ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

Τίτλος Πτυχιακής Εργασίας	***Μελέτη και ανάπτυξη εφαρμογής ακαδημαϊκής ταυτότητας σε φορητή συσκευή Android με χρήση PaaS*** ***Academic ID mobile app using PaaS: A study and implementation***
Ονοματεπώνυμο Φοιτητή	Σπυρίδων Κόκκας
Πατρώνυμο	Ιωάννης
Αριθμός Μητρώου	Π19073
Επιβλέπων	Αναπληρωτής Καθηγητής Ε. Σακκόπουλος

Ημερομηνία Παράδοσης **Σεπτέμβριος 2024**

Copyright ©

Επιτελική Σύνοψη

Το αντικείμενο της παρούσας πτυχιακής εργασίας είναι η ανάπτυξη λογισμικού για κινητές συσκευές με λειτουργικό σύστημα Android, με θέμα την "Ηλεκτρονική κάρτα φοιτητή στο κινητό" για χρήση υπηρεσιών Πανεπιστημίου. Η εφαρμογή αναπτύχθηκε χρησιμοποιώντας τη γλώσσα προγραμματισμού Kotlin στο περιβάλλον ανάπτυξης (IDE) Android Studio. Ο κύριος σκοπός ήταν η εξοικείωση με τις νεότερες τεχνολογίες που αφορούν την ανάπτυξη εφαρμογών για το λειτουργικό Android, με έμφαση στις τεχνολογίες Jetpack Compose και Kotlin Flow Coroutines.

Το Jetpack Compose είναι ένα εργαλείο ανάπτυξης UI/UX που χρησιμοποιεί ένα δηλωτικό API για τον ορισμό της διεπαφής χρήστη, αντικαθιστώντας το παραδοσιακό XML που χρησιμοποιείται για χρόνια στον σχεδιασμό Android εφαρμογών.

Το Kotlin Flow, από την άλλη, είναι ένα μέρος της βιβλιοθήκης Coroutines της Kotlin, το οποίο χρησιμοποιείται για την επεξεργασία ασύγχρονων ροών δεδομένων. Ο συνδυασμός αυτών των τεχνολογιών επιτρέπει τη δημιουργία εφαρμογών που αντιδρούν σε αλλαγές δεδομένων σε πραγματικό χρόνο.

Η εφαρμογή που ανέπτυξα κάνει χρήση αυτών των τεχνολογιών και έχει τη δυνατότητα να προβάλλει ένα σύγχρονο UI και ένα backend που ανταποκρίνεται στις αλλαγές δεδομένων στη βάση. Η εφαρμογή περιλαμβάνει:

- **Σελίδα Σύνδεσης:** Όπου οι χρήστες μπορούν να εισάγουν τα διαπιστευτήριά τους για να συνδεθούν.
- **Σελίδα Αναζήτησης Ακαδημαϊκής Ταυτότητας:** Όπου οι φοιτητές μπορούν να αναζητήσουν την ακαδημαϊκή τους ταυτότητα εισάγοντας τον Αριθμό Μητρώου και το επώνυμό τους.
- **Σελίδα Προβολής Πληροφοριών Ταυτότητας:** Όπου εμφανίζονται οι πληροφορίες της ακαδημαϊκής ταυτότητας του φοιτητή.
- **Οθόνη Εκκίνησης:** Μια σύντομη οθόνη κατά την έναρξη της εφαρμογής.

Η βάση δεδομένων που διαχειρίζεται τις συνδέσεις, τις εγγραφές χρηστών και την αποθήκευση των ταυτοτήτων, υλοποιήθηκε με τη χρήση της Firebase. Συγκεκριμένα, για τη σύνδεση και εγγραφή χρησιμοποιήθηκε η Firebase Authentication, επιτρέποντας σύνδεση μόνο με email και κωδικό. Για την αποθήκευση των δεδομένων των ταυτοτήτων χρησιμοποιήθηκε το Firebase Firestore, όπου κάθε αποθηκευμένη ταυτότητα περιλαμβάνει τα αντίστοιχα πεδία πληροφοριών.

