:

- **Διοργανωτές:** Ως διοργανωτές οι χρήστες έχουν τη δυνατότητα να δημιουργήσουν και να δημοσιεύσουν τις δικές τους εκδηλώσεις (events), καθώς και να επιλέξουν τον τρόπο με τον οποίο θέλουν να προσφέρουν τα εισιτήρια τους. Αφού δημοσιευτεί μια εκδήλωση, έχουν τη δυνατότητα να διαχειριστούν οποιαδήποτε πληροφορία σχετίζεται με αυτήν, όπως στοιχεία της εκδήλωσης, διαθέσιμες ημερομηνίες, προπώληση κ.ά.
- **Απλοί χρήστες:** Ως απλοί χρήστες έχουν τη δυνατότητα να περιηγηθούν στις διαθέσιμες διοργανώσεις, καθώς και να κάνουν κρατήσεις σε κάποια εκδήλωση που τους ενδιαφέρει. Αφού ολοκληρώσει την κράτηση ο χρήστης

λαμβάνει τα στοιχεία αυτής, όπως το τελικό κόστος των θέσεων που επέλεξε, το αναγνωριστικό των θέσεων αυτών, στοιχεία για την εκδήλωση καθώς και τη δυνατότητα ακύρωσης της.



ΕΙΚΟΝΑ 2.1 – USE CASE ΔΙΑΓΡΑΜΜΑ

Αφού ο χρήστης εγγραφεί και συνδεθεί στην ιστοσελίδα, τότε ανάλογα με την ιδιότητα του, εμφανίζονται και οι ανάλογες επιλογές ώστε να περιηγηθεί σε αυτήν. Για λόγους ατομικότητας, όπως ένας απλός χρήστης δεν μπορεί να δημιουργήσει μία νέα εκδήλωση, έτσι και ένας διοργανωτής δεν μπορεί να κάνει αντίστοιχη κράτηση.

2.2.2 TEMPLATE

Ως βάση για την ιστοσελίδα χρησιμοποιήθηκε ένα template, το οποίο δημιουργήθηκε με το πρόγραμμα Artisteer. Το Artisteer είναι ένας αυτόματος WYSIWYG (What You See Is What You Get) editor, που έχει τη δυνατότητα να παράγει βασικά templates στον χρήστη, προσαρμοσμένα στις απαιτήσεις τους. Στα πλαίσια της εργασίας αυτής έγινε χρήση ενός δωρεάν template, που παρέχεται από το www.artisteer.com. Το template αυτό, παρείχε το βασικό layout της ιστοσελίδας, καθώς και μέρος της μπάρας πλοήγησης.

2.2.3 ΑΡΧΙΚΗ ΣΕΛΙΔΑ

Στην αρχική σελίδα υπάρχει η λίστα με τις διαθέσιμες προς κράτηση εκδηλώσεις, ταξινομημένες με χρονολογική σειρά. Επιπλέον υπάρχουν τα εργαλεία για να εγγραφεί κάποιος στο site, καθώς και πληροφορίες για την ιστοσελίδα και το αντίστοιχο web service, η ανάλυση του οποίου θα γίνει παρακάτω.

ΕΙΚΟΝΑ 2.2 – ΑΡΧΙΚΗ ΣΕΛΙΔΑ

Όπως φαίνεται από την παραπάνω εικόνα οι δημοσιευμένες εκδηλώσεις αποτελούνται από δύο μέρη. Την εικόνα και τις πληροφορίες. Οι πληροφορίες που δίνονται είναι ο τίτλος της εκδήλωσης, το μέρος που λαμβάνει χώρα, οι διαθέσιμες ημερομηνίες και η περιγραφή. Ο χρήστης επιλέγει την ημερομηνία που τον ενδιαφέρει και έπειτα τα εισιτήρια ή τις θέσεις που προσφέρονται. Επίσης μπορεί να εγγραφτεί στο site ως απλός χρήστης ή διοργανωτής, καθώς και να κάνει είσοδο σε αυτό. Τέλος υπάρχει μία σύντομη περιγραφή των λειτουργιών της ιστοσελίδας και ένας σύνδεσμος που οδηγεί στο documentation του web service.

Οι εκδηλώσεις περικλείονται σε ένα table layout, του οποίου κάθε γραμμή είναι μία διαφορετική εκδήλωση. Για λόγους ταχύτητας και διευκόλυνσης στην πλοήγηση εμφανίζονται δέκα εκδηλώσεις τη φορά. Όταν υπάρχουν παραπάνω από δέκα δημοσιευμένα γεγονότα τότε εμφανίζεται ένα έξτρα κουμπί «next», το οποίο φορτώνει τα επόμενα διαθέσιμα δέκα γεγονότα. Αντιστοίχως όταν αλλάξει σελίδα ο χρήστης εμφανίζεται το κουμπί «previous» που τον επαναφέρει στην προηγούμενη.

ΕΙΚΟΝΑ 2.3 – ΤΑ ΚΟΥΜΠΙΑ PREVIOUS ΚΑΙ NEXT

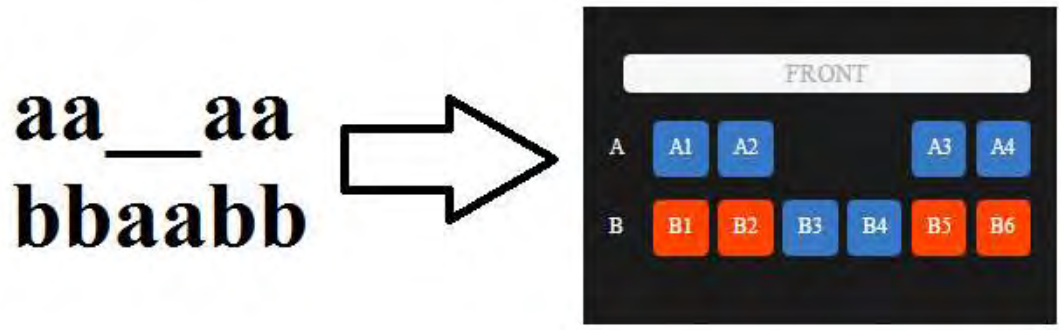
2.2.4 ΕΜΦΑΝΙΣΗ ΘΕΣΕΩΝ/ΕΙΣΙΤΗΡΙΩΝ

Με την επιλογή της ημερομηνίας που ενδιαφέρει τον χρήστη, γίνεται μετάβαση στη σελίδα παρουσίασης των διαθέσιμων θέσεων. Ένας διοργανωτής μπορεί να επιλέξει έναν από τους 2 παρακάτω τρόπους για να διαθέσει τα εισιτήρια του.

- **Χρήση floor plan:** Ο διοργανωτής έχει την επιλογή να ανεβάσει ένα χάρτη με τις θέσεις της εκδήλωσής του. Σε αυτήν την περίπτωση προβάλλεται στον χρήστη, που θέλει να κάνει κράτηση, ένας πίνακας με τη χωρική αναπαράσταση των θέσεων, από τις οποίες μπορεί να επιλέξει.
- **Μη χρήση floor plan:** Αν δεν υπάρχει floor plan, τότε ο χρήστης μπορεί να επιλέξει τον αριθμό των εισιτηρίων που θέλει από κάθε κατηγορία και ο server του παρέχει τον ζητούμενο αριθμό αυτομάτως.

Ανάλογα με το αν υπάρχει διαθέσιμο floor plan ή όχι, εμφανίζεται και η αντίστοιχη προβολή.

Όταν ένας διοργανωτής επιλέξει τη χρήση floor plan, του ζητείται να ανεβάσει ένα αρχείο κειμένου που περιγράφει το map του χώρου. Το floor plan αρχείο πρέπει να αποτελείται από αύξων λατινικούς χαρακτήρες (a για την πρώτη κατηγορία, b για τη δεύτερη κ.ο.κ.) και κάτω παύλες «_». Κάθε λατινικός χαρακτήρας αντιπροσωπεύει και μία κατηγορία εισιτηρίου, ενώ οι κάτω παύλες τον κενό χώρο. Στο παράδειγμα που ακολουθεί υπάρχουν 2 κατηγορίες εισιτηρίων:



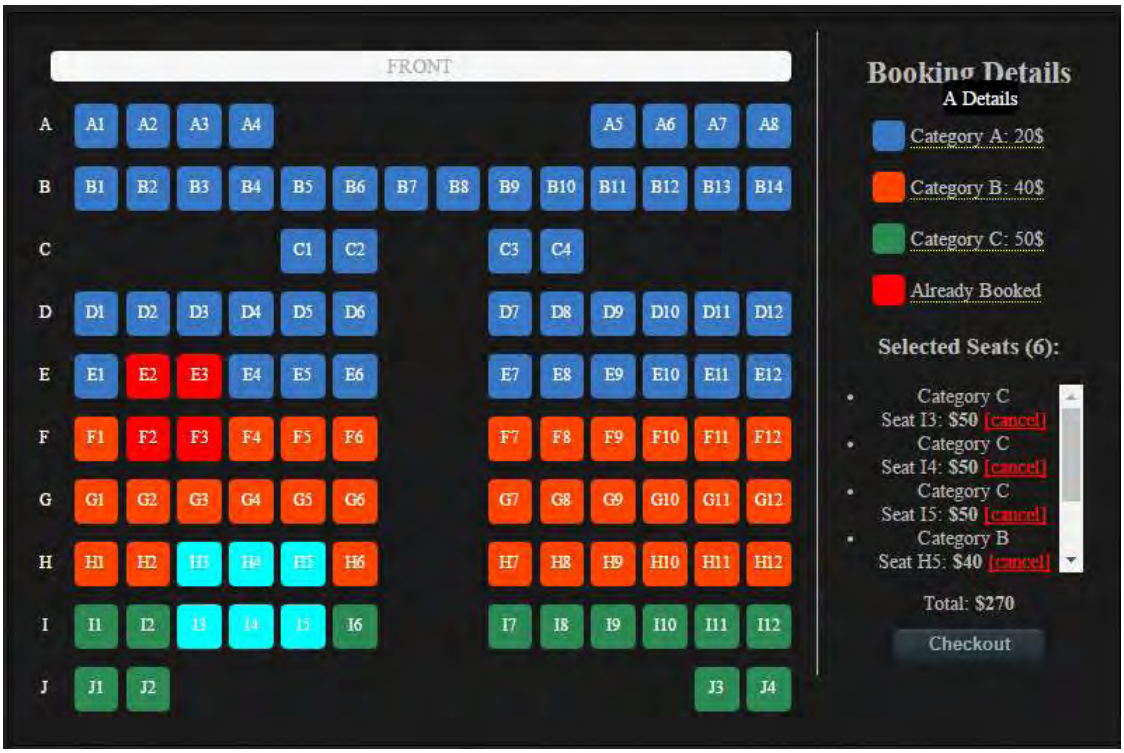
ΕΙΚΟΝΑ 2.4 – ΠΑΡΑΔΕΙΓΜΑ ΑΝΤΙΣΤΙΧΟΙΣΗΣ MAP ΣΕ FLOOR PLAN

Ο server αναλύει το αρχείο αυτό και δημιουργεί τις καινούριες θέσεις για τη συγκεκριμένη διοργάνωση. Έπειτα αποθηκεύεται ως String, επαυξάνοντας μετά τους λατινικούς χαρακτήρες το μοναδικό id της θέσης (ο server αναλύεται εκτενώς στο επόμενο κεφάλαιο) περικλεισμένο σε αγκύλες, διαχωρίζοντας την κάθε σειρά με κόμμα. Οπότε το παραπάνω map αποθηκεύεται ως: a[id1]a[id2]__a[id3]a[id4],b[id5]b[id6]a[id7]a[id8]b[id9]b[id10]
όπου idX είναι το μοναδικό id της κάθε θέσης που έγινε generated από το server.

Για τη μετατροπή του text map σε floor plan, χρησιμοποιήθηκε η JavaScript βιβλιοθήκη jQuery Seat Charts (JSC) (https://github.com/mateuszmarkowski/jQuery-Seat-Charts). Η βιβλιοθήκη αυτή δημιουργεί έναν προσβάσιμο χάρτη, το αντίστοιχο παράρτημα που επεξηγεί τις θέσεις, καθώς και listeners που ανταποκρίνονται στα κλικ του ποντικιού. Επίσης παρέχει και το αντίστοιχο style αρχείο CSS (Cascading Style Sheets), που καθορίζει την εμφάνιση του χάρτη. Με τη JSC επιλέγουμε το div (μπλοκ στοιχείο στην HTML) που θέλουμε να εμφανιστεί ο χάρτης και περνάμε τις κατάλληλες πληροφορίες. Η JSC δίνει ένα εύρος επιλογών για τον προγραμματιστή, όπως η ονομασία των σειρών και των στηλών, οι συναρτήσεις που θα τρέχουν όταν ο χρήστης κλικάρει μία θέση, τον τρόπο που θα αναγνωρίζεται η κάθε θέση, καθώς και το όνομα αυτής κ.ά.

Αρχικά καθώς οι πληροφορίες βρίσκονται αποθηκευμένες στο server, χρειαζόμαστε ένα τρόπο να τις φορτώσουμε στη JavaScript. Αυτό επιτυγχάνεται μέσω AJAX (Asynchronous JavaScript and XML), το οποίο μας επιτρέπει να στέλνουμε αιτήσεις στο server, χωρίς να ανανεώνουμε τη σελίδα. Με τη χρήση της συνάρτησης "**$.getJSON**" της jQuery, η οποία στέλνει ένα αίτημα AJAX και επιστραφεί ένα JSON αντικείμενο, φορτώνουμε το αντικείμενο αυτό. Το JSON περιέχει το string του floor map, τις πληροφορίες για τα εισιτήρια και τις κλεισμένες θέσεις, για τη συγκεκριμένη ημερομηνία, που επέλεξε ο χρήστης. Αφού φορτώσουμε τις απαραίτητες πληροφορίες στη JavaScript και τις μετατρέψουμε στους κατάλληλους, μπορούμε να τις χρησιμοποιήσουμε στη JSC. Η JSC αυτομάτως δημιουργεί τα κελιά του floor plan και αντιστοιχεί το κάθε ένα με το ανάλογο id, το οποίο βρίσκεται μέσα στις αγκύλες του String. Επειδή δεν ξέρουμε εκ των προτέρων πόσες κατηγορίες εισιτηρίων υπάρχουν, στο CSS υπάρχει για κάθε λατινικό χαρακτήρα και ένα αντίστοιχο χρώμα (σύνολο 16 χρωμάτων, όσο και οι

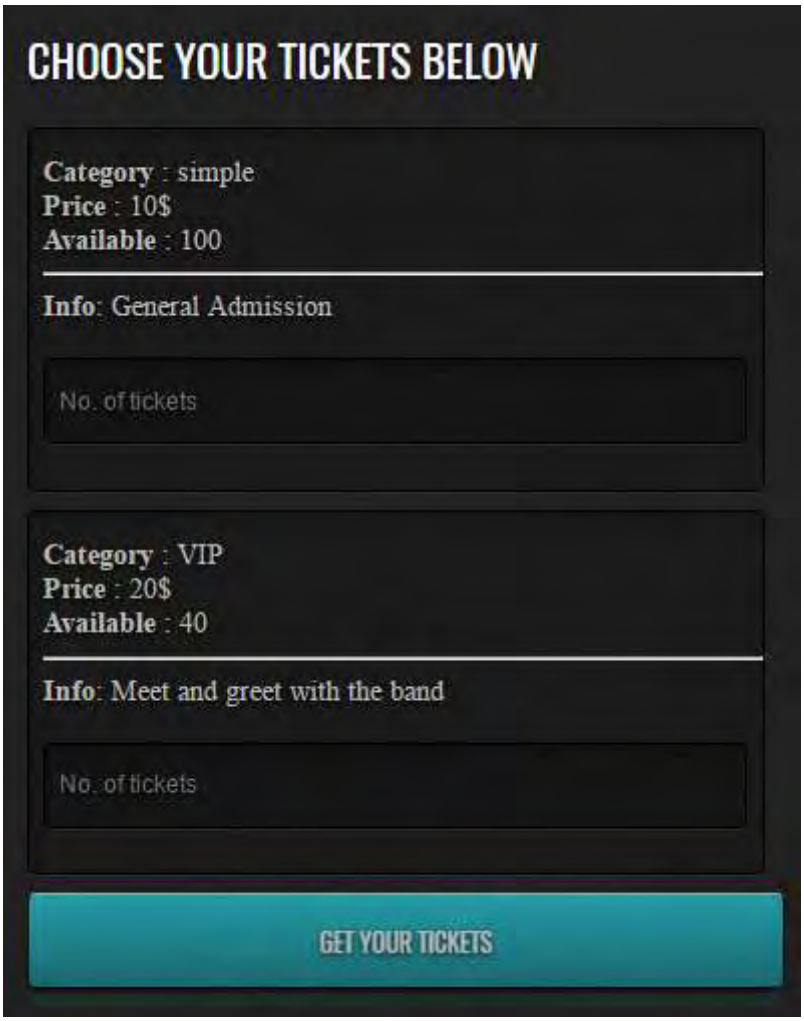
επιτρεπόμενες κατηγορίες εισιτηρίων). Επίσης ο αριθμός των σειρών μετατρέπεται μέσω συνάρτησης στο αντίστοιχο γράμμα (πχ 28 -> AB). Αφού περάσουμε το map με τα id, και τους τύπους των εισιτηρίων στη JSC αυτή δημιουργεί αυτόματα το floor plan και το αντίστοιχο legend με τις πληροφορίες.



ΕΙΚΟΝΑ 2.5 – ΕΠΙΛΟΓΗ ΘΕΣΕΩΝ ΜΕ ΔΙΑΘΕΣΙΜΟ FLOORPLAN

Στην παραπάνω εικόνα φαίνεται το floor plan και το legend (Booking Details) όπως δημιουργήθηκαν από τη JSC. Στο legend διευκρινίζεται το κάθε χρώμα σε ποια κατηγορία εισιτηρίου ανήκει, καθώς και αν κάνει hover ο χρήστης του εμφανίζονται οι διαθέσιμες πληροφορίες για την εκάστοτε κατηγορία (πχ A Details, με hover πάνω από την Category A). Επιπλέον υλοποιήθηκε ένα Cart με τα επιλεγμένα εισιτήρια. Με τη βοήθεια της JSC όταν ο χρήστης κλικάρει μία θέση, τότε μπορούμε να υλοποιήσουμε τις δικές μας συναρτήσεις που θα καλούνται (callback functions). Μόλις επιλεγεί μία θέση (γαλάζιο χρώμα) τότε επαυξάνουμε τη λίστα που βρίσκεται στο Cart, με τις αντίστοιχες πληροφορίες, οι οποίες βρίσκονται στη callback που μας προσφέρει η JCS και επίσης κρατάμε αυτή τη θέση σε έναν πίνακα. Αναλόγως όταν γίνει κλικ σε μία ήδη επιλεγμένη θέση, τότε αυτή αφαιρείται από το Cart και τον πίνακα με τις θέσεις και επιστρέφει στο αρχικό χρώμα της. Οι κόκκινες θέσεις σημαίνει ότι δεν είναι διαθέσιμες και δεν μπορούν να επιλεχθούν από το χρήστη. Όταν φορτώνεται αρχικά το floor plan, χρησιμοποιώντας τη JSC επιλέγουμε όλα τα id των κλεισμένων θέσεων που επιστράφηκαν με το JSON, και τα καθιστούμε μη διαθέσιμα. Τέλος με την επιλογή του Checkout στέλνονται τα επιλεγμένα id στο server και ο χρήστης μπορεί να προχωρήσει στο επόμενο βήμα, το οποίο αναλύεται παρακάτω στο κεφάλαιο αυτό.

Στην περίπτωση που ο διοργανωτής δεν έχει επιλέξει floor plan για τη διάθεση των εισιτηρίων του, τότε παρουσιάζονται στο χρήστη οι κατηγορίες των εισιτήριων, ο αριθμός των διαθέσιμων εισιτηρίων και πληροφορίες αυτών. Ο χρήστης επιλέγει των αριθμό που θέλει από κάθε κατηγορία και επιλέγοντας το κουμπί "GET YOUR TICKETS" ο server του επιστρέφει αυτόματα τα εισιτήρια που ζήτησε για κάθε κατηγορία. Η ανάλυση του επόμενου βήματος γίνεται παρακάτω στο κεφάλαιο.



ΕΙΚΟΝΑ 2.6 – ΕΠΙΛΟΓΗ ΕΙΣΙΤΗΡΙΩΝ ΧΩΡΙΣ FLOOR PLAN

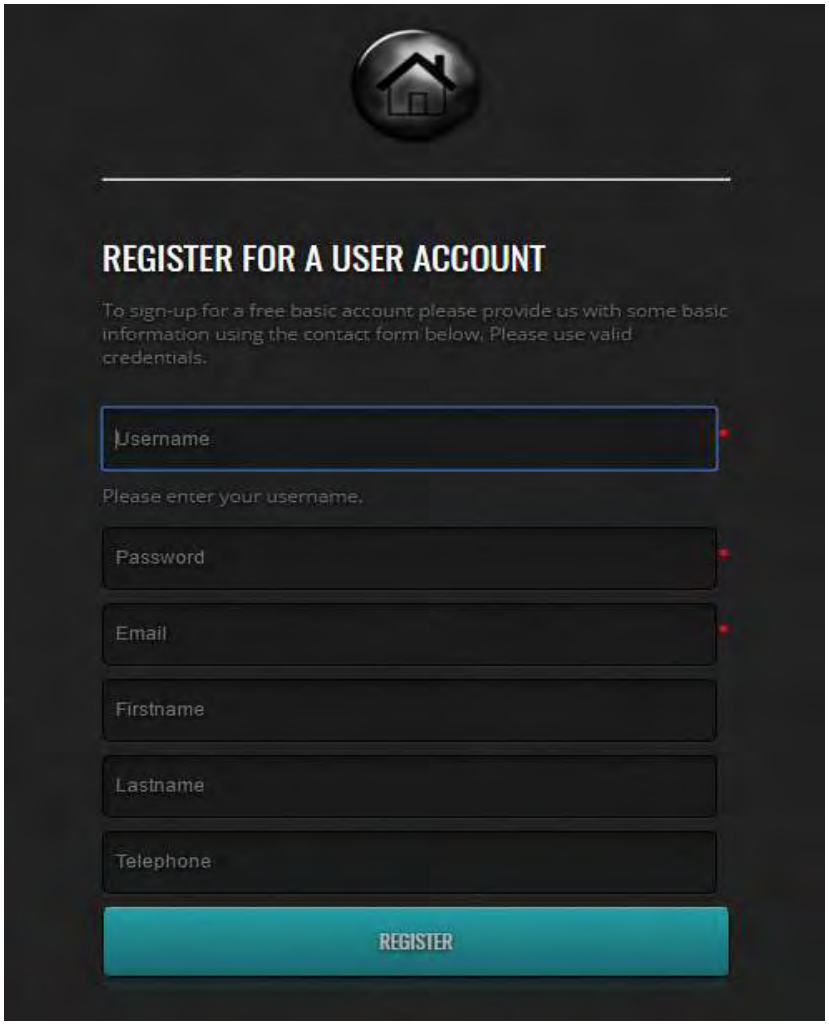
Όπως φαίνεται στην εικόνα, κάθε κουτί περιέχει και τις πληροφορίες για κάθε κατηγορία εισιτηρίου. Ο χρήστης μπορεί να βάλει μόνο αριθμητικές τιμές στην παραπάνω φόρμα. Σε περίπτωση που καταφέρει και περάσει οτιδήποτε άλλο εκτός από αριθμό, η αίτηση απορρίπτεται από τον server. Για να μπορέσει ο χρήστης να πραγματοποιήσει μία κράτηση πρέπει να είναι συνδεδεμένος ως απλός χρήστης στο σύστημα.

2.2.5 ΣΥΣΤΗΜΑ ΕΓΓΡΑΦΗΣ/ΕΙΣΟΔΟΥ

Από την αρχική σελίδα δίνεται στο χρήστη η επιλογή να εγγραφτεί ως διοργανωτής ή ως απλός χρήστης. Η φόρμα που εμφανίζεται για την εγγραφή είναι η ίδια και στις δύο περιπτώσεις. Τα στοιχεία που απαιτούνται για να ολοκληρωθεί η εγγραφή του χρήστη στο σύστημα είναι:

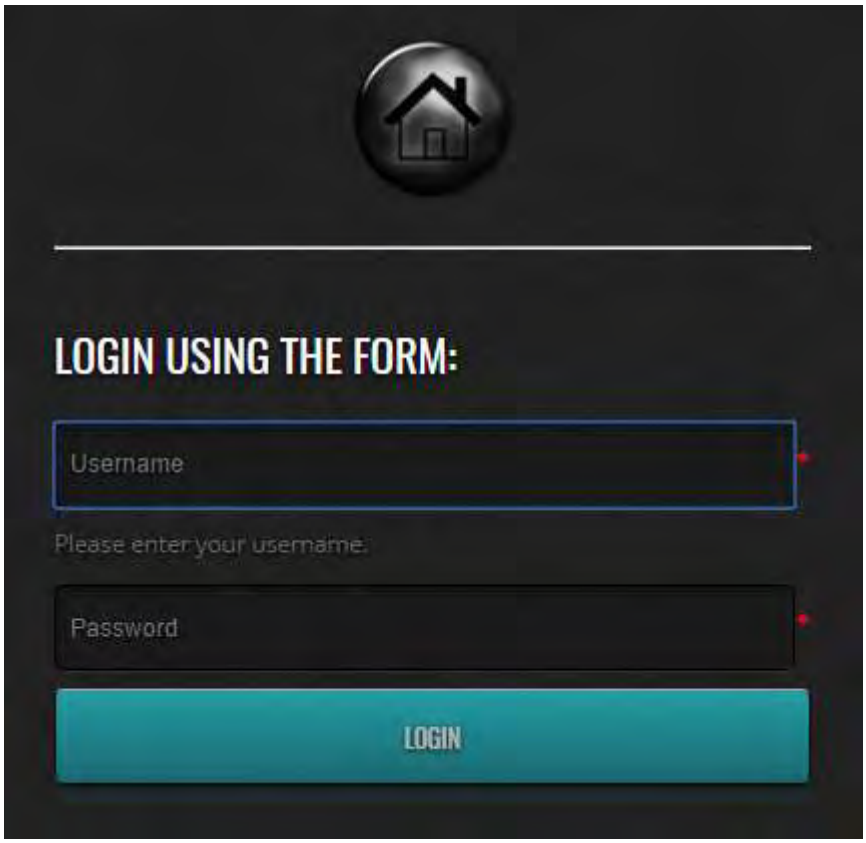
- **Username:** Είναι μοναδικό αναγνωριστικό του κάθε χρήστη.
- **Password:** Το συνθηματικό με το οποίο ζητείται κατά την είσοδο.
- **Email:** Το οποίο πρέπει να είναι μοναδικό.
- **Όνομα.**
- **Επώνυμο.**
- **Τηλέφωνο.**

Αφού ο χρήστης συμπληρώσει τη φόρμα, πατώντας το κουμπί Register ολοκληρώνεται η εγγραφή του, εισάγεται αυτομάτως στο σύστημα και ανακατευθύνεται στην αρχική σελίδα.



ΕΙΚΟΝΑ 2.7 – Η ΦΟΡΜΑ ΕΓΓΡΑΦΗΣ ΓΙΑ ΑΠΛΟΥΣ ΧΡΗΣΤΕΣ

Όταν γίνει με επιτυχία η εγγραφή, τότε ο χρήστης μπορεί μέσω της φόρμας εισόδου να μπει στο σύστημα. Τα στοιχεία που απαιτούνται είναι το username και το password. Αν πετύχει η είσοδος τότε γίνεται ανακατεύθυνση στη σελίδα των κρατήσεων/διοργανώσεων του, ανάλογα με την ιδιότητα και εμφανίζεται η μπάρα πλοήγησης που δίνει πρόσβαση στις αντίστοιχες λειτουργίες. Σε κάθε περίπτωση υπάρχει το Home Button πάνω από τη φόρμα, το οποίο οδηγεί στην αρχική σελίδα. Για την επίτευξη του drop down animation (πχ Please enter your username) χρησιμοποιήθηκε η βιβλιοθήκη JQuery της Javascipt. Η JQuery απλοποιεί τη χρήση της JavaScript, καθώς παρέχει ευέλικτους selectors, καθώς και έτοιμες συναρτήσεις, που γλυτώνουν τον προγραμματιστή από επιπλέον κώδικα JavaScript.



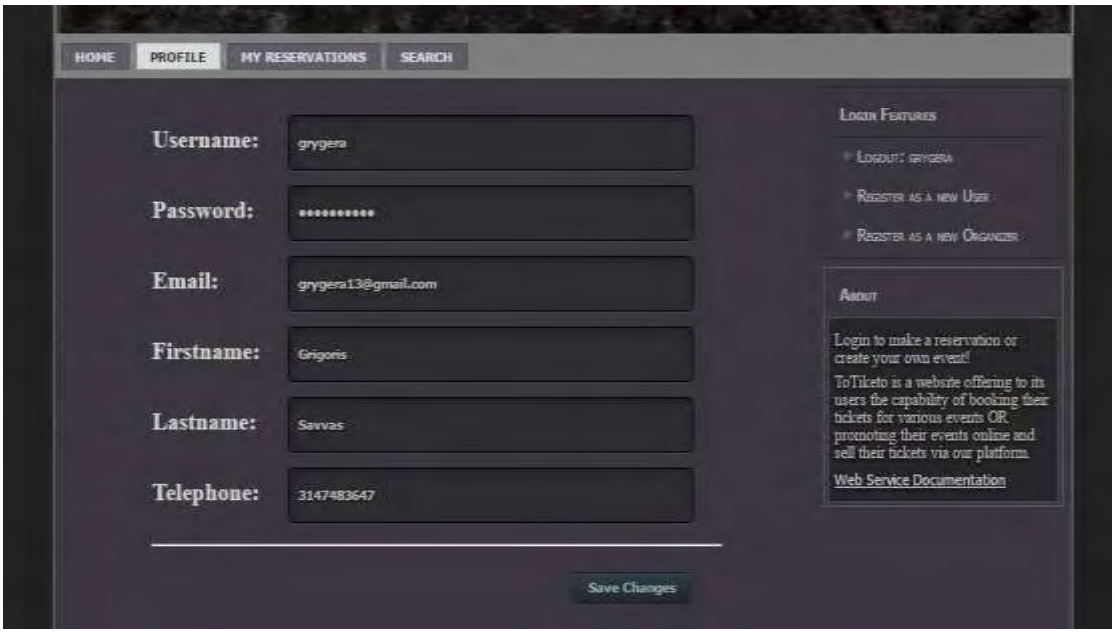
ΕΙΚΟΝΑ 2.8 – ΦΟΡΜΑ ΕΙΣΟΔΟΥ

Από τη μεριά του ο server αποθηκεύει το username και την ιδιότητα του χρήστη που κάνει εγγραφή/είσοδο σε SESSION μεταβλητές, ώστε να παρέχει τις ανάλογες λειτουργίες ανάλογα τον τύπο του χρήστη.

2.2.6 ΠΡΟΦΙΛ ΧΡΗΣΤΗ

Όπως προαναφέρθηκε, όταν ο χρήστης κάνει είσοδο στο σύστημα τότε εμφανίζεται η μπάρα πλοήγησης, με ανάλογες λειτουργίες με την ιδιότητα του χρήστη. Η μόνη λειτουργία, που είναι κοινή και για τα δύο είδη χρηστών – εκτός από την ανακατεύθυνση στην αρχική σελίδα – είναι η επεξεργασία του προφίλ.

Σε αυτή τη σελίδα παρουσιάζονται στο χρήστη τα στοιχεία που έδωσε κατά την εγγραφή του, καθώς και η δυνατότητα του να τα αλλάξει. Όλα τα στοιχεία, εκτός του username και του email, μπορούν να αλλαχτούν. εφόσον οι τιμές που δίνει ο χρήστης είναι μη μηδενικές. Επίσης παρατηρούμε ότι όταν ο χρήστης εισαχθεί στο σύστημα το κουμπί εισόδου αντικαταστείται πλέον με κουμπί αποσύνδεσης, το οποίο αποσυνδέει τον χρήστη και το ανακατευθύνει στην αρχική σελίδα.



ΕΙΚΟΝΑ 2.9 – ΣΕΛΙΔΑ ΕΠΕΞΕΡΓΑΣΙΑΣ ΠΡΟΦΙΛ

2.2.7 ΛΕΙΤΟΥΡΓΙΕΣ ΑΠΛΟΥ ΧΡΗΣΤΗ

Όταν ένας απλός χρήστης συνδεθεί στο σύστημα έχει τη δυνατότητα να διαχειριστεί τις κρατήσεις τους, καθώς και να κάνει καινούριες. Επίσης του δίνεται η δυνατότητα να αναζητήσει κάποια διοργάνωση που τον ενδιαφέρει. Με τη σύνδεση του στο σύστημα εμφανίζεται σε αυτόν η μπάρα πλοήγησης για απλούς χρήστες.

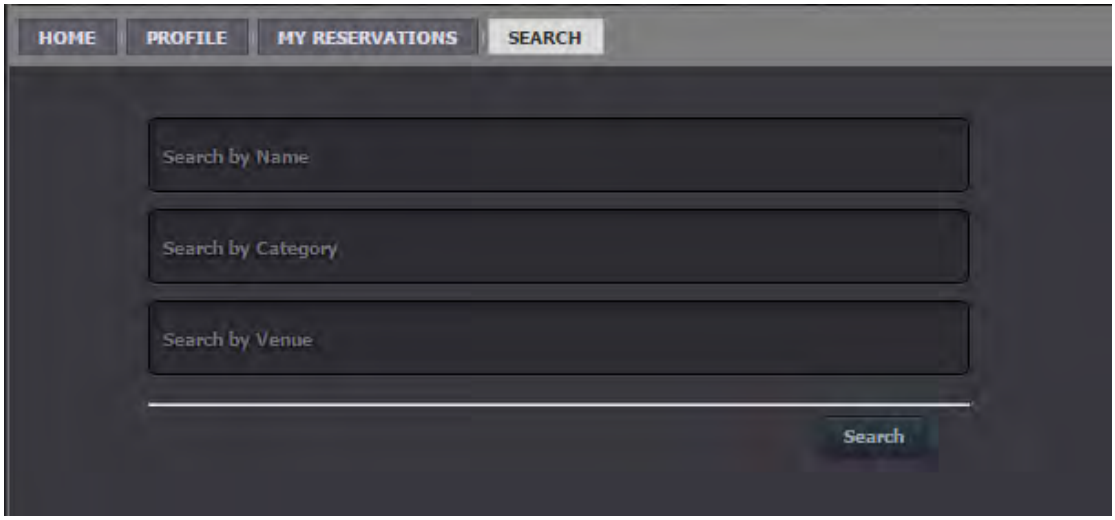
ΕΙΚΟΝΑ 2.10 – ΜΠΑΡΑ ΠΛΟΗΓΗΣΗ ΑΠΛΟΥ ΧΡΗΣΤΗ

Μέσω αυτής της μπάρας ο χρήστης μπορεί να κατευθυνθεί στην αρχική σελίδα, στο προφίλ του, στις κρατήσεις του ή στη σελίδα αναζήτησης.

Στη σελίδα αναζήτησης ο χρήστης έχει τη δυνατότητα μέσω της φόρμας που παρέχεται να αναζητήσει οποιαδήποτε διοργάνωση βάση 3 κριτηρίων:

1) **Όνομα**
2) **Κατηγορία**
3) **Χώρος Εκδήλωσης**

Ο χρήστης μπορεί να χρησιμοποιήσει οποιοδήποτε συνδυασμό των ανωτέρων κριτηρίων επιθυμεί. Αν η αναζήτηση είναι επιτυχής τότε παρουσιάζονται στο χρήστη τα αποτελέσματα αυτής στο ίδιο στυλ που παρουσιάζονται και στην αρχική σελίδα. Αυτό σημαίνει ότι ο χρήστης μπορεί να επιλέξει την ημερομηνία που τον ενδιαφέρει και κατόπιν να προβεί στην κράτηση. Επίσης εμφανίζονται δέκα αποτελέσματα τη φορά και ο χρήστης μπορεί να πλοηγηθεί σε αυτά μέσω των κουμπιών next και previous που αναλύθηκαν στην παράγραφο 2.2.2. Σε περίπτωση μη εύρεσης αποτελεσμάτων εμφανίζεται ανάλογο μήνυμα στο χρήστη.



ΕΙΚΟΝΑ 2.11 – Η ΦΟΡΜΑ ΑΝΑΖΗΤΗΣΗΣ

Στη σελίδα των κρατήσεων, ο χρήστης μπορεί να δει τις κρατήσεις που έχει κάνει. Η σελίδα αποτελείται από έναν πίνακα οποίος σε κάθε γραμμή του έχει πληροφορίες για κάθε κράτηση, καθώς και την επιλογή ακύρωσης αυτής. Τα στοιχεία που εμφανίζονται στο χρήστη είναι:

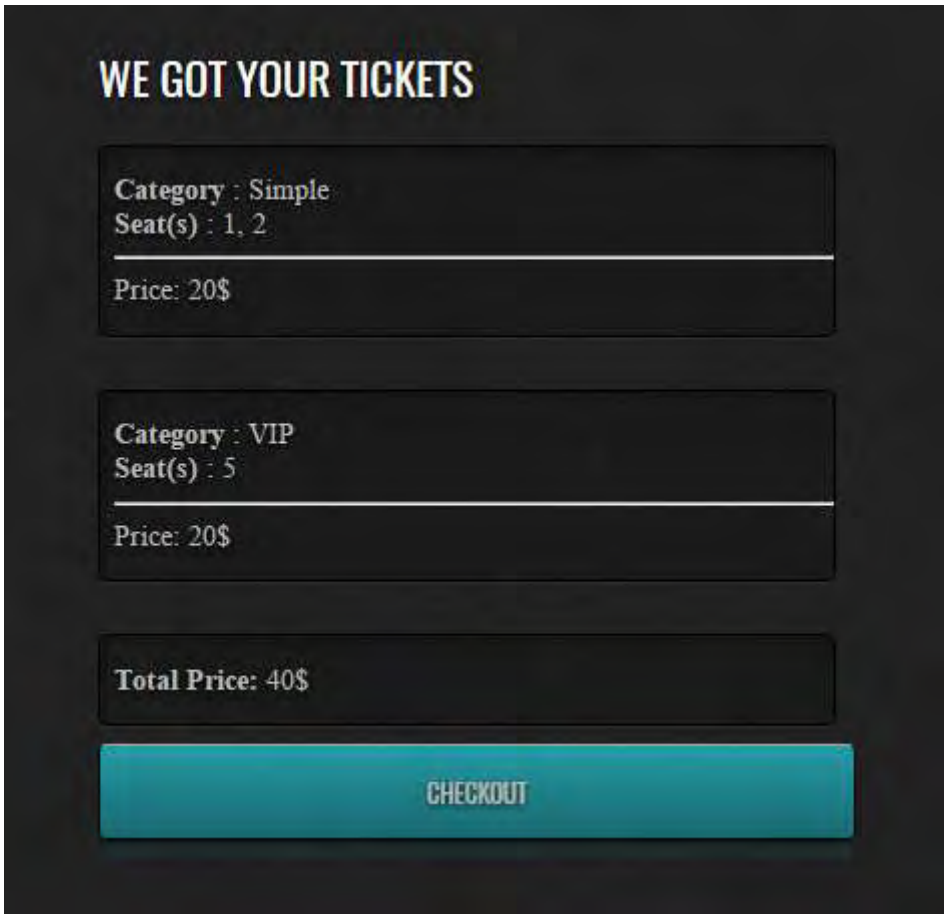
- **ReservationID:** Το αναγνωριστικό της κράτησης. Το id της κάθε κράτησης είναι μοναδικό.
- **Event:** Το όνομα της εκδήλωσης για την οποία έγινε η κράτηση.
- **Date:** Η ημερομηνία και η ώρα της εκδήλωσης.
- **Seats:** Τα αναγνωριστικά των κλεισμένων θέσεων που αντιστοιχούν στην κράτηση αυτή.
- **Status:** Το πεδίο αυτό δηλώνει την κατάσταση της κράτησης. Αν η κράτηση δεν έχει οριστικοποιηθεί τότε εμφανίζεται η ένδειξη “Open”, η οποία αποτελεί και σύνδεσμο, μέσω τον οποίο ο χρήστης μπορεί να οριστικοποιήσει την κράτηση. Αν η κράτηση είναι οριστικοποιημένη τότε το πεδίο έχει την τιμή “Closed”.
- **Delete:** Το τελευταίο πεδίο του πίνακα περιέχει ένα κουμπί-εικόνα με το οποίο ο χρήστης μπορεί να ακυρώσει την κράτηση αυτή.

