ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη Android εφαρμογής για ανταλλαγές βιβλίων σε γλώσσα Kotlin»

Της φοιτήτριας
Θεοδώρα Μπαξεβάνη
Αρ. Μητρώου: 144268

Επιβλέπων
Ευκλείδης Κεραμόπουλος
Αναπληρωτής Καθηγητής

Ημερομηνία 10 / 01 / 2020

Τίτλος Δ.Ε. «Ανάπτυξη Android εφαρμογής για ανταλλαγές βιβλίων σε γλώσσα Kotlin»
Κωδικός Δ.Ε. 20109
Ονοματεπώνυμο φοιτητή: Θεοδώρα Μπαξεβάνη
Ονοματεπώνυμο εισηγητή: Ευκλείδης Κεραμόπουλος
Ημερομηνία ανάληψης Δ.Ε. 6 / 4 / 2020
Ημερομηνία περάτωσης Δ.Ε. 10 / 01 / 2020

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Για τη μητέρα μου

Πρόλογος

Με το πέρας της παρακολούθησης όλων των μαθημάτων του προγράμματος σπουδών, κάθε φοιτητής έχει καταλήξει σε κάποια συμπεράσματα με βάση την πορεία του. Στη δική μου πορεία παρατήρησα πως ο αλγοριθμικός τρόπος σκέψης και η εφαρμογή του, αποτελούσαν τη μεγαλύτερη πρόκληση. Το γεγονός πως έπρεπε να σκεφτώ όλα τα δεδομένα ενός εκάστοτε προβλήματος και στη συνέχεια να βρω μια λύση δομημένη και με συνοχή, αποτελούσε δύσκολη συνήθως διαδικασία, η οποία όμως στο τέλος άξιζε την προσπάθεια που καταβλήθηκε. Δεδομένου επίσης πως πλέον οι συσκευές με λογισμικό Android είναι τόσο μαζικά χρησιμοποιούμενες, δεν δυσκολεύτηκα να επιλέξω τη δημιουργία μιας εφαρμογής ως θέμα διπλωματικής εργασίας. Για να γίνει πιο ενδιαφέρουσα, συμφωνήθηκε με τον καθηγητή κ. Κεραμόπουλο η συγγραφή της με χρήση της γλώσσας Kotlin, μια γλώσσα προγραμματισμού που μετρά λίγα μόνο χρόνια χρήσης. Έχοντας φτάσει πια στην περάτωση της εργασίας, νιώθω δικαιωμένη για την επιλογή αυτού του θέματος, καθώς κατά την εκπόνησή του είχα την αίσθηση πως καταπιάνομαι με κάτι σύγχρονο που έχει μέλλον, ενώ ταυτόχρονα είχα την ευκαιρία να γνωρίσω αρκετές από τις τελευταίες χρησιμοποιούμενες τάσεις στην ανάπτυξη εφαρμογών Android. Η επιλογή του θέματος της εργασίας, δηλαδή αυτό της ανταλλαγής βιβλίων, προήλθε μέσω πρότερης δικής μου χρήσης άλλων διαδικτυακών μέσων για την πραγματοποίηση πληθώρας ανταλλαγών ανά τα χρόνια, οι οποίες όμως λάμβαναν χώρα μέσα από πλατφόρμες όπως forum ή κοινωνικά δίκτυα που δεν είχαν φτιαχτεί γι' αυτόν τον σκοπό. Έτσι λοιπόν αυτή η εφαρμογή αφιερώθηκε αποκλειστικά και μόνο σε αυτή την διαδικασία, με στόχο να την καταστήσει ευκολότερη και γρηγορότερη για τους συναλλασσόμενους. Το τελικό αποτέλεσμα αποτελεί ηθική ανταμοιβή, καθώς πίσω από κάθε λειτουργία της εφαρμογής, βρίσκεται ένας συνδυασμός παλαιών και νέων γνώσεων που αποκτήθηκαν κατά την εξέλιξή της.

Περίληψη

Η παρούσα διπλωματική εργασία αφορά τη δημιουργία μιας εφαρμογής Android για ανταλλαγές βιβλίων. Μέσω αυτής της εφαρμογής ένας χρήστης μπορεί να διαθέτει όσα βιβλία επιθυμεί προς ανταλλαγή με τους υπόλοιπους χρήστες, αλλά και να αναζητά βιβλία που ο ίδιος θέλει να αποκτήσει. Η εκδήλωση ενδιαφέροντος για την έναρξη μιας ανταλλαγής και η μετέπειτα επικοινωνία, πραγματοποιούνται εντός της εφαρμογής μέσω μηνυμάτων μέχρι την ολοκλήρωσή της. Για τον κάθε χρήστη τηρείται ένα ιστορικό ανταλλαγών, στο οποίο παρουσιάζεται συνοπτικά κάθε πραγματοποιημένη ανταλλαγή, παρέχοντας έτσι μια συνολική εικόνα για τις συναλλαγές του. Η συγγραφή του κώδικα πραγματοποιήθηκε στο ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) Android Studio με τη γλώσσα προγραμματισμού Kotlin. Η βάση δεδομένων της εφαρμογής δημιουργήθηκε στην πλατφόρμα Firebase της εταιρίας Google, χρησιμοποιώντας το προϊόν Cloud Firestore.

«Development of an Android application in Kotlin for book exchanges»

Theodora Baxevani

Abstract

The subject of this thesis is the development of an Android application for book exchanges. With this application, the users can exchange books, and also search for books that they wants to acquire. A user expresses interest for an exchange and communicates through real-time conversation within the application until it’s completed. A swap history is kept for each user, which summarizes exchanges been made, thus providing an overall picture of their transactions. The code is written in Android Studio IDE, using Kotlin. The application’s database was created on Google's Firebase platform, using their Cloud Firestore product.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους τους αξιόλογους καθηγητές του τμήματος, για τις γνώσεις που απέκτησα χάρη σ' εκείνους, για τις απορίες που πρόθυμα έλυσαν, την πίστη τους ότι μπορούμε να πετύχουμε στις σπουδές μας αλλά και το γενικότερο ενδιαφέρον που έδειξαν σε όποια δυσκολία προέκυψε. Όλα τα παραπάνω αποτέλεσαν εφόδια που συνέβαλλαν σημαντικά στην δυνατότητα συγγραφής αυτής της διπλωματικής εργασίας. Για την δεκτικότητά του στην επιθυμία μου για ανάληψη της εργασίας ως επιβλέπων, τις προτάσεις βελτίωσης, την εμπιστοσύνη του μέχρι τέλους, και τη γενικότερη σοβαρή του στάση όλα τα χρόνια ως καθηγητή, ευχαριστώ ιδιαίτερα τον κ. Κεραμόπουλο.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

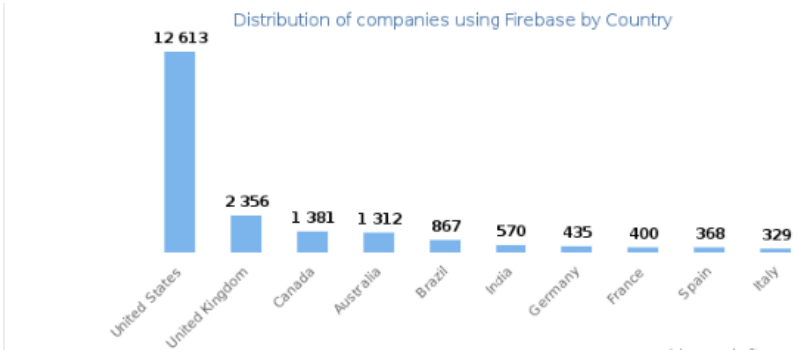
Δ.Ε.	Διπλωματική Εργασία
Η/Υ	Ηλεκτρονικός Υπολογιστής
SDK	Software Development Kit
UML	Unified Modeling Language
XML	Extensible Markup Language
BoM	Bill of Materials
AES	Advanced Encryption Standard
TLS	Transport Layer Security

Κεφάλαιο 1ο: Εισαγωγή

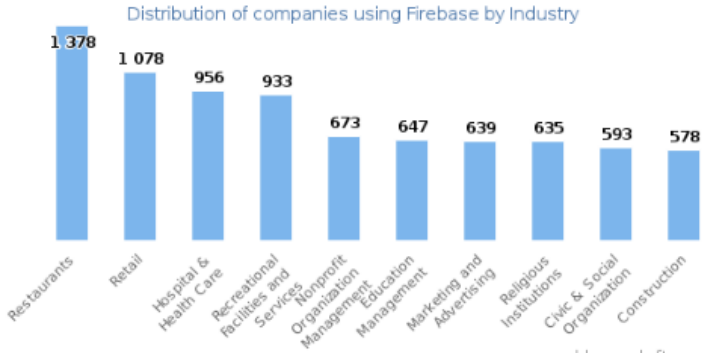
Η όλο και αυξανόμενη δημοφιλία των κινητών συσκευών Android την τελευταία δεκαετία, σε συνδυασμό με την ταυτόχρονη ανάπτυξη της τεχνολογίας των οθονών αφής, οδήγησαν το κοινό στη χρήση τους για ποικίλες δραστηριότητες. Το κοινό αυτό περιλαμβάνει ακόμη και ανθρώπους μεγαλύτερης ηλικίας, που δεν είχαν καμία πρότερη γνώση χειρισμού ηλεκτρονικού υπολογιστή (Η/Υ). Η συσκευή που παλαιότερα χρησιμοποιούσαν ως κινητό τηλέφωνο, σταδιακά άρχισε να εξελίσσεται, ώσπου τα τελευταία χρόνια κατέληξε να είναι ένας μικρού μεγέθους, πλήρης, πανίσχυρος Η/Υ. Ένα άλλο φαινόμενο που εξελίχθηκε ταυτόχρονα, είναι η ραγδαία διάδοση των ψηφιακών κοινωνικών μέσων δικτύωσης (Facebook, Twitter κ.α.). Νέοι τρόποι προσέγγισης δραστηριοτήτων οι οποίες παραδοσιακά είχαν σημείο αφετηρίας έντυπα μέσα (αγγελίες σε εφημερίδες ή περιοδικά), όπως είναι οι αγοραπωλησίες αντικειμένων, βρήκαν το χώρο τους μέσα στα κοινωνικά δίκτυα. Ομάδες (Groups) και σελίδες (Pages) στο Facebook δημιουργήθηκαν με σκοπό αγορές, πωλήσεις, ανταλλαγές και δωρεές αντικειμένων. Αυτή η τάση προσέγγισε μεγάλο πλήθος χρηστών εξ ’αρχής. Συγκεκριμένα στο Facebook πάνω από 450 εκατομμύρια ανθρώπους το μήνα, με αποτέλεσμα η εταιρία να προχωρήσει το 2016 στη δημιουργία ξεχωριστού χώρου γι’ αυτού του είδους τις συναλλαγές (Facebook Marketplace) [1]. Ένα είδος ανταλλαγής είναι η ανταλλαγή βιβλίων, η οποία από την εποχή ακόμα της εφεύρεσης της τυπογραφίας είναι δημοφιλής. Κατά κανόνα, λαμβάνει χώρα μέσα σε φυσικά καταστήματα (βιβλιοπωλεία, παλαιοπωλεία), όμως κι αυτή εξελίχθηκε μέσα από τα ψηφιακά εργαλεία δικτύωσης. Forums και pages ασχολούνται αποκλειστικά με ανταλλαγές βιβλίων, των οποίων οι αναγνώστες δεν επιθυμούν να κρατήσουν άλλο στη βιβλιοθήκη τους.

Βέβαια, τα παραπάνω δύσκολα θα είχαν επιτευχθεί χωρίς την παράλληλη εξέλιξη και βελτίωση των υπηρεσιών Νέφους (Cloud), οι οποίες προσέφεραν στους προγραμματιστές προσιτά και εύχρηστα περιβάλλοντα για την αποθήκευση και τον συγχρονισμό των δεδομένων των εφαρμογών τους. Μια τέτοια πλατφόρμα είναι το Firebase. Το Firebase προέκυψε από το Envolve, μια chat υπηρεσία για ιστοσελίδες, όταν παρατηρήθηκε πως οι χρήστες της χρησιμοποιούσαν και για άλλες εργασίες δεδομένα πραγματικού χρόνου (real-time data). Το Firebase αρχικά ήταν προϊόν της ανεξάρτητης ομώνυμης startup εταιρίας που ιδρύθηκε το 2011 από τους James Tamplin και Andrew Lee, ενώ το 2014 αγοράστηκε από την εταιρία Google η οποία το έκανε μέλος της Google Cloud πλατφόρμας της [2]. Κατά τη συγγραφή της παρούσας εργασίας, το Firebase παρείχε τα εξής προϊόντα: Cloud Firestore, Firebase ML, Cloud Functions, Authentication, Hosting, Cloud Storage, Realtime Database. Σήμερα το Firebase χρησιμοποιείται επίσημα σε 38.888 εταιρίες, ανάμεσα στις οποίες βρίσκεται η Atlassian, το Alibaba, το AliExpress, το CBS, και το Twich. Στο Σχήμα 1 παρουσιάζονται οι χώρες στις οποίες χρησιμοποιείται περισσότερο, και στο Σχήμα 2 παρουσιάζονται οι κλάδοι στους οποίους έχει τη μεγαλύτερη απήχηση.

Το 40% των Firebase πελατών βρίσκεται στις Η.Π.Α. και το 8% στο Η.Β.

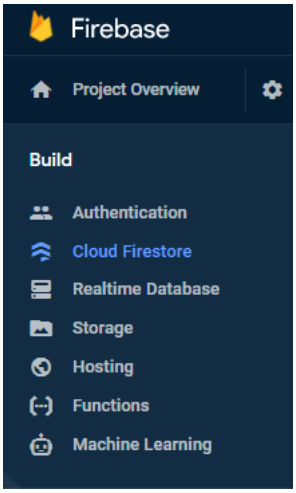


Σχήμα 1. Κατανομή εταιριών που χρησιμοποιούν το Firebase ανά χώρα (πηγή: https://enlyft.com/tech/products/firebase)



Σχήμα 2. Κατανομή εταιριών που χρησιμοποιούν το Firebase ανά κλάδο (πηγή: https://enlyft.com/tech/products/firebase)

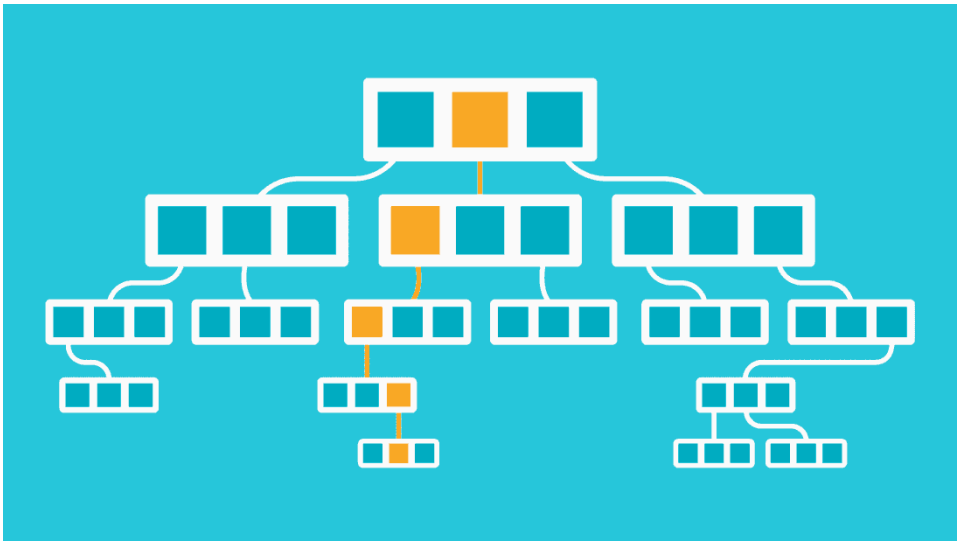
Στο Σχήμα 3 παρουσιάζονται τα κύρια εργαλεία που προσφέρει το Firebase για ένα project. Σε αυτά ανήκει επίσης το Google Analytics, το Crashlytics, το Remote Config, το Performance Monitoring, το Test Lab και το Cloud Messaging.



Σχήμα 3. Εργαλεία ενός Firebase project (πηγή: https://console.firebase.google.com)

Από τα παραπάνω, για τη δημιουργία της εφαρμογής χρησιμοποιήθηκαν:

- **Cloud Firestore**. Το Cloud Firestore είναι μια NoSQL (non-relational) cloud βάση δεδομένων (cloud database) που προσφέρεται σε εφαρμογές client και server side για αποθήκευση, συγχρονισμό δεδομένων, και δημιουργία ερωτημάτων προς τη βάση. Εφαρμογές web, iOS και Android έχουν απευθείας πρόσβαση στην βάση μέσω ενός native Κιτ Ανάπτυξης Λογισμικού (Software Development Kit - SDK). Το Cloud Firestore είναι επίσης διαθέσιμο για native Node.js, Java, Python, Unity, C++ και Go SDKs, όπως επίσης και για REST ή RPC APIs. Η δομή μιας βάσης δεδομένων αυτού του τύπου αποτελείται από συλλογές (collections) με αρχεία (documents) αντί για πίνακες με κελιά που χρησιμοποιούνται σε μια κλασσική σχεσιακή βάση δεδομένων. Πιο συγκεκριμένα, κάθε collection αποτελείται από ένα ή περισσότερα document του ίδιου τύπου. Κάθε document περιέχει fields στα οποία αποθηκεύονται δεδομένα διαφόρων τύπων (string, numbers, objects) που αφορούν το εκάστοτε document. Ένα document μπορεί να περιέχει ακόμη και μια υπό-συλλογή (subcollection) χτίζοντας έτσι ιεραρχικές δομές δεδομένων [3]. Μια πρώτη εικόνα της ευέλικτης δομής των βάσεων δεδομένων στο Firestore, φαίνεται στο Σχήμα 4.



Σχήμα 4. Η δομή μιας βάσης δεδομένων στο Firestore
(πηγή: https://firebase.google.com/products/firestore)

Ασφάλεια

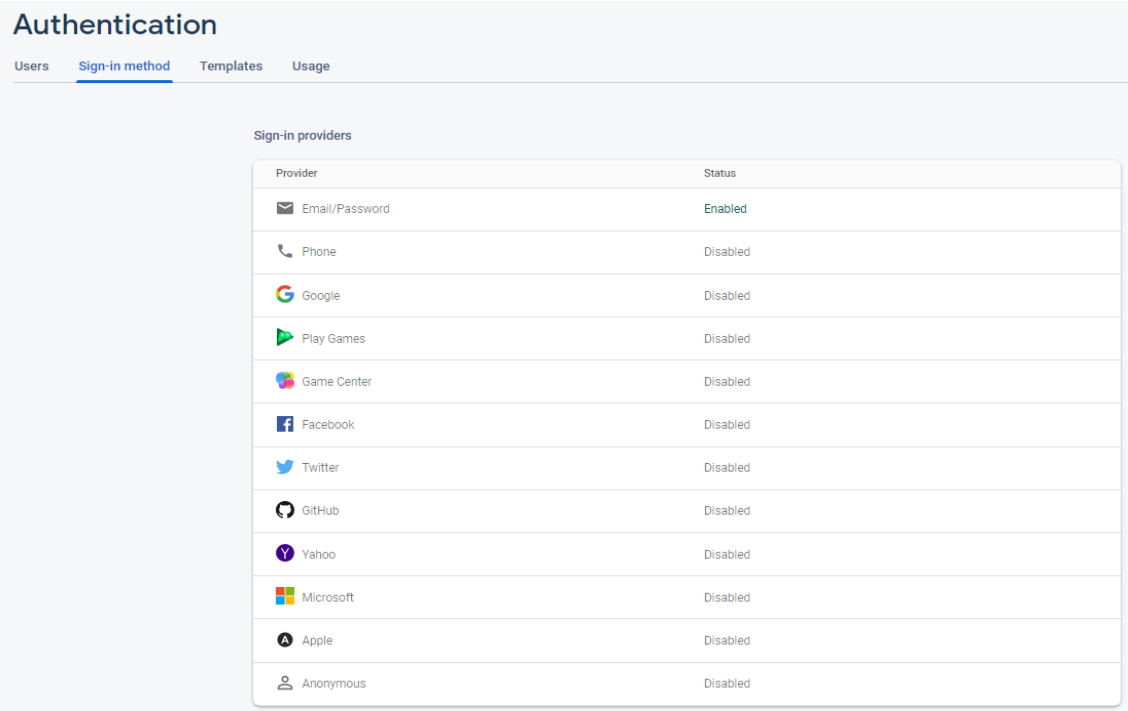
Το γεγονός ότι στο Firestore μπορεί να αποθηκευτεί οποιαδήποτε ποσότητα δεδομένων, από kilobytes μέχρι petabytes, χωρίς αυτό να επηρεάζει την απόδοση χειρισμού των δεδομένων, το καθιστά μια ικανή επιλογή για υποστήριξη μεγάλων εφαρμογών (large applications). Όλα τα δεδομένα κρυπτογραφούνται αυτόματα προτού αποθηκευτούν στην πλευρά του server (server-side encryption), και αποκρυπτογραφούνται κάθε φορά που πρόκειται να αναγνωστούν από έναν εγκεκριμένο (authorized) χρήστη. Η κρυπτογράφηση των Firestore αντικειμένων και metadata, γίνεται με βάση το 256-bit Advanced Encryption Standard (AES) πρότυπο, και κάθε κλειδί κρυπτογράφησης κρυπτογραφείται επίσης σε τακτά χρονικά διαστήματα. Προσφέρεται επίσης η δυνατότητα συνδυασμού του server-side encryption με client-side encryption, δηλαδή η κρυπτογράφηση των δεδομένων προτού εγγραφούν στο Firestore. Έτσι τα δεδομένα θα κρυπτογραφούνται δύον φορές, μία με τα τοπικά κλειδιά κρυπτογράφησης, και μια με εκείνα της Google. Τέλος, για την προστασία των δεδομένων καθώς αυτά ταξιδεύουν εντός του Internet κατά τη διάρκεια read και write λειτουργιών,

χρησιμοποιούνται τα πρωτόκολλα του Transport Layer Security (TLS), το οποίο αποτελεί τον αντικαταστάτη του πρώην Secure Sockets Layer (SSL) [4].

Συγχρονισμός δεδομένων

Ένα ισχυρό πλεονέκτημα του Cloud Firestore είναι το offline persistence. Δηλαδή η δυνατότητα συγχρονισμού των δεδομένων της εφαρμογής online αλλά και offline. Αυτό σημαίνει πως οι χρήστες μπορούν να έχουν πρόσβαση στα δεδομένα της εφαρμογής και να τα τροποποιούν, ακόμη και όταν η συσκευή τους δεν είναι συνδεδεμένη στο Internet. Οι αλλαγές που πραγματοποιούν διατηρούνται τοπικά, και μόλις η συσκευή αποκτήσει πρόσβαση στο Internet, γίνεται αυτόματα συγχρονισμός των τοπικών αλλαγών, με τα δεδομένα στο Cloud Firestore backend. Αυτή η λειτουργία (feature) παρέχεται μέσω caching των Cloud Firestore data που χρησιμοποιούνται ενεργά. Έτσι είναι εφικτή η ανάγνωση, η εγγραφή, η ακρόαση listeners και η υποβολή ερωτημάτων (queries) στα cached δεδομένα. Φυσικά είναι στην ευχέρεια του προγραμματιστή να απενεργοποιήσει αυτή τη δυνατότητα, η οποία είναι ενεργοποιημένη από προεπιλογή, εάν δεν είναι επιθυμητή σε μια εφαρμογή.

- **Firebase Authentication**. Το Firebase Authentication προσφέρει backend υπηρεσίες, SDKs και έτοιμες προς χρήση UI βιβλιοθήκες, ώστε να παρέχει επιβεβαίωση στοιχείων (αυθεντικοποίηση) στους χρήστες μιας εφαρμογής, προκειμένου να αποκτήσουν πρόσβαση (sign in) με το λογαριασμό τους. Η ταυτοποίηση μπορεί να γίνει με διάφορους τρόπους, όπως χρήση κωδικού (password), email, αριθμού κινητού τηλεφώνου ή μέσω δημοφιλών πιστοποιημένων (federated) παρόχων ταυτοποίησης (Google, Facebook, Twitter κ.α.) που παρουσιάζονται στο Σχήμα 5. Το Firebase Authentication σχετίζεται στενά με τις υπόλοιπες υπηρεσίες-προϊόντα του Firebase. Αξιοποιεί επίσης υπάρχοντα industrial πρότυπα (standards) όπως το OAuth 2.0 και το OpenID Connect, καταφέρνοντας έτσι να ενσωματώνεται έχοντας συμβατότητα με ένα custom backend [5]. Το Firebase αναπτύχθηκε από την ίδια ομάδα ανθρώπων που δημιούργησε το Google Sign-in, το Smart Lock και το Chrome Password Manager. Έτσι στο Firestore εφαρμόστηκε αυτή η εμπειρογνωσία σε ότι αφορά θέματα ασφάλειας και διαχείρισης της βάσης δεδομένων.



Σχήμα 5. Οι μέθοδοι αυθεντικοποίησης που προσφέρονται από το Firebase (πηγή: https://console.firebase.google.com/)

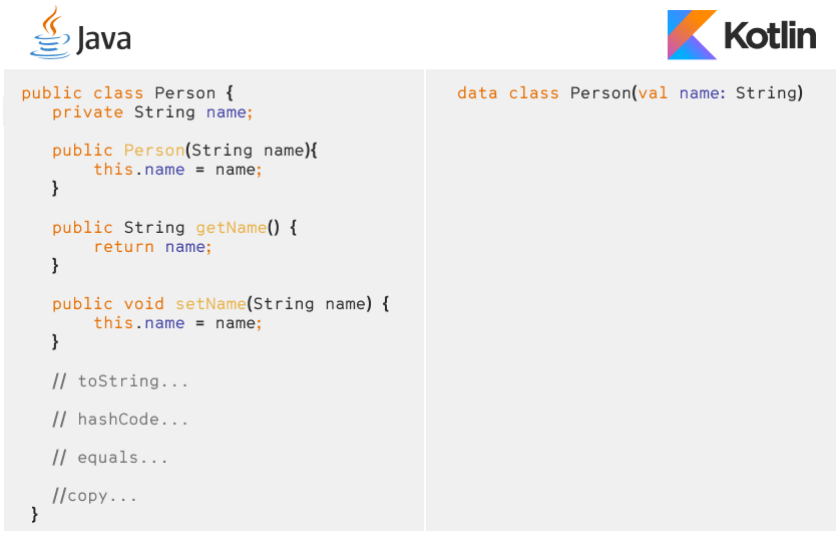
Για τη συγγραφή του κώδικα της εφαρμογής, επιλέχθηκε η γλώσσα προγραμματισμού Kotlin, καθώς κατά την χρονική στιγμή ανάθεσης της Δ.Ε. ήταν η προτεινόμενη γλώσσα ανάπτυξης εφαρμογών για το λειτουργικό σύστημα Android. Η Kotlin έχει πλήρη διαλειτουργικότητα (interoperability) με την Java και αποτελεί μια δωρεάν για χρήση γλώσσα, υπό την άδεια (license) Apache 2. Καθώς είναι προϊόν ανοικτού κώδικα (open-source), ο πηγαίος της κώδικας (source code) είναι ελεύθερα προσβάσιμος στο ψηφιακό αποθετήριο GitHub. Ανήκει στις statically typed γλώσσες προγραμματισμού και αναπτύχθηκε από την εταιρία JetBrains η οποία την ανακοίνωσε το 2011. Ωστόσο η πρώτη επίσημη κυκλοφορία (realise) 1.0 της, έγινε τον Φεβρουάριο του 2016. Μπορεί να μεταγλωττιστεί (compile down) σε Java bytecode, άρα να λειτουργεί σε Java virtual machine (JVM) και Android. Επίσης μπορεί να μεταγλωττιστεί σε JavaScript και έτσι να λειτουργεί σε ενσωματωμένες (embedded) και iOS συσκευές [3]. Οι δημιουργοί της γλώσσας εμπνεύστηκαν το όνομά της από το νησί Kotlin, που βρίσκεται κοντά στην Αγία Πετρούπολη της Ρωσίας, όπως αντίστοιχα πήρε την ονομασία της και η γλώσσα Java από το ομώνυμο νησί στην Ινδονησία [6]. Σύμφωνα με τον επικεφαλή της JetBrains, Dmitry Jemerov, καμία γλώσσα προγραμματισμού για JVM δεν είχε όλα τα χαρακτηριστικά (features) που η ομάδα τους χρειαζόταν, πέραν της γλώσσας Scala, η οποία όμως εξ' αιτίας του πολύ αργού της compilation, ήταν ανεπαρκής [7]. Η δημιουργία της νέας γλώσσας Kotlin λοιπόν, είχε ρητό της στόχο πως θα είναι εξίσου γρήγορη στο compile, όσο και η μέχρι τότε προτιμώμενη για προγραμματισμό Android γλώσσα, Java. Το 2017 κατά τη διάρκεια του Google I/O συνεδρίου, η Google ανακοίνωσε πως θα παρέχει πρώτης τάξεως υποστήριξη για τον προγραμματισμό με Kotlin για εφαρμογές Android, ενσωματώνοντάς την μάλιστα στην τότε πιο πρόσφατη έκδοση του Android Studio 3.0 [8]. Δύο χρόνια αργότερα, τον Μάιο του 2019 στο ίδιο συνέδριο, κοινοποιήθηκε ότι η Kotlin θα αποτελεί πλέον την προτεινόμενη γλώσσα προγραμματισμού για ανάπτυξη Android εφαρμογών, συστήνοντας τη χρήση της σε όσους ξεκινούν νέα project [9].