Οι χρήστες μπορούν να αποθηκεύσουν το πάσο τους για γρηγορότερη πρόσβαση σε μελλοντικές χρήσεις της εφαρμογής. Η εφαρμογή έχει επιδείξει επιτυχημένη λειτουργία, προσφέροντας στους φοιτητές μια σύγχρονη και εύχρηστη λύση για την προβολή και διαχείριση των ακαδημαϊκών τους ταυτοτήτων.

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τους γονείς μου καθώς και του φίλους μου που ήταν δίπλα μου όλα αυτά τα χρόνια και με υποστήριζαν συνεχώς κατά τη διάρκεια των σπουδών μου. Επίσης θα ήθελα να ευχαριστήσω τον Κ. Σακκόπουλο για την ευκαιρία να υλοποιήσω μια πτυχιακή πάνω στο αντικείμενο που επιθυμούσα και φυσικά θα ήθελα να ευχαριστήσω και όλους τους καθηγητές του τμήματος για τις γνώσεις που μου πρόσφεραν κατά τη διάρκεια των σπουδών μου.

Σεπτέμβριος 2024

Σπυρίδων Κόκκας

Περιεχόμενα

Πίνακας Περιεχομένων

Κεφάλαιο 1ο

1 Εισαγωγή

Η πτυχιακή αυτή εργασία αφορά την ανάπτυξη λογισμικού για κινητές συσκευές με λειτουργικό σύστημα Android, με αντικείμενο την υλοποίηση της ηλεκτρονικής κάρτας φοιτητή για το κινητό με σκοπό την χρήση των υπηρεσιών του Πανεπιστημίου.

Η ανάπτυξη εφαρμογών για κινητές συσκευές είναι ένας κλάδος με συνεχή εξέλιξη που απαιτεί να είσαι ενήμερος για τις νέες διαθέσιμες τεχνολογίες και μεθόδους που βοηθούν στην καλύτερη απόδοση και χρηστικότητα των εφαρμογών. Είναι ένας κλάδος με ολοένα και αυξανόμενες απαιτήσεις , καθώς τα κινητά τα τελευταία χρόνια έχουν μπει και παίζουν τεράστιο ρόλο στην καθημερινότητα μας και όλες πλέον οι εταιρίες κλπ. θέλουν να ενταχθούν στον κόσμο των εφαρμογών του κινητού με τον βέλτιστο δυνατό τρόπο.

Στόχος σε αυτή την εργασία είναι η μελέτη των νέων τεχνολογιών και η σωστή και αποδοτική χρήση τους για την δημιουργία μια λειτουργικής, αποδοτικής και εύχρηστης εφαρμογής για να την διευκόλυνση των φοιτητών με τις υπηρεσίες του πανεπιστημίου που απαιτούν την χρήση της ακαδημαϊκής ταυτότητας.

1.1 Περιγραφή του υπό μελέτη προβλήματος

Το υπό μελέτη πρόβλημα είναι η ανάγκη των φοιτητών για εύκολη και γρήγορη πρόσβαση στην ακαδημαϊκή τους ταυτότητα χωρίς την ανάγκη να έχουν μαζί τους το φυσικό αντίγραφο. Το φυσικό αντίγραφο , εφόσον είναι φτιαγμένο από πλαστικό έπεται στη φθορά του χρόνου και της χρήσης, επιπλέον είναι πιθανό να πέσει θύμα κλοπής. Όλα αυτά δείχνουν ότι η ακαδημαϊκή ταυτότητα είναι πολύ πιθανό να χρειαστεί αλλαγή μέχρι τη λήξη της , κάτι που απαιτεί χρόνο από μέρους του φοιτητή καθώς και χρήμα διότι η πρώτη έκδοση είναι δωρεάν ενώ οι επόμενες πληρώνονται. Η ψηφιοποίηση της ταυτότητας δίνει τη λύση σε αυτά τα προβλήματα καθώς ο φοιτητής έχει τη δυνατότητα να τη κουβαλάει παντού μαζί του , έχοντας απλά μια κινητή συσκευή στη διάθεση του χωρίς να υπάρχει η ανάγκη για το φυσικό αντίγραφο.

