ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Υλοποίηση ενός συστήματος διαχείρισης ραντεβού ιατρών για έξυπνες συσκευές

Αριστοτέλης Δελαγραμμάτικας

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων Καθηγητής

Αθανάσιος Κακαρούντας

Λαμία, Σεπτέμβριος 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Υλοποίηση ενός συστήματος διαχείρισης ραντεβού ιατρών για έξυπνες συσκευές

Αριστοτέλης Δελαγραμμάτικας

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων Καθηγητής

Αθανάσιος Κακαρούντας

Λαμία, Σεπτέμβριος 2015

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: 11/.09/2015

Ο – Η Δηλών

(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.

Υλοποίηση ενός συστήματος διαχείρισης ραντεβού ιατρών για έξυπνες συσκευές

Αριστοτέλης Δελαγραμμάτικας

Τριμελής Επιτροπή:

Αθανάσιος Κακαρούντας, Επίκουρος Καθηγητής, ΤΕΙ Ιόνιων Νήσων

Ιωάννης Αναγνωστόπουλος, Επίκουρος Καθηγητής, Παν/μιο Θεσσαλίας

Ευριπίδης Μάρκου, Επίκουρος Καθηγητής, Παν/μιο Θεσσαλίας

Περίληψη

Ο στόχος της πτυχιακής εργασίας ήταν η δημιουργία μίας εφαρμογής για κινητά τηλέφωνα και tablet PCs. Με τη βοήθεια αυτής της εφαρμογής, ο χρήστης (γιατρός) θα μπορεί να διαχειρίζεται το πρόγραμμά του (τα ραντεβού με ασθενείς καθώς και το επαγγελματικό του πρόγραμμα) ανά πάσα στιγμή και από οποιοδήποτε σημείο θελήσει. Η εφαρμογή παρέχει λειτουργικότητα που καλύπτει τη δημιουργία και διαχείριση ασθενών, ραντεβού καθώς και σημειώσεων, ενώ έχει υλοποιηθεί και κάποια βασική λειτουργικότητα στατιστικών στοιχείων.

Παράλληλα, και με την προοπτική της επέκτασης της εργασίας, υλοποιήθηκε μια ιστοσελίδα, ο στόχος της οποίας ήταν να υποστηρίζει τη λειτουργικότητα που παρέχει η εφαρμογή ώστε ο γιατρός να μπορεί να εργάζεται και σε Ηλεκτρονικό Υπολογιστή εφόσον το θελήσει.

Οι παραπάνω υλοποιήσεις εισάγουν την προοπτική μίας πολύ ενδιαφέρουσας επέκτασης , η οποία αφορά στη σύνδεση της εφαρμογής με την ιστοσελίδα, καθώς και την επικοινωνία τους με στόχο το συγχρονισμό των δεδομένων τους. Σε αυτό το πλαίσιο, υλοποιήθηκε ένα web service, το οποίο παρέχεται από την ιστοσελίδα έτσι ώστε ή εφαρμογή να μπορεί να κάνει χρήση του σε κάποια μελλοντική επέκταση της εργασίας.

Abstract

The purpose of this project was the development of an application which can run in mobile apps and tablet PCs. The user of the application (a doctor) may utilize its functionality in order to manage his schedule (both his medical appointments and his generic professional agenda) any given time from anywhere in the world, provided that there is an Internet connection available. The functionality ranges from creation and maintenance of patients, appointments with them as well as memo notes, to a basic support for statistics.

Furthermore, with a possible extension of the current project tin mind, a website has been developed in order to provide the doctor with the option of working from his/her PC providing the exact same set of functionality.

Both of the abovementioned implementations introduce a new, interesting aspect regarding the extension of the current project; communication between the website and the application in order to synchronize data. In this context, the basis for this step was provided by implementing a web service, which the website exposes, the purpose of which is that any client (such as the currently implemented mobile application) may utilize to communicate and synchronize data with it.

Περιεχόμενα

Περίληψη 6

Abstract 7

1 Εισαγωγή 13

1.1 Αναγνώριση του προβλήματος 13

1.2 Στόχος της εργασίας 13

1.3 Οργάνωση κεφαλαίων 13

2 Βασικές γνώσεις για ανάπτυξη εφαρμογών 15

2.1 Ανάπτυξη εφαρμογών για Android 15

2.1.1 Activity 15

2.1.2 Intents 19

2.1.3 Βάσεις δεδομένων 20

2.1.4 Υποστηρίζοντας τις διαφορετικές οθόνες 21

2.2 Drupal 25

2.2.1 Σύστημα διαχείρισης περιεχομένου 25

2.2.2 Τεχνολογίες 25

2.2.3 Ασφάλεια 25

2.2.4 Ενότητες 26

2.2.5 Ρόλοι χρηστών 27

2.2.6 Εμφάνιση 28

2.3 Μεταφορά δεδομένων μεταξύ εξυπηρετητή και πελάτη 30

2.3.1 SOAP 30

2.3.2 REST 32

2.3.3 REST εναντίον SOAP 34

3 Σχεδίαση και υλοποίηση συστήματος 38

3.1 Ανάλυση Προδιαγραφών 38

3.1.1 Λειτουργικές προδιαγραφές 38
3.1.2 Μη λειτουργικές προδιαγραφές 39
3.2 Αρχιτεκτονική εφαρμογής 40
3.3 Υλοποίηση web service 43
3.4 Υλοποίηση ιστοσελίδας 47
3.5 Εργαλεία 50
4 Εγχειρίδιο χρήστη 51
4.1 Ιστοσελίδα 51
4.1.1 Σύνδεση 51
4.1.2 Μενού δημιουργίας περιεχομένου 51
4.1.3 Δημιουργία ραντεβού 52
4.1.4 Δημιουργία ασθενούς 53
4.1.5 Δημιουργία σημείωσης 54
4.1.6 Τα ραντεβού μου 55
4.1.7 Οι ασθενείς μου 57
4.1.8 Οι σημειώσεις μου 58
4.2 Εφαρμογή 60
4.2.1 Σύνδεση 60
4.2.2 Αρχική οθόνη 60
4.2.3 Δημιουργία ραντεβού 62
4.2.4 Δημιουργία Ασθενούς 63
4.2.5 Δημιουργία σημείωσης 65
4.2.6 Τα ραντεβού μου 65
4.2.7 Οι ασθενείς μου 68
4.2.8 Οι σημειώσεις μου 71

4.2.9 Σύνολο ραντεβού ανά μήνα 75
5 Συμπεράσματα 76
5.1 Κάλυψη στόχων ιστοσελίδας 76
5.2 Παροχή Web Service 77
5.3 Εφαρμογή κινητών τηλεφώνων και Tablet PCs 77
5.4 Ανοιχτά θέματα – Βελτιώσεις - Επεκτάσεις 78
5.4.1 Εισαγωγή πολλαπλών ασθενών σε ένα ραντεβού 78
5.4.2 Συχγρονισμός δεδομένων μεταξύ ιστοσελίδας και εφαρμογής 78
5.4.3 Εισαγωγή μενού στις οθόνες λιστών 79
6 Πηγές 80
Παράρτημα: Κώδικας 82

Πίνακας εικόνων

Εικόνα 2.1 Επισκόπηση του κύκλου ζωής του activity (Google, 2014) 16

Εικόνα 2.2 Αδρανοποίηση και συνέχιση ενός activity (Google, 2014) 17

Εικόνα 2.3 Τερματισμός και επανεκκίνηση ενός activity (Google, 2014)..... 17

Εικόνα 2.4 Αποθήκευση δεδομένων κατά το σταμάτημα της εφαρμογής (Google, 2014)..... 18

Εικόνα 2.5 Παρουσίαση της μπάρας διαχείρισης του Drupal..... 26

Εικόνα 2.6 Οθόνη διαχείρισης των ενοτήτων στο Drupal 27

Εικόνα 2.7 Οθόνη βασικών ρόλων χρηστών στο Drupal..... 28

Εικόνα 2.8 Οθόνη αδειών στο Drupal. 28

Εικόνα 2.9 Οθόνη διαχείρισης θεμάτων στο Drupal..... 29

Εικόνα 2.10 Παρουσίαση της βασικής δομής ενός SOAP envelope (XML στοιχείου τύπου «φάκελος») (Curbera, 2002)..... 30

Εικόνα 2.11 Παρουσίαση των αποτελεσμάτων του πειράματος των Hamad, Saad και Abed (Hatem Hamad, Motaz Saad, and Ramzi Abed, 2010)..... 35

Εικόνα 2.12 Σύγκριση του HTTP ωφέλιμου φορτίου σε σύγχρονες web services (Fahad Aijaz, Syed Zahid Ali, Muzzamil Aziz Chaudhary, Bernhard Walke, 2009). 36

Εικόνα 2.13 Σύγκριση του HTTP ωφέλιμου φορτίου σε ασύγχρονες web services (Fahad Aijaz, Syed Zahid Ali, Muzzamil Aziz Chaudhary, Bernhard Walke, 2009). 37

Εικόνα 3.1 Αρχιτεκτονική εφαρμογής 40

Εικόνα 4.1 Φόρμα σύνδεσης στην ιστοσελίδα 51

Εικόνα 4.2 Μενού δημιουργίας περιεχομένου..... 52

Εικόνα 4.3 Φόρμα δημιουργίας νέου ραντεβού..... 53

Εικόνα 4.4 δημιουργίας νέου ασθενούς 53

Εικόνα 4.5 Φόρμα δημιουργίας νέας σημείωσης..... 55

Εικόνα 4.6 Μενού κορυφής για την πλοήγηση στη σελίδα των ραντεβού 55

Εικόνα 4.7 Παρουσίαση των ραντεβού του χρήστη με τη μορφή ημερολογίου 56

Εικόνα 4.8 Αναλυτική παρουσίαση ενός ραντεβού 56

Εικόνα 4.9 Μενού κορυφής για την πλοήγηση στη σελίδα των ασθενών 57

Εικόνα 4.10 Παρουσίαση των ασθενών του χρήστη με τη μορφή λίστας 57

Εικόνα 4.11 Αναλυτική παρουσίαση ενός ασθενούς 58

Εικόνα 4.12 Μενού κορυφής για την πλοήγηση στη σελίδα των σημειώσεων..... 58

Εικόνα 4.13 Παρουσίαση των σημειώσεων του χρήστη με τη μορφή λίστας χαρτιών σημείωσης (post it)..... 59

Εικόνα 4.14 Αναλυτική παρουσίαση μίας σημείωσης..... 59

Εικόνα 4.15 Οθόνη σύνδεσης στην εφαρμογή 60

Εικόνα 4.16 Αρχική οθόνη εφαρμογής 61

Εικόνα 4.17 Μενού εφαρμογής 62

Εικόνα 4.18 Οθόνη δημιουργίας νέου ραντεβού 63

Εικόνα 4.19 Οθόνη δημιουργίας νέου ασθενούς 64

Εικόνα 4.20 Οθόνη δημιουργίας νέας σημείωσης 65

Εικόνα 4.21 Οθόνη δημιουργίας νέας σημείωσης 66

Εικόνα 4.22 Οθόνη μενού έπειτα από παρατεταμένο κράτημα..... 67

Εικόνα 4.23 Οθόνη αναλυτικής παρουσίασης ενός ραντεβού 68

Εικόνα 4.24 Οθόνη παρουσίασης των ασθενών του χρήστη 69

Εικόνα 4.25 Οθόνη αναλυτικής παρουσίασης ενός ασθενούς 70

Εικόνα 4.26 Οθόνη επεξεργασίας ασθενούς 71

Εικόνα 4.27 Οθόνη παρουσίασης των σημειώσεων του χρήστη..... 72

Εικόνα 4.28 Οθόνη αναλυτικής παρουσίασης μίας σημείωσης 73

Εικόνα 4.29 επεξεργασίας σημείωσης 74

Εικόνα 4.30 Οθόνη υπολογισμού των συνολικών ραντεβού του χρήστη ανά μήνα .. 75

1 Εισαγωγή

1.1 Αναγνώριση του προβλήματος

Οι σύγχρονοι ρυθμοί εργασίας έχουν επιβαρύνει με ένα πολύ απαιτητικό και συνεχώς μεταβλητό πρόγραμμα τους επαγγελματίες. Ο κλάδος της ιατρικής δεν αποτελεί εξαίρεση. Πιθανή λύση για την διαχείριση ενός απαιτητικού εβδομαδιαίου προγράμματος μπορεί να αποτελέσει η καθημερινή χρήση κινητών τηλεφώνων και tablet PCs, τα οποία προσφέρουν σημαντική υπολογιστική ισχύ, ίσως μεγαλύτερη από ποτέ στα χέρια των χρηστών. Γίνεται λοιπόν αντιληπτό πως αυτή η υπολογιστική ισχύς θα μπορούσε να αξιοποιηθεί για να εξυπηρετήσει στη λύση προβλημάτων που δημιουργούνται από τα απαιτητικά προγράμματα των ιατρών στις μέρες μας.

1.2 Στόχος της εργασίας

Η πτυχιακή εργασία προσανατολίζεται στο να εισάγει μία αποδοτική και άρτια τεχνολογική λύση στο παραπάνω πρόβλημα. Συγκεκριμένα, μέσα από τη δημιουργία μίας εφαρμογής για κινητά τηλέφωνα και tablet PCs, ο γιατρός θα μπορεί να διαχειρίζεται το πρόγραμμά του ανά πάσα στιγμή και από οποιοδήποτε σημείο θελήσει. Παράλληλα, μία ιστοσελίδα θα προσφέρει την ευκαιρία για επεξεργασία των δεδομένων μέσα από την άνεση ενός υπολογιστή.

1.3 Οργάνωση κεφαλαίων

Παρακάτω αναλύεται η οργάνωση των κεφαλαίων της πτυχιακής εργασίας:

Το κεφάλαιο 2 είναι αφιερωμένο σε κάποιες βασικές έννοιες των 3 οντοτήτων που έπρεπε να υλοποιηθούν (εφαρμογή, ιστοσελίδα και web service) ώστε να εισάγει τον αναγνώστη σε βασικές έννοιες που θα διευκολύνουν την παρακολούθηση των επόμενων κεφαλαίων. Στην ενότητα 2.1 αναλύονται έννοιες που αφορούν στην ανάπτυξη εφαρμογών για Android, ενώ η ενότητα 2.2 αφορά στην ιστοσελίδα και συγκεκριμένα στις τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της. Η ενότητα 2.3 παρουσιάζει τις δύο κυρίαρχες επιλογές για την ανάπτυξη web services (REST και SOAP) και τις συγκρίνει μεταξύ τους για τις ανάγκες του έργου.

Το κεφάλαιο 3 είναι αφιερωμένο στη σχεδίαση και την υλοποίηση του έργου. Αρχικά, στην ενότητα 3.1 παρουσιάζονται οι προδιαγραφές, όπως αυτές συμφωνήθηκαν πριν την έναρξη της υλοποίησης. Η ενότητα 3.2 αναλύει την αρχιτεκτονική της εφαρμογής. Οι ενότητες 3.3 και 3.4 επικεντρώνονται στην υλοποίηση του web service και της ιστοσελίδας αντίστοιχα, ενώ στην ενότητα 3.5 παραθέτονται οι τεχνολογίες οι οποίες χρησιμοποιήθηκαν για την υλοποίηση του έργου.

Το κεφάλαιο 4 αποτελεί το εγχειρίδιο χρήστη. Η ενότητα 4.1 αφορά στην ιστοσελίδα, ενώ η ενότητα 4.2 αφορά στην εφαρμογή.

2 Βασικές γνώσεις για ανάπτυξη εφαρμογών

2.1 Ανάπτυξη εφαρμογών για Android

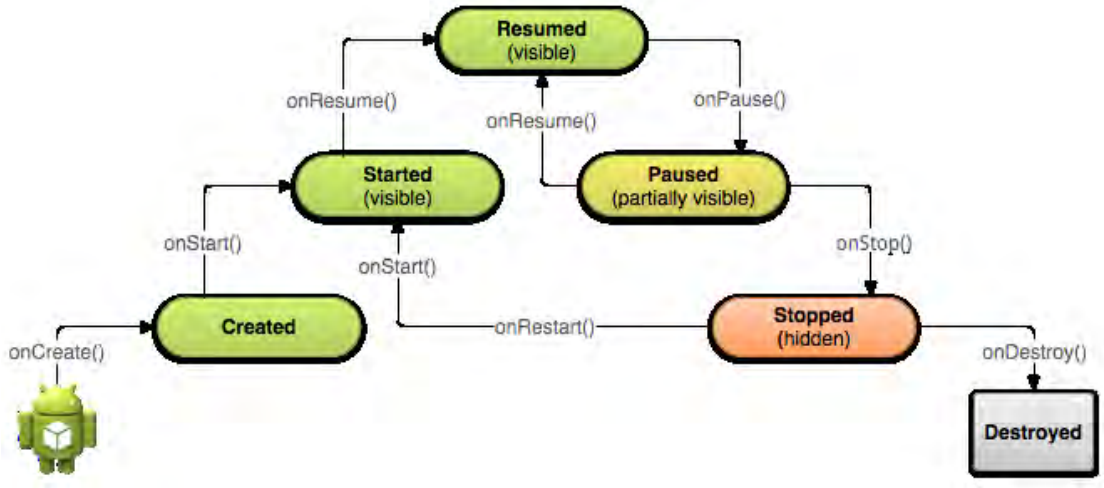
2.1.1 Activity

Το activity είναι ένα μοναδικό στοιχείο μιας εφαρμογής στο οποίο ο προγραμματιστής μπορεί να προσκολλήσει μία διεπαφή χρήστη (User Interface - UI), δίνοντάς του το ρόλο μίας «οθόνης». Σύμφωνα με την επίσημη ιστοσελίδα των προγραμματιστών του Android (Google, 2014), μία εφαρμογή συνήθως έχει πολλά activities, ένα από τα οποία χαρακτηρίζεται ως το κύριο και πρόκειται για αυτό που εμφανίζεται κάθε φορά που γίνεται εκκίνηση της εφαρμογής. Το ίδιο το activity μπορεί να ξεκινήσει ένα νέο activity, με αποτέλεσμα να κάνει τον εαυτό του να σταματήσει. Παρόλα αυτά, ένα σταματημένο activity δε θα μαζευτεί από το συλλογέα σκουπιδιών (garbage collector). Λαμβάνοντας ως δεδομένο πως μόνο ένα activity μπορεί να βρίσκεται στο προσκήνιο (οποιαδήποτε τυχαία χρονική στιγμή, η εστίαση – focus – μπορεί να είναι συγκεντρωμένη μόνο σε μία οθόνη) τα υπόλοιπα αποθηκεύονται σε μία στοίβα (stack - η οποία ονομάζεται «στοίβα παρασκηνίου» - back stack) η οποία διατηρείται από το σύστημα καθώς οποιαδήποτε στιγμή υπάρχει το ενδεχόμενο να ζητηθεί κάποιο activity να μεταβεί εκ’ νέου στο προσκήνιο (για παράδειγμα το activity που ήταν υπεύθυνο για τη διακοπή του, τερματίσει τη λειτουργία του). Η στοίβα παρασκηνίου λειτουργεί με τη λογική Τελευταίος-Μέσα-Πρώτος-Έξω (Last-In-First-Out - LIFO). Τα activities επικοινωνούν μεταξύ τους μέσω μεθόδων.

Στο σημείο αυτό θα αναλύσουμε τον κύκλο ζωής ενός activity και τις βασικές μεθόδους με τις οποίες αυτά επικοινωνούν.

Ο κύκλος ζωής των activities είναι μεγίστης σημασίας διότι καθώς ο χρήστης πλοηγείται, αδρανοποιεί, τερματίζει και επανεκκινεί την εφαρμογή, τα activities μεταβαίνουν στις διάφορες καταστάσεις του κύκλου ζωής. Κατά τη διάρκεια των μεταβάσεων πραγματοποιούνται κλήσεις στις αντίστοιχες μεθόδους ώστε να εκτελέσουν απαραίτητες ενέργειες, δίνοντας την ευκαιρία στον προγραμματιστή να χειριστεί με επιθυμητό τρόπο τις παραπάνω μεταβάσεις (για παράδειγμα να κλείσει τη σύνδεση στη βάση δεδομένων όταν ο χρήστης αδρανοποιήσει την εφαρμογή και να συνδεθεί ξανά όταν ο χρήστης επανεκκινήσει την εφαρμογή).

Η εικόνα 2.1 παρουσιάζει τον κύκλο ζωής ενός activity (παραθέτοντας αναλυτικά τόσο τις διακριτές καταστάσεις στις οποίες μπορεί να βρίσκεται, όσο και τις μεθόδους που καλούνται κατά τις μεταβάσεις ανάμεσα σε αυτές τις καταστάσεις) όπως αυτός παρουσιάζεται στην επίσημη ιστοσελίδα των προγραμματιστών του Android (Google, 2014).

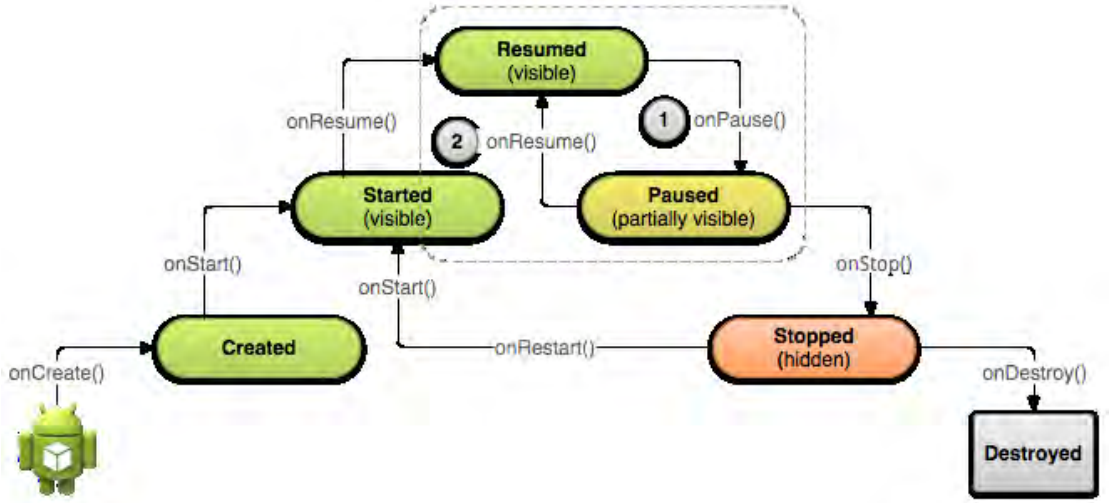


Εικόνα 2.1 Επισκόπηση του κύκλου ζωής του activity (Google, 2014)

Το πρώτο σημαντικό στοιχείο που πρέπει να τονιστεί είναι πως αντίθετα με την κοινή λογική των περισσότερων γλωσσών προγραμματισμού, κατά τη δημιουργία ενός νέου activity δεν υπάρχει μία μοναδική μέθοδος η οποία καλείται. Αντίθετα, μια ακολουθία από μεθόδους καλείται τόσο κατά τη δημιουργία όσο και κατά την καταστροφή του activity.

Όπως εξηγείται αναλυτικά στην επίσημη ιστοσελίδα των προγραμματιστών του Android (Google, 2014), οι μέθοδοι και οι καταστάσεις του κύκλου ζωής μπορούν να παρομοιαστούν με μία πυραμίδα (όπως γίνεται εύκολα αντιληπτό και από την εικόνα 2.2). Η κορυφή της πυραμίδας είναι το σημείο στο οποίο το activity βρίσκεται στο προσκήνιο και ο χρήστης μπορεί να αλληλεπιδράσει μαζί του. Καθώς το activity δημιουργείται, κάθε μέθοδος το φέρνει ένα βήμα πιο κοντά στην κορυφή, ενώ κατά την καταστροφή του, κάθε μέθοδος το απομακρύνει από αυτή.

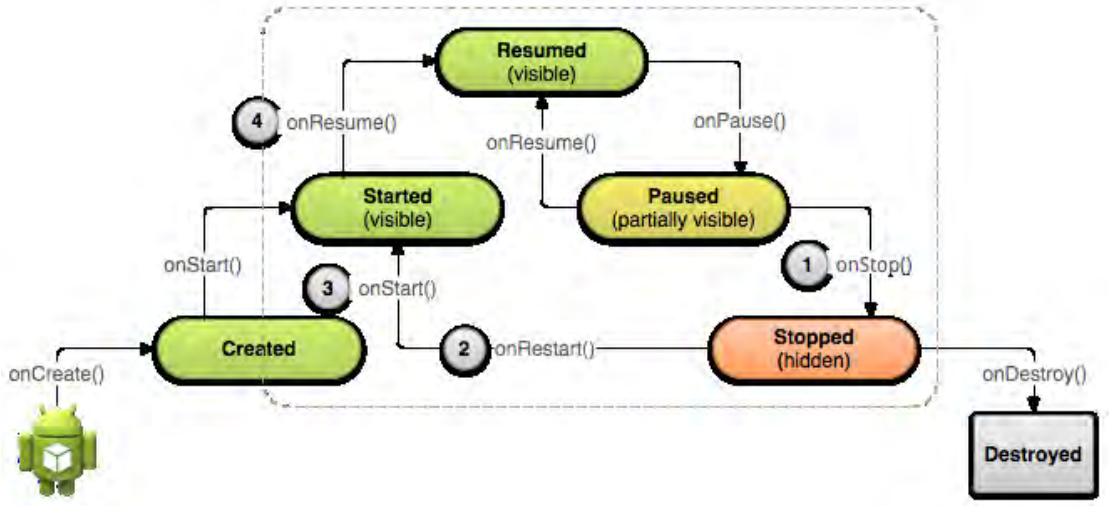
Αξίζει να σημειωθεί πως είναι πιθανό ένα activity να παραμείνει σε κάποιο ενδιάμεσο επίπεδο πριν κινηθεί προς την αντίθετη κατεύθυνση. Αν εξετάσουμε ένα σενάριο κατά το οποίο η οθόνη παρεμποδίζεται (για παράδειγμα από κάποιο διάλογο) η εφαρμογή θα αδρανοποιηθεί. Όπως φαίνεται στην εικόνα 2.2 το λειτουργικό σύστημα θα καλέσει τη μέθοδο *onPause()*. Αντίστοιχα, όταν το εμπόδιο αποχωρήσει, θα κληθεί η μέθοδος *onResume()* και η εφαρμογή θα συνεχίσει την εκτέλεσή της.



Εικόνα 2.2 Αδρανοποίηση και συνέχιση ενός activity (Google, 2014)

Κατά αντιστοιχία, όπως μετά την αδρανοποίηση της εφαρμογής η συνέχιση οφείλει να είναι ομαλή, έτσι και μετά από τον τερματισμό της η επανεκκίνησή της πρέπει να γίνει με την ίδια προσοχή. Μία εφαρμογή μπορεί να τερματιστεί και να επανεκκινηθεί για διάφορους λόγους. Για παράδειγμα, ο χρήστης μπορεί να δεχτεί κάποιο τηλεφώνημα ενώ η εφαρμογή εκτελείται ή μέσω της χρήσης των πρόσφατων εφαρμογών, ο χρήστης πλοηγείται σε μία άλλη εφαρμογή και έπειτα επιστρέφει.

Σύμφωνα με την επίσημη ιστοσελίδα των προγραμματιστών του Android (Google, 2014) οι μέθοδοι *onStop()* και *onRestart()* χρησιμοποιούνται στα παραπάνω σενάρια. Η εικόνα 2.3 επικεντρώνεται στις μεθόδους που αφορούν σενάρια τερματισμού και επανεκκίνησης.



Εικόνα 2.3 Τερματισμός και επανεκκίνηση ενός activity (Google, 2014)

Τέλος, υπάρχουν και τα σενάρια κατά τα οποία ένα activity καταστρέφεται. Το παραπάνω μπορεί να οφείλεται είτε σε φυσιολογική συμπεριφορά (για παράδειγμα ο χρήστης χρησιμοποιεί το κουμπί «πίσω», είτε η ίδια η εφαρμογή καλεί τη μέθοδο *finish()*. Το λειτουργικό σύστημα μπορεί επίσης να καταστρέψει την εφαρμογή είτε λόγω προσπάθειας εξοικονόμησης πόρων για άλλες εφαρμογές είτε επειδή η εφαρμογή έχει σταματήσει και δεν έχει χρησιμοποιηθεί για αρκετό χρονικό διάστημα.

Σε αυτές τις περιπτώσεις, ο προγραμματιστής μπορεί να χρησιμοποιήσει τη μέθοδο onSaveInstanceState() για να αποθηκεύσει κάποια δεδομένα για την τρέχουσα κατάσταση της εφαρμογής. Στην εικόνα 2.4 φαίνονται οι πιθανές χρήσης της μεθόδου.

Τέλος, παρακάτω θα αναλυθεί σύντομα η λειτουργία των πιο σημαντικών μεθόδων του κύκλου ζωής του activity:

onCreate(): Η πρώτη μέθοδος που καλείται κατά τη δημιουργία ενός activity. Η μέθοδος παρέχει την κατάλληλη ευκαιρία για κάποια παρασκευαστική εργασία από τον προγραμματιστή (για παράδειγμα άνοιγμα σύνδεσης με τη βάση δεδομένων, ανάλυση μεταβλητών που προέρχονται από προηγούμενο activity κλπ).

Εικόνα 2.4 Αποθήκευση δεδομένων κατά το σταμάτημα της εφαρμογής (Google, 2014)

onStart(): Ακολουθεί πάντα την *onCreate()* και είναι η τελευταία μέθοδος που καλείται πριν το activity γίνει ορατό στο χρήστη. Με τη σειρά της, καλεί την *onResume()*.

onPause(): Καλείται ακριβώς πριν το λειτουργικό σύστημα καλέσει την *onResume()* μέθοδο κάποιου άλλου activity. Αυτό σημαίνει πως αυτή η μέθοδος αποτελεί το σημείο κατά το οποίο ο προγραμματιστής πρέπει να εκτελέσει οτιδήποτε οφείλει να εκτελεστεί πριν το activity μεταβεί στη στοίβα παρασκηνίου.

onStop(): Η μέθοδος καλείται όταν το activity παύει να είναι ορατό στο χρήστη. Σε περίπτωση που ο χρήστης εκκινήσει ξανά την εφαρμογή, η *onRestart()* μέθοδος καλείται μετά.

onDestroy(): Καλείται όταν ο χρήστης καταστρέψει την εφαρμογή. Καμία άλλη μέθοδος δεν καλείται έπειτα.