Η Java μέχρι τον Μάιο του 2019 ήταν η προτεινόμενη γλώσσα προγραμματισμού για Andoid. Σύμφωνα με τους δημιουργούς της Kotlin, αυτή εξαλείφει κάποια προβλήματα της Java όπως:

- Έλεγχος των αναφορών σε null
- Απαιτεί συνήθως λίγες γραμμές κώδικα (verbose) όπως φαίνεται στο Σχήμα 6
- Δεν υπάρχουν τύποι raw
- Οι πίνακες στην Kotlin είναι αμετάβλητοι (invariant)
- Η Kotlin δεν έχει checked exceptions
- Συνολικά λιγότερα app crash
- Ευκολία εκμάθησης - κατάλληλη για junior developers
- Ανεξαρτησία από την πλατφόρμα στην οποία χρησιμοποιείται
- Είναι μια open-source γλώσσα
- Είναι μια μοντέρνα και εκφραστική γλώσσα
- Ανεξαρτησία από τις διάφορες εκδόσεις (versions) της Java

Ενώ ταυτόχρονα εισάγει νέες έννοιες και πρακτικές, όπως:

- Εκφράσεις Lambda με Inline functions
- Μεθόδους (functions) extension
- Δήλωση Nullable τύπων σε μεταβλητές
- Προστασία από τιμές Null
- Πλήρη συμπερασμό τύπων (type inference)
- Smart casts
- String templates
- Named και default properties
- Πρωταρχικοί δομητές (Primary constructors)
- First-class delegation
- Singletons
- Range expressions
- Υπερφόρτωση τελεστών
- Αντικείμενα Companion
- Data Classes
- Ξεχωριστά interfaces για collections μόνο-ανάγνωσης και mutable
- Coroutines
- Kotlin Standard library



Σχήμα 6. Σύγκριση μιας κλάσης σε Java και Kotlin (πηγή: https://mlsdev.com/blog/kotlin-vs-java)

Το σημαντικότερο γνώρισμα της Kotlin είναι η διαλειτουργικότητά της με τη Java. Δηλαδή, το γεγονός ότι είναι δυνατή η συνύπαρξη κώδικα Kotlin και Java εντός του ίδιου project (Interoperability) και το ότι η Kotlin μπορεί να χρησιμοποιήσει τις βιβλιοθήκες της Java [10]. Η διαλειτουργικότητα ήταν απαραίτητη προυπόθεση για την δημιουργία της Kotlin, έτσι ώστε να μπορεί να χρησιμοποιηθεί στις εφαρμογές που ήδη είχαν γραφεί σε Java.

Από την άλλη, υπάρχουν κάποια χαρακτηριστικά που λείπουν από την Kotlin, όπως:

- Πρωταρχικοί τύποι δεδομένων που δεν είναι κλάσεις
- Static μεταβλητές. Στη θέση του static μπορούν να χρησιμοποιηθούν companion objects, μέθοδοι top-level και extension, ή @JvmStatic.
- Τύποι δεδομένων Wildcard. Μπορούν να αντικατασταθούν από διακύμανση δήλωσης (declaration-site variance) και προβολές τύπων (type projections).
- Ternary τελεστές a ? b : c. Αντικαθίστανται με IF expressions.

Η εφαρμογή που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής εργασίας (Δ.Ε.), στοχεύει στο κοινό που επιθυμεί να πραγματοποιεί διαδικτυακά ανταλλαγές βιβλίων, προσφέροντάς τους ένα κατάλληλα σχεδιασμένο περιβάλλον όπου η διαδικασία καθίσταται εύκολη και ευχάριστη μέσω μιας Android κινητής συσκευής τους. Αρχικά ένας χρήστης αναζητά τον τίτλο κάποιου βιβλίου που τον ενδιαφέρει. Από τη λίστα των διαθέσιμων που εμφανίζεται πως διαθέτουν άλλοι χρήστες, ο ενδιαφερόμενος αποστέλλει μήνυμα στον ιδιοκτήτη για όποιο επιθυμεί. Έτσι οι δύο χρήστες διαπραγματεύονται μέσω προσωπικών μηνυμάτων εντός της εφαρμογής για τα βιβλία που θα συμμετέχουν στην ανταλλαγή τους, μέχρι αυτή να συμφωνηθεί. Όταν αυτή ολοκληρωθεί, ο χρήστης που έδειξε αρχικά ενδιαφέρον για την ανταλλαγή, ενημερώνει την εφαρμογή για την επιτυχή πραγματοποίησή της, και αυτή προστίθεται στο ιστορικό ανταλλαγών που τηρείται για τον καθένα ξεχωριστά. Το ιστορικό μπορεί να χρησιμοποιηθεί ως σημείο αναφοράς για μελλοντικές ανταλλαγές ή μη, προσφέροντας μια συνολική εικόνα χρήσης. Στα κεφάλαια που ακολουθούν, περιγράφεται η διαδικασία υλοποίησης της εφαρμογής, καθώς και τα εργαλεία που χρησιμοποιήθηκαν.

Στο Κεφάλαιο 1 βρίσκεται το θεωρητικό υπόβαθρο που στήριξε την παρούσα εργασία, παρουσιάζοντας τα δεδομένα που λήφθηκαν υπόψιν και καθόρισαν τις προδιαγραφές της εφαρμογής. Εκεί περιλαμβάνονται επίσης τα αρχικά στάδια προτυποποίησης καθώς και η επιλογή των κλάσεων που χρησιμοποιήθηκαν, με βάση την αντικειμενοστρεφή σχεδίαση.

Στο Κεφάλαιο 2 παρουσιάζεται η υλοποίηση όλων όσων αποφασίστηκαν στο Κεφάλαιο 1. Η υλοποίησή τους έγινε στο Android Studio, όπου με τη βοήθεια διαφόρων εικονικών συσκευών (virtual devices), του debugger, και των άλλων εργαλείων του, δημιουργήθηκαν οι οθόνες της εφαρμογής και τα αντίστοιχά τους Activities. Κατόπιν εξηγήθηκε λεπτομερώς η λειτουργικότητα κάθε οθόνης, με αναφορές στα components που την απαρτίζουν.

Στο Κεφάλαιο 3 επιλέχθηκαν ενδεικτικά τμήματα του κώδικα που κρίθηκαν ιδιαίτερα ενδιαφέροντα, ή παρουσίαζαν αυξημένο βαθμό δυσκολίας, και αναλύθηκαν διεξοδικά. Παρουσιάστηκε επίσης η διαδικασία δημιουργίας της βάσης δεδομένων στο Cloud Firestore, και οι μέθοδοι χειρισμού της.

Στο Κεφάλαιο 4 γίνεται μια συνολική αποτίμηση με βάση τους στόχους που τέθηκαν αρχικά, και τη χρήση των εργαλείων που χρησιμοποιήθηκαν, κλείνοντας με μερικές προτεινόμενες ιδέες για περαιτέρω βελτίωση της εφαρμογής.

Κεφάλαιο 2ο: Σχεδίαση του Λογισμικού

2.1 Εισαγωγή

Η σχεδίαση του λογισμικού είναι η δημιουργική διεργασία της μετατροπής του προβλήματος σε λύση [11]. Αρχικά έγινε η εξαγωγή των απαιτήσεων του συστήματος. Στη συνέχεια σχεδιάστηκαν οι πρώτες δοκιμαστικές οθόνες του λογισμικού και καθορίστηκαν οι μεταξύ τους σχέσεις και μεταβάσεις. Ακολούθησε η σχεδίαση και συσχέτιση των κλάσεων με αντικειμενοστρεφή αναπαράσταση, και έγινε διάγραμμα UML. Έπειτα δημιουργήθηκαν στο Android Studio τα Activities της εφαρμογής με τα αντίστοιχα layouts. Παρομοίως και οι απαραίτητες κλάσεις για τη νοηματική διασύνδεση της βάση δεδομένων στο Firestore.

Το μοντέλο ανάπτυξης λογισμικού που χρησιμοποιήθηκε, είναι αυτό του καταρράκτη (waterfall model). Προτιμήθηκε καθώς αποτελεί μια εύκολα κατανοητή διαδικασία, στην οποία περιγράφονται με σαφήνεια όλες οι επιμέρους φάσεις ανάπτυξης. Το μοντέλο αυτό λοιπόν καθοδήγησε τον τρόπο σκέψης και δράσης. Στο πρώτο στάδιο, έγινε η γενική περιγραφή του προβλήματος και η προτεινόμενη λύση-στόχος, όπως αυτά περιγράφηκαν στο τελευταίο σκέλος της Εισαγωγής. Έπειτα σειρά είχε το στάδιο της ανάλυσης του προβλήματος, στο οποίο καταγράφηκαν οι απαιτήσεις της εφαρμογής.

2.2 Απαιτήσεις του συστήματος

Για τον καθορισμό των προδιαγραφών του λογισμικού, λήφθηκαν υπ’όψιν τα κάτωθι:

- Η υπάρχουσα ανάγκη την οποία καλείται να καλύψει το λογισμικό. Η διαδικασία ανταλλαγής βιβλίων μεταξύ φυσικών προσώπων χρήζει διευκόλυνσης ως προς την αναζήτηση τίτλων, την επικοινωνία μεταξύ τους, τη διατήρηση ιστορικού ανταλλαγών και τη δυνατότητα επιλογής του βιβλίου-ανταλλάγματος από τη συλλογή του άλλου προσώπου.
- Το προφίλ των χρηστών – ο ανθρώπινος παράγοντας. Τα άτομα που ανταλλάσσουν βιβλία μπορούν να είναι κάθε ηλικίας, φύλου και μορφωτικού επιπέδου. Το γεγονός αυτό, καθιστά απαραίτητη την υλοποίηση μιας εφαρμογής εύχρηστης, κατανοητής, χωρίς πολύπλοκες λειτουργίες, με κατάλληλη σχεδίαση διεπαφών, επιλογή συνδυασμού χρωμάτων και τοποθέτηση των κουμπιών σε εμφανή σημεία με αποφυγή «κρυφών» μενού και άλλων λειτουργιών που απαιτούν πολλαπλές ενέργειες διάδρασης (touches) για την εμφάνισή τους.
- Ασφάλεια – Προστασία των δεδομένων του χρήστη. Η έναρξη χρήσης της εφαρμογής θα πρέπει να έπεται μιας διαδικασίας αυθεντικοποίησης – ταυτοποίησης του χρήστη. Η είσοδος στην εφαρμογή θα πρέπει να προστατεύεται με προσωπικό κωδικό (password). Αυτές οι απαιτήσεις είναι απαραίτητες, καθώς η βάση δεδομένων της εφαρμογής θα περιέχει προσωπικά δεδομένα και συνομιλίες.
- Στοχευόμενες συσκευές. Δεδομένου του προφίλ των χρηστών, το λογισμικό θα πρέπει να μπορεί να εκτελείται σε πλήθος διαφορετικών συσκευών, χωρίς ιδιαίτερες απαιτήσεις σε πόρους. Αυτό συνεπάγεται την υποστήριξη smartphones και tablets με την τρέχουσα έκδοση του λειτουργικού συστήματος Android, αλλά και παλαιότερων.

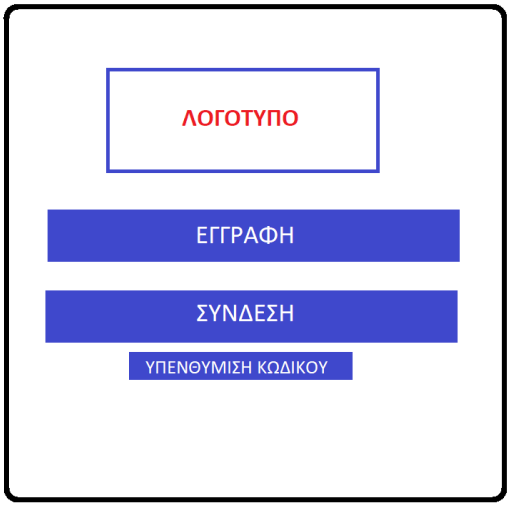
- Σύνδεση στο Διαδίκτυο. Ο χρήστης θα πρέπει να μπορεί να απαντά σε μηνύματα ή να εκδηλώνει ενδιαφέρον για μια ανταλλαγή, ακόμη και όταν βρίσκεται εκτός σύνδεσης. Τα δεδομένα θα πρέπει να συγχρονίζονται και να ανανεώνονται μόλις γίνει και πάλι διαθέσιμη η σύνδεση στο Διαδίκτυο.
- Δεδομένα. Τα στοιχεία σύνδεσης του χρήστη (email και password), οι πληροφορίες που αφορούν τα βιβλία (τίτλος, συγγραφέας, έτος έκδοσης, εκδοτικός οίκος, κατάσταση βιβλίου, εξώφυλλο), τα μηνύματα των συνομιλιών (κείμενο, ημερομηνία, αποστολέας, παραλήπτης) θα αποθηκεύονται στη βάση δεδομένων τόσο τοπικά, όσο και σε cloud server. Τα δεδομένα του χρήστη θα διασφαλίζονται σε περίπτωση απώλειας ή βλάβης της συσκευής του, και θα μπορούν να επαναφέρονται σε μια νέα συσκευή.

2.3 Σχεδίαση των οθονών

Με βάση τις απαιτήσεις της εφαρμογής που περιγράφηκαν παραπάνω, ακολούθησε το επόμενο στάδιο της ανάπτυξης, αυτό του σχεδιασμού. Δηλαδή, πραγματοποιήθηκε μια λεπτομερής περιγραφή της επιθυμητής τελικής εφαρμογής, επαρκής για την ανάπτυξή της. Η διεπαφή του λογισμικού (User Interface – UI) και οι μεταβάσεις μεταξύ των οθονών όφειλαν να γίνουν έχοντας κατά νου το προφίλ των υποψήφιων χρηστών. Το γραφικό περιβάλλον, τα κουμπιά και τα υπόλοιπα στοιχεία διάδρασης, πρέπει να είναι σαφή, και η εμφάνισή τους είναι αυτοεπεξηγούμενη (self-descriptive), δηλαδή να γίνεται εύκολα αντιληπτή η σημασία τους.

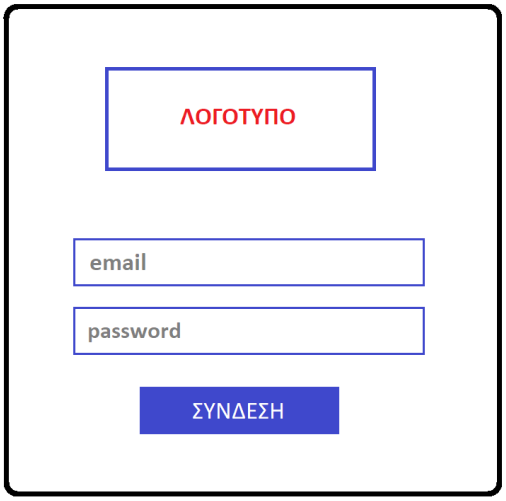
Με βάση τα παραπάνω, σχεδιάστηκαν οι παρακάτω οθόνες:

1. Αρχική οθόνη

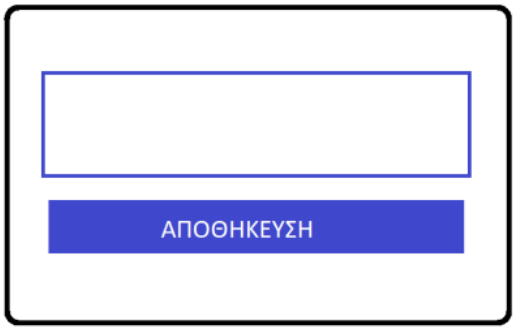


2. Οθόνη εγγραφής νέου χρήστη (Register)

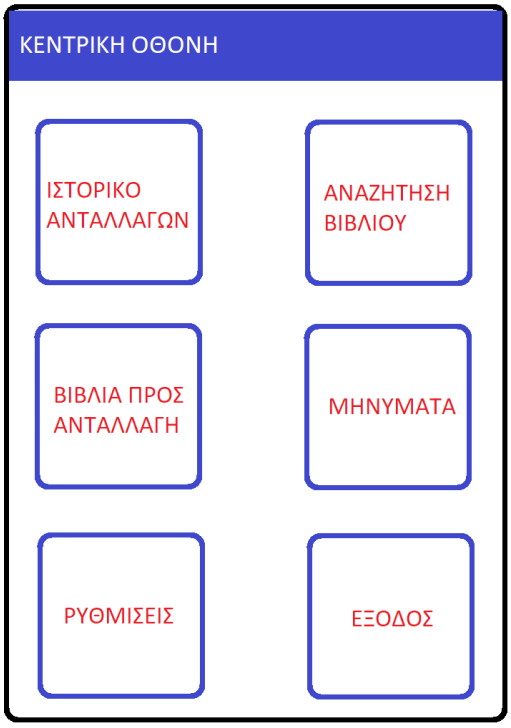
3. Οθόνη σύνδεσης χρήστη για είσοδο στην εφαρμογή (Login)



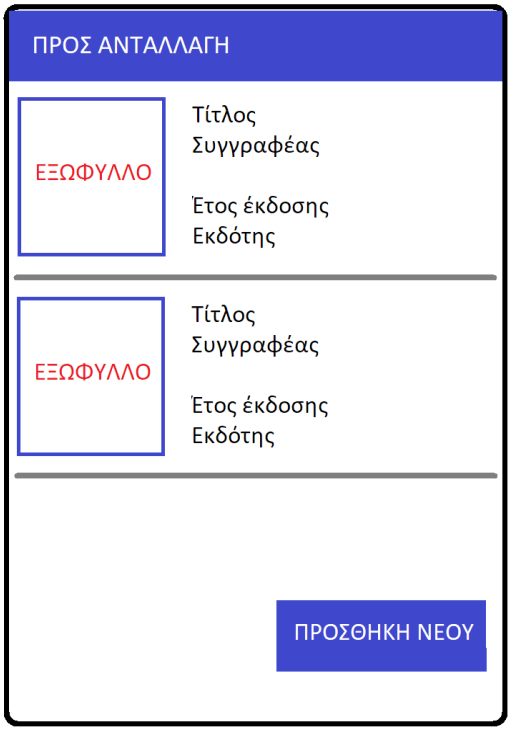
4. Οθόνη επιλογής username του χρήστη



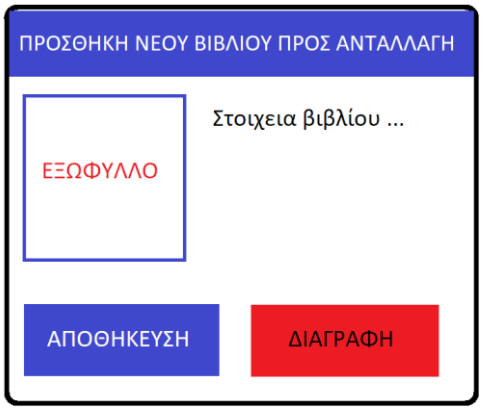
5. Κεντρική οθόνη



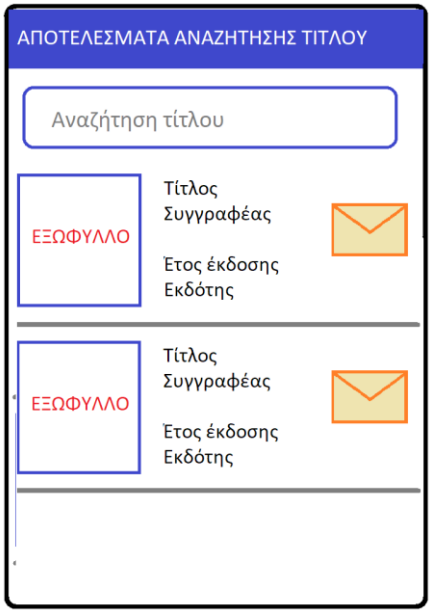
6. Οθόνη βιβλίων του χρήστη προς ανταλλαγή



7. Οθόνη προσθήκης - τροποποίησης ενός βιβλίου προς ανταλλαγή

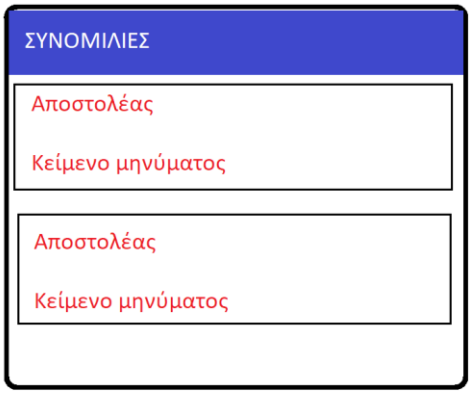


8. Οθόνη αναζήτησης βιβλίων άλλων χρηστών

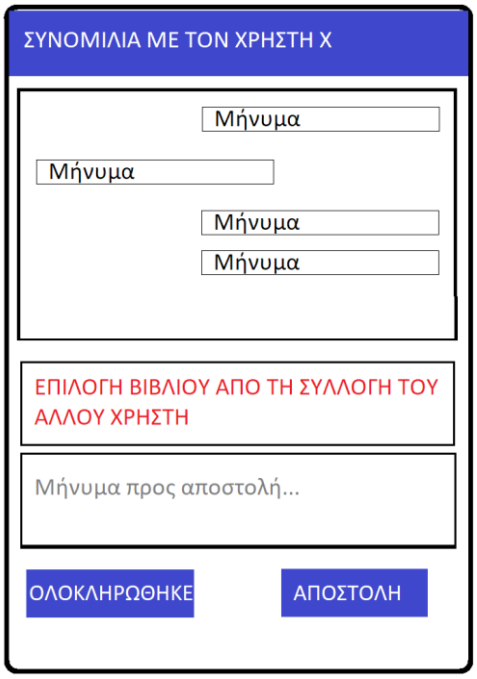


9. Οθόνη εκδήλωσης ενδιαφέροντος για ανταλλαγή βιβλίου

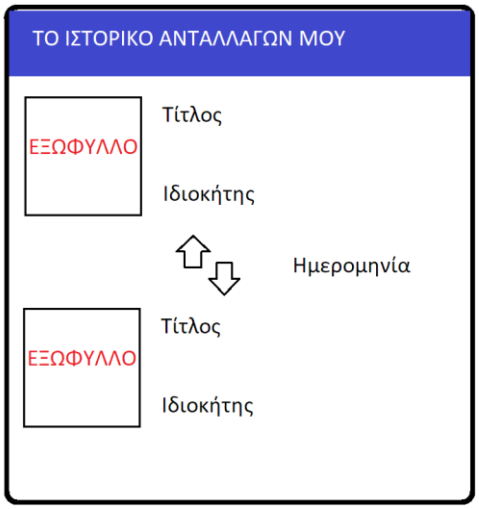
10. Οθόνη προβολής συνομιλιών με τους υπόλοιπους χρήστες της εφαρμογής



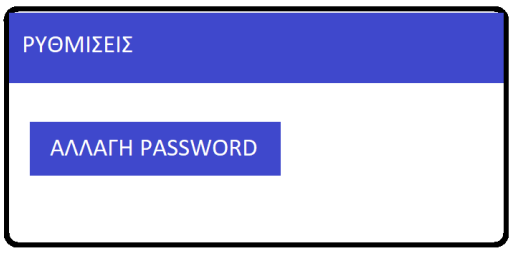
11. Οθόνη συνομιλίας - συμφωνίας βιβλίων προς ανταλλαγή



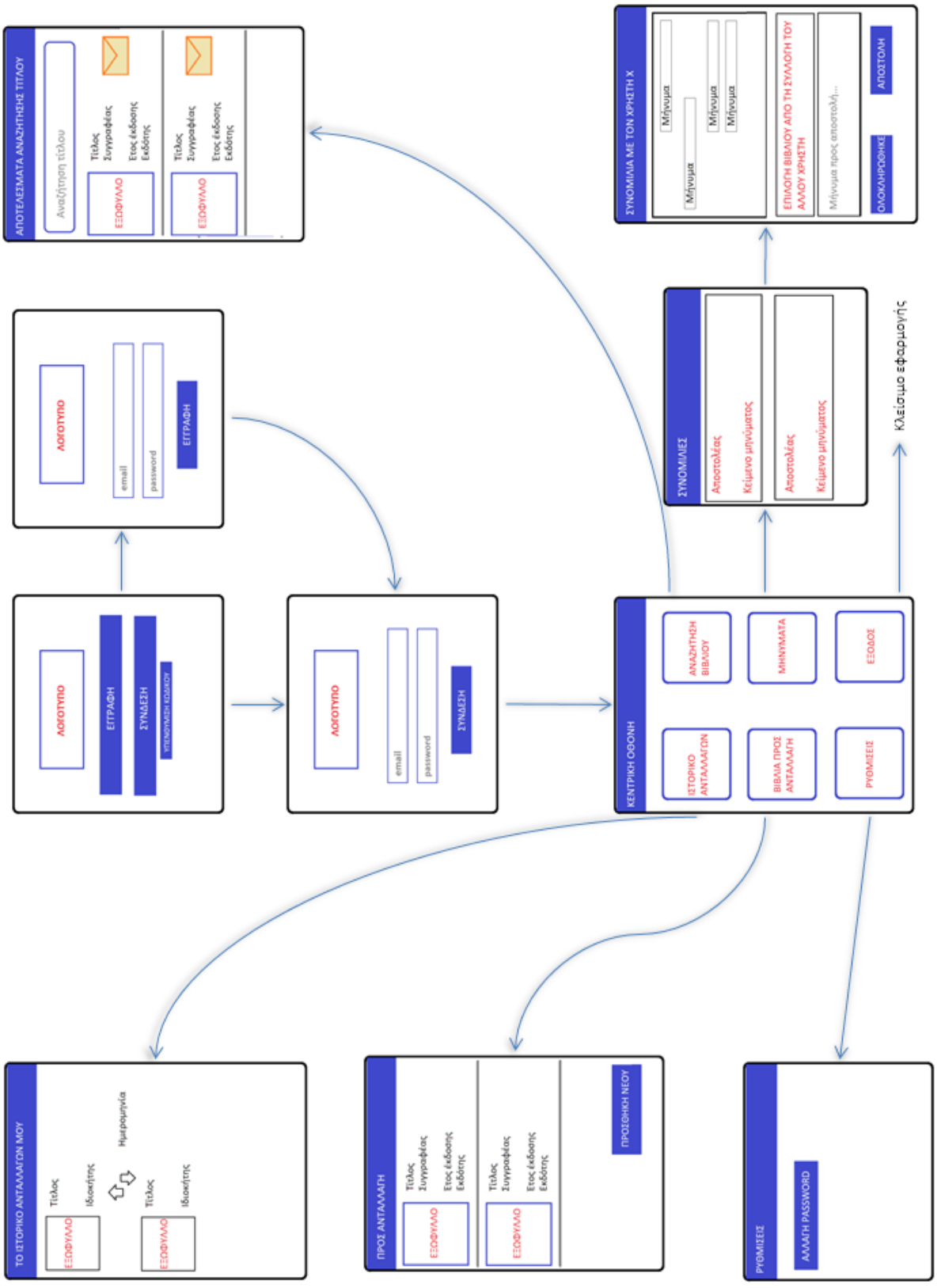
12. Οθόνη ιστορικού ανταλλαγών βιβλίων



13. Οθόνη ρυθμίσεων



Στο Σχήμα 7 παρουσιάζεται το αντίστοιχο διάγραμμα μεταβάσεων μεταξύ των βασικότερων οθονών, το οποίο σχεδιάστηκε με αφετηρία την Αρχική οθόνη. Ακολουθεί κατά περίπτωση η εγγραφή νέου χρήστη ή η είσοδος υπάρχοντος χρήστη. Επόμενη οθόνη είναι η Κεντρική οθόνη, από την οποία είναι προσβάσιμες όλες οι υπόλοιπες οθόνες της εφαρμογής μέσω των αντίστοιχων κουμπιών, σε μια διάταξη τύπου αστέρα. Όλες εκείνες οι οθόνες είναι τελικές, δηλαδή ο χρήστης από αυτές μπορεί να μεταβεί μόνο προς τα πίσω, στην Κεντρική οθόνη.



Σχήμα 7. Διάγραμμα μετάβασης οθονών

Κατά την εκκίνηση της εφαρμογής, ο χρήστης επιλέγει ανάμεσα στην είσοδο ή την εγγραφή. Εάν δεν είναι εγγεγραμμένος, τότε μετά την ολοκλήρωση της εγγραφής του, οδηγείται στην οθόνη εισόδου. Αφού συνδεθεί επιτυχώς, μεταβαίνει στην κεντρική οθόνη.

Για να προσθέσει ένα νέο βιβλίο προς ανταλλαγή στη συλλογή του, πατά το κουμπί «ΒΙΒΛΙΑ ΠΡΟΣ ΑΝΤΑΛΛΑΓΗ» και στη συνέχεια «ΠΡΟΣΘΗΚΗ ΝΕΟΥ». Στη συνέχεια επιστρέφει στην Κεντρική οθόνη με το κουμπί «BACK» της συσκευής του. Για να αναζητήσει έναν τίτλο βιβλίου που ενδιαφέρεται να αποκτήσει, πατά το κουμπί «ΑΝΑΖΗΤΗΣΗ ΒΙΒΛΙΟΥ». Εάν το βιβλίο αυτό βρεθεί ανάμεσα στα βιβλία προς ανταλλαγή των υπόλοιπων χρηστών, τότε πατά το εικονίδιο με το φάκελο αλληλογραφίας που βρίσκεται στο πλάι του αντίστοιχου βιβλίου, για να εκκινήσει μια διαδικασία ανταλλαγής εκφράζοντας το ενδιαφέρον του στον ιδιοκτήτη μέσω ενός μηνύματος. Με αυτό τον τρόπο δημιουργείται μια νέα συνομιλία η οποία αφορά μόνο το συγκεκριμένο βιβλίο. Δηλαδή, κάθε συνομιλία συνδέεται νοηματικά με την πρόθεση ανταλλαγής ενός βιβλίου. Ο ιδιοκτήτης του βιβλίου μπορεί να απαντήσει στο μήνυμα και να επιλέξει ένα βιβλίο – αντάλλαγμα, από τα βιβλία που διαθέτει ο πρώτος χρήστης. Τότε εκείνος, απαντά θετικά ή αρνητικά για το βιβλίο που ζητήθηκε.

Η διαδικασία της ανταλλαγής ολοκληρώνεται όταν έχουν συμφωνήσει και οι δύο χρήστες για τα βιβλία που θα ανταλλαχθούν. Τότε ο χρήστης που εκκίνησε την ανταλλαγή, πατά το κουμπί «ΟΛΟΚΛΗΡΩΘΗΚΕ». Έτσι δημιουργείται μια νέα εγγραφή στο ιστορικό ανταλλαγών του κάθε χρήστη, και τα βιβλία που ανταλλάχθηκαν, δεν εμφανίζονται πλέον στη λίστα με τα διαθέσιμα προς ανταλλαγή καθενός.