Η εφαρμογή που αναπτύχθηκε στα πλαίσια αυτής της πτυχιακής εργασίας, δίνει τη δυνατότητα στους φοιτητές να αποθηκεύουν και να προβάλλουν την ακαδημαϊκή τους ταυτότητα από το κινητό τους, όπου και να βρίσκονται. Παρέχει μια εύχρηστη και σύγχρονη διεπαφή χρήστη (UI) καθώς και έναν αποδοτικό και τελευταίας τεχνολογίας τρόπο διαχείρισης δεδομένων που ανταποκρίνεται στις αλλαγές σε πραγματικό χρόνο.

1.2 Σκοπός και στόχοι της εργασίας

Ο σκοπός αυτής της πτυχιακής εργασίας είναι η ανάπτυξη μιας εφαρμογής για κινητές συσκευές που "τρέχουν" το λειτουργικό android, η οποία θα δίνει στους φοιτητές τη δυνατότητα να έχουν την ακαδημαϊκή τους ταυτότητα στην κινητή του συσκευή.

Η εφαρμογή έχει ως στόχο να αντικαταστήσει την ανάγκη της πλαστικής ταυτότητας, προσφέροντας έτσι πιο γρήγορη, εύκολη και ευέλικτη πρόσβαση στην ταυτότητα του αλλά και στις υπηρεσίες του πανεπιστημίου, μειώνοντας ταυτόχρονα το περιβαλλοντικό αποτύπωμα καθώς μειώνεται η χρήση του πλαστικού. Μέσω αυτής της εφαρμογής οι φοιτητές θα μπορούν να έχουν άμεση πρόσβαση στην ακαδημαϊκή τους ταυτότητα και σε όλες τις πληροφορίες που τη συνοδεύουν (Φωτογραφία , αριθμός μητρώου , ονοματεπώνυμο κλπ.) έχοντας μαζί τους μόνο τη κινητή τους συσκευή.

Οι βασικοί στόχοι της εργασίας περιλαμβάνουν:

- **Εξοικείωση με Σύγχρονες Τεχνολογίες Ανάπτυξης Εφαρμογών Android:**
  - Μελέτη, κατανόηση και χρήση των τεχνολογιών Jetpack Compose και Kotlin Flow Coroutines.
  - Κατανόηση των πλεονεκτημάτων και των δυνατοτήτων που προσφέρουν αυτές οι νέες τεχνολογίες στη δημιουργία δυναμικών και αποδοτικών εφαρμογών.
- **Ανάπτυξη Μιας Λειτουργικής Εφαρμογής:**
  - Δημιουργία μιας εφαρμογής με σύγχρονο και φιλικό προς το χρήστη UI, που επιτρέπει την προβολή και διαχείριση της φοιτητικής ταυτότητας.
  - Χρήση των τελευταίων τεχνολογιών για την ανάπτυξη της διεπαφής χρήστη (UI) καθώς και για τη διαχείριση των δεδομένων
- **Ενσωμάτωση Υπηρεσιών Firebase:**
  - Χρήση της Firebase Authentication για την ασφαλή σύνδεση και εγγραφή χρηστών, επιτρέποντας μόνο την είσοδο με email και κωδικό.
  - Αποθήκευση και διαχείριση των δεδομένων των φοιτητικών ταυτοτήτων με τη χρήση του Firebase Firestore.
- **Ανάπτυξη και Υλοποίηση Αποδοτικών Λειτουργιών Αναζήτησης και Αποθήκευσης:**
  - Πραγματοποίηση της αναζήτησης ταυτότητας με βάση τον Αριθμό Μητρώου και το επώνυμο του φοιτητή.
  - Δυνατότητα αποθήκευσης της ταυτότητας για γρήγορη πρόσβαση κατά το άνοιγμα της εφαρμογής.

Μέσα από την υλοποίηση αυτών των στόχων , η εργασία θα προσφέρει μια ολοκληρωμένη λύση για τη μεταφορά της φοιτητής ταυτότητας από το κλασσικό πλαστικό σε μια σύγχρονη και εξελιγμένη ψηφιακή λύση. Ταυτόχρονα δίνει τη δυνατότητα για εξοικείωση με τις τελευταίες τεχνολογίες στο κόσμο του Android αλλά και των κινητών συσκευών γενικότερα.