ΕΙΚΟΝΑ 2.12 – Ο ΠΙΝΑΚΑΣ ΜΕ ΤΙΣ ΚΡΑΤΗΣΕΙΣ ΤΟΥ ΧΡΗΣΤΗ

Τα κουμπιά για τη διαγραφή περικλείονται σε μία φόρμα HTML (form). Όταν ο χρήστης επιλέξει τη διαγραφή, εμφανίζεται μία ειδοποίηση επιβεβαίωσης. Κατόπιν αποστέλλεται το id στο server και προχωράει με τη διαγραφή.

Όπως αναφέρθηκε παραπάνω στο κεφάλαιο, πρέπει να έχει γίνει σύνδεση απλού χρήστη για να προχωρήσει μία κράτηση. Αν ένας διοργανωτής επιχειρήσει να κάνει κράτηση τότε ανακατευθύνεται στη σελίδα εισόδου. Όταν ένας χρήστης επιλέξει τις θέσεις ή τον αριθμό των εισιτηρίων που θέλει με οποιοδήποτε τρόπο, floor plan ή χωρίς, τότε μέσω της HTML φόρμας τα δεδομένα στέλνονται στο server ο οποίος πραγματοποιεί προσωρινή κράτηση αυτών. Έπειτα παρουσιάζονται στο χρήστη τα εισιτήρια που επιλέχθηκαν από το server ανά κατηγορία, η συνολική τιμή κάθε κατηγορίας και τέλος το ολικό κόστος της κράτησης. Σε περίπτωση floor plan , τα εισιτήρια που επιλέγει ο server γίνονται με βάση τα id που αποστάλθηκαν, ενώ αν δεν υπάρχει floor plan τότε επιστρέφονται τα πρώτα διαθέσιμα εισιτήρια που θα βρει ο server στην εκάστοτε κατηγορία. Περαιτέρω ανάλυση για το πώς πραγματοποιούνται οι κρατήσεις από το server γίνεται στο $3^{ο}$ κεφάλαιο.

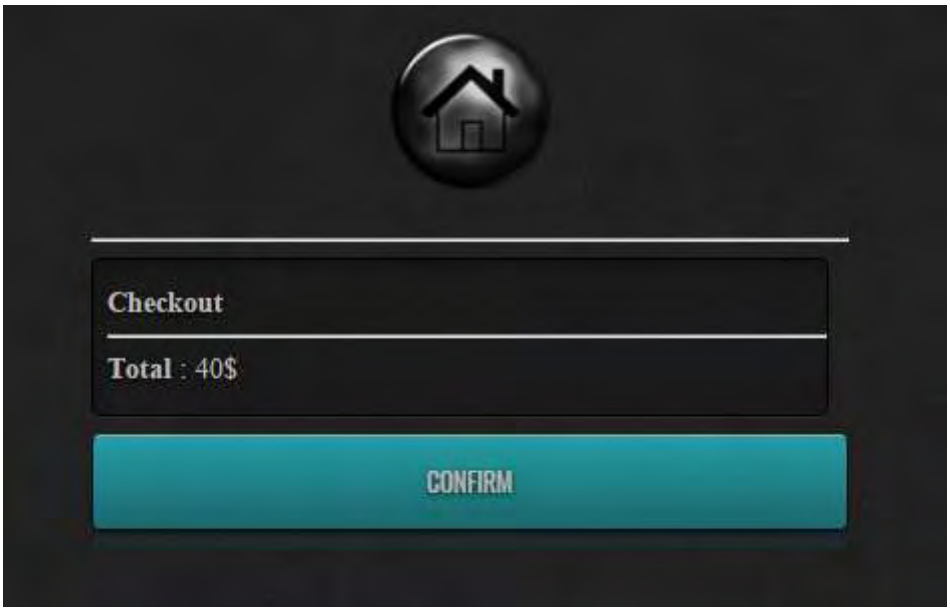
Τα εισιτήρια που επιλέχθηκαν κατοχυρώνονται προσωρινά στο χρήστη για δέκα λεπτά και η κράτηση θεωρείται «ανοιχτή» (open). Αφού ο χρήστης επιβεβαιώσει τις τιμές και τα εισιτήρια μπορεί να προχωρήσει στο επόμενο βήμα ώστε να οριστικοποιήσει την κράτηση. Σε περίπτωση που αποχωρήσει από τη σελίδα, μπορεί ακόμα να οριστικοποιήσει την κράτηση αυτήν από τον πίνακα των κρατήσεων του κάνοντας κλικ στο σύνδεσμο «open» που βρίσκεται στην αντίστοιχη κράτηση. Αυτό πρέπει να γίνει εντός δέκα λεπτών, καθώς μετά η κράτηση δεν θεωρείται έγκυρη και τα εισιτήρια αποδεσμεύονται. Με την επιλογή του checkout τα στοιχεία της κράτησης στέλνονται στο server και εμφανίζεται η σελίδα επιβεβαίωσης.



ΕΙΚΟΝΑ 2.13 – ΠΑΡΟΥΣΙΑΣΗ ΕΠΙΛΕΓΜΕΝΩΝ ΕΙΣΙΤΗΡΙΩΝ

Καθώς η επεξεργασία της κράτηση πραγματοποιείται από το server μέσω PHP στην ίδια σελίδα που παρουσιάζονται, δημιουργείται το πρόβλημα ότι με την ανανέωση της σελίδας τα εισιτήρια θα αποσταλούν ξανά και θα γίνει ξανά η κράτηση. Για την αντιμετώπιση αυτού του προβλήματος, όταν ο χρήστης επιλέξει τα εισιτήρια στέλνεται μαζί με αυτά, μέσω της HTML φόρμας, ένας random αριθμός ο οποίος αποθηκεύεται και ως SESSION μεταβλητή στο server. Έπειτα στο επόμενο βήμα, της παρουσίασης των εισιτηρίων, ελέγχεται αν η τιμή που περάστηκε από τη φόρμα είναι η ίδια με τη SESSION, και αν υπάρχει ταύτιση τότε προχωράει στην παρουσίαση των εισιτηρίων και διαγράφεται η SESSION μεταβλητή. Όταν ο χρήστης κάνει ανανέωση η ταύτιση αυτή δεν θα ισχύει, καθώς η SESSION έχει διαγραφτεί, με αποτέλεσμα να μην προχωράει εκ νέου σε κράτηση αλλά να ανακατευθύνει στην αρχική σελίδα.

Στο τελευταίο βήμα, ώστε να ολοκληρωθεί η κράτηση, παρουσιάζεται απλά η τιμή της κράτησης και ζητείται από τον χρήστη να την επιβεβαιώσει. Αφού ο χρήστης πατήσει το κουμπί CONFIRM, τότε ο server ελέγχει αν η κράτηση είναι έγκυρη και την οριστικοποιεί. Τα εισιτήρια πλέον δεσμεύονται στο χρήστη και η κράτηση εμφανίζεται ως «closed» στον πίνακα κρατήσεων.



ΕΙΚΟΝΑ 2.14 – ΕΠΙΒΕΒΑΙΩΣΗ ΚΡΑΤΗΣΗΣ

2.2.8 ΛΕΙΤΟΥΡΓΙΕΣ ΔΙΟΡΓΑΝΩΤΗ

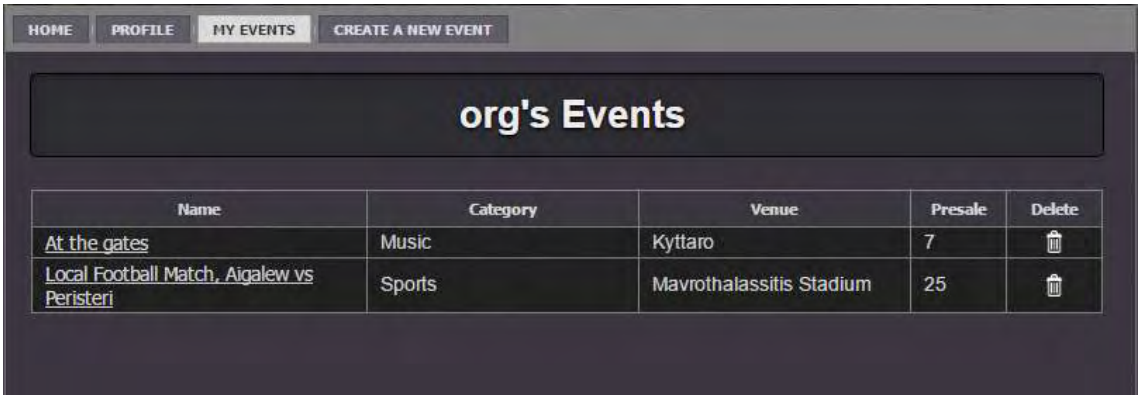
Όταν ένας διοργανωτής συνδεθεί στο σύστημα έχει τη δυνατότητα να δημιουργήσει νέες διοργανώσεις. Επιπλέον μπορεί να επεξεργαστεί τα στοιχεία μίας δημοσιευμένης διοργάνωσης καθώς και να δει πληροφορίες σχετικά με την προπώληση αυτής. Με τη σύνδεση του στο σύστημα εμφανίζεται σε αυτόν η μπάρα πλοήγησης για διοργανωτές.

ΕΙΚΟΝΑ 2.15 – ΜΠΑΡΑ ΠΛΟΗΓΗΣΗΣ ΔΙΟΡΓΑΝΩΤΗ

Μέσω αυτής της μπάρας ο χρήστης μπορεί να κατευθυνθεί στην αρχική σελίδα, στο προφίλ του, στις διοργανώσεις του ή στη σελίδα δημιουργίας νέας.

Η σελίδα των διοργανώσεων είναι το ίδιο αρχείο με τη σελίδα των κρατήσεων. Ανάλογα όμως την ιδιότητα του χρήστη εμφανίζονται και άλλες πληροφορίες στον πίνακα. Ο server ελέγχει τη SESSION μεταβλητή της ιδιότητας που αποθηκεύτηκε κατά την είσοδο του χρήστη και ανάλογα με την τιμή της (user ή organizer) γεμίζει τον πίνακα. Τα στοιχεία του πίνακα για κάθε διοργάνωση είναι τα εξής:

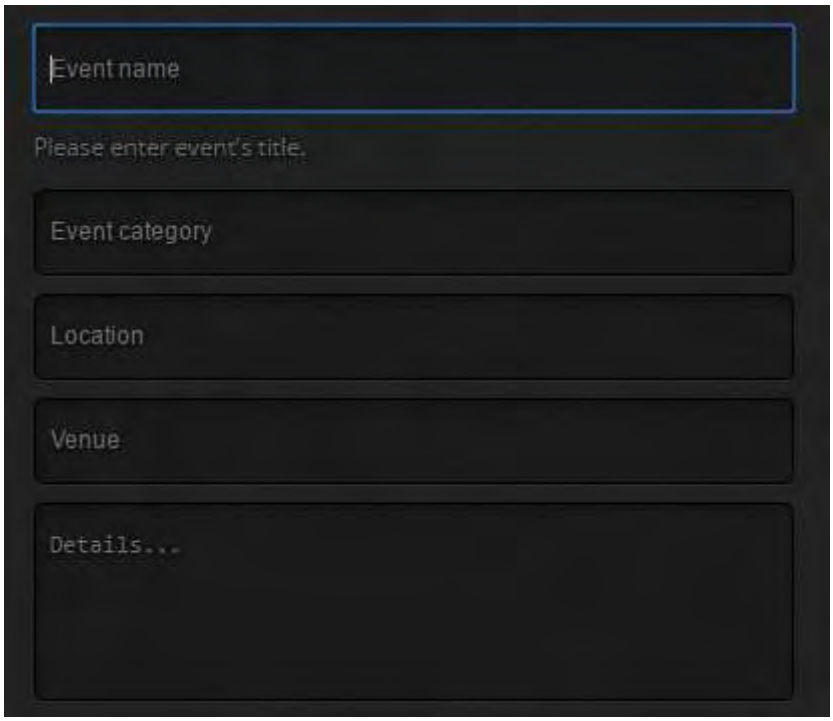
- **Name:** Το όνομα της διοργάνωσης.
- **Category:** Το είδος της εκδήλωσης που προσφέρεται.
- **Venue:** Το μέρος όπου λαμβάνει χώρα.
- **Presale:** Ο συνολικός αριθμός των προ-πωλημένων εισιτηρίων.
- **Delete:** Περιέχει το κουμπί-εικόνα για τη διαγραφή της διοργάνωσης.



ΕΙΚΟΝΑ 2.16 - Ο ΠΙΝΑΚΑΣ ΜΕ ΤΙΣ ΔΙΟΡΓΑΝΩΣΕΙΣ ΤΟΥ ΧΡΗΣΤΗ

Ο χρήστης μπορεί να κάνει κλικ στο εικονίδιο delete για να διαγράψει την αντίστοιχη διοργάνωση, μετά από σχετική επιβεβαίωση. Αν υπάρχει έστω και μία έγκυρη κράτηση για τη συγκεκριμένη διοργάνωση, τότε η διαγραφή δεν είναι δυνατή. Τέλος μπορεί μέσω του link, που βρίσκεται στο όνομα της εκάστοτε διοργάνωσης, να μεταβεί στη σελίδα επεξεργασίας αυτής.

Η σελίδα δημιουργίας νέας διοργάνωσης αποτελείται από μία μακροσκελή φόρμα, η οποία αναλύεται κομμάτι-κομμάτι και περιέχει όλες τις απαραίτητες πληροφορίες που χρειάζονται για τη δημιουργία μίας νέας διοργάνωσης. Όλα τα στοιχεία της φόρμας είναι απαραίτητα.

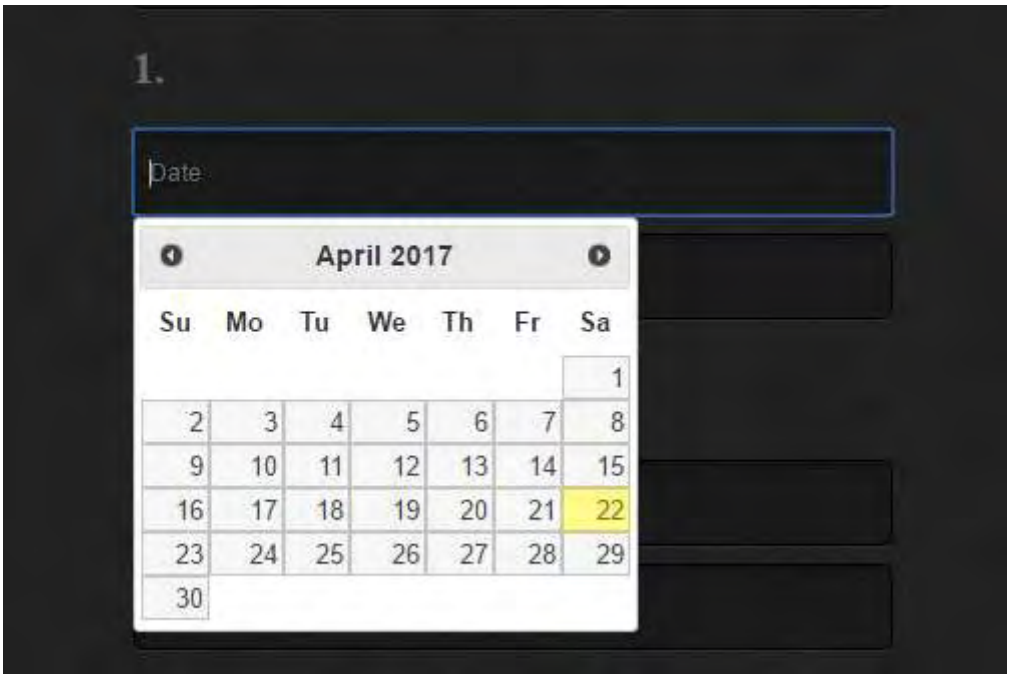


ΕΙΚΟΝΑ 2.17 – ΦΟΡΜΑ 1/5 ΒΑΣΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ

Τα πρώτα πεδία αναφέρονται στις βασικές πληροφορίες. Αυτές είναι το όνομα, η κατηγορία, η διεύθυνση, ο χώρος και μία εκτενής περιγραφή της διοργάνωσης. Όλα τα πεδία της φόρμας περιέχουν ένα στοιχείο Placeholder, ώστε να υποδεικνύει στο χρήστη τι πρέπει να συμπληρώσει, καθώς και ένα drop down animation με τον ίδιο σκοπό.

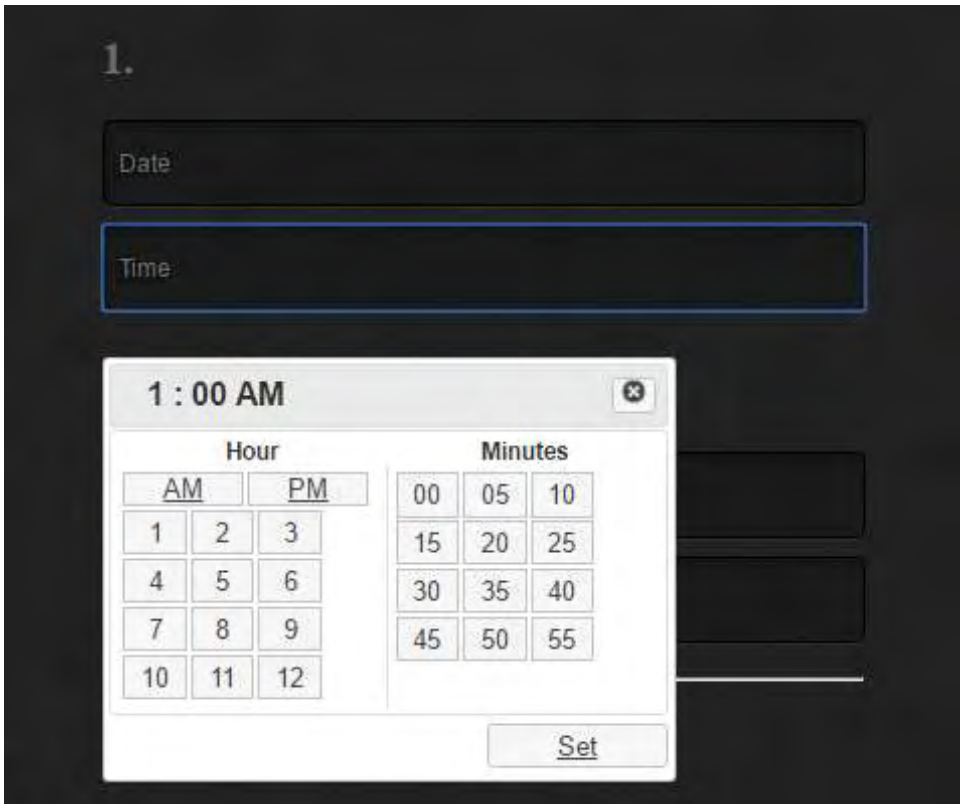
ΕΙΚΟΝΑ 2.18 – ΦΟΡΜΑ 2/5 ΕΠΙΛΟΓΗ ΗΜΕΡΟΜΗΝΙΩΝ

Ο χρήστης μπορεί να επιλέξει όσες ημερομηνίες θέλει για τη διοργάνωση του. Για κάθε επιλογή του συμπληρώνει την ημερομηνία στο πεδίο Date και την ώρα έναρξης στο πεδίο Time. Με το κουμπί Add New Date, προστίθεται ένα νέο πεδίο ημερομηνίας/ώρας στη φόρμα με αύξων αριθμό. Το κουμπί cancel αφαιρεί το τελευταίο πεδίο στη φόρμα. Όταν είναι μία ημερομηνία τότε το κουμπί Cancel εξαφανίζεται. Για το σωστό τύπο της ημερομηνίας και ώρας χρησιμοποιούμε 2 έτοιμα εργαλεία για να καθοδηγήσουμε το χρήστη. Το πρώτο είναι το DatePicker, που βρίσκεται έτοιμο στην βιβλιοθήκη jQuery-UI, που παρέχει έτοιμα widgets για τον προγραμματιστή, γραμμένα σε jQuery. Το DatePicker δίνει τη δυνατότητα στο χρήστη να εισάγει μία ημερομηνία από ένα έτοιμο ημερολόγιο.



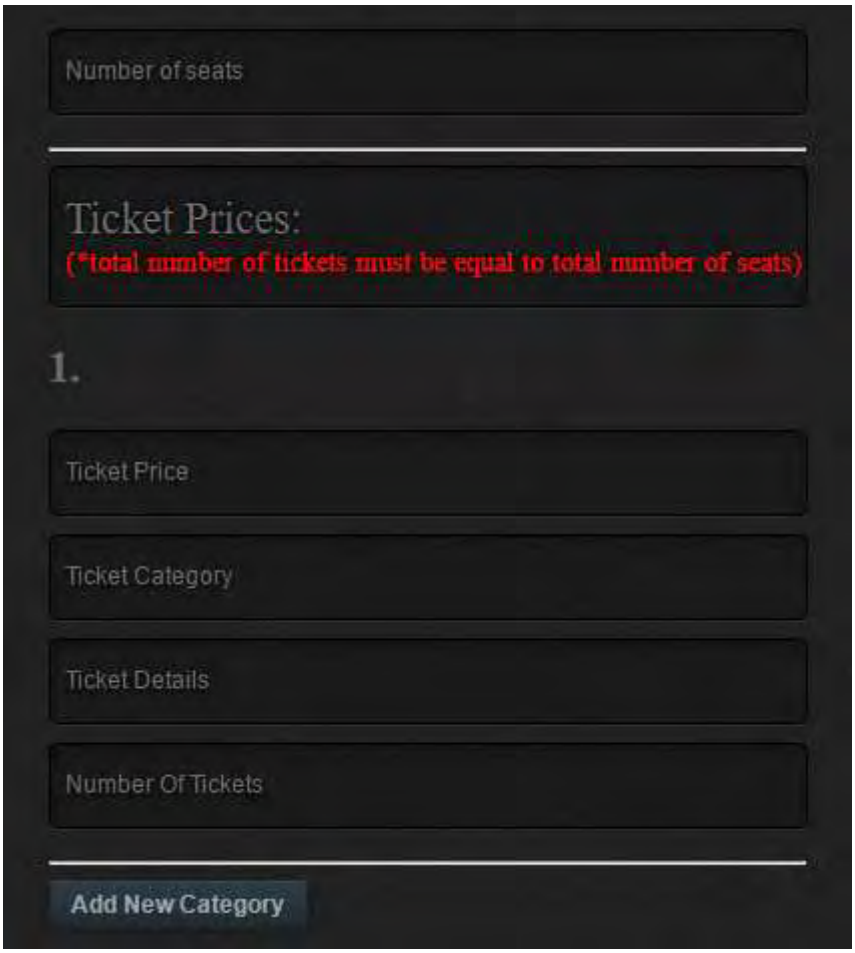
ΕΙΚΟΝΑ 2.19 – ΤΟ ΕΡΓΑΛΕΙΟ DATEPICKER

Το δεύτερο εργαλείο είναι το ptTimeSelect το οποίο αντίστοιχα εμφανίζει ένα pop παράθυρο, από το οποίο ο χρήστης μπορεί να επιλέξει την ώρα έναρξης. Με τα δύο αυτά εργαλεία βεβαιώνουμε το σωστό τύπο δεδομένων που θα περαστούν στο server.



ΕΙΚΟΝΑ 2.20 – ΤΟ ΕΡΓΑΛΕΙΟ ptTimeSelect

Έπειτα ζητείται από το χρήστη να εισάγει τον αριθμό των εισιτηρίων που προσφέρει. Όλες οι ημερομηνίες υπόκεινται στον ίδιο αριθμό και τύπο εισιτηρίων. Αφού συμπληρωθεί ο συνολικός αριθμός, τότε ο χρήστης εισάγει πληροφορίες για κάθε κατηγορία που είναι διαθέσιμη και τον αριθμό των εισιτηρίων που αντιστοιχεί στην κατηγορία αυτή.



ΕΙΚΟΝΑ 2.21 – ΦΟΡΜΑ 3/5 ΠΛΗΡΟΦΟΡΙΕΣ ΕΙΣΙΤΗΡΙΩΝ

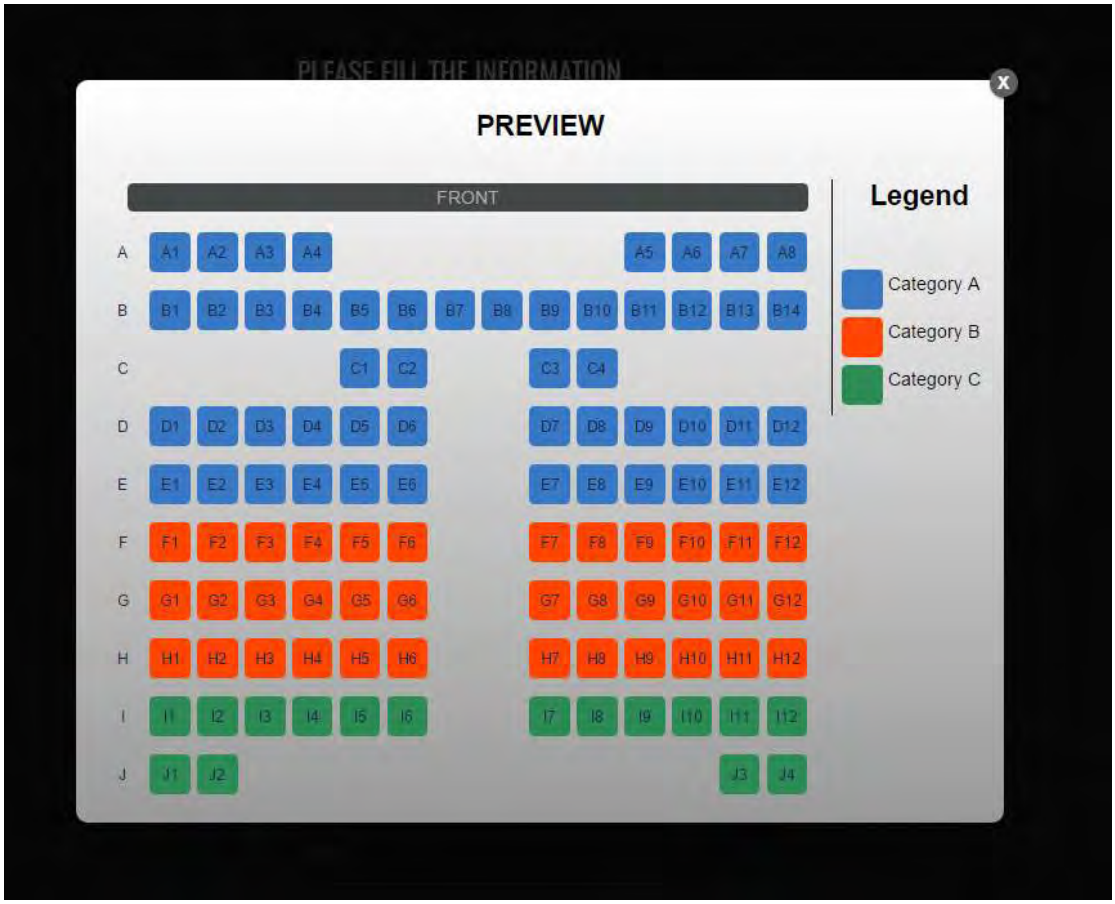
Για κάθε κατηγορία ο χρήστης πρέπει να συμπληρώσει την τιμή, το όνομα της κατηγορίας, μία περιγραφή της κατηγορίας και τον αριθμό των εισιτηρίων που αντιστοιχούν σε αυτήν. Όπως με τις ημερομηνίες, πατώντας το κουμπί Add New Category εμφανίζεται ένα νέο κουτί με αύξων αριθμό, καθώς και το αντίστοιχο κουμπί Cancel, που ακυρώνει το τελευταίο πεδίο που προστέθηκε. Επίσης διευκρινίζεται ότι το άθροισμα των εισιτηρίων της κάθε κατηγορίας πρέπει να ανταποκρίνεται στο συνολικό αριθμό που συμπληρώθηκε νωρίτερα.

Το επόμενο βήμα είναι η χρήση ή όχι floor plan. Αυτό γίνεται με τη χρήση ενός Radio Button, με προεπιλεγμένη την επιλογή «No». Με την επιλογή του «Yes» γίνεται drop down ένας οδηγός για τη χρήση του floor plan, καθώς και το κουμπί για το ανέβασμα του αρχείου κειμένου.

ΕΙΚΟΝΑ 2.22 – ΟΔΗΓΟΣ ΓΙΑ ΕΠΙΛΟΓΗ FLOOR PLAN

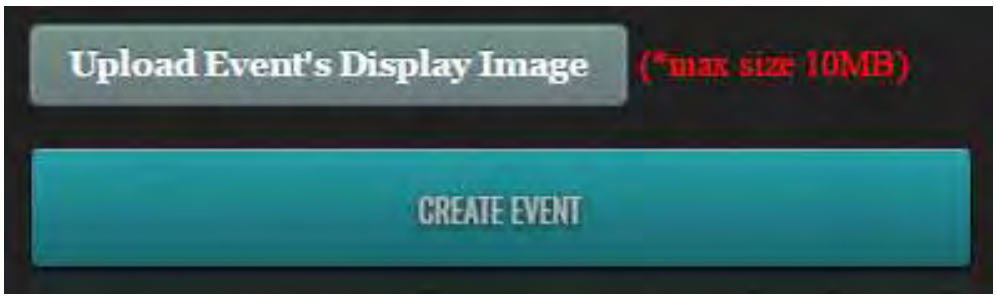
Ο οδηγός αναφέρει ένα παράδειγμα σωστά δομημένου floor plan, καθώς και απαραίτητες πληροφορίες για τη σωστή λειτουργία του. Κάθε κατηγορία εισιτηρίου που συμπλήρωσε ο χρήστης αντιστοιχεί και σε ένα αύξων γράμμα στο αρχείο (αρχίζοντας από το 'a'). Με την επιλογή του αρχείου δίνεται στο χρήστη η δυνατότητα να τεστάρει το floor plan πριν το ανεβάσει, Μόλις ανέβει το αρχείο κειμένου εμφανίζεται δίπλα από το Upload, το κουμπί Preview, με το οποίο μέσω CSS δημιουργούμε ένα modal παράθυρο (εμφανίζεται πάνω από τη σελίδα)με το preview.

Για το σχεδιασμό του floor plan χρησιμοποιούμε τη βιβλιοθήκη JSC. Η διαφορά είναι ότι οι πληροφορίες για τα εισιτήρια και τις θέσεις δεν βρίσκονται πλέον στο server, αλλά στο ανεβασμένο αρχείο. Για την προσπέλαση του αρχείου χρησιμοποιούμε την κλάση FileReader της JavaScript. Αυτή μας δίνει τη δυνατότητα να διαβάζουμε ασύγχρονα ένα αρχείο. Επίσης μας παρέχει μία callback συνάρτηση που τρέχει όταν ολοκληρωθεί η φόρτωση. Μέσω αυτής διαβάζουμε το αρχείο σειρά-σειρά και επεξεργαζόμαστε τα δεδομένα ώστε να τα φορτώσουμε στη JSC. Ουσιαστικά μετατρέπουμε το αρχείο σε έναν πίνακα, τον οποίο μετατρέπει η JSC στο floor plan αυτόματα. Επίσης αντιστοιχίζουμε κάθε πολιτική που έχει εισάγει ο χρήστης προηγουμένως με ένα αύξων γράμμα που βρίσκεται στο αρχείου κειμένου. Η JSC δέχεται σαν όρισμα για τις θέσεις έναν πίνακα αντικειμένων που κάθε στοιχείου του περιέχει την τιμή του εισιτηρίου, την κλάση του CSS που αντιστοιχεί (στην περίπτωση μας έχουμε έτοιμα 16 χρώματα - κλάσεις για κάθε αύξων γράμμα) και το όνομα της κατηγορίας. Επειδή παλιότερες εκδόσεις μερικών φυλλομετρητών δεν υποστηρίζουν το FileReader, ελέγχουμε μόλις κλικάρει ο χρήστης το κουμπί Preview και αναλόγως του εμφανίζεται το floor plan ή μήνυμα ότι δεν υποστηρίζεται η λειτουργία αυτή από το φυλλομετρητή του.



ΕΙΚΟΝΑ 2.23 – ΤΟ MODAL ΠΑΡΑΘΥΡΟ

Αφού ο χρήστης επιβεβαιώσει τη σωστή αναπαράσταση του floor plan κλείνει το παράθυρο και προχωράει στο τελευταίο μέρος της φόρμας, που είναι η επιλογή εικόνας. Υποδεικνύεται στο χρήστη ότι το μέγιστο επιτρεπτό μέγεθος είναι τα 10MB.



ΕΙΚΟΝΑ 2.24 – ΑΝΕΒΑΣΜΑ ΕΙΚΟΝΑΣ ΚΑΙ ΥΠΟΒΟΛΗ

Μετά το ανέβασμα της εικόνας ο χρήστης με το κουμπί CREATE EVENT, ολοκληρώνει τη δημιουργία της διοργάνωσης και αυτή δημοσιεύεται στην αρχική σελίδα. Σε περίπτωση που υπάρχει κάποιο λάθος στα δεδομένα του χρήστη, τότε εμφανίζεται το ανάλογο μήνυμα και επαναφορτώνεται η σελίδα.

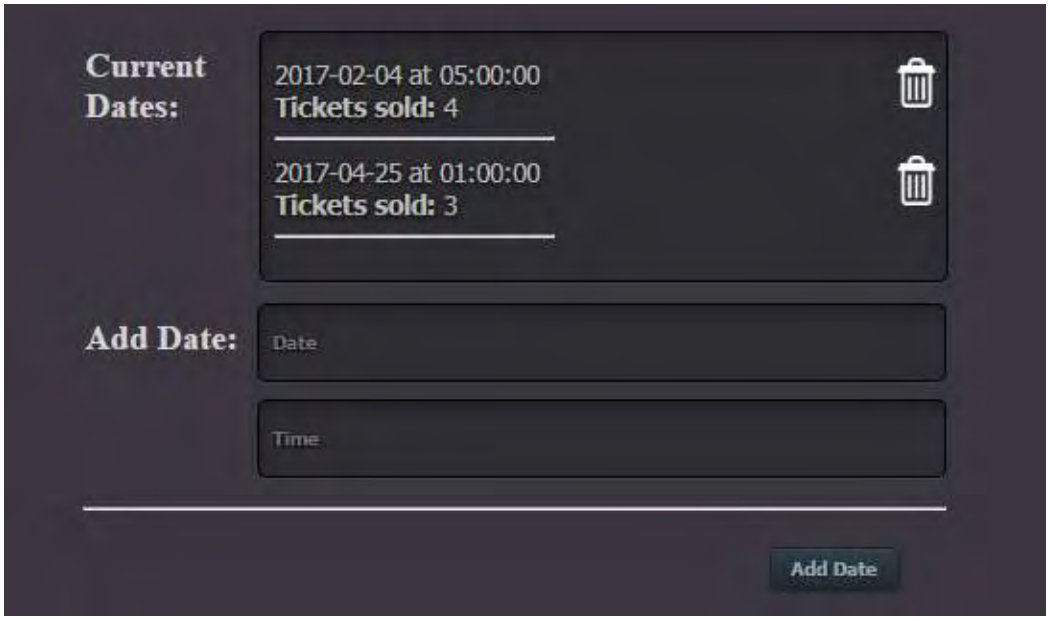
Στη σελίδα επεξεργασίας, ο χρήστης έχει τη δυνατότητα να επεξεργαστεί ορισμένες πληροφορίες που έδωσε κατά τη δημιουργία της διοργάνωσης, καθώς και να πάρει πληροφορίες σχετικά με την προπώληση. Η σελίδα μπορεί να χωριστεί σε δύο μέρη. Το πρώτο είναι η γενικές πληροφορίες που αφορούν τη διοργάνωση. Αποτελείται από μία φόρμα HTML που περιέχει την εικόνα της διοργάνωσης, το όνομα, την κατηγορία, τη διεύθυνση, το χώρο και την περιγραφή.

ΕΙΚΟΝΑ 2.25 – ΕΠΕΞΕΡΓΑΣΙΑ ΠΛΗΡΟΦΟΡΙΩΝ ΔΙΟΡΓΑΝΩΣΗΣ

Με κλικ στην εικόνα, ο χρήστης μπορεί να ανεβάσει μία καινούρια από τον υπολογιστή του και να αντικαταστήσει την τωρινή. Επίσης μπορεί να αλλάξει οποιαδήποτε από τις παραπάνω πληροφορίες. Με το κουμπί Save Changes, οι αλλαγές στέλνονται στο server και αποθηκεύονται.

Το δεύτερο κομμάτι περιλαμβάνει τις διαθέσιμες ημερομηνίες που έχει δηλώσει ο χρήστης. Κάτω από κάθε μία υπάρχει ο αριθμός των εισιτηρίων που έχουν πωληθεί μέχρι τώρα. Με την επιλογή της διαγραφής ο χρήστης μπορεί να διαγράψει τη συγκεκριμένη ημερομηνία, εφόσον δεν υπάρχει έγκυρη κράτηση σε αυτήν. Τέλος με του δίνεται η δυνατότητα να προσθέσει μία νέα ημερομηνία στη διοργάνωση του, με

τη χρήση των DatePicker και ptTimeSelect που αναλύθηκαν στην προηγούμενη παράγραφο.



ΕΙΚΟΝΑ 2.26 – ΠΛΗΡΟΦΟΡΙΕΣ ΗΜΕΡΟΜΗΝΙΩΝ ΚΑΙ ΕΙΣΑΓΩΓΗ ΝΕΑΣ

Η σελίδα αποτελείται από 3 φόρμες HTML. Μία για τις βασικές πληροφορίες της διοργάνωσης, μία για τη διαγραφή και μία για την εισαγωγή ημερομηνίας, με τα ανάλογα submit buttons (Save Changes, Add Date, Delete Icon). Ανάλογα το ποιο submit επιλέχθηκε ο server μέσω της PHP ελέγχει πιο ήταν αυτό και ανάλογα εκτελεί την αντίστοιχη λειτουργία, ανανεώνοντας τη σελίδα.

2.3 ΑΝΑΛΥΣΗ WEB SERVICE

2.3.1 ΕΙΣΑΓΩΓΗ

Η τεχνική που χρησιμοποιήθηκε για την ανάπτυξη του web service είναι αυτή της REST. Σε αυτήν την παράγραφο θα αναλυθούν τα προσφερόμενα paths και πώς μπορεί να τα χρησιμοποιήσει ο χρήστης. Αρχικά γίνεται ανάλυση των resources και κατόπιν περιγράφονται οι λειτουργίες που μπορούν να καλεστούν, με τη χρήση των HTTP VERBS για κάθε path. Επικεντρωνόμαστε μόνο στους χρήστες, καθώς ο server θα αναλυθεί στο επόμενο κεφάλαιο.

2.3.2 RESOURCES

Με τον όρο resource αναφερόμαστε στα «αντικείμενα» που προσφέρονται για προσπέλαση από τον χρήστη, μέσω των URL. Στο web service που αναπτύχθηκε προσφέρονται τα εξής resources:

- **Users:** Αντιπροσωπεύει τους χρήστες (απλούς ή διοργανωτές) που χρησιμοποιούν το σύστημα.
- **Events:** Αντιπροσωπεύει τις διοργανώσεις που αποθηκεύονται στο σύστημα.

- **Reservations:** Αντιπροσωπεύει τις κρατήσεις που πραγματοποιούν οι απλοί χρήστες.
- **Happenings:** Αντιπροσωπεύει τις ημερομηνίες στις οποίες ανταποκρίνεται μία διοργάνωση.