Από την Κεντρική οθόνη, πατώντας το κουμπί «ΡΥΘΜΙΣΕΙΣ» ο χρήστης έχει πρόσβαση στην οθόνη ρυθμίσεων, όπου μπορεί να αλλάξει τον κωδικό πρόσβασής του στην εφαρμογή. Το κουμπί «ΙΣΤΟΡΙΚΟ ΑΝΤΑΛΛΑΓΩΝ» εμφανίζει τα ζεύγη βιβλίων που έχει ανταλλάξει ο χρήστης καθώς και την ημερομηνία της ανταλλαγής. Τέλος, το κουμπί «ΕΞΟΔΟΣ» εμφανίζει ένα παράθυρο διαλόγου επιβεβαίωσης, για την αποσύνδεση του χρήστη.

2.4 Αντικειμενοστρεφής σχεδίαση

Για το επόμενο βήμα της σχεδίασης, αφού λήφθηκαν υπ'όψιν οι απαιτήσεις του συστήματος και οι μεταβάσεις των οθονών, καθορίστηκαν οι κλάσεις του λογισμικού με τις μεθόδους τους και οι μεταξύ τους συσχετίσεις. Στη συνέχεια δημιουργήθηκαν τα κατάλληλα διαγράμματα των κλάσεων σε Unified Modeling Language (UML).

Οι κλάσεις της εφαρμογής είναι οι εξής:

1. User. Η κλάση που περιγράφει ένα αντικείμενο τύπου χρήστη. Αποτελείται από δύο χαρακτηριστικά (attributes): username και email.

User
username
email

2. Book. Η κλάση που περιγράφει ένα αντικείμενο τύπου βιβλίο. Αποτελείται από ένα μοναδικό αναγνωριστικό (ID), το ID και το username του χρήστη-ιδιοκτήτη, τον τίτλο του βιβλίου,

συγγραφέα, το έτος έκδοσης, τον εκδοτικό οίκο, την κατάσταση του βιβλίου, το εξώφυλλό του. Τέλος, περιέχει το ID του βιβλίου με το οποίο έχει (τυχόν) ανταλλαχθεί, και την αντίστοιχη ημερομηνία.

Book
bookID
ownerID
ownerUsername
title
author
year
publisher
condition
cover
swappedWithBookID
swapTimestamp

3. Message. Η κλάση που περιγράφει ένα αντικείμενο τύπου μήνυμα. Αποτελείται από ένα ID, ημερομηνία αποστολής, το κείμενο του μηνύματος, ID και username του χρήστη-αποστολέα, ID του χρήστη-παραλήπτη, την μεταβλητή read η οποία υποδηλώνει εάν το μήνυμα έχει αναγνωσθεί από τον παραλήπτη, και τέλος το ID του αιτήματος της ανταλλαγής που αφορά.

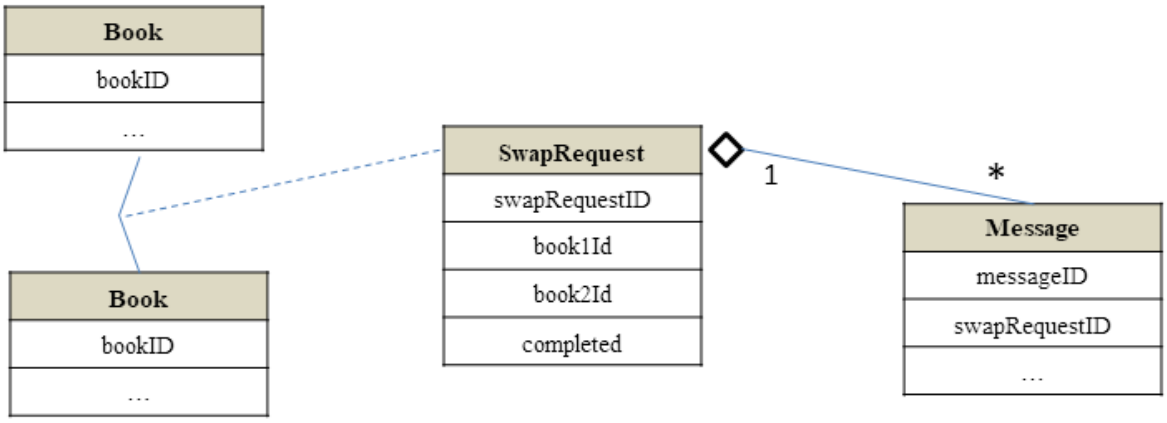
Message
messageID
timestamp
text
senderID
senderUsername
receiverID
read
swapRequestID

4. SwapRequest. Η κλάση που περιγράφει ένα αίτημα ανταλλαγής. Δηλαδή τη διασύνδεση μεταξύ των δύο αντικειμένων τύπου Book που συμμετέχουν σε μία ανταλλαγή. Περιλαμβάνει

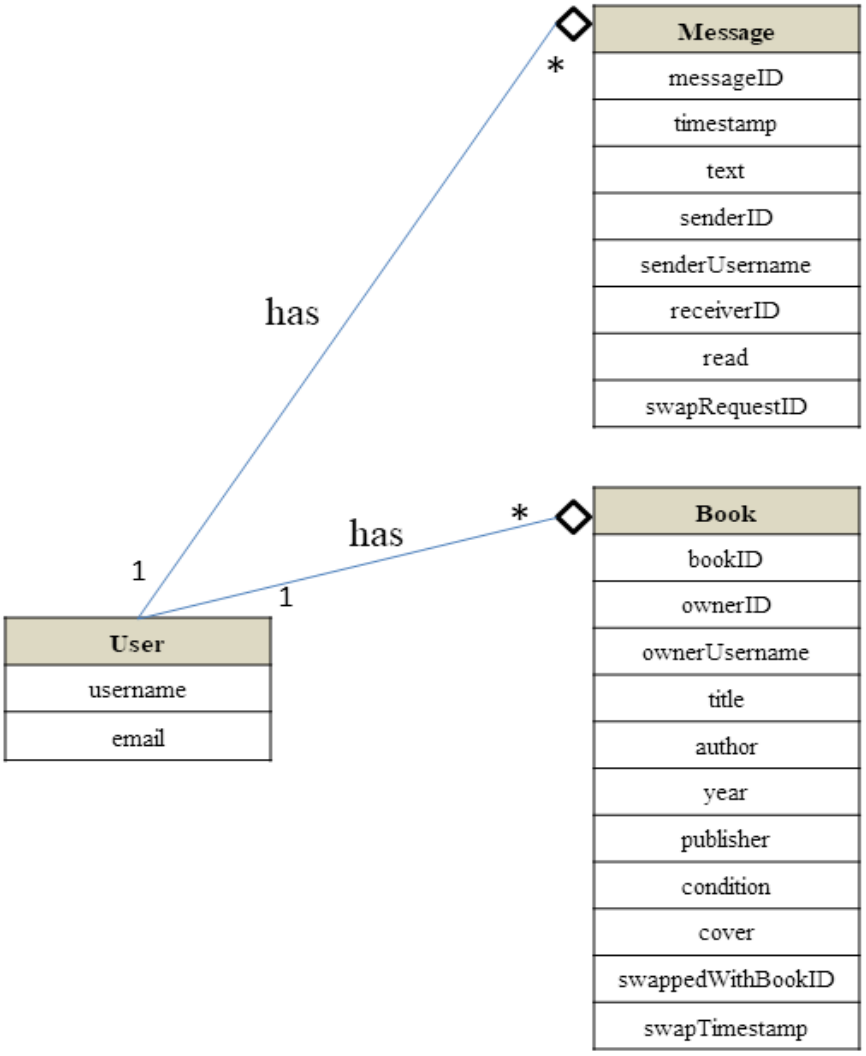
το ID του αιτήματος ανταλλαγής (swapRequestID), τα ID των βιβλίων που θα ανταλλαχθούν, καθώς και την πληροφορία για το εάν έχει ολοκληρωθεί αυτό το αίτημα ανταλλαγής.

SwapRequest
swapRequestID
book1Id
book2Id
Completed

Για να διευκολυνθεί η κατανόηση της διαχείρισης των αντικειμένων των κλάσεων και των συμπεριφορών τους, ακολουθούν στα Σχήματα 8 και 9, τα δύο διαγράμματα Οντοτήτων-Συσχετίσεων (Entity-Relationship (E-R)) UML που δημιουργήθηκαν ώστε να απεικονιστούν οι σχέσεις μεταξύ τους.



Σχήμα 8. UML διάγραμμα της σχέσης του SwapRequest με τα Book και Message



Σχήμα 9: UML διάγραμμα συσχέτισης των κλάσεων User, Book και Message

2.5 Επίλογος

Στο παρόν κεφάλαιο καθορίστηκαν οι απαιτήσεις του συστήματος με βάση τις προδιαγραφές της εφαρμογής. Έπειτα σχεδιάσθηκαν οι απαραίτητες οθόνες που θα την απαρτίζουν ώστε να καλύπτεται η λειτουργικότητά της. Τέλος, δημιουργήθηκαν οι αναγκαίες κλάσεις αντικειμένων και παρουσιάστηκαν οι μεταξύ τους νοηματικές διασυνδέσεις.

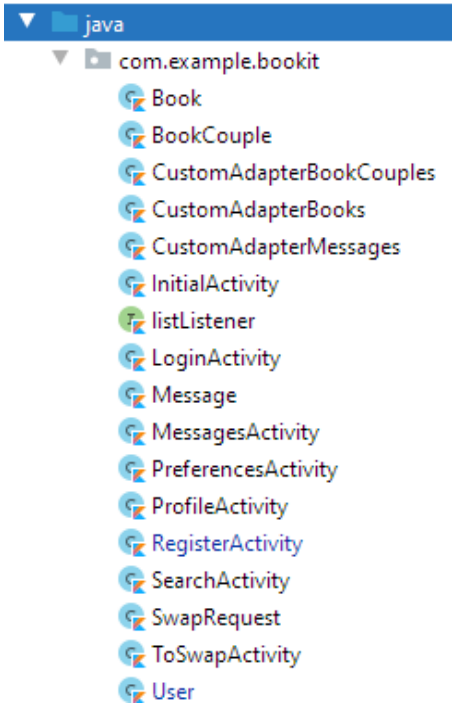
Κεφάλαιο 3ο: Δημιουργία του Project στο Android Studio

3.1 Εισαγωγή

Η εφαρμογή υλοποιήθηκε με το λογισμικό Android Studio 4.1.1. Τα περιεχόμενα του project είναι οι Κλάσεις γραμμένες σε γλώσσα Kotlin, Dialogs, Drawables τύπου PNG και Vector, Layouts γραμμένα σε γλώσσα XML, Colors σε κωδικοποίηση HTML, Strings σε Ελληνικά και Αγγλικά, και τέλος τα Styles. Στο κεφάλαιο αυτό, τα παραπάνω συστατικά θα περιγραφούν αναλυτικά.

3.2 Κλάσεις

Οι Κλάσεις της εφαρμογής φαίνονται στο στιγμιότυπο του Σχήματος 10. Ανήκουν σε έναν από τους εξής τύπους: data class, AppCompatActivity() και BaseAdapter().



Σχήμα 10: Οι Κλάσεις της εφαρμογής

Οι data classes χρησιμοποιήθηκαν για την δημιουργία αντικειμένων στις ιδιότητες (properties) των οποίων θα αποθηκεύονται τιμές. Μια data class μπορεί να περιέχει λειτουργικότητα μέσω μεθόδων, όμως συνήθως αυτές είναι από καμία έως ελάχιστες, καθώς δεν είναι αυτός ο σκοπός τους. Έπειτα αυτά θα μπορούν να τροποποιηθούν, να αναγνωστούν, να συγκριθούν, να αντιγραφούν ή να διαγραφούν. Μια κλάση ορίζεται ως data class μέσω της δεσμευμένης λέξης data που προηγείται του όρου class ως πρόθεμα στη δήλωσή της. Η δημιουργία μιας data class συνεπάγεται αυτόματη δημιουργία των setter και getter μεθόδων για τις ιδιότητές της, όπως επίσης και των εξής: toString(), equals(), copy(), hashCode(), component() από τον compiler. Εάν η κλάση δεν είχε οριστεί ως data, η κλήση της toString() θα εμφάνιζε τη θέση μνήμης του εκάστοτε αντικειμένου. Όμως με το prefix data, η toString() δημιουργείται αυτομάτως κατάλληλα, έτσι ώστε να εμφανίζει την αναμενόμενα επιθυμητή μορφή των δεδομένων. Μια άλλη χρήσιμη μέθοδος που δημιουργείται είναι η equals(), βασιζόμενη στους τύπους των ιδιοτήτων της κλάσης. Έτσι η σύγκριση δύο αντικειμένων μέσω του

τελεστή = = προκαλεί την κλήση της equals() η οποία επιστρέφει true εάν τα δύο αντικείμενα είναι του ίδιου τύπου και έχουν ίσες τιμές σε όλες τις ιδιότητές τους. Εάν κάποια στιγμή γίνουν αλλαγές στις ιδιότητες της κλάσης, όπως αλλαγές τύπων, ή προσθήκη νέων ιδιοτήτων, οι παραπάνω μέθοδοι δε χρειάζονται τροποποίηση καθώς ενημερώνονται αυτόματα. Τα παραπάνω πλεονεκτήματα πέραν της εξοικονόμησης χρόνου που παρείχαν σε σύγκριση με τη γραφή τους σε γλώσσα Java, συνεισφέρουν στην αναγνωσιμότητα (readability) και τη συνέπειά του (consistency) του κώδικα της εφαρμογής, άρα και στην ευκολότερη συντήρησή του (maintainability), κάτι που είναι ουσιαστικής σημασίας, αφού συνήθως σε μια εφαρμογή συχνότερα διαβάζεται ο κώδικάς της, παρά γράφεται νέος [12].

3.2.1 Book, Message, SwapRequest, User

Ακολουθούν στο Σχήμα 11 οι data κλάσεις των αντικειμένων που θα μπορούν να δημιουργηθούν και να αποθηκευτούν στη βάση δεδομένων, ως documents αντίστοιχων ομώνυμων collections στο Firestore.

```
data class SwapRequest (
    var swapRequestID : String,
    var book1Id : String,
    var book2Id : String,
    var completed : Boolean
)
```

```
data class User(
    val username: String,
    var email: String
)
```

```
data class Book(
    var bookID: String,
    var ownerID: String?,
    var ownerUsername: String?,
    var title: String,
    var author: String,
    var year: String,
    var publisher: String,
    var condition: String,
    var cover: Blob?,
    var swappedWithBookID: String,
    var swapTimestamp : Long
)
```

```
data class Message(
    var messageID: String,
    var timestamp: Long,
    var text: String,
    val senderID: String?,
    val senderUsername: String?,
    val receiverID: String?,
    var read: Boolean,
    var swapRequestID: String
)
```

Σχήμα 11: Ορισμός των data κλάσεων της εφαρμογής

3.2.2 Activities

1. InitialActivity. Αυτό είναι το πρώτο activity που εμφανίζεται κατά το άνοιγμα της εφαρμογής. Περιέχει τρία onClickListener τα οποία ανταποκρίνονται στην επιθυμία του χρήστη να δημιουργήσει νέο λογαριασμό, να συνδεθεί με υπάρχον, ή να αλλάξει password.
2. RegisterActivity. Ο χρήστης μπορεί να δημιουργήσει ένα νέο λογαριασμό, γράφοντας μια διεύθυνση email η οποία δεν χρησιμοποιείται ήδη από άλλο χρήστη, και επιλέγοντας ένα password. Περιέχει έναν onClickListener ο οποίος όταν κληθεί πυροδοτεί την εισαγωγή του νέου χρήστη στην βάση δεδομένων, αφού πρώτα ελέγξει ότι το email που δόθηκε δεν

αντιστοιχεί σε υπάρχον λογαριασμό. Όταν η εγγραφή επιτυγχάνει, στέλνεται email επιβεβαίωσης στο χρήστη. Έπειτα ξεκινά το LoginActivity μέσω Intent το οποίο περιέχει τα strings του email και του password. Αυτά εμφανίζονται προσυμπληρωμένα στα αντίστοιχα πεδία του LoginActivity, ενώ το τρέχον καταστρέφεται με την εντολή finish().

3. LoginActivity. Εδώ πραγματοποιείται ο έλεγχος εισόδου του χρήστη μέσω της signInWithEmailAndPassword() μεθόδου του Firebase. Σε περίπτωση επιτυχίας, γίνεται έναρξη του ProfileActivity, δηλαδή της κεντρικής οθόνης.

4. ProfileActivity. Πρόκειται για την κεντρική οθόνη της εφαρμογής, στην οποία βρίσκονται έξι κουμπιά, μέσω των οποίων ο χρήστης μπορεί να πλοηγηθεί στις λειτουργίες της. Την πρώτη φορά που ένας χρήστης συνδέεται στο λογαριασμό του, πρωτού να είναι σε θέση να χρησιμοποιήσει τα κουμπιά της εφαρμογής, θα πρέπει να επιλέξει ένα username με το οποίο θα είναι γνωστός στους υπόλοιπους χρήστες.

5. SearchActivity. Με την εκκίνηση του activity φορτώνονται από τη βάση δεδομένων οι τίτλοι των βιβλίων που διαθέτονται από τους χρήστες προς ανταλλαγή, και εισάγονται στον adapter του AutoCompleteTextView της Αναζήτησης. Έτσι καθώς ο χρήστης πληκτρολογεί, εμφανίζονται προτάσεις για τον τίτλο που αναζητά. Όταν επιλέξει κάποιον από τους προτεινόμενους, τότε φορτώνονται από τη βάση δεδομένων σε έναν custom adapter τα βιβλία (αντικείμενα τύπου Book) που φέρουν το συγκεκριμένο τίτλο, και έπειτα εμφανίζονται μέσα σε ένα ListView. Κάθε γραμμή (row) της λίστας περιλαμβάνει το εξώφυλλο και τις πληροφορίες ενός βιβλίου, όπως επίσης και ένα ImageButton με εικονίδιο φακέλου μηνύματος. Όταν δεχτεί ένα click event, εμφανίζεται το dialogSendMessage, μέσω του οποίου δημιουργείται ένα νέο αίτημα ανταλλαγής (ένα νέο αντικείμενο SwapRequest).

6. ToSwapActivity. Σ’ αυτό το activity γίνεται η διαχείριση των βιβλίων που ο χρήστης διαθέτει προς ανταλλαγή. Αρχικά φορτώνονται από τη βάση δεδομένων τα βιβλία των οποίων το ownerID είναι ίδιο με το currentUid και ταυτόχρονα έχουν κενή τιμή στο πεδίο swappedWithBookID . Αυτά είναι τα βιβλία που ο χρήστης έχει καταχωρήσει προς ανταλλαγή όμως δεν έχουν ανταλλαχθεί. Εισάγονται λοιπόν στον custom adapter CustomAdapterBooks και εμφανίζονται στην λίστα to_swap_list. Κάθε row της λίστας περιέχει το εξώφυλλο και τις πληροφορίες ενός βιβλίου. Όταν δεχθεί σε κάποιο σημείο εντός του ένα click event, εμφανίζεται το dialog dialogEditBook το οποίο επιτρέπει την επεξεργασία των στοιχείων του αντίστοιχου βιβλίου, ή τη διαγραφή του από τη βάση δεδομένων. Η προσθήκη ενός νέου βιβλίου γίνεται μέσω click event στο addNewBook_button, το οποίο είναι ένα ImageButton με εικονίδιο το σύμβολο της πρόσθεσης.

7. MessagesActivity. Παρουσιάζονται όλες οι συνομιλίες του χρήστη, ταξινομημένες με βάση τη χρονολογική τους σειρά. Κάθε μια συνομιλία αφορά ένα αίτημα ανταλλαγής. Αρχικά φορτώνονται από τη βάση δεδομένων τα μηνύματα στα οποία το receiverID είναι ίδιο με το currentUid, δηλαδή τα μηνύματα στα οποία ο χρήστης εμφανίζεται ως παραλήπτης. Τα μηνύματα κατηγοριοποιούνται με βάση το swapRequestID, δηλαδή το αίτημα ανταλλαγής το οποίο αφορούν. Από όλα τα μηνύματα που αφορούν ένα συγκεκριμένο αίτημα ανταλλαγής, εντοπίζεται το πιο πρόσφατο με βάση το πεδίο timestamp, και τοποθετείται σε ένα ArrayList, μαζί με τα υπόλοιπα πιο πρόσφατα μηνύματα των υπόλοιπων αιτημάτων ανταλλαγής. Τέλος αυτά ταξινομούνται χρονολογικά, και φορτώνονται στον adapter CustomAdapterMessages ώστε να εμφανιστούν στο messages_list ListView. Όταν ένα list item (μία συνομιλία) δεχθεί

click event, εμφανίζεται το dialog dialogConversation στο οποίο παρουσιάζεται η συγκεκριμένη συνομιλία, με τη δυνατότητα αποστολής νέων μηνυμάτων.

8. PreferencesActivity. Αφορά ρυθμίσεις που μπορεί να επιλέξει ο χρήστης. Περιέχει δύο Buttons. Μέσω ενός click event στο change_password_button εμφανίζεται ένα dialog στο οποίο ο χρήστης επιβεβαιώνει πως επιθυμεί να αλλάξει το προσωπικό του password. Η διαδικασία αλλαγής του, γίνεται μέσω κλήσης της μεθόδου sendPasswordResetEmail() του Firebase, με όρισμα το email του χρήστη. Το δεύτερο κουμπί του activity, about_button, εμφανίζει πληροφορίες σχετικά με την εφαρμογή όπως την τρέχουσα έκδοσή της κ.α.

3.2.3 Dialogs

1. dialogAddBook. Το dialog αυτό εμφανίζεται μέσα στο ToSwapActivity όταν γίνει κλικ στο addNewBook_button. Χρησιμοποιείται από το χρήστη για να καταχωρήσει ένα βιβλίο προς ανταλλαγή. Περιέχει τέσσερα EditText στα οποία συμπληρώνει τις βασικές πληροφορίες για κάθε βιβλίο, οι οποίες αποθηκεύονται ως string τιμές. Το dialog περιέχει επίσης ένα spinner για την επιλογή κατάστασης του βιβλίου (π.χ. σαν καινούργιο), και τέλος, ένα ImageButton. Όταν αυτό δεχτεί click event, καλείται με Intent το Image gallery της συσκευής, ώστε ο χρήστης να επιλέξει μια εικόνα, ως εξώφυλλο του νέου βιβλίου προς καταχώρηση. Εάν ο χρήστης πατήσει το κουμπί «OK» και εφόσον όλα τα στοιχεία που αφορούν το βιβλίο είναι συμπληρωμένα, τότε δημιουργείται ένα νέο αντικείμενο τύπου Book, που οι τιμές του αντιστοιχούν στις τιμές που δόθηκαν. Αυτό αποθηκεύεται ως νέο document στο collection books της βάσης δεδομένων.
2. dialogEditBook. Το dialog αυτό εμφανίζεται μέσα στο ToSwapActivity όταν γίνει κλικ σε ένα row του ListView to_swap_list που περιέχει τα βιβλία προς ανταλλαγή. Μέσω αυτού είναι δυνατή η τροποποίηση των πληροφοριών ενός καταχωρημένου προς ανταλλαγή βιβλίου. Το επιλεγμένο βιβλίο (αντικείμενο τύπου Book) προσπελάσσεται μέσα από το bookArrayList ArrayList που περιέχει όλα τα Book του χρήστη, κάνοντας χρήση του integer position της θέσης του row που επιλέχθηκε από το ListView to_swap_list. Οι τιμές των πεδίων αυτού του book αντικειμένου, εισάγονται στα αντίστοιχα EditText, Spinner και ImageButton. Το layout που χρησιμοποιείται είναι το ίδιο που χρησιμοποιεί και το dialogAddBook, με μόνη διαφορά την ορατότητα αυτή τη φορά του delete_button ImageButton, ώστε ο χρήστης να μπορεί να διαγράψει το εκάστοτε βιβλίο από τα διαθέσιμα προς ανταλλαγή. Αυτό προκαλεί διαγραφή του document με το αντίστοιχο bookID από το collection books της βάση δεδομένων.
3. dialogConversation. Το dialog αυτό εμφανίζεται μέσα στο MessagesActivity, όταν γίνει κλικ σε ένα row του ListView messages_list που περιέχει τις συνομιλίες. Στον κώδικά του υλοποιούνται οι παρακάτω λειτουργίες:

    - Εμφάνιση όλης της συνομιλίας μεταξύ δύο χρηστών, δηλαδή του κειμένου των Message αντικειμένων που αφορούν το εκάστοτε αίτημα ανταλλαγής.
    - Δυνατότητα σύνταξης νέων μηνυμάτων μέσα σε EditText και έπειτα και αποστολή τους. Τα νέα μηνύματα θα εμφανίζονται και στους δύο συνομιλητές σε πραγματικό χρόνο, με τη βοήθεια ενός SnaphotListener του Firestore.
    - Εμφάνιση του βιβλίου για το οποίο δημιουργήθηκε το αίτημα ανταλλαγής.

- Δίνεται η δυνατότητα στο δεύτερο χρήστη, να επιλέξει ένα από τα βιβλία του πρώτου ως αντάλλαγμα. Η επιλογή αυτή πραγματοποιείται μέσω ενός Spinner. Μόλις επιλεχθεί ένα βιβλίο, το bookID του συμπληρώνεται στο book2Id πεδίο του SwapRequest που αφορά τη συγκεκριμένη ανταλλαγή.
- Το βιβλίο που ο δεύτερος χρήστης διάλεξε ως αντάλλαγμα, εμφανίζεται στον πρώτο σε πραγματικό χρόνο. Εκείνος με τη σειρά του έχει τη δυνατότητα να συμφωνήσει ή να διαφωνήσει. Σε περίπτωση διαφωνίας, ο δεύτερος χρήστης καλείται να επιλέξει εκ νέου άλλο βιβλίο από το Spinner και να το εγκρίνει ο πρώτος. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να επιτευχθεί από κοινού συμφωνία.
- Όταν ο πρώτος χρήστης συμφωνήσει, πρέπει να κάνει κλικ στο Button «ΟΛΟΚΛΗΡΩΘΗΚΕ». Τότε σε κάθε book που συμφωνήθηκε, ορίζονται οι αντίστοιχες τιμές στα πεδία swappedWithBookID και swapTimeStamp.

Πιο συγκεκριμένα:

Αρχικά φορτώνονται από το collection messages της βάσης δεδομένων, όσα document έχουν τιμή στο πεδίο swapRequestID, ίση με την τιμή του swapRequestID πεδίου του μηνύματος που εμφανιζόταν στο list row (συνομιλία) που επιλέχθηκε. Κάθε ένα από αυτά τα document μετατρέπεται σε ένα αντικείμενο τύπου Message, και έπειτα προστίθεται στο convMessagesArrayList. Μέσω του CustomAdapterMessages αυτά τα μηνύματα εμφανίζονται στο conversation_list ListView, με στοίχιση στα δεξιά για όσα προέρχονται από τον τρέχον χρήστη και στα αριστερά για εκείνα του συνομιλητή του. Η σειρά εμφάνισής τους καθορίζεται μέσω χρονολογικής ταξινόμησης, βάσει της τιμής του πεδίου timestamp. Καθώς κάθε Message περιέχει το πεδίο SwapRequestID ως αναγνωριστικό της εκάστοτε ανταλλαγής, με βάση αυτό, γίνεται ερώτημα και φόρτωση από τη βάση δεδομένων, των βιβλίων που συμμετέχουν στη συγκεκριμένη ανταλλαγή. Το πρώτο βιβλίο (book1title) του αιτήματος ανταλλαγής είναι πάντα συμπληρωμένο, ενώ το δεύτερο (book2title) συμπληρώνεται όταν ο δεύτερος χρήστης επιλέξει ένα βιβλίο από το Spinner . Τότε δημιουργείται ταυτόχρονα ένα νέο Message με περιεχόμενο κειμένου «Ζητήθηκε βιβλίο», ώστε να ενημερωθεί ο πρώτος χρήστης. Οι τίτλοι των δύο βιβλίων εμφανίζονται στα αντίστοιχα book1title και book2title TextView όντας ορατά σε όλη τη διάρκεια της συνομιλίας και στους δύο χρήστες.

Με την κλήση της μεθόδου addSnapshotListener() στο collection messages του Firestore, επιτυγχάνεται η εμφάνιση των νέων μηνυμάτων σε πραγματικό χρόνο (real-time), προσομοιώνοντας δηλαδή μία συνομιλία τύπου chat. Όταν ανιχνευθεί από αυτόν τον Listener η δημιουργία ενός νέου Message που αφορά την εκάστοτε ανταλλαγή (SwapRequestID), τότε γίνεται καθαρισμός του ListView, εκ νέου φόρτωση των Messsage της συνομιλίας και τέλος μέσω του CustomAdapterMessages εμφανίζονται στην conversation_list.

Η ενημέρωση του συστήματος για την ολοκλήρωση της ανταλλαγής, γίνεται από τον πρώτο χρήστη, μέσω του completed_button. Αυτό το Button είναι αόρατο για τον δεύτερο χρήστη (visibility = View.GONE). Το αν μια ανταλλαγή έχει ολοκληρωθεί, ελέγχεται από το Boolean πεδίο completed του SwapRequest που την αντιπροσωπεύει. Μετά την ολοκλήρωση της ανταλλαγής, δεν επιτρέπεται στον δεύτερο χρήστη επιλογή βιβλίου από το Spinner, αλλά η συνομιλία μπορεί να συνεχιστεί μέσω νέων μηνυμάτων.