Τέλος, είναι ιδιαίτερης σημασίας για να τον προγραμματιστή να χρησιμοποιήσει σωστά τον κύκλο ζωής του activity. Μία εφαρμογή μπορεί να διακοπεί πολλαπλές φορές κατά την εκτέλεσή της και για διαφορετικούς προβλέψιμους και μη παράγοντες. Αυτός είναι ο κύριος λόγος για τον οποίο ο κύκλος ζωής είναι ιδιαίτερα πολύπλοκος σε σχέση με τον κύκλο ζωής μιας τυπικής εφαρμογής για υπολογιστές. Είναι λοιπόν ζωτικής σημασίας αυτές οι διακοπές και οι παράγοντες που τις προκαλούν να έχουν προβλεφθεί και να λάβουν σωστής διαχείρισης.

2.1.2 Intents

Στο Android, τα intents είναι αντικείμενα τα οποία μπορούν να χρησιμοποιηθούν για πολλαπλούς σκοπούς. Συνήθως κρατάει αποθηκευμένη πληροφορία για μια λειτουργία η οποία πρόκειται να εκτελεστεί ή για κάτι το οποίο συνέβη, λειτουργώντας ως «αγγελιοφόρος». Επίσης, είναι αρκετά συνηθισμένο να μεταφέρει ένα μήνυμα, ζητώντας από κάποιο άλλο στοιχείο της εφαρμογής να πραγματοποιήσει μία ενέργεια. Παρότι τα intents χρησιμοποιούνται με ποικίλους τρόπους για την επικοινωνία των επί μέρους κομματιών μιας εφαρμογής, οι κυριότερες περιπτώσεις χρήσης τους είναι οι παρακάτω:

- **Εκκίνηση ενός activity**. Σε αυτή την περίπτωση, ένα intent εισάγεται σαν όρισμα σε μία από τις μεθόδους *Context.startActivity()*, *Activity.startActivityForResult()* και *Acticity.setResult()* ώστει είτε να εκκινήσει ένα νέο activity είτε να επιστρέψει αποτέλεσμα στο activity που το δημιούργησε γι' αυτό τον σκοπό.

- **Εκκίνηση ενός service**. Σε αυτή την περίπτωση, ένα intent εισάγεται σαν όρισμα σε μία από τις μεθόδους *Context.startService()* ή *Context.bindService()* ώστε είτε να ξεκινήσει ένα καινούργιο service είτε να δημιουργήσει μία σύνδεση με ένα service το οποίο ήδη τρέχει.

- **Παράδοση ενός broadcast**. Σε αυτή την περίπτωση, ένα intent εισάγεται σαν όρισμα σε μία μέθοδο ενός broadcast και παραδίδεται σε ένα broadcast receiver.

Σύμφωνα με την επίσημη ιστοσελίδα των προγραμματιστών του Android (Google, 2014), υπάρχουν δύο τύποι intents:

- **Άμεσα intents.** Σε αυτό το είδος το όνομα του στοιχείου της εφαρμογής που πρόκειται να ξεκινήσει την εκτέλεσή του ορίζεται ρητά, με χρήση του ονόματός του (του ονόματος της κλάσης του).

- **Έμμεσα intents**. Σε αυτό το είδος, δεν ορίζεται το όνομα του από κάποιο στοιχείο της εφαρμογής, αλλά η γενικότερη πράξη στην οποία στοχεύει το intent.

Εξετάζοντας αναλυτικά ορισμένα παραδείγματα από intents, η πληροφορία η οποία μπορεί να είναι αποθηκευμένη σε αυτά μπορεί αν είναι μία σταθερά (constant) η οποία περιγράφει μία πράξη που πρέπει να ξεκινήσει. Για παράδειγμα, η σταθερά *ACTION_CALL* χρησιμοποιείται για να ξεκινήσει μία τηλεφωνική κλήση και η σταθερά *ACTION_BATTERY_LOW* χρησιμοποιείται για να σταλεί μία προειδοποίηση πως η μπαταρία βρίσκεται σε χαμηλό επίπεδο. Επιπλέον, μπορεί να υπάρχει περισσότερη πληροφορία σε ένα intent, όπως μία συμβολοσειρά (string) για το ποιός είναι υπεύθυνος να χειριστεί την πράξη. Για παράδειγμα, η συμβολοσειρά *CATEGORY_PREFERENCE* υποδηλώνει πως το activity είναι μία οθόνη προτιμήσεων. Επίσης ένα intent μπορεί να διαθέτει μία σημαία (flag) με σκοπό να προσδώσει επιπλέον πληροφορία σχετικά με το πως πρέπει να πραγματοποιηθεί μια ενέργεια (όπως για παράδειγμα πως να χειριστεί ένα activity μετά την εκκίνησή του).

Τέλος, μία δέσμη (bundle), η οποία ονομάζεται «extras» μπορεί να προστεθεί στο intent φέροντας επιπλέον πληροφορία για να ανταλλαχτεί μεταξύ activities. Με αυτό τον τρόπο αποσαφηνίζεται περισσότερο ο ρόλος του intent ως «αγγελιοφόρου» καθώς, εκτός από την τυποποιημένη πληροφορία σε σχέση με την πράξη για την οποία δημιουργήθηκε, μπορεί πλέον να μεταφέρει και αυθαίρετη (τόσο σε δομή όσο και σε μέγεθος) πληροφορία. Ο όρος αυθαίρετη σε δομή πληροφορία υποστηρίζεται διότι παρόλο που η δέσμη αποθηκεύει την πληροφορία σε ζευγάρια κλειδί-τιμή (key-value pairs), ο τύπος της τιμής κυμαίνεται σε ένα φάσμα από ένα απλό θεμελιακό τύπο (primitive type) ως μία δομή δεδομένων (για παράδειγμα συστοιχίες – arrays κλπ). Με αυτό τον τρόπο, τα intents παρέχουν ένα μεγάλο βαθμό ελευθερίας στον προγραμματιστή να τα χρησιμοποιήσει για να ανταλλάξει σημαντικό όγκο πληροφορίας ανάμεσα σε δύο activities.

2.1.3 Βάσεις δεδομένων

Το Android παρέχει υποστήριξη για SQLite βάσεις δεδομένων, οι οποίες είναι διαθέσιμες σε κάθε συσκευή. Ως αποτέλεσμα, ο προγραμματιστής δε χρειάζεται να ρυθμίσει με κάποιο τρόπο το περιβάλλον ώστε να χρησιμοποιήσει μία βάση δεδομένων. Αντίθετα, αυτό που πρέπει να γίνει είναι να οριστούν οι κατάλληλες SQL δηλώσεις για τη δημιουργία και την ενημέρωση μίας βάσης δεδομένων. Η κάθε βάση δεδομένων είναι προσβάσιμη μόνο από την εφαρμογή η οποία τη δημιούργησε και από καμία άλλη.

Η SQLite είναι, σύμφωνα με την επίσημη ιστοσελίδα της (SQLite, 2014), μία βιβλιοθήκη η οποία είναι υπεύθυνη για την υλοποίηση μίας μηχανής SQL βάσεων δεδομένων. Η SQLite βρίσκεται ενσωματωμένη στο Android και το βασικό στοιχείο που επέτρεψε αυτή την ενσωμάτωση είναι πως η SQLite δε χρησιμοποιεί κάποιο εξυπηρετητή (server), αντίθετα με την πλειοψηφία παρόμοιων συστημάτων. Αντίθετα, είναι υλοποιημένη ώστε να διαβάζει από και να γράφει απευθείας σε αρχεία που είναι αποθηκευμένα στο δίσκο. Τέλος, είναι βασισμένη στην SQL και παρέχει περιορισμένους τύπους δεδομένων, οι οποίοι όμως, με κατάλληλο χειρισμό, επαρκούν για τις ανάγκες κάθε εφαρμογής.

Σύμφωνα με την επίσημη ιστοσελίδα των προγραμματιστών του Android (Google, 2014), ο ενδεδειγμένος τρόπος για να δημιουργήσει ένας προγραμματιστής μία βάση δεδομένων είναι να δημιουργήσει μία υποκλάση της *SQLiteOpenHelper* και να παρακάμψει (override) την *onCreate()* μέθοδο, μέσα στην οποία να εκτελέσει τις SQLite εντολές για τη δημιουργία της επιθυμητής βάσης δεδομένων. Οι μέθοδοι *getWritableDatabase()* και *getReadableDatabase()* χρησιμοποιούνται για να επιστρέψουν ένα αντικείμενο *SQLiteDatabase* το οποίο επιτρέπει αντίστοιχα την εγγραφή και την ανάγνωση από τη βάση δεδομένων.

2.1.4 Υποστηρίζοντας τις διαφορετικές οθόνες

Το πρόβλημα της υποστήριξης διαφορετικών οθονών εισάγεται από μία θεμελιώδη ιδιότητα του Android ως λειτουργικού συστήματος. Σε αντίθεση με άλλα λειτουργικά συστήματα για κινητά τηλέφωνα και tablet PCs, το Android δε στοχεύει ένα συγκεκριμένο φάσμα συσκευών. Το σύνολο των συσκευών που χρησιμοποιούν Android κυμαίνεται ουσιαστικά χωρίς περιορισμούς ανάμεσα σε συσκευές με πολύ διαφορετικά τεχνικά χαρακτηριστικά (όπως για παράδειγμα επεξεργαστική ισχύ, μνήμη και, φυσικά, οθόνη).

Το πρόβλημα διογκώνεται ακόμα περισσότερο όταν ορίσουμε και τον όρο «διαφορετικές οθόνες». Είναι φανερό πως δύο οθόνες μπορεί να διαφέρουν στο μέγεθός τους, όμως ένα δεύτερο χαρακτηριστικό περιπλέκει αυτή τη διαφοροποίηση καθώς, μπορεί να διαφέρει και η πυκνότητά τους. Αυτό έχει ως αποτέλεσμα οι πιθανοί συνδυασμοί (μεγέθους και πυκνότητας) να πολλαπλασιάζονται καθώς και να εισάγεται το ενδεχόμενο δύο πανομοιότυπες σε μέγεθος οθόνες να διαφέρουν λόγω πυκνότητας.

Γίνεται αντιληπτό πως η ανάγκη για μία εφαρμογή να τρέχει σωστά σε οποιαδήποτε οθόνη είναι καίριας σημασίας. Το Android προσφέρει κάποιους μηχανισμούς βελτιστοποίησης, ο προγραμματιστής είναι υπεύθυνος να εξασφαλίσει πως η εφαρμογή θα λειτουργήσει σωστά.

Παρακάτω θα εξετάσουμε τους πιο σημαντικούς παράγοντες που διαφοροποιούν δύο οθόνες, όπως αυτοί αναλύονται στην επίσημη ιστοσελίδα των προγραμματιστών του Android (Google, 2014).

Μέγεθος οθόνης: Η φυσική διάσταση της διαγωνίου της οθόνης. Η μονάδα μέτρησης που χρησιμοποιείται είναι το dp (density – independent pixels), ώστε η πυκνότητα να μετριέται ανεξάρτητα από το πλήθος των εικονοστοιχείων (pixels). Αξίζει να σημειωθεί πως το Android χρησιμοποιεί την εξής κατηγοριοποίηση για λόγους απλότητας: μικρές (small), μεσαίες (normal), μεγάλες (large) και έξτρα μεγάλες (extra-large). Οι διαστάσεις που αντιστοιχούν σ' αυτή την κατηγοριοποίηση φαίνονται στον παρακάτω πίνακα:

Μικρές οθόνες	Μεσαίες οθόνες	Μεγάλες οθόνες	Έξτρα μεγάλες οθόνες
Τουλάχιστον 426dp x 320dp	Τουλάχιστον 470dp x 320dp	Τουλάχιστον 640dp x 480dp	Τουλάχιστον 960dp x 720dp

Πυκνότητα οθόνης: Η ποσότητα των εικονοστοιχείων (pixels) σε μία δεδομένη περιοχή της οθόνης. Η πυκνότητα της οθόνης μετριέται συνήθως σε τελείες ανά ίντσα (dots per inch - dpi). Ο αριθμός αυτός αναπαριστά τον αριθμό των εικονοστοιχείων που μπορούν να χωρέσουν σε μία γραμμή μήκους μίας ίντσας. Η κάθε οθόνη έχει τη δικιά της πυκνότητα (για παράδειγμα 167 dpi, 239 dpi κλπ) χωρίς να ακολουθείται κάποιο προσυμφωνημένο πρότυπο. Είναι σαφές πως είναι αδύνατον να προβλεφθεί και να καλυφθεί κάθε ξεχωριστή πυκνότητα οθόνης, οπότε, για λόγους απλότητας και πάλι, το Android εισάγει την κατηγοριοποίηση που φαίνεται στον ακόλουθο πίνακα:

ldpi	mdpi	hdpi	xhdpi	xxhdpi	xxxhdpi
Χαμηλή πυκνότητα (~120 dpi)	Μεσαία πυκνότητα (~160 dpi)	Υψηλή πυκνότητα (~240 dpi)	Έξτρα υψηλή πυκνότητα (~320 dpi)	Έξτρα έξτρα υψηλή πυκνότητα (~480 dpi)	Έξτρα έξτρα έξτρα υψηλή πυκνότητα (~640 dpi)

Προσανατολισμός: Ο προσανατολισμός της οθόνης μπορεί να έχει δύο πιθανές τιμές: τοπίο (landscape) και πορτραίτο (portrait) κατά τους οποίους η αναλογία διαστάσεων της οθόνης είναι ευρεία ή ψηλή αντίστοιχα. Ένα

σημαντικό στοιχείο που πρέπει να ληφθεί υπ'όψιν είναι πως ενώ το μέγεθος και η πυκνότητα της οθόνης είναι χαρακτηριστικά της συσκευής, ο προσανατολισμός μπορεί να αλλάζει αυθαίρετα όσο η εφαρμογή εκτελείται.

Ανάλυση: Η ανάλυση της οθόνης είναι ο συνολικός αριθμός των εικονοστοιχείων (pixels) σε αυτή. Εφόσον μία εφαρμογή προβλέπει και χειρίζεται σωστά το μέγεθος και την πυκνότητα της οθόνης, δε θα αντιμετωπίσει πρόβλημα με την ανάλυση.

Το Android παρέχει μηχανισμούς για την αντιμετώπιση των παραπάνω προβλημάτων. Η πολιτική που ακολουθείται σε σχέση με το πρόβλημα του μεγέθους της οθόνης είναι να παρέχονται από τον προγραμματιστή διαφορετικές διατάξεις (layouts) για διαφορετικά μεγέθη οθονών. Το σύστημα αρχείων που φιλοξενεί αυτές τις διατάξεις χωρίζεται σε υποφακέλους που αντιπροσωπεύουν το μέγεθος της οθόνης και περιέχουν τα κατάλληλα αρχεία γι'αυτό το μέγεθος. Ένα παράδειγμα φαίνεται στο παρακάτω υποθετικό σύστημα αρχείο για τη διάταξη «my_layout.xml».

```
res/layout-small/my_layout.xml
res/layout/my_layout.xml
res/layout-large/my_layout.xml
res/layout-xlarge/my_layout.xml
```

Επίσης, ο προγραμματιστής μπορεί να ορίσει και κάποιον δικό του φάκελο για να χειριστεί μία συγκεκριμένη περίπτωση εφόσον κριθεί σκόπιμο. Με αυτό τον τρόπο παρέχεται ακόμα μεγαλύτερη ελευθερία και ευελιξία.

Το πρόβλημα των διαφορετικών πυκνοτήτων, εφόσον είναι πανομοιότυπο, αντιμετωπίζεται με αντίστοιχο τρόπο. Συγκεκριμένοι φάκελοι ορίζονται για την κάθε κατηγορία πυκνότητας, οι οποίοι περιέχουν τα αντίστοιχα αρχεία. Παρακάτω φαίνεται ένα υποθετικό σύστημα αρχείων για το αρχείο «my_icon.png».

```
res/drawable/my_icon.png
res/drawable-ldpi/my_icon.png
res/drawable-mdpi/my_icon.png
res/drawable-hdpi/my_icon.png
res/drawable-xhdpi/my_icon.png
res/drawable-xxhdpi/my_icon.png
res/drawable-xxxhdpi/my_icon.png
```

Τέλος, το πρόβλημα του προσανατολισμού προσεγγίζεται εντελώς διαφορετικά λόγω της πολύ διαφορετικής φύσης του. Όπως επισημάνθηκε και παραπάνω, είναι κοινό

για μία συσκευή να βρίσκεται είτε σε προσανατολισμό τοπίο είτε σε πορτραίτο, αλλά και να εναλλάσσεται ανάμεσα στις δύο κατά τη διάρκεια της εκτέλεσης της εφαρμογής αυθαίρετα, απλώς γυρίζοντας φυσικά την εφαρμογή. Το πρόβλημα αντιμετωπίζεται με πολύ προσεκτικό σχεδιασμό και χρήση των κατάλληλων διατάξεων των οθονών ώστε να είναι αποδοτικό και στους δύο προσανατολισμούς, καθώς και με ενδελεχή έλεγχο (testing) και στις δύο καταστάσεις.

2.2 Drupal

2.2.1 Σύστημα διαχείρισης περιεχομένου

Το Drupal, σύμφωνα με την επίσημη ιστοσελίδα του (Drupal, 2014), είναι ένα σύστημα διαχείρισης περιεχομένου (Content Management System – CMS) το οποίο έχει σχεδιαστεί για την ανάπτυξη ιστοσελίδων. Δίνοντας ιδιαίτερη έμφαση στην ευελιξία και την απλότητα, το Drupal λειτουργεί βασισμένο σε μία λογική συναρμολόγησης. Τα κύρια κομμάτια μιας τυπικής ιστοσελίδας είναι ήδη υλοποιημένα και προσφέρονται για χρήση, δίνοντας παράλληλα την ελευθερία στον προγραμματιστή να υλοποιήσει τις δικές του λύσεις για λειτουργικότητες που είτε δεν υπάρχουν έτοιμες, είτε τροποποιημένη λειτουργικότητα είναι επιθυμητή. Με αυτό τον τρόπο είναι αρκετά εύκολη η δημιουργία ενός βασικού κορμού που μπορεί να υποστηρίξει με αξιοπιστία και ασφάλεια την πλειονότητα των ιστοσελίδων, πάνω στον οποίο μπορεί να δημιουργηθεί η εξειδικευμένη λειτουργικότητα που απαιτείται από την εκάστοτε ιστοσελίδα.

2.2.2 Τεχνολογίες

Σε αυτό το σημείο θα πρέπει να τονιστεί πως, σε αντίθεση με την πλειοψηφία των παρόμοιων συστημάτων που υπάρχουν και χρησιμοποιούνται, το Drupal απευθύνεται στον προγραμματιστή και όχι σε κάποιον απλό χρήστη. Από τεχνικής πλευράς, η γλώσσα προγραμματισμού στην οποία είναι γραμμένο το Drupal και τα τμήματα που χρησιμοποιεί είναι η PHP. Σε αυτή τη γλώσσα ο προγραμματιστής καλείται να υλοποιήσει και τη δική του λειτουργικότητα. Επίσης, χρησιμοποιείται μία βάση δεδομένων για την αποθήκευση και τη διαχείριση δεδομένων καθώς και για την αποθήκευση πολλαπλών ρυθμίσεων. Για το ρόλο αυτό χρησιμοποιείται μία SQL βάση δεδομένων (συνήθως MySQL ή PostgreSQL). Τέλος, ένας εξυπηρετητής είναι απαραίτητος. Στην πλειοψηφία των περιπτώσεων χρησιμοποιείται κάποιος Apache εξυπηρετητής.

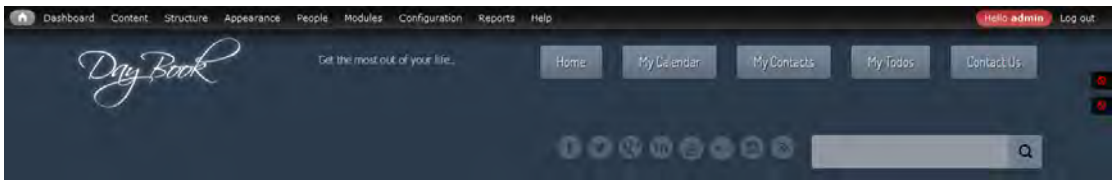
2.2.3 Ασφάλεια

Κατά την ανάπτυξη ιστοσελίδων, ένα κύριο πρόβλημα που απασχολεί τους προγραμματιστές είναι η ασφάλεια του προϊόντος. Το Drupal διαθέτει ένα ενσωματωμένο μηχανισμό και διαδικασίες διερεύνησης, πιστοποίησης και δημοσίευσης προβλημάτων ασφαλείας. Σύμφωνα με την επίσημη ιστοσελίδα του Drupal, ιδιαίτερη έμφαση έχει δοθεί σε κοινές αδυναμίες, όπως για παράδειγμα SQL injections, Cross Site Scripting, Session Management, Cross Site Request Forgeries κλπ. Αυτές οι αδυναμίες έχουν προβλεφθεί και καταπολεμηθεί από το βασικό στάδιο του API (Application Programming Interface) που προσφέρει το Drupal, στο οποίο έχουν ενσωματωθεί μηχανισμοί αποφυγής όλων των παραπάνω πιθανών κινδύνων.

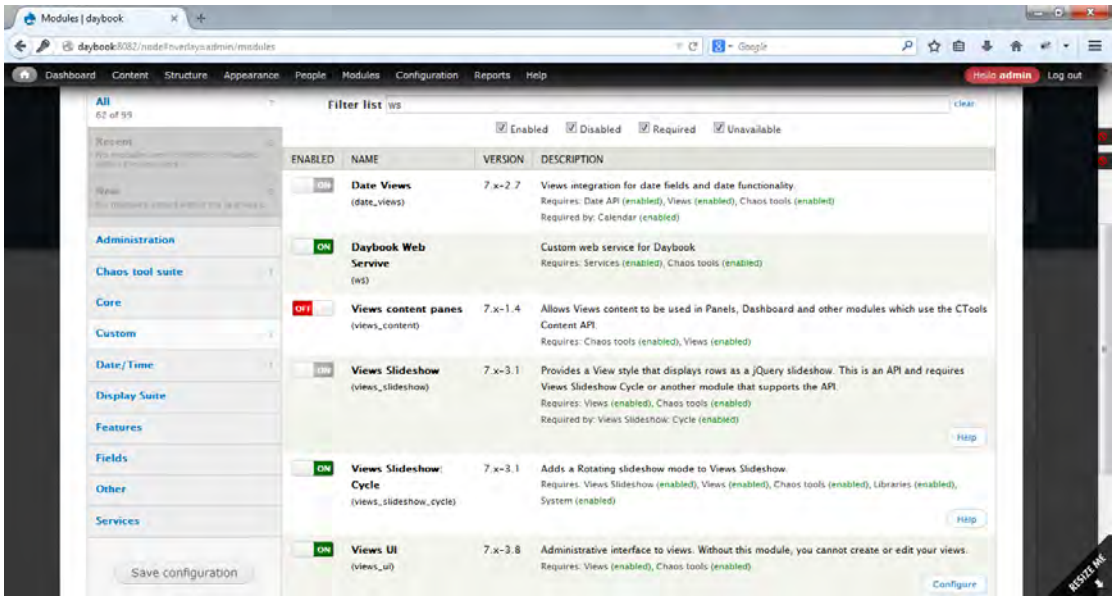
2.2.4 Ενότητες

Ο λόγος που το Drupal συμπεριφέρεται ιδιαίτερα με τη λογική της συναρμολόγησης είναι ότι είναι βασισμένο σε ενότητες (modules). Οι ενότητες είναι έτοιμα πακέτα λογισμικού, γραμμένα σε PHP, τα οποία είναι υλοποιημένα ώστε να μπορούν να ενσωματωθούν σε μία ιστοσελίδα που έχει αναπτυχθεί με Drupal. Η βασική εγκατάσταση του Drupal εγκαθιστά τις θεμελιώδεις σουίτες με ενότητες για μία ιστοσελίδα, ενώ προσφέρεται και πληθώρα άλλων κυρίως μέσα από την επίσημη ιστοσελίδα του συστήματος διαχείρισης περιεχομένου. Ο λόγος που η παραπάνω λογική δημιουργεί ένα πολύ ισχυρό πλαίσιο ανάπτυξης ιστοσελίδων είναι ότι ο χρήστης-προγραμματιστής του Drupal μπορεί να αναπτύξει τις δικές του ενότητες χρησιμοποιώντας το API του Drupal και την PHP. Με αυτό τον τρόπο μπορεί να εμπλουτίσει την ιστοσελίδα αλλά και να δώσει ακριβώς την επιθυμητή λειτουργικότητα σε αυτή. Η ευελιξία του συστήματος υποστηρίζεται από το γεγονός πως οι ενότητες, εφόσον εγκατασταθούν, μπορούν να ενεργοποιηθούν και απενεργοποιηθούν κατά βούληση τόσο από τον προγραμματιστή όσο και από το διαχειριστή της ιστοσελίδας.

Στην εικόνα 2.5 φαίνεται η μπάρα διαχείρισης που επιτρέπει λειτουργίες όπως την ενεργοποίηση και την απενεργοποίηση ενοτήτων, ενώ η εικόνα 2.6 παρουσιάζει ακριβώς την οθόνη διαχείρισης ενοτήτων.



Εικόνα 2.5 Παρουσίαση της μπάρας διαχείρισης του Drupal



Εικόνα 2.6 Οθόνη διαχείρισης των ενοτήτων στο Drupal

2.2.5 Ρόλοι χρηστών

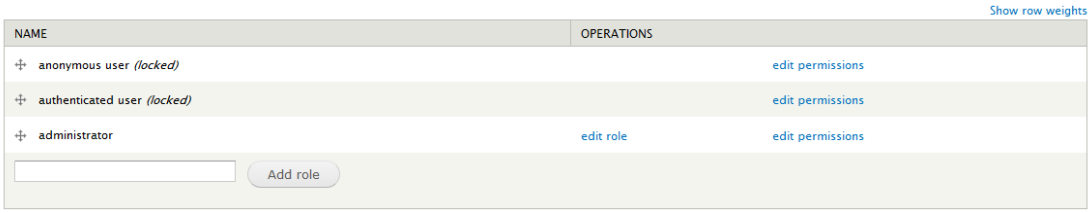
Το Drupal είναι σχεδιασμένο με τέτοιο τρόπο ώστε να υποστηρίζει πολύ αποδοτικά τους πολλαπλούς χρήστες για μία ιστοσελίδα και να παρέχει διαχωρισμό των επιτρεπτών ενεργειών ανά ομάδες χρηστών, τις οποίες αποκαλεί ρόλους. Το παραπάνω χαρακτηριστικό αποτελεί κλειδί για πολλές ιστοσελίδες, καθώς, θεωρώντας ένα απλό παράδειγμα μιας δημοσιογραφικής ιστοσελίδας γίνεται εύκολα αντιληπτό πως ο απλός χρήστης πρέπει να έχει διαφορετικά δικαιώματα πάνω στα άρθρα από ένα δημοσιογράφο.

Αρχικά, οι ρόλοι που προσφέρονται από το σύστημα είναι δύο:

- Ανώνυμος χρήστης: Ο ρόλος αυτός χρησιμοποιείται για επισκέπτες οι οποίοι δε διαθέτουν λογαριασμό στην ιστοσελίδα ή δεν έχουν πιστοποιηθεί.

- Πιστοποιημένοι χρήστες: Ο ρόλος αυτός χρησιμοποιείται για τους χρήστες που διαθέτουν λογαριασμό και έχουν ακολουθήσει επιτυχώς τη διαδικασία πιστοποίησης (login).

Οι παραπάνω ρόλοι, σε συνδυασμό με το ρόλο του διαχειριστή (administrator), ο οποίος δημιουργείται αυτόματα για κάθε ιστοσελίδα, φαίνονται στην εικόνα 2.7.



Εικόνα 2.7 Οθόνη βασικών ρόλων χρηστών στο Drupal

Αξίζει να σημειωθεί πως ο προγραμματιστής και ο διαχειριστής της ιστοσελίδας μπορούν να δημιουργήσουν επιπλέον ρόλους σύμφωνα με τις ανάγκες της ιστοσελίδας.

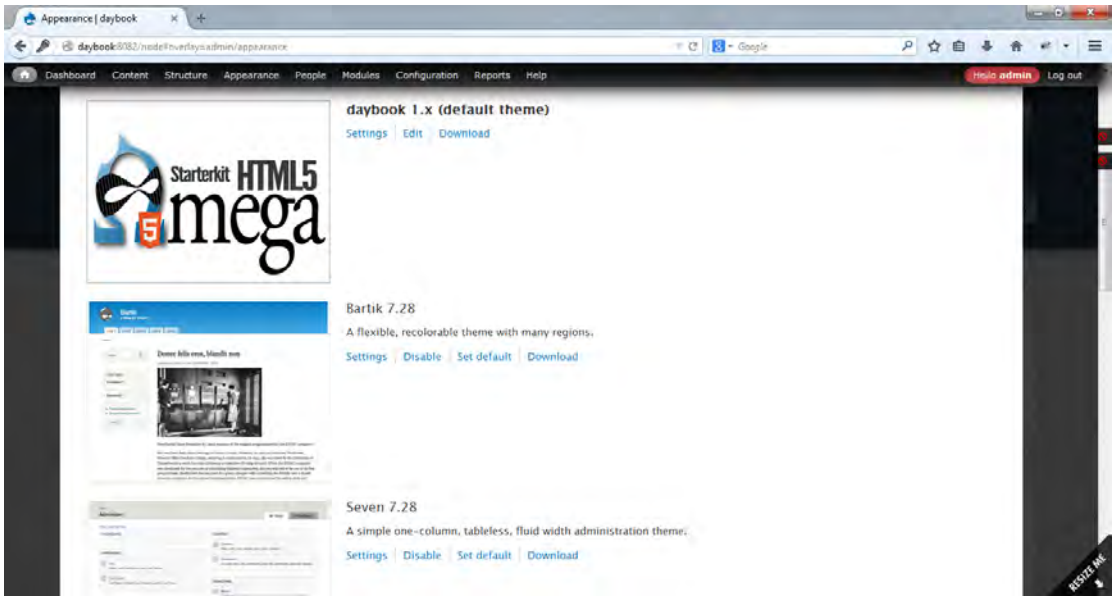
Με χρήση των παραπάνω ρόλων επιτυγχάνεται και ο διαχωρισμός στις ενέργειες που είναι επιτρεπτές για τον κάθε χρήστη. Ένα σύνολο αδειών (permissions) υποστηρίζεται για κάθε πιθανή ενέργεια και ο διαχειριστής είναι υπεύθυνος να δώσει στους σωστούς ρόλους χρηστών τις σωστές άδειες καθώς και να τοποθετήσει τους χρήστες σε ρόλους.