2.3.3 ΧΡΗΣΗ ΤΗΣ ΥΠΗΡΕΣΙΑΣ WEB

Για να καταναλώσει ένας χρήστης το web service πρέπει να στείλει μία αίτηση HTTP στο αντίστοιχο URL. Ο server ανάλογα με την μέθοδο της αίτησης (GET, POST, DELETE), καλεί και την αντίστοιχη λειτουργία που απαντάει στην αίτηση αυτήν. Πρόσβαση στην υπηρεσία έχουν μόνο οι εγγεγραμμένοι χρήστες, καθώς με κάθε αίτηση πρέπει να στέλνουν ένα JSON Web Token (περισσότερα για το JWT στο επόμενο κεφάλαιο) που αποστέλλεται από το server μέσω του header «Token», κάθε φορά που εισάγονται στο σύστημα. Το token έχει διάρκεια ζωής 1 ώρας, καθώς μετά τη λήξη του πρέπει να αιτήσουν καινούριο από το server, και περιέχει τις απαραίτητες πληροφορίες και ασφάλεια για να αναγνωριστεί ο χρήστης. Το token αυτό πρέπει να περιέχεται στο **Authorization Header**, χρησιμοποιώντας το σχήμα **Bearer:** Authorization: Bearer 'token'

Μία αίτηση HTTP αποτελείται από τη μέθοδο (GET,POST κ.ά.) ακολουθούμενη από το URI. Κατόπιν βρίσκουμε τα headers της αίτησης, μία κενή γραμμή και προαιρετικά το σώμα της, που μπορεί να περιέχει δεδομένα, όπως πχ μεταβλητές POST. Παράδειγμα αίτησης:

```
GET /index.html HTTP/1.1
Host: www.example.com
```

Αφού ο server επαληθεύσει το token στέλνει την απόκριση στο χρήστη. Μία απόκριση HTTP περιέχει μία γραμμή με το status code και ένα μήνυμα που δικαιολογεί τον κωδικό. Τα headers τις απόκρισης, μία κενή γραμμή και προαιρετικά το σώμα που περιέχονται δεδομένα, όπως για παράδειγμα ένα αντικείμενο JSON, που χρησιμοποιούμε στις αποκρίσεις του συγκεκριμένου web service.

Όλες οι αιτήσεις προς το server μπορούν να επιστρέψουν τα εξής status codes:

- **500 Internal Server Error:** σε περίπτωση που για το λάθος δεν ευθύνεται ο χρήστης, αλλά ο server.
- **400 Bad Request:** σε περίπτωση που ο χρήστης δεν εισάγει τις απαραίτητες μεταβλητές που απαιτούνται.
- **405 Method Not Allowed:** σε περίπτωση που ο χρήστης στείλει μία αίτηση με μέθοδο που δεν υποστηρίζεται.
- **440 Token Expired:** σε περίπτωση που το token που αποστέλλει ο χρήστης είναι ληγμένο ή αλλοιωμένο.
- **403 Forbidden:** σε περίπτωση που ο χρήστης προσπαθεί να αποκτήσει πρόσβαση σε έναν πόρο που δεν δικαιούται πρόσβαση.

- **440 Not Found:** σε περίπτωση που το URL της αίτησης δεν αποκρίνεται σε κάποιο έγκυρο path.
- **200 OK:** σε περίπτωση που η αίτηση καταναλώθηκε με επιτυχία και δεν επιστρέφεται κάποιο μήνυμα, αλλά ένα JSON αντικείμενο.

2.3.4 ΑΝΑΛΥΣΗ ΤΗΣ ΥΠΗΡΕΣΙΑΣ WEB

Σε αυτήν την παράγραφο θα αναλυθούν τα προσφερόμενα paths για κάθε πόρο/resource, καθώς και οι λειτουργίες που προσφέρει το καθένα. Για να μπορέσει να χρησιμοποιήσει ο χρήστης το web service θα πρέπει να έχει λογαριασμό σε αυτό, ώστε να μπορεί να λάβει το απαραίτητο token που επιτρέπει τη χρήση του. Τα προσφερόμενα paths που αφορούν τους χρήστες, δηλαδή τον πόρο **users**, είναι:

/users: παρέχει λειτουργίες σχετικά με τους χρήστες. Οι μέθοδοι που υποστηρίζονται είναι:

- **POST:** εγγράφει έναν χρήστη στο σύστημα και σε περίπτωση επιτυχίας επιστρέφει ένα access token. Δέχεται ως παραμέτρους: username, password, email, status, firstname, lastname, telephone. Όπου status η ιδιότητα που θέλει να εγγραφτεί ο χρήστης: user για απλός χρήστης ή organizer για διοργανωτής. Μπορεί να επιστρέψει τα headers:
  -200 User 'username' Successfully Registered
  -409 Username or Email already in use
- **GET:** επιστρέφει ένα αντικείμενο JSON με πληροφορίες σχετικές με το χρήστη που έκανε την αίτηση.
- **PUT:** αλλάζει τα δεδομένα του χρήστη ανάλογα με τις παραμέτρους που περάστηκαν. Οι παράμετροι αυτοί μπορεί να είναι: password, firstname, lastname, telephone. Τουλάχιστον μία είναι απαραίτητη. Μπορεί να επιστρέψει τα headers:
  -200 Changes Saved

/users/token: αποστέλλει τα access tokens στους χρήστες. Οι μέθοδοι που υποστηρίζονται είναι:

- **POST:** εγκρίνει το χρήστη και αν είναι επιτυχής η έγκριση επιστρέφεται μέσω του header «Token» το access token που επιτρέπει στο χρήστη να χρησιμοποιεί τις υπηρεσίες του web service. Απαιτούνται οι παράμετροι: username, password, status. Όπου status η ιδιότητα του χρήστη που εγκρίνεται. Μπορεί να επιστρέψει τα headers:
  -200 Authorization Success
  -401 Unauthorized

Τα paths που προσφέρονται για τον πόρο **reservations** σχετίζονται με τη διαχείριση των κρατήσεων. Τα προσφερόμενα paths είναι:

/reservations: παρέχει λειτουργίες σχετικά με τις κρατήσεις ως σύνολο. Οι μέθοδοι που υποστηρίζονται είναι:

- **POST:** κάνει μία καινούρια προσωρινή κράτηση στο όνομα του χρήστη που στέλνει την αίτηση. Αν υπάρχει floor plan οι παράμετροι της αίτησης είναι τα

id των θέσεων (seatIds[]), καθώς και το id του happening (hapId) για το οποίο γίνεται η κράτηση. Σε περίπτωση που δεν υπάρχει floor plan, τότε οι παράμετροι είναι οσηδήποτε συνδυασμοί key=>value, με key το id του τύπου των εισιτηρίων και value τον αριθμό των εισιτηρίων που θέλει ο χρήστης, καθώς και το id του happening (hapId). Μπορεί να επιστρέψει τα headers:
-200 Reservation Held
-400 Tickets Not Available
-400 Organizers can't make reservations

- **GET:** επιστρέφει ένα JSON αντικείμενο με τις κρατήσεις του χρήστη που έστειλε την αίτηση

/reservations/id: παρέχει λειτουργίες σχετικές με μία συγκεκριμένη κράτηση. Οι μέθοδοι που υποστηρίζονται είναι:

- **POST:** Οριστικοποιεί μία προσωρινή κράτηση. Μπορεί να επιστρέψει τα headers:
  -200 Reservation Closed
  -410 Reservation Has Expired
- **GET:** επιστρέφει ένα JSON αντικείμενο με πληροφορίες της συγκεκριμένης κράτησης.
- **DELETE:** ακυρώνει τη συγκεκριμένη κράτηση. Μπορεί να επιστρέψει τα headers:
  -200 Reservation Deleted

Για τον πόρο **events** παρέχονται paths για τη διαχείριση και παρουσίαση των διοργανώσεων Τα προσφερόμενα paths είναι:
/events: παρέχει λειτουργίες για τις δημοσιευμένες λειτουργίες ως σύνολο. Οι μέθοδοι που υποστηρίζονται είναι:

- **POST:** δημιουργεί μία νέα διοργάνωση. Οι παράμετροι που απαιτούνται είναι το όνομα της διοργάνωσης (**name**), η κατηγορία της (**category**), η διεύθυνση (**location**), ο χώρος (**venue**), μία περιγραφή (**details**), δύο πίνακες με κάθε ημερομηνία (**date[]**) και ώρα (**time[]**) που είναι διαθέσιμη η διοργάνωση, 4 πίνακες με κάθε κατηγορία εισιτηρίου που προσφέρεται. Ο κάθε πίνακας περιλαμβάνει την τιμή του εισιτηρίου (**ticket_price[]**), το όνομα της κατηγορίας (**ticket_category[]**), πληροφορίες για την κατηγορία αυτή (**ticket_details[]**) και τον αριθμό των εισιτηρίων που αντιστοιχούν στην κάθε κατηγορία (**ticket_count[]**). Μία παράμετρο για το αν θα υπάρχει floor plan (**floorplan**), που παίρνει τιμές true/false. Αν το floorplan είναι true, τότε απαιτείται επιπλέον η παράμετρος **floorplan_map**, που είναι το αρχείο κειμένου με το map. Τέλος απαιτείται η παράμετρος **preview**, που είναι η εικόνα της διοργάνωσης. Μπορεί να επιστρέψει τα headers:
  - 200 Event Created
  - 400 Bad Request Seats must be equal to total tickets
  - 400 Bad Request Floorplan file not supported. Try a \'.txt\' format.
  - 400 Bad Request Image not supported

- 400 Bad Request MAX image file size: 10MB
- 400 Bad Request Cannot input characters on numeric fields
- 400 Bad Request. Values cannot be empty

- **GET:** επιστρέφει ένα αντικείμενο JSON με όλες τις διαθέσιμες διοργανώσεις. Επιπλέον μπορεί να χρησιμοποιηθεί η παράμετρος **offset** (αρχίζοντας από offset=1), ώστε να επιστραφούν 10 διοργανώσεις τη φορά.

/events/id: παρέχει λειτουργίες για μία συγκεκριμένη διοργάνωση. Οι μέθοδοι που υποστηρίζονται είναι:

- **POST:** Αλλάζει τα στοιχεία της συγκεκριμένης διοργάνωσης. Τα στοιχεία που μπορούν να αλλαχτούν και περνιούνται ως παράμετροι είναι το όνομα (**name**), η διεύθυνση (**location**), ο χώρος (**venue**), η κατηγορία (**category**), η περιγραφή (**details**) και η εικόνα (**preview**). Μπορεί να επιστρέψει τα headers:
  -200 Changes Saved
  -400 Image Not Supported
- **GET:** Επιστρέφει ένα αντικείμενο JSON με πληροφορίες για τη διοργάνωση και τις διαθέσιμες ημερομηνίες της.
- **DELETE:** Διαγράφει την συγκεκριμένη διοργάνωση. Μπορεί να επιστρέψει τα headers:
  -200 Deleted Successfully
  -400 Cannot delete an even with a closed reservation

/events/search/keyword: παρέχει τη δυνατότητα αναζήτησης με βάση το keyword. Οι μέθοδοι που υποστηρίζονται είναι:

- **GET:** Επιστρέφει όλες τις διοργανώσεις που το όνομα τους αντιστοιχεί στην λέξη "keyword" που περάστηκε στο URL. Επιπλέον μπορεί να χρησιμοποιηθεί η παράμετρος **offset** (αρχίζοντας από offset=1), ώστε να επιστραφούν 10 διοργανώσεις τη φορά.

/events/myEvents: παρέχει πρόσβαση στις διοργανώσεις ενός διοργανωτή. Οι μέθοδοι που υποστηρίζονται είναι:

- **GET:** Επιστρέφει ένα αντικείμενο JSON με όλες τις διοργανώσεις που έχει δημιουργήσει ο χρήστης που κάνει την αίτηση.

/events/myEvents/id: παρέχει πληροφορίες για το διοργανωτή για μία συγκεκριμένη διοργάνωσή του. Οι μέθοδοι που υποστηρίζονται είναι:

- **GET:** Επιστρέφει ένα αντικείμενο JSON με πληροφορίες σχετικά με τη συγκεκριμένη διοργάνωση, καθώς και πληροφορίες για την προπώληση των εισιτηρίων της κάθε ημερομηνίας.

Τέλος τα path που σχετίζονται με τον πόρο **happenings** αφορούν τις ημερομηνίες των διοργανώσεων. Τα προσφερόμενα path είναι:

/happenings: παρέχει εργαλεία για τη διαχείριση των ημερομηνιών των διοργανώσεων. Οι μέθοδοι που υποστηρίζονται είναι:

- **POST:** Προσθέτει μία νέα ημερομηνία σε μία υπάρχον διοργάνωση. Απαιτείται το id της διοργάνωσης (**eventId**), καθώς και η ημερομηνία (**date**) και η ώρα που θα προστεθούν (**time**). Μπορεί να επιστρέψει τα headers:
  -200 Happening added successfully

/happenings/id: παρέχει λειτουργίες για τη διαχείριση μίας συγκεκριμένης ημερομηνίας.
Οι μέθοδοι που υποστηρίζονται είναι:

- **GET:** Επιστρέφει ένα αντικείμενο JSON με τις διαθέσιμες θέσεις μίας συγκεκριμένης ημερομηνίας. Αν η διοργάνωση περιέχει floor plan, τότε ο χάρτης του floor plan περιλαμβάνεται στο JSON, καθώς και η κλεισμένες θέσεις.
- **DELETE:** Διαγράφει τη συγκεκριμένη ημερομηνία από τη διοργάνωση που ανήκει. Μπορεί να επιστρέψει τα headers:
  -200 Deleted Successfully
  -400 Cannot delete a happening with a closed reservation

Οποιοδήποτε χρήστης/προγραμματιστής μπορεί να χρησιμοποιήσει την υπηρεσία αυτή με την προϋπόθεση ότι η πλατφόρμα στην οποία δουλεύει μπορεί να στείλει τις ανάλογες HTTP αιτήσεις. Το μεγάλο πλεονέκτημα των προτύπων JSON και HTTP, που χρησιμοποιούνται για την υλοποίηση του web service, είναι ανεξάρτητα πλατφόρμας. Αυτό δίνει ευελιξία στους χρήστες, καθώς μπορούν να χρησιμοποιήσουν το web service στις δικές τους εφαρμογές, ανεξάρτητα από τα εργαλεία που χρησιμοποιούν για την ανάπτυξη αυτής.

3□ □□□ □□□ □□

Η ΠΛΕΥΡΑ ΤΟΥ ΔΙΑΚΟΜΙΣΤΗ

3.1 ΕΙΣΑΓΩΓΗ

Έως αυτό το σημείο είδαμε το πώς ένας χρήστης μπορεί να χρησιμοποιήσει την ιστοσελίδα και το web service. Αναλύθηκε το interface που προσφέρει η ιστοσελίδα στο χρήστη, καθώς και οι λειτουργίες που δημοσιεύονται από το web service προς αυτόν. Όλα αυτό αναφέρονται στο front end (η πλευρά του πελάτη) της εφαρμογής μας. Σε αυτό το κεφάλαιο θα ασχοληθούμε με το back end (η πλευρά του διακομιστή), δηλαδή με όλα αυτά που δεν βλέπει ο χρήστης. Αυτά αφορούν τη βάση δεδομένων, το πώς ο διακομιστής παίρνει πληροφορίες από αυτήν και τις παρουσιάζει στο χρήστη, την εισαγωγή των νέων δεδομένων στη βάση, καθώς και η επαλήθευση τους, η επίτευξη της αρχιτεκτονικής REST του web service καθώς και άλλα στοιχεία που θα αναλυθούν παρακάτω.

3.2 SERVER-SIDE ΕΡΓΑΛΕΙΑ

3.2.1 Ο APACHE SERVER

Ο διακομιστής που χρησιμοποιήθηκε για την φιλοξενία τόσο της ιστοσελίδας, όσο και του web service είναι ο Apache HTTP Server (Apache). Ο apache παρέχει διάφορες λειτουργίες, όπως η χρήση της PHP (PHP: Hypertext Preprocessor), η οποία είναι μία server-side γλώσσα προγραμματισμού, που ενσωματώνεται με την HTML και επικοινωνεί με το server. Επιπλέον μπορεί να προσφέρει έξτρα ασφάλεια στην επικοινωνία του πελάτη με το διακομιστή με τη χρήση κρυπτογράφησης μέσω SSL (Secure Socket Layers). Τέλος ο Apache μας προσφέρει το εργαλείο mod_rewrite, μέσω του οποίου μπορούμε να επαναγράψουμε τα URL που στέλνει ο χρήστης, ώστε να πετύχουμε την αρχιτεκτονική REST.

Πηγή (*https://httpd.apache.org/ABOUT_APACHE.html)*

3.2.2 MARIADB/PHPMYADMIN

Για την ανάπτυξη της βάσης δεδομένων χρησιμοποιήθηκε η MariaDB. Η MariaDB είναι ένα σύστημα διαχείρισης βάσεων δεδομένων, το οποίο τρέχει έναν διακομιστή, που παρέχει πρόσβαση σε ένα σύνολο από βάσεις δεδομένων. Το συγκεκριμένο ΣΔΒΔ άρχισε να αναπτύσσεται όταν πωλήθηκε η MySQL στην Oracle, ως αντίγραφό της.

Για την παροχή ενός graphic interface που διαχειρίζεται τη βάση δεδομένων χρησιμοποιήθηκε το εργαλείο phpMyAdmin. Αυτό είναι ένα ανοιχτού κώδικα εργαλείο γραμμένο σε PHP, που επιτρέπει τη διαχείριση μίας βάσης MySQL ή MariaDB μέσω web browser.

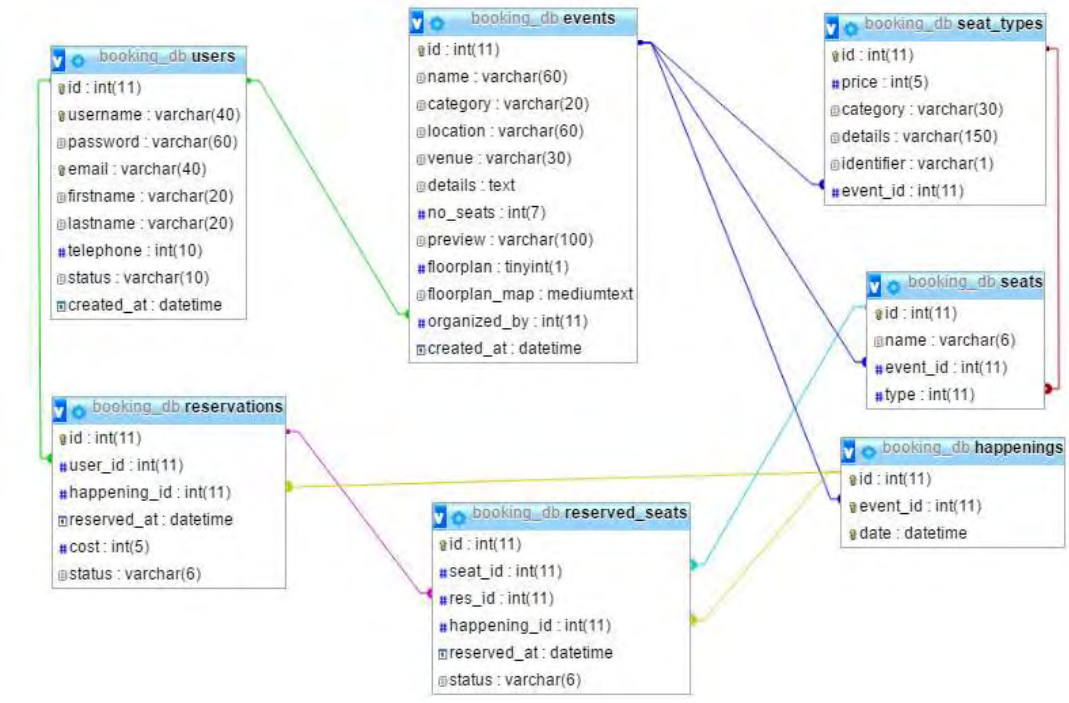
3.2.3 XAMPP

Για την αξιοποίηση όλων των παραπάνω χρησιμοποιήθηκε το πακέτο προγραμμάτων ελεύθερου λογισμικού, λογισμικού ανοικτού κώδικα και ανεξαρτήτου πλατφόρμας XAMPP. Το πακέτο XAMPP περιέχει τον διακομιστή Apache, τη βάση δεδομένων MariaDB καθώς και ένα διερμηνέα για κώδικα γραμμένο σε PHP. Επίσης παρέχει ένα φιλικό περιβάλλον για τη διαχείριση όλων των παραπάνω εργαλείων μέσω ενός πίνακα ελέγχου. Το XAMPP καθιστά απλό στους προγραμματιστές να δημιουργήσουν έναν τοπικό web server, ο οποίος μπορεί με ευκολία να ανέβει στο διαδίκτυο καθώς οι περισσότεροι online web servers χρησιμοποιούν τα ίδια εργαλεία με αυτό.

3.3 Η ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

3.3.1 ΣΧΗΜΑ ΚΑΙ ΠΙΝΑΚΕΣ

Όλα τα δεδομένα που αφορούν τους χρήστες, τις διοργανώσεις και τις κρατήσεις βρίσκονται αποθηκευμένα σε μία βάση δεδομένων. Τόσο η ιστοσελίδα, όσο και το web service επικοινωνούν με τη βάση, ώστε να παρέχουν τα δεδομένα αυτά στους χρήστες. Η βάση αποτελείται από 7 πίνακες:



ΕΙΚΟΝΑ 3.1 – DATABASE SCHEMA

1. **users:** Σε αυτόν τον πίνακα αποθηκεύονται οι πληροφορίες για τους χρήστες. Κάθε χρήστης έχει ένα μοναδικό αναγνωριστικό **id**, το οποίο εισάγεται αυτόματα από τη βάση, μαζί με τις πληροφορίες που δόθηκαν κατά την εγγραφή του. Για λόγους ασφαλείας στο πεδίο **password**, δεν αποθηκεύεται αυτούσιος ο κωδικός του χρήστη, αλλά η SHA-256 (Secure Hashing Algorithm) αναπαράσταση του. Τέλος αποθηκεύεται η ημερομηνία την οποία εγγράφτηκε ο χρήστης.
2. **events:** Ο πίνακας αυτός περιέχει πληροφορίες που σχετίζονται με τις διοργανώσεις. Εδώ θα βρούμε όλες τις βασικές πληροφορίες, όπως όνομα, κατηγορία, τόπος, περιγραφή, αριθμός θέσεων, ύπαρξη ή όχι floor plan, id του χρήστη που τη διοργανώνει καθώς και την εικόνα της διοργάνωσης. Για εξοικονόμηση ταχύτητας, αλλά και χώρου στη βάση, στο πεδίο preview δεν αποθηκεύεται η εικόνα ως έχειν, αλλά το path στο οποίο είναι αποθηκευμένη στο web server η πραγματική εικόνα. Αυτό μας επιτρέπει να αποθηκεύσουμε μεγάλου μεγέθους εικόνες, καθώς και να τις φορτώνουμε πιο γρήγορα. Τέλος, όπως και πριν, αποθηκεύεται η ημερομηνία δημιουργίας της διοργάνωσης.
3. **seats:** Στον πίνακα αυτόν αποθηκεύονται όλες οι θέσεις που είναι διαθέσιμες για τις διοργανώσεις. Οι θέσεις που δημιουργούνται είναι μοναδικές και ανεξάρτητες από το πόσες φορές λαμβάνει τόπο μία διοργάνωση. Κάθε θέση αναγνωρίζεται από το μοναδικό **id.** Στο πεδίο name, αποθηκεύεται μία πιο φιλική προς το χρήστη ονομασία, η οποία είναι είτε ένας αύξον αριθμός που αντιστοιχεί στο συνολικό αριθμό θέσεων, είτε η θέση της στο floor plan (πχ B13). Επίσης αποθηκεύεται το id της διοργάνωσης στην οποία ανήκει αυτή η θέση (**event_id**), καθώς και η πολιτική εισιτηρίου που την χαρακτηρίζει, με το αντίστοιχο id (**type**).
4. **seat_types:** Κάθε θέση που είναι αποθηκευμένη στον πίνακα **seats** ακολουθεί μία πολιτική εισιτηρίου. Ο πίνακας αυτός περιέχει όλους τους διαθέσιμους τύπους εισιτηρίων που υπάρχουν, χαρακτηριζόμενοι από ένα μοναδικό **id**. Οι πληροφορίες που περιγράφουν μία πολιτική είναι το κόστος της θέσης, το όνομα της κατηγορίας, η περιγραφή αυτής, καθώς και ένας αναγνωριστικός αύξον χαρακτήρας που αντιπροσωπεύει τη θέση αυτή στο floor plan. Σε περίπτωση μη ύπαρξης floor plan, το πεδίο αυτό αγνοείται. Τέλος αποθηκεύεται το id **(event_id)** της διοργάνωσης στο οποίο ανήκει η συγκεκριμένη πολιτική.
5. **happenings:** Σε αυτόν τον πίνακα αποθηκεύονται οι διαθέσιμες ημερομηνίες για κάθε διοργάνωση. Εκτός της ημερομηνίας και του id που αντιστοιχεί στη συγκεκριμένη διοργάνωση (**event_id)**, έχουμε και το μοναδικό **id** που χρησιμιοιείται ως αναγνωριστικό για κάθε διαθέσιμη ημερομηνία.
6. **reservations:** Ο πίνακας αυτός περιέχει γενικές πληροφορίες για τις κρατήσεις. Αυτές είναι το μοναδικό **id,** το id του χρήστη ο οποίος έκανε την

κράτηση (**user_id**), η ημερομηνία την οποία έγινε η κράτηση (**reserved_at**), το συνολικό κόστος της, το id της ημερομηνίας στην οποία αναφέρεται (**happening_id**), καθώς και την κατάσταση της, δηλαδή αν είναι οριστικοποιημένη (closed) ή προσωρινή (open). Μία κράτηση μπορεί να περιλαμβάνει πολλαπλές θέσεις.

7. **reserved_seats:** Στον πίνακα αυτόν γίνονται ουσιαστικά οι κρατήσεις, δηλαδή η ανάθεση μίας θέσης σε έναν χρήστη. Ο πίνακας αυτός μας δίνει τη δυνατότητα να κρατάμε μία θέση, μέσω reference από τον πίνακα seats, για πολλές ημερομηνίες μίας συγκεκριμένης διοργάνωσης. Κάθε φορά που πραγματοποιείται μία κράτηση (reservation), οι θέσεις που αντιστοιχούν σε αυτή αποθηκεύονται στον πίνακα αυτόν. Τα στοιχεία που κρατούνται για κάθε θέση είναι το id της που βρίσκεται στον πίνακα seats (**seat_id**), το id της κράτησης για την οποία είναι αναθετημένη αυτή η θέση (**res_id**), το id της ημερομηνίας για την οποία κρατήθηκε η θέση αυτή (**happening_id**), η ημερομηνία που πραγματοποιήθηκε η κράτηση της θέσης (**reserved_at**), την κατάστασή της (**status**), δηλαδή αν είναι οριστικοποιημένη ή όχι, καθώς και ένα μοναδικό αναγνωριστικό **id.**

3.3.2 Η ΜΗΧΑΝΗ ΑΠΟΘΗΚΕΥΣΗΣ

Η MariaDB προσφέρει ένα εύρος μηχανών αποθήκευσης των δεδομένων της βάσης. Στην εφαρμογή αυτή χρησιμοποιήθηκε η μηχανή InnoDB. Η InnoDB υποστηρίζει τη δημιουργία ξένων κλειδιών μεταξύ των πινάκων. Αυτό μας δίνει μεγαλύτερο έλεγχο των δεδομένων, καθώς αυτά συνδέονται μεταξύ τους. Έτσι για παράδειγμα όταν ένας διοργανωτής θέλει να ακυρώσει μία διοργάνωση του, τότε η διαγραφή όλων των δεδομένων που σχετίζονται με αυτή, διαγράφονται αυτόματα με τη διοργάνωση.

Ένας ακόμα λόγος για την επιλογή της InnoDB είναι η υποστήριξη συναλλαγών. Οι συναλλαγές μας δίνουν τη δυνατότητα να τρέξουμε εντολές στη βάση, τον οποίων τα αποτελέσματα δεν μονιμοποιούνται κατευθείαν. Σε περίπτωση κάποιου σφάλματος στη βάση μπορούμε να αναιρέσουμε όλες τις αλλαγές που έγιναν αφότου άρχισε η συναλλαγή, με την εντολή rollback. Όταν όλα κυλήσουν ομαλά και δεν υπάρχουν σφάλματα τότε οριστικοποιούμε τις αλλαγές αυτές με την εντολή commit. Επίσης μέσα σε μία συναλλαγή η InnoDB μας παρέχει λειτουργίες ταυτοχρονισμού, τις οποίες αξιοποιούμε για να αποφύγουμε τις διπλές κρατήσεις.

Πηγή (*https://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html)*

3.3.3 ΡΟΥΤΙΝΕΣ

Η ρουτίνες είναι κομμάτια κώδικα SQL που τρέχουν ανά τακτά χρονικά διαστήματα. Όταν ένας χρήστης αφήσει μία ανοιχτή κράτηση τότε μετά από δέκα λεπτά παύει να είναι έγκυρη. Όπως θα δούμε παρακάτω αυτό δεν σημαίνει ότι η εγγραφή στη βάση διαγράφεται. Για αυτό το λόγο χρησιμοποιούμε μία ρουτίνα σε

ημερήσια βάση η οποία βρίσκει τις κρατημένες θέσεις οι οποίες έχουν λήξει, δηλαδή έχουν περάσει πάνω από δέκα λεπτά από την έκδοση τους και η κατάσταση τους είναι “open”. Με την συνάρτηση DATE_ADD(reserved_at, INTERVAL +12 MINUTES), μπορούμε να επιλέξουμε αυτές τις θέσεις και να τις διαγράψουμε.

Όπως είδαμε ένας διοργανωτής έχει τη δυνατότητα να διαγράψει μία διοργάνωση ή μία ημερομηνία αυτής, όταν αυτή δεν έχει έγκυρες κρατήσεις. Σε περίπτωση που δεν τη διαγράψει χειροκίνητα τότε αυτή μένει στο σύστημα ως άχρηστη πληροφορία. Για το λόγο αυτό χρησιμοποιείται μία ρουτίνα, η οποία σε ημερήσια βάση βρίσκει και διαγράφει τις ημερομηνίες οι οποίες έχουν λήξει προ ημέρας. Σε περίπτωση εύρεσης και διαγραφής, η ρουτίνα ελέγχει τις διοργανώσεις και αν στην περίπτωση που βρει διοργάνωση χωρίς ανατεθειμένη ημερομηνία, δηλαδή η ημερομηνία που διαγράφτηκε ήταν η τελευταία, διαγράφει την διοργάνωση:

```
BEGIN
  DELETE FROM happenings WHERE DATE_ADD (date, INTERVAL +1 DAY) < NOW();
  IF (ROW_COUNT()> 0) THEN
    DELETE events FROM events LEFT JOIN happenings ON
    events.id=happenings.event_id WHERE happenings.id IS NULL;
  END IF;
END
```

3.3.4 ΕΠΙΚΟΙΝΩΝΙΑ PHP ΜΕ ΒΑΣΗ

Η server-side γλώσσα PHP μας δίνει τα απαραίτητα εργαλεία ώστε να μπορέσει να επικοινωνήσει με τη βάση δεδομένων. Η επέκταση mysqli της PHP παρέχει όλες τις συναρτήσεις ώστε να πετύχουμε την σύνδεση και την επικοινωνία αυτή. Με την εντολή: $conn mysqli_connect($host, $user, $pass) συνδεόμαστε στο server της βάσης και αποθηκεύουμε τη σύνδεση σε μια μεταβλητή $conn. Έπειτα επιλέγουμε τη συγκεκριμένη βάση δεδομένων που θέλουμε, με την εντολή: mysqli_select_db($conn, $db_name). Αφού ελέγξουμε αν η σύνδεση ήταν επιτυχής, τότε με το αντικείμενο $conn μπορούμε να αξιοποιήσουμε τις λειτουργίες της βάσης. Χρησιμοποιώντας την εντολή mysqli_query($conn, $query), μπορούμε να στέλνουμε ερωτήματα προς τη βάση. Για να διαβάσουμε τα αποτελέσματα του ερωτήματος αν υπάρχουν, χρησιμοποιούμε την εντολή mysqli_result() περνώντας ως όρισμα το αντικείμενο που επέστρεψε η mysqli_query. Έπειτα σε μία loop χρησιμοποιούμε την εντολή mysqli_fetch_assoc(), περνώντας ως όρισμα το mysqli_result, η οποία μας επιστρέφει ένα associative πίνακα, όπου το κλειδί είναι το όνομα του πεδίου της βάσης και η τιμή του η τιμή της βάσης.

```
$mysql_host = 'localhost';
$mysql_user = 'root';
$mysql_pass = '';
$mysql_db = 'booking_db';
$conn = mysqli_connect($mysql_host, $mysql_user, $mysql_pass);
if(!$conn){
      die('Connection failed: '.mysqli_connect_error());
```

```
}else{
      if(!@mysqli_select_db( $conn, $mysql_db)){
            die('Error    connecting    to    database    '.    $mysql_db);
            }
 }
```

3.4 SERVER-SIDE ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΙΣΤΟΣΕΛΙΔΑΣ

3.4.1 ΑΠΟΣΤΟΛΗ ΔΕΔΟΜΕΝΩΝ ΧΡΗΣΤΗ ΜΕ HTML FORM

Όπως είδαμε στο κεφάλαιο 2 ένα μεγάλο μέρος του client βασίζεται σε HTML φόρμες. Τέτοιες περιπτώσεις ήταν, η επιλογή των θέσεων/εισιτηρίων, η εγγραφή/είσοδος του χρήστη, η δημιουργία διοργάνωσης κ.ά. Όταν ένας χρήστης συμπληρώσει υποβάλλει μία φόρμα HTML, τότε τα δεδομένα αυτά στέλνονται μέσω μίας μεθόδου HTTP (GET/POST),η οποία καθορίζεται από την ιδιότητα "method" της φόρμας, σε ένα αρχείο το οποίο καθορίζεται μέσω της ιδιότητας "action". Τα δεδομένα κωδικοποιούνται, συνήθως με url-encoding (κεφαλίδα περιεχομένου/Content-type: application/x-www-form-urlencoded),και αποστέλλονται σε αυτό. Έπειτα από το αρχείο που διευκρινίστηκε στο "action" της φόρμας μπορούμε να διαβάσουμε τα δεδομένα αυτά μέσω της PHP, ανάλογα με τη μέθοδο που χρησιμοποιήθηκε. Κάθε input της φόρμας αναγνωρίζεται από την ιδιότητα name και χρησιμοποιώντας την τιμή αυτήν διαβάζουμε τα δεδομένα με τη βοήθεια των πινάκων $_POST['name'], $_GET['name'], για τις αντίστοιχες μεθόδους POST και GET.

Οι μεταβλητές GET μεταφέρονται μέσω του URL της σελίδας που φορτώνεται και είναι ορατές στο χρήστη, ενώ οι μεταβλητές που στέλνονται με τη μέθοδο POST περιέχονται στο σώμα του μηνύματος της HTTP αίτησης. Για αυτό το λόγο τα δεδομένα που στέλνονται μέσω GET δεν πρέπει να είναι ευαίσθητα. Για τη μεταφορά τύπου POST παραμέτρων απαιτείται η κεφαλίδα περιεχομένου να είναι η default "**application/x-www-form-urlencoded**" για url-encoding ή η **multipart/form-data**, η οποία χρησιμοποιείται για αποστολή δυαδικών αρχείων, όπως για παράδειγμα εικόνες.

3.4.2 ΣΥΝΔΕΣΗ/ΕΓΓΡΑΦΗ ΧΡΗΣΤΗ

Με το που συμπληρώσει ο χρήστης τη φόρμα σύνδεσης, τότε τα δεδομένα στέλνονται στην ίδια σελίδα, η οποία ελέγχει αν τα στοιχεία συμπληρώθηκαν. Σε περίπτωση που δεν είναι συμπληρωμένα τα στοιχεία προχωράει με την HTML. Ο έλεγχος αυτός γίνεται με τη χρήση της συνάρτησης isset($var), η οποία επιστρέφει false αν η παράμετρος που περνάμε δεν είναι ορισμένη. Συνεπώς ελέγχουμε αν είναι ορισμένο το username και το password με τη χρήση της παρακάτω συνάρτησης: **if(isset($_POST['username'], $_POST['password'])**. Έπειτα καθώς τα στοιχεία

αυτά θα εισαχθούν σε βάση δεδομένων πρέπει να βεβαιωθούμε ότι είναι ασφαλή. Με τη χρήση της συνάρτησης **mysqli_real_escape_string($conn, $var)**, που δέχεται σαν όρισμα τη σύνδεση με το server της βάσης και τη μεταβλητή που θέλουμε να ασφαλίσουμε, γίνονται escape όλοι οι χαρακτήρες που θα μπορούσαν να οδηγήσουν σε SQL injection. Με τη μέθοδο των SQL injections κακόβουλοι χρήστες επιδιώκουν να εκμεταλλευτούν ευπάθειες στην ασφαλεία μιας εφαρμογής web, ώστε να επέμβουν στα δεδομένα της βάσης. Στην περίπτωση ελλιπούς επαλήθευσης των δεδομένων εισόδου είναι σε κάποιες περιπτώσεις δυνατόν να περαστούν extra εντολές SQL προς εκτέλεση στην εφαρμογή ή να αλλαχτούν μέρη μιας ερώτησης SQL με τρόπο που δεν έχει προβλεφθεί. Με τη χρήση της παραπάνω εντολής, οι χαρακτήρες των δεδομένων που έβαλε ο χρήστης λαμβάνονται από τη βάση ως κανονικοί χαρακτήρες και δεν εμπλέκονται στο συντακτικό του interpreter. Ανάλογη λειτουργία γίνεται και για την επεξεργασία των δεδομένων στην εγγραφή του χρήστη.

Αφού διασφαλιστούν τα δεδομένα τότε με τη χρήση της συνάρτησης **hash('sha256', $pass),** λαμβάνουμε τη hash αναπαράσταση του κωδικού που έβαλε ο χρήστης. Ο αλγόριθμος SHA-256 παρέχει μία μονόδρομη κρυπτογράφηση της μεταβλητής που περνάμε. Σε περίπτωση που κάποιος κακόβουλος χρήστης αποκτήσει πρόσβαση στη βάση, τότε οι κωδικοί των χρηστών παραμένουν ασφαλισμένοι. Κατόπιν ανάλογα αν γίνεται σύνδεση η εγγραφή, κάνουμε το ανάλογο ερώτημα στη βάση με τα παραπάνω στοιχεία και σε περίπτωση επιτυχίας ανακατευθύνουμε το χρήστη, αλλιώς του εμφανίζεται μέσω JavaScript σχετικό μήνυμα αποτυχίας. Η ανακατεύθυνση επιτυγχάνεται με τη χρήση της συνάρτησης **header** της PHP, η οποία ορίζει τις κεφαλίδες HTTP στις αιτήσεις που γίνονται. Με την εντολή: **header('Location: index.php')**, στέλνεται ένα header το οποίο ανακατευθύνει το χρήστη στην αρχική σελίδα. Ανάλογα με τη λειτουργία κατευθύνουμε το χρήστη εκεί που θέλουμε.

Με την επιτυχή σύνδεση/εγγραφή του χρήστη δημιουργούμε τρεις SESSION μεταβλητές. Οι SESSION μεταβλητές κρατάνε πληροφορίες για ένα χρήστη, οι οποίες είναι προσβάσιμες από όλες τις σελίδες. Αυτές είναι το **username**, το **id** και το **status**. Το id μας βοηθάει ώστε να αναγνωρίζουμε αν ο χρήστης είναι διοργανωτής ή απλός και να του προσφέρουμε τις ανάλογες υπηρεσίες. Το **id** χρησιμεύει για την εισαγωγή στοιχείων στη βάση, όπως όταν δημιουργεί μία διοργάνωση ή όταν κάνει κάποια κράτηση. Τέλος το username χρησιμοποιείται σε διάφορα μέρη του interface. Για να δημιουργηθούν αυτές οι μεταβλητές στην αρχή κάθε σελίδας καλούμε τη συνάρτηση **session_start()** και μέσω του associative πίνακα $_SESSION, έχουμε πρόσβαση σε αυτές. Τέλος κατά την έξοδο του χρήστη από το σύστημα χρησιμοποιούμε την εντολή **session_destroy()**, η οποία διαγράφει οποιαδήποτε session μεταβλητή είναι μέχρι τώρα αποθηκευμένη.