4. dialogHistory. Το dialog αυτό εμφανίζεται μέσα στο ProfileActivity, όταν γίνει κλικ στο swap_history_button. Εμφανίζει τους τίτλους των βιβλίων από τα ζεύγη που ανταλλάχθηκαν,

με τα αντίστοιχα username των ιδιοκτητών τους. Είναι δεδομένο ότι όταν μια ανταλλαγή ολοκληρώνεται, η τρέχουσα ημερομηνία καταχωρείται στο πεδίο swapTimestamp των Book που την απαρτίζουν σε μορφή milliseconds, όπως επίσης ότι για κάθε Book καταχωρείται στο πεδίο του swappedWithBookID, το bookID του Book με το οποίο ανταλλάχθηκε. Έτσι είναι εφικτό ένα ερώτημα στη βάση δεδομένων για τα βιβλία στα οποία ο χρήστης είναι ιδιοκτήτης, και τα οποία έχουν ανταλλαχθεί. Δηλαδή φορτώνονται από το collection books όσα document έχουν ως ownerID το τρέχον currentUid, και τιμή διάφορη του κενού στο πεδίο swappedWithBookID. Για κάθε ένα από αυτά τα βιβλία (book1), γίνεται εκ νέου ερώτημα στο collection books ώστε να φορτωθεί το document (book2) που έχει id ίσο με την τιμή του αντίστοιχου swappedWithBookID πεδίου. Κάθε ζεύγος (book1,book2) αποθηκεύεται σε ένα βοηθητικό αντικείμενο τύπου BookCouple, μαζί με την ημερομηνία πραγματοποίησης της ανταλλαγής (swapTimestamp). Κάθε BookCouple που δημιουργείται, προστίθεται στο swappedBooksArrayList. Μόλις ολοκληρωθεί αυτή η διαδικασία για όλα τα ζεύγη βιβλίων που έχουν ανταλλαχθεί, το swappedBooksArrayList φορτώνεται στον custom adapter CustomAdapterBookCouples, ο οποίος εφαρμόζεται στο ListView του dialog.

5. dialogSendMessage. Το dialog αυτό εμφανίζεται μέσα στο SearchActivity, όταν γίνει κλικ στο send_message_button ενός row του ListView που περιέχει τα αποτελέσματα (βιβλία) που βρέθηκαν στην αναζήτηση. Δίνει στον χρήστη τη δυνατότητα να ξεκινήσει ένα νέο αίτημα ανταλλαγής, αποστέλλοντας ταυτόχρονα το πρώτο μήνυμα της νέας συνομιλίας. Πατώντας το Button «ΑΠΟΣΤΟΛΗ», δημιουργείται ένα νέο SwapRequest αντικείμενο, στου οποίου το πεδίο book1Id καταχωρείται ως τιμή το bookID του βιβλίου που ζητήθηκε. Επίσης δημιουργείται ένα νέο Message αντικείμενο, στου οποίου το πεδίο swapRequestID καταχωρείται ως τιμή το swapRequestID του SwapRequest που μόλις δημιουργήθηκε, ώστε να σημανθεί ποιο αίτημα ανταλλαγής αφορά αυτό μήνυμα. Στο πεδίο senderID καταχωρείται το uid του τρέχοντα συνδεδεμένου χρήστη, στο πεδίο receiverID καταχωρείται η τιμή του ownerID του Book που ζητήθηκε, και στο πεδίο text καταχωρείται το κείμενο του μηνύματος που πληκτρολόγησε ο αποστολέας. Τέλος, αυτό το νέο Message καταχωρείται ως νέο document στο collection messages της βάσης δεδομένων.

6. dialogCreateUsername. Το dialog αυτό εμφανίζεται μέσα στο ProfileActivity κατά την εκκίνησή του, μόνο στην περίπτωση που ο χρήστης δεν έχει επιλέξει username. Ζητείται τότε να πληκτρολογήσει ένα username στο addUsername EditText. Με το πάτημα του ok_button, γίνεται έλεγχος για το αν αυτό χρησιμοποιείται ήδη από άλλο χρήστη. Αυτός ο έλεγχος πραγματοποιείται μέσω ερωτήματος στο collection users της βάση δεδομένων, όπου αναζητείται η ύπαρξη User με το δοθέν username. Εάν βρεθεί User που το χρησιμοποιεί ήδη, τότε ο χρήστης καλείται και πάλι να πληκτρολογήσει ένα username, μέχρις ότου να είναι μοναδικό ανάμεσα στα υπάρχοντα.

7. dialogForgotPassword. Το dialog αυτό εμφανίζεται μέσα στο InitialActivity, όταν γίνει κλίκ στο clickable forgot_password_textview. Ζητείται τότε από το χρήστη να πληκτρολογήσει το email του λογαριασμού του στο email_to_send_reset_pas EditText. Με το πάτημα του ok_button, καλείται η μέθοδος sendPasswordResetEmail του Firebase, με όρισμα το email που έδωσε ο χρήστης. Σε αυτό έπειτα αποστέλλεται ένα μήνυμα με οδηγίες για την επιλογή νέου password.

3.3 Custom Adapters

Στην εφαρμογή χρησιμοποιούνται ListViews σε περιπτώσεις που χρειάζεται να εμφανιστούν πολλά δεδομένα του ίδιου τύπου. Πιο συγκεκριμένα, για την εμφάνιση των προς ανταλλαγή βιβλίων του χρήστη, των αποτελεσμάτων της αναζήτησης ενός τίτλου βιβλίου, των συνομιλιών του χρήστη, των μηνυμάτων μιας συνομιλίας, και τέλος για την εμφάνιση του ιστορικού ανταλλαγών του. Επειδή τα παραπάνω δεδομένα απαιτούν ένα σύνθετο τρόπο παρουσίασής τους, δεν επαρκεί η χρησιμοποίηση ενός απλού ArrayAdapter, αφού θα επέτρεπε μόνο την εμφάνιση ενός String. Για παράδειγμα, όταν σε μια λίστα πρέπει να εμφανιστούν βιβλία, χρειάζεται να παρουσιαστούν ταυτόχρονα αρκετά στοιχεία, όπως ο τίτλος, ο συγγραφέας, το εξώφυλλο κ.α., τα οποία πρέπει να βρίσκονται σε διαφορετικές θέσεις και με διαφορετική μορφοποίηση εντός του row της λίστας. Η δημιουργία λοιπόν ενός εξατομικευμένου (custom) ArrayAdapter με το αντίστοιχό του custom row xml αρχείο, αποτέλεσαν τη λύση για την επιθυμητή παρουσίαση των δεδομένων. Όλοι οι custom adapters υλοποιούν μια τεχνική recycle views για εξοικονόμηση μνήμης, η οποία θα αναλυθεί σε επόμενο κεφάλαιο.

Παρακάτω περιγράφονται οι custom ArrayAdapter που δημιουργήθηκαν για την εφαρμογή.

1. CustomAdapterBooks. Αυτός ο adapter δέχεται ως όρισμα ένα ArrayList με Books, μια Boolean μεταβλητή για την εμφάνιση ή όχι του ownerUsername ενός Book, μια Boolean μεταβλητή για την εμφάνιση ή όχι του ImageButton send_message_button, και τέλος ένα interface τύπου listListener για την υλοποίηση του onClickListener του send_message_button. Οι τιμές των πεδίων ενός Book τοποθετούνται στα κατάλληλα views του list_row_books.xml. Δηλαδή ο τίτλος του, ο συγγραφέας, το έτος, ο εκδότης, και ο ιδιοκτήτης, τοποθετούνται μέσα σε TextViews. Η τιμή του condition του Book διαβάζεται και αντιστοιχίζεται με το conditions array που περιέχεται μέσα στο Spinner, το οποίο και εμφανίζει την επιλεγμένη κατάσταση. Το εξώφυλλο (cover) του Book, μετατρέπεται από Blob σε Bitmap, και τοποθετείται στο cover_imageview. Μέσα στο δομητή (constructor) του CustomAdapterBooks, υλοποιείται η μέθοδος sendMessage του interface listListener. Αυτή καλείται όταν συμβεί ένα click event στο send_message_button. Δέχεται ως όρισμα το Book του επιλεγμένου row, και υλοποιεί τις διαδικασίες δημιουργίας ενός νέου SwapRequest και της αποστολής μηνύματος στον ιδιοκτήτη.
2. CustomAdapterBookCouples. Αυτός ο adapter δέχεται ως όρισμα ένα ArrayList με BookCouples, με σκοπό την εμφάνιση ενός ζεύγους βιβλίων που έχουν ανταλλαχθεί, και την αντίστοιχη ημερομηνία. Πιο συγκεκριμένα, οι τίτλοι των βιβλίων και τα username των ιδιοκτητών τοποθετούνται σε TextViews, τα covers τοποθετούνται σε ImageViews, και η ημερομηνία η οποία αφού μετατραπεί από milliseconds σε μορφοποίηση dd/MM/yyyy hh:mm, τοποθετείται στο TextView swap_date_textview του list_row_book_couples.xml.
3. CustomAdapterMessages. Αυτός ο adapter δέχεται ως όρισμα ένα ArrayList με Messages, μια Boolean μεταβλητή για την εμφάνιση ή όχι των username των συνομιλητών, και μια μεταβλητή String η οποία περιέχει το id του τρέχοντος χρήστη. Ο adapter αυτός χρησιμοποιείται σε δύο περιπτώσεις:
    - Παρουσίαση των μηνυμάτων μιας συνομιλίας. Γίνεται διαχωρισμός των Message σε αυτά που προέρχονται από τον τρέχοντα χρήστη, και σε εκείνα του συνομιλητή του, με βάση την τιμή του πεδίου senderID. Στα μεν, το κείμενο στοιχίζεται δεξιά εντός του row, ενώ στα δε, η στοίχιση του κειμένου γίνεται αριστερά. Επίσης εμφανίζονται γραμμένα με διαφορετικό χρώμα (TextColor).

- Παρουσίαση του πιο πρόσφατου μηνύματος κάθε συνομιλίας, εντός του ListView που περιέχει τις συνομιλίες στο MessagesActivity. Σε αυτή την περίπτωση εμφανίζεται το κείμενο του μηνύματος και το username του αποστολέα. Τα Message των οποίων η τιμή του πεδίου read είναι false, σημαίνονται με bold και italic γραμματοσειρά ώστε να ενδεικνύουν πως δεν έχουν αναγνωσθεί. Το timestamp ενός Message μετατρέπεται σε μορφή dd/MM/yyyy hh:mm και τοποθετείται στο TextView timestamp_textview του list_row_messages.xml.

3.4 Layouts

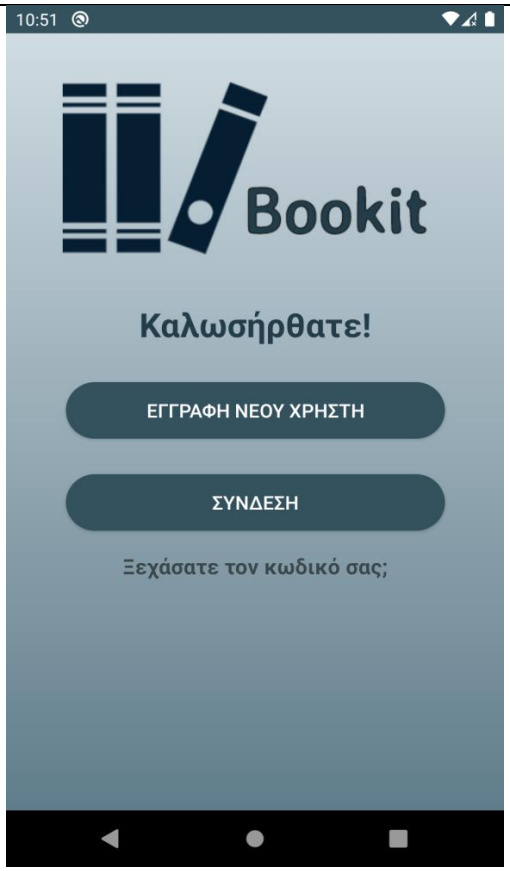
Τα layouts που χρησιμοποιήθηκαν για τα activities, dialogs, και custom adapters των λιστών, δημιουργήθηκαν στο Android Studio σε γλώσσα XML. Τα συστατικά διάδρασης της εφαρμογής περιλαμβάνουν: EditText, TextView, AutoCompleteTextView, ListView, Button, ImageButton, και Spinner. Όλα αυτά τοποθετήθηκαν μέσα σε ConstraintLayout και LinearLayout κατά περίπτωση. Ακολουθούν αναλυτικά όλα τα layouts με τις κλάσεις στις οποίες εφαρμόστηκαν.

Πίνακας 1. Χρήση layouts σε κλάσεις

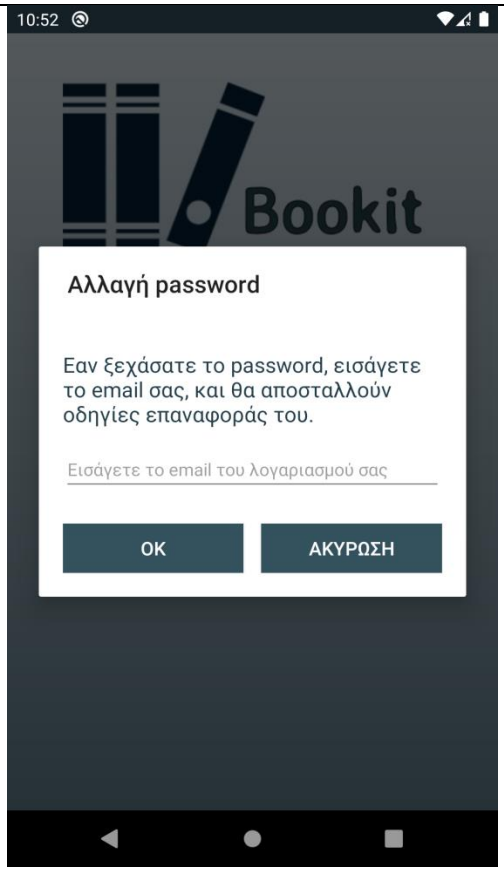
Όνομα layout	**Class που το χρησιμοποιεί**
activity_initial.xml	InitialActivity
activity_login.xml	LoginActivity
activity_messages.xml	MessagesActivity
activity_preferences.xml	PreferencesActivity
activity_profile.xml	ProfileActivity
activity_register.xml	RegisterActivity
activity_search	SearchActivity
activity_to_swap.xml	ToSwapActivity
dialog_add_new_book.xml	dialogAddBook, dialogEditBook
dialog_conversation.xml	dialogConversation
dialog_create_username.xml	dialogCreateUsername
dialog_history.xml	dialogHistory
dialog_reset_password	dialogForgotPassword
dialog_send_message	dialogSendMessage
list_row_books.xml	CustomAdapterBooks
list_row_book_couples.xml	CustomAdapterBookCouples
list_row_messages.xml	CustomAdapterMessages

Όπως γίνεται κατανοητό από τον παραπάνω πίνακα, δημιουργήθηκαν τριών ειδών layout: για Activities, Dialogs και rows των ListView (CustomAdapters).

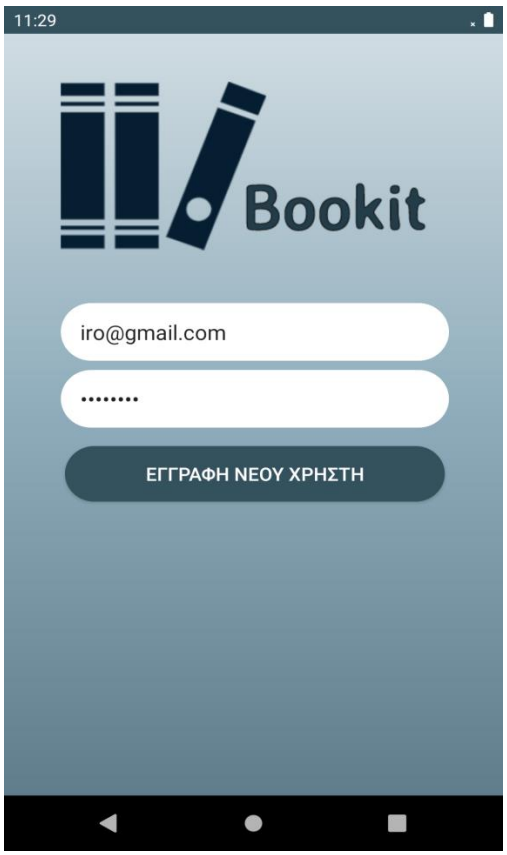
Ακολουθούν στιγμιότυπα των layout στα Σχήματα 12 - 27 :



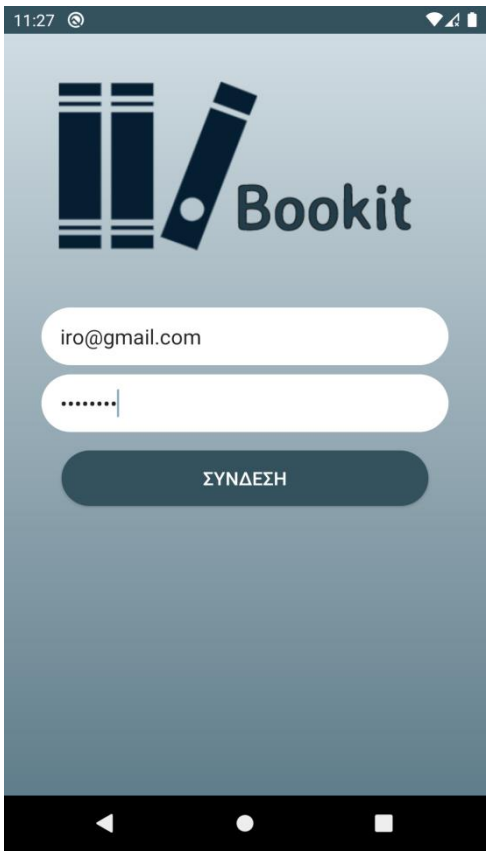
Σχήμα 12. activity_initial.xml



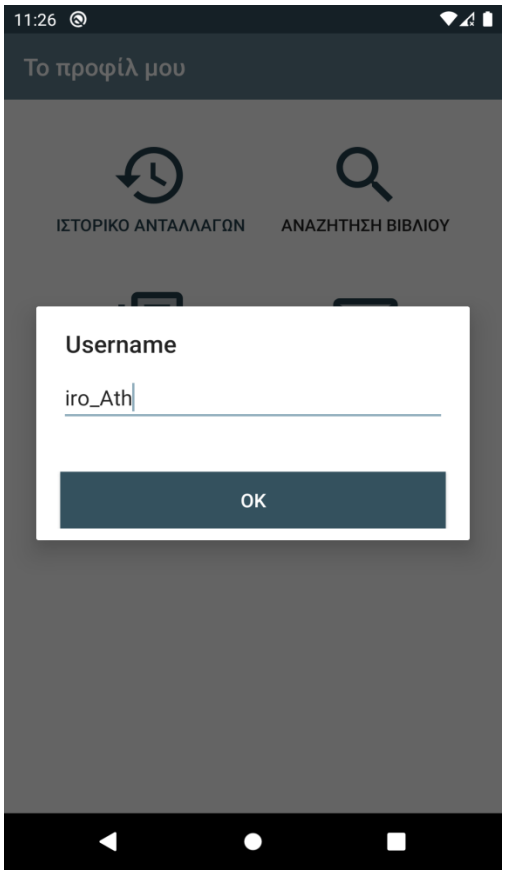
Σχήμα 13. dialog_reset_password.xml



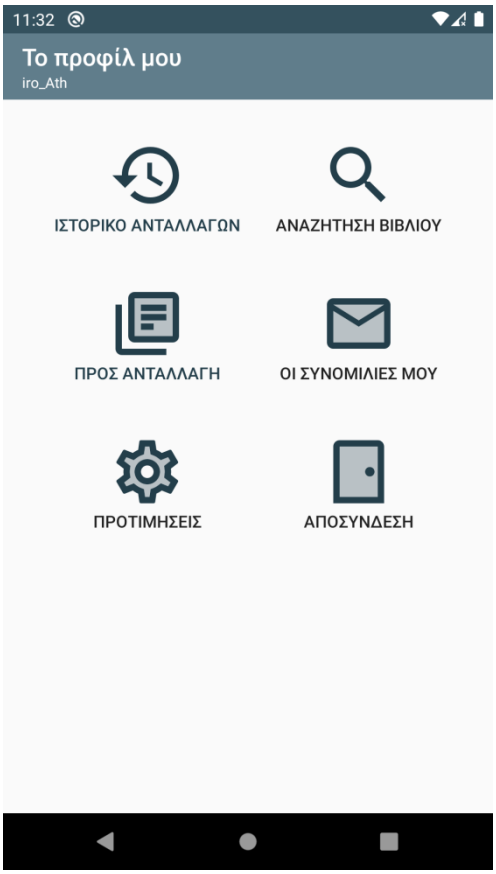
Σχήμα 14. activity_register.xml



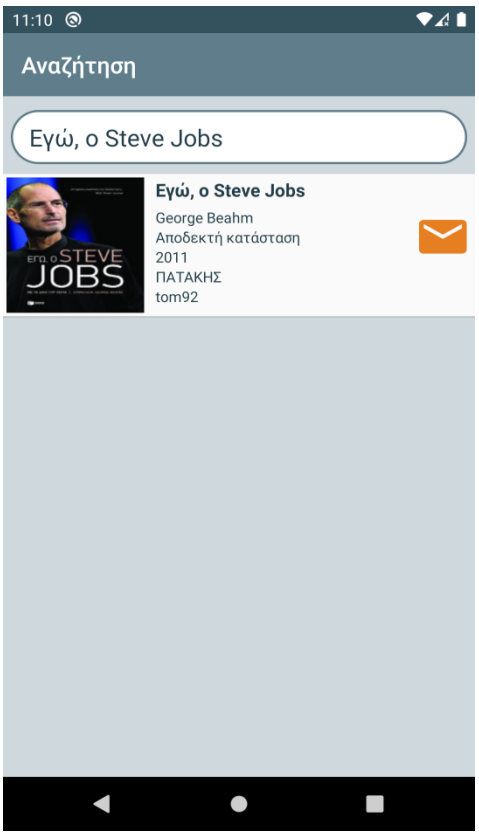
Σχήμα 15. activity_login.xml



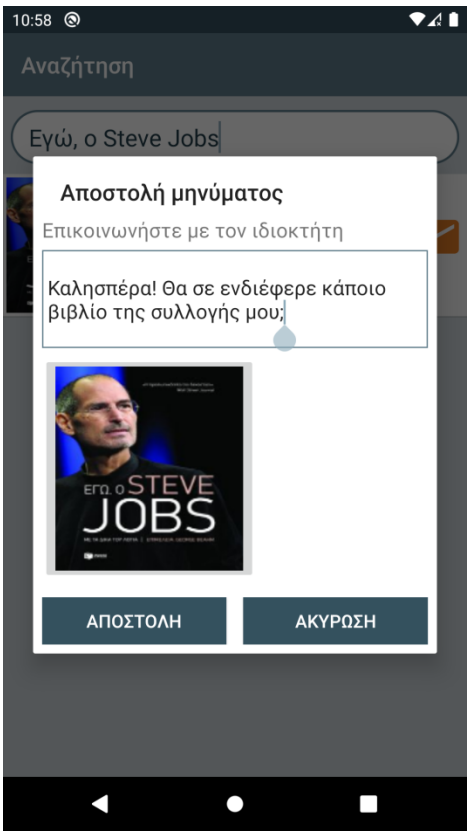
Σχήμα 16. dialog_create_username.xml



Σχήμα 17. activity_profile.xml

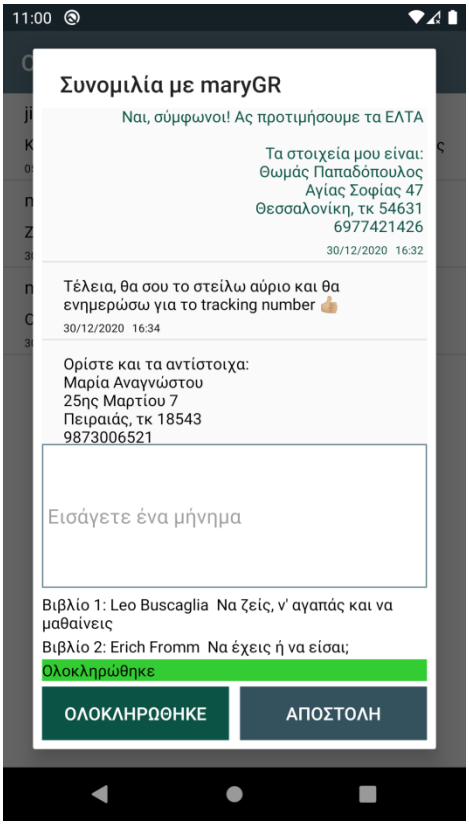


Σχήμα 18. activity_search.xml μαζί με list_row_books.xml

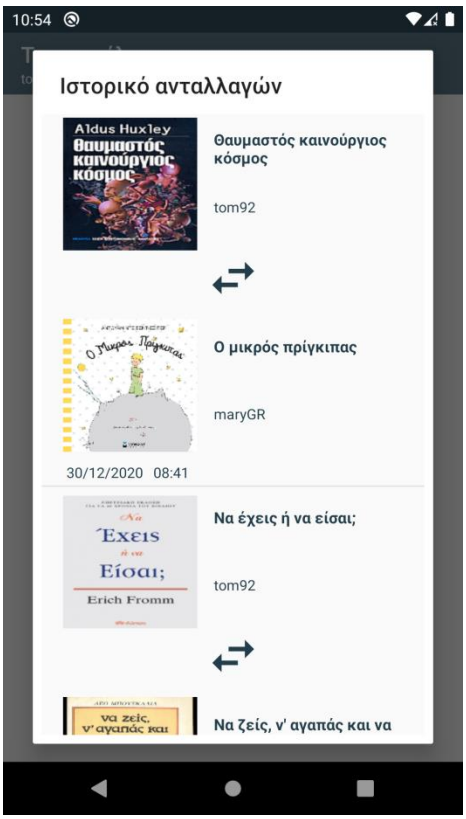


Σχήμα 19. dialog_send_message.xml

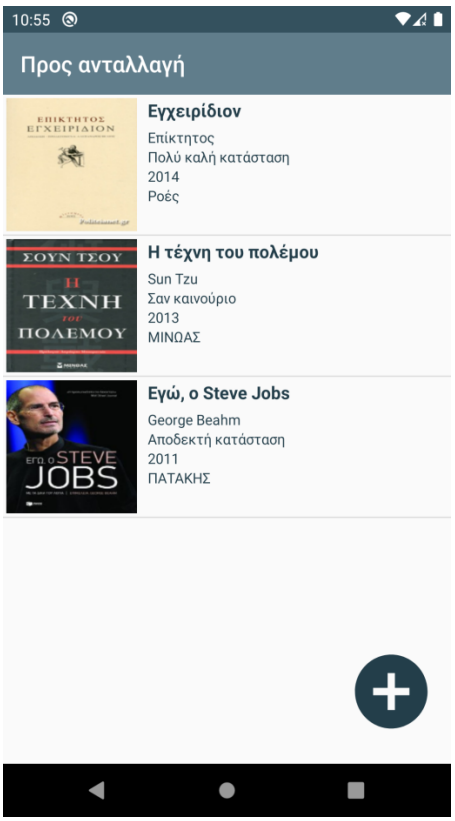
Σχήμα 20. activity_messages.xml μαζί με list_row_messages.xml



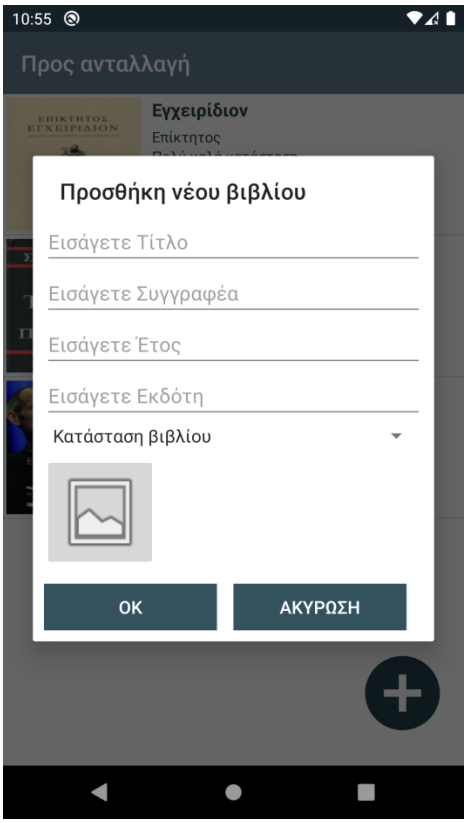
Σχήμα 21. activity_conversation.xml μαζί με list_row_messages.xml



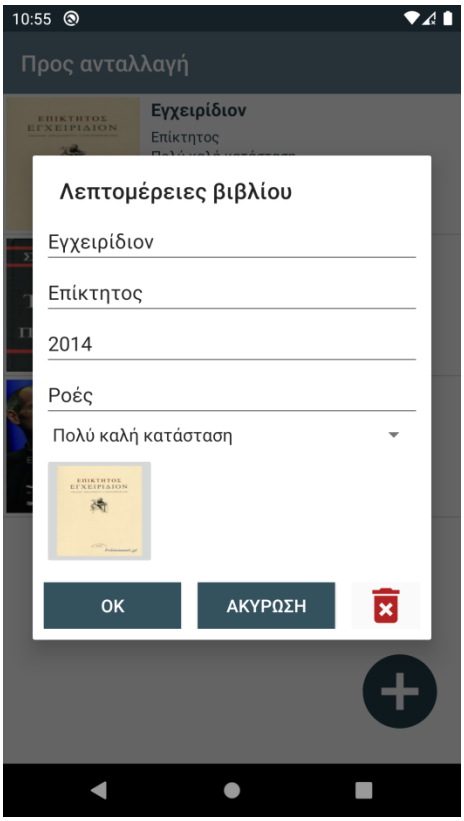
Σχήμα 22. dialog_history.xml μαζί με list_row_book_couples.xml