Ενδεικτικά, στην εικόνα 2.8 φαίνονται οι άδειες για τα σχόλια σε άρθρα και ο διαχωρισμός τους ανάλογα με τους ρόλους των χρηστών.

Εικόνα 2.8 Οθόνη αδειών στο Drupal.

2.2.6 Εμφάνιση

Η εμφάνιση μιας ιστοσελίδας είναι πολύ σημαντικός παράγοντας ως προς τη χρηστικότητα και την εμπειρία του χρήστη (User Experience). Οι λύσεις που προσφέρονται από το Drupal πάνω στο συγκεκριμένο τομέα φροντίζουν να δώσουν πλούσιες επιλογές για την αντιμετώπιση του προβλήματος. Παρόμοια με τις ενότητες, πληθώρα έτοιμων θεμάτων (themes) μπορούν να επιλεχθούν τόσο από την επίσημη ιστοσελίδα του Drupal όσο και από τρίτες ιστοσελίδες, όμως υποστηρίζεται και η επιλογή ανάπτυξης θεμάτων από τον προγραμματιστή με τη χρήση είτε CSS είτε LESS. Ένα σημαντικό πλεονέκτημα είναι πως τα θέματα μπορούν να ενεργοποιηθούν και να απενεργοποιηθούν αυθαίρετα. Στην εικόνα 2.9 παρουσιάζεται η διαχειριστική οθόνη του συστήματος μέσω της οποίας γίνεται η ενεργοποίηση και απενεργοποίηση των θεμάτων.



Εικόνα 2.9 Οθόνη διαχείρισης θεμάτων στο Drupal.

2.3 Μεταφορά δεδομένων μεταξύ εξυπηρετητή και πελάτη

2.3.1 SOAP

Το SOAP (Simple Object Access Protocol) είναι ένα πρωτόκολλο επικοινωνίας που επιτρέπει σε εφαρμογές να ανταλλάξουν πληροφορίες μέσω του Internet. Το SOAP βασίζεται σε XML απομακρυσμένες κλήσεις διαδικασιών (remote procedure calls - RPC). Όπως εξηγεί ο Jan Newmarch (Newmarch, 2005), το RPC είναι ένας μηχανισμός ο οποίος επιτρέπει σε μία εφαρμογή να πραγματοποιεί κλήσεις σε μία απομακρυσμένη υπηρεσία μέσω κλήσεων διαδικασιών οι οποίες φαινομενικά παρουσιάζονται να γίνονται τοπικά. Το επίπεδο του RPC μετατρέπει αυτές τις τοπικές κλήσεις σε ένα proxy στη μεριά του πελάτη (client-side proxy) σε TCP ή UDP πακέτα τα οποία προωθούνται σε ένα στέλεχος του εξυπηρετητή (server-side stub) όπου εκτελούνται οι διαδικασίες.

Οι Curebra και Duftler (Curbera, 2002) εξηγούν τον πυρήνα ενός SOAP μηνύματος, το οποίο καλείται «φάκελος» (envelope) και είναι ένα XML στοιχείο με ακριβώς δύο παιδιά: την «κορυφή» (header) και το «σώμα» (body). Ωστόσο, τόσο η κορυφή όσο και το σώμα έχουν αυθαίρετη δομή. Για την αξιοποίηση των μηνυμάτων, ο προσδιορισμός (specification) του SOAP περιλαμβάνει συγκεκριμένα πρότυπα (πρότυπα) με τη χρήση των οποίων μπορεί να γίνει η επεξεργασία (parsing) των μηνύματα αυτών.

Η εικόνα 2.10 παρουσιάζει τη βασική δομή ενός envelope (XML στοιχείου τύπου «φάκελος»).

```
<SOAP:Envelope xmlns:SOAP=
      "http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP:Header>
      <!– content of header goes here –>
   </SOAP:Header>
   <SOAP:Body>
      <!– content of body goes here –>
   </SOAP:Body>
</SOAP:Envelope>
```

Εικόνα 2.10 Παρουσίαση της βασικής δομής ενός SOAP envelope (XML στοιχείου τύπου «φάκελος») (Curbera, 2002)

Σε αυτό το σημείο ένα βασικό πρόβλημα είναι ορατό. Τόσο ο εξυπηρετητής όσο και ο πελάτης πρέπει να έχουν συμφωνήσει από την αρχή σχετικά με τα μηνύματα που

πρόκειται να ανταλλαχθούν. Για να λυθεί το παραπάνω πρόβλημα εισάγεται η χρήση των Web Services Description Language (WSDL) αρχείων. Τα WSDL είναι XML αρχεία τα οποία ορίζουν τη λειτουργικότητα που προσφέρεται από ένα web service.

Τέλος, το SOAP χρησιμοποιεί κυρίως τα SMTP (Simple Mail Transfer Protocol) και HTTP (HyperText Transfer Protocol) πρωτόκολλα.

Το SMTP είναι ένα πρωτόκολλο βασισμένο στο TCP/IP, το οποίο χρησιμοποιείται κυρίως για την ανταλλαγή μηνυμάτων ηλεκτρονικού ταχυδρομείου (e-mails). Το SMTP χρησιμοποιείται ευρέως, συνήθως με τον πελάτη (e-mail client) να χρησιμοποιεί POP ή IMAP (ώστε να αποθηκεύσει το μηνύματα σε ένα εξυπηρετητή και να τα «κατεβάσει»).

Το HTTP είναι το πρωτόκολλο που χρησιμοποιείται κατά κύριο λόγο στο διαδίκτυο (World Wide Web). Συνήθως χρησιμοποιείται για τη μεταφορά υπερκειμένου (hypertext). Επίσης, ορίζει πως πρέπει να τυποποιούνται και να στέλνονται τα μηνύματα και πως οι εξυπηρετητές και οι φυλλομετρητές (browsers) πρέπει να απαντούν σε αιτήματα (requests).

Τα κυριότερα χαρακτηριστικά του SOAP, όπως περιγράφονται από το World Wide Web Consortium (W3C) ((W3c), 2014) και τον Jan Newmarch (Newmarch, 2005), είναι τα παρακάτω:

Πλεονεκτήματα:

- Το SOAP είναι **ανεξάρτητο της πλατφόρμας** (platform independent). Το παραπάνω σημαίνει πως λειτουργεί ανεξάρτητα από τον εξυπηρετητή, ο οποίος μπορεί να είναι ένας εξυπηρετητής Linux, Mac κλπ. Λαμβάνοντας υπ' όψιν το πλήθος των πλατφορμών στο διαδίκτυο, αυτό πρόκειται για ένα σημαντικό πλεονέκτημα.

- Το SOAP είναι **ανεξάρτητο της γλώσσας** (language independent). Οι εφαρμογές που χρησιμοποιούν SOAP μπορεί να είναι ανεπτυγμένες σε διάφορες γλώσσες (όπως για παράδειγμα Java, C# κλπ). Παρόμοια με τις πλατφόρμες, υπάρχει μεγάλο πλήθος γλωσσών για την ανάπτυξη εφαρμογών στο διαδίκτυο. Η ανεξαρτησία του SOAP από τη γλώσσα ανάπτυξης της εφαρμογής παρέχει μεγάλη ελευθερία και ευελιξία στους προγραμματιστές καθώς είναι ιδιαίτερα προσαρμόσιμη.

- Το SOAP είναι **βασισμένο σε XML**. Σύμφωνα με τους Curbera, Duftler, Nagy, Mukhi και Weerawarana [2], η XML έχει πλέον καθιερωθεί ως η παγκόσμια γλώσσα της πληροφορίας και της κωδικοποίησης δεδομένων για την ανεξαρτησία πλατφορμών και την παγκοσμιοποίηση. Ως αποτέλεσμα,

πολλές εφαρμογές διαδικτύου επιλέγουν την XML ώστε να έχουν έναν κοινό και βολικό τρόπο επικοινωνίας με άλλες εφαρμογές.

- Το SOAP είναι **προτεινόμενο από το W3C** (W3C recommendation).

Μειονεκτήματα:

- Οι κλήσεις SOAP έχουν **αυξημένη πολυπλοκότητα** καθώς χρησιμοποιούν το δικό τους πρωτόκολλο, το οποίο είναι ένα επίπεδο πάνω στο HTTP (layered over HTTP).
- Το SOAP μπορεί να είναι **ιδιαίτερα αργό** λόγω της αρκετά φλύαρης XML μορφής που χρησιμοποιεί.
- Τυπικά, ένα αίτημα GET πρέπει να χρησιμοποιείται για να ανακτήσει πληροφορία από τον εξυπηρετητή και όχι για να κάνουν αλλαγές σε αυτόν. Για το σκοπό αυτό πρέπει να χρησιμοποιείται ένα POST αίτημα. Τα αιτήματα SOAP πρέπει να χρησιμοποιούν POST καθώς ανεβάζουν ένα XML αρχείο. Παρόλα αυτά δεν προορίζονται πάντα για να κάνουν κάποια αλλαγή στον εξυπηρετητή. Το παραπάνω μπορεί να οδηγήσει σε **μη ορθή χρήση του POST**.

2.3.2 REST

Το REST (Representational State Transfer) είναι στυλ αρχιτεκτονικής λογισμικού (software architecture style) για τη σχεδίαση εφαρμογών δικτύου. Δημιουργήθηκε για να παρέχει τον ιδανικό τρόπο παράδοσης εγγράφων μέσω HTTP. Οι Hamad, Saad και Abed (Hatem Hamad, Motaz Saad, and Ramzi Abed, 2010) εξηγούν πως η αρχιτεκτονική του REST είναι βασισμένη στην αρχιτεκτονική του μοντέλου πελάτη – εξυπηρετητή (client-server) και χρησιμοποιεί πάντα ένα πρωτόκολλο επικοινωνίας χωρίς καταστάσεις (stateless communication protocol – συνήθως το HTTP). Το παραπάνω σημαίνει πως το πρωτόκολλο χειρίζεται συνεχόμενα αιτήματα ανεξάρτητα το ένα από το άλλο. Τα web services που χρησιμοποιούν REST ονομάζονται «RESTful web services».

Σύμφωνα με τους Pautasso, Zimmerman και Leymann (C. Pautasso, 2008), το αρχιτεκτονικό μοντέλο REST βασίζεται σε τέσσερις βασικές αρχές:

- Η πρόσβαση σε πόρους (resources) επιτυγχάνεται μέσα από τη χρήση **URIs (Uniform Resource Identifiers)**. Μία RESTful web service παρέχει

(exposes) ένα σύνολο πόρων. Η λειτουργικότητα και τα δεδομένα θεωρούνται κι αυτά πόροι.

- Η πρόσβαση και ο χειρισμός των πόρων πραγματοποιείται μέσω τεσσάρων βασικών λειτουργιών: PUT, GET, POST και DELETE. Οι παραπάνω λειτουργίες δημιουργούν, διαβάζουν, ενημερώνουν και διαγράφουν πόρους αντίστοιχα.

- Οι πόροι είναι **αυτοεπεξηγούμενοι**. Αυτό σημαίνει πως δεν είναι συνδεδεμένοι με ένα συγκεκριμένο τύπο αρχείου και συνεπώς η πρόσβαση σε αυτούς μπορεί να γίνει με ποικίλους τρόπους (XML, plain text, προσαρμοσμένους τύπους – custom file format κλπ).

- Το πρωτόκολλο το οποίο χρησιμοποιείται είναι πάντα **χωρίς καταστάσεις** (stateless). Αυτό σημαίνει πως αντιμετωπίζει κάθε αίτημα ανεξάρτητα από τα προηγούμενα και τα επόμενα και δε διατηρεί αρχείο από προηγούμενα αιτήματα.

Μερικά από τα σημαντικότερα πλεονεκτήματα που προσφέρει το REST περιγράφονται από τον Δρ. Roy Fielding (Fielding, 2000) στη διδακτορική του διατριβή:

- Το REST **δε χρησιμοποιεί καταστάσεις** (stateless). Όπως αναφέρθηκε και παραπάνω, κάθε αίτημα που ο πελάτης στέλνει στον εξυπηρετητή πρέπει να περιέχει όλη την πληροφορία που απαιτείται από τον εξυπηρετητή για να επεξεργαστεί το αίτημα καθώς εκείνος δεν αποθηκεύει καθόλου περιεχόμενο. Αυτό επιτρέπει το χειρισμό πιθανών λαθών με καλύτερη αποτελεσματικότητα και επιτρέπει στον εξυπηρετητή να απελευθερώνει πόρους πιο γρήγορα, ενώ παράλληλα, μπορεί να μειώσει τη συνολική απόδοση καθώς η ίδια πληροφορία πρέπει να αποσταλεί εκ νέου εφόσον δεν αποθηκεύεται.

- Προσπαθώντας να διορθώσει το παραπάνω μειονέκτημα, το REST υποστηρίζει τη **λανθάνουσα μνήμη** (cache). Η απάντηση μπορεί να επισημανθεί ως «cacheable» και αν αποθηκευτεί, ο πελάτης μπορεί να επαναχρησιμοποιήσει αυτή την πληροφορία για μελλοντικά αιτήματα.

- Το REST χρησιμοποιεί μία **ομοιόμορφη διεπαφή** ανάμεσα στα στοιχεία (components). Με αυτό τον τρόπο η αρχιτεκτονική απλοποιείται αρκετά και δεν υπάρχουν σχέσεις εξάρτησης (dependencies) ανάμεσα στην υλοποίηση και την υπηρεσία που αυτή παρέχει. Παρόλα αυτά, το παραπάνω μπορεί να έχει αρνητικό αντίκτυπο στην αποτελεσματικότητα καθώς οι απαντήσεις ακολουθούν ένα γενικευμένο πρότυπο και όχι ένα στοχευόμενο στις ανάγκες της εκάστοτε εφαρμογής.

- Η χρήση **διακριτών επιπέδων** (distinct layers) βοηθάει στη διατήρηση της απλότητας σε κάθε ξεχωριστό επίπεδο, ενώ βοηθάει και στην αποσύνδεση (decoupling) καθώς κάθε επίπεδο μπορεί να αλληλεπιδρά με οποιοδήποτε άλλο και όχι μόνο με αυτά που βρίσκονται ακριβώς από πάνω και κάτω του. Το παραπάνω είναι μία επιλογή που προσφέρει πλεονεκτήματα, όμως, ταυτόχρονα, μπορεί να αυξήσει την καθυστέρηση (latency) και να προσθέσει επιπλέον όγκο δεδομένων (overhead).

Τέλος, δύο επιπλέον βασικά πλεονεκτήματα του REST είναι ότι είναι **ανεξάρτητο της πλατφόρμας** (platform independent) και **ανεξάρτητο της γλώσσας προγραμματισμού** (language independent).

2.3.3 REST εναντίον SOAP

Παρακάτω ακολουθεί μία σύγκριση του REST με το SOAP με κύριο γνώμονα την απόδοση.

Οι Hamad, Saad και Abed (Hatem Hamad, Motaz Saad, and Ramzi Abed, 2010) ανέλυσαν την απόδοση μίας REST και μίας SOAP web service σε δύο απλές καταστάσεις: μία πρόσθεση αριθμού κινητής υποδιαστολής (floating point number addition) και στη σύνδεση πινάκων συμβολοσειρών (string array concatenation). Μία απλή RESTful web service υλοποιήθηκε, καθώς και η αντίστοιχη SOAP και οι δύο τους φιλοξενήθηκαν σε ένα Glassfish εξυπηρετητή. Το πείραμα σχεδιάστηκε ώστε να μετρήσει το χρόνο (με χρήση της μεθόδου System.currentTimeMillis()) και το μέγεθος των μηνυμάτων που χρησιμοποιήθηκαν για τις δύο αυτές απλές λειτουργίες.

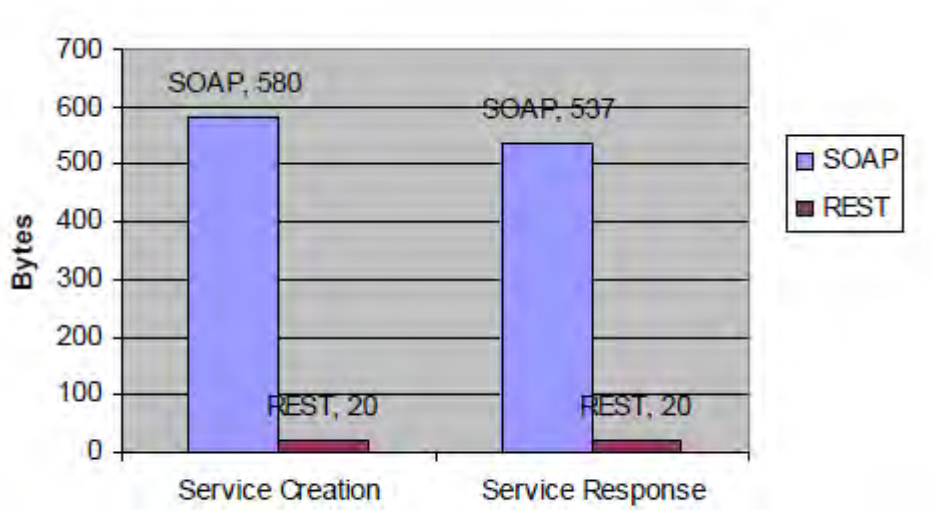
Τα αποτελέσματα φανέρωσαν σημαντικές επιπτώσεις που αφορούν στην απόδοση των δύο web services. Τόσο το μέγεθος των μηνυμάτων όσο και ο χρόνος τείνουν να είναι λιγότερο από μισά χρησιμοποιώντας REST σε σχέση με τις αντίστοιχες τιμές χρησιμοποιώντας SOAP και στις δύο λειτουργίες για πλήθος στοιχείων στους πίνακες μέχρι 8. Η επίδραση στην επεξεργασία (processing) είναι άμεση και στο εύρος ζώνης (bandwidth) που απαιτείται τα οποία επιδρούν σημαντικά στη διάρκεια ζωής της μπαταρίας της συσκευής ενός κινητού τηλεφώνου ή tablet PC. Τα αποτελέσματα του πειράματος παρουσιάζονται αναλυτικά στην εικόνα 2.11.

Number of array elements	Message Size (byte)				Time (Milliseconds)			
	SOAP/HTTP		REST (HTTP)		SOAP/HTTP		REST (HTTP)	
	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition
2	351	357	39	32	781	781	359	359
3	371	383	48	36	828	781	344	407
4	395	409	63	35	828	922	359	375
5	418	435	76	39	969	1016	360	359
6	443	461	93	43	875	953	359	359
7	465	487	104	47	875	875	469	360
8	493	513	127	51	984	875	437	344

Εικόνα 2.11 Παρουσίαση των αποτελεσμάτων του πειράματος των Hamad, Saad και Abed (Hatem Hamad, Motaz Saad, and Ramzi Abed, 2010)

Οι Aijaz και Ali (Fahad Aijaz, Syed Zahid Ali, Muzzamil Aziz Chaudhary, Bernhard Walke, 2009) συνέκριναν το REST και το SOAP με γνώμονα το HTTP ωφέλιμο φορτίο (HTTP payload) του αιτήματος. Για τις ανάγκες του πειράματος αναπτύχθηκαν δύο διακριτές δομές URL με στόχο να δοκιμαστούν τόσο η αλληλεπίδραση με σύγχρονες υπηρεσίες (synchronous services) όσο και εκείνη με ασύγχρονες υπηρεσίες (asynchronous services). Στα σύγχρονα αιτήματα ένα πόρος μπορεί να αναγνωριστεί από το URL του ενώ στα ασύγχρονα αιτήματα η αλληλεπίδραση του πελάτη με το web service πρέπει να υποστηρίζεται, όπως επίσης ο έλεγχος και η παρακολούθηση, με αποτέλεσμα τα μηνύματα να είναι πιο σύνθετα.

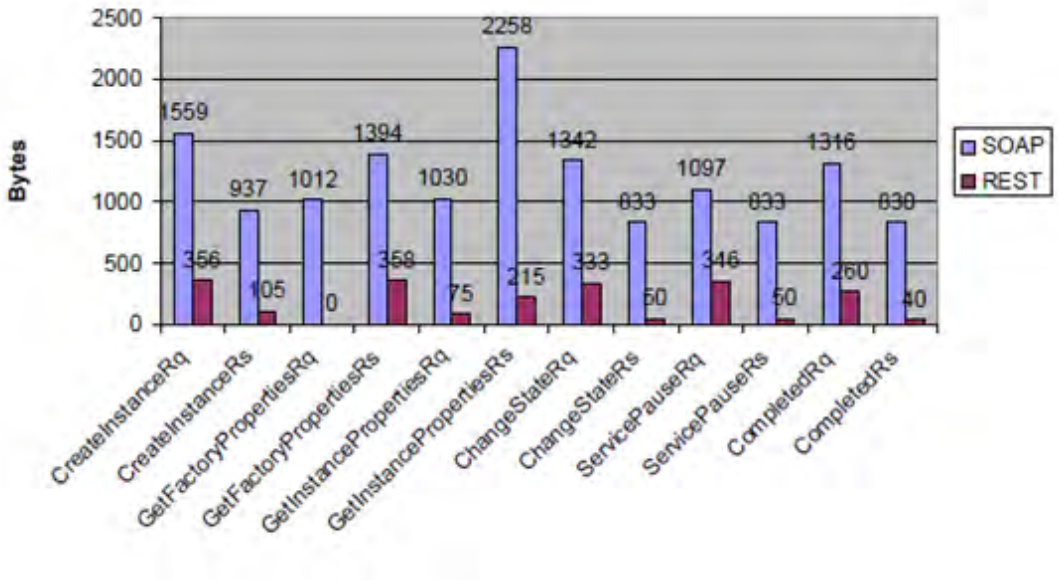
Ένα απλό σύγχρονο web service που λαμβάνει μία συμβολοσειρά και τη στέλνει πίσω στον πελάτη χρησιμοποιείται ώστε να επιδείξει τη διαφορά στο ωφέλιμο φορτίο για την πρώτη περίπτωση (σύγχρονη υπηρεσία). Έπειτα από κλήση τόσο της REST εφαρμογής όσο και της SOAP παρατηρείται μία μεγάλη διαφοροποίηση στο HTTP ωφέλιμο φορτίο. Το SOAP χρησιμοποιεί 580 bytes για τη δημιουργία της υπηρεσίας ενώ το REST χρησιμοποιεί μόλις 20 bytes. Επίσης, τα αντίστοιχα νούμερα για την απάντηση είναι 537 bytes και 20 bytes αντίστοιχα. Η διαφορά απεικονίζεται στην εικόνα 2.12.



Εικόνα 2.12 Σύγκριση του HTTP ωφέλιμου φορτίου σε σύγχρονες web services (Fahad Aijaz, Syed Zahid Ali, Muzzamil Aziz Chaudhary, Bernhard Walke, 2009)

Το ίδιο web service που λαμβάνει τη συμβολοσειρά και την επιστρέφει στον πελάτη χρησιμοποιήθηκε και στη δεύτερη περίπτωση με τη διαφορά πως η υλοποίηση έγινε με τέτοιο τρόπο ώστε η υπηρεσία να συμπεριφέρεται με ασύγχρονο τρόπο. Η διαφορά είναι και πάλι πολύ μεγάλη. Για παράδειγμα, για τη δημιουργία ενός αιτήματος, το SOAP χρειάζεται 1559 bytes σε αντίθεση με το REST το οποίο χρειάζεται μόλις 356 bytes. Μία μέθοδος αιτήματος αλλαγής κατάστασης (change state request method) χρησιμοποιεί 1342 bytes με το SOAP και 333 bytes με το REST ενώ τα νούμερα είναι ακόμα πιο εντυπωσιακά για κλήσεις ιδιοτήτων (get instance properties requests) όπου το SOAP χρειάζεται 2258 bytes σε αντίθεση με τα 215 bytes που χρειάζεται το REST. Τα αποτελέσματα απεικονίζονται στην εικόνα 2.13.

Μελετώντας τα αποτελέσματα των πειραμάτων που αναφέρθηκαν παραπάνω είναι εμφανές πως, με γνώμονα την απόδοση, το REST είναι καλύτερη επιλογή από το SOAP. Λαμβάνωντας υπ' όψιν και τα χαρακτηριστικά που προσφέρει το REST, τα οποία αναλύθηκαν παραπάνω και τη σημαντικότητά τους (για παράδειγμα το γεγονός πως το REST δε χρησιμοποιεί καταστάσεις είναι πολύ σημαντικό λαμβάνοντας υπ' όψιν την ασταθή φύση των συνδέσεων στο Internet για κινητά τηλέφωνα και tablet PCs), καταλήγουμε στο συμπέρασμα πως το REST είναι μία πολύ πιο ενδεδειγμένη επιλογή από το SOAP για εφαρμογές κινητών τηλεφώνων και tablet PCs.



Εικόνα 2.13 Σύγκριση του HTTP ωφέλιμου φορτίου σε ασύγχρονες web services (Fahad Aijaz, Syed Zahid Ali, Muzzamil Aziz Chaudhary, Bernhard Walke, 2009)

3 Σχεδίαση και υλοποίηση συστήματος

3.1 Ανάλυση Προδιαγραφών

Στο κεφάλαιο αυτό παραθέτονται οι προδιαφραφές της εργασίας που συζητήθηκαν τόσο με γιατρούς όσο και με τον επιβλέποντα της παρούσας πτυχιακής εργασίας και συμφωνήθηκαν πριν αρχίσει η υλοποίηση.

3.1.1 Λειτουργικές προδιαγραφές

Παρακάτω αναλύονται τα κύρια χαρακτηριστικά του συστήματος παραθέτοντας τις ενέργειες οι οποίες θα μπορούν να γίνουν καθώς και τις προδιαγραφές αυτών των χαρακτηριστικών.

3.1.1.1 Προδιαγραφές πελάτη κινητής συσκευής (Mobile Client Requirements - MCR)

- **MCR-01**: Η εφαρμογή θα παρέχει τη λειτουργικότητα δημιουργίας, προβολής, επεξεργασίας και διαγραφής ραντεβού.
- **MCR-02**: Η εφαρμογή θα παρουσιάζει όλα τα αποθηκευμένα ραντεβού σε μία οθόνη με τη δομή ημερολογίου.
- **MCR-03**: Η εφαρμογή θα παρουσιάζει αναλυτικά το κάθε ραντεβού παραθέτοντας όλες τις λεπτομέρειες του καθώς και τις επαφές που εμπλέκονται με το ραντεβού.
- **MCR-04**: Η εφαρμογή θα παρέχει τη λειτουργικότητα δημιουργίας, προβολής, επεξεργασίας και διαγραφής επαφής.
- **MCR-05**: Η εφαρμογή θα διατηρεί επεξεργάσιμες καρτέλες των επαφών καθώς και λίστα με όλες τις επαφές.
- **MCR-06**: Η εφαρμογή θα παρέχει τη λειτουργικότητα δημιουργίας, προβολής, επεξεργασίας και διαγραφής σύντομων σημειώσεων (todo notes)..

3.1.1.2 Προδιαγραφές εξυπηρετητή (Server Requirements - SR)

- **SR-01**: Η ιστοσελίδα θα παρέχει τη λειτουργικότητα δημιουργίας, προβολής, επεξεργασίας και διαγραφής ραντεβού.
- **SR-02**: Η ιστοσελίδα θα παρουσιάζει όλα τα αποθηκευμένα ραντεβού σε μία οθόνη με τη δομή ημερολογίου
- **SR-03**: Η ιστοσελίδα θα παρέχει τη λειτουργικότητα δημιουργίας, προβολής, επεξεργασίας και διαγραφής επαφής.

- **SR-04**: Η ιστοσελίδα θα παρέχει τη λειτουργικότητα δημιουργίας, προβολής, επεξεργασίας και διαγραφής σύντομων σημειώσεων (todo notes).
- **SR-05**: Η ιστοσελίδα θα μπορεί να δέχεται αιτήματα δημιουργίας ραντεβού, επαφής και σύντομης σημείωσης από την εφαρμογή.
- **SR-06**: Η ιστοσελίδα θα εμφανίζεται σωστά σε όλες τις οθόνες, ανεξαρτήτως τύπου, μεγέθους και ανάλυσης (responsive design).

3.1.2 Μη λειτουργικές προδιαγραφές

Παρακάτω παραθέτονται γενικά χαρακτηριστικά του συστήματος.

3.1.2.1 Προδιαγραφές ασφαλείας (Security Requirements – SER)

- **SER-01**: Η εφαρμογή θα διαθέτει οθόνη login απαγορεύοντας τη χρήση της από μη εξουσιοδοτημένους χρήστες.
- **SER-02**: Η ιστοσελίδα θα διαχωρίζει τους χρήστες σε ομάδες ανάλογα με τα επίπεδα δικαιωμάτων τους.
- **SER-03**: Η επικοινωνία πελάτη – εξυπηρετητή θα βασιστεί σε RESTful web services. Τα δεδομένα θα μεταφέρονται με μορφή JSON.

3.1.2.2 Εκδόσεις λογισμικού/λειτουργικών συστημάτων (Software/Operating Systems Versions)

- **SOSV-01**: Η εφαρμογή θα μπορεί να χρησιμοποιηθεί από συσκευές κινητών τηλεφώνων και tablet PCs με λειτουργικό σύστημα Android 4.0.3 (Ice Cream Sandwich – API level 15) και άνω.