ΕΙΣΟΔΟΣ ΧΡΗΣΤΗ ΣΤΟ ΣΥΣΤΗΜΑ:

```
session_start();
//Κάλεσμα αρχείου που περιέχει τη σύνδεση με το server
require_once 'connect.inc.php';
```

```
//Έλεγχος παραμέτρων POST
if(isset ($_POST['username'], $_POST['password'])){
    $name = mysqli_real_escape_string($conn, $_POST['username']);
    $pass = mysqli_real_escape_string($conn, $_POST['password']);
    if(!empty($name) && !empty($pass)){
        //Επαλήθευση χρήστη
        $query = "SELECT * FROM users WHERE username='$name' AND password
        = '".hash('sha256',$pass)."'";
        if($result = mysqli_query($conn, $query)){
            //Αν βρέθηκε εγγραφή με τα παραπάνω στοιχεία, θέσε τις SESSION
              μεταβλητές
            if(mysqli_affected_rows($conn)>0){
                $row = mysqli_fetch_assoc($result);
                $id = $row['id'];
                $name = $row['username'];
                $status = $row['status'];
                $_SESSION['user_id'] = $id;
                $_SESSION['username']= $name;
                $_SESSION['status']= $status;
                //Ανακατεύθυνση
                header('Location: ../myEvents.php');
            //Ανάλογα το λάθος βγάλε αντίστοιχο javascript μήνυμα
            }else{
                echo"<script type='text/javascript'>
                    alert('Username/password combination doesn\'t
                    exist.');</script>";
            }
        }else{
            echo "<script type='text/javascript'>alert('Server
                problem');</script>";
        }
    }else{
        echo "<script type='text/javascript'>alert('Cannot insert empty
            values');</script>";
    }
}
```

3.4.3 ΠΑΡΟΥΣΙΑΣΗ ΔΙΟΡΓΑΝΩΣΕΩΝ

Στην αρχική σελίδα οι διοργανώσεις που παρουσιάζονται στους χρήστες εμφανίζονται ανά δεκάδες. Όταν φορτώνει η σελίδα, στέλνεται ένα **SELECT** ερώτημα στη βάση το οποίο επιστρέφει τα δεδομένα για τις πρώτες δέκα διοργανώσεις. Αυτό επιτυγχάνεται με την εντολή **LIMIT**. Επίσης τα γεγονότα ταξινομούνται κατά ημερομηνία, χρησιμοποιώντας την εντολή **ORDER BY**. Τέλος με την εντολή **OFFSET**, μπορούμε να ορίσουμε από πού θα αρχίσει η αναζήτηση των εγγραφών.

Όταν τρέξει η αρχική σελίδα. μέσω PHP ελέγχουμε αν έχει οριστεί η **GET** παράμετρος **page** και την αποθηκεύουμε σε μία μεταβλητή **$page,** η οποία υποδεικνύει ποια δεκάδα εμφανίζεται. Σε περίπτωση που δεν είναι ορισμένη, η μεταβλητή αυτή παίρνει την τιμή **0**. Έπειτα κάνουμε το **SELECT** ερώτημα προς τη

βάση θέτοντας **OFFSET $page*10**. Ανάλογα, λοιπόν, τη μεταβλητή αυτή λαμβάνουμε και την αντίστοιχη δεκάδα διοργανώσεων με τη σελίδα που βρισκόμαστε. Για παράδειγμα ,την πρώτη φορά που επισκεπτόμαστε τη σελίδα, το **$page** είναι **0,** συνεπώς η εντολή **OFFSET** έχει τιμή 0 και εμφανίζονται τα δέκα πρώτα στοιχεία της βάσης. Με την εντολή **mysqli_affected_rows($conn)**, ελέγχουμε πόσες εγγραφές γύρισε το ερώτημα **SELECT**. Αν αυτές είναι τουλάχιστον 10, τότε εμφανίζουμε το κουμπί **next**, που παρουσιάστηκε στην παράγραφο 2.2.2. Επιπλέον αν το **page** δεν είναι **0**, δηλαδή δεν βρισκόμαστε στη σελίδα με την πρώτη δεκάδα, εμφανίζεται και το κουμπί **previous.** Ανάλογα με το ποιο κουμπί θα επιλέξει ο χρήστης, στέλνουμε με τη μέθοδο **GET** τον αριθμό της τωρινής σελίδας (**$page**) προσθέτοντας ή αφαιρώντας 1. Με αυτόν τον τρόπο μπορούμε να ελέγξουμε το ποιες δεκάδες εμφανίζονται ανάλογα με την τιμή της παραμέτρου που βρίσκεται στο URL. Η ίδια λειτουργία εφαρμόζεται και στη σελίδα της αναζήτησης.

ΛΗΨΗ ΔΙΟΡΓΑΝΩΣΕΩΝ ΑΝΑΛΟΓΑ ΤΗ ΣΕΛΙΔΑ

```
//Αν δεν έχει οριστεί η παράμετρος page βρσικόμαστε στη πρώτη σελίδα
if(empty($_GET['page'])){
    $page=0;
    $previous =false;
}else{
    $page = mysqli_real_escape_string($conn, $_GET['page']);
    $previous = true;
}
$offset = $page*10;
$query = "SELECT name, preview, details, venue, location, id, floorplan
          FROM events ORDER BY created_at DESC LIMIT 10 OFFSET $offset";
$result = mysqli_query($conn, $query);
//Αν δεν λήφθηκαν δέκα διοργανώσεις φτάσαμε στο τέλος
if(mysqli_affected_rows($conn)!=10){
    $next = false;
}else{
    $next = true;
}
//Ανάλογα τις μεταβλητές previous και next εμφανίζονται τα αντίστοιχα
κουμπιά
```

3.4.4 ΔΗΜΙΟΥΡΓΙΑ ΔΙΟΡΓΑΝΩΣΕΩΝ

Στο κεφάλαιο 2 έγινε ανάλυση των δεδομένων που απαιτούνται να συμπληρώσει ο χρήστης ώστε να δημιουργηθεί μία διοργάνωση. Στην παράγραφο αυτή θα δούμε πως επεξεργάζεται ο διακομιστής αυτά τα στοιχεία και ποιοι έλεγχοι γίνονται ώστε να εξασφαλιστεί η εγκυρότητα των δεδομένων αυτών.

Αφού ο χρήστης συμπληρώσει τη φόρμα με τα απαραίτητα στοιχεία, αυτά στέλνονται στην ίδια σελίδα μέσω της μεθόδου POST. Επειδή η φόρμα περιέχει αρχεία που στέλνονται μαζί με τις παραμέτρους, χρησιμοποιείται η κωδικοποίηση multipart/data-form. Μόλις σταλούν τα δεδομένα ο server ελέγχει αν έχουν οριστεί

όλες οι απαιτούμενες παράμετροι. Σε περίπτωση που όλα είναι ορισμένα συνεχίζει στην επαλήθευση τους, αλλιώς εμφανίζει σχετικό μήνυμα. Όπως προαναφέρθηκε με τη χρήση της **mysqli_real_escape_string()** ασφαλίζεται η βάση από SQL Injections. Επίσης με τη χρήση της **is_numeric()**, ελέγχουμε αν τα πεδία που απαιτούν μόνο αριθμητικές τιμές είναι έγκυρα, όπως οι τιμές των εισιτηρίων, ο αριθμός των θέσεων κ.ά.

Στο επόμενο βήμα ελέγχεται η εικόνα που ανέβασε ο χρήστης. Η PHP μας παρέχει πρόσβαση στα ανεβασμένα αρχεία και μερικές από τις ιδιότητες τους, μέσω του πίνακα **$_FILES**. Μέσω του ονόματος του πεδίου στη φόρμα (**preview**) αποκτούμε πρόσβαση στο μέγεθος του αρχείο μέσω της μεταβλητής **$_FILES[preview][size]**. Αν το συγκεκριμένο μέγεθος ξεπερνά τα 10MB εμφανίζεται ανάλογο μήνυμα. Στη συνέχεια με τη χρήση της συνάρτησης **finfo_file()**, λαμβάνουμε πληροφορίες σχετικά με το αρχείο που ανέβασε ο χρήστης και συγκεκριμένα τον τύπο MIME αυτού. Ο τύπος MIME χρησιμοποιείται από το πρωτόκολλο HTTP, για να περιγράψει τον τύπο ενός αρχείο που μεταδίδονται στο διαδίκτυο. Ουσιαστικά αποτελείται από τον βασικό τύπο και έναν δευτερεύον, που είναι υποκατηγορία του βασικού, διαχωρισμένοι με «/». Στην περίπτωση μας ελέγχουμε αν ο βασικός τύπος είναι «**image**» και εν συνέχεια αν ο δευτερεύον ανήκει στους επιτρεπόμενους τύπους εικόνας που έχουμε ορίσει. Αυτοί είναι «jpg ,jpeg, png, tiff, bmp». Σε περίπτωση που δεν είναι ένας από αυτούς τότε εμφανίζεται σχετικό μήνυμα.

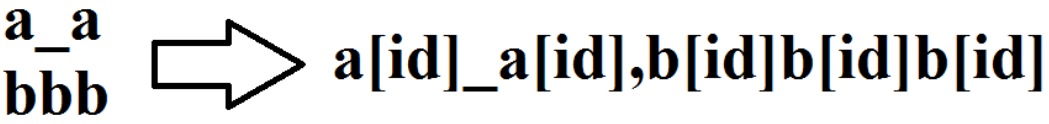
ΕΛΕΓΧΟΣ ΕΙΚΟΝΑΣ

```
//Check for Image Size
$size=$_FILES['preview']['size'];
if($size > 10 * 1048576){
    $size_error = true;

}
//Handle Image
$preview = $_FILES['preview']['tmp_name'];
//Εξαγωγή τύπου εικόνας και κατάληξης
$finfo = finfo_open(FILEINFO_MIME_TYPE);
$mime = finfo_file( $finfo, $_FILES['preview']['tmp_name']);
if(substr( $mime, 0, 5) == 'image'){
    $ext = substr($mime, 6);
    if(!in_array($ext, ['jpg','jpeg','png','tif','bmp'])){
        $file_error = true;
    }
}else{
    $file_error = true;
}
```

Κατόπιν ελέγχεται η ύπαρξη ή όχι του floor plan. Σε περίπτωση που ο χρήστης επέλεξε να ανεβάσει floor plan, τότε χρησιμοποιώντας τις ίδιες συναρτήσεις με πριν, εξάγουμε τον τύπο MIME του αρχείου και ελέγχουμε αν είναι «**text/plain**». Αυτό σημαίνει ότι το αρχείο περιέχει μόνο το κείμενο που εισήγαγε ο χρήστης και όχι meta πληροφορίες για αυτό. Αφού διαβεβαιωθούν τα σωστά format των παραμέτρων, της εικόνας και του floorplan, προχωράμε στην εισαγωγή των δεδομένων στη βάση.

Αρχικά ο server μετατρέπει τις ημερομηνίες που περάστηκαν σε ένα τύπο που είναι κατανοητός από την βάση δεδομένων και για κάθε πολιτική εισιτηρίου αντιστοιχεί το αντίστοιχο αύξον γράμμα, που αναφέρεται στο floor plan. Επειδή γίνονται πολλές εισαγωγές στη βάση, θέλουμε σε περίπτωση που έστω μία από αυτές αποτύχει να τις αναιρέσουμε όλες. Για αυτό το λόγο αρχίζουμε μία νέα συναλλαγή στη βάση δεδομένων στέλνοντας το ερώτημα «**START TRANSACTION**». Έπειτα εισάγουμε τις πληροφορίες για τη διοργάνωση στον πίνακα **events**. Οι πληροφορίες αυτές είναι το όνομα, κατηγορία, η διεύθυνση, ο χώρος, η περιγραφή, ο αριθμός των θέσεων η ύπαρξη ή όχι floor plan, το id του διοργανωτή και η ημερομηνία δημιουργίας. Με την εντολή **mysqli_insert_id($conn)**, λαμβάνουμε το μοναδικό αύξον id της διοργάνωσης που μόλις δημιουργήθηκε. Αφήνουμε το πεδίο "**preview**", που περιέχει το path για την εικόνα κενό καθώς χρειαζόμαστε το προαναφερθέν id, καθώς και το "**floorplan_plan**". Αφού λάβαμε το id σειρά έχει η αποθήκευση της εικόνας στο web server. Όλες οι εικόνες των διοργανώσεων αποθηκεύονται σε ένα φάκελο **"preview"**, με όνομα "evemt*ID*_preview.*EXTENSION*". Όπου ID, το μοναδικό id και EXTENSION η επέκταση του αρχείου. Η μεταφορά του αρχείου στο φάκελο αυτόν γίνεται με την εντολή **move_uploaded_file($preview, $path)**, και σε περίπτωση επιτυχίας της μεταφοράς ανανεώνουμε και το path στη βάση στο πεδίο **preview.** Κατόπιν εισάγονται οι ημερομηνίες και οι πολιτικές εισιτηρίων στους αντίστοιχους πίνακες και αποθηκεύουμε τα id των πολιτικών που μόλις δημιουργήθηκαν, καθώς και το πόσα εισιτήρια και πιο αναγνωριστικό γράμμα αντιστοιχεί σε κάθε πολιτική. Πλέον έχουμε τις απαραίτητες πληροφορίες για να δημιουργήσουμε τις θέσεις. Ανάλογα οι όχι την ύπαρξη floor plan ακολουθείται και ο αντίστοιχος αλγόριθμος:

- **ΥΠΑΡΞΗ FLOOR PLAN:** Στην περίπτωση που υπάρχει floor plan, ο server δημιουργεί τις θέσεις και το floorplan map ανάλογα με το ανεβασμένο αρχείο κειμένου. Γίνεται προσπέλαση του αρχείου αυτού ανά γράμμα (identifier) και για κάθε έγκυρο γράμμα που βρίσκεται εισάγεται στον πίνακα seats η αντίστοιχη θέση. Η εγγραφή της θέσης περιέχει τα στοιχεία της τιμής, του id της πολιτικής, το id της διοργάνωσης που ανήκει και το όνομα της θέσης. Επίσης επαυξάνεται η μεταβλητή String **floorplanString** κατά το γράμμα που διαβάστηκε ακολουθούμενο από αγκύλες που περιέχουν το id της θέσης που μόλις δημιουργήθηκε. Σε περίπτωση κάτω παύλας επαυξάνεται αυτούσια στη μεταβλητή, ενώ αν βρεθεί αλλαγή σειράς τότε εισάγεται κόμμα.



ΕΙΚΟΝΑ 3.2 – ΑΝΤΙΣΤΟΙΧΙΣΗ ΑΡΧΕΙΟΥ ΚΕΙΜΕΝΟΥ ΣΕ FLOORPLAN_MAP

Αφού ολοκληρωθεί η προσπέλαση του αρχείου, ενημερώνεται το πεδίο floorplan_map του πίνακα **events.** Το συγκεκριμένο format με το οποίο αποθηκεύεται το floor plan στη βάση, χρησιμοποιείται από τη βιβλιοθήκη JSC, ώστε να παρουσιάσει τις θέσεις στο χρήστη.

ΔΗΜΙΟΥΡΓΙΑ ΘΕΣΕΩΝ ΜΕ FLOORPLAN

```
//Τα κλειδιά του associative πίνακα identifiers περιέχουν το
αντίστοιχο γράμμα για κάθε πολιτική και οι τιμές αυτών το id της
εκάστοτε πολιτικής (type). Στη μεταβλητή maxLetter αποθηκεύεται το
πλήθος των πολιτικών.
$maxLetter = sizeof($identifiers);
$map = fopen($_FILES['floorplan_map']['tmp_name'],'r');
//Αρχικοποιούμε το τελικό αλφαριθμητικό που θα εισαχθεί στη βάση
$floorplanString = "";
$row = 0;
//Τρέχουμε το αρχείο γραμμή-γραμμή
while(($line = fgets($map))!== false){
    $col=1;
    //Τρέχουμε κάθε γραμμή γράμμα-γράμμα
    for($i=0; $i<strlen($line); $i++){
        //Ελέγχουμε την ASCII τιμή για επιτρεπόμενο γράμμα ή κενό
        if((ord($line[$i])>=97 && ord($line[$i])<97+$maxLetter)
            || $line[$i]=='_'){
                if($line[$i]=='_'){
                    $floorplanString .='_';
                }else{
                    //Η idof μετρατρέπει έναν αθριμό σε αντίστοιχο
                      γράμμα.
                    $ticketName = idof($row).$col;
                    //Εισάγουμε τη θέση στη βάση και επαυξάνουμε
                      το floorplanString με ο αναγνωριστικό της
                      πολιτκής, τις αγγύλες και το id
                    $query="INSERT INTO seats (name,
                            event_id,type) VALUES ('$ticketName',
                            '$eventId','".$identifiers[$line[$i]].
                            "')";
                    if(!mysqli_query($conn, $query)){
                        $db_error = true;
                    }
                    $floorplanString .= $line[$i].'['.
                    mysqli_insert_id($conn).']';
                    $col++;
                }
            }
        }
    $floorplanString.= ',';
    $row++;
}
```

- **ΜΗ ΥΠΑΡΞΗ FLOOR PLAN:** Σε αυτήν την περίπτωση τα εισιτήρια δημιουργούνται βάση του αριθμού που δόθηκε από το χρήστη. Για κάθε πολιτική που πέρασε ο χρήστης εισάγεται στον πίνακα seats το ανάλογο πλήθος θέσεων. Το όνομα της κάθε θέσης είναι ο αντίστοιχος αύξον αριθμός του συνολικού πλήθους εισιτηρίων που δήλωσε ο χρήστης, ανεξάρτητα με την πολιτική.

ΔΗΜΙΟΥΡΓΙΑ ΘΕΣΕΩΝ ΧΩΡΙΣ FLOORPLAN

```
$ticketName = 1;
//Τρέχουμε τον πίνακα types_id που σε κάθε θέση του περιέχει έναν
associative πίνακα με το id της πολιτικής και το πλήθος των
εισιτηρίων που αντιστοιχούν σε αυτή
foreach($types_id as $type){
    $tid = $type['type_id'];
    Τρέχουμε το πλήθος της κάθε πολιτικής και εισάγουμε το
    εκάστοτε εισιτήριο
    for($i=0;$i<$type['tCount'];$i++){
        $query = "INSERT INTO seats (name,event_id,type) VALUES
                          ('$ticketName','$eventId', '$tid')";
        if(!mysqli_query($conn, $query)){
            $db_error = true;
        }
        $ticketName++;
    }
}
```

Αφού ολοκληρωθούν επιτυχώς όλες οι εισαγωγές στη βάση, τότε με την εντολή **mysqli_commit($conn)**, αποθηκεύουμε τις αλλαγές στη βάση. Σε περίπτωση σφάλματος, αναιρούμε όλες τις αλλαγές με την εντολή **mysqli_rollback($conn)** και εμφανίζουμε ανάλογο μήνυμα στο χρήστη.

3.4.5 ΚΡΑΤΗΣΕΙΣ ΘΕΣΕΩΝ

Όταν ο χρήστης επιλέξει μία ημερομηνία μιας διοργάνωσης, τότε ανάλογα με το αν υπάρχει διαθέσιμο floor plan ή όχι, εμφανίζονται και οι αντίστοιχες προβολές. Αρχικά ο server επιστρέφει τα διαθέσιμα εισιτήρια/θέσεις για την επιλεγμένη ημερομηνία. Σε περίπτωση που δεν υπάρχει floor plan, τα διαθέσιμα εισιτήρια υπολογίζονται από την εξαίρεση (**NOT IN**) των κλεισμένων θέσεων (reserved_seats) από το γενικό πίνακα των θέσεων (seats), για κάθε πολιτική εισιτηρίου, για τη συγκεκριμένη ημερομηνία (**happening_id**). Επειδή σε αυτή την περίπτωση μας ενδιαφέρει μόνο το πλήθος και όχι τα συγκεκριμένα id, χρησιμοποιούμε την συνάρτηση **COUNT** ώστε να βρούμε το ζητούμενο πλήθος. Στην περίπτωση που υπάρχει floor plan οι πληροφορίες που χρειάζονται είναι οι κλεισμένες θέσεις και το floorplan map, καθώς το floorplan map περιέχει τα id όλων των θέσεων. Μία θέση θεωρείται κλεισμένη, όταν η κατάσταση της είναι “closed”, ή “open” και δεν έχουν περάσει 10 λεπτά από την έκδοση της. Ο έλεγχος αυτός γίνεται με τη χρήση της συνάρτησης **DATE_ADD** που προσφέρει η

βάση δεδομένων. Η συνάρτηση αυτή δέχεται μία ημερομηνία και ένα χρονικό διάστημα. Στην περίπτωση μας:

DATE_ADD(reserved_at, INTERVAL +10 MINUTE) > NOW().

Ελέγχουμε, δηλαδή, το αν προσθέτοντας δέκα λεπτά από την ημερομηνία έκδοσης της (reserved_at) ξεπερνάει την τωρινή ώρα. Αν την ξεπερνάει και η κατάσταση της είναι "open", τότε η θέση θεωρείται προσωρινά κλεισμένη και σε συνδυασμό με τον έλεγχο για την κατάσταση "closed", ανεξαρτήτου το πότε κλείστηκε, βρίσκουμε τις κλεισμένες θέσεις.

ΕΡΩΤΗΜΑ ΓΙΑ ΤΗΝ ΕΥΡΕΣΗ ΕΙΣΙΤΗΡΙΩΝ ΧΩΡΙΣ FLOORPLAN

```
//Για κάθε τύπο πολιτικής ($row['id']) τρέχουμε το παρακάτω ερώτημα μέσω
  PHP για τη συγκεκριμένη ημερομηνία ($hapId). Αρχικά βρίσκει όλα τα id της
συγκεκριμένης πολιτικής και τα εξαιρούμε από τα id που έχουν κλειστεί
"SELECT COUNT(id) AS available
FROM seats
WHERE type=".$row['id']." AND id NOT IN
(SELECT seat_id
 FROM reserved_seats
 WHERE happening_id =".$hapId." AND (status='closed' OR (status='open' AND
 DATE_ADD(reserved_at,INTERVAL +12 MINUTE) > NOW() ) )
)"
```

Μόλις ο χρήστης επιλέξει τα εισιτήρια ή τις θέσεις που θέλει, ο server αρχικά ελέγχει αν υπάρχουν έγκυρες ανοιχτές κρατήσεις στο όνομα του. Σε περίπτωση που βρεθούν, τότε εμφανίζεται σχετικό μήνυμα και δεν επιτρέπεται στο χρήστη να προχωρήσει σε νέα κράτηση.

- **ΚΡΑΤΗΣΗ ΜΕ FLOORPLAN:** σε αυτήν την περίπτωση μέσω της HTML φόρμας περνάνε στον server τα ids των θέσεων που θέλει να κλείσει ο χρήστης. Αρχικά εισάγουμε στον πίνακα **reservations** μία νέα κράτηση με το id του χρήστη. Για να διαβεβαιώσουμε ότι δεν θα γίνει διπλή κράτηση της ίδιας θέσης, χρησιμοποιούμε την εντολή **FOR UPDATE** που προσφέρει η InnoDB ώστε να κλειδώσουμε τις θέσεις που επέλεξε ο χρήστης. Προσθέτοντας την εντολή **FOR UPDATE** στο τέλος ενός ερωτήματος **SELECT**, οι επιλεγμένες εγγραφές κλειδώνονται και δεν μπορούν να διαβαστούν από άλλη συνεδρία. Έτσι διαβεβαιώνουμε ότι οι θέσεις αυτές δεν θα επιλεχτούν από άλλο χρήστη. Για να χρησιμοποιήσουμε την εντολή **FOR UPDATE** το κλείδωμα πρέπει να γίνει μέσα σε μία συναλλαγή SQL. Οι κλειδωμένες εγγραφές απελευθερώνονται όταν η συνεδρία γίνει commit ή rollback. Αυτό σημαίνει ότι όταν γίνει η κράτηση των θέσεων αυτών, η συνεδρία που περιμένει να τις διαβάσει δεν θα πρέπει να είναι σε θέση να τις εντοπίσει. Αυτό το επιτυγχάνουμε με τη χρήση της εντολής **NOT IN**. Οι εγγραφές με τα id των θέσεων που κλειδώνονται γίνονται **SELECT… FOR UPDATE** με τη συνθήκη ότι δεν βρίσκονται (**NOT IN**) στις έγκυρες κλεισμένες θέσεις του πίνακα **reserved_seats**. Έτσι όταν τελειώσει η

συνεδρία και κλείσει τις επιλεγμένες θέσεις, θα τις εισάγει στον πίνακα αυτόν, επομένως η επόμενη συνεδρία που περιμένει να διαβάσει τις θέσεις αυτές δεν θα τις βρει, καθώς η συνθήκη **NOT IN** δεν θα ικανοποιείται, και θα κλειδώσει μόνο τις ελεύθερες.

ΨΕΥΔΟΚΩΔΙΚΑΣ ΕΡΩΤΗΜΑΤΟΣ:

```
SELECT id
FROM seats
WHERE id IN (chosenIds) AND NOT IN
 (SELECT id FROM reserved_seats WHERE (“chosenHappeningId” AND
  “closedSeat”)
 )
FOR UPDATE
```

Αφού επιστρέψει το ερώτημα **SELECT… FOR UPDATE**, ελέγχουμε αν ο αριθμός των θέσεων που επιστράφηκαν είναι ίδιος με το συνολικό αριθμό θέσεων που ζητήθηκε, σε περίπτωση που δεν είναι σημαίνει ότι απέτυχε να διαβαστεί κάποια θέση, διότι κρατήθηκε από κάποιον άλλον χρήστη. Τότε γίνεται rollback και εμφανίζεται ανάλογο μήνυμα. Στην περίπτωση που είναι ίδιος, το οποίο σημαίνει ότι οι επιλεγμένες θέσεις κλειδώθηκαν, εισάγει όλα τα ids στον πίνακα **reserved_seats**, ανανεώνει το κόστος του reservation που δημιουργήθηκε αρχικά και κάνει commit αποθηκεύοντας τις αλλαγές.

- **ΚΡΑΤΗΣΗ ΧΩΡΙΣ FLOORPLAN:** στην περίπτωση αυτήν ο χρήστης εισάγει τον αριθμό των εισιτηρίων που θέλει για κάθε πολιτική εισιτηρίου. Τα ids των εισιτηρίων δεν είναι προκαθορισμένα όπως πριν, αλλά είναι δουλειά του server να βρει και να κρατήσει το πλήθος των ζητούμενων εισιτηρίων. Αρχικά εισάγεται μία εγγραφή στον πίνακα **reservations**, με το id του χρήστη. Όπως και πριν χρησιμοποιούμε την εντολή **SELECT… FOR UPDATE** για να κλειδώσουμε τα εισιτήρια που επιλέγονται, ώστε να μην είναι προσβάσιμα από άλλες συνεδρίες. Για κάθε πολιτική που ζήτησε ο χρήστης, ο server δημιουργεί μία νέα συναλλαγή κλειδώνοντας τόσες εγγραφές όσες και το αντίστοιχο πλήθος που ζητήθηκε. Αυτό επιτυγχάνεται με την εντολή **LIMIT.** Χρησιμοποιώντας τον περιορισμό **NOT IN** για να ελέγξουμε ότι τα εισιτήρια δεν είναι στον πίνακα **reserved_seats**, αποτρέπουμε τις συνεδρίες που περιμένουν να διαβάσουν τις κλειδωμένες θέσεις από το να έχουν πρόσβαση σε αυτές, αφότου ολοκληρωθεί η κράτηση τους.

ΨΕΥΔΟΚΩΔΙΚΑΣ ΕΡΩΤΗΜΑΤΟΣ:

```
SELECT id
FROM seats
WHERE type=”typeAsked” AND id NOT IN
(SELECT id
 FROM reserved_seats
 WHERE (‘chosenHappeningId’ AND ‘closedSeat’)
)
LIMIT numberOfTicketsAsked FOR UPDATE
```

Ουσιαστικά ο server βρίσκει και κλειδώνει τα πρώτα **N** εισιτήρια που θα βρει με το ζητούμενο τύπο, με **N** το ζητούμενο πλήθος από το χρήστη. Αν μία άλλη συνεδρία ζητήσει τον ίδιο τύπο εισιτηρίων, τότε θα περιμένει να πάρει πρόσβαση στις κλειδωμένες εγγραφές. Όταν απελευθερωθούν αυτές οι εγγραφές, θα έχουν εισαχθεί στον πίνακα **reserved_seats**, οπότε η συνθήκη **NOT IN** δεν θα ικανοποιείται, με αποτέλεσμα να κλειδώσει τα επόμενα **N** διαθέσιμα. Αφού επιστρέψει το ερώτημα **SELECT… FOR UPDATE**, ελέγχεται αν το πλήθος των εγγραφών που επιλέχθηκε είναι ίσο με το πλήθος που ζητήθηκε. Σε περίπτωση που είναι, οι θέσεις εισάγονται στον πίνακα **reserved_seats**, γίνεται commit η συναλλαγή και προχωράει στην επόμενη πολιτική που ζητήθηκε. Η αντίθετη περίπτωση σημαίνει ότι τα εισιτήρια που ζητήθηκαν δεν είναι διαθέσιμα και τότε γίνεται rollback η συγκεκριμένη συναλλαγή και διαγράφεται η αρχική κράτηση που δημιουργήθηκε. Καθώς οι θέσεις που αποθηκεύονται στον πίνακα **reserved_seats** αποθηκεύουν με σχέση ξένου κλειδιού το id της κράτησης που δημιουργήθηκε αρχικά, με τη διαγραφή της κράτησης αυτής από τον πίνακα **reservations**, διαγράφονται επίσης και όσες θέσεις διαφορετικού τύπου κλείστηκαν για την ίδια κράτηση από προηγούμενες συναλλαγές. Αφού δεν υπάρχει κάποιο σφάλμα και εισαχθούν όλες οι ζητούμενες θέσεις στον πίνακα **reserved_seats**, ανανεώνεται το κόστος στην αρχική κράτηση που δημιουργήθηκε και ολοκληρώνεται η προσωρινή κράτηση.

Όλες οι θέσεις που εισάγονται στον πίνακα **reserved_seats**, καθώς και η κράτηση του πίνακα **reservations** έχουν αρχικά κατάσταση "open", δηλαδή προσωρινά κρατημένη για 10 λεπτά. Όταν ο χρήστης θελήσει να οριστικοποιήσει την κράτηση αυτή, τότε ανανεώνεται η κατάσταση τους σε "closed" και ολοκληρώνεται η κράτηση. Το μόνο δεδομένο που χρειάζεται είναι ο αριθμός της κράτησης, καθώς οι κρατημένες θέσεις συνδέονται με αυτήν μέσω του id της. Σε περίπτωση που ο χρήστης ακυρώσει την κράτηση ή διαγραφεί από την ρουτίνα της βάσης, τότε διαγράφονται μαζί της και οι εγγραφές από τον πίνακα **reserved_seats.**

3.5 ΑΝΑΛΥΣΗ WEB SERVICE

3.5.1 PHP WEB SERVICES

Για την ανάπτυξη του web service χρησιμοποιήθηκε η server-side γλώσσα προγραμματισμού PHP. Η PHP η οποία η οποία σχεδιάστηκε κυρίως για την ανάπτυξη εφαρμογών διαδικτύου, παρέχει τις απαραίτητες βιβλιοθήκες και μεθόδους ώστε να αναπτύξουμε ένα web service. Σε συνδυασμό με ορισμένες λειτουργίες του Apache Server καταφέρνουμε να υλοποιήσουμε την αρχιτεκτονική REST. Το πλεονέκτημα με την PHP είναι ότι τρέχει ήδη πάνω στο web server μας και μπορεί να

διαχειριστεί τις αιτήσεις HTTP, που χρειάζονται για τη λειτουργία του web service, με τη χρήση απλών εντολών. Όλες οι αιτήσεις που γίνονται από τους χρήστες δρομολογούνται στο κατάλληλο **.php** αρχείο, το οποίο είναι υπεύθυνο για την εξυπηρέτηση του συγκεκριμένου αιτήματος. Οι απαντήσεις που παρέχει ο server, χρησιμοποιούν τις κατάλληλες κεφαλίδες (headers), ώστε να ενημερώσουν το χρήστη για την κατάληξη της αίτησης τους. Οι κεφαλίδες αυτές περιλαμβάνουν τον κατάλληλο κωδικό HTTP, καθώς και το μήνυμα που το συνοδεύει. Επίσης χρησιμοποιώντας το πρότυπο JSON, στέλνονται οποιαδήποτε δεδομένα μπορεί να σχετίζονται με την αίτηση του χρήστη. Όλες οι λειτουργίες που προσφέρει το web service είναι αντίστοιχες με αυτές της ιστοσελίδας και ο server τις διεκπερώνει με τους ίδιους τρόπους που αναλύθηκαν προηγουμένως στο κεφάλαιο αυτό. Στις παραγράφους που ακολουθούν επικεντρώνουμε στα χαρακτηριστικά του web service και τις λειτουργίες που χρησιμοποιεί, για καταστεί δυνατή η επικοινωνία.

3.5.2 ΠΡΟΤΥΠΟ JSON

Όλα τα δεδομένα της βάσης που επιστρέφονται στο χρήστη γίνονται με τη χρήση του προτύπου JSON. Το JSON (JavaScript Object Notation) είναι ένα ελαφρύ πρότυπο ανταλλαγής δεδομένων. Είναι εύκολο για τους ανθρώπους να το διαβάσουν και γράψουν. Είναι εύκολο για τις μηχανές να το αναλύσουν (parse) και να το παράγουν (generate). Το JSON είναι ένα πρότυπο κειμένου το οποίο είναι τελείως ανεξάρτητο από γλώσσες προγραμματισμού αλλά χρησιμοποιεί πρακτικές (conventions) οι οποίες είναι γνωστές στους προγραμματιστές της οικογένειας προγραμματισμού C. Αυτές οι ιδιότητες κάνουν το JSON μια ιδανική γλώσσα προγραμματισμού ανταλλαγής δεδομένων. Το JSON είναι χτισμένο σε δύο δομές:

- Μια συλλογή από ζευγάρια ονομάτων/τιμών. Σε διάφορες γλώσσες προγραμματισμού, αυτό αντιλαμβάνεται ως ένα object, καταχώριση, δομή, λεξικό, πίνακα hash (hash table), λίστα κλειδιών, ή associative πίνακα.
- Μία ταξινομημένη λίστα τιμών. Στις περισσότερες γλώσσες προγραμματισμού, αυτό αντιλαμβάνεται ως ένας πίνακας (array), διάνυσμα, λίστα, ή ακολουθία.

Πηγή (*http://www.json.org*)

Για παράδειγμα όταν ένας χρήστης στέλνει μία αίτηση GET, ώστε να λάβει πληροφορίες για τις κρατήσεις του, τότε τα δεδομένα αυτά στέλνονται ως ένα αντικείμενο JSON, του οποίου το όνομα είναι “reservations” και η τιμή ένας πίνακας JSON. Κάθε στοιχείο του πίνακα είναι μία συλλογή από JSON αντικείμενα τα οποία περιέχουν πληροφορίες για την κάθε κράτηση.

ΕΙΚΟΝΑ 3.3 – ΑΝΤΙΚΕΙΜΕΝΟ JSON ΜΕ ΠΛΗΡΟΦΟΡΙΕΣ ΓΙΑ ΤΙΣ ΚΡΑΤΗΣΕΙΣ ΤΟΥ ΧΡΗΣΤΗ

Για να δημιουργήσουμε ένα αντικείμενο JSON με την PHP, πρέπει να δομήσουμε τα δεδομένα μέσα σε ένα associative πίνακα (πίνακας του οποίου οι δείκτες θέσεων είναι κείμενο και όχι αριθμοί). Για κάθε κράτηση που έχει ο χρήστης δημιουργείται ένας associative πίνακας, που περιέχει τις αντίστοιχες πληροφορίες σε κάθε πεδίο του Για παράδειγμα:

```
$jsonAssoc['id'] = $resId;
$jsonAssoc['title'] = $eventTitle;
```

Όλοι αυτοί οι πίνακες εισάγονται σε έναν άλλον πίνακα ($json), όπου η κάθε θέση του περιέχει έναν τέτοιον associative πίνακα, με τις πληροφορίες τις κάθε κράτησης.

```
array_push($json, $jsonAssoc);
```

Ο τελικός αυτός πίνακας εισάγεται στην τιμή του πεδίου "reservations" ενός άλλου associative πίνακα που τα περιέχει όλα και αποτελεί το τελικό JSON αντικείμενο που θα σταλεί.

```
$jsonFinal = array("reservations"=> $json);
```

Έπειτα θέτουμε την κατάλληλη κεφαλίδα, που περιγράφει τον τύπο δεδομένων που θα σταλούν, με την εντολή:

```
header('Content-Type: application/json');
```

Τέλος κωδικοποιούμε τον τελικό associative πίνακα που φτιάξαμε ως αντικείμενο JSON, με τη χρήση της συνάρτησης **json_encode** και το στέλνουμε στο χρήστη:

```
echo json_encode($jsonFinal);
```

3.5.3 ΔΡΟΜΟΛΟΓΗΣΗ ΑΙΤΗΣΕΩΝ

Όπως είδαμε στο κεφάλαιο 2, οι αιτήσεις στέλνονται σε ορισμένα URL, τα οποία ανάλογα με τη μέθοδο HTTP που χρησιμοποιήθηκε εκτελούν και μία ανάλογη λειτουργία. Τα URL αυτά, δεν αντιστοιχούν σε κάποιο φυσικό αρχείο στο server. Η ονομασία τους γίνεται στα πλαίσια των περιορισμών της αρχιτεκτονικής REST, ώστε να καταλαβαίνει ο χρήστης σε ποιο resource αναφέρεται και τι προσφέρει το κάθε URL μόνο από το όνομα. Όταν γίνεται μία αίτηση σε ένα συγκεκριμένο URL, τότε αυτό δρομολογείται στο φυσικό αρχείο .php, που είναι υπεύθυνο για την εξυπηρέτηση της εκάστοτε αίτησης.