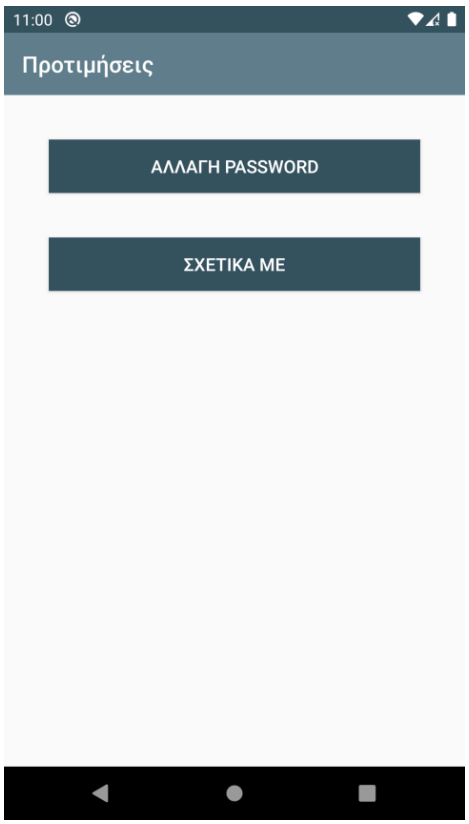
Σχήμα 23. activity_to_swap.xml μαζί με list_row_books.xml



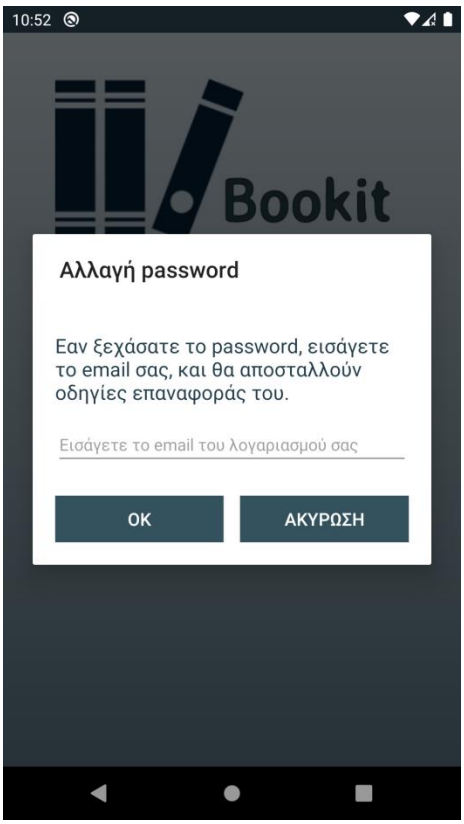
Σχήμα 24. dialog_add_new_book.xml για προσθήκη βιβλίου



Σχήμα 25. dialog_add_new_book.xml για επεξεργασία βιβλίου

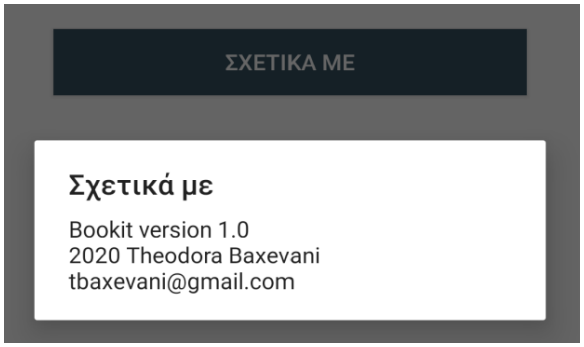


Σχήμα 26. activity_preferences.xml



Σχήμα 27. dialog_reset_password.xml

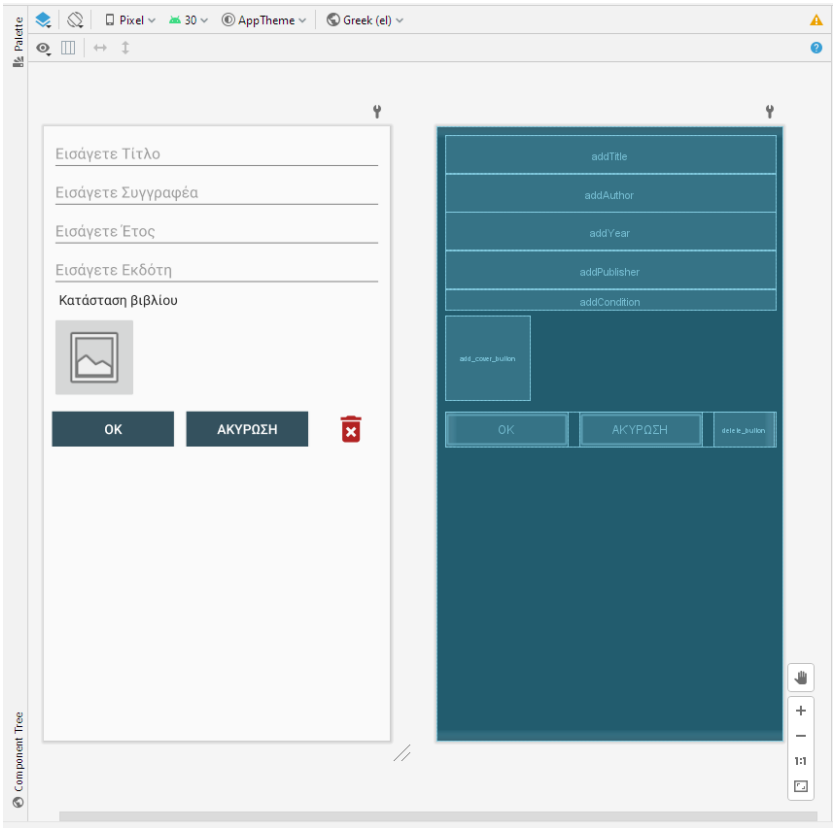
Τέλος, έγινε χρήση και απλοϊκών dialog τα οποία δεν αναμένουν ανάδραση από το χρήστη, αλλά χρησιμεύουν λειτουργώντας πληροφοριακά, όπως στην περίπτωση του «ABOUT» Button (Σχήμα 28).



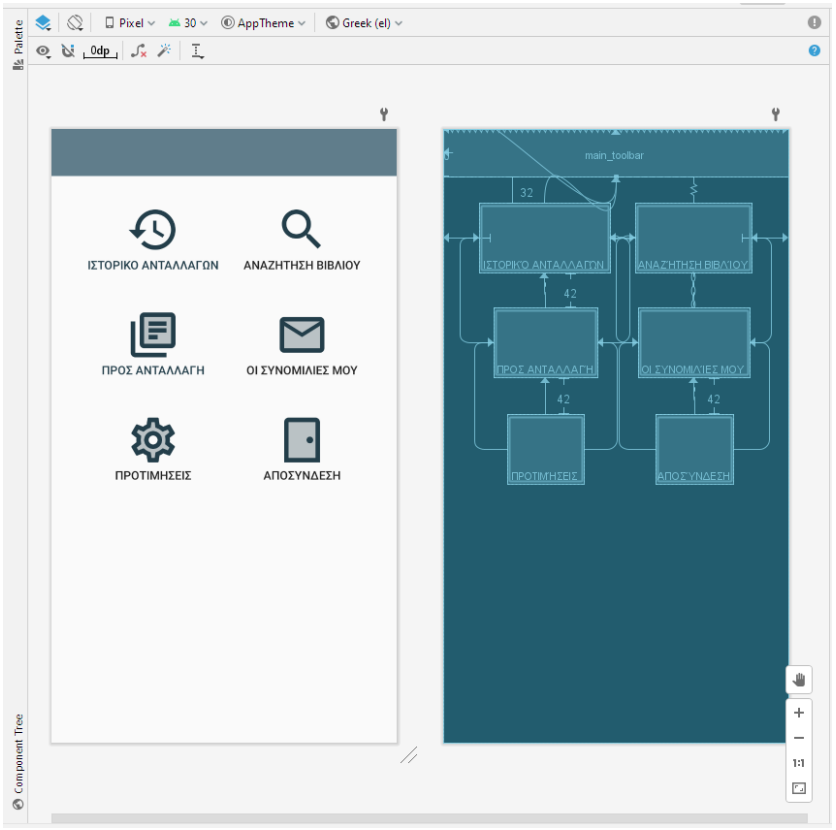
Σχήμα 28. Dialog χωρίς ανάδραση

Τα στοιχεία της διάδρασης των layouts (EditText,TextView κ.λ.π.), τοποθετήθηκαν μέσα στους εξής τύπους ViewGroup: Linear Layout για γραμμικές οριζοντίως ή καθέτως διατάξεις, και Constraint Layout για πιο περίπλοκες σχεδιαστικά οθόνες.

Ακολουθούν δύο παραδείγματα χρήσης τους στα Σχήματα 29 και 30.

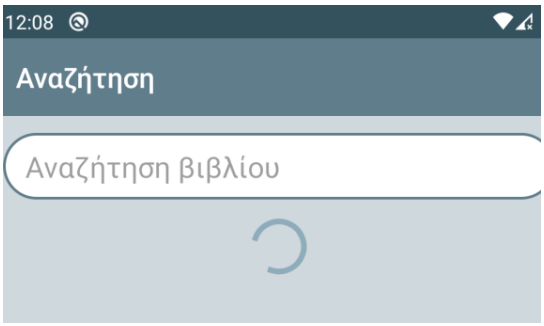


Σχήμα 29. Σχεδίαση με LinearLayout



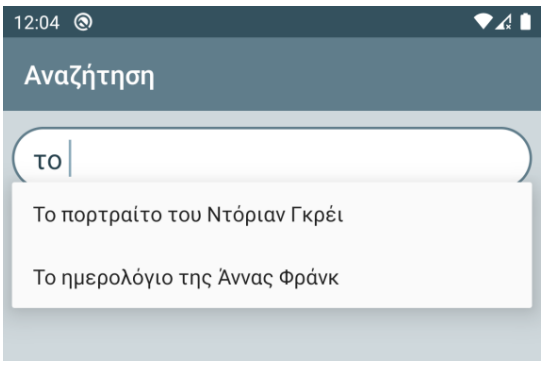
Σχήμα 30. Σχεδίαση με ConstraintLayout

Μόλις ένας χρήστης συνδεθεί στην εφαρμογή, κάτω από τον τίτλο του action bar «Το προφίλ μου», εμφανίζεται ως υπότιτλος το username του, ενημερώνοντάς τον έτσι για την επιτυχή του πρόσβαση με τον συγκεκριμένο λογαριασμό. Η ορατότητα της κατάστασης του συστήματος υποστηρίζεται επίσης μέσω κυκλικών progress bar όπως αυτό που φαίνεται στο Σχήμα 31, τα οποία είναι ορατά κατά τα τη διάρκεια εξέλιξης διαδικασιών που ενδέχεται να αποβούν χρονοβόρες, προκειμένου να αποφευχθεί η σύγχιση του χρήστη. Τέτοια παραδείγματα είναι η εμφάνιση των αποτελεσμάτων της αναζήτησης ενός βιβλίου, και η εμφάνιση των μηνυμάτων μιας συνομιλίας.



Σχήμα 31. Χρήση κυκλικού progress bar

Στο πεδίο της Αναζήτησης βιβλίου, επιλέχθηκε να είναι AutoCompleteTextView αντί για EditText. Αυτό επιλέχθηκε έτσι ώστε κατά τη διάρκεια που ο χρήστης πληκτρολογεί, να εμφανίζονται προτάσεις τίτλων βιβλίων που έχουν καταχωρηθεί προς ανταλλαγή και περιέχουν το κείμενο που μέχρι εκείνη τη στιγμή έχει εισάγει. Στο Σχήμα 32 φαίνεται το αποτέλεσμα μιας σύντομης πληκτρολόγησης Αυτή η λειτουργία έχει στόχο την εξοικονόμηση χρόνου του χρήση. Στην εφαρμογή γίνεται ταίριασμα (matching) αφού εισαχθεί και ο δεύτερος χαρακτήρας.

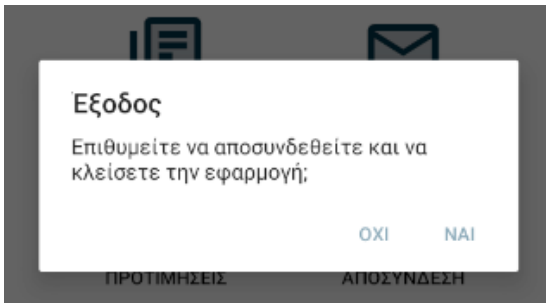


Σχήμα 32. Αυτόματη συμπλήρωση υπαρχόντων τίτλων

Ακολουθώντας την αρχή της συνέπειας, συχνά χρησιμοποιούμενα συστατικά όπως τα Buttons, σχεδιάστηκαν έχοντας κατά το δυνατό ίδια μορφή, μέσω ίδιας γραμματοσειράς και χρωμάτων. Αυτό βοηθά μειώνοντας το χρόνο εκπαίδευσης των χρηστών και ελαχιστοποιώντας την απαιτούμενη προσπάθεια, αφού η γνώση που έχει αποκτηθεί από μια οθόνη, μπορεί να εφαρμοστεί και σε άλλες παρόμοιες, όπως π.χ. στις λειτουργίες προσθήκης και επεξεργασίας ενός βιβλίου προς ανταλλαγή.

Στο σχεδιασμό των layout καταβλήθηκε επίσης προσπάθεια έτσι ώστε να μην παρουσιάζονται ταυτόχρονα πολλές πληροφορίες σε μια οθόνη, κάνοντάς την δυσνόητη και υπερφορτωμένη. Αντιθέτως, μέσω σαφών περιγραφών σε TextViews και Buttons, στόχος ήταν να αποφευχθεί η έκπληξη στους χρήστες από απρόβλεπτες συμπεριφορές της εφαρμογής, βοηθώντας τους να κατανοήσουν χωρίς δυσκολία πως λειτουργεί κάθε οθόνη. Τις περισσότερες φορές που επρόκειτο να πραγματοποιηθεί μια ενέργεια, δόθηκε ευκαιρία διαφυγής και επαναφοράς στην προηγούμενη

κατάσταση, μέσω των Button με την ένδειξη «ΑΚΥΡΩΣΗ» ή του default «BACK» της συσκευής. Πριν από ενέργειες όπως η διαγραφή ενός βιβλίου ή το κλείσιμο της εφαρμογής (βλ. Σχήμα 33), ζητείται επιβεβαίωση μέσω αντίστοιχου dialog.

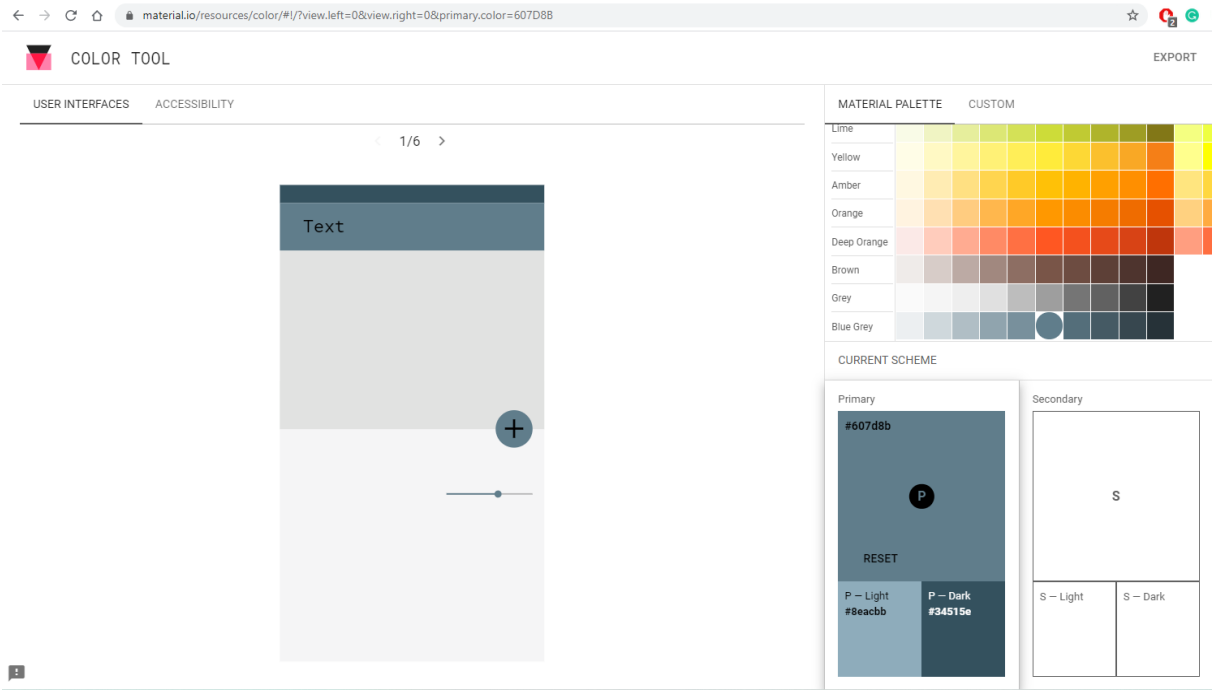


Σχήμα 33. Dialog με μήνυμα επιβεβαίωσης

Όταν ωστόσο αναπόφευκτα συμβούν λάθη όπως μη έγκυρη τιμή εισόδου κατά την πληκτρολόγηση ή σοβαρότερα σφάλματα, παρέχεται καθοδήγηση για ανάκαμψη από αυτά μέσω σύντομων μηνυμάτων που εμφανίζονται άμεσα σε Toast, αν και ο σχεδιασμός έχει γίνει με τρόπο που ελαχιστοποιεί την πιθανότητα να συμβούν. Για παράδειγμα, η φυσική κατάσταση ενός βιβλίου αποτελεί σημαντικό χαρακτηριστικό για τους συναλλασσόμενους, και έτσι η επιλογή της κρίθηκε πως θα πρέπει να γίνεται μόνο μέσω των προϋπαρχόντων διαθέσιμων ιεραρχικών τιμών ενός μενού σε Spinner, αποφεύγοντας μια λανθασμένη ή ασαφή πιθανόν καταχώρηση. Προτάσεις και πρακτικές εφαρμόστηκαν επίσης με βάση το σύγγραμμα του Luke Wroblewski [13]. Η τοποθέτηση των μηνυμάτων προτροπής για τη συμπλήρωση EditText έγινε εντός των πεδίων αυτών, με αριστερή στοίχιση κειμένου. Ακόμη, αποφεύχθηκαν πλήρως τα προαιρετικά πεδία συμπλήρωσης. Τα μηνύματα επεξήγησης, όπως αυτό του dialog_reset_password.xml, τοποθετήθηκαν σε κοντινή απόσταση με τα πεδία στα οποία αναφέρονται.

Χρώματα

Η δημιουργία μιας εφαρμογής, πέρα από τη λειτουργικότητα που προσφέρει, θα πρέπει να διαθέτει επίσης μια φιλική για το χρήστη εμφάνιση. Οι αποχρώσεις που θα χρησιμοποιηθούν αλλά και το πώς αυτές θα συνδυαστούν μεταξύ τους, επηρεάζει την εμπειρία του χρήστη. Το Material Design είναι ένα σύνολο σχεδιαστικών κανόνων και οδηγιών (guidelines) που δημιουργήθηκε και ανακοινώθηκε από τη Google το 2014 στο συνέδριο Google I/O, ενώ το επόμενο μόλις έτος είχε εφαρμοστεί σε δημοφιλείς Android εφαρμογές της Google όπως το Gmail και το YouTube. Για τους παραπάνω λόγους, επιλέχθηκε η χρήση του για την επιλογή χρωμάτων. Μετά από αρκετές δοκιμές που έγιναν μέσω του εργαλείου Color Tool που προσφέρεται από το Material Design, επιλέχθηκαν από την Blue Grey Material Palette τα τρία βασικά χρώματα που θα συμμετέχουν στην εφαρμογή, όπως φαίνεται στο Σχήμα 34. Πιο συγκεκριμένα, αποτελούνται από ένα κύριο (primary), ένα ανοιχτό (light) και ένα σκούρο (dark) χρώμα. Έπειτα, με βάση αυτά, δημιουργήθηκαν μερικά επιπλέον χρώματα, και όλα μαζί συνολικά, παρουσιάζονται στο Σχήμα 35. Η χρήση μιας παλέτα του Material Design προσέφερε στην εφαρμογή αρμονικές χρωματικά οθόνες και βοήθησε στην αναγνωσιμότητα κειμένου μέσω των αντιθέσεων των χρωμάτων.



Σχήμα 34. Χρήση του εργαλείου Color Tool για την επιλογή χρωμάτων

Όλα τα χρώματα που επιλέχθηκαν για την εφαρμογή, αποθηκεύτηκαν με τον δεκαεξαδικό κωδικό τους, στο αρχείο colors.xml εντός του project στο Android Studio. Αυτό έγινε έτσι ώστε να υπάρχει μια πλήρης και οργανωμένη εικόνα των αποχρώσεων που χρησιμοποιούνται, αλλά και για την ευκολία διαχείρισής τους, καθώς εάν αλλάξει εντός του αρχείου ο δεκαεξαδικός ενός χρώματος, αυτόματα θα εφαρμοστεί η αλλαγή και όπου αυτό χρησιμοποιείται. Κάθε φορά που έπρεπε να χρωματιστεί κάτι, γινόταν αναφορά στο επιθυμητό χρώμα μέσω του ονόματος που του είχε αποδοθεί στην ιδιότητα (attribute) name.

```
android:textColor="@color/colorDarker"
```

colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorDarkGrey">#222222</color>
    <color name="colorVeryDark">#051e32</color>
    <color name="colorDarker">#233e4a</color>
    <color name="colorDark">#34515E</color>
    <color name="colorPrimary">#607D8B</color>
    <color name="colorLight">#8EACBB</color>
    <color name="colorVeryLight">#D0DCE2</color>
    <color name="white">#FFFFFF</color>
    <color name="yellow">#E67E22</color>
    <color name="greenBlue">#0B5345</color>
    <color name="background_main">#fafafa</color>
    <color name="launcher_background">#CFD8DC</color>
</resources>
```

Σχήμα 35. Τα χρώματα που ορίστηκαν στο αρχείο colors.xml

Αναφορικά με το στυλ παρουσίασης των πληροφοριών, λήφθηκε ως παράμετρος η χρησιμότητα των χρωμάτων, ως εργαλείο εργονομίας. Στο παράθυρο μιας συνομιλίας, τα μηνύματα που έχει στείλει ο τρέχον χρήστης ως αποστολέας, στοιχίζονται στο δεξί μέρος της οθόνης έχοντας πράσινο χρώμα, ενώ τα μηνύματα του συνομιλητή εμφανίζονται στα αριστερά, με μαύρο χρώμα. Στο ίδιο παράθυρο, από τη στιγμή που μια ανταλλαγή σημαίνεται ως ολοκληρωμένη, η αντίστοιχη ένδειξη «Ολοκληρώθηκε» στη γραμμή κατάστασης εμφανίζεται και στους δύο συμμετέχοντες πλέον υπογραμμισμένη με πράσινο χρώμα. Άλλα χρώματα που χρησιμοποιήθηκαν είναι το κίτρινο για τον χρωματισμό του εικονιδίου με τον φάκελο, ο οποίος μιμείται τον κλασσικό φάκελο αλληλογραφίας. Τέλος, το κόκκινο χρώμα επιλέχθηκε για την προειδοποιητική σημασία του στο Button της διαγραφής ενός βιβλίου από τα προς ανταλλαγή.

Styles

Στο αρχείο styles.xml, ορίστηκε ένα σύνολο από attributes, μέσω των οποίων διαμορφώνεται η εμφάνιση των Views που θα λάβουν το εκάστοτε style. Στα style μπορούν να ορίζονται ιδιότητες όπως τα background και foreground color, το font size κ.α. Ένα style πρέπει να έχει τα attributes name και parent. Στο name επιτρέπεται οποιαδήποτε τιμή ως όνομα, ενώ στο parent πρέπει να επιλεχθεί ένα από τα υπάρχοντα Themes. Ένα Theme είναι ένα style το οποίο εφαρμόζεται σε ολόκληρη την εφαρμογή, όχι μόνο σε ένα View. Όταν ένα style εφαρμόζεται ως theme, κάθε View της εφαρμογής ακολουθεί αυτό το theme. Το colorPrimary υποδεικνύει το χρώμα του app bar. Το colorPrimaryDark υποδεικνύει το χρώμα του status bar. Το colorAccent υποδεικνύει το χρώμα των Views όπως check boxes, radio buttons, και edit texts. Οι τιμές των χρωμάτων που αποδίδονται δεν έχουν καταχωρηθεί απευθείας, αλλά είναι αναφορές ανακατεύθυνσης σε άλλο αρχείο με resources. Για τις ανάγκες της εφαρμογής χρειάστηκε η δημιουργία τριών style. Αυτά αφορούν τη μορφοποίηση του ToolBar, καθώς επίσης και την εμφάνιση του τίτλου και του υποτίτλου που εμπεριέχονται στο ToolBar, όπως φαίνεται στο Σχήμα 36.

styles.xml

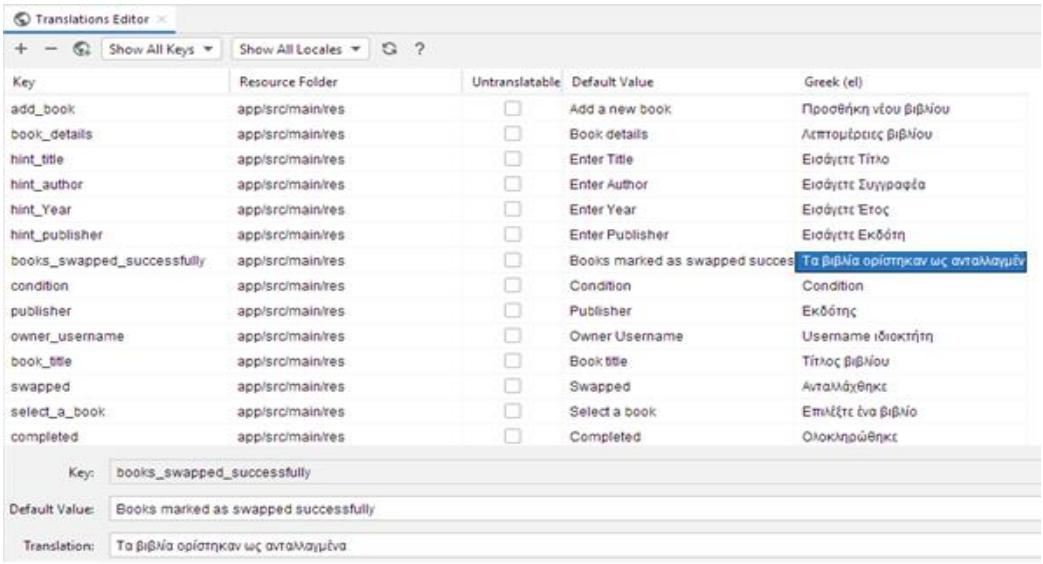
```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorDark</item>
        <item name="colorAccent">@color/colorLight</item>
        <item name="toolbarStyle">@style/ToolBarStyle</item>
    </style>

    <style name="ToolBarStyle" parent="Widget.AppCompat.Toolbar">
        <item name="titleTextAppearance">@style/TitleAppearance</item>
        <item name="subtitleTextAppearance">@style/SubTitleAppearance</item>
    </style>

    <style name="TitleAppearance" parent="TextAppearance.Widget.AppCompat.Toolbar.Title">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">@color/white</item>
    </style>