3.1.2.3 Επιλογή τεχνολογιών (Technologies Selection)

- **TS-01**: Η εφαρμογή θα αναπτυχθεί με χρήση Java.
- **TS-02**: Η εφαρμογή θα υποστηρίζεται από βάση δεδομένων SQLite.
- **TS-03**: Η ιστοσελίδα θα αναπτυχθεί με χρήση του Drupal 7 CMS (core 7.28).
- **TS-04**: Η ιστοσελίδα θα υποστηρίζεται από βάση δεδομένων MySQL.
- **TS-05**: Για την τοπική ανάπτυξη της εφαρμογής θα χρησιμοποιηθούν Apache Web Server 2.2.22, PHP 5.3.18 και MySQL Database Server 5.1.66.

3.2 Αρχιτεκτονική εφαρμογής

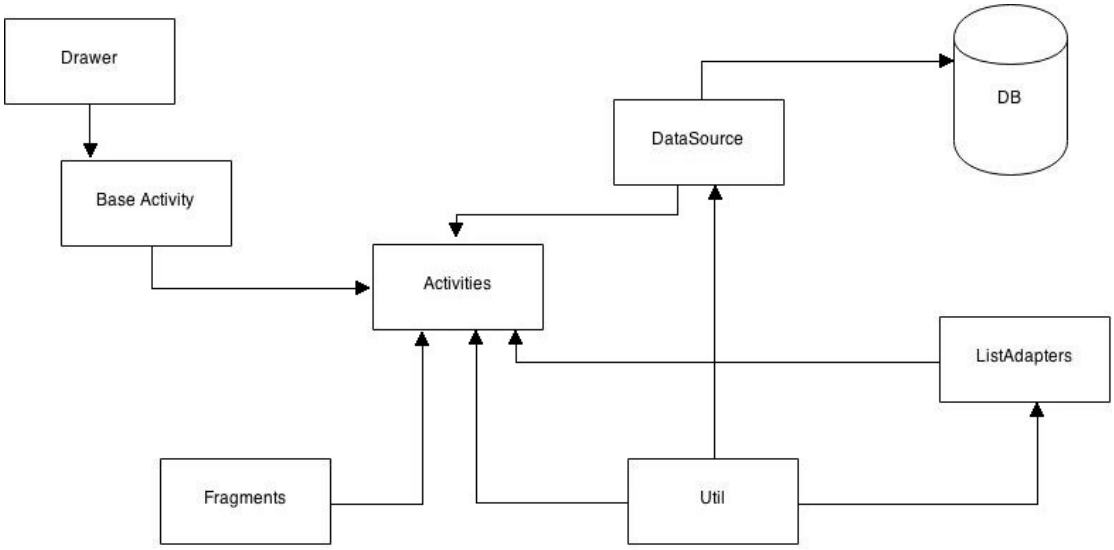
Βασικός στόχος της αρχιτεκτονικής της εφαρμογής είναι να διαχωρίσει τους ρόλους και τα καθήκοντα των κλάσεων του κώδικα. Με αυτό τον τρόπο η κάθε ομάδα κλάσεων (package) έχει δημιουργηθεί για συγκεκριμένο σκοπό με τελικό στόχο να παρέχει λειτουργικότητα στα activities. Ένα σύνολο βασικών πλεονεκτημάτων δημιουργείται με αυτό το διαχωρισμό:

- Η **συντήρηση του κώδικα** γίνεται πιο απλή και γρήγορη εργασία. Σε περίπτωση αλλαγών υπάρχει σαφής δομή και διαχωρισμός με αποτέλεσμα να επιταχύνεται η διαδικασία αλλά και να μειώνεται η πιθανότητα λάθους.

- **Ο κώδικας τροποποιείται για συγκεκριμένους λόγους**. Μία κλάση θα αλλάξει μόνο εφόσον υπάρχει η ανάγκη να αλλάξει. Το συγκεκριμένο σημείο θα γίνει πιό κατανοητό μέσω ενός απλού παραδείγματος. Θεωρώντας πως έχουμε δημιουργήσει μία κλάση η οποία είναι υπεύθυνη για την επικοινωνία με τη βάση δεδομένων, αν στο μέλλον προκύψει η ανάγκη να αλλάξει κάποιο ερώτημα (query) θα τροποποιηθεί μόνο αυτή η κλάση (και όχι πιθανώς κάποια που είναι υπεύθυνη για την εμφάνιση ενός activity).

- **Ο κώδικας γίνεται επαναχρησιμοποιήσιμος**. Οι κλάσεις ομαδοποιούνται σε δομές που αρχίζουν να θυμίζουν βιβλιοθήκες με αποτέλεσμα να προσφέρονται για μελλοντική χρήση.

Η εικόνα 3.1 παρουσιάζει την αρχιτεκτονική της εφαρμογής.



Εικόνα 3.1 Αρχιτεκτονική εφαρμογής

Εύκολα παρατηρούμε πως το κεντρικό στοιχείο της εφαρμογής είναι τα activities. Οι υπόλοιπες ομάδες κώδικα ουσιαστικά λειτουργούν υποστηρικτικά σε αυτές τις κλάσεις. Γίνεται αντιληπτό πως αυτή η δομή είναι λειτουργική καθώς τα activities είναι ο κώδικας που τελικώς «φτάνει» στο χρήστη και καθορίζει τι βλέπει στην οθόνη. Ο κώδικας των κλάσεων activities ασχολείται κυρίως με θέματα εμφάνισης και λειτουργικότητας της εκάστοτε οθόνης. Φροντίζει για τη σωστή αρχικοποίηση των οθονών, προβλέπει γεγονότα του κύκλου ζωής και αλληλεπιδρά με το χρήστη.

Η πρώτη σχέση εξάρτησης του κώδικα που θα μελετήσουμε είναι αυτή με τις κλάσεις που ασχολούνται με την επικοινωνία με τη βάση δεδομένων (DataSource κλάσεις). Παρατηρούμε πως η ίδια η βάση δεδομένων είναι εντελώς αποκομμένη από τις κλάσεις των activities και όλη η επικοινωνία μεταξύ τους βασίζεται στην ενδιάμεση κλάση (DataSource). Με αυτό τον τρόπο τα activities δε «γνωρίζουν» για τις λεπτομέρειες της υλοποίησης της βάσης δεδομένων (όπως για παράδειγμα το σχήμα της), αλλά περιορίζονται στη δυνατότητα να μπορούν να διαβάζουν και να γράφουν τις πληροφορίες που τους ενδιαφέρουν. Αν ένα ερώτημα (query) αλλάξει δε θα το «μάθει» ποτέ κάποιο activity, το οποίο θα συνεχίσει να διαβάζει και να γράφει δεδομένα. Η κλάση DataSource αντιθέτως περιλαμβάνει όλες τις λεπτομέρειες της υλοποίησης της βάσης δεδομένων. Αφαιρετικά, θα μπορούσαμε να πούμε πως η κλάση αυτή παρέχει μία διεπαφή (interface) με τη βάση δεδομένων. Έχει υλοποιημένες μεθόδους που τρέχουν ερωτήματα (queries) και φροντίζει με τη χρήση βοηθητικών POJO (Plain Old Java Object – κλάσεις-αντικείμενα που διευκολύνουν στη μοντελοποίηση εννοιών μέσα στον κώδικα) κλάσεων να μεταφέρει πληροφορίες με ασφάλεια και ακεραιότητα ανάμεσα στα activities και στη βάση δεδομένων.

Ένα βασικό κομμάτι της εφαρμογής είναι η κλάση BaseActivity, η οποία κληρονομείται από όλα τα activities και παρέχει το βασικό σετ της λειτουργικότητάς τους. Οποιαδήποτε μέθοδος είναι κοινή ανάμεσα στα activities έχει υλοποιηθεί σε αυτή την κλάση πατέρα. Με αυτό τον τρόπο αποφεύγεται ο διπλός κώδικας.

Ένα χαρακτηριστικό παράδειγμα τέτοιας λειτουργικότητας είναι το «συρταρωτό» μενού. Η κλάση BaseActivity χρησιμοποιεί τις κλάσεις που είναι υπεύθυνες για αυτό το μενού (Drawer) και εισάγει την πλήρη λειτουργικότητα γι’ αυτό η οποία παραμένει κοινή σε όλα τα activities. Με αυτό τον τρόπο, μεταξύ άλλων πλεονεκτημάτων, διατηρείται μία αίσθηση συνοχής ανάμεσα στα activities η οποία βοηθάει στην εξοικείωση του χρήστη με την εφαρμογή.

Όσον αφορά στην εμφάνιση των οθονών, οι κλάσεις Fragments και ListAdapters χρησιμοποιούνται παρέχοντας είτε έτοιμα κομμάτια της διεπαφής χρήστη είτε τον «οδηγό» για τις λίστες που πιθανώς χρησιμοποιεί η οθόνη. Με αυτό τον τρόπο διατηρείται μία σταθερή δομή αναλόγως με τον τύπο της οθόνης (για παράδειγμα οθόνη λίστας).

Τέλος, οι κλάσεις Util έχουν υλοποιηθεί για βοηθητικούς σκοπούς. Στην πλειοψηφία τους είναι κλάσεις POJO. Εξυπηρετούν σε διάφορα κομμάτια της εφαρμογής, όπως για παράδειγμα στη μεταφορά δεδομένων από και προς τη βάση δεδομένων.

3.3 Υλοποίηση web service

Όπως αναλύθηκε στην ενότητα 2.3 υπήρχαν δύο επιλογές για την υλοποίηση του web service: REST και SOAP. Μελετώντας τα χαρακτηριστικά της εφαρμογής, αλλά και συγκρίνοντας τις δύο τεχνολογίες, επιλέχθηκε η REST. Όσον αφορά την απόδοση η REST προσφέρει σημαντικά πλεονεκτήματα σε σχέση με τη SOAP τεχνολογία. Επιπλέον, λαμβάνοντας υπ' όψιν τα χαρακτηριστικά και τα πλεονεκτήματα της REST, τα οποία αναλύθηκαν εκτενώς στην ενότητα 2.3.2 και είναι πολύ σημαντικά για την εφαρμογή (για παράδειγμα, το γεγονός ότι λειτουργεί χωρίς καταστάσεις είναι πολύ μεγάλο όφελος αναλογιζόμενοι την ασταθή φύση των συνδέσεων στο διαδίκτυο συσκευών κινητών τηλεφώνων και tablet PCs) συμπεραίνουμε πως η REST είναι καλύτερη επιλογή από τη SOAP για τις ανάγκες της εφαρμογής.

Για την υλοποίηση του web service χρησιμοποιήθηκε το module «Services» (module, 2014) που υπάρχει διαθέσιμο για Drupal. Το συγκεκριμένο module, μετά την εγκατάστασή του παρέχει κάποιες βασικές κλήσεις, αλλά και ένα API σε PHP για την ανάπτυξη ειδικευμένων κλήσεων από τον προγραμματιστή. Το δεύτερο χαρακτηριστικό ήταν αυτό που αξιοποιήθηκε τελικώς με αποτέλεσμα την ανάπτυξη ενός εντελώς ειδικευμένου στην εφαρμογή web service.

Το module που αναπτύχθηκε ονομάστηκε «ws» ενώ μπορεί να βρεθεί στο ακόλουθο σχετικό μονοπάτι (relative path) «daybook\sites\all\modules», στο οποίο τοποθετούνται όλα τα ειδικευμένα modules (custom modules) σε μία ιστοσελίδα που έχει δημιουργηθεί με Drupal.

Το .info αρχείο που είναι απαραίτητο σε κάθε Drupal module περιέχει σημαντική πληροφορία για το module. Παρακάτω παραθέτονται και έπειτα επεξηγούνται τα σημεία κλειδί του αρχείου:

```
name = Daybook Web Servive
description = Custom web service for Daybook
core = 7.x
package = Custom
dependencies[] = services
```

Ουσιαστικά πρόκειται για ένα σύνολο παραμέτρων που ορίζουν χαρακτηριστικά του module. Συγκεκριμένα το «name» αφορά στο όνομα του module, το οποίο θα είναι ορατό στη διαχειριστική σελίδα της ιστοσελίδας, όπως και η περιγραφή που καθορίζεται από την τιμή της παραμέτρου «description». Το «core» είναι από τις σημαντικότερες παραμέτρους καθώς καθορίζει την έκδοση του Drupal με την οποία το module είναι συμβατό (στη συγκεκριμένη περίπτωση είναι συμβατό με όλες τις εκδόσεις του Drupal 7). Το «package» χρησιμοποιείται στην ομαδοποίηση των modules στη διαχειριστική σελίδα της ιστοσελίδας ενώ τέλος, το «dependencies»

είναι επίσης ιδιαίτερα σημαντικό ορίζοντας τα modules από τα οποία εξαρτάται το συγκεκριμένο. Για παράδειγμα, το module που υλοποιήθηκε δε θα μπορεί να εγκατασταθεί σε μία Drupal ιστοσελίδα στην οποία δεν έχει εγκατασταθεί νωρίτερα το module «services».

Παρακάτω ακολουθεί μία ανάλυση του κώδικα του web service η οποία θα παραμείνει σε εισαγωγικό επίπεδο με στόχο να μην κουράσει τον αναγνώστη.

Τα αρχεία που περιέχουν τον κώδικα του web service είναι τα ws.module και ws.resource.sync.inc. Τα δύο αρχεία είναι γραμμένα σε PHP. Το ws.module ορίζει το σύνολο των κλήσεων που είναι διαθέσιμες (καθώς και διαχειρίσιμες ως προς την ενεργοποίηση/απενεργοποίηση τους από το διαχειριστή της ιστοσελίδας). Παρακάτω ακολουθεί και επεξηγείται ένα μικρό κομμάτι του κώδικα:

```
        'retrieve_todos' => array(
          'help' => 'Retrieves all todos',
          'file' => array('type' => 'inc', 'module' => 
'ws', 'name' => 'resources/ws.resource.sync'),
          'callback' => '_ws_resource_retrieve_todos',
          'access callback' => '_ws_resource_access',
          'args' => array(
            array(
              'name' => 'uid',
              'optional' => FALSE,
              'source' => array('data' => 'uid'),
              'type' => 'int',
              'description' => 'The user whose todos are 
to be retrieved',
            ),
            array(
              'name' => 'last_sync',
              'optional' => FALSE,
              'source' => array('data' => 'last_sync'),
              'type' => 'string',
              'description' => 'The last synchronization 
date (leave empty to ignore)',
            ),
          ),
        ),
```

Το παραπάνω κομμάτι κώδικα είναι υπεύθυνο για τον ορισμό της κλήσης που επιστρέφει το σύνολο των σημειώσεων του χρήστη. Πρόκειται για κομμάτι ενός πίνακα (array). Μεταξύ άλλων ορίζεται το όνομα και η περιγραφή της κλήσης καθώς και που μπορεί να βρεθεί η υλοποίησή της (resources/ws.resource.sync) και ποιά είναι η μέθοδος η οποία θα κληθεί όταν πραγματοποιηθεί η κλήση (_ws_resource_retrieve_todos), η οποία βρίσκεται υλοποιημένη στο αρχείο που ορίστηκε παραπάνω. Έπειτα προσδιορίζεται η πρόσβαση σε αυτή την κλήση μέσω της συνάρτησης «_ws_resource_access» η οποία βρίσκεται επίσης στο παραπάνω αρχείο και κάνει χρήση των μεθόδων του API του Drupal για τη διαχείριση της πρόσβασης σε πόρους.

Τέλος, ορίζεται ένας πίνακας από πίνακες ο οποίος ορίζει τις παραμέτρους που δέχεται η κλήση. Για κάθε παράμετρο απαιτείται το όνομά της, η σημαία που καθορίζει αν είναι υποχρεωτική, ο τρόπος με τον οποίο θα περαστεί, ο τύπος της και η περιγραφή της.

Το αρχείο ws.resource.sync.inc περιέχει την υλοποίηση των μεθόδων που έχουν οριστεί, οι οποίες περιέχουν τη λογική της κλήσεις. Παρακάτω παραθέτεται και επεξηγείται το αντίστοιχο κομμάτι για την κλήση που περιγράφηκε από το αρχείο ws.module:

```
/**
 * Retrieve all todos of a particular user
 */
function _ws_resource_retrieve_todos($uid, $last_sync) {
    $results = array();
    // Get the node ID of all user todos
    $query_results = db_select('node')
                     ->fields('node', array('nid'))
                     ->condition('type', 'todo')
                     ->condition('changed',    $last_sync,
'>=')
                     ->condition('uid', $uid)
                     ->execute()
                     ->fetchAll();

    foreach ($query_results as $key => $value) {
        $todo = node_load($value->nid);
        array_push($results, array(            // append
array with useful contact info for the mobile app (id,
first name, last name, address, phone)
```

```
                "id"        => $todo->nid,
                "title"     => $todo->title,
                "body"                          =>    $todo-
>body['und']['0']['value'],
            )
        );
    }
    return $results;
}
```

Η συνάρτηση «_ws_resource_retrieve_todos» επιστρέφει όλες τις σημειώσεις του χρήστη από την τελευταία ενημέρωση (τελευταία κλήση της συνάρτησης).

Αρχικά ορίζεται και εκτελείται ένα ερώτημα (query) στη βάση δεδομένων της ιστοσελίδας όπου ζητάει όλες τις σημειώσεις για ένα συγκεκριμένο χρήστη (προσδιορίζεται από την παράμετρο «uid») από την τελευταία κλήση (προσδιορίζεται από την παράμετρο «last_sync»). Έπειτα με μία επαναληπτική διαδικασία τα δεδομένα που επιστράφηκαν από τη βάση τοποθετούνται σε ένα πίνακα (array) ο οποίος επιστρέφεται.

3.4 Υλοποίηση ιστοσελίδας

Όπως αναλύθηκε και στην ενότητα 2.2.1 το Drupal παρέχει ένα βασικό κορμό που είναι κοινός για όλες τις ιστοσελίδες και έπειτα ο προγραμματιστής είναι υπεύθυνος να επεκτείνει τη βασική λειτουργικότητα ανάλογα με τις ανάγκες του υλοποιώντας εξειδικευμένα modules. Ο τρόπος με τον οποίο λειτουργούν αυτά αναλύθηκε με το παράδειγμα του module για το web service. Κατ' αντιστοιχία ο προγραμματιστής μπορεί να υλοποιήσει modules που καλύπτουν το σύνολο των απαιτήσεων της ιστοσελίδας.

Σε αυτό το σημείο θα αναφερθούμε στην εμφάνιση της ιστοσελίδας. Ένα από τα πλέον σημαντικά σημεία εστίασης της προσοχής του προγραμματιστή κατά την ανάπτυξη ιστοσελίδων είναι η εμφάνισή της. Συγκεκριμένα είναι κομβικό σημείο η ιστοσελίδα να εμφανίζεται και να αλληλεπιδρά σωστά με το χρήστη σε όλες τις οθόνες, τους φυλλομετρητές και τις συσκευές που μπορεί να φορτωθεί.

Το βασικό εργαλείο που χρησιμοποιήθηκε για την ανάπτυξη της ιστοσελίδας από άποψη μακέτας είναι τα CSS3. Με τη χρήση του module Omega (Drupal, 2014) η ιστοσελίδα χωρίζεται σε πολλές περιοχές (divs) τις οποίες πλέον ο προγραμματιστής μπορεί να τροποποιήσει ώστε να επιτευχθεί το επιθυμητό αποτέλεσμα στον τομέα της εμφάνισης.

Η μακέτα που αναπτύχθηκε για την ιστοσελίδα μπορεί να βρεθεί στο σχετικό μονοπάτι «daybook\sites\all\themes\daybook». Σε αυτό το φάκελο μεταξύ άλλων βρίσκεται το .info αρχείο, οι εικόνες που χρησιμοποιούνται και τα .css αρχεία. Παρακάτω δίνεται και επεξηγείται ένα παράδειγμα από τα CSS3 που έχουν χρησιμοποιηθεί για την ιστοσελίδα.

```
.node-type-page #zone-content h1#page-title,
.page-node-add h1#page-title,
.node-type-contact h1#page-title,
.node-type-todo h1#page-title,
.node-type-appointment h1#page-title {padding: 10px 20px;
background: #eee; border-bottom: 1px solid #ddd; border-
top: 1px solid #fff;

  -moz-border-top-left-radius: 5px;
    -webkit-border-top-left-radius: 5px;
    -khtml-border-top-left-radius: 5px;
    border-top-left-radius: 5px;

  -moz-border-top-right-radius: 5px;
```

```
    -webkit-border-top-right-radius: 5px;
    -khtml-border-top-right-radius: 5px;
    border-top-right-radius-radius: 5px;
}

.node-type-page #zone-content #region-content,
.page-node-add #zone-content .region-content-inner,
.node-type-contact .region-content-inner,
.node-type-todo .region-content-inner,
.node-type-appointment   .region-content-inner   {margin-
bottom: 20px; background: #fff; border: 1px solid #bbb;

     -moz-border-radius: 5px;
  -webkit-border-radius: 5px;
  -khtml-border-radius: 5px;
  border-radius: 5px;

  -moz-box-shadow: 0 0 3px #bbb;
     -webkit-box-shadow: 0 0 3px #bbb;
     box-shadow: 0 0 3px #bbb;

     -webkit-box-sizing: border-box;
     -moz-box-sizing: border-box;
     -ms-box-sizing: border-box;
     box-sizing: border-box;
}

.node-type-page #zone-content .content,
.page-node-add #zone-content .content,
.node-contact.view-mode-full,
.node-todo,
.node-appointment {padding: 10px 20px;}
.node-type-page #zone-content #region-content a,
.node-type-appointment  #zone-content  #region-content  a
{color:   #f0814a   !important;   text-decoration:   none
!important;}
```

Το παραπάνω κομμάτι είναι κάποιες βασικές μορφοποιήσεις για την πλειοψηφία των σελίδων της ιστοσελίδας. Μεταξύ άλλων ορίζονται στοιχεία εμφάνισης για τον τίτλο

των σελίδων, όπως οι αποστάσεις και το χρώμα του. Ένα σημείο που αξίζει να παρατηρηθεί ιδιαιτέρως είναι οι πολλαπλές δηλώσεις για να καλυφθούν οι διαφορετικές τεχνολογίες που χρησιμοποιούν διαφορετικοί φυλλομετρητές.

Με μία επιφανειακή ματιά ο αναγνώστης μπορεί να θεωρήσει εσφαλμένα πως τα CSS καλύπτουν μόνο στοιχεία εμφάνισης όπως το χρώμα, τα πλαίσια, οι σκιές κλπ. Παρόλα αυτά, η χρήση τους λύνει πολύ πιο σημαντικά προβλήματα, όπως τη δομή (layout) της ιστοσελίδας. Είναι μέγιστης σημασίας η σωστή χρήση των CSS για να διασφαλιστεί η σωστή εμφάνιση της σελίδας από άποψη δομής σε διαφορετικούς φυλλομετρητές, διαφορετικές συσκευές και διαφορετικά μεγέθη και αναλύσεις οθονών.

3.5 Εργαλεία

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε το Eclipse, ένα από τα πιο δημοφιλή και ευρέως χρησιμοποιούμενα περιβάλλοντα ανάπτυξης λογισμικού. Το Android Development Tools (ADT) είναι μία προσθήκη του Eclipse (plug-in) που επιτρέπει τη χρήση του περιβάλλοντος για την ανάπτυξη εφαρμογών για Android.

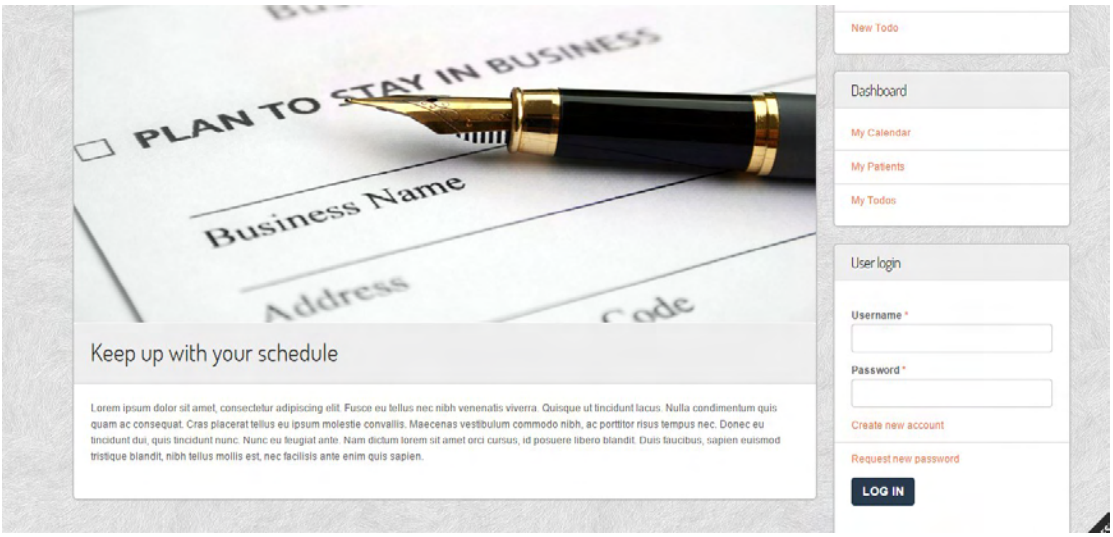
Για την τοπική ανάπτυξη της ιστοσελίδας χρησιμοποιήθηκε το Acquia Stack, το οποίο προσφέρει ένα περιβάλλον με Apache Web Server 2.2.22, PHP 5.3.18 και MySQL Database Server 5.1.66.

4 Εγχειρίδιο χρήστη

4.1 Ιστοσελίδα

4.1.1 Σύνδεση

Το πρώτο βήμα για ένα χρήστη κατά την επίσκεψη στην ιστοσελίδα είναι να συνδεθεί στο λογαριασμό του. Στην αρχική σελίδα, όπως φαίνεται στην εικόνα 4.1 βρίσκεται η φόρμα σύνδεσης.



Εικόνα 4.1 Φόρμα σύνδεσης στην ιστοσελίδα

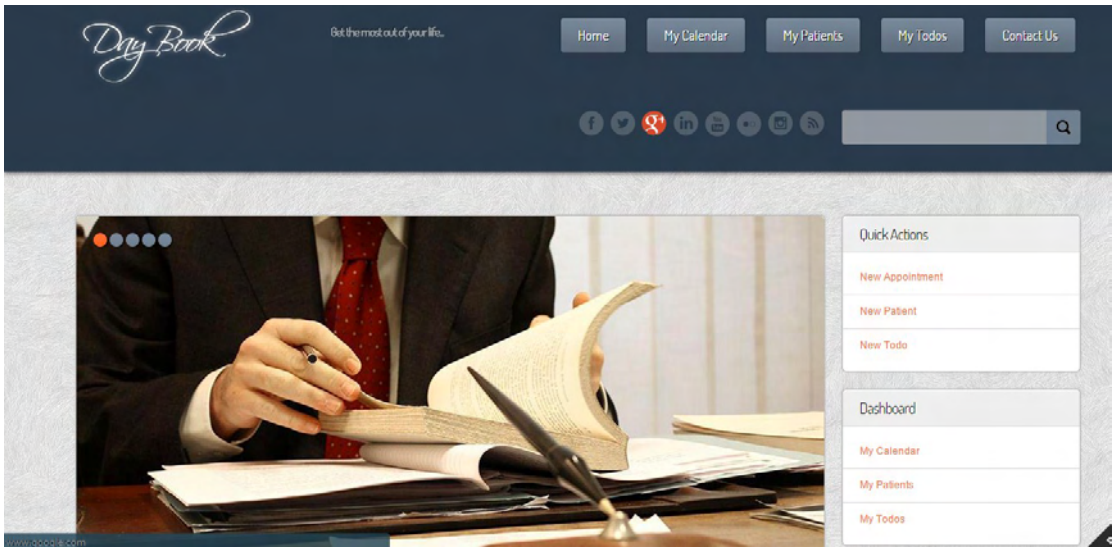
Ο χρήστης συμπληρώνει τα πεδία «username» και «password» και χρησιμοποιεί το κουμπί «Log in» για να συνδεθεί. Τόσο στην περίπτωση επιτυχούς σύνδεσης όσο και στην αποτυχημένη προσπάθεια, αντίστοιχα μηνύματα εμφανίζονται στην οθόνη.

Στην περίπτωση που ο χρήστης δε διαθέτει ήδη λογαριασμό στην ιστοσελίδα μπορεί να χρησιμοποιήσει το σύνδεσμο «Create new account».

Τέλος, εάν ο χρήστης έχει ξεχάσει τον κωδικό πρόσβασής του μπορεί να χρησιμοποιήσει το σύνδεσμο «Request new password».

4.1.2 Μενού δημιουργίας περιεχομένου

Στην αρχική σελίδα, καθώς και στην πλειοψηφία των εσωτερικών σελίδων στο δεξί μέρος της οθόνης εμφανίζεται το μενού δημιουργίας περιεχομένου, με τίτλο «Quick Actions», το οποίο φαίνεται στην εικόνα 4.2.



Εικόνα 4.2 Μενού δημιουργίας περιεχομένου

Το μενού παρέχει τρείς συνδέσμους οι οποίοι επιτρέπουν την εύκολη και γρήγορη μετάβαση στις σελίδες που ο χρήστης μπορεί να δημιουργήσει νέο περιεχόμενο: ραντεβού, ασθενή και σημείωση.