Για να εξυπηρετήσει ο server την εισερχόμενη αίτηση ελέγχεται το URL, το οποίο ανεξάρτητα τη μέθοδο HTTP που χρησιμοποιήθηκε δρομολογείται σε ένα αρχείο PHP. Αυτό επιτυγχάνεται με τη χρήση του **mod_rewrite** module του apache server. Το **mod_rewrite** μας επιτρέπει, μέσω ενός συνόλου κανόνων που βασίζονται σε regular expressions, να «ξαναγράψουμε» τα URL που φτάνουν στο server. Ουσιαστικά αντιστοιχεί ένα URL σε κάποιο path ενός αρχείου που βρίσκεται στο σύστημα. Αυτό μας επιτρέπει να χρησιμοποιούμε πιο εύμορφα URLs, καθώς και να κρύβουμε τα paths των αρχείων του συστήματος μας. Το **mod_rewrite** ελέγχει κάποιες συνθήκες αναγραφής για τα URL που έρχονται στο server, οι οποίες όταν ικανοποιούνται εφαρμόζονται οι κανόνες αναγραφής πάνω στα URL αυτά. Οι κανόνες και οι συνθήκες αυτές βρίσκονται μέσα σε ένα αρχείο **.htaccess** που βρίσκεται στο web server. Οι συνθήκες που ελέγχονται είναι το να μην αντιστοιχούν τα URL σε κάποιο φυσικό φάκελο ή αρχείο στο server. Στην περίπτωση που δεν αντιστοιχούν τότε εφαρμόζουμε τους κανόνες αναγραφής, οι οποίοι δρομολογούν την αίτηση στο επιθυμητό αρχείο. Για παράδειγμα, όταν ο χρήστης στέλνει μία αίτηση GET στο "/events/2", τότε ο server αφού αρχικά ελέγξει τη συνθήκη αναγραφής, ότι δηλαδή δεν υπάρχει έγκυρο path "/events/2", δρομολογεί την αίτηση αυτήν στο υπάρχον αρχείο "getPostDeleteEvent.php". Αυτό γίνεται με τον κανόνα αναγραφής:

RewriteRule ^events/([0-9]+)?$ getPostDeleteEvent.php\?event_id=$1

Ακολουθώντας την εντολή RewriteRule βρίσκουμε το URL το οποίο πρόκειται να αναγραφεί. Στην συγκεκριμένη περίπτωση "events/ΕκφρασηΑριθμού". Έπειτα βρίσκουμε το url με το οποίο θα αντικατασταθεί, δηλαδή το "getPostDeleteEvent.php\?event_id=*ΕκφρασηΑριθμού*"
Προφανώς το αρχείο **getPostDeleteEvent.php** υπάρχει στο server και από το όνομα του καταλαβαίνουμε ότι διαχειρίζεται συγκεκριμένες μεθόδους HTTP. Επίσης το id της διοργάνωσης που περιλάμβανε ο χρήστης στο URL που κάλεσε, μεταφέρεται στο νέο αρχείο ως μία παράμετρος GET και μπορεί να χρησιμοποιηθεί από αυτό μέσω του πίνακα **$_GET** της PHP. Η όλη διαδικασία αποκρύπτεται από το χρήστη, ο

οποίος το μόνο που βλέπει και χρειάζεται να ξέρει είναι το αρχικό URL. Κατ'αναλογία, για κάθε URL που προσφέρεται από το web service, υπάρχει και το αντίστοιχο αρχείο στο server που το εξυπηρετεί.

3.5.4 ΕΠΕΞΕΡΓΑΣΙΑ ΑΙΤΗΣΕΩΝ/ JWT

Αφού ολοκληρωθεί η δρομολόγηση της αίτησης στο κατάλληλο php αρχείο, ελέγχεται η μέθοδος HTTP. Κάθε αρχείο που αντιστοιχεί σε ένα URL, μπορεί να εξυπηρετήσει ένα πλήθος μεθόδων που ορίζονται από το web service. Στη μεταβλητή **$_SERVER['REQUEST METHOD']** της PHP, βρίσκεται αποθηκευμένη η μέθοδος με την οποία έγινε η αίτηση. Ελέγχοντας, λοιπόν, αυτή τη μεταβλητή τρέχουμε και το αντίστοιχο κομμάτι κώδικα. Σε περίπτωση που δεν υποστηρίζεται η μέθοδος, ο server απαντάει με την HTTP κεφαλίδα «**405 Method Not Allowed**».

Στο επόμενο βήμα γίνεται ο έλεγχος της εξουσιοδότησης του χρήστη. Για τη χρήση του web service, ο χρήστης πρέπει να φέρει το κατάλληλο Authorization Token, το οποίο αποκτά με την είσοδο του στο σύστημα και έχει διάρκεια ζωής 1 ώρας. Το Token στέλνεται από το χρήστη μέσω της κεφαλίδας Authorization και το σχήμα Bearer. Η PHP μας προσφέρει πρόσβαση στις κεφαλίδες του χρήστη, μέσω της μεταβλητής **$_SERVER[HTTP_*HEADERNAME*]**. Οπότε μέσω της μεταβλητής **$_SERVER[HTTP_AUTHORIZATION]** εξάγουμε το Token του χρήστη.

Για την έκδοση και τον έλεγχο των token χρησιμοποιήθηκε η βιβλιοθήκη **JWT (JSON Web Token)**. Με τις συναρτήσεις που μας προσφέρει η βιβλιοθήκη αυτή, μπορούμε να εκδώσουμε ένα υπογεγραμμένο JWT και να αποθηκεύσουμε σε αυτό πληροφορίες, όπως το id του χρήστη, την ιδιότητα του και το username του. Έχοντας αυτές τις πληροφορίες αποθηκευμένες στο Token που στέλνει ο χρήστης, παίρνουμε τις απαραίτητες πληροφορίες για αυτόν χωρίς να στείλουμε ερώτημα στη βάση. Επίσης ικανοποιείται ο περιορισμός stateless της αρχιτεκτονικής REST, ότι δηλαδή τα δεδομένα της αίτησης θα πρέπει να είναι αρκετά ώστε να μπορεί να εξυπηρετηθεί από το server. Όταν ένας χρήστης ζητήσει την έκδοση ενός token, μέσα από το url: 'users/token', ο server αφού τον επαληθεύσει του επιστρέφει ένα JWT, μέσω της κεφαλίδας Token. Σε περίπτωση που αποτύχει η επαλήθευση του, στέλνει την κεφαλίδα HTTP «**401 Unauthorized**». Στο JWT βρίσκονται κωδικοποιημένες οι εξής πληροφορίες σε μορφή αντικειμένου JSON:

- Ημερομηνία έκδοσης
- Μοναδικό ID του Token, το οποίο υπολογίζεται από μία συνάρτηση hash
- Όνομα εκδότη
- Ημερομηνία λήξης
- Δεδομένα χρήστη, του οποίου η τιμή είναι ένας πίνακας JSON που περιέχει το id, username και την ιδιότητα του χρήστη

Ένα JWT έχει τη μορφή: **xxxxx.yyyyy.zzzzz**.Το πρώτος μέρος περιέχει την Base64 κωδικοποίηση της κεφαλής του JWT, η οποία είναι ένα JSON αντικείμενο που περιέχει πληροφορίες για τον αλγόριθμο κρυπτογράφησης (HMAC SHA256) και τον τύπο του Token (JWT). Στο δεύτερο σκέλος περιέχεται η Base64 κωδικοποίηση του JSON αντικειμένου, που περιέχει τις πληροφορίες για το JWT που προαναφέρθηκαν. Το τρίτο σκέλος περιέχει την υπογραφή, ώστε να εξασφαλίζεται η εγκυρότητα και η αξιοπιστία του Token που στέλνει ο χρήστης. Ουσιαστικά πρόκειται για την υπογραφή του πρώτου και του δεύτερου σκέλους, με τη χρήση ενός κλειδιού που γνωρίζει μόνο ο server, με τον αλγόριθμο HMAC SHA256. Με τη χρήση της συνάρτησης "**encode**" της βιβλιοθήκης, η οποία έχει ως ορίσματα τα δεδομένα του Token και το κλειδί του server, εκδίδεται το υπογεγραμμένο JWT. Με τη συνάρτηση "**decode**", η οποία έχει ορίσματα το JWT και το κλειδί με το οποίο υπογράφτηκε το Token, ο server ελέγχει αν το JWT είναι έγκυρο και αποκτά πρόσβαση στα δεδομένα χρήστη που είναι αποθηκευμένα, συνεχίζοντας με τη διεκπαιρέωση του αιτήματος. Σε περίπτωση που το JWT έχει λήξει ή αποτύχει η αποκρυπτογράφηση, γιατί πειράχτηκαν τα δεδομένα που είχε εκδώσει ο server, το token θεωρείται άκυρο και επιστρέφεται στο χρήστη η κεφαλίδα HTTP «**440 Token Expired**».

ΣΥΝΑΡΤΗΣΗ ΕΚΔΟΣΗΣ TOKEN:

```
function issueToken($userId, $username, $status){
    $tokenId = md5(time()+rand(0,10000));
    $issuedAt = time();
    //Διάρκεια ζωής μίας ώρας
    $expire = $issuedAt + 60*60;
    $serverName = "serverName";
    //Εισαγωγή δεδομένων του Token
    $data=[
                //Metadata
               'iat' => $issuedAt,
               'jti' => $tokenId,
               'iss' => $serverName,
               'exp' => $expire,
               'data'=> [
                //Δεδομένα Χρήστη
                            'userId' => $userId,
                            'username' => $username,
                            'status' => $status
               ]
    ];
    //Υπογραφή Token και επιστροφή
    return JWT::encode($data,'customKey');
}
```

ΣΥΝΑΡΤΗΣΗ ΕΠΑΛΗΘΕΥΣΗΣ TOKEN:

```
function validate($token){
    $key = 'customKey';
    try{
        //Αποκρυπτογράφηση Token και σε περίπτωση επιτυχίας επιστροφή
```

```
            δεδομένων χρήστη
            $decoded = JWT::decode($token, $key, array('HS256'));
            return $decoded->data;
            //Σε περίπτωση αποτυχίας επιστροφή NULL
        }catch(Exception $e){
            return null;
        }
}
```

Αφού ολοκληρωθεί η επαλήθευση του χρήστη, ο server εξυπηρετεί την αίτηση που έλαβε και ανάλογα με το αποτέλεσμά της στέλνει ως απάντηση μία κεφαλίδα HTTP, η οποία περιέχει ένα status code και ένα μήνυμα. Επιπλέον μαζί με την κεφαλίδα αυτή μπορούν να σταλούν και δεδομένα στο χρήστη, όπως πχ οι διοργανώσεις, οι κρατήσεις του κ.ά, καθώς και άλλες κεφαλίδες όπως ο τύπος των δεδομένων της απάντησης, caching πληροφορίες κ.ά. Τα status code είναι προκαθορισμένα από το HTTP/1.1 πρότυπο και μπορούμε να τα στείλουμε χρησιμοποιώντας την εντολή της PHP: header(HTTP/1.1 *STATUS_CODE STATUS_MESSAGE*). Στην περίπτωση που δεν καθορίζεται κάποιο status code και η αίτηση είναι επιτυχής, στέλνεται η κεφαλίδα «**200 OK**».

4ο ΚΕΦΑΛΑΙΟ

ΕΦΑΡΜΟΓΕΣ ANDROID

ΕΙΚΟΝΑ 4.1 – ΤΟ ΛΟΓΟΤΥΠΟ ΤΟΥ ANDROID

4.1 ΕΙΣΑΓΩΓΗ

Στα πλαίσια της εργασίας αυτής αναπτύχθηκαν δύο native εφαρμογές android, οι οποίες παρέχουν ανάλογες λειτουργίες με την ιστοσελίδα. Αυτό επιτυγχάνεται μέσω του web service, το οποίο αναλύθηκε στο προηγούμενο κεφάλαιο. Οι εφαρμογές android καταναλώνουν τις υπηρεσίες που προσφέρει το web service, ώστε να προσφέρουν στο χρήστη τη δυνατότητα να κάνει κρατήσεις, αλλά και να δημιουργήσει τις δικές του διοργανώσεις. Ουσιαστικά παρέχουν ένα περιβάλλον χρήστη μέσω του οποίου γίνεται η επικοινωνία με την υπηρεσία. Οι λειτουργίες που προσφέρονται για τους διοργανωτές και τους απλούς χρήστες χωρίζονται διακριτά στις δύο εφαρμογές. Έτσι η πρώτη εφαρμογή απευθύνεται στους χρήστες που θέλουν να κάνουν κρατήσεις, καθώς και να διαχειριστούν αυτές, ενώ η δεύτερη απευθύνεται στους διοργανωτές οι οποίοι έχουν τη δυνατότητα της δημιουργίας και διαχείρισης των διοργανώσεων τους. Σε αυτό το κεφάλαιο θα δοθεί σημασία στα τεχνικά χαρακτηριστικά, τους τρόπους ανάπτυξης και επικοινωνίας των εφαρμογών αυτών.

4.2 ΠΛΑΤΦΟΡΜΑ ΑΝΑΠΤΥΞΗΣ

Για την ανάπτυξη των εφαρμογών χρησιμοποιήθηκε το Android Studio. Το Android Studio είναι ένα ολοκληρωμένο προγραμματιστικό περιβάλλον (IDE) για ανάπτυξη εφαρμογών στην πλατφόρμα Android. Προσφέρει στο χρήστη ένα φιλικό περιβάλλον διευκολύνοντας την εισαγωγή και την ανάπτυξη των βασικών στοιχείων

από τα οποία αποτελείται μία εφαρμογή Android, καθώς και εργαλεία για τον εντοπισμό και τη διόρθωση σφαλμάτων. Επίσης παρέχει τη δυνατότητα δημιουργίας εικονικών συσκευών, οι οποίες είναι αντίστοιχες με αυτές που υπάρχουν στο εμπόριο, στις οποίες μπορούμε να επιλέξουμε οποιαδήποτε έκδοση Android θέλουμε και να δοκιμάσουμε την εφαρμογή μας πάνω σε αυτές. Η γλώσσα προγραμματισμού που χρησιμοποιείται από το Android Studio και γενικά από τις εφαρμογές Android είναι η JAVA. Μαζί με την πλατφόρμα βρίσκουμε και το Android SDK που μας παρέχει διάφορες βιβλιοθήκες που διευκολύνουν την ανάπτυξη των εφαρμογών,.

4.3 ΒΑΣΙΚΑ ΔΟΜΙΚΑ ΣΤΟΙΧΕΙΑ

Τα βασικά συστατικά μιας Android εφαρμογής είναι τα εξής:

- Δραστηριότητα (Activity)
- Υπηρεσία (Service)
- Πάροχος Περιεχομένου (Content Provider)
- Δέκτες Μετάδοσης (Broadcast Receiver)

Παρακάτω θα περιγράψουμε συνοπτικά αυτά τα συστατικά και τη βασική λειτουργεία του καθενός

4.3.1 ΔΡΑΣΤΗΡΙΟΤΗΤΑ (ACTIVITY)

Δραστηριότητα είναι μια οθόνη διεπαφής χρήστη σε μια εφαρμογή Android όπου τα οπτικά στοιχεία που ονομάζονται Προβολές(Views), επίσης γνωστά ως widgets, μπορούν να τοποθετηθούν μέσα σε αυτή και ο χρήστης μπορεί να εκτελέσει διάφορες ενέργειες αλληλοεπιδρώντας με αυτά. Τα widgets σε μια δραστηριότητα μπορούν να δημιουργηθούν με δύο διαφορετικούς τρόπους. Ο πρώτος τρόπος είναι με κώδικα Java και ο άλλος με τη προσθήκη έτοιμου κώδικα XML (Layout) το οποίο καθορίζει τη διεπαφή χρήστη. Σχεδόν πάντα προτιμάται ο δεύτερος τρόπος. Μια εφαρμογή μπορεί να έχει περισσότερες από μία δραστηριότητες οι οποίες λειτουργούν ανεξάρτητα η μία από την άλλη, αλλά μπορεί να υπάρχει σύνδεση μεταξύ τους.

4.3.2 ΥΠΗΡΕΣΙΑ (SERVICE)

Μια υπηρεσία είναι ένα συστατικό της εφαρμογή Android που τρέχει στο παρασκήνιο και δεν έχει καμία οπτική διεπαφή χρήστη. Οι υπηρεσίες χρησιμοποιούνται για την εκτέλεση των τμημάτων επεξεργασίας της εφαρμογή στο παρασκήνιο. Ενώ ο χρήστης εργάζεται στο προσκήνιο, οι υπηρεσίες μπορούν να χρησιμοποιηθούν για να χειριστούν τις διαδικασίες που πρέπει να γίνουν στο παρασκήνιο. Μια υπηρεσία μπορεί να ξεκινήσει από ένα άλλο συστατικό της εφαρμογής Android, όπως μια δραστηριότητα ή άλλες υπηρεσίες, και θα συνεχίσει να εκτελείται στο παρασκήνιο ακόμα και μετά την αλλαγή σε μια άλλη εφαρμογή από το

χρήστη. Έτσι, οι υπηρεσίες είναι λιγότερο πιθανό να καταστραφούν από το σύστημα για την εξοικονόμηση πόρων απ' ότι στις δραστηριότητες.

4.3.3 ΠΑΡΟΧΟΣ ΠΕΡΙΕΧΟΜΕΝΟΥ (CONTENT PROVIDER)

Οι πάροχοι περιεχομένου στο Android παρέχουν έναν ευέλικτο τρόπο για να κάνουν διαθέσιμα τα δεδομένα σε όλες τις εφαρμογές. Μέσω των παρόχων περιεχομένου οι εφαρμογές μπορούν να αναζητήσουν ή ακόμα και να τροποποιήσουν τα δεδομένα που έχετε δημιουργήσει, όσο ο πάροχος περιεχομένου σας το επιτρέπει. Το Android έρχεται με πολλούς προ εγκατεστημένους παρόχους περιεχομένου που μπορούμε να χρησιμοποιήσουμε στην εφαρμογή μας.

4.3.4 ΔΕΚΤΗΣ ΜΕΤΑΔΟΣΗΣ (BROADCAST RECEIVER)

Οι δέκτες μετάδοσης είναι ένα από τα συστατικά της εφαρμογής Android που χρησιμοποιείται για τη λήψη μηνυμάτων που μεταδίδονται από το σύστημα Android ή άλλων εφαρμογών του. Υπάρχουν πολλές εκπομπές που προκαλούνται από το ίδιο το σύστημα Android και οι άλλες εφαρμογές μπορούν να τις λάβουν με τη χρήση του δέκτη μετάδοσης. Παραδείγματα των εκπομπών αυτών είναι:

- Προειδοποίηση ότι τελειώνει η μπαταρία
- Κλείσιμο οθόνης
- Αλλαγή ζώνη ώρας
- Ειδοποίηση κάμερας για λήψη φωτογραφίας

Στο προγραμματισμό μπορούμε να χρησιμοποιήσουμε τους δέκτες μετάδοσης για να λάβουμε αυτά τα μηνύματα και να συμπεριφερθούμε ανάλογα. Οι εφαρμογές μπορούν επίσης να προκαλέσουν εκπομπές. Μπορούμε να προκαλέσουμε όσες εκπομπές θέλουμε και δεν υπάρχει κανένα όριο για το πόσες εκπομπές μπορεί να προκληθούν την ίδια στιγμή.

Πηγή (*Yao Yumin, Liu Weiguo 2008. Study of Android's Architecture and its Application Development, Compute Systems & Applications*)

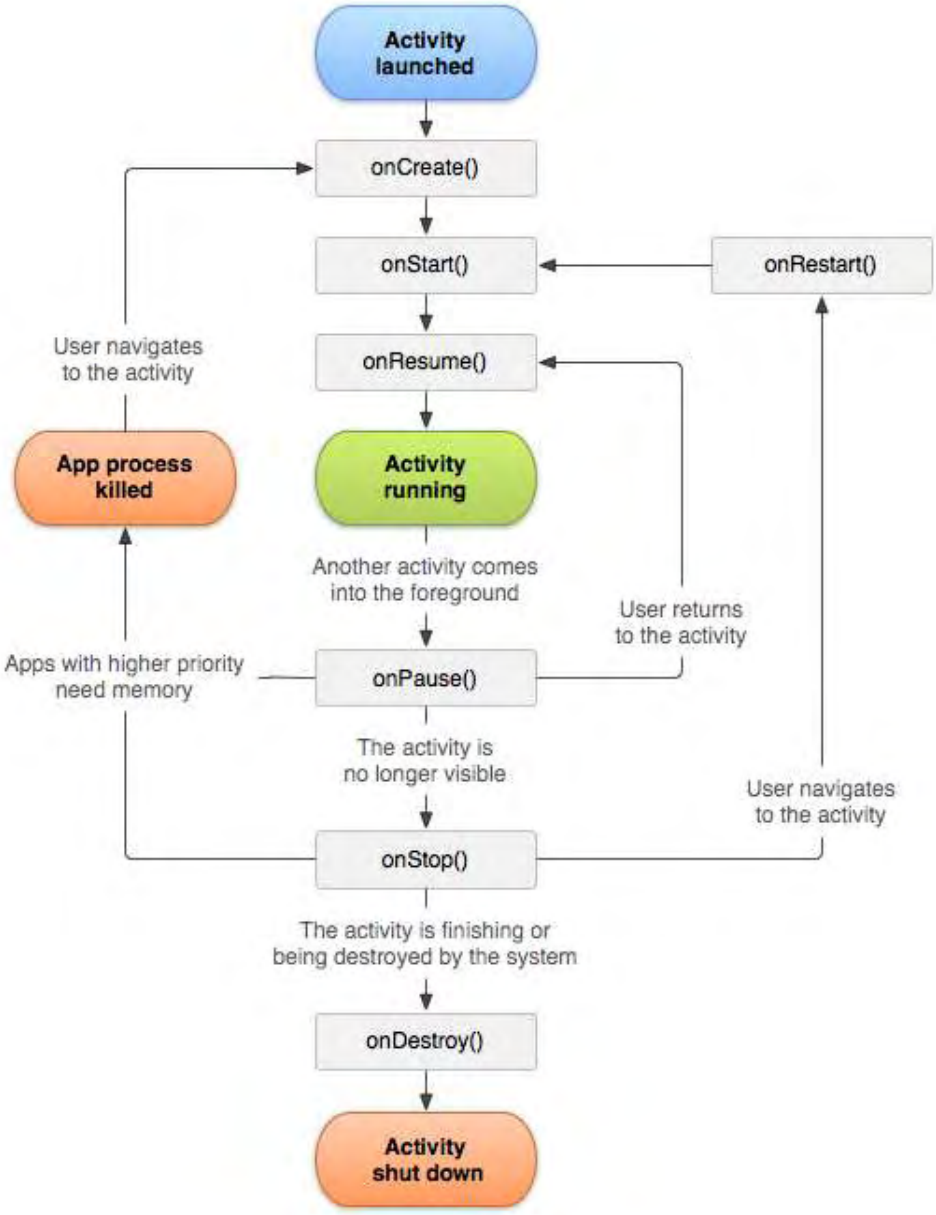
4.4 ΑΝΑΛΥΣΗ ΕΦΑΡΜΟΓΩΝ

4.4.1 ΣΚΕΛΕΤΟΣ ΜΙΑΣ ΔΡΑΣΤΗΡΙΟΤΗΤΑΣ

Όπως προαναφέρθηκε μία δραστηριότητα αποτελείται από Views τα οποία μπορούν να εισαχθούν προγραμματιστικά ή να οριστούν μέσω ενός XML αρχείου, που αποτελεί το Layout της δραστηριότητας. Όταν δημιουργείται μία δραστηριότητα καλούνται ορισμένες συναρτήσεις από το android, οι οποίες περιέχονται στον κύκλο ζωής της δραστηριότητας. Στη συνάρτηση **onCreate()** αρχικοποιούμε τις μεταβλητές που θα χρειαστούμε και θέτουμε το Layout της δραστηριότητας. Το Layout αρχείο περιέχει όλα τα views που εμφανίζονται στη δραστηριότητα. Συνήθως το root

element ενός layout αρχείου είναι ένα ViewGroup, δηλαδή ένα view το οποίο περιέχει άλλα views και ορίζει τη διάταξη αυτών στην οθόνη. Παραδείγματα ViewGroup είναι:

- **LinearLayout:** εμφανίζει τα views σε κάθετη ή οριζόντια σειρά, ανάλογα με τη σειρά που τα συναντάει στο xml αρχείο.
- **RelativeLayout:** τα views τοποθετούνται ανάλογα με κανόνες που ορίζουν το που βρίσκεται το ένα σε σχέση με το άλλο.
- **TableLayout:** χρησιμοποιεί τη λογική πίνακα. Αποτελείται από sub ViewGroup, τα TableRow, που ορίζουν τα views που υπάρχουν σε κάθε σειρά
- **FrameLayout:** εμφανίζει τα views σε στοίβα, καθώς το ένα μπορεί να πέσει πάνω στο άλλο.



ΕΙΚΟΝΑ 4.2 – Ο ΚΥΚΛΟΣ ΖΩΗΣ ΜΙΑΣ ΔΡΑΣΤΗΡΙΟΤΗΤΑΣ

Ανάλογα με το πώς θέλουμε να εμφανίζονται τα views στην οθόνη μας χρησιμοποιούμε και το ανάλογο layout. Αφού ορίσουμε το layout αρχείο μας, τότε στην **onCreate()** εφαρμόζουμε το αντίστοιχο layout με τη συνάρτηση **setContentView().** Έπειτα μπορούμε να διαχειριστούμε το κάθε view, αποκτώντας πρόσβαση μέσω του μοναδικού id που το ορίζει στο layout, με τη χρήση της συνάρτησης **findViewById()** ή ανάλογα με τη θέση του σε ένα viewgroup, με τη χρήση της συνάρτησης **getChildAt()**. Κάθε view προσφέρει διάφορες συναρτήσεις τις οποίες μπορούμε να αξιοποιήσουμε αφού αποκτήσουμε πρόσβαση σε αυτό. Για παράδειγμα ένα **TextView**, το οποίο χρησιμοποιείται για την προβολή κειμένου, μας προσφέρει συναρτήσεις επεξεργασίας του κειμένου του. Τέλος σε κάθε view μπορούμε να ορίσουμε αντίστοιχους listeners, η οποίοι τρέχουν ορισμένα κομμάτια κώδικα, όταν γίνει μία ορισμένη ενέργεια, όπως για παράδειγμα να πατήσει ο χρήστης ένα κουμπί.

4.4.2 ANDROID MANIFEST

Κάθε εφαρμογή απαιτεί την ύπαρξη ενός xml αρχείου, με το όνομα AndroidManifest.xml. Σε αυτό περιέχονται απαραίτητες πληροφορίες για την εφαρμογή, τις οποίες το σύστημα χρησιμοποιεί πριν τρέξει οποιοδήποτε κομμάτι κώδικα της εφαρμογής. Μεταξύ άλλων το Manifest έχει τις εξής λειτουργίες:

- Ορίζει το όνομα του Java πακέτου της εφαρμογής, το οποίο λειτουργεί ως μοναδικό αναγνωριστικό της εφαρμογής.
- Περιγράφει τα δομικά στοιχεία, τα οποία αναλύθηκαν παραπάνω, που αποτελούν την εφαρμογή. Επίσης ορίζει τις κλάσεις που περιέχουν κάθε τέτοιο στοιχείο και δηλώνει τις ικανότητες της, όπως πχ το είδος των λειτουργιών που μπορεί να διαχειριστεί ή το αν αποτελεί την κλάση που τρέχει κατά την εκκίνηση της εφαρμογής.
- Προσδιορίζει τις διαδικασίες που φιλοξενούν τα στοιχεία της εφαρμογής.
- Δηλώνει τις άδειες που πρέπει να έχει η εφαρμογή ώστε να αποκτήσει πρόσβαση σε διάφορες ευαίσθητες λειτουργίες του συστήματος, όπως π.χ. η χρήση internet.
- Δηλώνει τη μικρότερη δυνατή έκδοση του λειτουργικού, στο οποίο μπορεί να τρέξει η εφαρμογή.
- Καταγράφει τις βιβλιοθήκες που χρειάζονται από την εφαρμογή.

***Πηγή (https://developer.android.com/guide/topics/manifest/manifest-intro.html*).**

4.4.3 ΕΠΙΚΟΙΝΩΝΙΑ ΜΕ ΤΟ ΔΙΑΚΟΜΙΣΤΗ

Για να καταστεί δυνατή η επικοινωνία των εφαρμογών με το web server που φιλοξενεί την υπηρεσία web χρησιμοποιείται η κλάση **HttpUrlConnection** που μας προσφέρει το Android. Η κλάση αυτή παρέχει τη δυνατότητα δημιουργίας και διαχείρισης μίας URL σύνδεσης με HTTP χαρακτηριστικά.

Αρχικά δημιουργούμε ένα αντικείμενο **URL** περνώντας στον constructor τη διεύθυνση στην οποία θέλουμε να συνδεθούμε. Έπειτα δημιουργούμε ένα αντικείμενο

HttpUrlConnection καλώντας την συνάρτηση **openConnection()** του **URL** αντικειμένου. Πάνω στο αντικείμενο **HttpUrlConnection** μπορούμε να καλέσουμε ορισμένες συναρτήσεις με τις οποίες ορίζουμε τις κεφαλίδες HTTP (**setRequestProperty**), την HTTP μέθοδο που χρησιμοποιούμε (**setRequestMethod**), τη χρήση ή όχι caching, καθώς και το όριο χρόνου σύνδεσης με το διακομιστή (**setconnectTimeout**). Αφού οριστούν αυτά ανάλογα με τις απαιτήσεις μας, καλούμε την συνάρτηση **connect()** του **HttpUrlConnection** αντικειμένου με την οποία πραγματοποιείται η σύνδεση. Στην περίπτωση που μαζί με την HTTP αίτηση θέλουμε να στείλουμε και δεδομένα POST/PUT/DELETE, δημιουργούμε ένα αντικείμενο **OutputStream** καλώντας τη συνάρτηση **getOutputStream()** στο **HttpUrlConnection** αντικείμενο. Μέσω του **OutputStream** στέλνουμε τα δεδομένα του μηνύματος σε μορφή bytes. Τα δεδομένα αυτά τα δημιουργούμε ανάλογα με το περιεχόμενο τους, το οποίο καθορίζουμε στην κεφαλίδα **Content-Type**. Τα Content-Type που χρησιμοποιούμε είναι δύο είδη και επιλέγονται ανάλογα με τα δεδομένα που στέλνονται. Αν τα δεδομένα είναι key-value ζευγάρια, όπως μία default HTML φόρμα, τότε χρησιμοποιείται ο τύπος "**application/x-www-form-urlencoded**", ενώ σε περίπτωση που θέλουμε να στείλουμε αρχεία ή γενικά μεγάλο όγκο μηνύματος χρησιμοποιούμε το "**multipart/form-data"**.

- **x-www-form-url-encoded:** στην περίπτωση αυτή τα δεδομένα στέλνονται ως "key=value" τιμές, διαχωρισμένα με το χαρακτήρα "&". Αφού δημιουργηθεί η συμβολοσειρά με τα δεδομένα που θα σταλούν στο διακομιστή, τα μετατρέπουμε σε bytes με τη συνάρτηση **getBytes()** και τα στέλνουμε με τη χρήση του **OutputStream**.
- **multipart/form-data:** σε περίπτωση που θέλουμε να στείλουμε αρχεία και όχι απλά αλφαριθμητικά, τότε χρησιμοποιείται αυτός ο τύπος για το σώμα του μηνύματος. Στην κεφαλίδα ορίζεται μία συμβολοσειρά ως διαχωριστικό (boundary) και η κάθε παράμετρος που στέλνεται διαχωρίζεται από αυτό. Κάθε κομμάτι του μηνύματος αποτελείται από το διαχωριστικό, το όνομα και το είδος (MIME) της παραμέτρου και τα bytes που την αντιπροσωπεύουν, το καθένα σε καινούριο σειρά. Τέλος συναντάμε το διαχωριστικό που υποδεικνύει το τέλος του μηνύματος.

Μετά την αποστολή του μηνύματος διαβάζουμε την απάντηση του διακομιστή. Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο μία απάντηση από το web service περιέχει έναν κωδικό και ένα μήνυμα. Τα δεδομένα αυτά μπορούμε να τα λάβουμε χρησιμοποιώντας τις συναρτήσεις "**getResponseCode**" και "**getResponseMessage**" αντίστοιχα του **HttpUrlConnection** αντικειμένου. Πέρα των βασικών δεδομένων που βρίσκονται σε μία απόκριση, ο διακομιστής μπορεί να στείλει και δεδομένα σε μορφή JSON. Για να διαβάσουμε αυτά τα δεδομένα δημιουργούμε ένα αντικείμενο **InputStreamReader** με τη συνάρτηση **getInputStream** του **HttpUrlConnection** αντικειμένου, την οποία περνάμε σε ένα αντικείμενο **BufferedReader** με το οποίο διαβάζουμε το μήνυμα του διακομιστή. Το JSON που στέλνεται από το διακομιστή λαμβάνεται ως ένα αλφαριθμητικό, το οποίο μετατρέπεται σε αντικείμενο JSON από την εφαρμογή.

Καθώς όλη η εφαρμογή βασίζεται στην αποστολή HTTP αιτήσεων και το διάβασμα των αποκρίσεων τους, δημιουργήσαμε την κλάση WebServiceAPI η οποία μας παρέχει εργαλεία ώστε να γλυτώσουμε την επανάληψη κώδικα, αλλά και να διευκολύνουμε την ανάπτυξη της κύριας εφαρμογής. Οι λειτουργίες των συναρτήσεων που αναπτύχθηκαν περιγράφονται παρακάτω και όλες χρησιμοποιούν την κλάση HttpUrlConnection και τις HTTP ιδιότητες της, που αναφέρθηκαν παραπάνω:

- **String getJSONStringFromUrl_GET(String Url):** Στέλνει μία αίτηση GET στο URL που ορίζεται ως παράμετρος. Έπειτα διαβάζει και επιστρέφει την απόκριση του διακομιστή. Σε περίπτωση ύπαρξης παραμέτρων GET. αυτές περιέχονται κατευθείαν στο URL.
- **String getStringWithoutParam_POST(String action, String serviceURL)**: Στέλνει μία αίτηση POST ή DELETE, ανάλογα με τη μεταβλητή action, στο URL που περάστηκε, χωρίς όμως να περιέχονται παράμετροι POST στο σώμα μηνύματος της HTTP αίτησης.
- **String getStringWithParam_POST(String action, String serviceURL, Map<String, String> params):** Στέλνει μία αίτηση POST στο συγκεκριμένο URL, καθώς και παραμέτρους POST, οι οποίοι είναι τύπου x-www-form-urlencoded. Η συνάρτηση αυτή διαβάζει της παραμέτρους που περάστηκαν μέσω της params και τις μετατρέπει στη μορφή key1=value1&key2=value2 κ.ο.κ Αφού τις στείλει σε μορφή bytes, διαβάζει την απόκριση του διακομιστή και επιστρέφει το μήνυμα ή τα δεδομένα.
- **String getStringWithParam_POST(String action, String serviceUrl, String params):** Ίδια λειτουργία με την προηγούμενη, αλλά οι παράμετροι POST περνιούνται ως το καθωσπρέπει String για να σταλούν στο διακομιστή.
- **String multiPartRequest(String action, String serviceURL, Bitmap bitmap, Uri floorplan_uri, Map<String, String> params, ArrayList<DateObject> dates, ArrayList<PolicyObject> policies):** Στέλνει μία αίτηση POST ή DELETE, ανάλογα με το action, τύπου multipart/form-data. Μορφοποιεί τα δεδομένα στη μορφή μίας multipart/form-data αίτησης και τα στέλνει ως bytes στο διακομιστή. Έπειτα διαβάζει την απάντηση και την επιστρέφει. Τα δεδομένα εισόδου θα αναλυθούν παρακάτω.
- **JSONObject convertJSONString2Obj(String jsonString):** Μετατρέπει ένα αλφαριθμητικό σε ένα αντικείμενο JSON. Αυτό γίνεται περνώντας το αλφαριθμητικό αυτό στο constructor ενός αντικειμένου **JSONObject()**.

Παρακάτω θα αναφερόμαστε σε αυτές τις συναρτήσεις ώστε να διευκρινίζουμε το είδος της Http σύνδεσης που γίνεται με το server.

ΔΙΑΧΕΙΡΙΣΗ URL ΣΥΝΔΕΣΗΣ ΣΤΗ ΣΥΝΑΡΤΗΣΗ getStringWithParam_Post

```
public String getStringWithParam_POST(String action, String serviceURL,
Map<String, String> params) throws IOException{

    String jsonString = null;
    String line;
    URL url;
    //Create the URL object passed Url
    try {
        url = new URL(serviceURL);
    }catch(MalformedURLException e){
        throw new IllegalArgumentException("invalid url: "+
                  serviceURL);
    }
    //Create post body message from the passed map
    StringBuilder bodyBuilder = new StringBuilder();
    Iterator<Map.Entry<String, String>> iterator =
            params.entrySet().iterator();

    //Build parameter string from the map iteration
    while(iterator.hasNext()){
        Map.Entry<String, String> param = iterator.next();
        bodyBuilder.append(param.getKey()).append('=').
                    append(param.getValue());
        if(iterator.hasNext()){
            bodyBuilder.append('&');
        }
    }
    String body = bodyBuilder.toString();
    //Convert parameter String to bytes
    byte[] bytes = body.getBytes();
    try{
        sharedPref =
        context.getSharedPreferences(MainActivity.LOGINVALUES, 0);

        //Create HTTPUrlConnection object and set up the request headers
        conn = (HttpURLConnection) url.openConnection();
        conn.setDoOutput(true);
        conn.setUseCaches(false);
        conn.setFixedLengthStreamingMode(bytes.length);
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type","application/x-www-
                                form-urlencoded;charset=UTF-8");
        conn.setRequestProperty("Connection", "close");

        //Set stored Token in the Authorization Header
        String token = sharedPref.getString("token",null);
       if(token != null){
            conn.setRequestProperty("Authorization", "Bearer "+token);
        }
```

```
        conn.setConnectTimeout(10 * 1000);
        conn.connect();
        //post therequest
        OutputStream out = conn.getOutputStream();
        out.write(bytes);
        out.close();

        //handle the response
        int status = conn.getResponseCode();
        String msg = conn.getResponseMessage();

        //Actions based on the server's response
        if(action.equals("login") || action.equals("register")){
           //Decrypt and store the JWT's values in SharedPreferences
           handleLogin(status);
           conn.disconnect();
           return msg;
        //Token has expired
        }else if(action.equals("reservation") && status==440){
            handleLogout();
            conn.disconnect();
            return ("sessionexpired");
        //Bad Request
        }else if(status==400) {
            conn.disconnect();
            return status+msg;
        //Server error. Eg Timeout
        }else if(status==500){
            conn.disconnect();
            return null;
        }else {
            //Read from server and return answer
            BufferedReader bufferedReader = new BufferedReader(new
                        InputStreamReader(conn.getInputStream()));
            StringBuilder stringBuilder = new StringBuilder();

            while ((line = bufferedReader.readLine()) != null) {
                stringBuilder.append(line +'\n');
            }

            jsonString = stringBuilder.toString();
            if(jsonString.isEmpty()){
                return msg;
            }
        }
    //If there was an error return null, otherwise return the JSON response
    }catch(MalformedURLException | ProtocolException |
           SocketTimeoutException e){
        e.printStackTrace();
        jsonString = null;
```

```
    }finally{
        conn.disconnect();
    }

    return jsonString;
}
```

Όλες οι άλλες συναρτήσεις της κλάσης WebServiceAPI υλοποιούνται με παρόμοιο τρόπο, αλλάζοντας ορισμένες παραμέτρους.

Κάθε εφαρμογή έχει ένα ειδικό νήμα στο οποίο τρέχουν όλα τα αντικείμενα διεπαφής χρήστη, όπως είναι για παράδειγμα τα views, το οποίο ονομάζεται UI Thread. Το Android δεν επιτρέπει τη δημιουργία Url συνδέσεων πάνω σε αυτό το νήμα Για αυτό το λόγω χρησιμοποιούμε την κλάση **AsyncTask**, μέσω της οποίας γίνεται η επικοινωνία με το διακομιστή, με τη χρήση της HttpUrlConnection. Η **AsyncTask** επιτρέπει την εκτέλεση ενός κομματιού κώδικα σε νήμα που τρέχει στο παρασκήνιο, καθώς και τη δημοσίευση των αποτελεσμάτων στο UI Thread.

Όταν δημιουργούμε ένα αντικείμενο που κάνει extend την κλάση **AsyncTask<Params, Progress, Result>**, κάνουμε τις εξής συναρτήσεις:

- **onPreExecute():** Η συνάρτηση αυτή τρέχει στο UI Thread ακριβώς πριν τρέξει το νήμα στο παρασκήνιο και την χρησιμοποιούμε για την εμφάνιση ενός διαλόγου που εμφανίζεται στο χρήστη μέχρι να ολοκληρωθεί διαδικασία επικοινωνίας με το διακομιστή.
- **doInBackground():** Αφού επιστρέφει η **onPreExecute(),** τρέχει στο παρασκήνιο ο κώδικας που ορίζουμε σε αυτήν τη συνάρτηση. Εδώ χρησιμοποιούμε την HttpUrlConnection για να επικοινωνήσουμε με το server. Η συνάρτηση αυτή δέχεται οσαδήποτε ορίσματα ίδιου τύπου με το **Params** που δηλώθηκε κατά τον ορισμό. Ως ορίσματα αυτής της συνάρτησης περνάμε τις παραμέτρους που θα σταλούν προς το διακομιστή. Αφού λάβουμε την απάντηση από το διακομιστή, η συνάρτηση αυτή επιστρέφει το μήνυμα ή τα δεδομένα του διακομιστή στην επόμενη.
- **onPostExecute():** Η συνάρτηση αυτή έχει ως ορίσματα τα δεδομένα που επιστράφηκαν από τη **doInBackground()** και είναι ίδιου τύπου με το **Result** που δηλώθηκε στον ορισμό της κλάσης. Ανάλογα με τον κωδικό της απάντησης ή τα δεδομένα που λήφθηκαν εκτελούμε και τον ανάλογο κώδικα, ο οποίος τρέχει στο UI Thread.
- **onProgressUpdate():** Προαιρετικά μπορούμε να καλέσουμε τη συνάρτηση **publishProgress()**, για να τρέξουμε τον κώδικα που ορίζεται στη συνάρτηση **onProgressUpdate()** στο UI Thread, από οποιαδήποτε από τις παραπάνω συναρτήσεις. Τα ορίσματα της πρέπει να είναι ίδιου τύπου με το **Progress**.