    <style name="SubTitleAppearance" parent="TextAppearance.Widget.AppCompat.Toolbar.Subtitle">
        <item name="android:textSize">12sp</item>
        <item name="android:textColor">@color/white</item>
    </style>
</resources>
```

Σχήμα 36. Τα style που ορίστηκαν στο αρχείο styles.xml

3.5 Strings

Όλες οι λέξεις και τα κείμενα που εμφανίζονται στις οθόνες της εφαρμογής, πέραν αυτών που εισάγει ο χρήστης, από τα πιο απλά μέχρι τα πιο σύνθετα, όπως οι προτροπές προς το χρήστη (hints), οι τίτλοι

των action_bar στα activities, τα πληροφοριακά μηνύματα των Toast και τα κείμενα των TextView, έχουν οριστεί εντός του αρχείου strings.xml, καθένα ξεχωριστά με την ετικέτα (tag) <string>, εντός των <resources>. Οι όροι που χρησιμοποιήθηκαν είναι απλοί και οικείοι για το μέσο χρήστη. Αρχικά τα strings γράφτηκαν στην Αγγλική γλώσσα, και έπειτα μεταφράστηκαν στην Ελληνική, στο αντίστοιχο strings.xml (el) αρχείο που δημιουργήθηκε στον Translations Editor μέσω της λειτουργίας Add Locale, όπως φαίνεται στο Σχήμα 37.

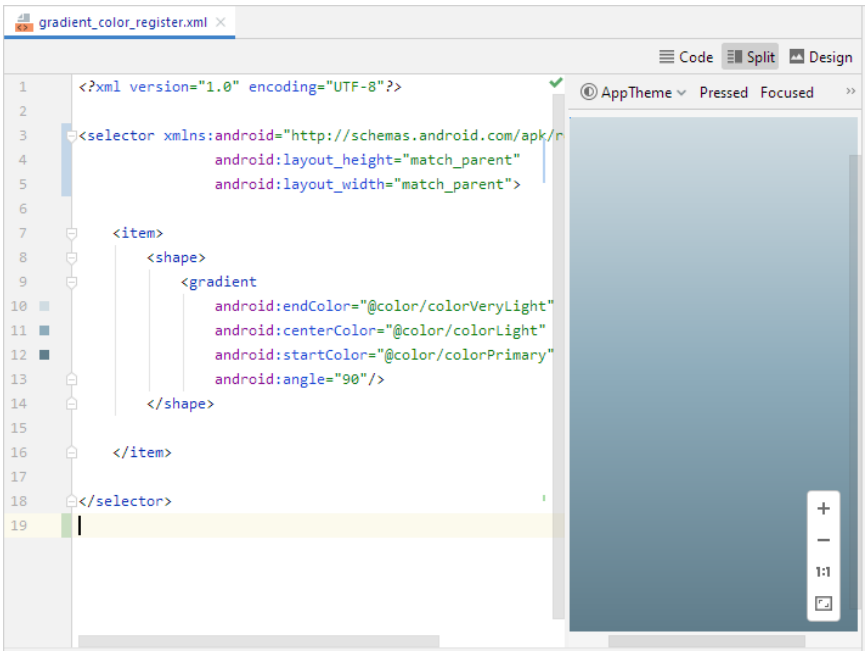


Σχήμα 37. Δημιουργία Ελληνικής μετάφρασης στον Translations Editor

3.6 Drawables

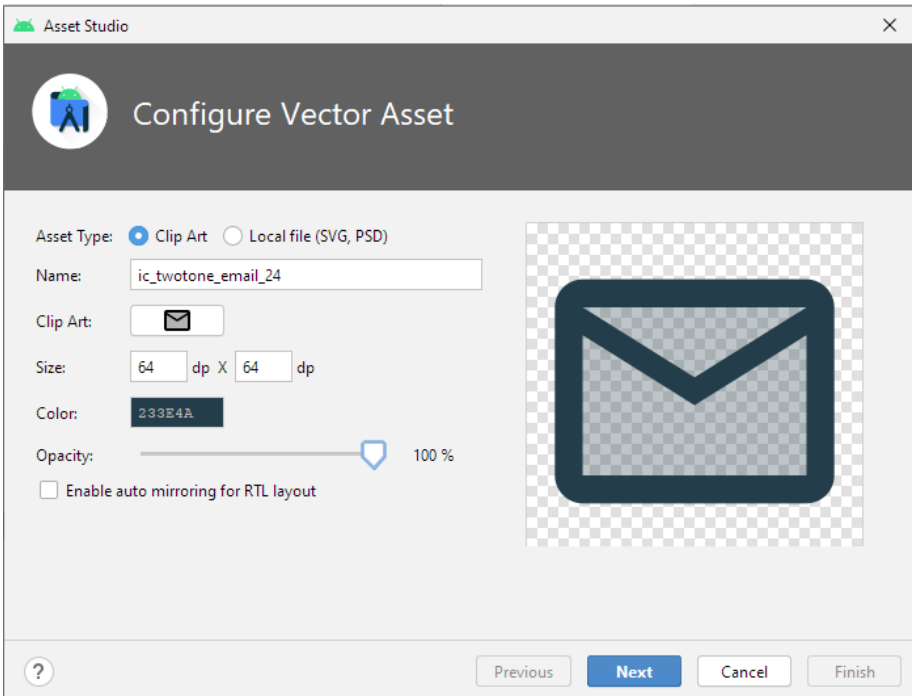
Για τις οθόνες της εφαρμογής δημιουργήθηκαν:

- Σχήματα με διαβαθμισμένα χρώματα (shape με gradient colors), όπως αυτό που φαίνεται στο Σχήμα 38, σε xml αρχεία με σκοπό τη χρήση τους ως background σε ViewGroup layouts.



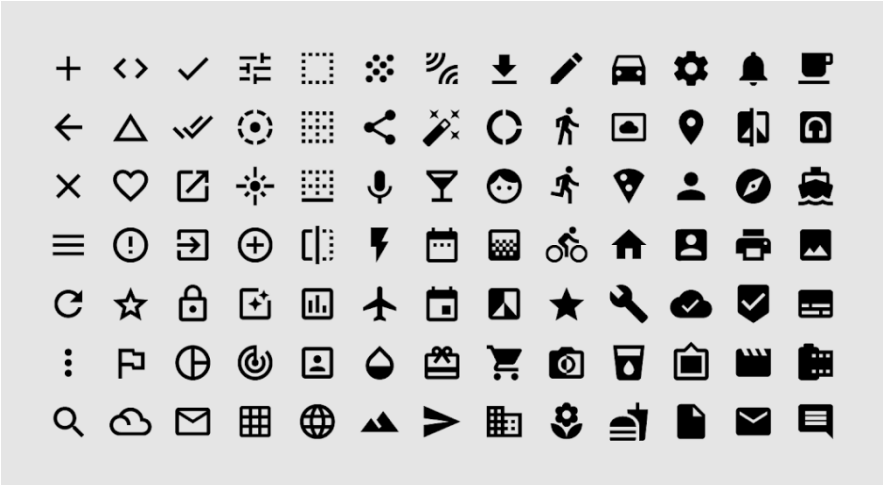
Σχήμα 38. Δημιουργία gradient shape σε XML

- Vector Asset εικονίδια μέσω του εργαλείου Vector Asset Studio στο Android Studio που παρουσιάζεται στο Σχήμα 39 και το οποίο εντοπίζεται ακολουθώντας τη διαδρομή File → New → Vector Asset.



Σχήμα 39. Δημιουργία Vector Asset στο Asset Studio

Το Android Studio περιλαμβάνει το εργαλείο Vector Asset Studio, το οποίο επιτρέπει την εισαγωγή εικονιδίων στο project, ως Vector drawable resources. Αυτά τα εικονίδια προέρχονται από το Material Icons, που αποτελεί μέρος του Material Design που αναφέρθηκε στην ενότητα 2.4. Τα Material icons είναι όμορφα και απλοϊκά σχεδιασμένα σύμβολα που απεικονίζουν συνηθισμένες ενέργειες ή αντικείμενα όπως φαίνεται στο Σχήμα 40.



Σχήμα 40. Μερικά από τα εικονίδια του Material Icons

Ένα Vector drawable είναι κατάλληλο για χρήση εικονιδίων. Σε σύγκριση με τη χρήση bitmaps μειώνει το μέγεθος του APK αρχείου, επειδή το ίδιο αρχείο μπορεί να αλλάζει μέγεθος (resize) ώστε

να προσαρμόζεται σε διάφορες πυκνότητες οθόνης (screen densities) χωρίς απώλειες στην ποιότητα της εικόνας. Τα εικονίδια που χρησιμοποιήθηκαν στην εφαρμογή (βλ. Σχήμα 41), όπως ο φάκελος αλληλογραφίας και ο μεγεθυντικός φακός, επιλέχθηκαν με βάση την αρχή της αντιληπτής δράσης (affordance) που προτείνεται στο βιβλίο «The design of everyday things» [14]. Δηλαδή τα αντικείμενα που απεικονίζονται υπαινίσσονται τη χρήση τους, εκφράζοντας έννοιες που προέρχονται από τις εμπειρίες των δυνητικών χρηστών στον πραγματικό κόσμο.

Σχήμα 41. Τα εικονίδια που δημιουργήθηκαν στο Vector Asset Studio

Τέλος, το λογότυπο της εφαρμογής, με βάση την ίδια φιλοσοφία, επιλέχθηκε να αποτελείται από μια αναπαράσταση βιβλίων, συνοδευόμενη από το όνομα της εφαρμογής όπως φαίνεται στο Σχήμα 42. Το όνομα Bookit επιλέχθηκε ως λογοπαίγνιο, από τις λέξεις book (στα Αγγλικά: βιβλίο), book (στα Αγγλικά: κάνω κράτηση) και it (στα Αγγλικά: αυτό). Η γραμματοσειρά που χρησιμοποιήθηκε είναι η δωρεάν για χρήση Averia Sans Libre, η οποία δημιουργήθηκε ως αποτέλεσμα του μέσου όρου όλων των sans-serif fonts. Οι μη αυστηρά καθορισμένες άκρες των χαρακτήρων της δίνουν μια φιλικότερη αίσθηση στο χρήστη, ενώ το συνολικό αποτέλεσμα παραμένει σοβαρό.

Σχήμα 42. Το λογότυπο της εφαρμογής

3.7 Επίλογος

Στο κεφάλαιο 3, παρουσιάστηκαν διεξοδικά όλα τα συστατικά μέρη που απαρτίζουν το Project της εφαρμογής στο Android Studio (Classes, Activities, Dialogs, Custom Adapters, Layouts, Strings και Drawables). Για το καθένα περιγράφηκαν οι λόγοι δημιουργίας του, η χρησιμότητά του, οι συσχετίσεις του με άλλα στοιχεία του project,το πώς χρησιμοποιήθηκε, καθώς και συνοδευτικά στιγμιότυπα όπου ήταν αναγκαίο. Η κατανόηση αυτού του κεφαλαίου ισοδυναμεί με πλήρη κατανόηση του τρόπου χρήσης της εφαρμογής.

Κεφάλαιο 4ο: Λεπτομέρειες Υλοποίησης

4.1 Εισαγωγή

Μετά την περιγραφή των περιεχομένων του Project που έγινε στο κεφάλαιο 3, θα ακολουθήσει επεξήγηση διαδικασιών και προγραμματιστικών λεπτομερειών, όπου κρίνεται απαραίτητη μια βαθύτερη ανάλυση του κώδικα. Πιο συγκεκριμένα, η προσοχή θα εστιαστεί στις μεθόδους του Firebase και του Firestore, στη δημιουργία νέων αντικειμένων και εισαγωγή τους στη Β.Δ. ως document, στη λήψη document από τη Β.Δ και μετατροπή τους σε αντικείμενα, στην αποθήκευση των εικόνων του χρήστη, και στη τεχνική recycle views των adapter.

4.2 Μέθοδοι Firebase Authentication

Ένας από τους λόγους για τους οποίους επιλέχθηκε η χρήση της πλατφόρμας Firebase, ήταν οι λειτουργίες οι οποίες έχουν υλοποιηθεί ήδη σε έτοιμες μεθόδους, και μπορούν να χρησιμοποιηθούν απλά μέσω της κλήσης τους. Έτσι λοιπόν ακολουθήθηκαν οι οδηγίες προσθήκης του Firebase στο project του Android Studio. Έπειτα στο Firebase console, δημιουργήθηκε ένα Firebase project ως container της εφαρμογής. Μέσω αυτού αποκτάται πρόσβαση σε υπηρεσίες (features) όπως το Firestore, το Data Analytics κ.α. που μπορούν πλέον να χρησιμοποιηθούν για το συγκεκριμένο project. Έπειτα έγινε καταχώρηση (register) της εφαρμογής στο Firebase project. Στη συνέχεια έγινε download και προστέθηκε στον φάκελο app του project της εφαρμογής, το αρχείο Firebase Android configuration (google-services.json) το οποίο φτιάχτηκε αποκλειστικά για το συγκεκριμένο project. Προκειμένου να ενεργοποιηθεί η δυνατότητα χρήσης των Firebase products για την εφαρμογή, ακολούθησε η προσθήκη του google-services plugin στα Gradle αρχεία. Τέλος, προστέθηκε (add) στην εφαρμογή το Firebase SDK χρησιμοποιώντας το Firebase Android Bill of Materials (BoM), δηλώνοντας στο module (app-level) Gradle αρχείο του project τα κατάλληλα dependencies. Χρησιμοποιώντας το BoM, εξασφαλίζεται πως η εφαρμογή θα χρησιμοποιεί πάντα συμβατές (compatible) version των Firebase Android libraries.

4.2.1 createUserWithEmailAndPassword() και sendEmailVerification()

Η εγγραφή ενός χρήστη στην εφαρμογή, υλοποιήθηκε με χρήση των Firebase Authentication μεθόδων, στην κλάση RegisterActivity. Στο Authentication section του Firebase project, ενεργοποιήθηκε από την καρτέλα (tab) Sign-in method, η πρώτη επιλογή εκ των providers, “Email/Password”. Σε αυτό το σημείο ξεκίνησε η αντίστοιχη συγγραφή κώδικα. Αφού εισήχθησαν οι απαραίτητες γραμμές στο τμήμα των import, έγινε αρχικοποίηση (initialization) του Firebase Auth αντικειμένου εντός της onCreate μεθόδου της κλάσης RegisterActivity.

```
private lateinit var auth: FirebaseAuth
// ...
auth = Firebase.auth
```

Έτσι κατέστη δυνατή η κλήση της μεθόδου createUserWithEmailAndPassword με ορίσματα το email και το password τα οποία θα αποτελούν τα στοιχεία αυθεντικοποίησης του εν λόγω χρήστη εφέξής.

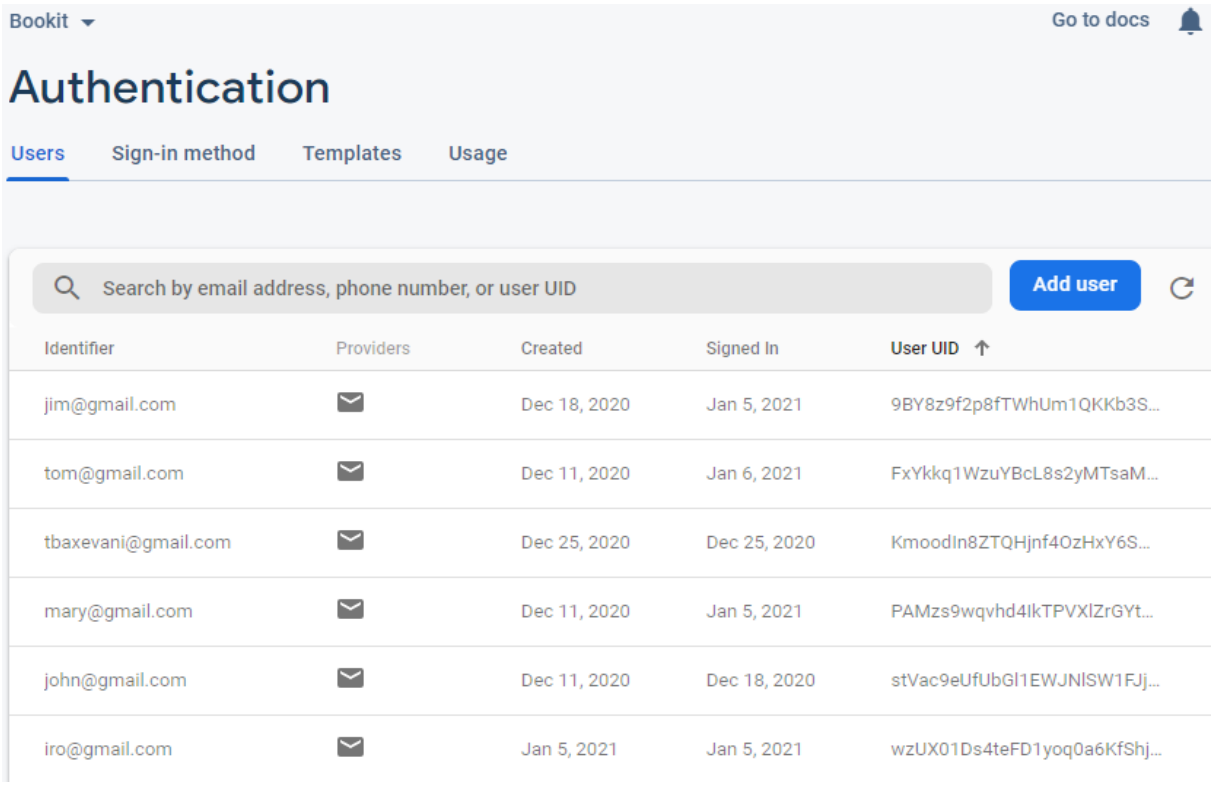
```
auth.createUserWithEmailAndPassword(email,password)
    .addOnSuccessListener {
```

```
        val user = auth.currentUser
        sendEmailVerification(user)
    }
```

```
private fun performRegister() {
    val email = email_edittext_register.text.toString()
    val password = password_edittext_register.text.toString()
    if (email.isEmpty() || password.isEmpty()) {
        Toast.makeText( context: this,"Fill all fields", Toast.LENGTH_LONG).show()
        return
    }
    // Create a user with email & password on Firebase
    auth.createUserWithEmailAndPassword(email,password)
        .addOnSuccessListener { it: AuthResult!
            val user = auth.currentUser
            val newUser = User( username: "",email)
            db.collection( collectionPath: "users").document(user!!.uid)
                .set(newUser)
                .addOnFailureListener { it: Exception
                    Toast.makeText( context: this, "Registration failed",Toast.LENGTH_LONG).show()
                }
                .addOnSuccessListener { it: Void!
                    auth.signOut()
                    Toast.makeText( context: this,"Successful registration",Toast.LENGTH_LONG).show()
                    //update the UI with Login activity
                    val intent = Intent( packageContext: this,LoginActivity::class.java)
                        intent.putExtra( name: "email",email)
                        intent.putExtra( name: "password",password)
                    startActivity(intent)
                    sendEmailVerification(user)
                    finish()
                }
        }
        .addOnFailureListener { it: Exception
            Log.d( tag: "ProfileActivity", "Failed" + " ${it.message}")
            if ((password.length < 6) && (!email.contains( char: '@')))
                Toast.makeText( context: this,"Invalid data", Toast.LENGTH_LONG).show()
            else if (password.length < 6)
                Toast.makeText( context: this,"Password requires \nat least 6 characters", Toast.LENGTH_LONG).show()
            else
                Toast.makeText( context: this,"Invalid email", Toast.LENGTH_LONG).show()
        }
}
private fun sendEmailVerification(user: FirebaseUser?) {
    user!!.sendEmailVerification()
}
```

Με την επιτυχή κλήση της μεθόδου, δηλαδή εντός του successListener, κλήθηκε η μέθοδος sendEmailVerification, η οποία αποστέλλει στο email του χρήστη ένα email επιβεβαίωσης εγγραφής. Το email αυτό λειτουργεί κυρίως ενημερωτικά, καθώς ακόμη και αν ο χρήστης δεν επιβεβαιώσει την εγγραφή του, μια νέα εγγραφή προστίθεται στον «πίνακα» Users του Authentication στο αντίστοιχο Firebase project. Κάθε εγγραφή του πίνακα αυτού, όπως φαίνεται στο Σχήμα 43, αφορά τη δημιουργία

ενός χρήστη και παρουσιάζει κάποιες πληροφορίες σχετικά με αυτόν, όπως το email του και το μοναδικό UID που του αποδόθηκε. Ο διαχειριστής αυτής της πλατφόρμας έχει τη δυνατότητα να αλλάξει το password ενός χρήστη, να απενεργοποιήσει το λογαριασμό του ή ακόμη και να τον διαγράψει.



Σχήμα 43. Οι εγγεγραμμένοι χρήστες της εφαρμογής στον πίνακα του Authentication

4.2.2 sendPasswordResetEmail()

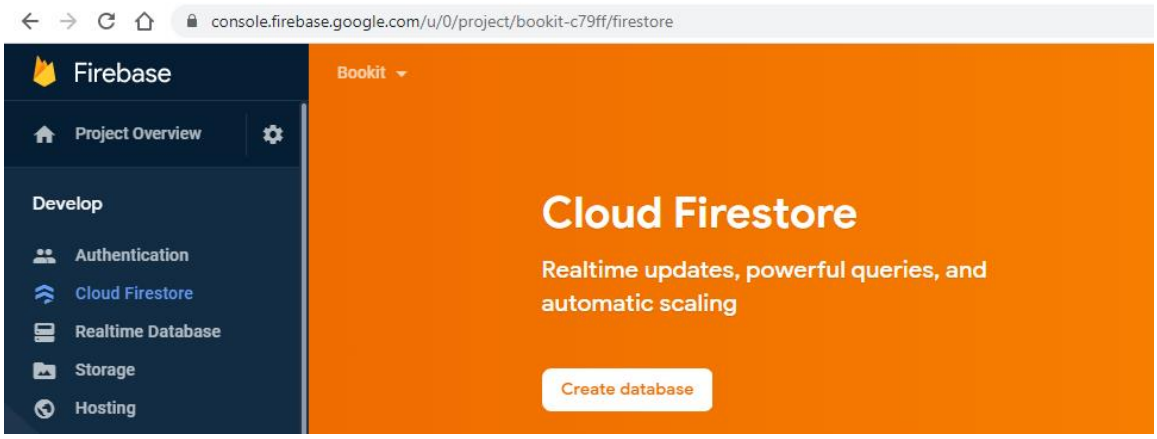
Στις περιπτώσεις που ο χρήστης ξεχάσει το password του ή επιθυμεί να το αλλάξει, η διαδικασία αυτή πραγματοποιείται μέσω αναλυτικών οδηγιών που λαμβάνει στο email του λογαριασμού του. Ακολουθώντας τες μπορεί να επιλέξει νέο password, καθώς η λειτουργία υπενθύμισης password δεν παρέχεται από το Firebase κατά το χρονικό διάστημα εκπόνησης της διπλωματικής εργασίας. Παρακάτω φαίνεται η χρήση της sendPasswordResetEmail Firebase μεθόδου εντός του InitialActivity για την περίπτωση που ο χρήστης έχει ξεχάσει το password του.

```
dialogForgotPassword.ok_button.setOnClickListener{
    val email = dialogForgotPassword.email_to_send_reset_pas.text.toString()
    if (email=="" || email == null) {
        Toast.makeText(this, getString((R.string.invalid_email)),
          Toast.LENGTH_SHORT).show()
        return@setOnClickListener
    }
    auth.sendPasswordResetEmail(email)
        .addOnSuccessListener {
            // Registered and Signed in automatically }
```

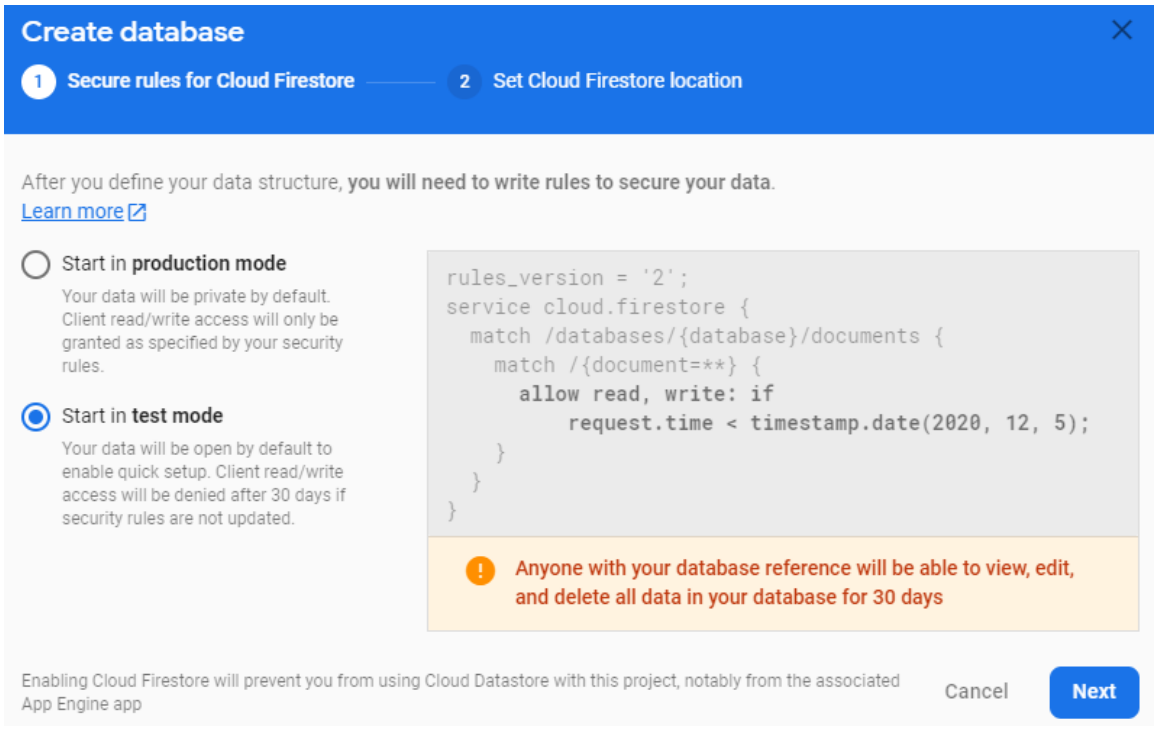
4.3 Μέθοδοι Firestore

Οι χρήστες που πραγματοποιούν εγγραφή (register), είναι απαραίτητο να αποθηκεύονται επίσης και στη βάση δεδομένων της εφαρμογής, μαζί με όσες επιπλέον πληροφορίες χρειάζεται ώστε να επιτευχθούν οι διάφορες λειτουργίες της εφαρμογής. Γι' αυτό το σκοπό δημιουργήθηκε μέσω κώδικα η κλάση User που παρουσιάστηκε στο Σχήμα 9. Ένα αντικείμενο τύπου User αποφασίστηκε πως θα πρέπει να συνοδεύεται πέραν του email, και από ένα username, το οποίο θα λειτουργεί ως μοναδικό αναγνωριστικό όνομα κατά την αλληλεπίδραση με τους υπόλοιπους χρήστες.

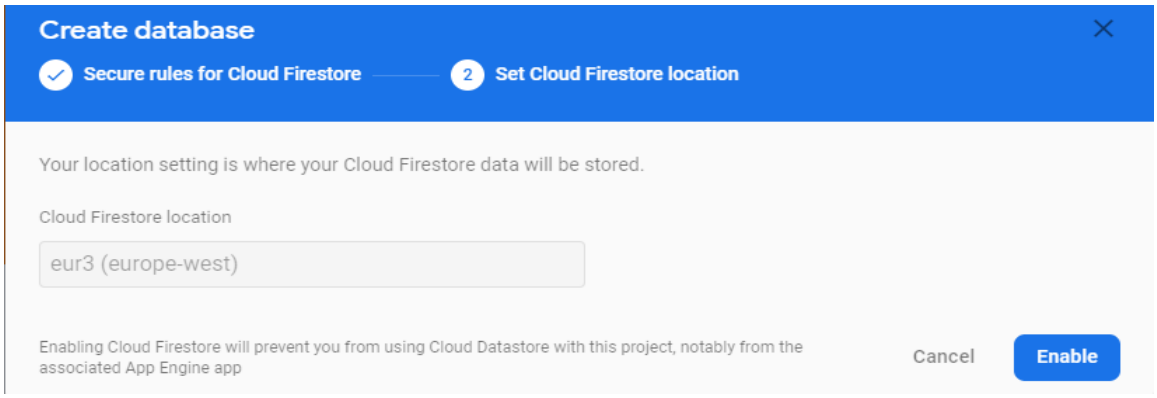
Σε αυτό το σημείο ξεκίνησε η δημιουργία της βάσης δεδομένων. Από το Firebase project της εφαρμογής στο Firebase console, επιλέχθηκε το Cloud Firestore από το αριστερό μέρος των προϊόντων-εργαλείων (βλ. Σχήμα 44) και ακολούθησαν τα βήματα των Σχημάτων 45 και 46 οδηγώντας τελικά στη δημιουργία της βάσης, όπως αυτή φαίνεται στο Σχήμα 47.



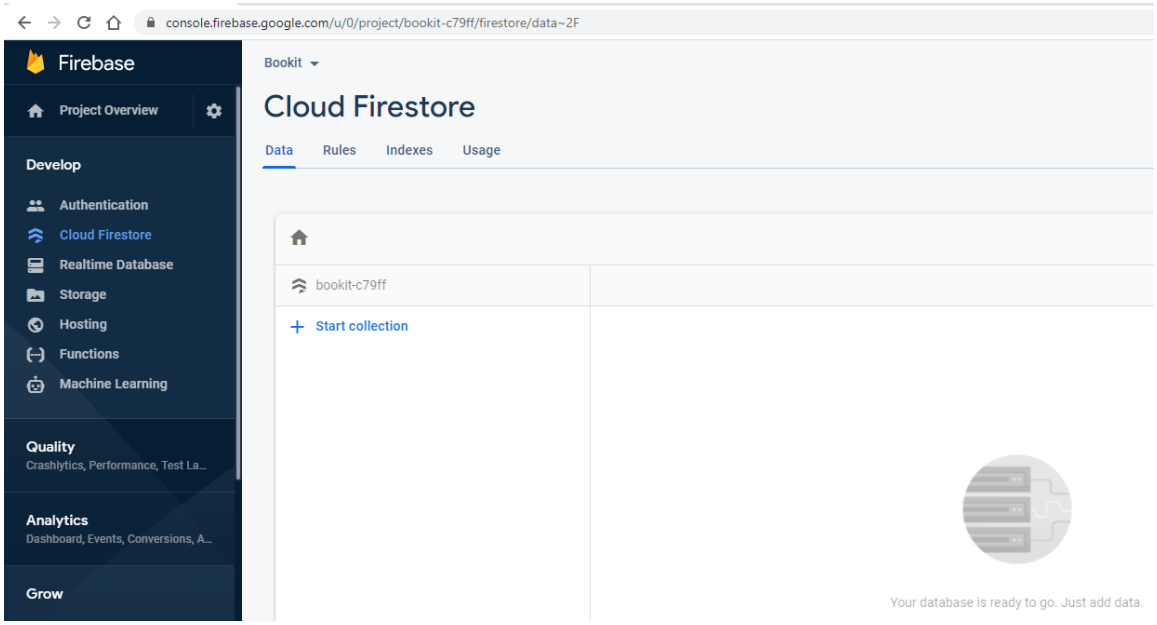
Σχήμα 44. Δημιουργία της βάσης δεδομένων



Σχήμα 45. Επιλογή Test mode για τα Security rules

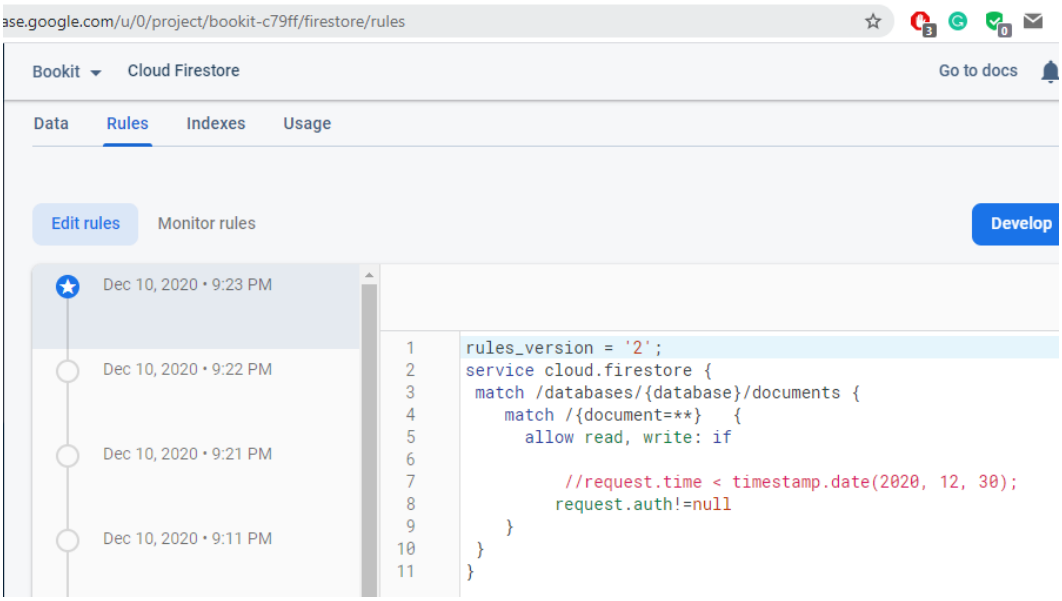


Σχήμα 46. Επιλογή επιθυμητής γεωγραφικής περιοχής για GCP services



Σχήμα 47. Η βάση δεδομένων, αμέσως μετά τη δημιουργία της

Κατόπιν, επιλέχθηκαν τα Rules της βάσης δεδομένων όπως φαίνεται στο Σχήμα 48, δηλαδή ένα σύνολο οδηγιών για τον έλεγχο πρόσβασης στα δεδομένα. Τα rules αποτελούνται από match δηλώσεις, οι οποίες χρησιμεύουν για αναγνωρισμό documents, και από allow εκφράσεις οι οποίες καθορίζουν την πρόσβαση και τις επιτρεπτές λειτουργίες σε αυτά. Για παράδειγμα η δήλωση match /{document=**} αναφέρεται σε οποιοδήποτε document βρίσκεται στη βάση δεδομένων. Κάθε αίτημα προς το Firestore (π.χ. read, write) προτού εκτελεστεί, αξιολογείται με βάση τα rules, και εάν η πρόσβαση δεν γίνει αποδεκτή (deny), τότε όλο το request αποτυγχάνει.



Σχήμα 48. Τα Rules που ορίστηκαν για τη βάση δεδομένων στο Firestore

Υπάρχουν τρείς τρόποι εγγραφής (write) δεδομένων στο Cloud Firestore:

- Προσθήκη ενός νέου document με δεδομένα σε ένα collection. Σε αυτή την περίπτωση το Firestore δημιουργεί αυτόματα γι' αυτό το document ένα μοναδικό αναγνωριστικό (id).

- Προσθήκη ενός νέου κενού document σε ένα collection. Σε αυτή την περίπτωση το Firestore δημιουργεί αυτόματα για αυτό το document ένα μοναδικό αναγνωριστικό (identifier). Στα field του document μπορούν να ανατεθούν τιμές αργότερα.

- Τροποποίηση των τιμών στα field ενός document. Φυσικά θα πρέπει να γίνεται αναφορά στο συγκεκριμένο document μέσω του id του.

Για τα field ενός document, το Firestore υποστηρίζει τους παρακάτω τύπους δεδομένων: strings, booleans, numbers, dates, bytes, binary blobs, null, arrays, maps, Cloud Firestore references και Geographical point values. Οι αριθμοί αποθηκεύονται πάντα ως double, ανεξάρτητα από τον τύπο τους στον κώδικα.

Η δημιουργία των collection της βάσης δεν έγινε χειροκίνητα, αλλά έμμεσα, προγραμματιστικά. Η δημιουργία ενός document που προορίζεται για ένα collection που δεν υπάρχει τη δεδομένη χρονική στιγμή, πυροδοτεί (triggers) τη δημιουργία του ώστε να το φιλοξενήσει.

4.3.1 set()

Κατά αυτό τον τρόπο, αφού ολοκληρωθεί η εγγραφή ενός χρήστη, δημιουργείται μέσω κώδικα ένα αντικείμενο τύπου User με κενό username προς το παρόν, και email αυτό που καταχώρησε ο χρήστης για την εγγραφή του. Έπειτα δίνεται εντολή στη βάση του Firestore, μέσω της μεθόδου set, να δημιουργηθεί εντός του collection users, ένα document με μοναδικό αναγνωριστικό του το uid του χρήστη που μόλις εγγράφηκε. Την πρώτη φορά που θα εκτελεστούν αυτές οι γραμμές, θα δημιουργηθεί το collection users, και εντός του θα καταχωρηθεί ένα document με στοιχεία στα field του τις τιμές των πεδίων του αντικειμένου newUser. Τα field ενός document ταυτίζονται με τα πεδία

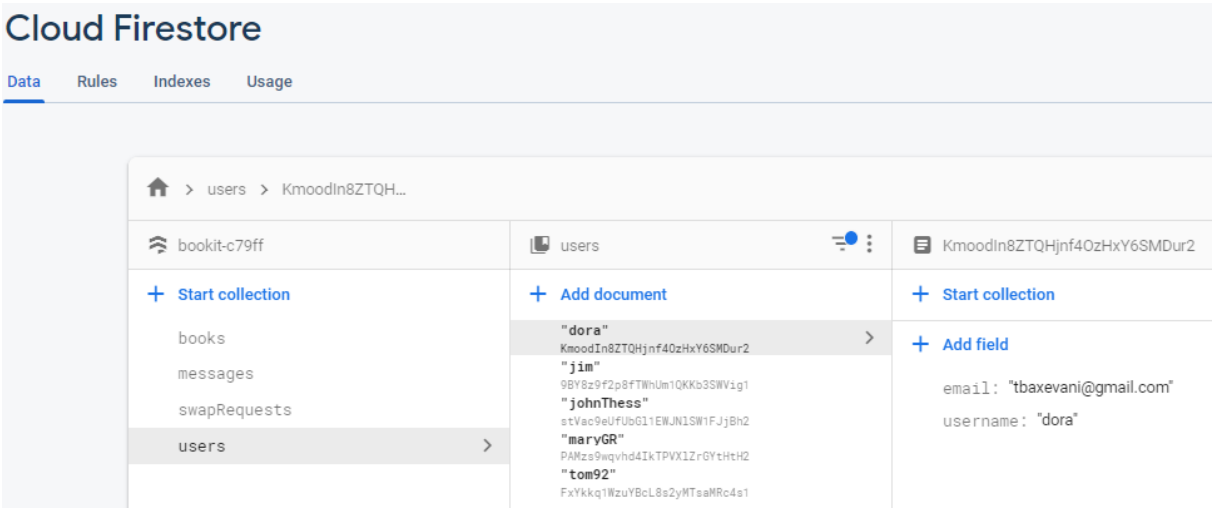
του τύπου αντικειμένου που το δημιούργησαν. Τις επόμενες φορές που θα δημιουργείται ένας χρήστης, αφού το collection users θα υπάρχει ήδη (existing) απλώς θα προστίθεται ένα νέο document κάθε φορά. Αξίζει να σημειωθεί πως εάν ένα collection περιέχει ένα μόνο document, η διαγραφή του θα προκαλέσει αυτόματα και τη διαγραφή του collection. Δηλαδή, στη βάση δεδομένων δεν επιτρέπεται να υπάρχει collection χωρίς document.