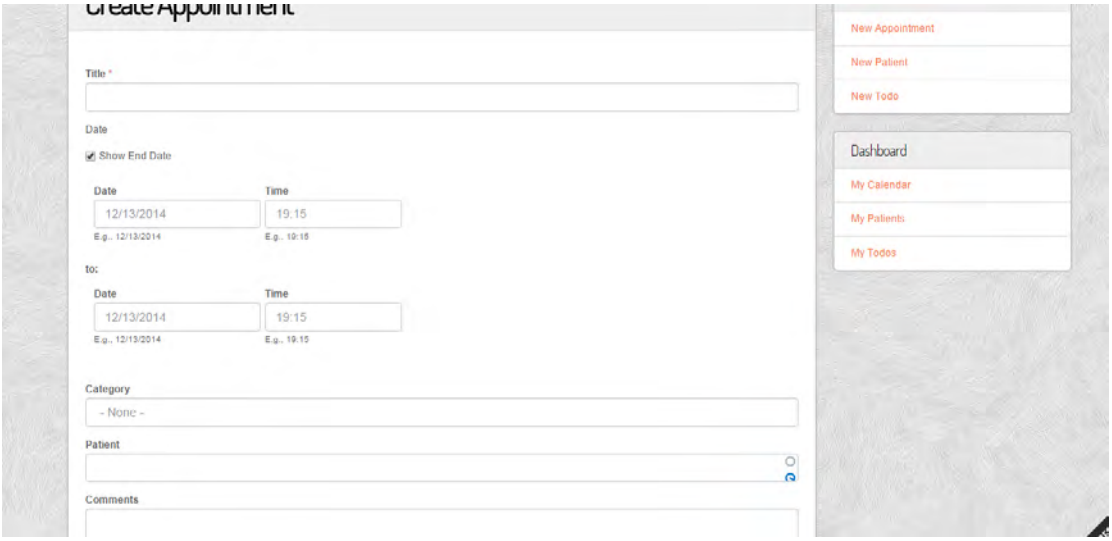
4.1.3 Δημιουργία ραντεβού

Κάνοντας χρήση του συνδέσμου «New Appointment» από το μενού δημιουργίας περιεχομένου, ο χρήστης πλοηγείται στη σελίδα δημιουργίας ενός νέου ραντεβού, η οποία φαίνεται στην εικόνα 4.3.

Για τη δημιουργία του ραντεβού συμπληρώνονται τα στοιχεία της φόρμας ως ακολούθως:

- **Title:** Ο τίτλος του νέου ραντεβού
- **Category:** Η κατηγορία του ραντεβού. Εδώ παρέχεται μία λίστα από την οποία ο χρήστης μπορεί να επιλέξει. Δεν υπάρχει η δυνατότητα εισαγωγής μίας νέας κατηγορίας σε αυτή την οθόνη.
- **Patient:** Ο ασθενής με την οποία θα πραγματοποιηθεί το ραντεβού. Ο χρήστης αρχίζει να πληκτρολογεί και το πεδίο «προτείνει» πιθανούς ασθενείς με βάση τους ήδη καταχωρημένους ασθενείς. Ο χρήστης καλείται να επιλέξει έναν απο αυτούς.
- **Comments:** Τα σχόλια που ο χρήστης επιθυμεί να συνοδεύουν το ραντεβού. Το πεδίο αυτό είναι ελεύθερου κειμένου.

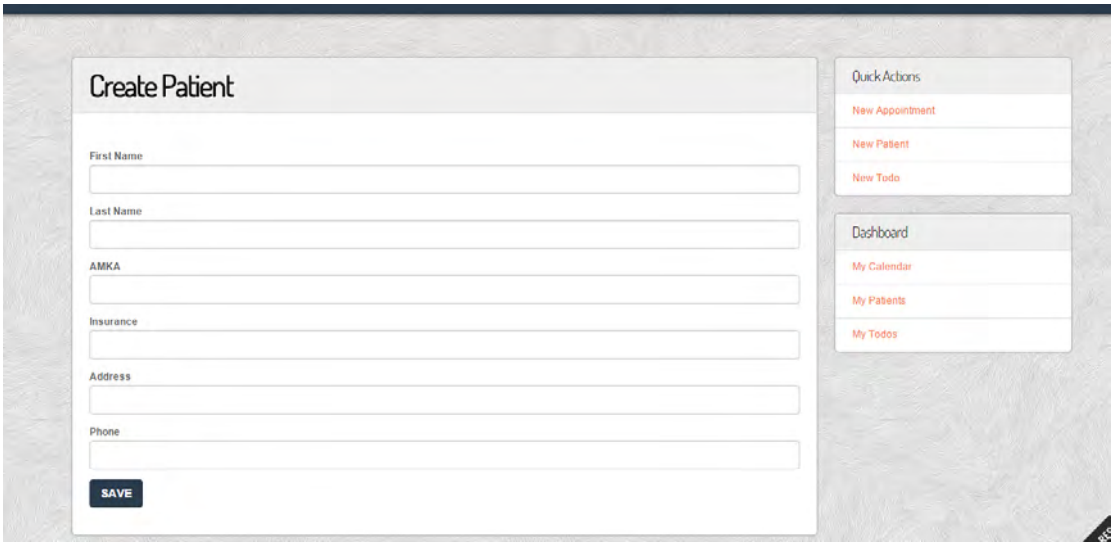
Έπειτα, εφόσον η φόρμα συμπληρωθεί σύμφωνα με τις επιθυμίες του χρήστη, με χρήση του κουμπιού «Save» το νέο ραντεβού δημιουργείται.



Εικόνα 4.3 Φόρμα δημιουργίας νέου ραντεβού

4.1.4 Δημιουργία ασθενούς

Κάνοντας χρήση του συνδέσμου «New Patient» από το μενού δημιουργίας περιεχομένου, ο χρήστης πλοηγείται στη σελίδα δημιουργίας ενός νέου ασθενούς, η οποία φαίνεται στην εικόνα 4.4.



Εικόνα 4.4 δημιουργίας νέου ασθενούς

4.1.5 Δημιουργία σημείωσης

Κάνοντας χρήση του συνδέσμου «New Todo» από το μενού δημιουργίας περιεχομένου, ο χρήστης πλοηγείται στη σελίδα δημιουργίας μίας νέας σημείωσης, η οποία φαίνεται στην εικόνα 4.5.

Για τη δημιουργία του ασθενούς συμπληρώνονται τα στοιχεία της φόρμας ως ακολούθως:

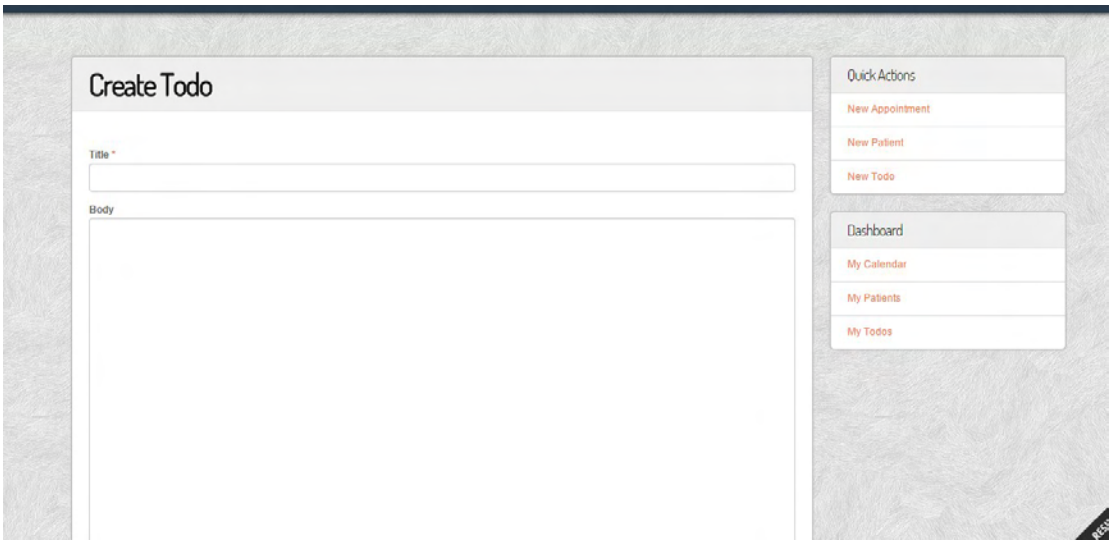
- **First Name:** Το όνομα του νέου ασθενούς.
- **Last Name:** Το επώνυμο του νέου ασθενούς.
- **ΑΜΚΑ:** Το ΑΜΚΑ του νέου ασθενούς.
- **Insurance:** Η ασφάλιση του νέου ασθενούς.
- **Address:** Η διεύθυνση του νέου ασθενούς.
- **Phone:** Το τηλέφωνο του νέου ασθενούς.

Έπειτα από τη συμπλήρωση της φόρμας, με χρήση του κουμπιού «Save» ο νέος ασθενής δημιουργείται.

Για τη δημιουργία της σημείωσης συμπληρώνονται τα στοιχεία της φόρμας ως ακολούθως:

- **Title:** Ο τίτλος της νέας σημείωσης.
- **Body:** Το κείμενο της νέας σημείωσης.

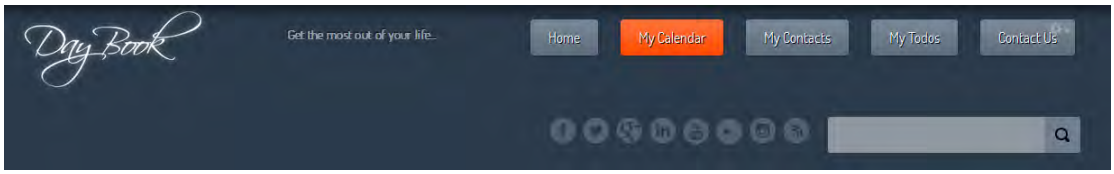
Έπειτα από τη συμπλήρωση της φόρμας, με χρήση του κουμπιού «Save» η νέα σημείωση δημιουργείται.



Εικόνα 4.5 Φόρμα δημιουργίας νέας σημείωσης

4.1.6 Τα ραντεβού μου

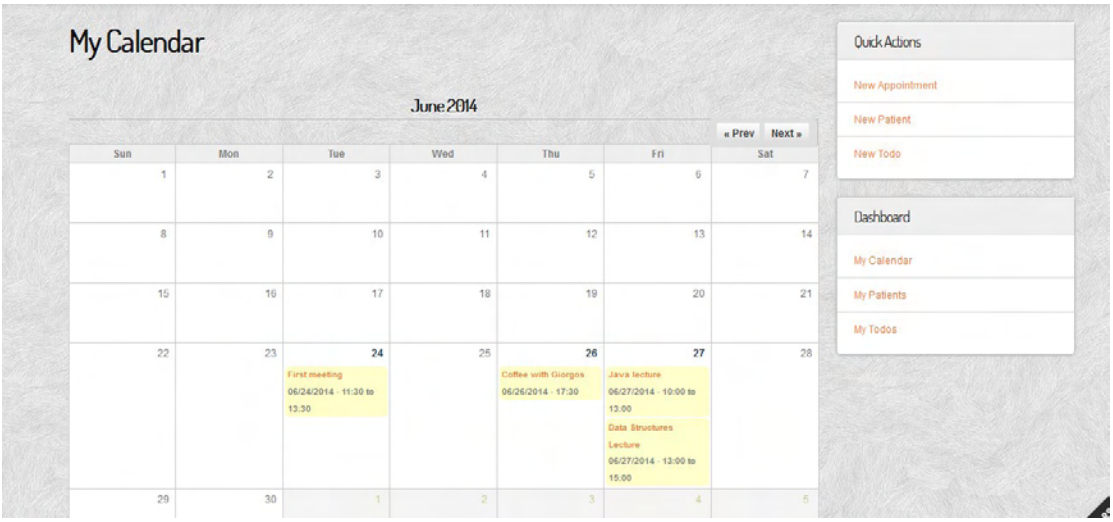
Μέσω της ιστοσελίδας ο χρήστης μπορεί να δει το σύνολο των ραντεβού του σε μορφή ημερολογίου. Η σελίδα είναι προσβάσιμη τόσο από το μενού στο δεξί μέρος της οθόνης με τίτλο «Dashboard» όσο και από το μενού στην κορυφή της σελίδας, με χρήση του συνδέσμου «My Calendar» και στις δύο περιπτώσεις, όπως φαίνεται στην εικόνα 4.6.



Εικόνα 4.6 Μενού κορυφής για την πλοήγηση στη σελίδα των ραντεβού

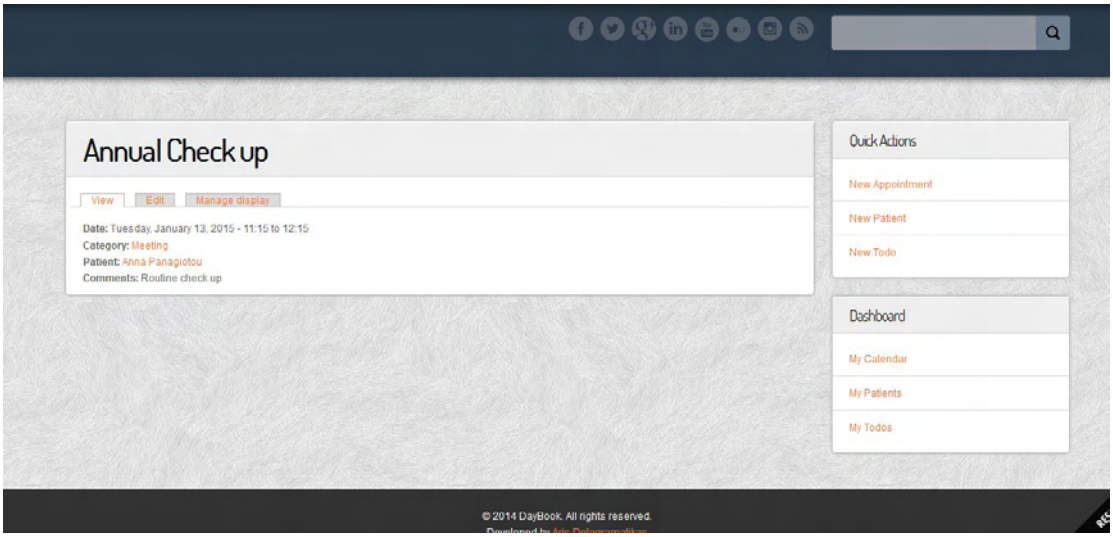
Ο χρήστης πλοηγείται στην οθόνη ημερολογίου που παρουσιάζει συνολικά τα ραντεβού του, η οποία φαίνεται στην εικόνα 4.7.

Το ημερολόγιο σε αυτή τη σελίδα είναι διαδραστικό και ο χρήστης μπορεί να πλοηγηθεί στους μήνες και να έχει μία συνολική εικόνα τόσο των περασμένων όσο και των μελλοντικών του ραντεβού.



Εικόνα 4.7 Παρουσίαση των ραντεβού του χρήστη με τη μορφή ημερολογίου

Οι βασικές πληροφορίες του κάθε ραντεβού (τίτλος, ημερομηνία/ώρα έναρξης και ημερομηνία/ώρα λήξης) παρουσιάζονται στο ημερολόγιο, με κλικ στον τίτλο του ραντεβού ο χρήστης πλοηγείται στη σελίδα του ραντεβού για το σύνολο των πληροφοριών του. Η σελίδα αυτή παρουσιάζεται στην εικόνα 4.8.

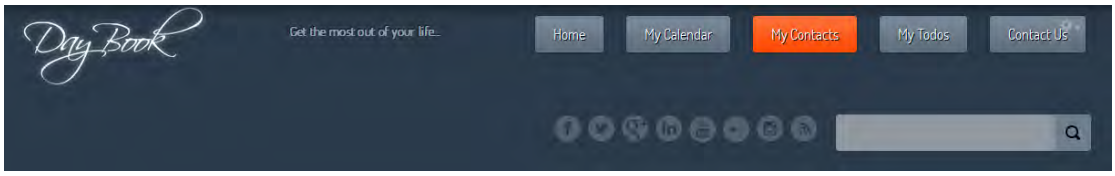


Εικόνα 4.8 Αναλυτική παρουσίαση ενός ραντεβού

Ο χρήστης μπορεί να επεξεργαστεί το ραντεβού με χρήση του κουμπιού «Edit».

4.1.7 Οι ασθενείς μου

Οι ασθενείς του χρήστη παρουσιάζονται συνολικά στην ιστοσελίδα με τη μορφή λίστας. Η σελίδα είναι προσβάσιμη τόσο από το μενού στο δεξί μέρος της οθόνης με τίτλο «Dashboard» όσο και από το μενού στην κορυφή της σελίδας, με χρήση του συνδέσμου «My Patients» και στις δύο περιπτώσεις, όπως φαίνεται στην εικόνα 4.9.

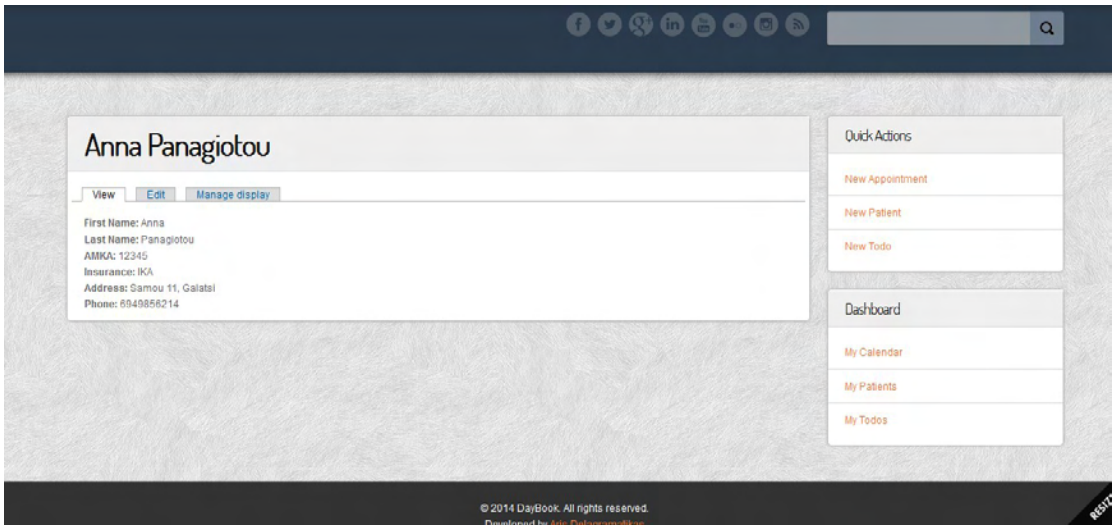


Εικόνα 4.9 Μενού κορυφής για την πλοήγηση στη σελίδα των ασθενών

Ο χρήστης πλοηγείται στην οθόνη των ασθενών του, η οποία φαίνεται στην εικόνα 4.10.

Εικόνα 4.10 Παρουσίαση των ασθενών του χρήστη με τη μορφή λίστας

Αντίστοιχα με τη σελίδα του ημερολογίου στην οποία παρουσιάζονται τα ραντεβού του χρήστη, έτσι και στην παρουσίαση των ασθενών του, κάποια βασική πληροφορία προσφέρεται στη λίστα (όνομα, διεύθυνση και τηλέφωνο) ενώ με κλικ στο όνομα, ο χρήστης πλοηγείται στην αναλυτική οθόνη του ασθενούς, η οποία παρουσιάζεται στην εικόνα 4.11.

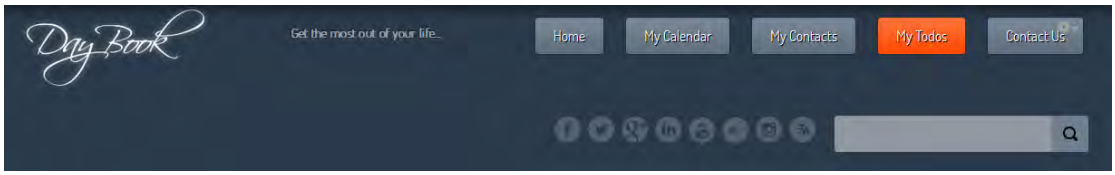


Εικόνα 4.11 Αναλυτική παρουσίαση ενός ασθενούς

Ο χρήστης μπορεί να επεξεργαστεί τον ασθενή με χρήση του κουμπιού «Edit».

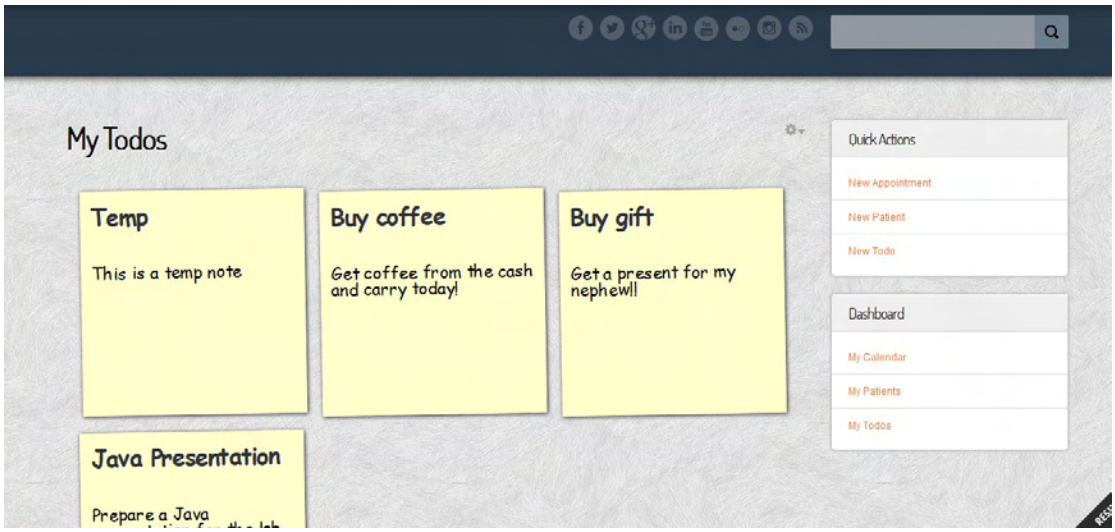
4.1.8 Οι σημειώσεις μου

Οι σημειώσεις του χρήστη εμφανίζονται ομαδοποιημένα στην ιστοσελίδα με τη μορφή μίας λίστας από χαρτιά σημειώματος (post it). Η σελίδα είναι προσβάσιμη τόσο από το μενού στο δεξί μέρος της οθόνης με τίτλο «Dashboard» όσο και από το μενού στην κορυφή της σελίδας, με χρήση του συνδέσμου «My Todos» και στις δύο περιπτώσεις, όπως φαίνεται στην εικόνα 4.12.



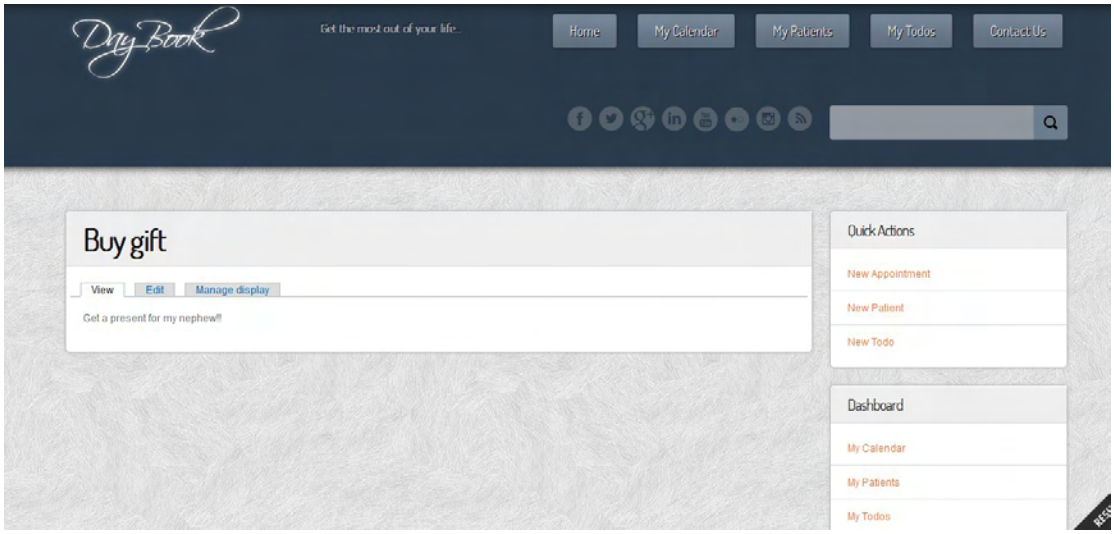
Εικόνα 4.12 Μενού κορυφής για την πλοήγηση στη σελίδα των σημειώσεων

Ο χρήστης πλοηγείται στην οθόνη των σημειώσεων του, η οποία φαίνεται στην εικόνα 4.13.



Εικόνα 4.13 Παρουσίαση των σημειώσεων του χρήστη με τη μορφή λίστας χαρτιών σημείωσης (post it)

Ακολουθώντας την καθιερωμένη και στις προηγούμενες οθόνες πολιτική, ένα μέρος της πληροφορίας εμφανίζεται αρχικά (τίτλος και κομμάτι του κειμένου) και ο χρήστης μπορεί να πλοηγηθεί μέσω του τίτλου στην αναλυτική σελίδα της σημείωσης, η οποία φαίνεται στην εικόνα 4.14.



Εικόνα 4.14 Αναλυτική παρουσίαση μίας σημείωσης

Ο χρήστης μπορεί να επεξεργαστεί τη σημείωση με χρήση του κουμπιού «Edit».

4.2 Εφαρμογή

4.2.1 Σύνδεση

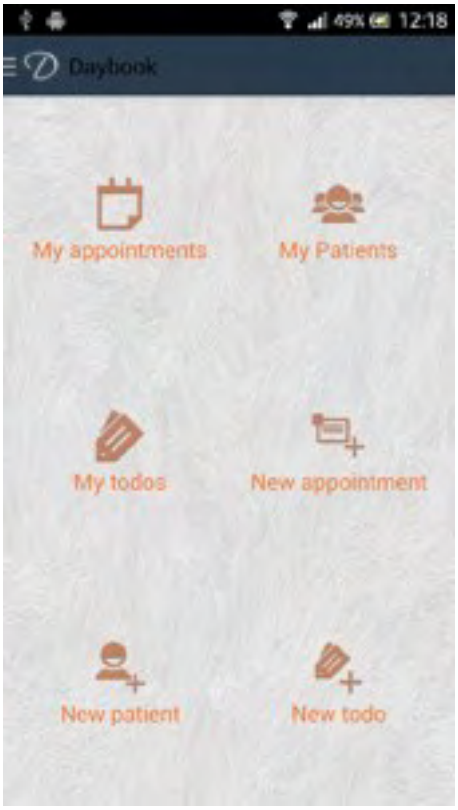
Το πρώτο βήμα του χρήστη της εφαρμογής είναι, όπως και στην ιστοσελίδα, να συνδεθεί. Η οθόνη σύνδεσης παρουσιάζεται στην εικόνα 4.15.

Εικόνα 4.15 Οθόνη σύνδεσης στην εφαρμογή

Ο χρήστης συμπληρώνει το όνομα χρήστη (username) και κωδικό (password) και με χρήση του κουμπιού «Sing in» επιχειρεί να συνδεθεί στην εφαρμογή. Σε περίπτωση εσφαλμένου ονόματος χρήστη ή κωδικού η εφαρμογή ενημερώνει το χρήστη για το λάθος του κι εκείνος μπορεί να προσπαθήσει ξανά να συνδεθεί. Σε περίπτωση επιτυχούς σύνδεσης, ο χρήστης πλοηγείται στην αρχική οθόνη της εφαρμογής.

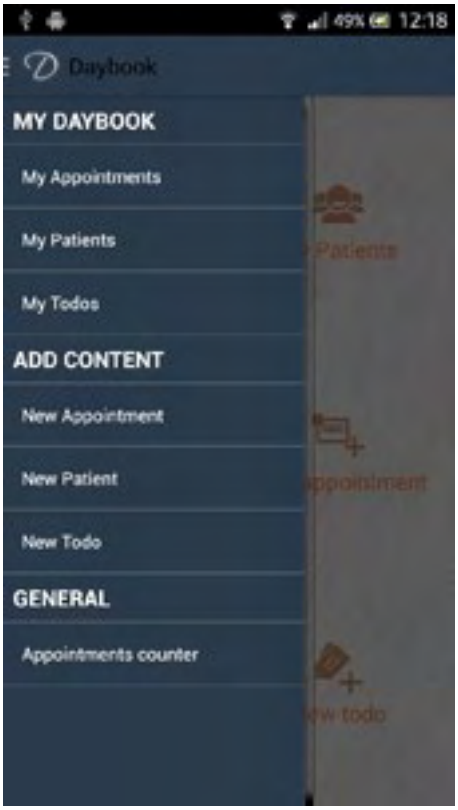
4.2.2 Αρχική οθόνη

Η αρχική οθόνη της εφαρμογής είναι ένα πλέγμα (dashboard). Μέσα από αυτή την οθόνη ο χρήστης έχει εύκολη και γρήγορη πρόσβαση στις βασικές λειτουργίες της εφαρμογής, οι οποίες παραθέτονται στη μορφή πίνακα με τίτλο και εικονίδια. Η οθόνη παρουσιάζεται στην εικόνα 4.16.



Εικόνα 4.16 Αρχική οθόνη εφαρμογής

Το βασικό μενού της εφαρμογής εμφανίζεται εφόσον ο χρήστης σύρει το άγγιγμά του μέσα στην οθόνη της συσκευής από το αριστερό μέρος της. Το μενού έχει τη διάταξη λίστας και παρουσιάζεται στην εικόνα 4.17.



Εικόνα 4.17 Μενού εφαρμογής

4.2.3 Δημιουργία ραντεβού

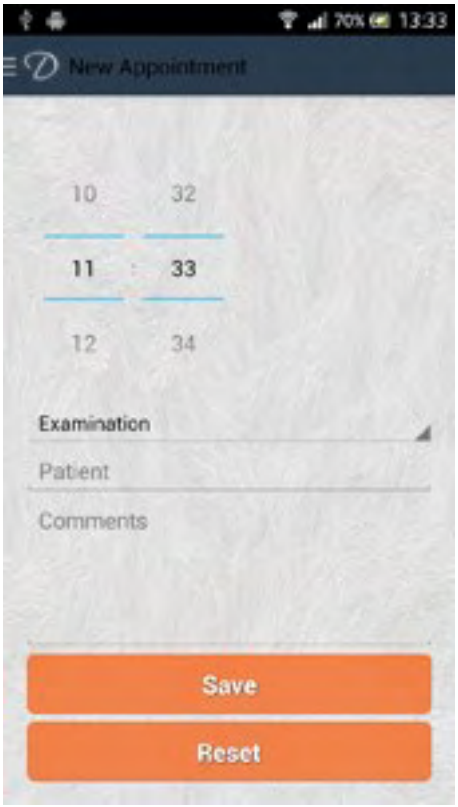
Ο χρήστης μπορεί να πλοηγηθεί στην οθόνη δημιουργίας νέου ραντεβού τόσο από την αρχική οθόνη όσο και από το μενού της εφαρμογής. Η συγκεκριμένη οθόνη εμφανίζεται στην εικόνα 4.18.