Αφού δημιουργήσουμε ένα αντικείμενο της κλάσης AsyncTask, μπορούμε να καλέσουμε σε αυτό τη συνάρτηση **execute()**, περνώντας τις παραμέτρους που θα περαστούν στην **doInBackground()**, ώστε να τρέξουν όλες οι παραπάνω

συναρτήσεις.

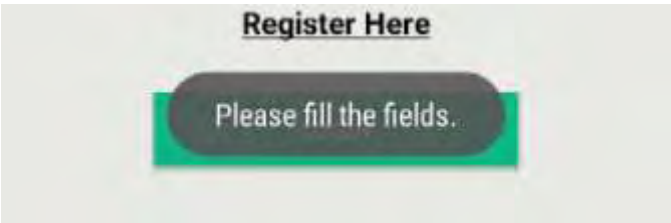
4.4.4 Η ΕΦΑΡΜΟΓΗ ΚΡΑΤΗΣΕΩΝ

Σε αυτή την παράγραφο θα αναλυθεί η εφαρμογή που απευθύνεται στους απλούς χρήστες, δηλαδή σε αυτούς που θέλουν να κάνουν κρατήσεις. Ο χρήστης μέσω της εφαρμογής αυτής θα μπορεί να δει τις διαθέσιμες διοργανώσεις, καθώς και να πραγματοποιήσει νέα κράτηση για την ημερομηνία που τον ενδιαφέρει. Επιπλέον μπορεί να δει τις κρατήσεις που έχει ήδη πραγματοποιήσει μέχρι στιγμής, καθώς και να διαχειριστεί αυτές. Τέλος σε περίπτωση που δεν έχει λογαριασμό στην υπηρεσία, του δίνεται η δυνατότητα να δημιουργήσει νέο και να εισαχθεί με αυτόν στο σύστημα.

ΕΙΚΟΝΑ 4.3 – LOGO ΤΗΣ ΕΦΑΡΜΟΓΗΣ

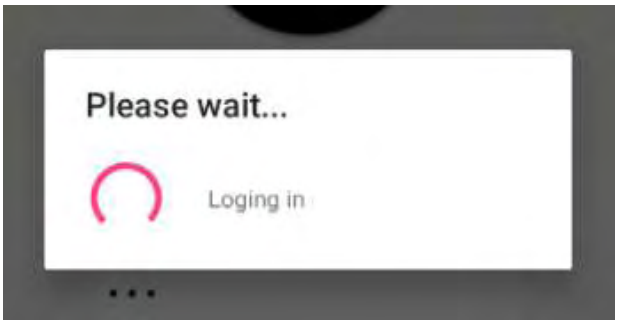
Για τη χρήση της εφαρμογής απαιτείται η ύπαρξη λογαριασμού στην υπηρεσία. Στην αρχική οθόνη ζητείται από το χρήστη να συνδεθεί παρέχοντας το username και το password του. Αφού το web service και η ιστοσελίδα χρησιμοποιούν την ίδια βάση δεδομένων, ο χρήστης μπορεί να συνδεθεί με τα στοιχεία του λογαριασμού που έχει στην ιστοσελίδα. Στην περίπτωση που δεν έχει λογαριασμό στην υπηρεσία, τότε του δίνεται η δυνατότητα να φτιάξει καινούριο, κάνοντας κλικ στο αντίστοιχο κείμενο (**TextView**), το οποίο τον οδηγεί στη δραστηριότητα εγγραφής.

Η αρχική οθόνη αποτελείται από δύο **EditText** views, τα οποία επιτρέπουν στο χρήστη να εισάγει δεδομένα με το πληκτρολόγιο της συσκευής (hardware ή software). Αφού συμπληρώσει τα δεδομένα με το πάτημα του κουμπιού **LOGIN**, επιχειρείται είσοδος στο σύστημα, εκτελείται το AsyncTask που στέλνει την αίτηση στο διακομιστή. Σε περίπτωση που τα στοιχεία δεν είναι συμπληρωμένα εμφανίζεται ένα μήνυμα **Toast** στο χρήστη που τον παραπέμπει να συμπληρώσει τα πεδία. Το μήνυμα **Toast** είναι ένα σύντομο pop-up μήνυμα που εμφανίζεται για περιορισμένο χρονικό διάστημα στην οθόνη.



ΕΙΚΟΝΑ 4.4 – ΤΟ ΜΗΝΥΜΑ TOAST

Στην **onPreExecute()** του AsyncTask εμφανίζουμε ένα παράθυρο διαλόγου στο χρήστη, το οποίο τον αποτρέπει να χρησιμοποιήσει το Interface καθώς περιμένει να ολοκληρωθεί η αίτηση προς τον server. Αυτό το επιτυγχάνουμε με τη δημιουργία ενός αντικειμένου **ProgressDialog**, στο οποίο καλούμε τη μέθοδο **show()**, που έχει ως ορίσματα το context στο οποίο τρέχει, το μήνυμα που εμφανίζει στο χρήστη και τον τίτλο του παραθύρου.



ΕΙΚΟΝΑ 4.5 – ΤΟ ProgressDialog

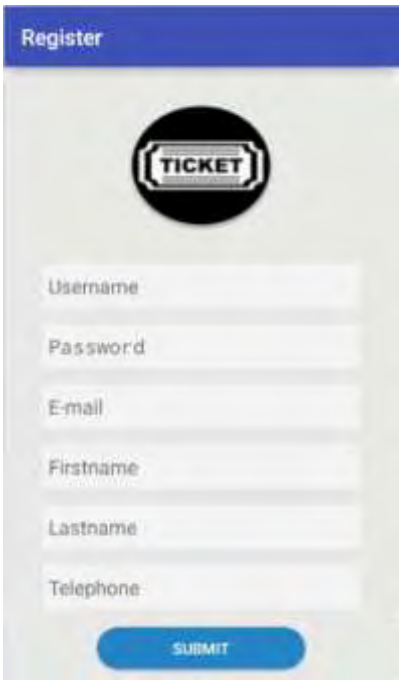
Αφού εμφανιστεί ο διάλογος τρέχει στο παρασκήνιο η doInBackground. Χρησιμοποιώντας τη συνάρτηση **getStringWithParam_POST**, στέλνουμε τα δεδομένα εισόδου στο διακομιστή και διαβάζουμε την απάντηση στην **onPostExecute()**. Στην περίπτωση που η επαλήθευση ήταν ανεπιτυχής, εμφανίζεται ανάλογο Toast. Όταν ο χρήστης επαληθευτεί επιτυχώς, τότε ο διακομιστής στέλνει το Access Token (JWT). Το Android μας προσφέρει τη δυνατότητα να αποθηκεύσουμε δεδομένα με τη μορφή key/value, με τη χρήση της κλάσης **SharedPreferences**. Τα δεδομένα αυτά διατηρούνται ακόμα και αν κλείσει η εφαρμογή. Αφού εξάγουμε το username από το δεύτερο part του JWT με τη χρήση της **Base64.decode** συνάρτησης, αποθηκεύουμε στα **SharedPreferences** το username, το οποίο εμφανίζεται στο χρήστη, και το JWT το οποίο στέλνεται με την κάθε αίτηση. Η αποθήκευση του Token στη συσκευή δίνει μία ψευδή αίσθηση ενός login/logout συστήματος, αφού ουδέποτε η συσκευή διατηρεί μόνιμη σύνδεση με το web service, το οποίο είναι stateless και κάθε αίτηση πρέπει να δύναται να εξυπηρετηθεί με τα δεδομένα που μόνο αυτή φέρνει. Έπειτα μεταφέρουμε το χρήστη στην κεντρική δραστηριότητα της εφαρμογής.

Όσο το token είναι έγκυρο αποτρέπουμε το χρήστη από το να ξανακάνει login. Κατά την εκκίνηση της εφαρμογής, στη δραστηριότητα εισόδου ελέγχεται η ύπαρξη αποθηκευμένου Token στα SharedPreferences. Στην περίπτωση που δεν βρεθεί τότε εμφανίζεται κανονικά η οθόνη εισαγωγής, αλλιώς ξεκινάει η κύρια δραστηριότητα της εφαρμογής. Για την έναρξη μίας δραστηριότητας χρησιμοποιείται η κλάση **Intent**, η οποία δίνει μία γενική περιγραφή της πράξης που θέλουμε να γίνει. Για να ξεκινήσουμε μία νέα δραστηριότητα δημιουργούμε ένα καινούριο αντικείμενο **Intent**, στο οποίο περνάμε ως ορίσματα το **context** στο οποίο τρέχει ο κώδικας, καθώς και την κλάση που θέλουμε να ξεκινήσουμε. Σε αυτήν την περίπτωση, δηλαδή, την κλάση της δραστηριότητας εισόδου και την κλάση της κύριας δραστηριότητας.

Έπειτα καλούμε τη συνάρτηση **startActivity(Intent intent)**. Κάθε νέα δραστηριότητα αποθηκεύεται σε ένα Stack (Back Stack) το οποίο κρατάει το σύστημα, ώστε όταν πατήσουμε το πλήκτρο **"Πίσω"** να μας πάει στην ακριβώς προηγούμενη. Επειδή όταν κάνουμε είσοδο δεν θέλουμε ο χρήστης να μπορεί να μεταφερθεί πάλι στην δραστηριότητα εισόδου, μαζί με το Intent παίρναμε το flag **FLAG_ACTIVITY_CLEAR_TASK**, ώστε να διαγράψουμε το Back Stack, με συνέπεια όταν ο χρήστης πατήσει το **"Πίσω"** να κλείσει την εφαρμογή και όχι να εμφανίσει τη δραστηριότητα εισόδου.

ΕΙΚΟΝΑ 4.5 – Η ΑΡΧΙΚΗ ΟΘΟΝΗ ΣΥΝΔΕΣΗΣ

Στην περίπτωση που ο χρήστης θέλει να δημιουργήσει καινούριο λογαριασμό, τότε πατώντας στο αντίστοιχο text, μεταφέρεται στην δραστηριότητα εγγραφής. Η δραστηριότητα αυτή, περιέχει ένα σύνολο **EditText** views, με τα στοιχεία που πρέπει να συμπληρώσει ο χρήστης. Με την επιλογή του κουμπιού Register στέλνεται η αίτηση POST στο διακομιστή στο αντίστοιχο URL (/users), μέσω μίας AsyncTask και του WebServiceAPI. Σε περίπτωση επιτυχής εγγραφής αποθηκεύουμε το Token και το username στα SharedPreferences και ξεκινάμε την κύρια δραστηριότητα της εφαρμογής. Σε αντίθετη περίπτωση εμφανίζεται Toast με το μήνυμα λάθους του διακομιστή.



ΕΙΚΟΝΑ 4.6 – Η ΔΡΑΣΤΗΡΙΟΤΗΤΑ ΕΓΓΡΑΦΗΣ

Όλα τα views περιέχονται σε ένα **LinearLayout** με κάθετο προσανατολισμό. Επειδή σε μικρές οθόνες η δραστηριότητα μπορεί να μην χωράει ολόκληρη, κλείνουμε το LinearLayout μέσα σε ένα **ScrollView**, το οποίο παρέχει δυνατότητα scrolling στο view που περιέχει.

Αφού επαληθευτεί ο χρήστης και αποθηκευτεί το Token, τότε κάθε φορά που ξεκινάει η εφαρμογή εμφανίζεται η κύρια δραστηριότητα. Για την υλοποίηση της κύριας δραστηριότητας χρησιμοποιήθηκε η αρχιτεκτονική "μία δραστηριότητα και πολλά fragments (τμήματα διεπαφής)". Το fragment είναι ένα κομμάτι διεπαφής της κύριας δραστηριότητας. Έχει το δικό του κύκλο ζωής και πιάνει ένα μέρος της δραστηριότητας. Στην αρχιτεκτονική αυτή χρησιμοποιείται μία δραστηριότητα που περιέχει ένα **FrameLayout** το οποίο καταλαμβάνει όλη την οθόνη και μέσα σε αυτό εναλλάσσουμε διάφορα fragments που έχουμε δημιουργήσει για κάθε λειτουργία. Με αυτό τον τρόπο σπάμε τον κώδικα μας σε κομμάτια, τα οποία διαχειρίζονται πιο εύκολα και γλυτώνουμε την ανταλλαγή δεδομένων μέσω δραστηριοτήτων, καθώς και την εναλλαγή των ίδιων των δραστηριοτήτων, κάθε φορά που θέλουμε να αλλάξουμε την λειτουργία που βλέπει ο χρήστης. Τα δεδομένα αυτά διαχειρίζονται μέσω της κύριας δραστηριότητας, η οποία είναι υπεύθυνη για την εναλλαγή των fragments, καθώς και για την επικοινωνία μεταξύ τους, καθώς δεν υπάρχει άμεση επικοινωνία μεταξύ δύο fragments. Κάθε fragment παρέχει ένα **interface**, το οποίο υλοποιεί η κύρια δραστηριότητα και με αυτόν τον τρόπο ανταλλάσσονται τα δεδομένα μεταξύ τους.

Αρχικά η δραστηριότητα μέσω ενός AsyncTask, στέλνει μία GET αίτηση στο διακομιστή ώστε να λάβει τις κρατήσεις του χρήστη. Σε περίπτωση που το token έχει λήξει, δηλαδή ο διακομιστής απάντησε με "**440 Token Expired**", τότε τερματίζεται η δραστηριότητα αυτή και ξεκινάει η δραστηριότητα εισόδου. Αν η αίτηση ήταν

επιτυχής, ο διακομιστής επιστρέφει ένα JSON αλφαριθμητικό που περιέχει τα δεδομένα κάθε κράτησης. Τότε η κύρια δραστηριότητα δημιουργεί το fragment που είναι υπεύθυνο για την προβολή των κρατήσεων και το τοποθετεί στο FrameLayout, περνώντας σε αυτό το JSON αλφαριθμητικό που γύρισε ο διακομιστής. Αυτό γίνεται με τη βοήθεια του **fragmentManager()**, ο οποίος μας δίνει συναρτήσεις για τη διαχείριση των fragments. Τα δεδομένα που μπορούμε να περάσουμε στο fragment είναι ένα σύνολο από key/value ζευγάρια.

ΕΙΚΟΝΑ 4.7 – ΤΟ FRAGMENT ΜΕ ΤΙΣ ΚΡΑΤΗΣΕΙΣ

Ένα fragment έχει το δικό του layout.xml αρχείο, που καθορίζει τα views που περιέχει. Όπως αναφέρθηκε κάθε fragment έχει το δικό του κύκλου ζωής και τις ανάλογες συναρτήσεις που καλούνται. Στην **onCreateView()**, φορτώνουμε το αντίστοιχο xml αρχείο με τη συνάρτηση inflater() του LayoutInflator που μας παρέχει η **onCreateView()**. Έπειτα στην **onActivityCreated()** μπορούμε να πάρουμε reference σε κάθε view και να τα διαχειριστούμε ανάλογα. Το fragment αυτό αποτελείται από μία λίστα (**ListView**), η οποία περιέχει τις κρατήσεις του χρήστη. Το **ListView** είναι ένα ViewGroup του οποίου τα δεδομένα διαχειρίζονται από έναν αντάπτορα. Ο αντάπτορας χρησιμοποιεί το δικό του xml αρχείο, το οποίο ορίζει το πώς θα προβάλλονται τα αντικείμενα της λίστας. Ως όρισμα στον αντάπτορα, ο οποίος είναι μία κλάση που κάνει extend το **BaseAdapter**, περνάμε το JSON πίνακα που περιέχει τα στοιχεία των κρατήσεων που έστειλε ο διακομιστής. Η κλάση αυτή

μας δίνει τη δυνατότητα να κάνουμε overwrite ένα πλήθος συναρτήσεων οι οποίες μας παρέχουν λειτουργίες σχετικές με τη διαχείριση του κάθε αντικειμένου της λίστας, όπως για παράδειγμα να ορίσουμε ένα id καθώς και το JSON αντικείμενο που θα σχετίζεται με το κάθε πεδίο της λίστας. Ως id ορίζουμε το id της κάθε κράτησης και ως αντικείμενο το αντίστοιχο JSON αντικείμενο του πίνακα JSON που επέστρεψε ο διακομιστής, πχ στη πρώτη θέση της λίστας θα βρίσκεται το πρώτο JSON αντικείμενο κ.ο.κ. Τέλος στη **getView()** συνάρτηση του αντάπτορα ορίζουμε το view που θα προβληθεί. Σε αυτό το σημείο με τη χρήση της συνάρτησης **findViewById()** παίρνουμε τα references των views που βρίσκονται στο xml αρχείο του αντάπτορα και τους αναθέτουμε τις τιμές των JSON αντικειμένων. Όπως βλέπουμε από την εικόνα **4.7** το τελικό view περιλαμβάνει τον τίτλο της κράτησης, το id, την ημερομηνία και την ώρα, τις κλεισμένες θέσεις καθώς και την κατάστασή της.

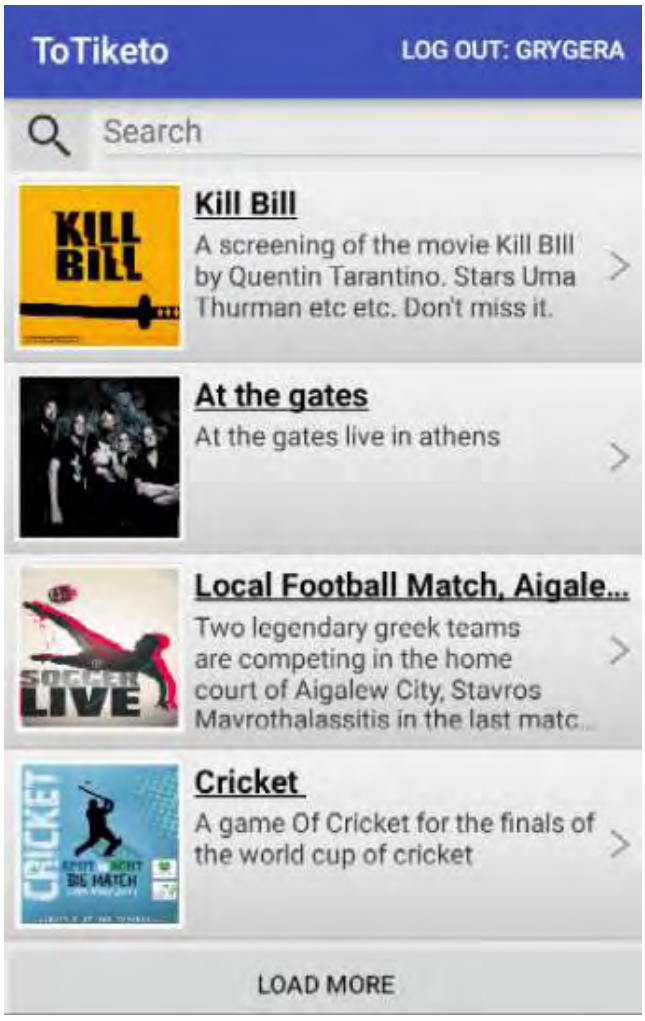
ΠΑΡΑΔΕΙΜΓΑ AsyncTask ΓΙΑ ΤΗ ΛΗΨΗ ΚΑΙ ΕΜΦΑΝΙΣΗ ΚΡΑΤΗΣΕΩΝ

```
private class GetReservationsTask extends AsyncTask<Void, Void, String>{

    //Εμφάνιση παραθύρου διαλόγου
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        m_ProgressDialog = ProgressDialog.show(MainActivity.this, "Please
                           wait...", "Processing...", true);
    }
    //Αποστολή HTTP αίτησης
    @Override
    protected String doInBackground(Void... params){

        String url = "http://10.0.2.2/bookItWebService/reservations";
        String jsonString;

        try{
            //Η συνάρτηση getJsonStringFromUrl_GET φροντίζει για την
            αποστολή της αίτησης
            jsonString = m_Web.getJSONStringFromUrl_GET(url);
            //Σε περίπτωση επιτυχίας, η απάντηση μεταφέρεται στην
              onPostExecute, αλλιώς επιστρέφεται null
            return jsonString;
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }

    @Override
    protected void onPostExecute(String jsonString){
        super.onPostExecute(jsonString);

        //Τερματισμός παραθύρου διαλόγου
```

```
        m_ProgressDialog.dismiss();
        //Προετοιμασία καινούριου fragment
        Bundle bundle = new Bundle();
        bundle.putString("json", jsonString);

        //Σε περίπτωση αποτυχίας σύνδεσης εμφάνισε ανάλογο μήνυμα
        if(null == jsonString){
            Toast.makeText(MainActivity.this, "Server not available",
                           Toast.LENGTH_SHORT).show();
            return;
        //Σε περίπτωση που έληξε το Token, ανακατεύθυνση στη Login
          δραστηριότητα
        }else if(jsonString.equals("sessionexpired")){
            Toast.makeText(MainActivity.this, "Session Expired Please Login
                           Again", Toast.LENGTH_SHORT).show();
            Intent nxt = new Intent(getApplicationContext() ,Login.class);
            nxt.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
                         Intent.FLAG_ACTIVITY_CLEAR_TASK);
            startActivity(nxt);
        }else{
            //Αρχικοποίηση fragment και εμφάνιση του
            ReservationsFragment reservationsFragment = new
                                 ReservationsFragment();
            reservationsFragment.setArguments(bundle);

            FragmentTransaction addMainMenu =
                                 mFragmentManager.beginTransaction();
            //Αντικατάσταση υπάρχων fragment με το fragment των κρατήσεων
            addMainMenu.replace(R.id.main_container, reservationsFragment,
                                 "reserve");
            addMainMenu.commit();
        }
    }
}
```

Εκτός από τη λίστα που περιέχει τις πληροφορίες για τις κρατήσεις του χρήστη, υπάρχει το κουμπί **MAKE A NEW RESERVATION**, με το οποίο ο χρήστη μπορεί να κάνει μία νέα κράτηση. Μόλις επιλέξει ο χρήστης το κουμπί αυτό, καλείται η callback συνάρτηση του interface, την οποία υλοποιεί η κύρια δραστηριότητα. Η κύρια δραστηριότητα στέλνει ένα αίτημα GET στο διακομιστή ώστε να λάβει τις διαθέσιμες διοργανώσεις. Αφού λάβει με επιτυχία το JSON αλφαριθμητικό, τότε αντικαταστεί το fragment των κρατήσεων, με το fragment των διοργανώσεων. Με τη χρήση της παραμέτρου **offset** στο ερώτημα GET, λαμβάνουμε 10 διοργανώσεις τη φορά. Το fragment των διοργανώσεων περιέχει ένα **ListView**, το οποίο χρησιμοποιώντας έναν αντάπτορα, προβάλει τις πληροφορίες για τις διαθέσιμες διοργανώσεις. Όπως και πριν το αντικείμενο του αντάπτορα δέχεται ως όρισμα το πίνακα JSON που περιέχει τα αντικείμενα JSON των διοργανώσεων. Το view που επιστρέφει η **getView()** του αντάπτορα περιέχει το όνομα της διοργάνωσης, την περιγραφή της και την εικόνα της. Ως id του κάθε αντικειμένου ορίζουμε το id της

διοργάνωσης που αντιπροσωπεύει, οπότε με το πάτημα ενός αντικειμένου γνωρίζουμε το id του. Η εικόνα που λαμβάνεται κωδικοποιημένη με Base64, αποκωδικοποιείται με τη χρήση της συνάρτησης **Base64.decode** και φορτώνεται στο αντίστοιχο **ImageView**. Σε περίπτωση που οι διοργανώσεις είναι 10 τότε εμφανίζεται το πλήκτρο "LOAD MORE" στο κάτω μέρος της λίστας. Σε αντίθετη περίπτωση το πλήκτρο αναγράφει "YOU 'VE REACHED THE END" και δεν μπορεί να επιλεχθεί. Με την επιλογή του "LOAD MORE", στέλνεται νέα αίτηση στο διακομιστή με παράμετρο **offset++.** Αφού λάβουμε τις νέες διοργανώσεις επαυξάνουμε τον πίνακα του αντάπτορα με τα καινούρια δεδομένα.



ΕΙΚΟΝΑ 4.8 – ΤΟ FRAGMENT ΜΕ ΤΙΣ ΔΙΟΡΓΑΝΩΣΕΙΣ

Στο fragment αυτό δίνεται η δυνατότητα στο χρήστη να αναζητήσει διοργανώσεις με βάση το όνομα τους. Αφού συμπληρώσει το αντίστοιχο **EditView** και πατήσει το κουμπί της αναζήτησης, τότε μέσω ενός AsyncTask που υλοποιείται στο fragment, στέλνουμε μία αίτηση **GET** στο URL "**/events/search/*keyword?offset=0***", ώστε να πάρουμε τις δέκα πρώτες διοργανώσεις της αναζήτησης. Μετά ανανεώνουμε τον πίνακα JSON του αντάπτορα με τα καινούρια δεδομένα και σε περίπτωση που αυτά

ήταν 10, εμφανίζεται το κουμπί “LOAD MORE”, αλλιώς το “YOU ‘VE REACHED THE END”.

ΠΑΡΑΔΕΙΓΜΑ ΒΑΣΙΚΗΣ ΔΟΜΗΣ LAYOUT ΤΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ ΤΟΥ ΑΝΤΑΠΤΟΡΑ

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

        <LinearLayout
            android:id="@+id/thumbnail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true">

            <ImageView
                android:id="@+id/event_preview"
                android:layout_width="90dip"
                android:layout_height="90dip"/>

        </LinearLayout>

        <TextView
            android:id="@+id/event_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignTop="@id/thumbnail"
            android:layout_toRightOf="@id/thumbnail" />

        <TextView
            android:id="@+id/event_description"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_below="@id/event_title"
            android:layout_toRightOf="@+id/thumbnail"
            android:layout_toLeftOf="@+id/arrow" />

        <ImageView
            android:id="@+id/arrow"
            android:layout_width="20dip"
            android:layout_height="20dip"
            android:src="@mipmap/arrow"
            android:layout_alignParentRight="true"
            android:layout_centerVertical="true" />

</RelativeLayout>
```

ΒΑΣΙΚΑ ΣΤΟΙΧΕΙΑ ΜΙΑΣ ΚΛΑΣΗΣ ΑΝΤΑΠΤΟΡΑ

```
//Στον constructor περνάμε τον json πίνακα που θέλουμε να προβάλλουμε
public EventsAdapter(Activity context, JSONArray jsonArray){
    this.context = context;
    this.jsonArray = jsonArray;
}
```

```
//Η συνάρτηση getItem επιστρέφει το json αντικείμενο που σζετίζεται με την
τρέχουσα θέση
@Override
public JSONObject getItem(int position){
    return jsonArray.optJSONObject(position);
}

//Το αντικείμενο ViewHolder αποθηκεύει τα reference καθενός view της λίστας
public static class ViewHolder{
    TextView title;
    TextView category;
    TextView description;
    ImageView preview;
}

public View getView(int position, View convertView, ViewGroup parent){
    //Το αντικείμενο ViewHolder αποθηκεύει τα references κάθε
αντικειμένου/view του αντάπτορα σε ένα Tag, ώστε να μην τα ξανα αναζητάει
κάθε φορα που ανακυκλώνεται το συσγκεκριμένο view.
    ViewHolder viewHolder;
    if(convertView == null){
        viewHolder = new ViewHolder();
        //Φόρτωσει του Layout και αποθήκευση των reference
        LayoutInflater inflater = context.getLayoutInflater();
        convertView = inflater.inflate(R.layout.events_adapter_layout,
                      null, false);
        viewHolder.title = (TextView) convertView.
                           findViewById(R.id.event_title);
        viewHolder.description = (TextView) convertView.
                                 findViewById(R.id.event_description);
        viewHolder.preview = (ImageView) convertView.
                             findViewById(R.id.event_preview);
        convertView.setTag(viewHolder);
    }else{
        //Αν πρόκειται για ανακυκλωμένο view ανακάλεσε τα refenece
        viewHolder = (ViewHolder) convertView.getTag();
    }
    //Πρόσβαση στο JSON αντικείμενο που σχετίζεται με τη θέση της λίστας
    JSONObject jsonObject = getItem(position);

    //Αποκωδικοποιήση εικόνας
    byte[] decodedString = Base64.decode(jsonObject.optString("preview"),
                           Base64.DEFAULT);
    Bitmap decodedByte = BitmapFactory.decodeByteArray(decodedString, 0
                         ,decodedString.length);
    //Θέσιμο τιμών στα αντίστοιχα views
    viewHolder.title.setText(Html.fromHtml("<u>"+jsonObject.
        optString("title")+"</u>"));
    viewHolder.description.setText(jsonObject.optString("details"));
    viewHolder.preview.setImageBitmap(decodedByte);
    return convertView;
    }
}
```

Με την επιλογή μίας διοργάνωσης καλείται η callback συνάρτηση που είναι υλοποιημένη στην κύρια δραστηριότητα και έχει ως όρισμα το id της διοργάνωσης που επιλέχτηκε. Αυτή στέλνει μία αίτηση **GET** στο χρήστη ώστε να λάβει

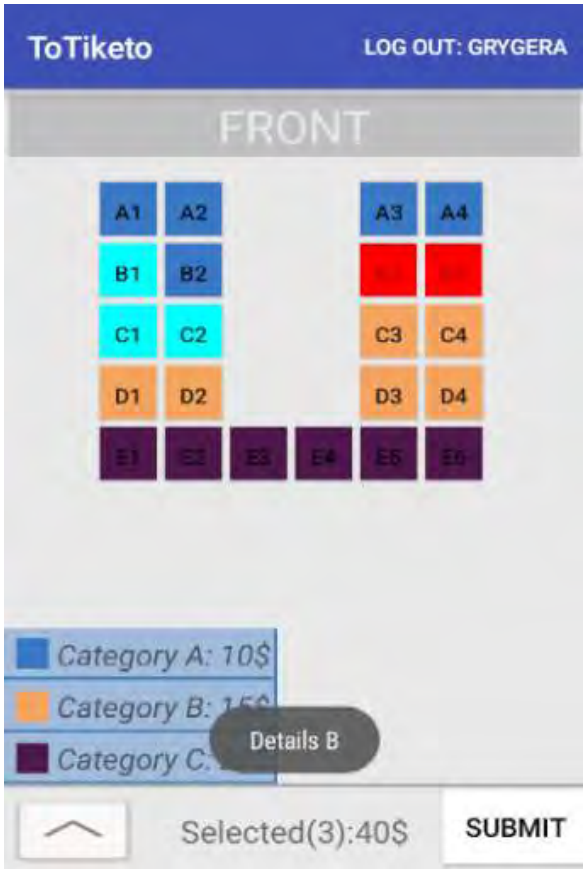
πληροφορίες για τη συγκεκριμένη διοργάνωση. Έπειτα αντικαταστεί το fragment των διοργανώσεων, με το fragment των πληροφοριών. Αυτό περιέχει πληροφορίες για τη συγκεκριμένη διοργάνωση, καθώς και τις διαθέσιμες ημερομηνίες της. Σε κάθε view μίας ημερομηνίας αντιστοιχούμε το αντίστοιχο id της, το οποίο ξέρουμε από το JSON αντικείμενο που επιστράφηκε. Μπορούμε να δώσουμε ένα αναγνωριστικό σε ένα view, με τη χρήση της συνάρτησης setTag(). Όταν επιλεχθεί ένα αντικείμενο τότε με τη συνάρτηση getTag() λαμβάνουμε την τιμή που είχαμε θέση και ξέρουμε σε ποια ημερομηνία αναφερόμαστε. Αυτό το κάνουμε διότι σε αυτήν την περίπτωση δεν χρησιμοποιείται **ListView**, με το οποίο μπορούμε να ορίσουμε το id στην κλάση του αντάπτορα, αλλά ένα LinearLayout που προσαυξάνουμε με το view της ημερομηνίας καθώς προσπελαύνουμε το JSON αντικείμενο που έστειλε ο διακομιστής. Προτιμάται η χρήση LinearLayout αντί ListView, γιατί η χρήση ListView μέσα σε ένα ScrollView, το οποίο περικλύει το LinearLayout ώστε να μπορέσουμε να κάνουμε scroll σε περίπτωση που τα δεδομένα δεν χωράνε, δημιουργεί προβλήματα με το scroll, καθώς το ListView φροντίζει από μόνο του για αυτό και έρχεται σε σύγκρουση με το scroll του ScrollView δημιουργώντας αναπάντεχες συμπεριφορές.

ΕΙΚΟΝΑ 4.9 – ΤΟ FRAGMENT ΤΩΝ ΠΛΗΡΟΦΟΡΙΩΝ

Με του που ο χρήστης επιλέξει μία ημερομηνία καλείται η callback συνάρτηση της κύριας δραστηριότητας με ορίσματα το id της ημερομηνίας. Έπειτα αυτή στέλνει μία

αίτηση GET στον διακομιστή ώστε να λάβει πληροφορίες με τα εισιτήρια που προσφέρονται. Στο JSON αντικείμενου που στέλνει ο διακομιστής ελέγχεται αν υπάρχει το πεδίο floorplan και ανάλογα με την ύπαρξη ή όχι προστίθεται το ανάλογο fragment, αφού αφαιρεθεί αυτό των πληροφοριών.

- **Floorplan Fragment:** Σε περίπτωση ύπαρξης floor plan, το JSON αντικείμενο που στέλνει ο διακομιστής περιέχει, εκτός από τις πληροφορίες των πολιτικών των εισιτηρίων, το αλφαριθμητικό του map, καθώς και τα ids των κλεισμένων θέσεων. Για την δημιουργία του map χρησιμοποιείται ένα TableLayout, στο οποίο κάθε θέση αναπαρίσταται από ένα **ToggleButton**, στο οποίο αποθηκεύεται το id με τη χρήση της συνάρτησης **setTag()**. Το text του ToggleButton είναι το όνομα της θέσης. Επίσης για κάθε πολιτική θέτουμε και άλλο χρώμα στο **ToggleButton.** Για κάθε αύξον γράμμα έχουμε αποθηκευμένο το αντίστοιχο χρώμα. Καθώς δημιουργούμε τα ToggleButtons ελέγχουμε αν το id της συγκεκριμένης θέσης βρίσκεται στον πίνακα των κλεισμένων θέσεων. Σε αυτή την περίπτωση απενεργοποιούμε το ToggleButton και θέτουμε κόκκινο χρώμα. Τα ToggleButtons δημιουργούνται προγραμματιστικά καθώς διαβάζουμε το αλφαριθμητικό του map. Το πλήθος των επιλεγμένων θέσεων, καθώς και το κόστος τους εμφανίζεται στο κάτω μέρος της οθόνης σε ένα **TextView.** Για την ενημέρωση του χρήστη σχετικά με τις πολιτικές των εισιτηρίων, παρέχεται ένας πίνακας ο οποίος εμφανίζεται με την επιλογή του αντίστοιχου **ToggleButton.** Αυτός περιέχει τα ονόματα των κατηγοριών και τις τιμές, τα οποία άμα πατήσουμε εμφανίζεται ένα μήνυμα Toast με τις πληροφορίες της αντίστοιχης θέσης.



ΕΙΚΟΝΑ 4.10 – ΤΟ FLOORPLAN FRAGMENT

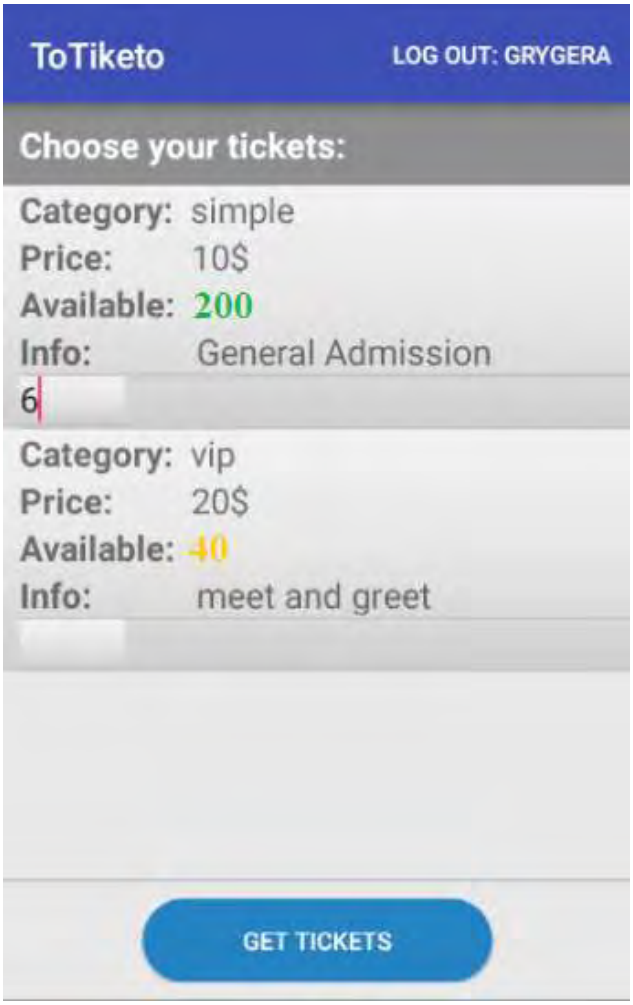
Αφού ο χρήστης επιλέξει τις θέσεις που θέλει, οι οποίες εμφανίζονται με γαλάζιο χρώμα, πατάει το κουμπί **SUBMIT**. Τότε στέλνεται μία λίστα με τα ids των επιλεγμένων θέσεων στην callback συνάρτηση που υλοποιεί η κύρια δραστηριότητα, η οποία στέλνει την αντίστοιχη αίτηση **POST** στο διακομιστή για να κρατήσει προσωρινά τις θέσεις.

Η ΣΥΝΑΡΤΗΣΗ ΔΗΜΙΟΥΡΓΙΑΣ ΕΝΟΣ ToggleButton

```
private ToggleButton createToggle(int id, String row, int col, String
status, char identifier){

    //Αρχικοποιήση  και θέσιμο xml παραμέτρων
    ToggleButton toggleButton = new ToggleButton(getActivity());
    TableRow.LayoutParams params = new TableRow.LayoutParams(70,70);
    params.setMargins(5,5,5,5);

    toggleButton.setLayoutParams(params);
    toggleButton.setText(row+col);
    //Αποθήκευση id και γράμμα πολιτικής επάνω στο ToggleButton
    toggleButton.setId(id);
    toggleButton.setTag(identifier);
    toggleButton.setTextOff(row+col);
    toggleButton.setTextOn(row+col);

    if(status.equals("free")){
         //Ανάλογα το αναγνωριστικό γράμμα επιλέγουμε και άλλο χρώμα
         toggleButton.setBackgroundColor(Color.
             parseColor(getColor(identifier)));
    }else if(status.equals("reserved")){
        toggleButton.setEnabled(false);
        toggleButton.setBackgroundColor(Color.RED);
    } else if (status.equals("space")) {
        toggleButton.setVisibility(View.INVISIBLE);
    }

    if(status.equals("free")) {
        toggleButton.setOnCheckedChangeListener(new
            CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
                Boolean isChecked) {
                String text = selectedTv.getText().toString();
                if (isChecked) {
                    seatsSelected.add(buttonView.getId());

                    //Ενημέρωση του συνόλου
                    totalTickets++;