```
val user = auth.currentUser
val newUser = User("",email)
db.collection("users").document(user!!.uid)
    .set(newUser)
```

Χρήση της εντολής set έγινε επίσης για προσθήκη τιμών στα πεδία ενός υπάρχοντος κενού document, μέσω αναφοράς σε αυτό. Για παράδειγμα η προσθήκη ενός νέου βιβλίου στη βάση ως document, συνοδεύεται από ένα id το οποίο αποκτά αυτόματα. Αυτό το id όμως είναι χρήσιμο να αποθηκεύεται και ως ξεχωριστή πληροφορία, σε ένα ομώνυμο field εντός του. Έτσι, αφού αρχικά δημιουργηθεί στη βάση το νέο document και κρατήσουμε σε μια μεταβλητή το id που του αποδόθηκε, μπορούμε έπειτα να δημιουργήσουμε ένα αντικείμενο Book δίνοντάς του αυτό το id μαζί με τις υπόλοιπες επιθυμητές τιμές στα πεδία του. Το αντικείμενο αυτό θα δώσει τις τιμές του στα field του ζητούμενου document, μέσω αναφοράς στο id του.

```
val newBookDocRef : DocumentReference = db.collection("books").document()
val newBook = Book(newBookDocRef.id,ownerid,currentUsername,title,author,year,
                   publisher,conditionSelected,cover,"",0L)
db.collection("books").document(newBookDocRef.id)
    .set(newBook)
```

Το αποτέλεσμα των κλήσεων της set() που πραγματοποιήθηκαν, καθόρισε την τελική μορφή της βάσης δεδομένων, που παρουσιάζεται στο Σχήμα 49. Φυσικά αυτή μπορεί να αλλάζει καθώς γίνονται νέες εγγραφές ή διαγραφές.



Σχήμα 49. Η βάση δεδομένων κατά την ολοκλήρωση της εφαρμογής

4.3.2 update()

Η μέθοδος update επιτρέπει την τροποποίηση υπαρχόντων τιμών σε fields ενός document. Μπορεί να χρησιμοποιηθεί για την ενημέρωση ενός ή περισσότερων field όπως φαίνεται παρακάτω.

Όταν ένας χρήστης επιλέξει ένα username το οποίο γίνει αποδεκτό, τότε ενημερώνεται μόνο το αντίστοιχο field (username) του document που αντιπροσωπεύει το συγκεκριμένο χρήση.

```
db.collection("users").document(user.uid)
    .update("username",givenUsername)
```

Στο παράθυρο επεξεργασίας των στοιχείων ενός βιβλίου ωστόσο, με το πάτημα του «OK» οφείλουν να ενημερωθούν οι τιμές όλων των field του document που το αντιπροσωπεύει.

```
db.collection("books").document(book.bookID)
   .update("bookId",book.bookID,"title",title,"author",author,
           "year",year,"publisher",publisher,"condition",conditionSelected,
            ,"cover",cover)
```

4.3.3 get()

Το Cloud Firestore παρέχει δύο τρόπους ανάκτησης δεδομένων, καθένας εκ των οποίων μπορεί να χρησιμοποιηθεί για λήψη document, collection με documents, ή αποτελέσματα ερωτημάτων στη βάση (queries).

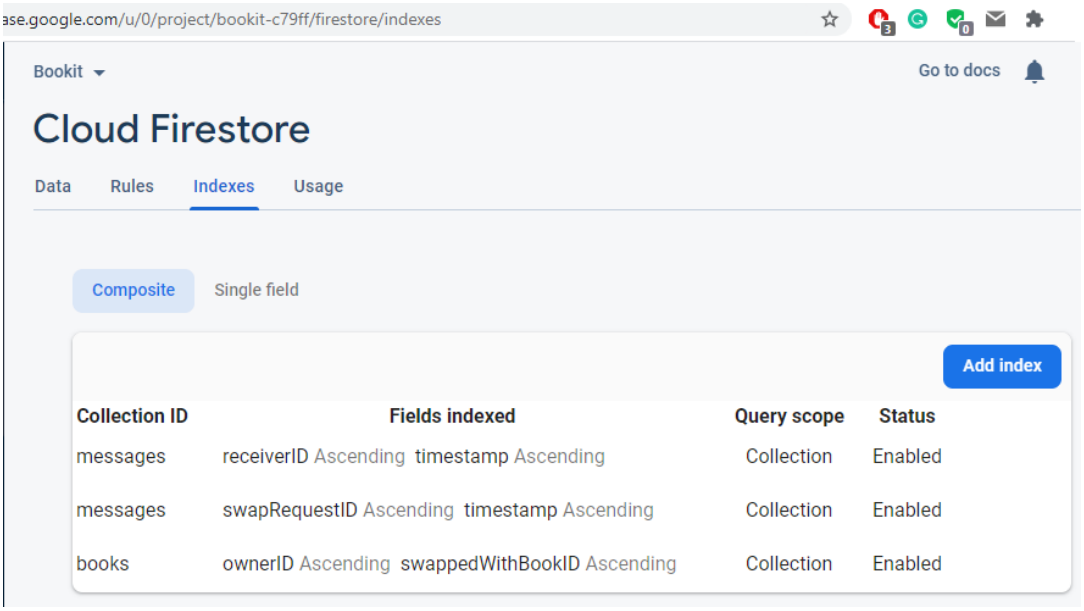
- Κλήση μεθόδου για λήψη δεδομένων
- Ορισμός listener για άμεση λήψη αλλαγών σε δεδομένα

Όταν εφαρμόζεται ένας listener, το Firestore του αποστέλλει αρχικά ένα στιγμιότυπο (snapshot) των δεδομένων, και έπειτα κάθε φορά που συμβεί μια αλλαγή σε αυτά, αποστέλλεται εκ νέου snapshot.

Ένα παράδειγμα χρήσης του get στην εφαρμογή, αποτελεί η φόρτωση των document που αντιστοιχούν στα βιβλία που ο τρέχον συνδεδεμένος χρήστης έχει καταχωρήσει προς ανταλλαγή, ώστε να προβληθούν εντός του αντίστοιχου ListView, στο ToSwapActivity.

```
db.collection("books")
    .whereEqualTo("ownerID", currentUid)
    .whereEqualTo("swappedWithBookID", "")
    .get()
```

Για την εκτέλεση σύνθετων ερωτημάτων όπως το παραπάνω, το Firestore έχει προνοήσει έτσι ώστε να εκτελούνται γρήγορα, χωρίς να μειώνεται η απόδοση της εφαρμογής. Αυτό επιτυγχάνεται μέσω των Indexes, που αποτελούν ευρετήρια στα οποία εκτελούνται τα αντίστοιχα queries. Τα Indexes που απαιτούνται για τα βασικότερα queries στην εφαρμογή, δημιουργούνται αυτόματα, ενώ για πιο στοχευμένα ερωτήματα, ο προγραμματιστής καλείται να τα δημιουργήσει, μέσω της εύχρηστης μεν διεπιφάνειας που προσφέρεται στην κονσόλα του Firestore. Για τα queries της εφαρμογής χρειάστηκε να δημιουργηθούν τρία Indexes τα οποία παρουσιάζονται στο Σχήμα 50. Το get() ερώτημα που αναφέρθηκε πιο πάνω, θα εκτελεστεί με βάση το τελευταίο (τρίτο) Index του Σχήματος 50.



Σχήμα 50. Τα Indexes που δημιουργήθηκαν στο Firestore

Με τη μέθοδο get() του Firestore, τα δεδομένα που επιστρέφονται είναι τύπου document. Τα document είναι απαραίτητο να μετατραπούν στους αντίστοιχους τύπους δεδομένων που αντιπροσωπεύουν, έτσι ώστε να γίνει χειρισμός τους προγραμματιστικά. Αυτή η μετατροπή γίνεται με τη μέθοδο toObject(). Βέβαια, προϋπόθεση είναι η ύπαρξη constructor στις αντίστοιχες κλάσεις.

Για παράδειγμα στην περίπτωση του παραπάνω αποσπάσματος κλήσης της get(), τα document που λαμβάνονται, πρέπει να μετατραπούν σε αντικείμενα τύπου Book.

```
val book = d.toObject(Book::class.java)
```

```
private fun updateList(){
    db.collection( collectionPath: "books")
        .whereEqualTo( field: "ownerID", currentUid)
        .whereEqualTo( field: "swappedWithBookID", value: "")
        .get()
        .addOnSuccessListener { documents ->
            for (d in documents) {
                val book = d.toObject(Book::class.java)
                bookArrayList.add(book)
            }
            adapter = CustomAdapterBooks( context: this, bookArrayList, mListener: null, showEnvelope: false, currentUsername)
            to_swap_list.adapter = adapter
        }
        .addOnFailureListener { exception ->
            Toast.makeText( context: this, exception.toString(), Toast.LENGTH_LONG).show()
            val tag = "toSwapActivity"
            Log.d(tag, exception.toString())
        }
}
```

Η μέθοδος get(), χρησιμοποιήθηκε ακόμη για την φόρτωση ενός ολόκληρου collection, όπως στην περίπτωση του collection users, εντός του ProfileActivity. Αυτό έγινε προκειμένου να ελεγχθούν τα username όλων των χρηστών, και να αποφανθεί εαν κάποιο ήδη υπάρχει.

```
db.collection("users").get()
```

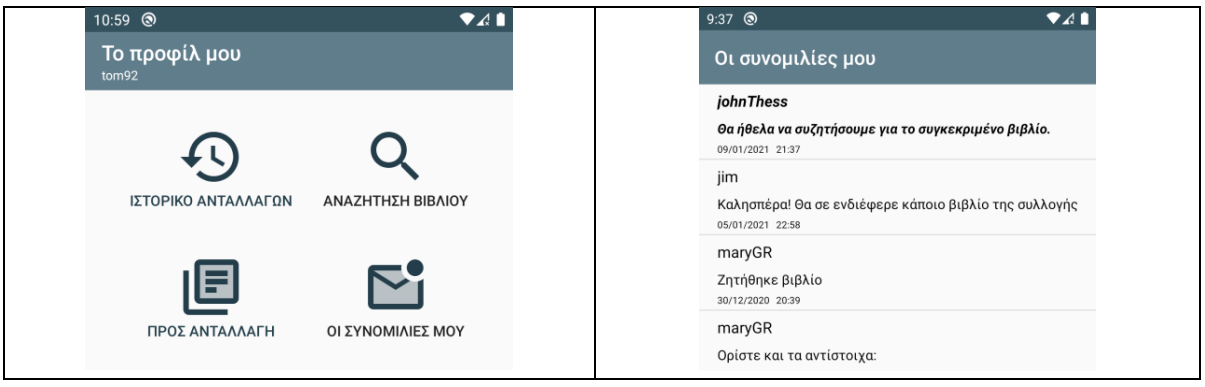
Τέλος, η εντολή get μπορεί να προσκομίσει μια μεμονωμένη τιμή από ένα field ενός document της βάση δεδομένων, όπως λ.χ. το username ενός χρήστη.

```
db.collection("users").document(user!!.uid)
    .get()
    .addOnSuccessListener { d ->
        currentUsername = d.get("username").toString()
```

Η χρήση listener επέτρεψε στην εφαρμογή να παρέχει συνομιλίες τύπου chat μεταξύ των χρηστών, ανανεώνοντας σε πραγματικό χρόνο τη λίστα προβολής μηνυμάτων με νέα μηνύματα που αποστέλλονται. Η μέθοδος που χρησιμοποιείται για την εφαρμογή του listener είναι η onSnapshot(). Αυτή επιστρέφει ένα νέο documentSnapshot κάθε φορά που δημιουργείται νέο μήνυμα που σχετίζεται με το εκάστοτε swapRequest.

```
db.collection("messages")
    .whereEqualTo("swapRequestID", swapRequestID)
    .addSnapshotListener { documentSnapshot, _ ->
```

Κάθε φορά που ένας χρήστης λαμβάνει ένα μήνυμα ενώ δεν βρίσκεται εντός της συγκεκριμένης συνομιλίας τη δεδομένη χρονική στιγμή, το μήνυμα αυτό θεωρείται και είναι μη αναγνωσμένο. Με την τοποθέτηση listener σε όσα μηνύματα (document του collection messages) έχουν τον εκάστοτε συνδεδεμένο χρήστη ως παραλήπτη, και είναι αδιάβαστα, ο χρήστης ειδοποιείται σε πραγματικό χρόνο για την έλευσή τους, μέσω του εικονιδίου των μηνυμάτων στην Κεντρικής οθόνης, αλλά και εντός των συνομιλιών, καθώς αυτή εμφανίζεται πρώτη στη λίστα, με bold και italic μορφή.




```
db.collection("messages")
    .whereEqualTo("receiverID", currentUid)
    .whereEqualTo("read", false)
    .addSnapshotListener { documentSnapshot, _ ->
```

4.3.4 delete()

Το Firestore δίνει τη δυνατότητα διαγραφής προγραμματιστικά για fields, documents και collections.

Όταν ένας χρήστης διαγράφει ένα βιβλίο που έχει καταχωρήσει προς ανταλλαγή, διαγράφεται το αντίστοιχο document του collection books στη βάση δεδομένων καλώντας τη μέθοδο delete().

```
db.collection("books").document(book.bookID)
    .delete()
```

4.4 Χειρισμός εικόνων

Όπως περιγράφηκε πιο πάνω, για κάθε βιβλίο ενός χρήστη, δημιουργείται ένα αντικείμενο τύπου Book, και στο cover πεδίο του απαιτείται η αποθήκευση μιας εικόνας ως εξώφυλλο. Το Firebase δίνει τη δυνατότητα αποθήκευσης αρχείων πολυμέσων μέσω χρήσης του εργαλείου Cloud Storage. Με αυτό, η εικόνα αποστέλλεται στη βάση δεδομένων και αποθηκεύεται ως αρχείο εικόνας. Μετέπειτα, κάθε φορά που χρειάζεται προσπέλασή της, αυτή γίνεται μέσω reference στη διαδρομή (path) του αρχείου π.χ. "bookimages/little_prince.png".

Ωστόσο αυτή η προσέγγιση προϋποθέτει πέρα από τη χρήση του Firestore, τη χρήση και ενός ακόμη προϊόντος, γεγονός που αυξάνει την πολυπλοκότητα και το κόστος δημιουργίας της εφαρμογής. Άλλωστε, η χρήση του Cloud Storage ενδείκνυται για αποθήκευση μεγάλων αρχείων εικόνας, ήχου ή βίντεο. Για τις ανάγκες της διπλωματικής εργασίας επαρκεί η αποθήκευση μιας μικρής σχετικά εικόνας για κάθε βιβλίο, η οποία προορίζεται να καταλαμβάνει ένα μικρό μόνο μέρος στην οθόνη της συσκευής. Αυτό σημαίνει πως δε θα καταλαμβάνει μεγάλο χώρο η αποθήκευσή της, και έτσι αποφασίστηκε η αξιοποίηση των δυνατοτήτων μόνο του Firestore για τη διαδικασία αυτή.

Δεδομένου ότι ένα document του Firestore δε δύναται να έχει μέγεθος μεγαλύτερο του 1 MB, θα πρέπει να εξασφαλιστεί ότι κάθε εικόνα θα έχει κατάλληλο μέγεθος ώστε να είναι εφικτή η αποθήκευσή της. Αυτό επιτυγχάνεται τόσο με την αλλαγή μεγέθους (resize) του bitmap που παράγεται από την είσοδο του χρήστη, όσο και με τη συμπίεση (compress) στην οποία υποβάλλεται, με κατάλληλες παραμέτρους.

4.4.1 Δημιουργία Bitmap

Δεδομένου ότι ένα document του Firestore δε δύναται να έχει μέγεθος μεγαλύτερο του 1 MB, θα πρέπει να εξασφαλιστεί ότι κάθε εικόνα θα έχει κατάλληλο μέγεθος ώστε να είναι εφικτή η αποθήκευσή της. Αυτό επιτυγχάνεται τόσο με την αλλαγή μεγέθους (resize) του bitmap που παράγεται από την είσοδο του χρήστη, όσο και με τη συμπίεση (compress) στην οποία υποβάλλεται, με κατάλληλες παραμέτρους.

Αρχικά ο χρήστης επιλέγει μια εικόνα από το Gallery της συσκευής του, μέσω Intent. Η επιλεγμένη εικόνα επιστρέφεται ως URI, στο τρέχον Activity (ToSwapActivity) με τη μέθοδο OnActivityResult(). Ακολουθεί η δημιουργία ενός InputStream από το επιστρεφόμενο URI, από το οποίο και θα προκύψει ένα Bitmap μέσω κλήσης της μεθόδου decode(). Η decode() προαιρετικά δέχεται το όρισμα options, το οποίο όμως είναι χρήσιμο, καθώς περιλαμβάνει παραμέτρους που καθορίζουν την ποιότητα και το μέγεθος του παραγόμενου Bitmap. Τέλος, πραγματοποιήθηκε σμίκρυνση του Bitmap σε μέγεθος 150x150 pixels και αποθήκευσή του στη μεταβλητή tempBitmap η οποία είναι ορατή σε όλο το Activity, έτσι ώστε να είναι προσβάσιμη από το dialogAddBook και το dialogEditBook.

```
//get the URI and convert it to InputStream
val selectedImage: Uri? = data?.data
val `is`: InputStream? = selectedImage?.let {
```

```
contentResolver.openInputStream(it) }

//selection of the Bitmap parameters
val options: BitmapFactory.Options = BitmapFactory.Options()
    options.inJustDecodeBounds = false
    options.inScaled = false

// every pixel is stored in 4 bytes with the ARGB_8888
options.inPreferredConfig = Bitmap.Config.ARGB_8888
val bitmap = BitmapFactory.decodeStream(`is`, null, options)

//downsizing
tempBitmap = bitmap?.let { Bitmap.createScaledBitmap(it, 150, 150, true) }
dialogAddBook.add_cover_button.setImageBitmap(bitmap)
```

Το περιεχόμενο της μεταβλητής tempBitmap είναι το Bitmap που προέκυψε με τις επιθυμητές διαστάσεις, γι' αυτό και τοποθετείται ως background στο ImageButton add_cover_button. Βέβαια όταν ο χρήστης ολοκληρώσει την καταχώρηση στοιχείων ενός νέου βιβλίου ή την επεξεργασία ενός υπάρχοντος, θα πρέπει να πατήσει το Button «OK» ώστε να αποθηκευτούν οι νέες τιμές των πεδίων.

4.4.2 Μετατροπή Bitmap σε Blob

Με το onClick event στο ok_button ξεκινά η μετατροπή του tempBitmap σε αντικείμενο τύπου ByteArray.Με το πέρας της μετατροπής αυτής, η δεσμευμένη μνήμη του tempBitmap απελευθερώνεται με τη μέθοδο recycle() όπως συστήνεται για τα αντικείμενα τύπου Bitmap.

```
val baos = ByteArrayOutputStream()
tempBitmap?.compress(Bitmap.CompressFormat.PNG, 100, baos)
val b = baos.toByteArray()
tempBitmap?.recycle()
```

Στο σημείο αυτό υπήρξαν δύο πιθανοί τρόποι προσέγγισης για την τελική αποθήκευση της εικόνας στο πεδίο cover του Book:

1. Η κωδικοποίηση του ByteArray σε String με τη μέθοδο Base64 και η αποθήκευση του String.
2. Η μετατροπή του ByteArray σε τύπο Blob και η αποθήκευση του Blob.

Με την πρώτη μέθοδο όμως, προκύπτει μεγαλύτερου μεγέθους αρχείο. Επιπλέον, δεδομένου ότι το FireStore περιλαμβάνει στους αποδεκτούς τύπους data το Blob, αποφασίστηκε η αποθήκευση της εικόνας ως τέτοιο. Το Blob (Binary Large Object) του Firestore είναι ένας τύπος δεδομένων για την αποθήκευση αμετάβλητων (immutable) array από bytes [15].

Η μετατροπή από ByteArray σε Blob γίνεται εύκολα με τη μέθοδο fromBytes().

```
val cover = Blob.fromBytes(b)
```

Όταν γίνεται ανάκτηση της εικόνας από τη βάση δεδομένων, το cover μετατρέπεται αρχικά από Blob σε ByteArray με τη μέθοδο toByteArray() και στη συνέχεια το ByteArray σε Bitmap ώστε να παρουσιαστεί στον χρήστη μέσα σε ImageView ή σε ImageButton όπου χρειάζεται.

Ακολουθεί ο κώδικας που χρησιμοποιήθηκε για την προβολή του cover ενός βιβλίου εντός του ListView των αποτελέσματα της Αναζήτησης (SearchActivity)

```
val imageBytes : ByteArray = arrayList[position].cover.toByteArray()
val decodedImage = BitmapFactory.decodeByteArray(imageBytes,0,imageBytes.size)
holder.cover.setImageBitmap(decodedImage)
```

4.5 Χειρισμός ημερομηνίας

Κατά τη δημιουργία ενός αντικειμένου τύπου Message, είναι απαραίτητη η αποθήκευση της χρονικής στιγμής κατά την οποία στάλθηκε το μήνυμα. Η τιμή αυτή είναι η εκάστοτε τρέχουσα ημερομηνία, και αποθηκεύεται στο τύπου Long πεδίο timestamp. Δηλαδή σε αυτό καταχωρείται το αποτέλεσμα της μεθόδου currentTimeMillis(), αφού η επιστρεφόμενη τιμή είναι η τρέχουσα ώρα σε μορφή milliseconds. Αυτά μετρώνται από την την 1η Ιανουαρίου του 1970 UTC (Coordinated Universal time), και λ.χ. για την ημερομηνία Sun Jan 10 2021 08:49:00, η επιστρεφόμενη τιμή της currentTimeMillis() είναι: 1610268540389.

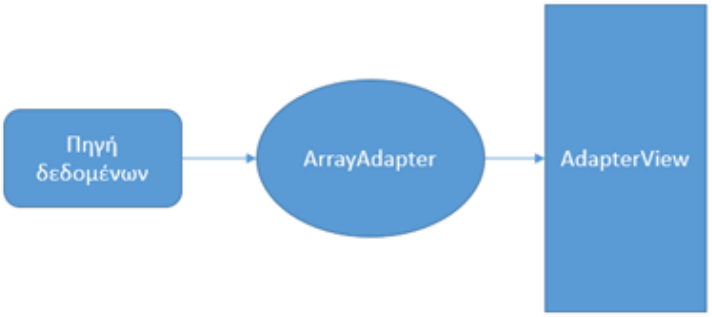
```
val timestamp = System.currentTimeMillis()
```

Όταν χρειάζεται να εμφανιστεί η ημερομηνία και η ώρα αποστολής ενός μηνύματος, είναι αναγκαία η μετατροπή των milliseconds τα οποία βρίσκονται αποθηκευμένα στο πεδίο timestamp, σε μια ευανάγνωστη για τους ανθρώπους μορφή. Αρχικά, δημιουργείται ένα αντικείμενο τύπου Date [16] με όρισμα τα milliseconds του πεδίου timestamp. Στη συνέχεια ορίζεται η επιθυμητή μορφοποίηση της ημερομηνίας με τη μέθοδο SimpleDateFormat [17], και τελικά εφαρμόζεται στο Date αντικείμενο.
Για παράδειγμα, για την τιμή timestamp=1610268540389, το result είναι : 10/01/2020 08:49

```
val date = Date(message.timestamp)
val sdf = SimpleDateFormat("dd/MM/yyyy HH:mm") // HH for 24h, hh for 12h
val result= sdf.format(date).toString()
dateText.text = result
```

4.6 Παρουσίαση δεδομένων μέσα σε λίστες

Για την παρουσίαση σύνθετων δεδομένων με τη μορφή λίστας, επιλέχθηκε η χρήση του ListView ή του Spinner σε συνδυασμό με τη δημιουργία κατάλληλων Custom ArrayAdapters. Ένα AdapterView είναι ένα ViewGroup τα παιδιά του οποίου καθορίζονται από έναν Adapter. Δηλαδή προβάλει (displays) τα αντικείμενα που φορτώνονται σε έναν Adapter. Συχνά χρησιμοποιούμενες υποκλάσσεις (subclasses) της AdapterView class είναι οι εξής: ListView, GridView, Spinner και Gallery [18]. Όταν πρόκειται να χρησιμοποιηθεί ένα AdapterView στο Android, ο Adapter είναι το απαραίτητο ενδιάμεσο εργαλείο το οποίο γεμίζει Views με δεδομένα, και πιο συγκεκριμένα μετατρέπει ένα ArrayList αντικειμένων σε αντίστοιχα Views τα οποία φορτώνονται στο εσωτερικό του ListView ή του Spinner. Ο ArrayAdapter όπως φαίνεται και στο Σχήμα 51, βρίσκεται μεταξύ της πηγής των δεδομένων η οποία μπορεί να είναι ένα ArrayList ή ένας Cursor, και της αντίστοιχης οπτικής αναπαράστασής της (AdapterView) η οποία μπορεί να είναι ένα ListView, ένα Spinner κ.α.. Ο ArrayAdapter είναι αυτός που καθορίζει ποιο είναι το array των δεδομένων και πώς πρέπει να μετατραπεί κάθε item του array σε ένα αντικείμενο τύπου View.



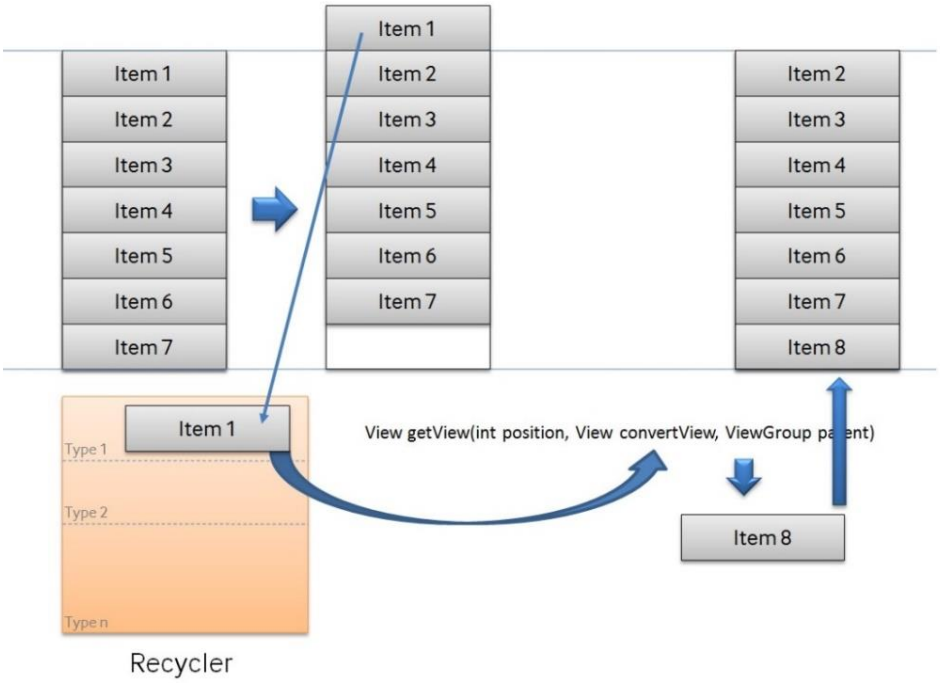
Σχήμα 51. Συσχέτιση μεταξύ δεδομένων, ArrayAdapter, και AdapterView

Για παράδειγμα, το booksFoundArrayList στο SearchActivity λειτουργεί ως πηγή δεδομένων για τον CustomAdapterBooks, ο οποίος με βάση αυτά τα αντικείμενα θα γεμίσει με κατάλληλο τρόπο τα Views που δημιούργησε, και θα τα στείλει ως επιστρεφόμενο αποτέλεσμα στο return της κλήσης της getView().