Για τη δημιουργία του ραντεβού συμπληρώνονται τα στοιχεία της φόρμας ως ακολούθως:

- **Title:** Ο τίτλος του νέου ραντεβού
- **Start date and time:** Η ημερομηνία και ώρα έναρξης του νέου ραντεβού
- **End date and time:** Η ημερομηνία και ώρα λήξης του νέου ραντεβού
- **Category:** Η κατηγορία του ραντεβού. Εδώ παρέχεται μία λίστα από την οποία ο χρήστης μπορεί να επιλέξει. Δεν υπάρχει η δυνατότητα εισαγωγής μίας νέας κατηγορίας σε αυτή την οθόνη.
- **Patient:** Ο ασθενής με τον οποίο θα πραγματοποιηθεί το ραντεβού. Ο χρήστης αρχίζει να πληκτρολογεί και το πεδίο «προτείνει» πιθανούς ασθενείς

με βάση τους ήδη καταχωρημένους ασθενείς. Ο χρήστης καλείται να επιλέξει έναν από αυτούς.

- **Comments:** Τα σχόλια που ο χρήστης επιθυμεί να συνοδεύουν το ραντεβού. Το πεδίο αυτό είναι ελεύθερου κειμένου.

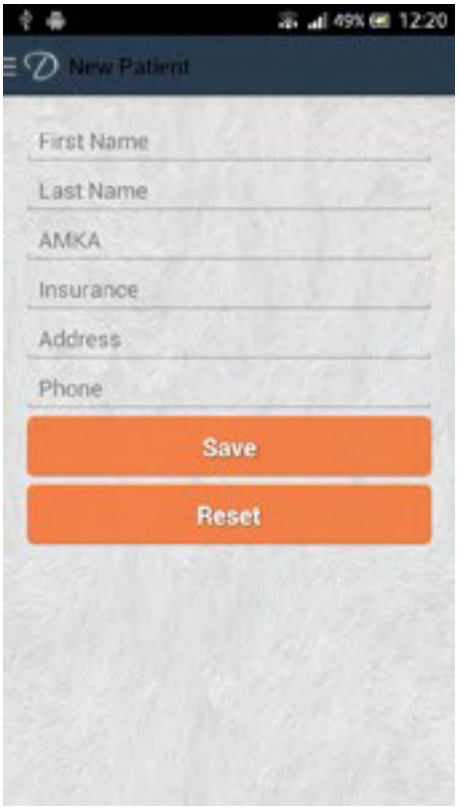


Εικόνα 4.18 Οθόνη δημιουργίας νέου ραντεβού

Έπειτα, εφόσον η φόρμα συμπληρωθεί σύμφωνα με τις επιθυμίες του χρήστη, με χρήση του κουμπιού «Save» το νέο ραντεβού δημιουργείται. Το κουμπί «Reset» μπορεί να χρησιμοποιηθεί για να επαναφερθεί η φόρμα στην αρχική της κατάσταση. Σε περίπτωση λανθασμένης συμπλήρωσης της φόρμας, η εφαρμογή ενημερώνει το χρήστη για τα λάθη του.

4.2.4 Δημιουργία Ασθενούς

Ο χρήστης μπορεί να πλοηγηθεί στην οθόνη δημιουργίας νέου ασθενούς τόσο από την αρχική οθόνη όσο και από το μενού της εφαρμογής. Η συγκεκριμένη οθόνη εμφανίζεται στην εικόνα 4.19.



Εικόνα 4.19 Οθόνη δημιουργίας νέου ασθενούς

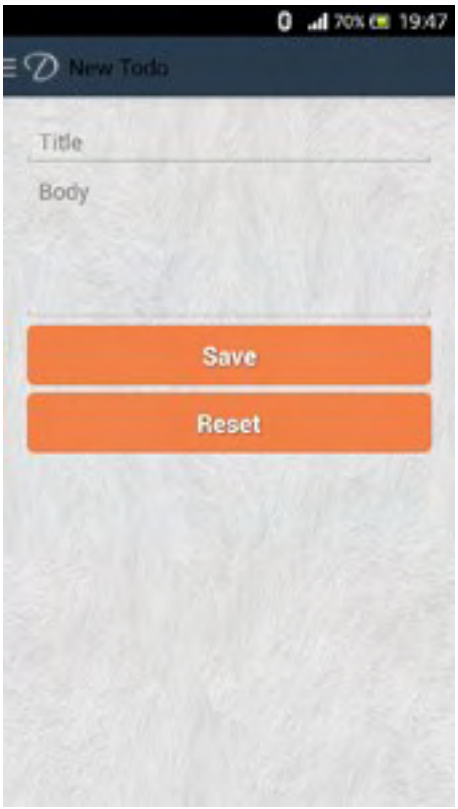
Για τη δημιουργία νέου ασθενούς συμπληρώνονται τα στοιχεία της φόρμας ως ακολούθως:

- **First Name:** Το όνομα του νέου ασθενούς.
- **Last Name:** Το επώνυμο του νέου ασθενούς.
- **AMKA:** Το ΑΜΚΑ του νέου ασθενούς.
- **Insurance:** Η ασφάλιση του νέου ασθενούς.
- **Address:** Η διεύθυνση του νέου ασθενούς.
- **Phone:** Το τηλέφωνο του νέου ασθενούς.

Έπειτα από τη συμπλήρωση της φόρμας, με χρήση του κουμπιού «Save» ο νέος ασθενής δημιουργείται. Το κουμπί «Reset» μπορεί να χρησιμοποιηθεί για να επαναφερθεί η φόρμα στην αρχική της κατάσταση. Σε περίπτωση λανθασμένης συμπλήρωσης της φόρμας, η εφαρμογή ενημερώνει το χρήστη για τα λάθη του.

4.2.5 Δημιουργία σημείωσης

Ο χρήστης μπορεί να πλοηγηθεί στην οθόνη δημιουργίας νέου ασθενούς τόσο από την αρχική οθόνη όσο και από το μενού της εφαρμογής. Η συγκεκριμένη οθόνη εμφανίζεται στην εικόνα 4.20.



Εικόνα 4.20 Οθόνη δημιουργίας νέας σημείωσης

Για τη δημιουργία της σημείωσης συμπληρώνονται τα στοιχεία της φόρμας ως ακολούθως:

- **Title:** Ο τίτλος της νέας σημείωσης.
- **Body:** Το κείμενο της νέας σημείωσης.

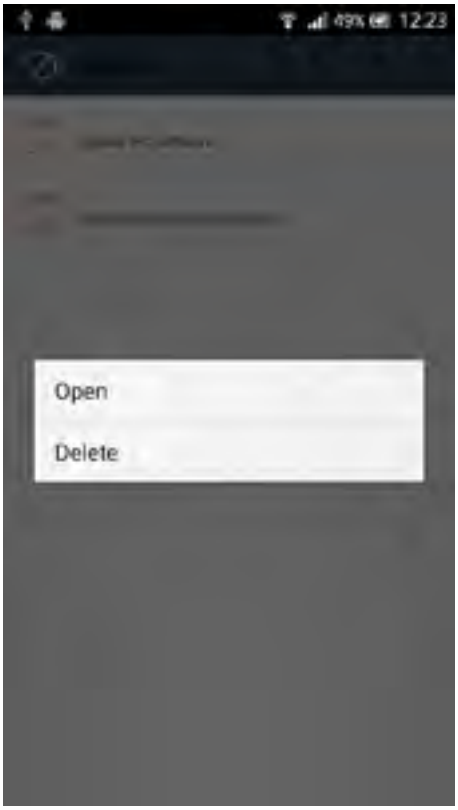
Έπειτα από τη συμπλήρωση της φόρμας, με χρήση του κουμπιού «Save» η νέα σημείωση δημιουργείται. Το κουμπί «Reset» μπορεί να χρησιμοποιηθεί για να επαναφερθεί η φόρμα στην αρχική της κατάσταση. Σε περίπτωση λανθασμένης συμπλήρωσης της φόρμας, η εφαρμογή ενημερώνει το χρήστη για τα λάθη του.

4.2.6 Τα ραντεβού μου

Η οθόνη που παρουσιάζει συγκεντρωτικά τα ραντεβού του χρήστη είναι προσβάσιμη τόσο από την αρχική οθόνη της εφαρμογής όσο και από το μενού. Η οθόνη των ραντεβού εμφανίζεται στην εικόνα 4.21.

Εικόνα 4.21 Οθόνη δημιουργίας νέας σημείωσης

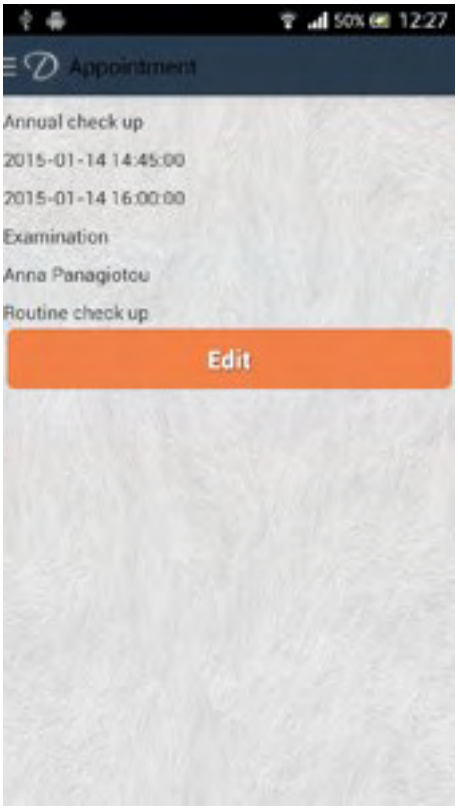
Τα ραντεβού παρουσιάζονται σε διάταξη λίστας στην οποία δίνεται ένα βασικό μέρος της πληροφορίας του ραντεβού (τίτλο, ημερομηνία/ώρα έναρξης και ημερομηνία/ώρα λήξης). Παρατεταμένο κράτημα (long tap) σε κάποιο ραντεβού θα εμφανίσει ένα μενού επιλογών για το συγκεκριμένο ραντεβού. Η οθόνη που παρουσιάζει αναλυτικά αυτό το μενού φαίνεται στην εικόνα 4.22.



Εικόνα 4.22 Οθόνη μενού έπειτα από παρατεταμένο κράτημα

Ο χρήστης μπορεί να οδηγηθεί στην αναλυτική σελίδα ενός ραντεβού με ένα «χτύπημα» (tap) πάνω στο ραντεβού που επιθυμεί. Η οθόνη που παρουσιάζει αναλυτικά ένα ραντεβού παρουσιάζεται στην εικόνα 4.23.

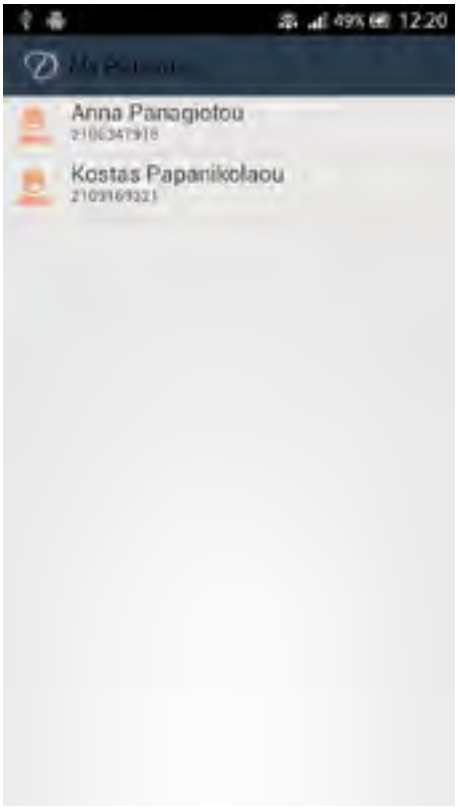
Με χρήση του κουμπιού «Edit» δίνεται η δυνατότητα επεξεργασίας του ραντεβού.



Εικόνα 4.23 Οθόνη αναλυτικής παρουσίασης ενός ραντεβού

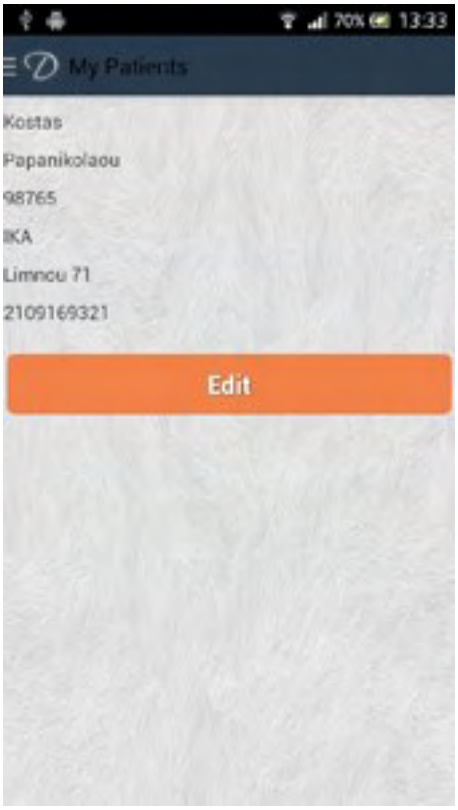
4.2.7 Οι ασθενείς μου

Η οθόνη που παρουσιάζει τους ασθενείς του χρήστη μπορεί να προσπελαστεί τόσο από την αρχική οθόνη της εφαρμογής όσο και από το μενού. Η συγκεκριμένη οθόνη παρουσιάζεται στην εικόνα 4.24.



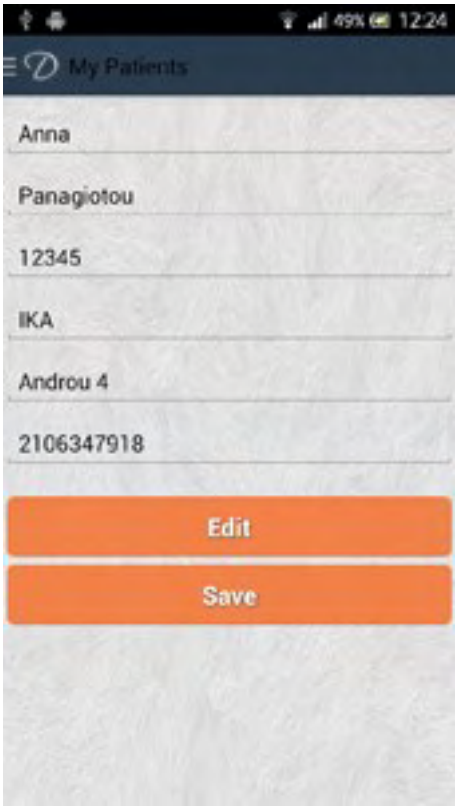
Εικόνα 4.24 Οθόνη παρουσίασης των ασθενών του χρήστη

Οι ασθενείς παρουσιάζονται σε διάταξη λίστας στην οποία δίνεται ένα βασικό μέρος της πληροφορίας του ασθενούς (όνομα και τηλέφωνο). Παρατεταμένο κράτημα (long tap) σε κάποιο ασθενή θα εμφανίσει ένα μενού επιλογών για το συγκεκριμένο ασθενή. Ο χρήστης μπορεί να οδηγηθεί στην αναλυτική σελίδα ενός ασθενούς με ένα «χτύπημα» πάνω στον ασθενή που επιθυμεί. Η οθόνη που παρουσιάζει αναλυτικά έναν ασθενή φαίνεται στην εικόνα 4.25.



Εικόνα 4.25 Οθόνη αναλυτικής παρουσίασης ενός ασθενούς

Με χρήση του κουμπιού «Edit» δίνεται η δυνατότητα επεξεργασίας του ασθενούς. Η οθόνη επεξεργασίας ενός ασθενούς φαίνεται στην εικόνα 4.26.



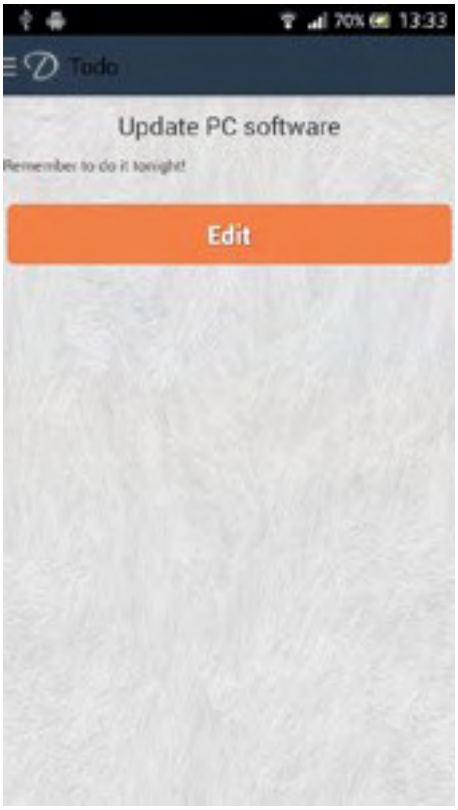
Εικόνα 4.26 Οθόνη επεξεργασίας ασθενούς

4.2.8 Οι σημειώσεις μου

Η οθόνη που παρουσιάζει συγκεντρωτικά τις σημειώσεις του χρήστη είναι προσβάσιμη τόσο από την αρχική οθόνη της εφαρμογής όσο και από το μενού. Η οθόνη των σημειώσεων εμφανίζεται στην εικόνα 4.27.

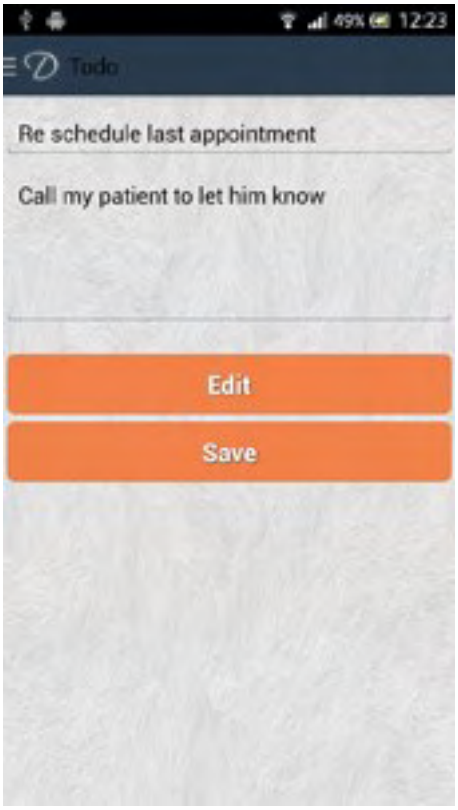
Εικόνα 4.27 Οθόνη παρουσίασης των σημειώσεων του χρήστη

Οι σημειώσεις παρουσιάζονται σε διάταξη λίστας στην οποία δίνεται ένα βασικό μέρος της πληροφορίας του ασθενούς (τίτλος). Παρατεταμένο κράτημα (long tap) σε κάποια σημείωση θα εμφανίσει ένα μενού επιλογών για τη συγκεκριμένη σημείωση. Ο χρήστης μπορεί να οδηγηθεί στην αναλυτική σελίδα μιας σημείωσης με ένα «χτύπημα» πάνω στη σημείωση που επιθυμεί. Η οθόνη που παρουσιάζει αναλυτικά μία σημείωση φαίνεται στην εικόνα 4.28.



Εικόνα 4.28 Οθόνη αναλυτικής παρουσίασης μίας σημείωσης

Με χρήση του κουμπιού «Edit» δίνεται η δυνατότητα επεξεργασίας της σημείωσης. Η οθόνη επεξεργασίας μίας σημείωσης φαίνεται στην εικόνα 4.29.



Εικόνα 4.29 επεξεργασίας σημείωσης

4.2.9 Σύνολο ραντεβού ανά μήνα

Η οθόνη που παρουσιάζει τη φόρμα υπολογισμού των συνολικών ραντεβού του χρήστη ανά μήνα είναι προσπελάσιμη από το μενού της εφαρμογής. Η συγκεκριμένη οθόνη εμφανίζεται στην εικόνα 4.30.

Εικόνα 4.30 Οθόνη υπολογισμού των συνολικών ραντεβού του χρήστη ανά μήνα

Ο χρήστης επιλέγει τόσο το μήνα όσο το έτος που τον ενδιαφέρει και με χρήση του κουμπιού «count» εμφανίζεται το αποτέλεσμα στο κάτω μέρος της οθόνης.

5 Συμπεράσματα

5.1 Κάλυψη στόχων ιστοσελίδας

Ο αρχικός στόχος όσον αφορά την ανάπτυξη της ιστοσελίδας ήταν η χρήση συγκεκριμένων τεχνολογιών για τη δημιουργία μίας πλήρως λειτουργικής, σύγχρονης και ιστοσελίδας, η οποία ανταποκρινεται στην οθόνη όπου προβάλλεται (responsive).

Ως βάση της ιστοσελίδας αποφσίστηκε να χρησιμοποιηθεί το Drupal 7.0 CMS, το οποίο και πραγματοποιήθηκε ως στόχος. Ο βασικός κορμός που προσφέρει το συγκεκριμένο Σύστημα Διαχείρισης Περιεχομένου αποτέλεσε τον πυρήνα της ιστοσελίδας, ενώ έγινε και εκτενής χρήση επιπλέον ενοτήτων (modules) για την επίτευξη πιό πλούσιας λειτουργικότητας. Ο συνδυασμός αυτός προσφέρει στον προγραμματιστή την πλήρη εξοικήωση και γνώση του συστήματος τόσο σε επίπεδο χρήστη, διαχειριστή όσο και προγραμματιστή της. Η βάση δεδομένων που παρέχεται από το Drupal και κατ' επέκταση χρησιμοποιήθηκε είναι MySQL.

Η εμφάνιση της ιστοσελίδας βασίστηκε στην ανάπτυξη ενός ειδικού Drupal θέματος (custom Drupal theme) με βάση την PHP και κυρίως CSS 3.0. Η σωστή χρήση των παραπάνω τεχνολογιών (ιδιαιτέρως των CSS 3.0) είχε ως αποτέλεσμα η ιστοσελίδα να εμφανίζεται και να «συμπεριφέρεται» σωστά σε οθόνες ποικίλων μεγεθών και φυλλομετρητές (browsers) διαφορετικών τεχνολογιών. Ενδεικτικά, αξίζει να αναφερθεί πως η εμφάνιση παραμένει ανεπηρέαστη στους φυλλομετρητές Chrome, Mozilla Firefox, Opera, Internet Explorer και Safari.

Το παραπάνω πρόβλημα αποτελεί βασικό εμπόδιο για όλους τους προγραμματιστές ιστοσελίδων και εφαρμογών διαδυκτίκου (web applications) και είναι απαραίτητη η επίλυση του σε κάθε έργο.

5.2 Παροχή Web Service

Ως επέκταση της προηγούμενης εργασίας, η οποία πέτυχε το στόχο της ανάπτυξης μίας ιστοσελίδας με χρήση του Drupal 7.0 CMS και σύμφωνη με τις προδιαγραφές που υπήρχαν, υλοποιήθηκε και η παροχή (exposure) ενός web service.

Η παραπάνω ενέργεια, η οποία αποσκοπεί στην παροχή των δεδομένων της ιστοσελίδας σε τρίτες εφαρμογές, επιτεύχθηκε με την ανάπτυξη μίας επιπλέον ειδικής ενότητας (custom module) για το Drupal. Για την υλοποίηση αυτής της ενότητας απαιτήθηκε η σε βάθος κατανόηση και γνώση της λειτουργίας του συτήματος, σε επίπεδο προγραμματιστή πλεον. Η υλοποίηση έγινε με τη χρήση της γλώσσας PHP ενώ την ανάπτυξη της ενότητας ακολούθησε η ενσωμάτωσή της στον πυρήνα της ιστοσελίδας με χρήση γνώσεων διαχείρισης του συστήματος.

Η εργασία αυτή (ανάπτυξη ειδικής ενότητας – custom module) ολοκληρώνει τη γνώση και την κατανόηση του συστήματος καθώς απαιτείται η μέγιστη εκξοικήωση με το Drupal για την επιτυχία αυτού του εγχειρήματος.

Το αποτέλεσμα είναι η παροχή (exposure) ενός τυποποιημένου (standardized) web service, το οποίο παρέχει πληροφορίες της ιστοσελίδας σε συγκεκριμένες τρίτες εφαρμογές. Τα δεδομένα παρέχοντε σε μορφή JSON διευκολύνοντας την επεξεργασία τους από την τρίτη εφαρμογή

5.3 Εφαρμογή κινητών τηλεφώνων και Tablet PCs

Το τρίτο κύριο κομμάτι της παρούσας πτυχιακής εργασίας ήταν η ανάπτυξη μίας εφαρμογής κινητών τηλεφώνων και Tablet PCs για συσκευές που χρησιμοποιούν λειτουργικό σύστημα Android.

Για την υλοποίηση του παραπάνω στόχου χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java καθώς και το εργαλείο Eclipse IDE. Ιδιαίτερη έμφαση δόθηκε στη διεπαφή χρήστη (User Interface - UI) της εφαρμογής καθώς αυτό επιβάλεται για όλες τις εφαρμογές για λειτουργικό σύστημα Android. Το παραπάνω προκύπτει διότι το συγκεκριμένο λειτουργικό σύστημα είναι επιλεγμένο από πάρα πολλούς κατασκευαστές συσκευών, με αποτέλεσμα το πλήθος των συσκευών – και κατ' επέκταση των οθονών στις οποίες θα κληθεί να εμφανιστεί η εφαρμογή – είναι πρακτικά άπειρο.

Ιδιαίτερη σημασία δόθηκε λοιπόν στις πρακτικές και τις συμβουλές που παρέχει η ίδια η Google (στην οποία ανήκει το λειτουργικό σύστημα Android) μέσω της

επίσημης ιστοσελίδας των προγραμματιστών του Android. Ως αποτέλεσμα, όλη η υλοποίηση έγινε με βάση αυτές τις προδιαγραφές.

Η βάση δεδομένων που χρησιμοποιεί το Android είναι SQLite και αυτή χρησιμοποιήθηκε για τις ανάγκες της εφαρμογής.

Τα παραπάνω εμπόδια και η αποφυγή τους έφερε μία ώριμη γνώση πάνω σε πρακτικά θέματα που καλείται να αντιμετωπίσει ένας προγραμματιστής εφαρμογών για Android. Παράλληλα η γνώση της επίλυσης τέτοιων προβλημάτων έφερε εμπειρία σε πραγματικό περιβάλλον ανάπτυξης λογισμικού (hands on experience).

5.4 Ανοιχτά θέματα – Βελτιώσεις - Επεκτάσεις

Αυτή η ενότητα είναι αφιερωμένη στην ανάλυση των ανοιχτών θεμάτων της εργασίας και τις πιθανές βελτιώσεις και επεκτάσεις που μπορούν να υλοποιηθούν στο μέλλον.

5.4.1 Εισαγωγή πολλαπλών ασθενών σε ένα ραντεβού

Η παραδοτέα έκδοση τόσο της ιστοσελίδας όσο και της εφαρμογής κινητών τηλεφώνων υποστηρίζει την εισαγωγή ενός μόνο ασθενούς σε κάθε ραντεβού. Είναι κατανοητό πως η πλειοψηφία των ιατρικών ασθενών πραγματοποιείται ανάμεσα στο γιατρό και σε ένα και μόνο ασθενή. Παρόλα αυτά, δεν υπάρχει κάποιος κανόνας ο οποίος να απαγορεύει στο γιατρό, για οποιονδοίποτε λόγο να εξετάσει δύο η και περισσότερους ασθενείς στο ίδιο ραντεβού.

Η δυνατότητα εισαγωγής πολλαπλών ασθενών στο ίδιο ραντεβού πρόκειτα για ένα επιθυμητό χαρακτηριστικό και η υλοποίησή της θα παρουσίαζε και ένα σύνολο ενδιαφέροντων προβλημάτων που θα έπρεπε να λυθούν.

Επίσης, η πρόσθεση ασθενών σε ένα ραντεβού δε θα είναι σε καμία περίπτωση απαραίτητη, προσφέροντας ένα επιπλέον κομμάτι λειτουργικότητας, το οποίο μπορεί να μην αξιποίησει ο γιατρός σε περίπτωση που δεν το επιθυμεί. Η ανάπτυξη αυτής της λειτουργικότητας όμως δε θα εμποδίζει τη χρήση της εφαρμογής ως έχει (με ένα ασθενή σε κάθε ραντεβού).

5.4.2 Συχγρονισμός δεδομένων μεταξύ ιστοσελίδας και εφαρμογής

Στην παρούσα εργασία, στο πλαίσιο της ιστοσελίδας αναπτύχθηκε και ένα web service, υλοποιώντας μία ειδική ενότητα (custom module). Με τον τρόπο αυτό, η ιστοσελίδα παρέχει (exposes) ένα RESTful web service καθιστώντας δυνατή την επικοινωνία της ιστοσελίδας με άλλες εφαρμογές, παρέχοντας κλήσεις με σκοπό την ανάκτηση και τη δημιουργία δεδομένων στην ιστοσελίδα.

Η προοπτική του συγχρονισμού δεδομένων μεταξύ της ιστοσελίδας και της εφαρμογής είναι πλέον ανοιχτή. Ως συνέπεια, μία ιδιαιτέρως ενδιαφέρουσα βελτίωση της εργασίας θα ήταν η υλοποίηση του συγρονισμού, κάνοντας χρήση των κλήσεων του web service.