                    //Ενημέρωση του TextView του συνόλου των
                    εισιτηρίων
                    selectedTv.setText(text.replace(text.
                        substring(text.indexOf('(') + 1,
                        text.indexOf(')')
                        +1), Integer.toString(totalTickets) + ")"));

                    text = selectedTv.getText().toString();
```

```
                        //Ενημέρωση της συνολικής τιμής και του TextView
                        totalPrice  +=  prices.get(buttonView.getTag());
                        selectedTv.setText(text.replace
                            (text.substring(text.indexOf(':') + 1,
                            text.indexOf('$') + 1),
                            Integer.toString(totalPrice) + "$"));

                        //Αλλαγή χρώματος σεεπιλεγμένο
                        buttonView.setBackgroundColor(Color.CYAN);
                    } else {
                        seatsSelected.remove((Integer)
                            buttonView.getId());

                        //Αφαίρεση από το σύνολο των εισιτηρίων
                        totalTickets--;
                        selectedTv.setText(text.replace
                            (text.substring(text.indexOf('(')+1,
                            text.indexOf(')') + 1),
                            Integer.toString(totalTickets) + ")"));

                        text = selectedTv.getText().toString();

                        //Αφαίρεση από τη συνολική τιμή
                        totalPrice -= prices.get(buttonView.getTag());
                        selectedTv.setText(text.replace(text.substring
                            (text.indexOf(':') + 1, text.indexOf('$')
                            +1), Integer.toString(totalPrice) + "$"));

                        buttonView.setBackgroundColor(Color.parseColor
                            (getColor((char) buttonView.getTag())));
                    }
                }
            });
        }
        return toggleButton;

    }
```

- **No Floorplan Fragment:** Στην περίπτωση που δεν υπάρχει floorplan, τότε το JSON αντικείμενο περιέχει, εκτός από τις πληροφορίας για τις πολιτικές, τον αριθμό των διαθέσιμων εισιτηρίων για κάθε πολιτική. Το fragment αυτό αποτελείται από ένα LinearLayout, το οποίο περιέχει πληροφορίες για τις πολιτικές των εισιτηρίων, καθώς και των διαθέσιμο αριθμό αυτών. Σε κάθε πολιτική δίνεται ένα **EditText**, στο οποίο ο χρήστης δηλώνει πόσα εισιτήρια θέλει για κάθε μία.

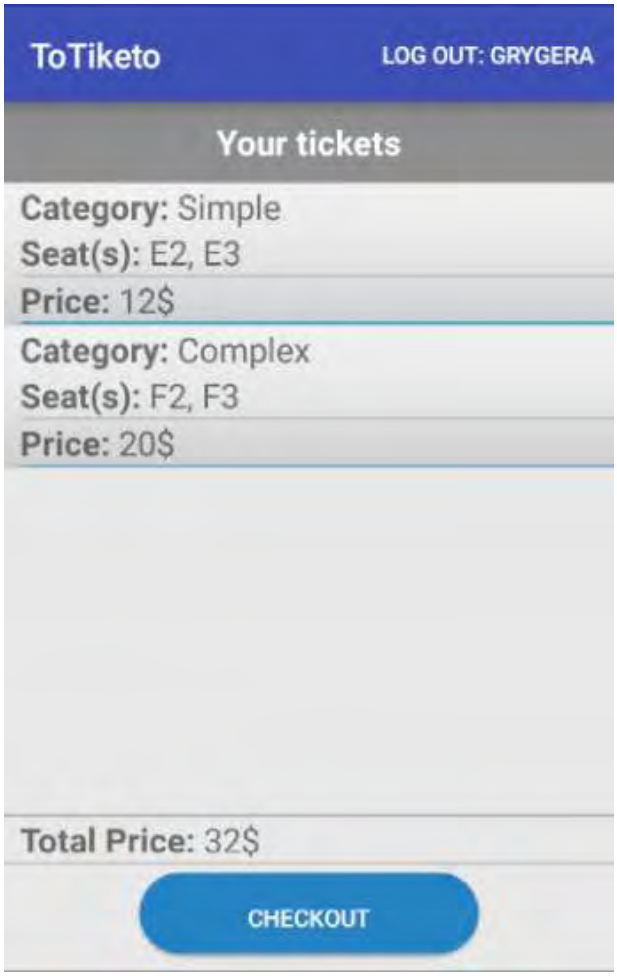


ΕΙΚΟΝΑ 4.11 – ΤΟ NOFLOORPLAN FRAGMENT

Με την επιλογή του κουμπιού GET TICKETS, στέλνεται ένα Map που έχει ως key το id της πολιτικής και ως τιμή τον αριθμό που επέλεξε. Αυτό το Map επιστρέφει στην callback συνάρτηση της διεπαφής που υλοποιεί η κύρια δραστηριότητα, η οποία με τη σειρά της στέλνει την ανάλογη αίτηση στο διακομιστή με τη χρήση ενός αντίστοιχου AsyncTask.

Ανεξάρτητα με τον τρόπο που στάλθηκε η αίτηση για την κράτηση των εισιτηρίων, το JSON αντικείμενο που επιστρέφεται από το διακομιστή είναι μία λίστα με τα ονόματα των εισιτηρίων. Σε περίπτωση που δεν μπορέσει να γίνει η κράτηση εμφανίζεται ανάλογο Toast μήνυμα. Ελέγχεται το status code της απάντησης και αν αυτό είναι **200**, τότε η κύρια δραστηριότητα αφαιρεί το fragment με τα εισιτήρια/θέσεις και προσθέτει το fragment με την παρουσίαση των κλεισμένων εισιτηρίων, περνώντας σε αυτό το JSON με τα εισιτήρια που επέστρεψε ο διακομιστής. Το fragment εμφανίζει τα κλεισμένα εισιτήρια ανά κατηγορία. Αφού ο χρήστης διαβεβαιώσει την επιλογή προχωράει στην ολοκλήρωση της κράτησης με την επιλογή του κουμπιού CHECKOUT, το οποίο καλεί την callback συνάρτηση της

κύριας δραστηριότητας, η οποία ξεκινάει τη δραστηριότητα ολοκλήρωσης της κράτησης.



ΕΙΚΟΝΑ 4.12 – ΤΟ FRAGMENT ΜΕ ΤΑ ΚΡΑΤΗΜΕΝΑ ΕΙΣΙΤΗΡΙΑ

Όπως μπορούμε να περάσουμε δεδομένα σε ένα fragment, μπορούμε να περάσουμε και key/value ζευγάρια σε μία δραστηριότητα. Αφού δημιουργήσουμε το Intent που θα αρχίσει τη δραστηριότητα, καλώντας τη συνάρτηση **putExtra(String key, Int value)** του intent, ορίζουμε το όνομα του κλειδιού και την αντίστοιχη τιμή που θέλουμε να περάσουμε. Έπειτα με τη συνάρτηση **startActivity(Intent intent)**, ξεκινάμε τη νέα δραστηριότητα, μέσα στην οποία καλώντας τη συνάρτηση **getIntent().getExtra(String key)**, λαμβάνουμε την τιμή που περάστηκε πριν την ξεκινήσουμε. Με αυτό τον τρόπο περνάμε το id της κράτησης που επέστρεψε ο διακομιστής με την προηγούμενη αίτηση, καθώς και την τελική τιμή αυτής.

CALLBACK ΣΥΝΑΡΤΗΣΗ ΓΙΑ ΕΚΚΙΝΗΣΗ ΔΡΑΣΤΗΡΙΟΤΗΤΑΣ

```
public void onCheckoutPressed(int resId, int totalPrice){
    Intent i = new Intent(this, PaymentActivity.class);
    i.putExtra("resId",resId);
    i.putExtra("price",totalPrice);
```

```
    startActivity(i);
}
```

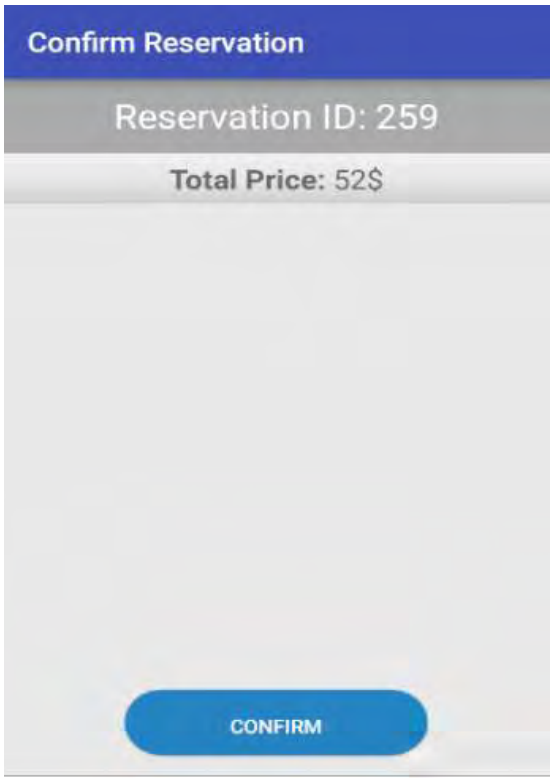
Τέλος η κύρια δραστηριότητα παρέχει ένα μενού μέσω του οποίου ο χρήστης μπορεί να κάνει **Log Out.** Η δραστηριότητα προσφέρει ένα action bar, στο οποίο μπορούμε να βάλουμε δικές μας λειτουργίες. Για να εμφανίσουμε το action bar κάνουμε extend την **AppCompatActivity** κλάση, στην κλάση της κύριας δραστηριότητας. Στο action bar εμφανίζεται το όνομα της δραστηριότητας, το οποίο είναι ορισμένο στο Android Manifest, καθώς και ένα κουμπί μενού που ορίζουμε εμείς. Το μενού αυτό βρίσκεται δίπλα από το όνομα της δραστηριότητας και περιλαμβάνει την επιλογή Log Out. Το μενού αυτό περιγράφεται σε ένα xml αρχείο το οποίο φορτώνουμε στην συνάρτηση **onCreateOptionsMenu()** της κύριας δραστηριότητας. Κάθε επιλογή του μενού φέρει ένα μοναδικό id, το οποίο ελέγχουμε για να αποφασίσουμε ποια επιλογή πατήθηκε. Όταν επιλέγεται ένα αντικείμενο του μενού καλείται η συνάρτηση **onOptionsItemSelected()** η οποία φέρει πληροφορίες για την επιλογή που πατήθηκε. Από αυτές ελέγχουμε το μοναδικό id και τρέχουμε το ανάλογο κομμάτι κώδικα. Στην περίπτωση μας κάνουμε log out τον χρήστη. Όπως αναφέρθηκε δεν υπάρχει μόνιμη σύνδεση με την υπηρεσία web, αλλά μία εικονική σύνδεση η οποία βασίζεται στο αποθηκευμένο token στα SharedPreferences. Μόλις ο χρήστης πατήσει το log out, τότε καθαρίζονται τα **SharedPreferences** από το token και το username του και οδηγείται στη δραστηριότητα σύνδεσης.

ΔΗΜΙΟΥΡΓΙΑ OPTIONSMENU ΣΤΗΝ ΚΥΡΙΑ ΔΡΑΣΤΗΡΙΟΤΗΤΑ

```
public boolean onCreateOptionsMenu(Menu menu){

    //Φόρτωση xml αρχείου που περιγράφει το menu
    getMenuInflater().inflate(R.menu.main_menu, menu);
    String user = sharedPref.getString("username",null);
    if(user!=null){
        //Περικοπή του username
        if(user.length()>11) {
            user = user.substring(0, 10);
            user += "...";
        }
        menu.findItem(R.id.logStatusMenu).setTitle("Log Out: "+user);
    }
    return true;
}
```

Η επόμενη δραστηριότητα που συναντάμε είναι η δραστηριότητα επιβεβαίωσης. Στη δραστηριότητα αυτή δίνεται η δυνατότητα στο χρήστη να ολοκληρώσει μία ανοιχτή κράτηση. Το Layout αποτελείται από δύο **TextViews** τα οποία εμφανίζουν το id της κράτησης, καθώς και το τελικό κόστος της. Έπειτα με το κουμπί CONFIRM, στέλνεται η αίτηση **POST** στο διακομιστή με το id της κράτησης και εφόσον είναι ακόμα έγκυρη την οριστικοποιεί, αλλιώς εμφανίζεται ανάλογο Toast μήνυμα. Αφού οριστικοποιηθεί η κράτηση ξαναρχίζουμε την κύρια δραστηριότητα.

Υπάρχουν δύο τρόποι πρόσβασης σε αυτή τη δραστηριότητα. Ο πρώτος, ο οποίος περιγράφηκε στην προηγούμενη παράγραφο, είναι μέσω του fragment των κρατημένων εισιτηρίων και την επιλογή του κουμπιού CHECKOUT. Ο δεύτερος είναι μέσω του fragment των κρατήσεων. Με τη χρήση της συνάρτησης **setOnItemClickListener()** στο **ListView** που περιέχει τις κρατήσεις, μπορούμε να ορίσουμε έναν listener, ο οποίος θα τρέχει όταν ο χρήστης επιλέγει ένα αντικείμενο της λίστας. Η συνάρτηση **onItemClick()** του listener αυτού, μας επιστρέφει το id που είναι συσχετισμένο με το επιλεγμένο αντικείμενο, το οποίο έχουμε ορίσει από τον αντάπτορα ως το id της κράτησης, Επίσης επιστρέφεται η θέση του αντικειμένου, από την οποία παίρνουμε πρόσβαση στο JSON αντικείμενο με τη χρήση της συνάρτησης **getItemAtPosition()** στο **ListView** και εξάγουμε το κόστος της κράτησης. Έπειτα τα δεδομένα αυτά στέλνονται σε μία callback συνάρτηση ενός interface που υλοποιεί η κύρια δραστηριότητα και από εκεί ξεκινάμε τη δραστηριότητα επιβεβαίωσης.



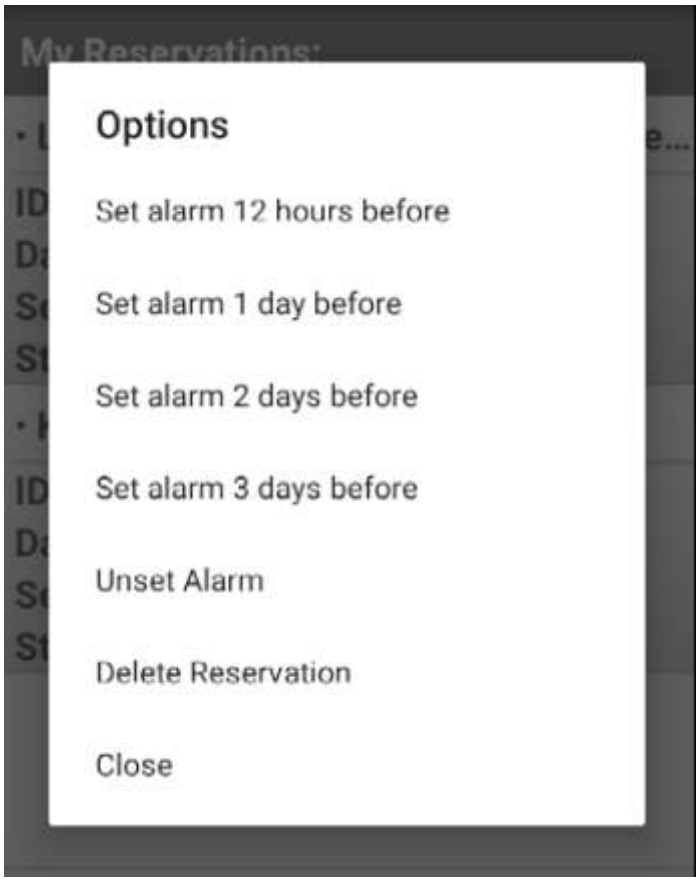
ΕΙΚΟΝΑ 4.13 – ΔΡΑΣΤΗΡΙΟΤΗΤΑ ΕΠΙΒΕΒΑΙΩΣΗΣ

Επιπλέον στην εφαρμογή αυτή, ο χρήστης έχει τη δυνατότητα να ορίσει μία υπενθύμιση για μία κλεισμένη κράτηση που έχει πραγματοποιήσει. Η λειτουργία αυτή προσφέρεται στο fragment των κρατήσεων, με τη χρήση ενός **Context Menu.** Το context menu είναι ένα pop-up menu το οποίο εμφανίζεται πάνω σε ένα view όταν αυτό πατηθεί παρατεταμένα. Όπως και το option menu, ορίζεται σε ένα xml αρχείο το οποίο περιέχει ένα μοναδικό id για κάθε επιλογή του context μενού, καθώς και το όνομα αυτής που θα εμφανίζεται στο χρήστη.

ΤΟ XML ΑΡΧΕΙΟ ΤΟΥ CONTEXTMENU

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

    <item android:id="@+id/SET_ALARM_TWELVE_HOURS"
        android:title="Set alarm 12 hours before">
    </item>
    <item android:id="@+id/SET_ALARM_ONE_DAY"
        android:title="Set alarm 1 day before">
    </item>
    <item android:id="@+id/SET_ALARM_TWO_DAYS"
        android:title="Set alarm 2 days before">
    </item>
    <item android:id="@+id/SET_ALARM_THREE_DAYS"
        android:title="Set alarm 3 days before">
    </item>
    <item android:id="@+id/CANCEL_ALARM"
        android:title="Unset Alarm">
    </item>
    <item android:id="@+id/CANCEL_RESERVATION"
        android:title="Delete Reservation">
    </item>
    <item android:id="@+id/CLOSE_MENU"
        android:title="Close">
    </item>

</menu>
```

Με τη χρήση της συνάρτησης **registerForContextMenu()** στην **onActivityCreated()**, περνάμε ως όρισμα το ListView που περιέχει τις κρατήσεις, ώστε με το παρατεταμένο πάτημα ενός αντικειμένου του ListView να εμφανίζεται το context menu. Στη συνάρτηση **onCreateContextMenu()** φορτώνουμε το xml αρχείο του context menu. Μόλις επιλεχθεί μία επιλογή του context menu καλείται η **onContextItemSelected(Menu item)**, η οποία φέρνει ένα αντικείμενο Menu που περιέχει πληροφορίες με την επιλογή που πατήθηκε. Από αυτό βρίσκουμε το μοναδικό id της επιλογής που αντιστοιχεί στο context menu, καθώς και το ποιο αντικείμενο του ListView επιλέχθηκε και εκτελούμε τον ανάλογο κώδικα.



ΕΙΚΟΝΑ 4.14 – ΤΟ CONTEXT MENU

Οι επιλογές που παρέχονται στο χρήστη είναι ένα εύρος τιμών για τον ορισμό της υπενθύμισης, η ακύρωση της υπενθύμισης, η διαγραφή της κράτησης και το κλείσιμο του menu. Με την επιλογή της διαγραφής στέλνεται μέσω ενός AsyncTask μία αίτηση DELETE στο URL με το id της κράτησης (reservations/id). Για τον ορισμό μίας υπενθύμισης χρησιμοποιείται ένα **PendingIntent** το οποίο μεταδίδεται σε έναν δέκτη μετάδοσης (Broadcast Receiver). Τα pending intents είναι περιτυλίγματα ενός κανονικού intent τα οποία μπορούν να εκτελεστούν σε κάποια χρονική στιγμή στο μέλλον. Αρχικά δημιουργούμε ένα Intent το οποίο θα έχει ως πρόθεση την εκκίνηση του δέκτη μετάδοσης. Στο intent αυτό περνάμε ως δεδομένα το όνομα της διοργάνωσης, την ημερομηνία και το id της κράτησης. Έπειτα με τη χρήση της συνάρτησης **PendingIntent.getBroadcast()**, στην οποία περνάμε ως ορίσματα το context που τρέχει, ένα id που χαρακτηρίζει το pending intent, το intent που θέλουμε να τρέξουμε στο μέλλον, καθώς και το flag FLAG_UPDATE_CURRENT, με το οποίο βεβαιώνουμε ότι άμα υπάρχει ήδη pending intent με το συγκεκριμένο id (πχ ο χρήστης άλλαξε την ώρα της υπενθύμισης), δεν θα δημιουργηθεί δεύτερο intent με το ίδιο id αλλά θα ενημερωθούν οι πληροφορίες του ήδη υπάρχουν. Για να τρέξουμε αυτό το Pending Intent στη ζητούμενη ημερομηνία χρησιμοποιούμε τον **AlarmManager**. Αφού πάρουμε reference του Alarm Manager καλούμε τη συνάρτηση **set**(), στην οποία περνάμε τον τύπο υπενθύμισης, ο οποίος ορίζει πληροφορίες σχετικά με το πώς θα μεταβιβαστεί το alarm, την ημερομηνία που

θέλουμε να τρέξει το intent η οποία υπολογίζεται προσθέτοντας στην τωρινή ώρα την επιλογή του χρήστη, καθώς και το ίδιο το pending intent. Αφού οριστεί η υπενθύμιση εμφανίζεται το αντίστοιχο εικονίδιο δίπλα από τη κράτηση.

ΕΙΚΟΝΑ 4.15 – ΟΡΙΣΜΟΣ ΜΙΑΣ ΕΙΔΟΠΟΙΗΣΗΣ

ΟΡΙΣΜΟΣ ΥΠΕΝΘΥΜΙΣΗΣ

```
//Πρόσβαση στον AlarmManager
final AlarmManager mAlarmManager = (AlarmManager)
    getActivity().getSystemService(Context.ALARM_SERVICE);

//Δημιουργία Intent και προσθήκη πληροφοριών διοργάνωσης
final Intent intentToBroadcast = new
    Intent(getActivity(),AlarmNotificationReceiver.class);
intentToBroadcast.putExtra("date",date);
intentToBroadcast.putExtra("id",id);
intentToBroadcast.putExtra("title", title);

//Μετατροπή της ημερομηνίας διοργάνωσης σε αντικείμενο Calendar
Calendar cal = Calendar.getInstance();

//Σπάσιμο ημερομηνίας από μορφή YYYY/MM/DD
cal.set(Calendar.YEAR,Integer.parseInt(date.substring(0,4)));
cal.set(Calendar.MONTH,Integer.parseInt(date.substring(5,7))-1);
cal.set(Calendar.DAY_OF_MONTH,Integer.parseInt(date.substring(8,10)));
```

```
//Σπάσιμο της ώρας από μορφή HH:MM:SS
cal.set(Calendar.HOUR_OF_DAY,Integer.parseInt(date.substring(11,13)));
cal.set(Calendar.MINUTE,Integer.parseInt(date.substring(14,16)));
cal.set(Calendar.SECOND,Integer.parseInt(date.substring(17)));

//Έλεγχος επιλογής context menu
switch (item.getItemId()) {
    case R.id.SET_ALARM_TWELVE_HOURS:

        //Αν είναι ολοκληρωμένη κράτηση, θέσε την υπενθύμιση
        if(status.equals("closed")) {
            //Δημιουργία PendingIntent
            PendingIntent mPending =
                PendingIntent.getBroadcast(getActivity(),
                id,intentToBroadcast,PendingIntent.FLAG_UPDATE_CURRENT);

            //Δημιουργία alarm δώδεκα ώρες πριν τη διοργάνωση
            mAlarmManager.set(AlarmManager.RTC_WAKEUP,
                cal.getTimeInMillis() - (12 * 60 * 60 * 1000L), mPending);

            //Εμφάνιση εικονιδίου ειδοποίησης
            alarmIcon.setVisibility(View.VISIBLE);
            Toast.makeText(getActivity(), "Alarm set
                12 hours prior to event", Toast.LENGTH_SHORT).show();
            //Εισαγωγή alarm στη βάση δεδομένων **Αναλύεται αργότερα**
            new Thread(new Runnable() {
                @Override
                public void run() {
                    dbManager.insertAlarm(id, date, 1, title);
                }
            }).start();
        }else{
            Toast.makeText(getActivity(), "Cannot set alarm on an open
                reservation", Toast.LENGTH_SHORT).show();
        }
        return true;
    case R.id.SET_ALARM_ONE_DAY:
        ...

}
```

Αφού ενεργοποιηθεί το alarm, όταν έρθει η στιγμή, αυτό τρέχει το pending intent που ορίσαμε, δηλαδή ξεκινάει το δέκτη μετάδοσης. Ο δέκτης μετάδοσης είναι υπεύθυνος για την εμφάνιση της ειδοποίησης στο χρήστη. Η ειδοποίηση αυτή δημιουργείται με τη χρήση της κλάσης **Notification**. Η κλάση αυτή μας παρέχει έναν **Builder** με τον οποία ορίζουμε τα βασικά στοιχεία ενός Notification. Το Notification είναι μία ειδοποίηση που εμφανίζεται στο πάνω μέρος της συσκευής. Καθώς το notification δημιουργείται από το δέκτη μετάδοσης ο οποίος τρέχει στο παρασκήνιο, δεν απαιτείται η εφαρμογή να είναι ανοιχτή, ώστε να σταλεί. Μέσω του Builder θέτουμε τον τίτλο, το εικονίδιο και το μήνυμα του notification. Έπειτα με τη χρήση του **NotificationManager** και την συνάρτηση **notify()**, στην οποία περνάμε το Notification που φτιάξαμε, στέλνει την ειδοποίηση στο χρήστη. Ο χρήστης μπορεί να κλείσει την ειδοποίηση αυτή με την επιλογή της.

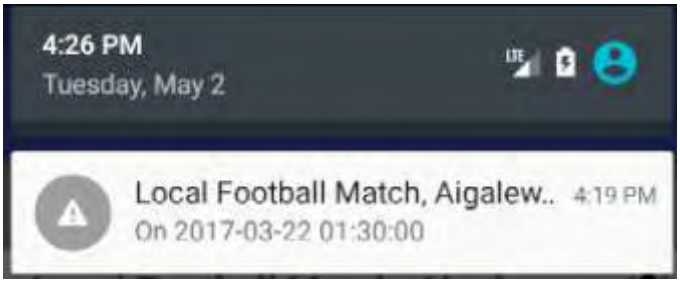
ΑΠΟΣΤΟΛΗ NOTIFICATION ΜΕΣΩ ΤΟΥ BROADCAST RECEIVER

```
public void onReceive(Context context, Intent intent) {

    //Άδειο PendingIntent ώστε να κλείσει το Notification μόλις το επιλέξει
    χρήστης
    PendingIntent contentIntent = PendingIntent.getActivity(
            context,
            0,
            new Intent(),
            PendingIntent.FLAG_UPDATE_CURRENT);

    //Δημιουργία Notification
    Notification.Builder notificationBuilder = new
        Notification.Builder(context)
            .setTicker("You have an upcoming event.")
            .setContentTitle(intent.getStringExtra("title"))
            .setSmallIcon(android.R.drawable.stat_sys_warning)
            .setAutoCancel(true)
            .setContentText("On "+intent.getStringExtra("date"))
            .setDefaults(Notification.DEFAULT_SOUND |
                 Notification.DEFAULT_VIBRATE)
            .setContentIntent(contentIntent);

    //Αποστολή Notification
    NotificationManager mNotificationManager = (NotificationManager)
        context.getSystemService(Context.NOTIFICATION_SERVICE);
    mNotificationManager.notify(1, notificationBuilder.build());

    //Διαγραφή από τη βάση δεδομένων **Αναλύεται αργότερα**
    final SQLiteAlarmManager dbManager = new SQLiteAlarmManager(context);
    final int id = intent.getIntExtra("id",0);
    new Thread(new Runnable() {
        @Override
        public void run() {
            dbManager.deleteAlarm(id);
        }
    }).start();

}
```

Αν ο χρήστης θέλει να ακυρώσει μία υπενθύμιση που έχει ορίσει, τότε επιλέγει το unset στη συγκεκριμένη κράτηση. Αφού το id του pending intent είναι ίδιο με το id της κράτησης, το μόνο που χρειάζεται για να το ακυρώσουμε είναι να δημιουργήσουμε ένα pending intent με το ίδιο id και να το περάσουμε στη συνάρτηση **cancel()** του **AlarmManager**.

ΑΚΥΡΩΣΗ ΥΠΕΝΘΥΜΙΣΗΣ ΜΕΣΩ ΤΟΥ CONTEXT MENU

```
case R.id.CANCEL_ALARM:

    //Δημιουργία PendingIntent με ίδιο id ώστε να το ακυρώσουμε
    PendingIntent mPending = PendingIntent.getBroadcast(getActivity(),
        id,intentToBroadcast,PendingIntent.FLAG_UPDATE_CURRENT);
    mAlarmManager.cancel(mPending);
    mPending.cancel();

    //Απόκρυψη εικονιδιου ειδοποίησης και διαγραφή από τη βάση **Αναύεται
αργότερα**

    alarmIcon.setVisibility(View.GONE);
    new Thread(new Runnable() {
        @Override
        public void run() {
            dbManager.deleteAlarm(id);
        }
    }).start();
    return true;
```



ΕΙΚΟΝΑ 4.16 – Η ΕΙΔΟΠΟΙΗΣΗ ΓΙΑ ΜΙΑ ΔΙΟΡΓΑΝΩΣΗ

Τα alarms που δημιουργούμε με τον AlarmManager διαγράφονται όταν η συσκευή απενεργοποιηθεί. Αυτό απαιτεί την αποθήκευση των alarm σε μία βάση δεδομένων και τον ορισμό τους κάθε φορά που ανοίγει η συσκευή. Το Android μας προσφέρει το σύστημα διαχείρισης βάσεων δεδομένων SQLite. Σε αντίθεση με άλλα συστήματα διαχείρισης βάσης δεδομένων, το SQLite δεν είναι μια ξεχωριστή διεργασία που προσπελάζεται από μια εφαρμογή πελάτη, αλλά ένα ενσωματωμένο μέρος της. Για την αποθήκευση των δεδομένων χρησιμοποιούμε έναν πίνακα, ο οποίος έχει τρία πεδία:

- **id**: αποθηκεύει το id της κράτησης για την οποία ισχύει η υπενθύμιση και αποτελεί το πρωτεύον κλειδί του πίνακα
- **date**: η ημερομηνία της διοργάνωσης
- **type**: ένας ακέραιος από το 1 έως το 4, ο οποίος δηλώνει ποια από τις 4 επιλογές επέλεξε ο χρήστης
- **title**: το όνομα της διοργάνωσης

Το Android μας παρέχει ένα ευέλικτο τρόπο για να επεξεργαστούμε τη βάση, με την κλάση SQLiteOpenHelper την οποία κάνουμε extend και υλοποιούμε τις συναρτήσεις που μας προσφέρει. Στο constructor της κλάσης αυτής περνάμε το όνομα της βάσης δεδομένων, που θα χρησιμοποιήσουμε. Επιπλέον η κλάση αυτή μας προσφέρει τη συνάρτηση **onCreate(SQLiteDatabase db)**, η οποία μας παρέχει ένα αντικείμενο **SQLiteDatabase** με το οποίο μπορούμε να στείλουμε ένα ερώτημα στη βάση με τη συνάρτηση **db.execSQL()**. Η **onCreate()** τρέχει την πρώτη φορά που δημιουργείται η βάση δεδομένων από το σύστημα, οπότε εδώ δημιουργούμε τον πίνακα μας. Αφού δημιουργήσουμε τη βάση μπορούμε να αποκτήσουμε πρόσβαση σε αυτή δημιουργώντας ένα αντικείμενο SQLiteDatabase με τη χρήση των συναρτήσεων **getWriteableDatabase()/getReadableDatabase()** ανάλογα με το τι θέλουμε να κάνουμε.

Κάθε φορά που ο χρήστης ορίζει μία υπενθύμιση, τα στοιχεία της αποθηκεύονται στη βάση με τη συνάρτηση **insertAlarm(int id, String date, int type, String title)**, που έχουμε ορίσει, μέσα στην κλάση που κάνει extend τη **SQLiteOpenHelper**. Αρχικά λαμβάνουμε το αντικείμενο SQLiteDatabase ώστε να γράψουμε σε αυτό με τη χρήση της **getWriteableDatabase()**. Έπειτα ορίζουμε ένα αντικείμενο **ContentValues** στο οποίο προσθέτουμε με τη συνάρτηση **put()** ένα σύνολο από key/value ζευγάρια που αντιπροσωπεύουν την τιμή για κάθε πεδίο. Τέλος τα εισάγουμε στον πίνακα με τη συνάρτηση **replace()** του αντικειμένου SQLiteDatabase, η οποία δέχεται ως ορίσματα το όνομα του πίνακα και το αντικείμενο ContentValues. Σε περίπτωση που υπάρχει ήδη μία κράτηση με το ίδιο id τότε ενημερώνεται το πεδίο αυτό.

ΠΡΟΣΘΗΚΗ ΕΙΔΟΠΟΙΗΣΗΣ ΣΤΗ ΒΑΣΗ

```
protected boolean insertAlarm(int id, String date, int type, String title){
    SQLiteDatabase db = getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("id", id);
    contentValues.put("date", date);
    contentValues.put("type", type);
    contentValues.put("title", title);
    if(db.replace("alarms_table",null,contentValues) == -1){
        return false;
    }else{
        return true;
    }
}
```

Αφού παραδοθεί μία υπενθύμιση, ο δέκτης μετάδοσης εκτός από το να εμφανίζει το μήνυμα στο χρήστη, διαγράφει και την υπενθύμιση από τη βάση δεδομένων. Το intent που μεταδίδεται στο δέκτη μετάδοσης περιέχει το id της κράτησης το οποίο είναι το πρωτεύον κλειδί του πίνακα. Αυτή τη φορά χρησιμοποιούμε τη συνάρτηση **delete()** για να διαγράψουμε την εγγραφή, η οποία δέχεται το όνομα του πίνακα, τη συνθήκη διαγραφής με placeholders αντί τιμές και έναν πίνακα με τις τιμές των placeholders.

Όταν ανοίγει η συσκευή όλες οι υπενθυμίσεις διαγράφονται. Για να τις επαναφέρουμε χρησιμοποιούμε ένα δέκτη μετάδοσης, ο οποίος διαβάζει όλα τα

δεδομένα του πίνακα με τις υπενθυμίσεις και τις ορίζει ξανά με τον ίδιο τρόπο που ορίζονται όταν επιλεγεί το context menu. Ο δέκτης αυτός έχει οριστεί μέσω του Android Manifest να δέχεται intents που φέρουν το action "**android.intent.action.BOOT_COMPLETED**". Αυτό το intent το μεταδίδει το σύστημα όταν ενεργοποιείται η συσκευή. Με τη χρήση της κλάσης SQLiteOpenHelper που δημιουργήσαμε χρησιμοποιούμε τη συνάρτηση **getReadableDatabase()** για να πάρουμε ένα αντικείμενο **SQLiteDatabase.** Σε αυτό τρέχουμε το SQL ερώτημα για να επιλέξουμε όλες τις εγγραφές του πίνακα με της υπενθυμίσεις. Αυτό το πετυχαίνουμε με τη συνάρτηση rawQuery() του SQLiteDatabase, η οποία στέλνει ένα custom ερώτημα στη βάση. Αυτή μας επιστρέφει ένα αντικείμενο **Cursor**, που περιέχει τα δεδομένα της απάντησης, το οποίο προσπελαύνουμε και ορίζουμε ξανά τα pending intents και τα alarms για κάθε εγγραφή που επιστράφηκε από τη βάση ανάλογα με το δεδομένα της κάθε εγγραφής.

Όλες οι εγγραφές στη βάση γίνονται ασύγχρονα. Επειδή η είσοδος των υπενθυμίσεων στη βάση δεν έχει κάποια επιρροή στο interface της κύριας εφαρμογής και επειδή η συνάρτηση **getWritableDatabase()** μπορεί να αργήσει να επιστρέψει καθυστερώντας χωρίς λόγω το UI Thread, χρησιμοποιούμε νήματα ώστε να τρέξουμε τις συναρτήσεις της κλάσης που κάνει extend την SQLiteOpenHelper. Αντίθετα όταν δημιουργούμε το αντικείμενο **SQLiteOpenHelper** και καλούμε τον constructor του, αυτό δεν θα δημιουργήσει κατευθείαν τον πίνακα μέσω της **onCreate()**, σε περίπτωση που δεν έχει δημιουργηθεί, αλλά θα επιστρέψει κατευθείαν και ο πίνακας θα δημιουργηθεί την πρώτη φορά που θα γίνει χρήση της **getWritableDatabase()** ή **getReadableDatabase()**. Συνεπώς το αντικείμενο της κλάσης αυτής το φτιάχνουμε στο UI Thread.

4.4.5 Η ΕΦΑΡΜΟΓΗ ΤΩΝ ΔΙΟΡΓΑΝΩΤΩΝ

Σε αυτή την παράγραφο αναλύεται η εφαρμογή που απευθύνεται στους διοργανωτές. Μέσω αυτής ο χρήστης μπορεί διαχειριστεί τις διοργανώσεις που έχει δημοσιεύσει, καθώς και να δημιουργήσει καινούριες.

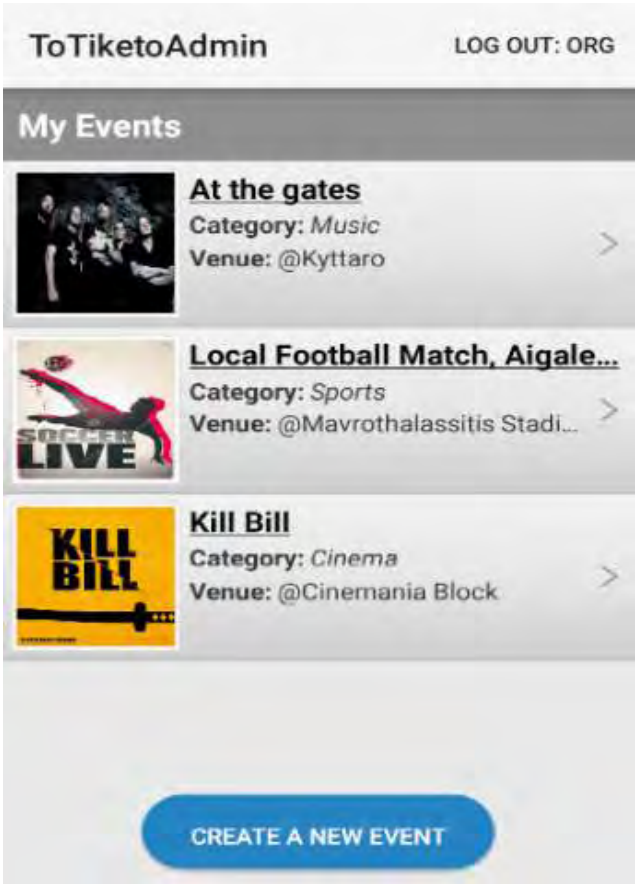
ΕΙΚΟΝΑ 4.17 – ΤΟ LOGO ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Ομοίως με την εφαρμογή των απλών χρηστών προσφέρεται ένα σύστημα εγγραφής και εισόδου στο σύστημα, το οποίο είναι πανομοιότυπο με αυτό της προηγούμενης εφαρμογής. Αφού συνδεθεί ο χρήστης αποθηκεύει το JWT στα SharedPreferences και το αποστέλλει με κάθε αίτηση του. Αυτή τη φορά η ιδιότητα του χρήστη που

στέλνεται ως παράμετρος κατά την είσοδο/εγγραφή έχει την τιμή "**organizer**", καθώς και το token που στέλνει ο διακομιστής περιέχει την ανάλογη πληροφορία για την ιδιότητα αυτή.