```
// δήλωση της μεταβλητής στην οποία θα αποθηκευτεί ο adapter
lateinit var adapter: CustomAdapterBooks

// ορισμός της πηγής δεδομένων
val booksFoundArrayList: ArrayList<Book> = ArrayList()

// ερώτημα στη βάση δεδομένων, και γέμισμα του arraylist με τα αποτελέσματα
db.collection("books")
    .whereEqualTo("title", wantedTitle)
    .whereEqualTo("swappedWithBookID", "")
    .get()
    .addOnSuccessListener { documents ->
        for (d in documents) {
            val book = d.toObject(Book::class.java)
            book.bookID = d.id
            booksFoundArrayList.add(book)
        }
        adapter = CustomAdapterBooks(this, booksFoundArrayList, object:listListener{
          // ...
        }

        // σύνδεση του listView με τον adapter
        books_found_list.adapter = adapter
    }
```

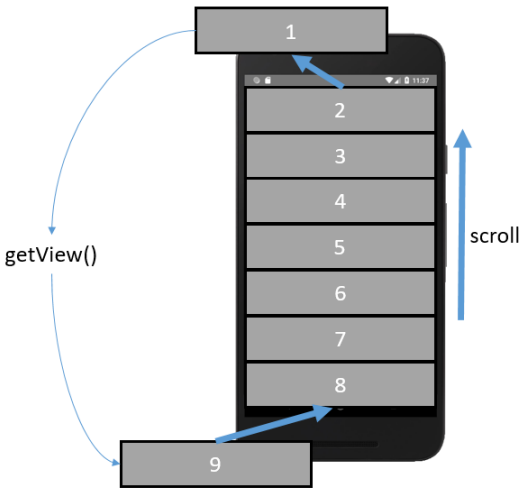
Όταν πρόκειται να παρουσιαστεί μία λίστα με δεδομένα στο χρήστη, όπως π.χ. κατά το γέμισμα του books_found_list ListView με βιβλία όταν ο χρήστης πραγματοποιεί αναζήτηση ενός τίτλου, ο Adapter αρχικά δημιουργεί μόνο τόσα Views όσα χρειάζονται για να γεμίσουν τα rows που χωράνε στο ύψος της οθόνης της εκάστοτε συσκευής. Αυτή η συμπεριφορά συμβαίνει από προεπιλογή στο Android για εξοικονόμηση μνήμης. Καθώς ο χρήστης κάνει scroll up στη λίστα, τα πρώτα (άνω) rows μετακινούνται προς τα πάνω βγαίνοντας εκτός οθόνης, και νέα rows εμφανίζονται στο κάτω μέρος. Τα rows που δεν είναι πλέον ορατά δεν καταστρέφονται, αλλά χρησιμοποιούνται για τη φόρτωση των δεδομένων ενός άλλου βιβλίου στο View που περιέχουν όπως φαίνεται στο Σχήμα 52, έτσι ώστε στη συνέχεια όταν γίνει scroll up να επανεμφανιστούν στο κάτω μέρος της οθόνης. Αυτή η κυκλική

διαδικασία επαναλαμβάνεται επιτυγχάνοντας ανακύκλωση των υπαρχόντων Views (View recycling), αντί για τη δημιουργία ενός νέου View για κάθε νέο αντικείμενο που πρέπει να εμφανιστεί.



Σχήμα 52. Ανακύκλωση Views σε ListView (πηγή: android.amberfog.com)

Το recycling είναι μέθοδος υλοποιημένη στον κώδικα του λειτουργικού συστήματος με σκοπό την εξοικονόμηση μνήμης και τη βελτίωση της αποδοτικότητας. Με αυτόν τον τρόπο για παράδειγμα, αν μία λίστα περιέχει 500 αντικείμενα Book τα οποία περιλαμβάνουν κείμενο για τα στοιχεία του κάθε βιβλίου και εικόνα για το εξώφυλλο, δεν θα δημιουργηθούν 500 views αμέσως με τη φόρτωση της λίστας, αλλά μόνο όσα χωράνε στην οθόνη (π.χ. 7 Views στη συσκευή του Σχήματος 53) συν ένα ή ελάχιστα ακόμη ώστε να επιτευχθεί η διαδικασία της εναλλαγής δεδομένων. Με κάθε κίνηση scroll up, τα Views που κρύβονται θα ανακυκλώνονται, δηλαδή θα γεμίζουν με τα περιεχόμενα του επόμενου Book με σκοπό την εμφάνιση στο επόμενο row. Το recycling χρησιμοποιείται με τη μέθοδο getView() του Adapter.



Σχήμα 53. Μικρή οθόνη κινητού που χωρά μόνο 7 Views

4.7 Δημιουργία Custom ArrayAdapter

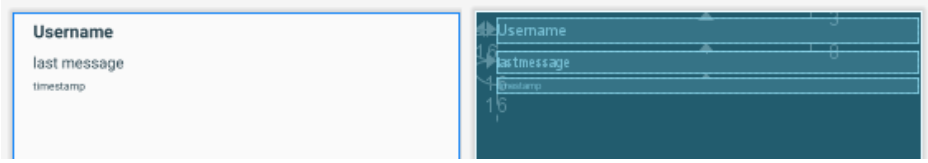
Παρακάτω περιγράφονται τα βήματα που ακολουθήθηκαν για τη δημιουργία του Custom ArrayAdapter των αντικειμένων Message. Με παρόμοιο τρόπο δημιουργήθηκαν και οι αντίστοιχοι Adapter για τα Book και τα BookCouple:

1. Δημιουργία custom XML layout: Επειδή τα προς εμφάνιση δεδομένα (βιβλία, μηνύματα κ.λ.π.) έχουν μια πιο σύνθετη μορφή από αυτήν που υποστηρίζει ένας απλός ArrayAdapter (μία γραμμή κειμένου), δημιουργήθηκαν εξατομικευμένα αρχεία XML, κατάλληλα για την εμφάνιση των περιεχομένων των αντικειμένων Message. Κάθε layout περιγράφει την μορφοποίηση ενός row (list item) της λίστας. Ουσιαστικά, αποτελεί ένα template για τα Views του ListView.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="3dp"
    android:background="@color/background_main">

    <TextView
        android:id="@+id/sender_username_textview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="3dp"
        android:text="@string/username"
        android:textColor="@color/colorDarker"
        android:textSize="16dp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/last_message_textview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:text="last message"
        android:textColor="@color/colorDarker"
        android:textSize="14dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="@id/sender_username_textview"
        app:layout_constraintTop_toBottomOf="@+id/sender_username_textview"
/>

    <TextView
        android:id="@+id/timestamp_textview"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="timestamp"
        android:textColor="@color/colorDarker"
        android:textSize="10dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="5dp"
        app:layout_constraintStart_toStartOf="@id/last_message_textview"
        app:layout_constraintTop_toBottomOf="@+id/last_message_textview" />
    </androidx.constraintlayout.widget.ConstraintLayout>
```

Το οπτικό αποτέλεσμα της παραπάνω σχεδίασης που φαίνεται στο Σχήμα 54, είναι το περιεχόμενο ενός row με list item ένα Message. Έχουν σχεδιαστεί τρία Textview τα οποία θα προβάλλουν το Username του αποστολέα, μια σύντομη προεπισκόπηση του κειμένου του μηνύματος, και τέλος την ημερομηνία και ώρα αποστολής του.



Σχήμα 54. Σχεδίαση ενός list item μέσα σε row

2. Συγγραφή κώδικα Kotlin για το αντίστοιχου αρχείο CustomAdapter: Κάθε CustomAdapter class επεκτείνει (extends) την abstract κλάση BaseAdapter, η οποία όπως δηλώνει και το όνομά της, είναι μια κλάση βάσης για την υλοποίηση Adapters που μπορούν να χρησιμοποιηθούν σε ListView (υλοποιώντας το εξειδικευμένο ListAdapter interface) και Spinner (υλοποιώντας το εξειδικευμένο SpinnerAdapter interface). Η επέκταση της κλάσης συνεπάγεται και υλοποίηση (implementation) των abstract μεθόδων της getCount(), getItem(), getItemId() και getView(). Η getCount() ενημερώνει το ListView για το πόσα list items πρέπει να εμφανίσει. Ο αριθμός αυτού του μεγέθους ισούται φυσικά με το πλήθος των αντικειμένων εντός του ArrayList που χρησιμοποιήθηκε ως πηγή. Η getItem() επιστρέφει το list item που βρίσκεται στο συγκεκριμένο position που της δόθηκε ως όρισμα κατά την κλήση της. Η getItemId() επιστρέφει το id του list item που βρίσκεται στο position που της δόθηκε ως όρισμα κατά την κλήση της. Παρακάτω περιγράφεται προγραμματιστικά η τοποθέτηση των τιμών των αντικειμένων (Messages) στα αντίστοιχα πεδία του row που παρουσιάστηκαν στο Σχήμα 43, με σκοπό τη δημιουργία των Views τα οποία θα τοποθετηθούν τελικά στο ListView conversation_list με τη μέθοδο getView(). Στη getView(), γίνεται η μετατροπή των αντικειμένων σε αντίστοιχα Views. Πιο συγκεκριμένα, η μέθοδος getView() καλείται κάθε φορά που το ListView πρέπει να εμφανίσει ένα νέο row στην οθόνη. Επιστρέφει το View του row που βρίσκεται στη θέση position του arraylist που χρησιμοποιεί ως πηγή δεδομένων. Το position χρησιμεύει στην ανάκτηση του αντίστοιχου αντικειμένου Message από το ArrayList των δεδομένων. Η getView() δέχεται ως όρισμα ένα View (convertView) το οποίο μπορεί να είναι null ή μη. Ο έλεγχος της τιμής του convertView έχει σκοπό την αναγνώριση των Views που πρόκειται να επαναχρησιμοποιηθούν. Εάν το View είναι null, τότε αρχικοποιείται ένα νέο View από το list_row_messages.xml αρχείο με inflation, και στη συνέχεια τοποθετούνται οι τιμές του νέου αντικειμένου Message στα αντίστοιχα components (TextViews). Στην αντίθετη περίπτωση (View!=null) γίνεται ανακύκλωση (recycling) ενός προηγουμένως δημιουργημένου View και δεν γίνεται inflation, αλλά απευθείας τοποθέτηση των τιμών του νέου αντικειμένου στα αντίστοιχα components.

```
class CustomAdapterMessages(private val context: Context, private val arrayList:
ArrayList<Message>,private var currentUserID : String?, private  val
showUsernames : Boolean) : BaseAdapter() {

    override fun getCount(): Int {
        return arrayList.size
    }
    override fun getItem(position: Int): Any {
        return position
    }
    override fun getItemId(position: Int): Long {
        return position.toLong()
    }
    override fun getView(position: Int, convertView: View?, parent:
                         ViewGroup): View? {
        var view: View? = convertView
//έλεγχος αν το τρέχον view είναι ανακυκλωμένο ή θα πρέπει να δημιουργηθεί νέο
        if (convertView == null) {
//εάν δεν πρόκειται για ανακυκλωμένο view, δημιουργείται νέο view με inflation
 view = LayoutInflater.from(context).inflate(R.layout.list_row_messages,
                                             parent, false)
        }
//αρχικοποιήσεις των components του XML row
val senderUsername : TextView = view!!.findViewById(R.id.sender_username_textview)
val message : TextView = view!!.findViewById(R.id.last_message_textview)
val timestamp : TextView = view!!.findViewById(R.id.timestamp_textview)
//γέμισμα των components του ViewHolder με τις τιμές του αντικειμένου Message
για του τρέχον position του ArrayList
        message.text = arrayList[position].text
        //...
        return view
    }
}
```

3. Δημιουργία ενός ArrayList, φόρτωσή του με αντικείμενα Message και στη συνέχεια αρχικοποίηση του Adapter με ανάθεση του ArrayList.

```
val messages: ArrayList<Message> = ArrayList()
messages.add(message)
val adapter: CustomAdapterMessages = CustomAdapterMessages(this,
                                     messages,currentUid,true)
```

4. Εφαρμογή (attach) του Adapter στο ListView

```
messages_list.adapter = adapter
```

Βελτιστοποίηση του κώδικα με χρήση View caching

Από τον παραπάνω μηχανισμό, προκύπτει το πρόβλημα της μειωμένης ταχύτητας του scrolling. Αυτό οφείλεται στις δηλώσεις των TextViews που περιέχονται στο XML layout για κάθε ένα row, μέσα

στην getView(), ανεξάρτητα από το αν πρόκειται για ανακυκλωμένο View ή όχι. Οι απαιτούμενες κλήσεις της μεθόδου findViewById() είναι χρονοβόρες και προκαλούν καθυστέρηση στο scrolling. Η παρουσίαση δεδομένων μέσα σε λίστες είναι ζωτικής σημασίας για την εφαρμογή αυτή, καθώς χρησιμοποιείται στα βιβλία που καταχωρούνται προς ανταλλαγή, στα αποτελέσματα της αναζήτησης ενός βιβλίου, και στα μηνύματα μιας συνομιλίας. Ως γνωστόν, κανένας χρήστης δε θέλει να χρησιμοποιεί μια αργή εφαρμογή, πόσω μάλλον όταν πρόκειται για τις κρίσιμες λειτουργίες που παρέχει. Για το λόγο αυτό, ο παραπάνω κώδικας χρειάστηκε βελτιστοποίηση. Το πρόβλημα αντιμετωπίστηκε με τη δημιουργία μιας βοηθητικής εμφωλευμένης (nested) κλάσης ViewHolder μέσα σε κάθε Custom Adapter Class [19]. Μέσα στην εκάστοτε κλάση ViewHolder δηλώνονται σε μεταβλητές όλα τα components του row και αποθηκεύονται απευθείας αναφορές στα View τους. Ακολουθεί η εμφωλευμένη κλάση ViewHolder, που βρίσκεται εντός του CustomAdapterMessages.

```
private class ViewHolder {
    lateinit var senderUsername: TextView
    lateinit var message: TextView
    lateinit var timestamp: TextView
}
```

Όταν δημιουργείται ένα View της λίστας για πρώτη φορά (συνθήκη convertView = = null), τότε μέσα στη getView(), τα components του row ως μέλη της ViewHolder, αρχικοποιούνται με τη βοήθεια της μεθόδου findViewById(). Στη συνέχεια το αντικείμενο ViewHolder ανατίθεται ως tag στο τρέχον view, δηλαδή στο convertView. Όταν πρόκειται να γίνει ανακύκλωση ενός προηγουμένως δημιουργημένου View (συνθήκη convertView != null), γίνεται ανάκτηση του αντικειμένου ViewHolder μέσα από το tag του convertView. Το αντικείμενο ViewHolder περιέχει τα αρχικοποιημένα components και έτσι δεν χρειάζεται χρήση της findViewById(), αλλά ο κώδικας συνεχίζει με το γέμισμα των TextViews με τις αντίστοιχες τιμές του αντικειμένου Message. Η κλάση ViewHolder δηλαδή, λειτουργεί ως cache των TextViews με σκοπό την αποφυγή πολλαπλών κλήσεων της findViewById() (εντός της getView()) η οποία τείνει να μειώνει την απόδοση (performance) της εφαρμογής. Για παράδειγμα, στη συσκευή του Σχήματος 42, εάν γινόταν scroll για προβολή όλων των βιβλίων μιας λίστας με 500 βιβλία, η μέθοδος findViewById() θα έπρεπε να κληθεί 500 φορές, ενώ μέσω της χρήσης view caching, θα κληθεί μόνο μια φορά για κάθε View που δημιουργήθηκε, δηλαδή ελάχιστα παραπάνω από 7.

Η μέθοδος getView() λοιπόν τροποποιήθηκε για να λάβει την τελική της μορφή.

```
override fun getView(position: Int, convertView: View?, parent: ViewGroup): View?
{
    var view: View? = convertView
    val viewHolder: ViewHolder //λειτουργεί ως cache των components του view

//έλεγχος για αν το τρέχον view είναι ανακυκλωμένο ή πρέπει να  δημιουργηθεί νέο
    if (convertView == null) {

//εάν δεν πρόκειται για ανακυκλωμένο view, δημιουργείται νέο view με inflation και
αρχικοποιήσεις των components του XML row
        viewHolder = ViewHolder()
        view =LayoutInflater.from(context).inflate(R.layout.list_row_messages,
               parent, false)
        viewHolder.senderUsername = view.findViewById(R.id.sender_username_textview)
        viewHolder.message = view.findViewById(R.id.last_message_textview)
```

```
        viewHolder.timestamp = view.findViewById(R.id.timestamp_textview)
        view.tag = viewHolder
    }
    else {

// το view προέρχεται από ανακύκλωση, άρα ανακτούμε το viewHolder που περιέχεται
ως tag στο convertView
        viewHolder = convertView.tag as ViewHolder
    }

//γέμισμα των components του ViewHolder με τις τιμές του αντικειμένου Message για
του τρέχον position του ArrayList
      viewHolder.senderUsername.text = arrayList[position].senderUsername
      viewHolder.message.text = arrayList[position].text
      val sdf = SimpleDateFormat("dd/MM/yyyy   HH:mm")
      val netDate = Date(arrayList[position].timestamp)
      val result= sdf.format(netDate).toString()
      holder.timestamp.text = result

    return view
}
```

4.8 Επίλογος

Στο κεφάλαιο αυτό, δόθηκε έμφαση σε ενδιαφέροντα ή δυσνόητα σημεία του κώδικα, περιγράφοντας αναλυτικά γιατί ήταν απαραίτητα, αλλά και πώς λειτουργούν. Παρουσιάστηκε επίσης η διαδικασία δημιουργίας της βάσης δεδομένων στο Cloud Firestore και επεξηγήθηκε ο τρόπος χειρισμού της μέσω των μεθόδων που προσφέρονται. Στόχος ήταν η παρουσίαση της φιλοσοφίας με βάση την οποία υλοποιήθηκαν διάφορες λειτουργίες της εφαρμογής, καθώς αρκετές από τις διαδικασίες που περιγράφηκαν χρησιμοποιήθηκαν επαναληπτικά και σε άλλα σημεία του κώδικα με μικρές διαφοροποιήσεις.

Κεφάλαιο 5ο: Συμπεράσματα και προτάσεις βελτίωσης

Με την επιτυχή πραγματοποίηση των αρχικών στόχων που τέθηκαν για την δημιουργία της ζητούμενης εφαρμογής για ανταλλαγή βιβλίων, επιβεβαιώθηκε η αρμονική συνεργατική χρήση της του Android Studio, της γλώσσας Kotlin, και της πλατφόρμας Firebase. Η λειτουργικότητα αρκετές φορές προήλθε από χρήση έτοιμων μεθόδων του Firebase ή του Firestore, εξοικονομώντας έτσι χρόνο που θα χρειαζόταν για τη δημιουργία τους και τον έλεγχο ορθής λειτουργίας τους. Η διαφορετική προσέγγιση της βάσης δεδομένων που παρέχει το Cloud Firestore μπορεί αρχικά να δημιούργησε κάποια σύγχιση καθώς εκλείπουν οι συμβατικές έννοιες των πινάκων και των κελιών τους. Ωστόσο, χάρη στην επαρκή τεκμηρίωση (documentation) που προσφέρεται στην επίσημη ιστοσελίδα αλλά και σε ιστότοπους σχετικούς με τον προγραμματισμό όπως το Stack Overflow, παρά την πρόσφατη δημιουργία του (Οκτώβριο 2017) μπόρεσε να καλύψει τις ανάγκες χειρισμού μιας βάσης δεδομένων μέσω ενός απλοϊκού σχεδιασμού της κονσόλας διαχείρισης, και επαρκούς πλήθους εύχρηστων μεθόδων. Η εξοικείωση με τα παραπάνω συνέβη γρηγορότερα απ' ότι ήταν αναμενόμενο, έπειτα από την κατανόηση των τριών κυρίων συστατικών, που δεν είναι άλλα από τα collections, documents, fields. Το μοναδικό μειονέκτημα που συναντήθηκε, ήταν η έλλειψη ευελιξίας στη διατύπωση ερωτημάτων (queries) προς τη βάση. Για παράδειγμα η χρήση του OR προς στιγμήν τουλάχιστον, δεν υποστηρίζεται για τιμές διαφορετικών πεδίων. Έτσι η ανάκτηση από τη βάση ενός document που να έχει x τιμή σε ένα field και y τιμή σε ένα άλλο field, δεν είναι εφικτή μέσω ενός ερωτήματος. Θα πρέπει να γίνει πρώτα ερώτημα για τα document που έχουν τιμή x στο ζητούμενο field, και σε περίπτωση επιτυχίας του ερωτήματος, εντός του SuccessListener του, να γίνει νέο ερώτημα στη βάση για τα document που έχουν τιμή y στο άλλο ζητούμενο field. Τέλος, θα πρέπει φυσικά τα document που επιστράφηκαν από τα δύο ερωτήματα να συγχωνευτούν, ώστε να αποτελούν το τελικό αποτέλεσμα του ερωτήματος. Αυτή η περίπλοκη και φλύαρη αλυσιδωτή διαδικασία εμφωλευμένων ερωτημάτων ωστόσο, στην SQL π.χ. θα μπορούσε να γραφεί πολύ ευκολότερα, μέσα σε μία γραμμή. Όσο αφορά το συνολικό μέγεθος του κώδικα, η χρήση της Kotlin συνείσφερε θετικά, καθώς διάφορα κομμάτια κώδικα, όπως οι μέθοδοι toString() των κλάσεων, παράγονται αυτόματα από τη γλώσσα. Ακόμη, στη συντομία του κώδικα συνείσφερε η άμεση πρόσβαση των κλάσεων στα components των xml αρχείων. Αυτό επετεύχθη διότι δεν χρειαζόταν η ανεύρεση του εκάστοτε View με βάση το id που του αποδόθηκε, όπως θα γινόταν με χρήση της Java, αλλά απευθείας, μέσω χρήσης του ίδιου του id.

Καθώς με την πάροδο του χρόνου το περιβάλλον συνήθως αλλάζει, απαιτείται προσαρμογή σε αυτό. Έτσι προκύπτει πως μια εν λειτουργία εφαρμογή λογισμικού, έχει σχεδόν πάντοτε περιθώρια ανάπτυξης και βελτίωσης. Ο τρόπος που οι χρήστες χρησιμοποιούν την εφαρμογή, συχνά συμβαίνει να μην είναι επακριβώς εκείνος που είχε προβλεφθεί. Ο τρόπος που αλληλεπιδρούν μεταξύ τους αποτελεί σημείο ενδιαφέροντος το οποίο θα πρέπει να παρατηρηθεί από τη συμπεριφορά τους, και να συμπληρωθούν πιθανά κενά λειτουργικότητας. Για παράδειγμα, είναι πιθανόν μελλοντικά οι χρήστες να ζητούν περαιτέρω πληροφορίες από το χρήστη με τον οποίο συναλλάσσονται, έτσι ώστε να αισθάνονται περισσότερο ασφαλής κατά τη διάρκεια της επικοινωνία τους, οδηγώντας έτσι στην ανάγκη δημιουργίας προσωπικού προφίλ για κάθε χρήστη, στο οποίο να παρέχονται και άλλα διαδικτυακά μέσα στα οποία μπορεί να εντοπιστεί, όπως το προφίλ του σε κάποιο κοινωνικό δίκτυο. Επίσης, στην οθόνη αναζήτησης βιβλίων, μπορεί να προστεθεί ένα Button το οποίο να λειτουργεί ως φίλτρο αποτελεσμάτων. Έτσι θα δίνεται στο χρήστη η δυνατότητα να αναζητήσει βιβλία λ.χ. μόνο σε άριστη κατάσταση. Ακόμη, στην ίδια οθόνη μπορεί να προστεθεί δίπλα σε κάθε βιβλίο των αποτελεσμάτων, ένα Button προσθήκης τους σε μια λίστα τύπου Wishlist, ώστε να αποθηκεύονται για μετέπειτα εξέτασή τους από το χρήστη.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] “Introducing Marketplace: Buy and Sell With Your Local Community”, *Facebook*, 2016. [Online] Available: https://about.fb.com/news/2016/10/introducing-marketplace-buy-and-sell-with-your-local-community/

[2] “Welcome Firebase to the Google Cloud Platform Team”, *Google Cloud Platform Blog*, 2014. [Online] Available: https://cloudplatform.googleblog.com/2014/10/welcome-firebase-to-google-cloud-platform.html

[3] “Cloud Firestore”, *Google Developers*, 2020. [Online] Available: https://firebase.google.com/docs/firestore

[4] Firestore documentation, Server-side encryption, *Google Developers*, 2020, [Online] Available: https://cloud.google.com/firestore/docs/server-side-encryption

[5] “Firebase Authentication”, *Google Developers*, 2020. [Online] Available: https://firebase.google.com/docs/auth

[6] Mobius Conference, Андрей Бреслав - Kotlin для Android: коротко и ясно, St. Petersburg, 2014

[7] “JetBrains readies JVM-based language”, *InfoWorld*, 2011. [Online] Available: https://www.infoworld.com/article/2622405/jetbrains-readies-jvm-based-language.html

[8] “Kotlin on Android. Now official”, *JetBrains blog*, 2017. [Online] Available: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/

[9] “JetBrains readies JVM-based language”, *Android Developers Blog*, 2019. [Online] Available: https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html

[10] JetBrains, Kotlin Language Documentation, 2020, [Online] Available: https://kotlinlang.org/docs/kotlin-pdf.html

[11] “Τεχνολογία Λογισμικού, Θεωρία και Πράξη”, *Shari Lawrence Pfleeger*, Κλειδάριθμος, 2008

[12] Peter Sommerhoff, *Kotlin for Android App Development*, Pearson, 2019

[13] Luke Wroblewski, *Web Form Design: Filling in the Blanks*, Rosenfeld Media, 2008

[14] Don Norman, *The design of everyday things*, The Perseus Books Group, 2013

[15] “Blob”, Firebase Documentation, *Google Developers*, 2020, [Online] Available: https://firebase.google.com/docs/reference/android/com/google/firebase/firestore/Blob

[16] Class Date, Java 8 Documentation, *Oracle*, [Online] Available: https://docs.oracle.com/javase/8/docs/api/java/util/Date.html

[17] Class SimpleDateFormat, Java 7 Documentation, *Oracle*, [Online] Available: https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

[18] Android Documentation, AdapterView, *Android Developers*, [Online] Available: https://developer.android.com/reference/android/widget/ListView

[19] Using an ArrayAdapter with ListView, *Codepath*, [Online] Available: https://guides.codepath.com/android/Using-an-ArrayAdapter-with-ListView