5.4.3 Εισαγωγή μενού στις οθόνες λιστών

Λόγω της υλοποίησης του μενού σε μία κλάσση «πατέρα» από την οποία κληρονομούν όλα τα activities, ένα σύνολο προβλημάτων και περιορισμών εισήχθησαν στην υλοποίηση της εφαρμογής. Τα περισσότερα ξεπεράστηκαν κατά την υλοποίηση.

Ένα βασικό πρόβλημα που παρουσιάστηκε ήταν η σχεδίαση (layout) των activities καθώς όλα τα activities έπρεπε να κληρονομούν από την κλάσση «πατέρα» με συνέπεια η σχεδίαση να πρέπει να ακολουθεί πολύ συγκεκριμένες γραμμές. Το πρόβλημα ξεπεράστηκε με τη χρήση fragments στο σχεδιασμό των activities.

Τα activites που παρουσιάζουν τα δεδομένα με τη μορφή λίστας ομως, είναι υποχρεωμένα να κληρονομούν από την κλάσση ListActivity, η οποία με τη σειρά της κληρονομεί από την κλάσση Activity. Στο κομμάτι αυτό παρατηρήθηκε μία σοβαρή ασυμφωνία (incompatibility) καθώς μία κλάσση που κληρονομεί από το ListActivity δεν μπορεί ταυτόχρονα να κληρονομεί και από την κλάσση «πατέρα» που σχεδιάστηκε για όλα τα activities. Επιπλέον, καθώς το ListActivity έχει σχεδιαστεί ώστε να έχει ένα μόνο view (ένα listview), ήταν αδύνατο να χρησιμοποιηθούν fragments.

Ως συνέπεια τα 3 activities που παρουσιάζουν δεδομένα με τη μορφή λίστας δε διαθέτουν το πλάγιο μενού. Μία από τις χρησιμότερες επεκτάσεις που αναγνωρίζονται για την εφαρμογή θα ήταν η τροποποίηση αυτών των activities έτσι ώστε να υποστηρίζουν πλέον το μενού.

6 Πηγές

(W3c), W. W. W., 2014. *World Wide Web (W3c).* [Ηλεκτρονικό] Available at: http://www.w3.org/TR/soap/ [Πρόσβαση 30 10 2014].

C. Pautasso, O. Z. a. F. L., 2008. *RESTful Web services vs. "big" Web services: making the right architectural decision.* ACM, WWW '08: Proceeding of the 17th international conference on World Wide Web.

Curbera, F., 2002. Unraveling the Web Services: An Introduction. *IEEE Internet Computing,* 6(2), pp. 86-93.

Drupal, 2014. *Επίσημη ιστοσελίδα Drupal.* [Ηλεκτρονικό] Available at: https://www.drupal.org/documentation/understand [Πρόσβαση 08 11 2014].

Drupal, O., 2014. *Επίσημη ιστοσελίδα Drupal - Omega module.* [Ηλεκτρονικό] Available at: https://www.drupal.org/project/omega [Πρόσβαση 06 12 2014].

Fahad Aijaz, Syed Zahid Ali, Muzzamil Aziz Chaudhary, Bernhard Walke, 2009. *Enabling High Performance Mobile Web Services Provisioning,* Department of Communication Networks Faculty 6, RWTH Aachen University: Vehicular Technology Conference Fall.

Fielding, D. R. T., 2000. *Architectural Styles and the Design of Network-based Software Architectures,* University of Carolina, Irvine: PhD Thesis.

Google, 2014. *Επίσημη ιστοσελίδα των προγραμματιστών του Android.* [Ηλεκτρονικό] Available at: http://developer.android.com/guide/components/activities.html [Πρόσβαση 20 10 2104].

Google, 2014. *Επίσημη ιστοσελίδα των προγραμματιστών του Android.* [Ηλεκτρονικό] Available at: http://developer.android.com/training/basics/activity-lifecycle/index.html [Πρόσβαση 20 10 2014].

Google, 2014. *Επίσημη ιστοσελίδα των προγραμματιστών του Android.* [Ηλεκτρονικό] Available at: http://developer.android.com/training/basics/activity-lifecycle/starting.html [Πρόσβαση 20 10 2014].

Google, 2014. *Επίσημη ιστοσελίδα των προγραμματιστών του Android.* [Ηλεκτρονικό] Available at: http://developer.android.com/training/basics/activity-

lifecycle/pausing.html
[Πρόσβαση 21 10 2014].

Google, 2014. *Επίσημη ιστοσελίδα των προγραμματιστών του Android.* [Ηλεκτρονικό] Available at: http://developer.android.com/training/basics/activity-lifecycle/stopping.html
[Πρόσβαση 21 10 2014].

Google, 2014. *Επίσημη ιστοσελίδα των προγραμματιστών του Android.* [Ηλεκτρονικό] Available at: http://developer.android.com/training/basics/activity-lifecycle/recreating.html
[Πρόσβαση 21 10 2014].

Google, 2014. *Επίσημη ιστοσελίδα των προγραμματιστών του Android.* [Ηλεκτρονικό] Available at: http://developer.android.com/guide/components/intents-filters.html
[Πρόσβαση 23 10 2014].

Google, 2014. *Επίσημη ιστοσελίδα των προγραμματιστών του Android.* [Ηλεκτρονικό] Available at: http://developer.android.com/guide/topics/data/data-storage.html
[Πρόσβαση 25 10 2014].

Google, 2014. *Επίσημη ιστοσελίδα των προγραμματιστών του Android.* [Ηλεκτρονικό] Available at: http://developer.android.com/guide/practices/screens_support.html
[Πρόσβαση 26 10 2014].

Hatem Hamad, Motaz Saad, and Ramzi Abed, 2010. *Performance Evaluation of RESTful Web Services for Mobile Devices*, Computer Engineering Department, Islamic University of Gaza, Palestine: International Arab Journal of e-Technology.

module, D. -. S., 2014. *Επίσημη ιστοσελίδα Drupal - Services module.* [Ηλεκτρονικό] Available at: https://www.drupal.org/project/services
[Πρόσβαση 06 12 2014].

Newmarch, J., 2005. *A RESTful Approach: Clean UPnP without SOAP, Communications and Networking Conference (CCNC): Proceedings of the 2nd IEEE Consumer,* s.l.: s.n.

SQLite, 2014. *Επίσημη ιστοσελίδα της SQLite.* [Ηλεκτρονικό] Available at: http://www.sqlite.org/about.html
[Πρόσβαση 20 10 2014]

Παράρτημα: Κώδικας

AppointmentActivity

```
package daybook.activities;

import java.util.ArrayList;

import daybook.db.DataSource;

import daybook.fragments.AppointmentFragment;

import daybook.main.R;

import daybook.util.Appointment;

import daybook.util.Category;

import daybook.util.Contact;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.ArrayAdapter;

import android.support.v4.app.NavUtils;


public class AppointmentActivity extends BaseActivity {


  private DataSource datasource;


  private AppointmentFragment fragment;


  private Appointment appointment;
```

```
  private int id;


  private ArrayList<String> categoryChoices;
  private ArrayList<String> contactChoices;
  private ArrayList<Contact> contactInfoChoices;


  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (savedInstanceState == null) {
      fragment = new AppointmentFragment();
     getSupportFragmentManager().beginTransaction().replace(R.id.content_frame,
fragment).commit();
    }
    setupActionBar();


    datasource = new DataSource(this);
    datasource.open();


    Bundle extras = getIntent().getExtras();
    id = Integer.parseInt(extras.getString("id"));
    appointment = datasource.getAppointmentById(id);
    fragment.setCategory(datasource.getCategoryById(appointment.getCategory()));
    fragment.setContact(datasource.getContactById(appointment.getContact()));
    fragment.setAppointment(appointment);
```

```
  populateContacts();
  populateCategories();
}

@Override
protected void onResume() {
  super.onResume();
  datasource.open();
}

@Override
protected void onPause() {
  super.onPause();
  datasource.close();
}

private void setupActionBar() {
  getActionBar().setDisplayHomeAsUpEnabled(true);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.appointment, menu);
  return true;
}
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home:
      NavUtils.navigateUpFromSameTask(this);
      return true;
  }
  return super.onOptionsItemSelected(item);
}

private void populateCategories() {
  ArrayList<Category> categories = datasource.getAllCategories();
  categoryChoices = new ArrayList<String>();

  for (Category c : categories)
    categoryChoices.add(c.getTitle());

  fragment.setCategoryAdapter(new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, categoryChoices));
}

private void populateContacts() {
  ArrayList<Contact> contacts = datasource.getAllContacts();
  contactChoices = new ArrayList<String>();
  contactInfoChoices = new ArrayList<Contact>();
```

```
    for (Contact c : contacts) {
      contactChoices.add(c.getFirstName() + " " + c.getLastName());
      contactInfoChoices.add(c);
    }

    fragment.setContactAdapter(new ArrayAdapter<String>(this,
android.R.layout.simple_dropdown_item_1line, contactChoices));
    fragment.setContactIdAdapter(contactInfoChoices);
  }

  public void toggleEdit(View button) {
    fragment.toggleEdit();
  }

  public void saveAppointment(View button) {
    datasource.updateAppointment(new Appointment(appointment.getId(),
fragment.getTitle(), fragment.getStartDate(), fragment.getEndDate(),
fragment.getSelectedCategory(), fragment
        .getSelectedContact(), fragment.getComments()));
    finish();
  }

}
```

BaseActivity

```
package daybook.activities;

import daybook.drawer.NavDrawerActivityConfiguration;

import daybook.drawer.NavDrawerAdapter;

import daybook.drawer.NavDrawerItem;

import daybook.drawer.NavMenuItem;

import daybook.drawer.NavMenuSection;

import daybook.main.R;

import android.content.Intent;

import android.content.res.Configuration;

import android.os.Bundle;

import android.support.v4.app.ActionBarDrawerToggle;

import android.support.v4.app.FragmentActivity;

import android.support.v4.view.GravityCompat;

import android.support.v4.widget.DrawerLayout;

import android.view.KeyEvent;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ListView;


public abstract class BaseActivity extends FragmentActivity {


        private DrawerLayout mDrawerLayout;
```

```
private ActionBarDrawerToggle mDrawerToggle;

private ListView mDrawerList;

private CharSequence mDrawerTitle;
private CharSequence mTitle;

private NavDrawerActivityConfiguration navConf;

protected NavDrawerActivityConfiguration getNavDrawerConfiguration() {

        NavDrawerItem[] menu = new NavDrawerItem[] {
                NavMenuSection.create(100, "My Daybook"),
                NavMenuItem.create(101, "My Appointments", "navdrawer_friends", false, this),
                NavMenuItem.create(102, "My Patients", "navdrawer_airport", true, this),
                NavMenuItem.create(103, "My Todos", "navdrawer_airport", true, this),
                NavMenuSection.create(200, "Add Content"),
                NavMenuItem.create(201, "New Appointment", "navdrawer_rating", false, this),
                NavMenuItem.create(202, "New Patient", "navdrawer_eula", false, this),
                NavMenuItem.create(203, "New Todo", "navdrawer_quit", false, this),
    NavMenuSection.create(300, "General"),
```

```
NavMenuItem.create(301, "Appointments counter", "navdrawer_rating", false, this) };


        NavDrawerActivityConfiguration navDrawerActivityConfiguration = new NavDrawerActivityConfiguration();
        navDrawerActivityConfiguration.setMainLayout(R.layout.main);

    navDrawerActivityConfiguration.setDrawerLayoutId(R.id.drawer_layout);
        navDrawerActivityConfiguration.setLeftDrawerId(R.id.left_drawer);
        navDrawerActivityConfiguration.setNavItems(menu);

    navDrawerActivityConfiguration.setDrawerShadow(R.drawable.drawer_shadow);

    navDrawerActivityConfiguration.setDrawerOpenDesc(R.string.drawer_open);

    navDrawerActivityConfiguration.setDrawerCloseDesc(R.string.drawer_close);
        navDrawerActivityConfiguration.setBaseAdapter(new NavDrawerAdapter(this, R.layout.navdrawer_item, menu));
        return navDrawerActivityConfiguration;
    }

    protected void onNavItemSelected(int id) {
        Intent intent;
        switch ((int) id) {
        case 101:
            intent = new Intent(getApplicationContext(), MyAppointmentsActivity.class);
```

```
                startActivity(intent);
                break;
            case 102:
                intent = new Intent(getApplicationContext(),
MyPatientsActivity.class);
                startActivity(intent);
                break;
            case 103:
                intent = new Intent(getApplicationContext(),
MyTodoActivity.class);
                startActivity(intent);
                break;
            case 201:
                intent = new Intent(getApplicationContext(),
NewAppointmentActivity.class);
                startActivity(intent);
                break;
            case 202:
                intent = new Intent(getApplicationContext(),
NewPatientActivity.class);
                startActivity(intent);
                break;
            case 203:
                intent = new Intent(getApplicationContext(),
NewTodoActivity.class);
                startActivity(intent);
                break;
```

```
case 301:
 intent = new Intent(getApplicationContext(), CounterActivity.class);
 startActivity(intent);
 break;
            }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            navConf = getNavDrawerConfiguration();

            setContentView(navConf.getMainLayout());

            mTitle = mDrawerTitle = getTitle();

            mDrawerLayout = (DrawerLayout) findViewById(navConf.getDrawerLayoutId());
            mDrawerList = (ListView) findViewById(navConf.getLeftDrawerId());
            mDrawerList.setAdapter(navConf.getBaseAdapter());
            mDrawerList.setOnItemClickListener(new DrawerItemClickListener());

            this.initDrawerShadow();
```

```
        getActionBar().setDisplayHomeAsUpEnabled(true);
        getActionBar().setHomeButtonEnabled(true);

        mDrawerToggle = new ActionBarDrawerToggle(this, mDrawerLayout,
                getDrawerIcon(), navConf.getDrawerOpenDesc(),
                navConf.getDrawerCloseDesc()) {
            public void onDrawerClosed(View view) {
                getActionBar().setTitle(mTitle);
                invalidateOptionsMenu();
            }

            public void onDrawerOpened(View drawerView) {
                getActionBar().setTitle(mDrawerTitle);
                invalidateOptionsMenu();
            }
        };
        mDrawerLayout.setDrawerListener(mDrawerToggle);
    }

    protected void initDrawerShadow() {
        mDrawerLayout.setDrawerShadow(navConf.getDrawerShadow(),
GravityCompat.START);
    }
```

```
        protected int getDrawerIcon() {
                return R.drawable.ic_drawer;
        }

        @Override
        protected void onPostCreate(Bundle savedInstanceState) {
                super.onPostCreate(savedInstanceState);
                mDrawerToggle.syncState();
        }

        @Override
        public void onConfigurationChanged(Configuration newConfig) {
                super.onConfigurationChanged(newConfig);
                mDrawerToggle.onConfigurationChanged(newConfig);
        }

        @Override
        public boolean onPrepareOptionsMenu(Menu menu) {
                if (navConf.getActionMenuItemsToHideWhenDrawerOpen() != null)
{
                        boolean drawerOpen =
mDrawerLayout.isDrawerOpen(mDrawerList);
                        for (int iItem :
navConf.getActionMenuItemsToHideWhenDrawerOpen()) {
                                menu.findItem(iItem).setVisible(!drawerOpen);
                        }
```

```
        }
        return super.onPrepareOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
        if (mDrawerToggle.onOptionsItemSelected(item)) {
                return true;
        } else {
                return false;
        }
}

@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_MENU) {
                if (this.mDrawerLayout.isDrawerOpen(this.mDrawerList)) {
                        this.mDrawerLayout.closeDrawer(this.mDrawerList);
                } else {
                        this.mDrawerLayout.openDrawer(this.mDrawerList);
                }
                return true;
        }
        return super.onKeyDown(keyCode, event);
}
```

```
protected DrawerLayout getDrawerLayout() {
        return mDrawerLayout;
}

protected ActionBarDrawerToggle getDrawerToggle() {
        return mDrawerToggle;
}

private class DrawerItemClickListener implements
                ListView.OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
                selectItem(position);
        }
}

public void selectItem(int position) {
        NavDrawerItem selectedItem = navConf.getNavItems()[position];

        this.onNavItemSelected(selectedItem.getId());
        mDrawerList.setItemChecked(position, true);

        if (selectedItem.updateActionBarTitle()) {
                setTitle(selectedItem.getLabel());
```

```
            }

            if (this.mDrawerLayout.isDrawerOpen(this.mDrawerList)) {
                mDrawerLayout.closeDrawer(mDrawerList);
            }
        }

        @Override
        public void setTitle(CharSequence title) {
            mTitle = title;
            getActionBar().setTitle(mTitle);
        }

}
```

CounterActivity

```
package daybook.activities;
import java.util.ArrayList;
import java.util.Arrays;
import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.NavUtils;
import android.view.Menu;
import android.view.MenuItem;
```

```
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;
import daybook.db.DataSource;
import daybook.main.R;

public class CounterActivity extends Activity {

  private DataSource datasource;

  private Spinner monthSpinner;
  private Spinner yearSpinner;

  private ArrayAdapter<String> monthAdapter;
  private ArrayAdapter<String> yearAdapter;

  private ArrayList<String> monthChoices;
  private ArrayList<String> yearChoices;

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

```
  populateComponents();
}

@Override
protected void onResume() {
  super.onResume();
  datasource = new DataSource(this);
  datasource.open();
}

@Override
protected void onPause() {
  super.onPause();
  datasource.close();
}

private void setupActionBar() {
  getActionBar().setDisplayHomeAsUpEnabled(true);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.count, menu);
  return true;
}
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home:
      NavUtils.navigateUpFromSameTask(this);
      return true;
  }
  return super.onOptionsItemSelected(item);
}

public void calculateCount(View button) {
  String m = Integer.toString(monthSpinner.getSelectedItemPosition() + 1);
  String y = yearSpinner.getSelectedItem().toString();

  if (m.length() < 2)
    m = "0" + m;

  Toast.makeText(getApplicationContext(), datasource.getAppointmentsCount(m, y)
+ " appointments", Toast.LENGTH_SHORT).show();
}

private void populateComponents() {
  monthSpinner = (Spinner) findViewById(R.id.CounterMonth);
  yearSpinner = (Spinner) findViewById(R.id.CounterYear);
```

```
    addMonths();
    addYears();

    monthAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, monthChoices);
    yearAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, yearChoices);

    monthSpinner.setAdapter(monthAdapter);
    yearSpinner.setAdapter(yearAdapter);
  }

  private void addMonths() {
    monthChoices = new ArrayList<String>(Arrays.asList("January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"));
  }

  private void addYears() {
    yearChoices = new ArrayList<String>(Arrays.asList("2014", "2015"));
  }

}
```

DashboardActivity

```
package daybook.activities;


import daybook.fragments.DashboardFragment;

import daybook.main.R;

import android.os.Bundle;

import android.view.Menu;


public class DashboardActivity extends BaseActivity {


	@Override

	protected void onCreate(Bundle savedInstanceState) {

		super.onCreate(savedInstanceState);

		if (savedInstanceState == null) {


	getSupportFragmentManager().beginTransaction().replace(R.id.content_frame
, new DashboardFragment()).commit();

		}

	}


	@Override

	public boolean onCreateOptionsMenu(Menu menu) {

		getMenuInflater().inflate(R.menu.dashboard, menu);

		return true;

	}

}
```

LoginActivity

```
package daybook.activities;

import daybook.main.R;

import android.animation.Animator;

import android.animation.AnimatorListenerAdapter;

import android.annotation.TargetApi;

import android.app.Activity;

import android.content.Intent;

import android.os.AsyncTask;

import android.os.Build;

import android.os.Bundle;

import android.text.TextUtils;

import android.view.KeyEvent;

import android.view.Menu;

import android.view.View;

import android.view.inputmethod.EditorInfo;

import android.widget.EditText;

import android.widget.TextView;



public class LoginActivity extends Activity {

  private static final String[] CREDENTIALS = new String[] { "admin:demo" };

  public static final String EXTRA_USERNAME = "test";
```

```
  private UserLoginTask authTask = null;

  private String username;
  private String password;

  private EditText usernameView;
  private EditText passwordView;
  private View loginFormView;
  private View loginStatusView;
  private TextView loginStatusMessageView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_login);

    username = getIntent().getStringExtra(EXTRA_USERNAME);
    usernameView = (EditText) findViewById(R.id.email);
    usernameView.setText(username);

    passwordView = (EditText) findViewById(R.id.password);
    passwordView.setOnEditorActionListener(new
TextView.OnEditorActionListener() {
      @Override
```

```
    public boolean onEditorAction(TextView textView, int id, KeyEvent keyEvent) {
      if (id == R.id.login || id == EditorInfo.IME_NULL) {
        attemptLogin();
        return true;
      }
      return false;
    }
  });

  loginFormView = findViewById(R.id.login_form);
  loginStatusView = findViewById(R.id.login_status);
  loginStatusMessageView = (TextView) findViewById(R.id.login_status_message);

  findViewById(R.id.sign_in_button).setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View view) {
      attemptLogin();
    }
  });
 }

 @Override
 public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);
  getMenuInflater().inflate(R.menu.login, menu);
```

```
  return true;
}

public void attemptLogin() {
  if (authTask != null) {
    return;
  }

  usernameView.setError(null);
  passwordView.setError(null);

  username = usernameView.getText().toString();
  password = passwordView.getText().toString();

  boolean cancel = false;
  View focusView = null;

  if (TextUtils.isEmpty(password)) {
    passwordView.setError(getString(R.string.error_field_required));
    focusView = passwordView;
    cancel = true;
  } else if (password.length() < 4) {
    passwordView.setError(getString(R.string.error_invalid_password));
    focusView = passwordView;
    cancel = true;
```

```
    }

    if (TextUtils.isEmpty(username)) {
      usernameView.setError(getString(R.string.error_field_required));
      focusView = usernameView;
      cancel = true;
    }

    if (cancel) {
      focusView.requestFocus();
    } else {
      loginStatusMessageView.setText(R.string.login_progress_signing_in);
      showProgress(true);
      authTask = new UserLoginTask();
      authTask.execute((Void) null);
    }
  }

  @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)
  private void showProgress(final boolean show) {
    if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.HONEYCOMB_MR2) {
      int shortAnimTime =
getResources().getInteger(android.R.integer.config_shortAnimTime);

      loginStatusView.setVisibility(View.VISIBLE);
```

```
    loginStatusView.animate().setDuration(shortAnimTime).alpha(show ? 1 :
0).setListener(new AnimatorListenerAdapter() {
      @Override
      public void onAnimationEnd(Animator animation) {
        loginStatusView.setVisibility(show ? View.VISIBLE : View.GONE);
      }
    });

    loginFormView.setVisibility(View.VISIBLE);
    loginFormView.animate().setDuration(shortAnimTime).alpha(show ? 0 :
1).setListener(new AnimatorListenerAdapter() {
      @Override
      public void onAnimationEnd(Animator animation) {
        loginFormView.setVisibility(show ? View.GONE : View.VISIBLE);
      }
    });
  } else {
    loginStatusView.setVisibility(show ? View.VISIBLE : View.GONE);
    loginFormView.setVisibility(show ? View.GONE : View.VISIBLE);
  }
 }

 public class UserLoginTask extends AsyncTask<Void, Void, Boolean> {
  @Override
  protected Boolean doInBackground(Void... params) {
    try {
```

```
    Thread.sleep(2000);
   } catch (InterruptedException e) {
    return false;
   }

   for (String credential : CREDENTIALS) {
    String[] pieces = credential.split(":");
    if (pieces[0].equals(username)) {
     return pieces[1].equals(password);
    }
   }
   return true;
}

@Override
protected void onPostExecute(final Boolean success) {
   authTask = null;
   showProgress(false);

   if (success) {
    Intent intent = new Intent(getApplicationContext(), DashboardActivity.class);
    startActivity(intent);
    finish();
   } else {
    passwordView.setError(getString(R.string.error_incorrect_password));
```

```
        passwordView.requestFocus();
      }
    }

    @Override
    protected void onCancelled() {
      authTask = null;
      showProgress(false);
    }
  }
}
```

MyAppointmentsActivity

```
package daybook.activities;

import daybook.db.DataSource;

import daybook.main.R;

import daybook.util.Appointment;

import ListAdapters.ListAdapterAppointments;

import android.os.Bundle;

import android.app.ListActivity;

import android.content.Intent;

import android.view.ContextMenu;

import android.view.Menu;

import android.view.MenuItem;
```

```
import android.view.View;
import android.view.ContextMenu.ContextMenuInfo;
import android.widget.AdapterView;
import android.widget.AdapterView.AdapterContextMenuInfo;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.TextView;
import android.support.v4.app.NavUtils;

public class MyAppointmentsActivity extends ListActivity {

  private DataSource datasource;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setupActionBar();

    datasource = new DataSource(this);
    datasource.open();

    loadList();
  }

  @Override
  protected void onResume() {
```

```
  super.onResume();
  datasource.open();
  setListAdapter(new ListAdapterAppointments(MyAppointmentsActivity.this,
datasource.getAllAppointments()));
 }

 @Override
 protected void onPause() {
  super.onPause();
  datasource.close();
 }

 private void setupActionBar() {
  getActionBar().setDisplayHomeAsUpEnabled(true);
 }

 @Override
 public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.list_appointments, menu);
  return true;
 }

 @Override
 public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home:
```

```
        NavUtils.navigateUpFromSameTask(this);
        return true;
    }
    return super.onOptionsItemSelected(item);
  }

  @Override
  public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    getMenuInflater().inflate(R.menu.myappointments_context_menu, menu);
  }

  @Override
  public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
    Appointment selectedAppointment = (Appointment) getListAdapter().getItem(info.position);
    switch (item.getItemId()) {
      case R.id.MyAppointmentsContextMenuOpen:
        openAppointmentActivity(String.valueOf(selectedAppointment.getId()));
        break;
      case R.id.MyAppointmentsContextMenuDelete:
        datasource.deleteAppointment(selectedAppointment.getId());
        setListAdapter(new ListAdapterAppointments(MyAppointmentsActivity.this, datasource.getAllAppointments()));
```

```
        break;
      default:
        break;
    }
    return true;
  }

  private void loadList() {
    setListAdapter(new ListAdapterAppointments(MyAppointmentsActivity.this, datasource.getAllAppointments()));
    getListView().setOnItemClickListener(new OnItemClickListener() {
      @Override
      public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        TextView rowID = (TextView) view.findViewById(R.id.list_row_appointment_hidden_id);
        openAppointmentActivity(rowID.getText().toString());
      }
    });
    registerForContextMenu(getListView());
  }

  private void openAppointmentActivity(String id) {
    Intent intent = new Intent(getApplicationContext(), AppointmentActivity.class);
    intent.putExtra("id", id);
    startActivity(intent);
```

```
  }

}
```

MyPatientsActivity

```
package daybook.activities;

import ListAdapters.ListAdapterContact;

import android.app.ListActivity;

import android.content.Intent;

import android.os.Bundle;

import android.support.v4.app.NavUtils;

import android.view.ContextMenu;

import android.view.ContextMenu.ContextMenuInfo;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.AdapterView;

import android.widget.AdapterView.AdapterContextMenuInfo;

import android.widget.AdapterView.OnItemClickListener;

import android.widget.TextView;

import daybook.db.DataSource;

import daybook.main.R;

import daybook.util.Contact;


public class MyPatientsActivity extends ListActivity {
```

```
private DataSource datasource;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  // Show the Up button in the action bar.
  setupActionBar();

  datasource = new DataSource(this);
  datasource.open();

  loadList();
}

@Override
protected void onResume() {
  super.onResume();
  datasource.open();
  setListAdapter(new ListAdapterContact(MyPatientsActivity.this,
datasource.getAllContacts()));
}

@Override
protected void onPause() {
  super.onPause();
```

```
  datasource.close();
 }

 private void setupActionBar() {
  getActionBar().setDisplayHomeAsUpEnabled(true);
 }

 @Override
 public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.list_contacts, menu);
  return true;
 }

 @Override
 public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home:
      NavUtils.navigateUpFromSameTask(this);
      return true;
  }
  return super.onOptionsItemSelected(item);
 }

 @Override
 public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo
menuInfo) {
```

```
  super.onCreateContextMenu(menu, v, menuInfo);
  getMenuInflater().inflate(R.menu.mycontacts_context_menu, menu);
 }

 @Override
 public boolean onContextItemSelected(MenuItem item) {
  AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
  Contact selectedContact = (Contact) getListAdapter().getItem(info.position);
  switch (item.getItemId()) {
   case R.id.MyContactsContextMenuOpen:
    openContactActivity(String.valueOf(selectedContact.getId()));
    break;
   case R.id.MyContactsContextMenuDelete:
    datasource.deleteContact(selectedContact.getId());
    setListAdapter(new ListAdapterContact(MyPatientsActivity.this,
datasource.getAllContacts()));
    break;
   default:
    break;
  }
  return true;
 }

 private void loadList() {
  setListAdapter(new ListAdapterContact(MyPatientsActivity.this,
datasource.getAllContacts()));
```

```
    getListView().setOnItemClickListener(new OnItemClickListener() {
      @Override
      public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        TextView rowID = (TextView) view.findViewById(R.id.list_row_contact_hidden_id);
        openContactActivity(rowID.getText().toString());
      }
    });
    registerForContextMenu(getListView());
  }

  private void openContactActivity(String id) {
    Intent intent = new Intent(getApplicationContext(), PatientActivity.class);
    intent.putExtra("id", id);
    intent.putExtra("id2", "test");
    startActivity(intent);
  }

}
```

MyTodoActivity

```
package daybook.activities;

import daybook.db.DataSource;
import daybook.main.R;
```

```
import daybook.util.Todo;
import ListAdapters.ListAdapterTodo;
import android.os.Bundle;
import android.app.ListActivity;
import android.content.Intent;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.AdapterContextMenuInfo;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.TextView;
import android.support.v4.app.NavUtils;

public class MyTodoActivity extends ListActivity {

  private DataSource datasource;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Show the Up button in the action bar.
    setupActionBar();
```

```
    datasource = new DataSource(this);
    datasource.open();

    loadList();
  }

  @Override
  protected void onResume() {
    super.onResume();
    datasource.open();
    setListAdapter(new ListAdapterTodo(MyTodoActivity.this,
datasource.getAllTodos()));
  }

  @Override
  protected void onPause() {
    super.onPause();
    datasource.close();
  }

  private void setupActionBar() {
    getActionBar().setDisplayHomeAsUpEnabled(true);
  }