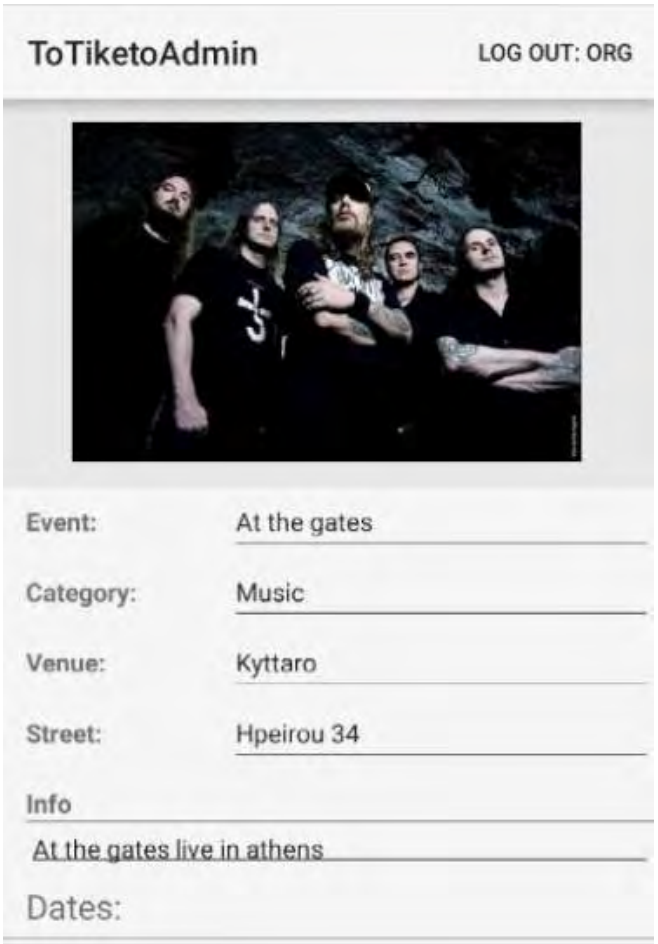
Όπως και πριν χρησιμοποιείται η αρχιτεκτονική της μίας δραστηριότητας με πολλαπλά fragment για την υλοποίηση της κύριας δραστηριότητας. Η κύρια δραστηριότητα, η οποία εμφανίζεται μετά την είσοδο του χρήστη στο σύστημα, παρέχει ένα **FrameLayout** που καταλαμβάνει ολόκληρη την οθόνη και σε αυτό εναλλάσσουμε τα διάφορα fragments.

Μόλις ο χρήστης εισαχθεί στο σύστημα τότε η κύρια δραστηριότητα με τη χρήση ενός **AsyncTask** στέλνει μία αίτηση **GET** στο διακομιστή ώστε να λάβει τις πληροφορίες για τις διοργανώσεις του χρήστη. Αφού επιστραφεί το αντικείμενο JSON με τις πληροφορίες, τότε εισάγει το fragment των διοργανώσεων στο FrameLayout. Το fragment αυτό περιλαμβάνει ένα **ListView**, του οποίου κάθε αντικείμενο περιέχει βασικές πληροφορίες για κάθε διοργάνωση του χρήστη. Τα αντικείμενα του **ListView**, διαχειρίζονται από έναν αντάπτορα ο οποίος δέχεται το JSON αντικείμενο που επέστρεψε ο διακομιστής και παρέχει το ανάλογο view για κάθε διοργάνωση της λίστας. Τέλος το fragment αυτό προσφέρει ένα κουμπί με το οποίο ο χρήστης μπορεί να δημιουργήσει μία νέα διοργάνωση.



ΕΙΚΟΝΑ 4.18 – ΤΟ FRAGMENT ΤΩΝ ΔΙΟΡΓΑΝΩΣΕΩΝ ΤΟΥ ΧΡΗΣΤΗ

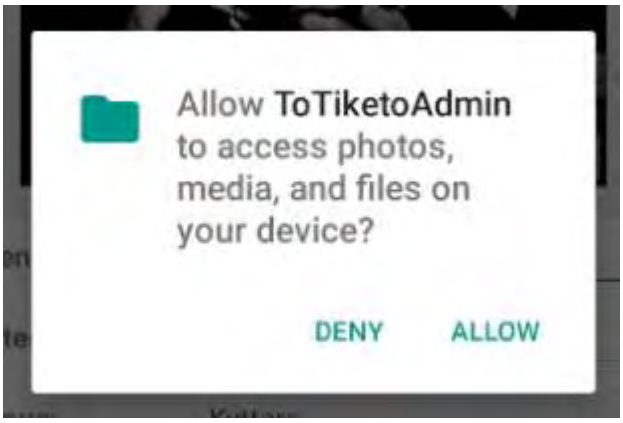
Με τη χρήση ενός **OnClickListener()** που εφαρμόζουμε στο **ListView**, μόλις ο χρήστης πατήσει μία διοργάνωση τότε το id της στέλνεται σε μία callback συνάρτηση που υλοποιεί η κύρια δραστηριότητα, η οποία στέλνει μία αίτηση **GET** στο διακομιστή για να πάρει παραπάνω πληροφορίες σχετικά με τη διοργάνωση αυτή. Αν η αίτηση είναι επιτυχής, δηλαδή ο διακομιστής απαντήσει με κωδικό "**200**", τότε η κύρια δραστηριότητα αφαιρεί το τωρινό fragment και εισάγει το fragment της επεξεργασίας. Μέσα από αυτό το fragment ο χρήστης μπορεί να δει πληροφορίες σχετικά με την προπώληση της διοργάνωσης του, καθώς και να επεξεργαστεί διάφορα στοιχεία της. Όλες οι αιτήσεις στέλνονται μέσω ένα **AsyncTask** που είναι υλοποιημένο στην κλάση του fragment, το οποίο χρησιμοποιεί την **multiPartRequest()** της **WebServiceAPI**, η οποία περιγράφηκε στην παράγραφο 4.4.3, για την εξυπηρέτηση όλων των αιτήσεων, που στέλνονται από αυτό το fragment. Αυτό μπορεί να χωριστεί σε 4 κομμάτια, τα οποία είναι οι αλλαγές των βασικών πληροφοριών της διοργάνωσης, η προσθήκη μίας νέας ημερομηνίας, η αφαίρεση μίας ημερομηνίας και η διαγραφή της διοργάνωσης.



ΕΙΚΟΝΑ 4.19 – ΤΟ FRAGMENT ΕΠΕΞΕΡΓΑΣΙΑΣ 1/2

Ο χρήστης μέσα από τα EditText views που προσφέρονται μπορεί να αλλάξει τις βασικές πληροφορίες που θέλει για την διοργάνωση, δηλαδή το όνομα, την

κατηγορία, το χώρο, τη διεύθυνση και τις πληροφορίες. Επιπλέον του δίνεται η δυνατότητα να αλλάξει και την εικόνα της διοργάνωσης, εξού και η χρήση του multipart request. Με την επιλογή της εικόνας, η οποία αποτελεί ένα **ImageButton** view, ο χρήστης μπορεί να περιηγηθεί στις εικόνες του και να επιλέξει ποια θέλει. Για την πρόσβαση στα δεδομένα στα αποθηκευμένα δεδομένα του χρήστη το Android απαιτεί τις κατάλληλες άδειες ώστε να επιτρέψει την πρόσβαση. Ανάλογα με την έκδοση του Android που χρησιμοποιείται χρησιμοποιούνται 2 τρόποι. Για τις εκδόσεις που είναι μικρότερες από το Android 6.0, αρκεί να ζητήσουμε τις άδειες αυτές κατά την εγκατάσταση της εφαρμογής. Αυτό γίνεται μέσω του Android Manifest και την εισαγωγή του στοιχείου <user-permissions> το οποίο θα περιλαμβάνει τις απαιτούμενες άδειες, στην περίπτωση μας την **android.permission.READ_EXTERNAL_STORAGE**. Ο χρήστης πληροφορείται ότι η εφαρμογή χρησιμοποιεί τις παραπάνω άδειες κατά την εγκατάσταση και αναλόγως την επιτρέπει ή όχι. Οι συσκευές που χρησιμοποιούν εκδόσεις μεγαλύτερες από την Android 6.0, αιτούνται τις άδειες αυτές κατά την εκτέλεση της εφαρμογής. Αρχικά ελέγχουμε αν έχουν δοθεί αυτές οι άδειες με τη συνάρτηση **checkSelfPermission()**, η οποία δέχεται ως όρισμα το context που τρέχει, την άδεια που θέλουμε να ελέγξουμε και έναν αναγνωριστικό κωδικό. Αν δεν υπάρχουν τότε τις ζητάμε με τη συνάρτηση **requestPermission()**, η οποία δέχεται το context που τρέχει και τις άδειες που θέλουμε. Η αίτηση γίνεται ασύγχρονα, αλλά επιστρέφει σχεδόν αμέσως και εμφανίζει ένα διάλογο στον οποίο ζητάει την έγκριση του χρήστη.



ΕΙΚΟΝΑ 4.20 – ΠΑΡΟΧΗ ΑΔΕΙΩΝ ΚΑΤΑ ΤΗΝ ΕΚΤΕΛΕΣΗ

Μόλις δώσει ο χρήστης την έγκριση του τρέχει η callback συνάρτηση **onRequestPermissionsResult()**, η οποία παρέχει πληροφορίες με το αποτέλεσμα της αίτησης. Ελέγχουμε αν η άδεια δόθηκε και έπειτα ξεκινάμε μία δραστηριότητα η οποία έχει ως λειτουργία την επιλογή και επιστροφή εικόνας. Αυτό γίνεται με τη χρήση ενός Intent το οποίο δεν προσδιορίζει την κλάση που θα τρέξει, αλλά περιγράφει την ενέργεια που θέλει να κάνει.

ΑΙΤΗΣΗ ΑΔΕΙΩΝ ΚΑΤΑ ΤΗΝ ΕΚΤΕΛΕΣΗ

```
//Έλεγχος αν έχει δωθεί η άδεια
if (ContextCompat.checkSelfPermission(getActivity(),
    android.Manifest.permission.READ_EXTERNAL_STORAGE) !=
    PackageManager.PERMISSION_GRANTED) {

    //Αν δεν έχει δωθεί, ζήτα τις άδειες από το σύστημα
    ActivityCompat.requestPermissions(getActivity(), new
        String[]{android.Manifest.permission.READ_EXTERNAL_STORAGE}, 200 );
}else{

    //Αν έχει δωθεί τότε άρχισε τη δραστηριότητα επιλογής εικόνας
    Intent photoPickerIntent = new Intent(Intent.ACTION_PICK);
    photoPickerIntent.setType("image/*");
    startActivityForResult(photoPickerIntent, SELECT_PHOTO);
}
```

CALLBACK ΣΥΝΑΡΤΗΣΗ ΓΙΑ ΟΤΑΝ ΔΩΘΟΥΝ ΑΔΕΙΕΣ

```
public void onRequestPermissionsResult(int requestCode, @NonNull String
permissions[], @NonNull  int[] grantResults){

    switch(requestCode){
        case 200:
            if(grantResults.length > 0 && grantResults[0] ==
                PackageManager.PERMISSION_GRANTED){

                Intent photoPickerIntent = new Intent(Intent.ACTION_PICK);
                photoPickerIntent.setType("image/*");
                startActivityForResult(photoPickerIntent, SELECT_PHOTO);
            }else{
                Toast.makeText(getActivity(), "Can't access Gallery",
                    Toast.LENGTH_SHORT).show();
            }
    }
}
```

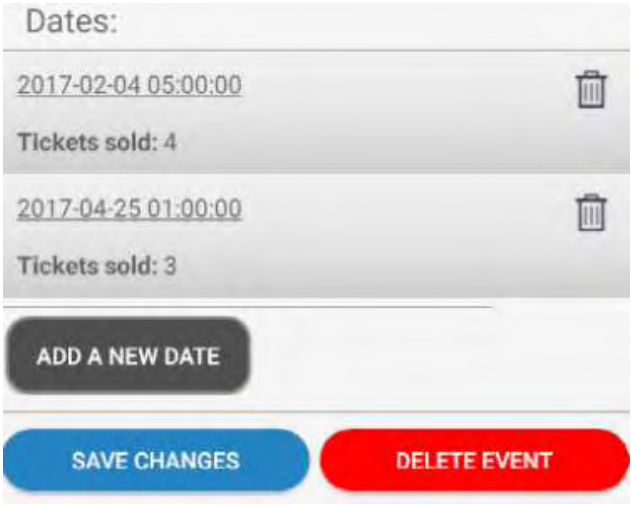
Για να αρχίσουμε μία δραστηριότητα η οποία επιστρέφει ένα αποτέλεσμα σε αυτήν που την άρχισε χρησιμοποιούμε την συνάρτηση **startActivityForResult()**, περνώντας το intent και ένα αναγνωριστικό κωδικό. Έπειτα ανοίγει ένα μενού στο χρήστη με όλες της εφαρμογές/δραστηριότητες που μπορούν να ικανοποιήσουν αυτή την ενέργεια, όπως πχ η συλλογή φωτογραφιών του χρήστη. Επίσης διευκρινίζοντας το mime type του intent με τη χρήση της συνάρτησης **setType()**, περιορίζουμε τα αρχεία που μπορούν να επιλεγούν μόνο σε εικόνες, καθώς και "σημαδεύουμε" εφαρμογές που προσφέρουν επιλογή εικόνας. Αφού επιλέξει ο χρήστης την εικόνα που θέλει τότε η νέα δραστηριότητα τερματίζεται και το αποτέλεσμα της επιστρέφει ως ένα intent στην callback συνάρτηση **onActivityResult()** του fragment. Σε αυτήν ελέγχουμε το κωδικό που περάσαμε και εξάγουμε το path της εικόνας από το επιστραμμένο intent, στέλνοντας ένα ερώτημα σε έναν **ContentProvider** που περιέχει πληροφορίες για τις αποθηκευμένες εικόνες του συστήματος. Έπειτα με τη

συνάρτηση **File** δημιουργούμε ένα αρχείο από το path αυτό, το οποίο μετατρέπουμε στο τελικό **Bitmap**. Η νέα εικόνα εμφανίζεται προσωρινά στη θέση της παλιάς και με την επιλογή του κουμπιού SAVE CHANGES στέλνονται όλα τα δεδομένα στο διακομιστή, αφού μετατραπούν στην κατάλληλη multipart/form-data δομή.

ΦΟΡΤΩΣΗ ΕΙΚΟΝΑΣ ΑΠΟ ΤΟ ΕΠΙΣΤΡΑΜΜΕΝΟ INTENT ΣΤΗΝ onActivityResult()

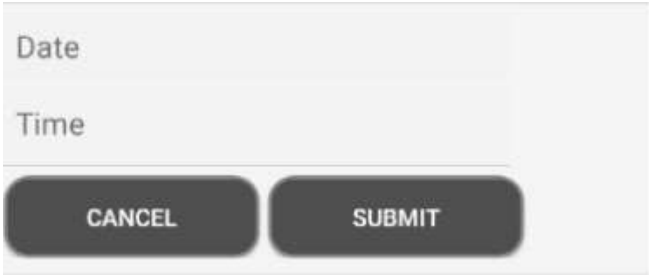
```
//Εξαγωγή Uri από το επιστραμμένο intent
Uri selectedImage = fileReturnedIntent.getData();

//Δημιουργία προβολής για ερώτηση του ContentProvider
String[] filePathColumn = {MediaStore.Images.Media.DATA};

//Ερώτηση στον ContentProvider για το path της εικόνας
Cursor cursor = getContentResolver().query(
        selectedImage, filePathColumn, null, null, null);

cursor.moveToFirst();

int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
String filePath = cursor.getString(columnIndex);
cursor.close();

//Δημιουργία αρχείου που αντπροσωπεύει την εικόνα
File f = new File(filePath);

//Έλεγχος μεγέθους
if(f.length()>1024*1024*10){
    Toast.makeText(CreateEventActivity.this, "Image exceeds the size limit
        allowed.", Toast.LENGTH_SHORT).show();
}else{

    //Δημιουργία bitmap από το path της εικόνας και θέσιμο στο αντίστοιχο
    ImageButton
n
    Bitmap previewImage = BitmapFactory.decodeFile(filePath);
    previewBtn.setImageBitmap(previewImage);
}
```

Κάτω από τις βασικές πληροφορίες της διοργάνωσης βρίσκονται οι διαθέσιμες ημερομηνίες καθώς και η προπώληση για κάθε μία. Ο χρήστης πατώντας το αντίστοιχο εικονίδιο μπορεί να διαγράψει μία ημερομηνία με την προϋπόθεση ότι δεν υπάρχει έγκυρη κράτηση. Επειδή χρησιμοποιείται ένα AsyncTask το οποίο καλεί την **multiPartRequest()**, περνάμε στις παραμέτρους που δεν μας ενδιαφέρουν την τιμή null. Μέσα στη συνάρτηση ελέγχονται ποιες μεταβλητές είναι null και δεν προσθέτονται στην multipart αίτηση.



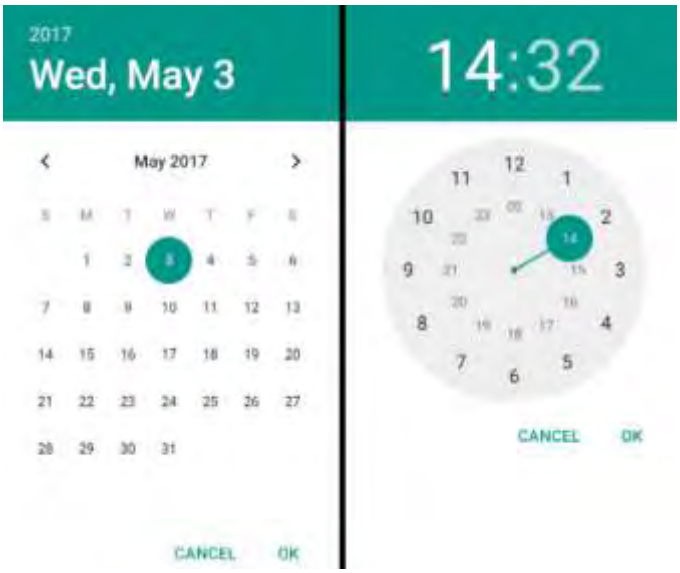
ΕΙΚΟΝΑ 4.22 – ΤΟ FRAGMENT ΤΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ 2/2

Με την επιλογή του **ADD A NEW DATE** εμφανίζονται δύο EditText πεδία στα οποία ο χρήστης συμπληρώνει την ημερομηνία και την ώρα που θέλει να προσθέσει. Αυτό γίνεται με τη βοήθεια δύο παραθύρων διαλόγου που μας προσφέρει το Android. Αυτά είναι το **DatePickerDialog** και το **TimePickerDialog**, για την ημερομηνία και την ώρα αντίστοιχα. Αυτά εμφανίζουν ένα ημερολόγιο και ένα ρολόι από τα οποία ο χρήστης επιλέγει τις τιμές οι οποίες εισάγονται έτοιμες στα αντίστοιχα EditText τους.



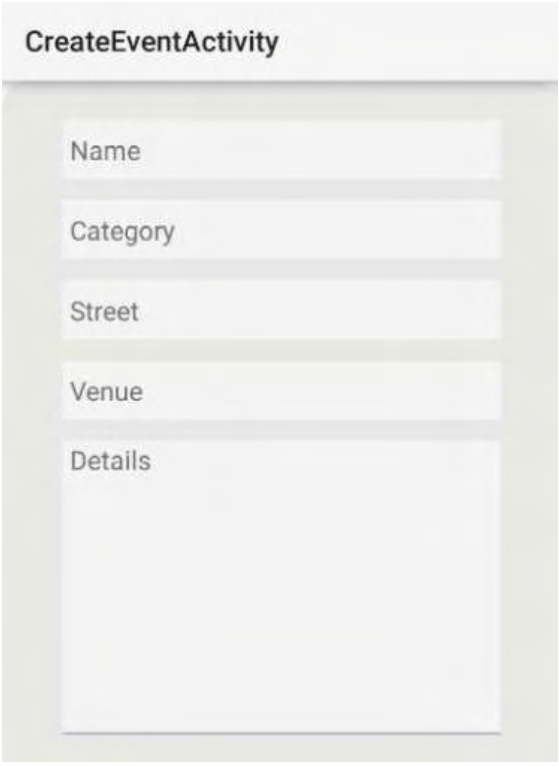
ΕΙΚΟΝΑ 4.23 – ΠΡΟΣΘΗΚΗ ΗΜΕΡΟΜΗΝΙΑΣ

Με την επιλογή SUBMIT στέλνεται η αντίστοιχη αίτηση στο κατάλληλο URL, ώστε να προστεθεί η ημερομηνία, ενώ με το CANCEL κλείνει τις επιλογές αυτές. Τέλος με το κουμπί SAVE CHANGES στέλνονται οι αλλαγές που αφορούν τις βασικές πληροφορίες, ενώ με το DELETE EVENT διαγράφεται ολόκληρη η διοργάνωση στην περίπτωση που δεν υπάρχει έγκυρη κράτηση. Η **multipartRequest** συνάρτηση που εξυπηρετεί όλες τις παραπάνω λειτουργίες δέχεται ως παράμετρο τη μέθοδο HTTP που θα χρησιμοποιηθεί, δίνοντας μας έτσι τη δυνατότητα να έχουμε μία συνάρτηση για την εξυπηρέτηση όλων των λειτουργιών του fragment.



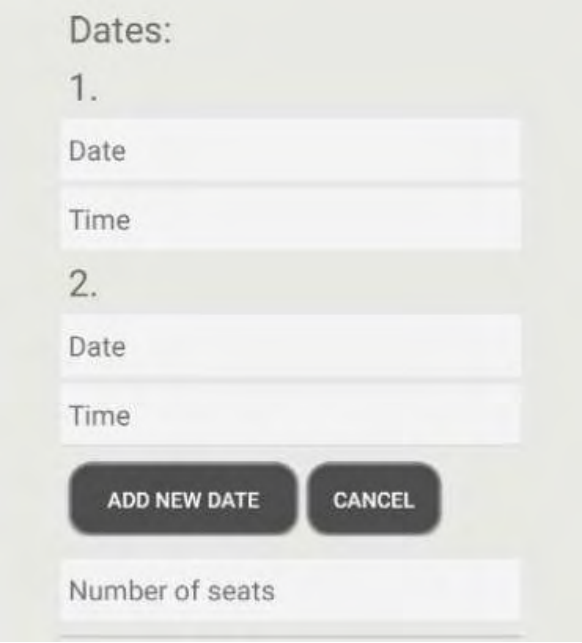
ΕΙΚΟΝΑ 4.24 – ΟΙ ΔΙΑΛΟΓΟΙ DATEPICKER ΚΑΙ TIMEPICKER

Η δεύτερη δραστηριότητα που περιέχει η εφαρμογή αυτή είναι η δραστηριότητα δημιουργίας νέας διοργάνωσης. Αυτή παρέχει τα απαραίτητα εργαλεία, ώστε να μπορέσει ο χρήστης να εισάγει όλες τις πληροφορίες που απαιτούνται για τη δημιουργία μιας νέας διοργάνωσης. Η δραστηριότητα αυτή αποτελείται από μία μακροσκελή φόρμα την οποία καλείται να συμπληρώσει ο χρήστης.



ΕΙΚΟΝΑ 4.25 – ΔΡΑΣΤΗΡΙΟΤΗΤΑ 1/4 - ΕΙΣΑΓΩΓΗ ΒΑΣΙΚΩΝ ΠΛΗΡΟΦΟΡΙΩΝ

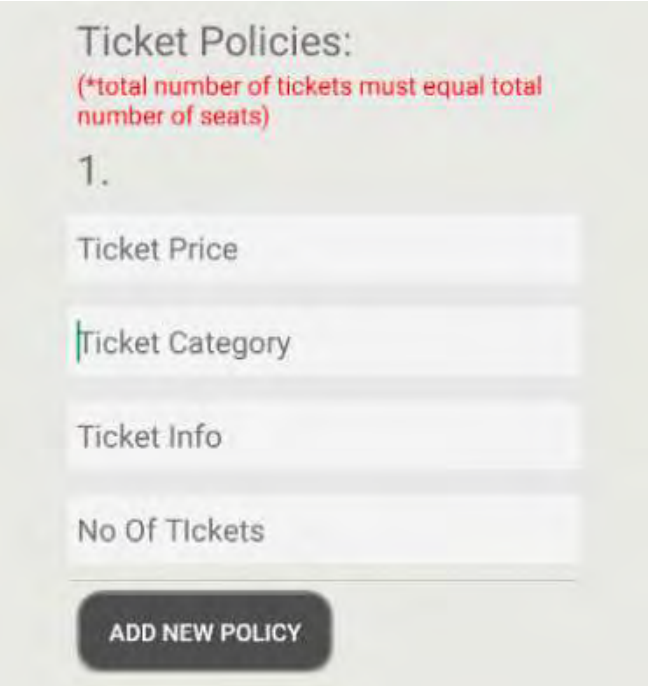
Αρχικά ζητούνται οι βασικές πληροφορίες, τις οποίες εισάγει στα αντίστοιχα **EditTexts** views. Αυτές αφορούν το όνομα της διοργάνωσης, την κατηγορία της, την διεύθυνση, τον χώρο και την περιγραφή. Έπειτα συμπληρώνει όσες ημερομηνίες επιθυμεί. Αυτές εμφανίζονται με ένα αύξον αριθμό και περιέχουν δύο **EditText** views, στα οποία συμπληρώνει την ημερομηνία και την ώρα με τη χρήση των DatePickerDialog και TimePickerDialog αντίστοιχα. Με την επιλογή του κουμπιού ADD NEW DATE προστίθεται και νέα πεδίο για να εισάγει την ημερομηνία και την ώρα. Με την προσθήκη νέας ημερομηνίας εμφανίζεται και το κουμπί CANCEL με το οποίο ακυρώνει την τελευταία που άνοιξε. Το layout που περιλαμβάνει τα 2 αυτά EditView είναι αποθηκευμένο σε ξεχωριστό xml αρχείο, το οποίο φορτώνεται σε ένα γενικό LinearLayout που περιέχει τις ημερομηνίες αυτές κάθε φορά που ο χρήστης πατάει το κουμπί ADD NEW DATE. Επιπλέον ζητείται από το χρήστη ο συνολικός αριθμός των θέσεων σε ένα αντίστοιχο **EditText** view.



ΕΙΚΟΝΑ 4.26 – ΔΡΑΣΤΗΡΙΟΤΗΤΑ 2/4 – ΕΙΣΑΓΩΓΗ ΗΜΕΡΟΜΗΝΙΩΝ ΚΑΙ ΘΕΣΕΩΝ

Αντίστοιχα με τις ημερομηνίες δουλεύει και το πεδίο με τις πολιτικές των εισιτηρίων, το οποίο περιέχει 4 **EditTexts**, στα οποία ο χρήστης συμπληρώνει τις πληροφορίες για κάθε πολιτική. Σε περίπτωση που ο χρήστης θέλει να χρησιμοποιήσει floor plan, τότε επιλέγει το αντίστοιχο **RadioButton**, το οποίο εμφανίζει ένα διάλογο με οδηγίες για το floorplan και αφού κλείσει τότε εμφανίζεται ένα κουμπί για την εισαγωγή του αρχείου. Με την επιλογή του κουμπιού για την εισαγωγή αρχείου ελέγχουμε αν έχουμε τα απαραίτητα δικαιώματα για να αποκτήσουμε πρόσβαση στα αρχεία του χρήστη και ξεκινάμε μία νέα δραστηριότητα με την συνάρτηση **startActivityForResult()**, στην οποία περνάμε ένα intent το οποίο

έχει ως action την επιλογή αρχείου, καθώς και έναν αναγνωριστικό κωδικό. Με το που επιλέξει ο χρήστης το αρχείο που αντιπροσωπεύει το floor plan τρέχει η onActivityResult() η οποία μας επιστρέφει το intent που σχετίζεται με το αρχείο που επέλεξε ο χρήστης. Έπειτα ελέγχουμε τον τύπο του αρχείο και αν δεν είναι "text/plain", εμφανίζεται σχετικό μήνυμα.



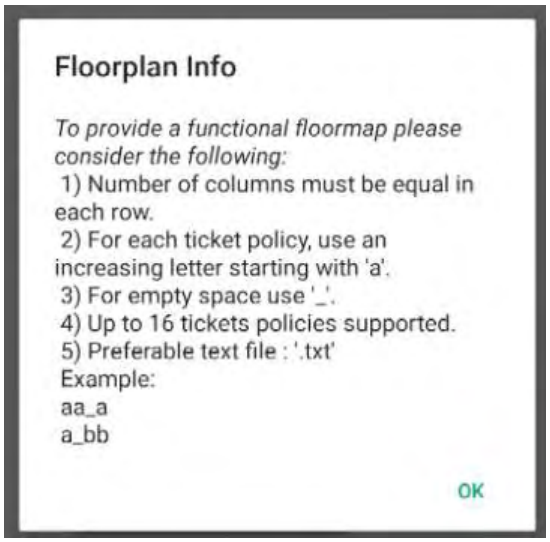
ΕΙΚΟΝΑ 4.27 ΔΡΑΣΤΗΡΙΟΤΗΤΑ 3/4 – ΕΙΣΑΓΩΓΗ ΠΟΛΙΤΙΚΩΝ ΕΙΣΙΤΗΡΙΩΝ

Σε αντίθεση με τα intent που γυρνάνε πληροφορίες για τις εικόνες, δεν μπορούμε να εξάγουμε το πραγματικό path του αρχείου κειμένου. Το intent που επιστράφηκε περιέχει ένα URI το οποίο περιγράφει το αρχείο αυτό. Το Android μας παρέχει την κλάση **ContentResolver** η οποία μας προσφέρει τη δυνατότητα να διαβάσουμε τα bytes του αρχείου που επιλέχθηκε ως stream. Αφού αποθηκεύσουμε τα bytes αυτά, τα στέλνουμε στο διακομιστή με τη αίτηση multipart που γίνεται στο τέλος.

ΔΙΑΒΑΣΜΑ BYTES ΤΟΥ FLOORPLAN ΑΡΧΕΙΟΥ ΑΠΟ ΤΟ URI

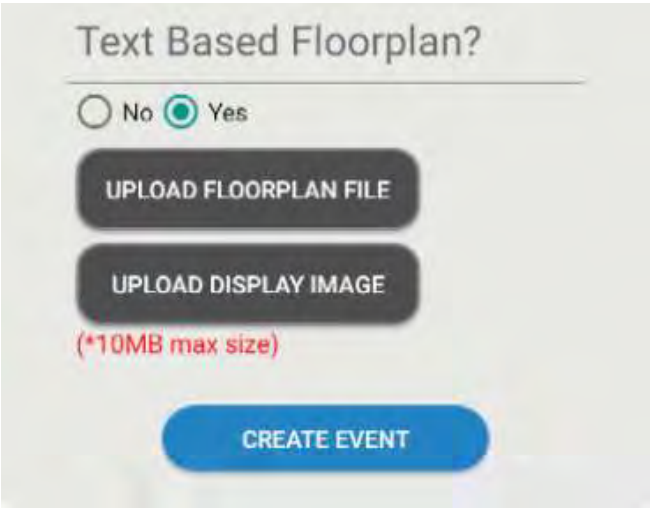
```
byte[] fileBytes;

//Άνοιγμα stream για διάβασμα του αρχείου
InputStream is =
     context.getContentResolver().openInputStream(floorplan_uri);

//Διάβασμα και αποθήκευση του αρχείου σε bytes
ByteArrayOutputStream buffer = new ByteArrayOutputStream();
int nRead;
byte[] data = new byte[16384];

while((nRead = is.read(data,0,data.length)) != -1){
```

```
    buffer.write(data,0,nRead);
}
buffer.flush();
fileBytes = buffer.toByteArray();
```



ΕΙΚΟΝΑ 4.28 – ΟΔΗΓΙΕΣ ΓΙΑ ΤΟ FLOORPLAN

Στο τελευταίο βήμα ο χρήστης διαλέγει μία εικόνα που αντιπροσωπεύει τη διοργάνωση που δημιουργεί. Με τον ίδιο τρόπο που αναλύθηκε στην προηγούμενη παράγραφο εξάγουμε το αρχείο από το intent που επιστρέφεται στην **onActivityResult**, αφού την ξεκινήσουμε με την **startActivityForResult()** παρέχοντας ένα intent που έχει ως action την επιλογή εικόνας.



ΕΙΚΟΝΑ 4.29 – ΔΡΑΣΤΗΡΙΟΤΗΤΑ 4/4 – ΕΠΙΛΟΓΗ ΕΙΚΟΝΑΣ ΚΑΙ FLOOR PLAN

Αφού συμπληρωθούν όλα τα στοιχεία, με την επιλογή του CREATE EVENT στέλνεται η αίτηση στο διακομιστή. Επειδή όλα τα στοιχεία αυτά πρέπει να σταλούν μέσω του AsyncTask που καλεί την multipartRequest του WebServiceAPI, τα τυλίγουμε σε μία κλάση και δηλώνουμε ως παραμέτρους του AsyncTask ένα αντικείμενο της κλάσης αυτής. Αφού γίνουν οι απαραίτητοι έλεγχοι και διαβεβαιωθούμε ότι όλα έχουν οριστεί σωστά καλείται το AsyncTask που στέλνει την αίτηση POST στο URL ”events/”. Αν η δημιουργία είναι επιτυχής και ο διακομιστής απαντήσει με ‘‘2**00**”, εμφανίζεται ανάλογο μήνυμα και ξεκινάει η κύρια δραστηριότητα. Σε αντίθετη περίπτωση εμφανίζεται το ανάλογο μήνυμα του διακομιστή.

5ο ΚΕΦΑΛΑΙΟ

ΕΠΙΛΟΓΟΣ

5.1 ΣΥΜΠΕΡΑΣΜΑΤΑ

Οι τεχνολογίες του διαδικτύου συνεχώς εξελίσσονται. Όπως το πρότυπο SOAP χάνει όλο και περισσότερο σε δημοτικότητα από την αρχιτεκτονική REST, έτσι μπορεί σε μερικά χρόνια να παρουσιαστεί κάποια καινούρια τεχνολογία η οποία να είναι πιο εύχρηστη και πιο εύρωστη από την REST. Σε αυτή την πτυχιακή δημιουργήσαμε ένα web service το οποίο αποκρίνεται στην αρχιτεκτονική REST, η οποία είναι και η πιο διαδεδομένη αυτή τη χρονική περίοδο. Το web service αυτό έχει ως σκοπό τη χορήγηση ενός πακέτου λειτουργιών, που θα διευκολύνει τους χρήστες τόσο στην εύρεση όσο και στην προώθηση διαφόρων ειδών διοργανώσεων. Οποιοσδήποτε μπορεί να φτιάξει τη δική του διεπαφή για το συγκεκριμένο web service χρησιμοποιώντας τις λειτουργίες αυτές. Επιπροσθέτως αναλύθηκε η ιστοσελίδα που παρέχει τις ίδιες δυνατότητες με το web service και αποτελεί έναν εναλλακτικό και πιο εύκολο τρόπο πρόσβασης στις λειτουργίες αυτές, καθώς όλο και περισσότερες συσκευές σχεδιάζονται έτσι, ώστε να έχουν πρόσβαση στο διαδίκτυο.

Παράλληλα με τις τεχνολογίες του διαδικτύου παρατηρείται και ραγδαία ανάπτυξη στις τεχνολογίες των κινητών συσκευών. Η σύνδεση των συσκευών αυτών με το διαδίκτυο θεωρείται αυτονόητο πλέον, για αυτό και πολλές εφαρμογές που αναπτύσσονται απαιτούν πρόσβαση στο ίντερνετ για την εξυπηρέτηση των βασικών λειτουργιών τους. Οι Android εφαρμογές που αναλύθηκαν στα πλαίσια της πτυχιακής αυτής δεν αποτελούν εξαίρεση, καθώς βασίζονται στην επικοινωνία με το web service. Το μοντέλο αυτό του πελάτη-διακομιστή αποτελεί το πρότυπο για πολλές εφαρμογές των οποίων τα δεδομένα βρίσκονται αποθηκευμένα σε έναν διακομιστή και ο χρήστης μπορεί να αποκτήσει πρόσβαση σε αυτά με τη κινητή συσκευή-πελάτη του σε οποιοδήποτε χρόνο και χώρο, αρκεί να διαθέτει σύνδεση στο διαδίκτυο. Το σύστημα Android, στο οποίο αναπτύχθηκαν οι εφαρμογές αυτές, αποτελεί ένα πετυχημένο λειτουργικό σύστημα και έχει θέσει γερά θεμέλια στην ανάπτυξη εφαρμογών για smartphones και tablets. Ο καθένας μπορεί να δημιουργήσει μία δική του εφαρμογή android αρκεί να είναι εξοικειωμένος με τη γλώσσα Java.

Ανάλογα τα μέσα που διαθέτει ο χρήστης μπορεί να χρησιμοποιήσει την ιστοσελίδα, το web service ή τις εφαρμογές Android, ώστε να αποκτήσει πρόσβαση στις λειτουργίες που προσφέρονται. Οι διοργανώσεις που προσφέρονται μπορούν να αφορούν μία απλή καθημερινή εκδήλωση όπως η προβολή μίας ταινίας, καθώς και μία πιο επίσημη κοινωνική συνάθροιση όπως μία διάλεξη. Οι εφαρμογές που

αναπτύχθηκαν για αυτή την εργασία μπορούν να καλύψουν οποιοδήποτε σκοπό εξυπηρετούν αυτές οι διοργανώσεις.

5.2 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Όπως ισχύει για όλες τις εφαρμογές πάντα υπάρχει η δυνατότητα για περαιτέρω εξέλιξη. Τόσο στον τομέα του web service όσο και στον τομέα των Android συσκευών μπορούν να παρουσιαστούν καινούριες τεχνολογίες με την πάροδο του χρόνου. Το λειτουργικό σύστημα android που κάθε χρόνο εξελίσσεται προσφέρει συνεχώς καινούριες δυνατότητες ώστε να δημιουργούμε πιο αποδοτικές και εύχρηστες εφαρμογές. Αυτό όμως έχει ως συνέπεια παλιότερες εφαρμογές να μην είναι συμβατές με τις καινούριες αυτές τεχνολογίες, με αποτέλεσμα την ανάγκη για συντήρηση και προσαρμογή των εφαρμογών με τα διαθέσιμα λειτουργικά τις κάθε εποχής. Εκτός όμως από τη συντήρηση και διατήρηση των εφαρμογών σε σχέση με τα λειτουργικά συστήματα που προσφέρονται, υπάρχουν περιθώρια βελτίωσης των λειτουργιών που προσφέρει η ίδια η εφαρμογή. Παρακάτω αναφέρονται μερικές από τις ενδεχόμενες βελτιώσεις και επεκτάσεις των λειτουργιών που αναπτύχθηκαν στην πτυχιακή αυτή:

- Υποστήριξη πληρωμών τόσο μέσω της ιστοσελίδας, όσο και από τις εφαρμογές android και το web service.
- Υποστήριξη Google Maps για την παρουσίαση και την αναζήτηση των χώρων διεξαγωγής των διοργανώσεων.
- Παροχή πακέτων εγγραφής/χρέωσης για τους διοργανωτές.
- Υποστήριξη περισσότερων τύπων αρχείων για το floor plan.
- Αποθήκευση διοργανώσεων στο προφίλ του χρήση για μελλοντική επαναχρησιμοποίηση.
- Λειτουργία σχολίων στις διοργανώσεις.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

1. Paul Deitel, Harvey Deitel (2011), Προγραμματισμός Ίντερνετ & World Wide Web 4η Έκδοση.
2. Walter Savich (2008) Απόλυτη Java.
3. Lorna Jane Mitchell (2013), PHP Web Services
4. Robin Nixon (2015), Learning PHP, MySQL & JavaScript with JQuery, CSS & HTML5 4th Edition
5. Reto Meier (2012), Professional Android Application Development 4th Edition
6. Yao Yumin, Liu Weiguo (2008). Study of Android's Architecture and its Application Development, Compute Systems & Applications
7. Sebastian Peyrott (2016) JWT Handbook Έκδοση 0.11.0

Περιοδικά

8. Butler Margaret (2011), Pervasive Computing, IEEE (Volume 10, Issue 1)

Συνέδρια

9. Incorporating Web Services in Mobile Applications - Web 2.0 San Fran 2009

Ιστοσελίδες

10. http://www.medialab.ntua.gr/education/MultimediaTechnology/MultimediaTechnologyNotes/chap3a_4.htm
11. http://pdplab.it.uom.gr/project/soap/Theory/introduction.html
12. https://en.wikipedia.org/wiki/Representational_state_transfer
13. http://en.wikipedia.org/wiki/Google_Play
14. https://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html
15. https://dev.mysql.com/doc/refman/5.5/en/innodb-locking-transaction-model.html
16. https://el.wikipedia.org/wiki/Διαδίκτυο
17. http://jwt.io
18. http://www.json.org
19. http://httpd.apache.org/docs/current/mod/mod_rewrite.html
20. https://httpd.apache.org/ABOUT_APACHE.html
21. https://www.w3.org/TR/wsdl