  @Override
```

```
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.list_todo, menu);
  return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home:
      NavUtils.navigateUpFromSameTask(this);
      return true;
  }
  return super.onOptionsItemSelected(item);
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo
menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);
  getMenuInflater().inflate(R.menu.mytodos_context_menu, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
  AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
  Todo selectedTodo = (Todo) getListAdapter().getItem(info.position);
```

```
   switch (item.getItemId()) {
     case R.id.MyTodosContextMenuOpen:
       openTodoActivity(String.valueOf(selectedTodo.getId()));
       break;
     case R.id.MyTodosContextMenuDelete:
       datasource.deleteTodo(selectedTodo.getId());
       setListAdapter(new ListAdapterTodo(MyTodoActivity.this,
datasource.getAllTodos()));
       break;
     default:
       break;
   }
   return true;
  }


  private void loadList() {
   setListAdapter(new ListAdapterTodo(MyTodoActivity.this,
datasource.getAllTodos()));
   getListView().setOnItemClickListener(new OnItemClickListener() {
     @Override
     public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {
       TextView rowID = (TextView)
view.findViewById(R.id.list_row_todo_hidden_id);
       openTodoActivity(rowID.getText().toString());
     }
   });
```

```
    registerForContextMenu(getListView());
  }

  private void openTodoActivity(String id) {
    Intent intent = new Intent(getApplicationContext(), TodoActivity.class);
    intent.putExtra("id", id);
    startActivity(intent);
  }

}
```

NewAppointmentActivity

```
package daybook.activities;

import java.util.ArrayList;

import daybook.db.DataSource;

import daybook.fragments.NewAppointmentFragment;

import daybook.main.R;

import daybook.util.Appointment;

import daybook.util.Category;

import daybook.util.Contact;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.ArrayAdapter;
```

```
import android.widget.Toast;
import android.support.v4.app.NavUtils;

public class NewAppointmentActivity extends BaseActivity {

  static final int DATE_DIALOG_ID = 1;
  static final int TIME_DIALOG_ID = 2;

  private DataSource datasource;

  private NewAppointmentFragment fragment;

  private ArrayList<String> categoryChoices;
  private ArrayList<String> contactChoices;
  private ArrayList<Contact> contactInfoChoices;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (savedInstanceState == null) {
      fragment = new NewAppointmentFragment();
     getSupportFragmentManager().beginTransaction().replace(R.id.content_frame,
fragment).commit();
    }
    setupActionBar();
```

```
  datasource = new DataSource(this);
  datasource.open();

  populateContacts();
  populateCategories();
}

@Override
protected void onResume() {
  super.onResume();
  datasource.open();
}

@Override
protected void onPause() {
  super.onPause();
  datasource.close();
}

private void populateCategories() {
  ArrayList<Category> categories = datasource.getAllCategories();

  if (categories.size() == 0) {
    datasource.insertCategory(new Category(1, "Examination"));
    datasource.insertCategory(new Category(2, "Simple chat"));
```

```
    categories = datasource.getAllCategories();
  }

  categoryChoices = new ArrayList<String>();

  for (Category c : categories)
    categoryChoices.add(c.getTitle());

  fragment.setCategoryAdapter(new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, categoryChoices));
 }

 private void populateContacts() {
  ArrayList<Contact> contacts = datasource.getAllContacts();
  contactChoices = new ArrayList<String>();
  contactInfoChoices = new ArrayList<Contact>();

  for (Contact c : contacts) {
    contactChoices.add(c.getFirstName() + " " + c.getLastName());
    contactInfoChoices.add(c);
  }

  fragment.setContactAdapter(new ArrayAdapter<String>(this,
android.R.layout.simple_dropdown_item_1line, contactChoices));
  fragment.setContactIdAdapter(contactInfoChoices);
 }
```

```
private void setupActionBar() {
  getActionBar().setDisplayHomeAsUpEnabled(true);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.new_appointment, menu);
  return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home:
      NavUtils.navigateUpFromSameTask(this);
      return true;
  }
  return super.onOptionsItemSelected(item);
}

public void newAppointmentSave(View button) {
  if (fragment.formIsValid()) {
    datasource.insertAppointment(new Appointment(0, fragment.getTitle(), fragment.getStartDate(), fragment.getEndDate(), fragment.getSelectedCategory(), fragment
```

```
        .getSelectedContact(), fragment.getComments()));
      finish();
    } else
      Toast.makeText(getApplicationContext(), "Title and contact form fields are required", Toast.LENGTH_SHORT).show();

  }

  public void newAppointmentReset(View button) {
    fragment.resetForm();
  }

}
```

NewPatientActivity

```
package daybook.activities;

import daybook.db.DataSource;

import daybook.fragments.NewContactFragment;

import daybook.main.R;

import daybook.util.Contact;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.widget.Toast;

import android.support.v4.app.NavUtils;
```

```
public class NewPatientActivity extends BaseActivity {

        private DataSource datasource;

        private NewContactFragment fragment;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                if (savedInstanceState == null) {
                        fragment = new NewContactFragment();

        getSupportFragmentManager().beginTransaction().replace(R.id.content_frame
, fragment, "NewContactFragment").commit();
                }
                setupActionBar();

   datasource = new DataSource(this);
   datasource.open();
        }

  @Override
  protected void onResume() {
   super.onResume();
   datasource.open();
```

```
}

@Override
protected void onPause() {
  super.onPause();
  datasource.close();
}

        private void setupActionBar() {
                getActionBar().setDisplayHomeAsUpEnabled(true);
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                getMenuInflater().inflate(R.menu.new_contact, menu);
                return true;
        }

        @Override
        public boolean onOptionsItemSelected(MenuItem item) {
                switch (item.getItemId()) {
                case android.R.id.home:
                        NavUtils.navigateUpFromSameTask(this);
                        return true;
                }
```

```
            return super.onOptionsItemSelected(item);
        }

        public void newContactSave(View button) {
            if (fragment.formIsValid()) {
                datasource.insertContact(new Contact(0, fragment.getFirstName(), fragment.getLastName(), fragment.getAMKA(), fragment.getInsurance(), fragment.getAddress(), fragment.getPhone()));
                finish();
            }
            else
                Toast.makeText(getApplicationContext(), "All form fields are required", Toast.LENGTH_SHORT).show();
        }

        public void newContactReset(View button) {
            fragment.resetForm();
        }

}
```

NewTodoActivity

```
package daybook.activities;

import daybook.db.DataSource;

import daybook.fragments.NewTodoFragment;

import daybook.main.R;
```

```
import daybook.util.Todo;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;
import android.support.v4.app.NavUtils;

public class NewTodoActivity extends BaseActivity {

  private DataSource datasource;

  private NewTodoFragment fragment;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (savedInstanceState == null) {
      fragment = new NewTodoFragment();
      getSupportFragmentManager().beginTransaction().replace(R.id.content_frame,
fragment).commit();
    }
    setupActionBar();

    datasource = new DataSource(this);
  }
```

```
@Override
protected void onResume() {
  super.onResume();
  datasource.open();
}

@Override
protected void onPause() {
  super.onPause();
  datasource.close();
}

private void setupActionBar() {
  getActionBar().setDisplayHomeAsUpEnabled(true);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.new_todo, menu);
  return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

```
    switch (item.getItemId()) {
      case android.R.id.home:
        NavUtils.navigateUpFromSameTask(this);
        return true;
    }
    return super.onOptionsItemSelected(item);
  }

  public void newTodoSave(View button) {
    if (fragment.formIsValid()) {
      datasource.insertTodo(new Todo(0, fragment.getTitle(), fragment.getBody()));
      finish();
    } else
      Toast.makeText(getApplicationContext(), "All form fields are required",
Toast.LENGTH_SHORT).show();
  }

  public void newTodoReset(View button) {
    fragment.resetForm();
  }
}
```

PatientActivity

```
package daybook.activities;

import daybook.db.DataSource;

import daybook.fragments.ContactFragment;
```

```
import daybook.main.R;
import daybook.util.Contact;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.support.v4.app.NavUtils;

public class PatientActivity extends BaseActivity {

  private DataSource datasource;

  private ContactFragment fragment;

  private Contact contact;
  private int id;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (savedInstanceState == null) {
      fragment = new ContactFragment();
      getSupportFragmentManager().beginTransaction().replace(R.id.content_frame,
fragment).commit();
    }
    setupActionBar();
```

```
  datasource = new DataSource(this);
  datasource.open();

  Bundle extras = getIntent().getExtras();
  id = Integer.parseInt(extras.getString("id"));
  contact = datasource.getContactById(id);
  fragment.setContact(contact);
}

@Override
protected void onResume() {
  super.onResume();
  datasource.open();
}

@Override
protected void onPause() {
  super.onPause();
  datasource.close();
}

private void setupActionBar() {
  getActionBar().setDisplayHomeAsUpEnabled(true);
}
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.contact, menu);
  return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home:
      NavUtils.navigateUpFromSameTask(this);
      return true;
  }
  return super.onOptionsItemSelected(item);
}

public void toggleEdit(View button) {
  fragment.toggleEdit();
}

public void saveContact(View button) {
   datasource.updateContact(new Contact(contact.getId(), fragment.getFirstName(),
fragment.getLastName(), fragment.getAMKA(), fragment.getInsurance(),
fragment.getAddress(),
      fragment.getPhone()));
```

```
    finish();
  }

}
```

TodoActivity

```
package daybook.activities;

import daybook.db.DataSource;

import daybook.fragments.TodoFragment;

import daybook.main.R;

import daybook.util.Todo;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.support.v4.app.NavUtils;


public class TodoActivity extends BaseActivity {

  private DataSource datasource;

  private TodoFragment fragment;

  private Todo todo;

  private int id;


  @Override
```

```
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (savedInstanceState == null) {
      fragment = new TodoFragment();
     getSupportFragmentManager().beginTransaction().replace(R.id.content_frame,
fragment).commit();
    }
    setupActionBar();
    datasource = new DataSource(this);
    datasource.open();

    Bundle extras = getIntent().getExtras();
    id = Integer.parseInt(extras.getString("id"));
    todo = datasource.getTodoById(id);
    fragment.setTodo(todo);
  }

  @Override
  protected void onResume() {
    super.onResume();
    datasource.open();
  }

  @Override
  protected void onPause() {
    super.onPause();
```

```
  datasource.close();
}

private void setupActionBar() {
  getActionBar().setDisplayHomeAsUpEnabled(true);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.todo, menu);
  return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home:
      NavUtils.navigateUpFromSameTask(this);
      return true;
  }
  return super.onOptionsItemSelected(item);
}

public void toggleEdit(View button) {
  fragment.toggleEdit();
```

```
  }
  public void saveTodo(View button) {
    datasource.updateTodo(new Todo(todo.getId(), fragment.getTitle(), fragment.getBody()));
    finish();
  }

}
```

DataSource

```
package daybook.db;

import java.util.ArrayList;

import daybook.util.Appointment;

import daybook.util.Category;

import daybook.util.Contact;

import daybook.util.Todo;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;


public class DataSource {


        private SQLiteDatabase db;

        private DBTableHelper dbHelper;
```

```
private final String[] APPOINTMENT_ALL_COLUMNS = {DBTableHelper.COLUMN_APPOINTMENT_ID,
DBTableHelper.COLUMN_APPOINTMENT_TITLE,

DBTableHelper.COLUMN_APPOINTMENT_START_DATE,
DBTableHelper.COLUMN_APPOINTMENT_END_DATE,

DBTableHelper.COLUMN_APPOINTMENT_CATEGORY,
DBTableHelper.COLUMN_APPOINTMENT_CONTACT,

DBTableHelper.COLUMN_APPOINTMENT_COMMENTS};


private final String[] CONTACT_ALL_COLUMNS = {DBTableHelper.COLUMN_CONTACT_ID,
DBTableHelper.COLUMN_CONTACT_FIRST_NAME,

DBTableHelper.COLUMN_CONTACT_LAST_NAME,
DBTableHelper.COLUMN_CONTACT_AMKA,

DBTableHelper.COLUMN_CONTACT_INSURANCE,
DBTableHelper.COLUMN_CONTACT_ADDRESS,

DBTableHelper.COLUMN_CONTACT_PHONE};


private final String[] TODO_ALL_COLUMNS = {DBTableHelper.COLUMN_TODO_ID, DBTableHelper.COLUMN_TODO_TITLE,

DBTableHelper.COLUMN_TODO_BODY};


private final String[] CATEGORY_ALL_COLUMNS = {DBTableHelper.COLUMN_CATEGORY_ID,
DBTableHelper.COLUMN_CATEGORY_TITLE};
```

```
	public DataSource(Context context) {
		dbHelper = new DBTableHelper(context);
	}

	public void open() throws SQLException {
		db = dbHelper.getWritableDatabase();
	}

	public void close() {
		dbHelper.close();
	}

	/**
	 * SQL statement to insert an appointment
	 */
	public void insertAppointment(Appointment appointment) {
		ContentValues values = new ContentValues();
		values.put(DBTableHelper.COLUMN_APPOINTMENT_TITLE,
appointment.getTitle());

	values.put(DBTableHelper.COLUMN_APPOINTMENT_START_DATE,
appointment.getStartDate());

	values.put(DBTableHelper.COLUMN_APPOINTMENT_END_DATE,
appointment.getEndDate());
```

```
        values.put(DBTableHelper.COLUMN_APPOINTMENT_CATEGORY,
appointment.getCategory());

                values.put(DBTableHelper.COLUMN_APPOINTMENT_CONTACT,
appointment.getContact());


        values.put(DBTableHelper.COLUMN_APPOINTMENT_COMMENTS,
appointment.getComments());

                db.insert(DBTableHelper.TABLE_APPOINTMENT, null, values);

        }


        /**

         * SQL statement to update an appointment

         */

        public void updateAppointment(Appointment appointment) {

                ContentValues values = new ContentValues();

                values.put(DBTableHelper.COLUMN_APPOINTMENT_TITLE,
appointment.getTitle());


        values.put(DBTableHelper.COLUMN_APPOINTMENT_START_DATE,
appointment.getStartDate());


        values.put(DBTableHelper.COLUMN_APPOINTMENT_END_DATE,
appointment.getEndDate());


        values.put(DBTableHelper.COLUMN_APPOINTMENT_CATEGORY,
appointment.getCategory());

                values.put(DBTableHelper.COLUMN_APPOINTMENT_CONTACT,
appointment.getContact());
```

```
        values.put(DBTableHelper.COLUMN_APPOINTMENT_COMMENTS,
appointment.getComments());

                db.update(DBTableHelper.TABLE_APPOINTMENT,          values,
DBTableHelper.COLUMN_APPOINTMENT_ID + " = ?", new String[]
{String.valueOf(appointment.getId())});

        }


        /**

         * SQL statement to return the apppointment with the given id

         * @param id The id criterion for the search

         * @return The appointment with the given id

         */

        public Appointment getAppointmentById(int id) {

                Cursor cursor = db.query(DBTableHelper.TABLE_APPOINTMENT,
APPOINTMENT_ALL_COLUMNS,
DBTableHelper.COLUMN_APPOINTMENT_ID + " = " + id, null, null, null, null);

                if (cursor.isAfterLast()){

                        return null;

                }

                cursor.moveToFirst();

                Appointment appointment = cursorToAppointment(cursor);

                cursor.close();

                return appointment;

        }


        /**

         * SQL statement to retrieve all appointments
```

```
         * @return An ArrayList with all the retrieved appointments
         */
        public ArrayList<Appointment> getAllAppointments() {
                ArrayList<Appointment> appointments = new
ArrayList<Appointment>();
                Cursor cursor = db.query(DBTableHelper.TABLE_APPOINTMENT,
APPOINTMENT_ALL_COLUMNS, null, null, null, null, null);
                cursor.moveToFirst();
                while (!cursor.isAfterLast()) {
                        appointments.add(cursorToAppointment(cursor));
                        cursor.moveToNext();
                }
                cursor.close();
                return appointments;
        }

        /**
         * SQL statement to delete an appointment
         */
        public void deleteAppointment(int id){
                db.delete(DBTableHelper.TABLE_APPOINTMENT,
DBTableHelper.COLUMN_APPOINTMENT_ID + " = " + id, null);
        }

 /**
  * SQL statement to count all appointments on a given month and year
```

```
   * @param month The month criterion for the search
   * @param year The year criterion for the search
   * @return The count of the appointments
   */
  public int getAppointmentsCount(String month, String year) {
    ArrayList<Appointment> appointments = new ArrayList<Appointment>();
    Cursor cursor = db.query(DBTableHelper.TABLE_APPOINTMENT,
APPOINTMENT_ALL_COLUMNS,DBTableHelper.COLUMN_APPOINTMENT_
START_DATE + " LIKE '" + year + "-" + month + "%'", null, null, null, null);
    cursor.moveToFirst();
    while (!cursor.isAfterLast()) {
      appointments.add(cursorToAppointment(cursor));
      cursor.moveToNext();
    }
    cursor.close();
    return appointments.size();
  }


        /**
         * SQL statement to insert a contact
         */
        public void insertContact(Contact contact) {
                ContentValues values = new ContentValues();
                values.put(DBTableHelper.COLUMN_CONTACT_FIRST_NAME,
contact.getFirstName());
                values.put(DBTableHelper.COLUMN_CONTACT_LAST_NAME,
contact.getLastName());
```

```
   values.put(DBTableHelper.COLUMN_CONTACT_AMKA, contact.getAmka());

   values.put(DBTableHelper.COLUMN_CONTACT_INSURANCE,
contact.getInsurance());

            values.put(DBTableHelper.COLUMN_CONTACT_ADDRESS,
contact.getAddress());

            values.put(DBTableHelper.COLUMN_CONTACT_PHONE,
contact.getPhone());

            db.insert(DBTableHelper.TABLE_CONTACT, null, values);

        }


        /**

         * SQL statement to update a contact

         */

        public void updateContact(Contact contact) {

            ContentValues values = new ContentValues();

            values.put(DBTableHelper.COLUMN_CONTACT_FIRST_NAME,
contact.getFirstName());

            values.put(DBTableHelper.COLUMN_CONTACT_LAST_NAME,
contact.getLastName());

   values.put(DBTableHelper.COLUMN_CONTACT_AMKA, contact.getAmka());

   values.put(DBTableHelper.COLUMN_CONTACT_INSURANCE,
contact.getInsurance());

            values.put(DBTableHelper.COLUMN_CONTACT_ADDRESS,
contact.getAddress());

            values.put(DBTableHelper.COLUMN_CONTACT_PHONE,
contact.getPhone());

            db.update(DBTableHelper.TABLE_CONTACT,          values,
DBTableHelper.COLUMN_CONTACT_ID    +    "    =    ?",    new    String[]
{String.valueOf(contact.getId())});
```

```
}

/**
 * SQL statement to return the contact with the given id
 * @param id The id criterion for the search
 * @return The contact with the given id
 */
public Contact getContactById(int id) {
        Cursor cursor = db.query(DBTableHelper.TABLE_CONTACT, CONTACT_ALL_COLUMNS, DBTableHelper.COLUMN_CONTACT_ID + " = " + id, null, null, null, null);
        if (cursor.isAfterLast()){
                return null;
        }
        cursor.moveToFirst();
        Contact contact = cursorToContact(cursor);
        cursor.close();
        return contact;
}

/**
 * SQL statement to retrieve all contacts
 * @return An ArrayList with all the retrieved contacts
 */
public ArrayList<Contact> getAllContacts() {
        ArrayList<Contact> contacts = new ArrayList<Contact>();
```

```
		Cursor cursor = db.query(DBTableHelper.TABLE_CONTACT,
CONTACT_ALL_COLUMNS, null, null, null, null, null);

		cursor.moveToFirst();

		while (!cursor.isAfterLast()) {

			contacts.add(cursorToContact(cursor));

			cursor.moveToNext();

		}

		cursor.close();

		return contacts;

	}


	/**

	 * SQL statement to delete a contact

	 */

	public void deleteContact(int id){

		db.delete(DBTableHelper.TABLE_CONTACT,
DBTableHelper.COLUMN_CONTACT_ID + " = ?", new String[]
{String.valueOf(id)});

	}


	/**

	 * SQL statement to insert a todo

	 */

	public void insertTodo(Todo todo) {

		ContentValues values = new ContentValues();

		values.put(DBTableHelper.COLUMN_TODO_TITLE,
todo.getTitle());
```

```
        values.put(DBTableHelper.COLUMN_TODO_BODY,
todo.getBody());

        db.insert(DBTableHelper.TABLE_TODO, null, values);

    }


    /**

     * SQL statement to update a todo

     */

    public void updateTodo(Todo todo) {

        ContentValues values = new ContentValues();

        values.put(DBTableHelper.COLUMN_TODO_TITLE,
todo.getTitle());

        values.put(DBTableHelper.COLUMN_TODO_BODY,
todo.getBody());

        db.update(DBTableHelper.TABLE_TODO, values,
DBTableHelper.COLUMN_TODO_ID + " = ?", new String[]
{String.valueOf(todo.getId())});

    }


    /**

     * SQL statement to return a todo with the given id

     * @param id The id criterion for the search

     * @return The todo with the given id

     */

    public Todo getTodoById(int id) {

        Cursor cursor = db.query(DBTableHelper.TABLE_TODO,
TODO_ALL_COLUMNS, DBTableHelper.COLUMN_TODO_ID + " = " + id, null,
null, null, null);
```

```
            if (cursor.isAfterLast()){
                return null;
            }
            cursor.moveToFirst();
            Todo todo = cursorToTodo(cursor);
            cursor.close();
            return todo;
        }

        /**
         * SQL statement to retrieve all todos
         * @return An ArrayList with all the retrieved todos
         */
        public ArrayList<Todo> getAllTodos() {
            ArrayList<Todo> todos = new ArrayList<Todo>();
            Cursor cursor = db.query(DBTableHelper.TABLE_TODO,
TODO_ALL_COLUMNS, null, null, null, null, null);
            cursor.moveToFirst();
            while (!cursor.isAfterLast()) {
                todos.add(cursorToTodo(cursor));
                cursor.moveToNext();
            }
            cursor.close();
            return todos;
        }
```

```
        /**
         * SQL statement to delete a todo
         */
        public void deleteTodo(int id){
                db.delete(DBTableHelper.TABLE_TODO,
DBTableHelper.COLUMN_TODO_ID + " = " + id, null);
        }

        /**
         * SQL statement to insert a category
         */
        public void insertCategory(Category category) {
                ContentValues values = new ContentValues();
                values.put(DBTableHelper.COLUMN_CATEGORY_TITLE,
category.getTitle());
                db.insert(DBTableHelper.TABLE_CATEGORY, null, values);
        }

        /**
         * SQL statement to retrieve all categories
         * @return An ArrayList with all the retrieved categories
         */
        public ArrayList<Category> getAllCategories() {
                ArrayList<Category> categories = new ArrayList<Category>();
                Cursor cursor = db.query(DBTableHelper.TABLE_CATEGORY,
CATEGORY_ALL_COLUMNS, null, null, null, null, null);
```

```
            cursor.moveToFirst();
            while (!cursor.isAfterLast()) {
                categories.add(cursorToCategory(cursor));
                cursor.moveToNext();
            }
            cursor.close();
            return categories;
        }

        /**
         * SQL statement to return a category with the given id
         * @param id The id criterion for the search
         * @return The category with the given id
         */
        public Category getCategoryById(int id) {
            Cursor cursor = db.query(DBTableHelper.TABLE_CATEGORY,
CATEGORY_ALL_COLUMNS, DBTableHelper.COLUMN_CATEGORY_ID + " =
" + ++id, null, null, null, null);
            if (cursor.isAfterLast()){
                return null;
            }
            cursor.moveToFirst();
            Category category = cursorToCategory(cursor);
            cursor.close();
            return category;
        }
```

```
	/**
	 * Retrieve the cursor info and return them as an appointment object
	 * @return An Appointment object with the cursor info
	 */
	private Appointment cursorToAppointment(Cursor cursor) {
		return new Appointment(cursor.getInt(0), cursor.getString(1), cursor.getString(2),
				cursor.getString(3), cursor.getInt(4), cursor.getInt(5),
				cursor.getString(6));
	}

	/**
	 * Retrieve the cursor info and return them as a Contact object
	 * @return A Contact object with the cursor info
	 */
	private Contact cursorToContact(Cursor cursor) {
		return new Contact(cursor.getInt(0), cursor.getString(1), cursor.getString(2), cursor.getString(3), cursor.getString(4), cursor.getString(5), cursor.getString(6));
	}

	/**
	 * Retrieve the cursor info and return them as a Todo object
	 * @return A Todo object with the cursor info
	 */
```

```
        private Todo cursorToTodo(Cursor cursor) {
                return new Todo(cursor.getInt(0), cursor.getString(1),
cursor.getString(2));
        }

        /**
         * Retrieve the cursor info and return them as a Category object
         * @return A Category object with the cursor info
         */
        private Category cursorToCategory(Cursor cursor) {
   System.out.println("FETCHING: " + cursor);
                return new Category(cursor.getInt(0), cursor.getString(1));
        }

}
```

DBTableHelper

```
package daybook.db;

import android.content.Context;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.util.Log;


public class DBTableHelper extends SQLiteOpenHelper {


        // TABLE: APPOINTMENT
```

```
        public static final String TABLE_APPOINTMENT                    =
"appointment";

        public static final String COLUMN_APPOINTMENT_ID                =
"_id";

        public static final String COLUMN_APPOINTMENT_TITLE             =
"title";

        public static final String COLUMN_APPOINTMENT_START_DATE    =
"start_date";

        public static final String COLUMN_APPOINTMENT_END_DATE
        = "end_date";

        public static final String COLUMN_APPOINTMENT_CATEGORY
        = "category";

        public static final String COLUMN_APPOINTMENT_CONTACT
        = "contact";

        public static final String COLUMN_APPOINTMENT_COMMENTS
        = "comments";


        // TABLE: CONTACT

        public static final String TABLE_CONTACT
        = "contact";

        public static final String COLUMN_CONTACT_ID
        = "_id";

        public static final String COLUMN_CONTACT_FIRST_NAME            =
"first_name";

        public static final String COLUMN_CONTACT_LAST_NAME             =
"last_name";

 public static final String COLUMN_CONTACT_AMKA      = "amka";

 public static final String COLUMN_CONTACT_INSURANCE      = "insurance";

        public static final String COLUMN_CONTACT_ADDRESS
        = "address";
```

```
        public static final String COLUMN_CONTACT_PHONE                 =
"phone";


        // TABLE: TODO

        public static final String TABLE_TODO
        = "todo";

        public static final String COLUMN_TODO_ID
        = "_id";

        public static final String COLUMN_TODO_TITLE
        = "title";

        public static final String COLUMN_TODO_BODY
        = "body";


        // TABLE: CATEGORY

        public static final String TABLE_CATEGORY
        = "category";

        public static final String COLUMN_CATEGORY_ID
        = "_id";

        public static final String COLUMN_CATEGORY_TITLE                =
"title";


        // DATABASE NAME

        private static final String DATABASE_NAME = "daybook.db";


        // DATABASE VERSION

        private static final int DATABASE_VERSION = 2;


        // SQL STATEMENT: CREATE TABLE APPOINTMENT
```

```
        private static final String CREATE_TABLE_APPOINTMENT = "create table
"
                                        + TABLE_APPOINTMENT
                                        + "("
                                        + COLUMN_APPOINTMENT_ID + " integer
primary key autoincrement, "
                                        + COLUMN_APPOINTMENT_TITLE + " text,
"
                                        + COLUMN_APPOINTMENT_START_DATE
+ " text, "
                                        + COLUMN_APPOINTMENT_END_DATE +
" text, "
                                        + COLUMN_APPOINTMENT_CATEGORY +
" integer, "
                                        + COLUMN_APPOINTMENT_CONTACT + "
integer, "
                                        +   COLUMN_APPOINTMENT_COMMENTS
+ " text "
                                        + ");";


        // SQL STATEMENT: CREATE TABLE CONTACT
        private static final String CREATE_TABLE_CONTACT = "create table "
                        + TABLE_CONTACT
                        + "("
                        + COLUMN_CONTACT_ID + " integer primary key
autoincrement, "
                        + COLUMN_CONTACT_FIRST_NAME + " text, "
                        + COLUMN_CONTACT_LAST_NAME + " text, "
                + COLUMN_CONTACT_AMKA + " text, "
```

```
                + COLUMN_CONTACT_INSURANCE + " text, "
                        + COLUMN_CONTACT_ADDRESS + " text, "
                        + COLUMN_CONTACT_PHONE + " text "
                        + ");";

        // SQL STATEMENT: CREATE TABLE TODO
        private static final String CREATE_TABLE_TODO = "create table "
                        + TABLE_TODO
                        + "("
                        + COLUMN_TODO_ID + " integer primary key autoincrement, "
                        + COLUMN_TODO_TITLE + " text, "
                        + COLUMN_TODO_BODY + " text "
                        + ");";

        // SQL STATEMENT: CREATE TABLE CATEGORY
        private static final String CREATE_TABLE_CATEGORY = "create table "
                        + TABLE_CATEGORY
                        + "("
                        + COLUMN_CATEGORY_ID + " integer primary key autoincrement, "
                        + COLUMN_CATEGORY_TITLE + " text "
                        + ");";

        public DBTableHelper(Context context) {
                super(context, DATABASE_NAME, null, DATABASE_VERSION);
```

```
	}

	@Override
	public void onCreate(SQLiteDatabase db) {
		db.execSQL(CREATE_TABLE_APPOINTMENT);
		db.execSQL(CREATE_TABLE_CONTACT);
		db.execSQL(CREATE_TABLE_TODO);
		db.execSQL(CREATE_TABLE_CATEGORY);
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
		Log.w(DBTableHelper.class.getName(), "Upgrading database from version "
+ oldVersion + " to " + newVersion
+ ", which will destroy all old data");

		db.execSQL("DROP TABLE IF EXISTS " + TABLE_APPOINTMENT);
		db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACT);
		db.execSQL("DROP TABLE IF EXISTS " + TABLE_TODO);
		db.execSQL("DROP TABLE IF EXISTS " + TABLE_CATEGORY);
		onCreate(db);
	}
}
```