1.3 Βασικοί ορισμοί

1.3.1 Τεχνολογίες και Εργαλεία

1.3.1.1 Kotlin

Η Kotlin είναι μια μοντέρνα αλλά ολοκληρωμένη γλώσσα προγραμματισμού που αναπτύχθηκε από την JetBrains και χρησιμοποιείται κυρίως και πολύ συχνά για την ανάπτυξη εφαρμογών Android. Προσφέρει μεγαλύτερη ασφάλεια , συντομότερο κώδικα και συμβατότητα με τη Java καθιστώντας τη την ιδανική επιλογή για την ανάπτυξη ενός αξιόπιστου και συντηρήσιμου λογισμικού.

Εικόνα 1 Λογότυπο Kotlin

1.3.1.2 Android Studio

Το Android Studio είναι το επίσημο IDE (Integrated Development Environment) της Google για την ανάπτυξη εφαρμογών για το λειτουργικό Android. Περιλαμβάνει έναν επεξεργαστή κώδικα, εργαλεία κατασκευής και έναν διαχειριστή πακέτων, καθώς και πολλά άλλα εργαλεία που βοηθούν στην πιο αποδοτική και γρήγορη ανάπτυξη και επεξεργασία του κώδικα.

Εικόνα 2 Λογότυπο Android Studio

1.3.1.3 Jetpack Compose

Το Jetpack Compose αποτελεί το πιο σύγχρονο και τελευταίας τεχνολογίας εργαλείο ανάπτυξης και σχεδίασης UI/UX για Android. Σε αντίθεση με την προηγούμενη τεχνολογία UI την XML , το Jetpack Compose προσφέρει πιο εύκολη , ευέλικτη και δυναμική δημιουργία διεπαφών χρήστη με λιγότερο και πιο κατανοητό κώδικα ενώ επίσης έχει και το προνόμιο της ευκολότερης συντήρησης.

Εικόνα 3 Λογότυπο Jetpack Compose

1.3.1.4 Kotlin Flow Coroutines

Το Kotlin Flow είναι τμήμα της βιβλιοθήκης Coroutines της Kotlin , που χρησιμοποιείται για τη διαχείριση ασύγχρονων ροών δεδομένων. Είναι ένα API που αναπτύχθηκε από την JetBrains και έχει ως σκοπό να αντικαταστήσει την RxJava που χρησιμοποιούνταν στο Android.

Εικόνα 4 - Kotlin Flows

1.3.1.5 Firebase

Το Firebase είναι προϊόν της Google με σκοπό να βοηθήσει τους προγραμματιστές να δημιουργήσουν , διαχειριστούν και να αναπτύξουν της εφαρμογές τους. Το γεγονός ότι δεν απαιτεί κώδικα είναι ένα τεράστιο πλεονέκτημα καθώς προσφέρει μια μεγάλη ευκολία στη χρήση. Προσφέρει υπηρεσίες σε εφαρμογές Android , iOS , web και Unity. Προσφέρει Cloud αποθήκευση δεδομένων καθώς λειτουργεί σαν NoSQL βάση δεδομένων.

Εικόνα 5 - Λογότυπο Firebase

1.3.2 Ορισμοί και Εφαρμογές

1.3.2.1 Ηλεκτρονική Κάρτα φοιτητή

Η ηλεκτρονική κάρτα φοιτητή είναι μια ψηφιακή αναπαράσταση της κλασσικής πλαστικής κάρτας που διανέμεται από το υπουργείο στους φοιτητές και επέχει θέση ταυτότητας εντός του χώρου του πανεπιστημίου. Στην ηλεκτρονική της μορφή προσφέρεται η ευκολία της πρόσβασης στις πανεπιστημιακές υπηρεσίες ενώ επίσης διευκολύνει την ταυτοποίηση του φοιτητή.

1.3.2.2 User Authentication (Αυθεντικοποίηση Χρήστη)

Η αυθεντικοποίηση χρήστη είναι η διαδικασία με την οποία επαληθεύουμε την ταυτότητα του χρήστη προκειμένου να του επιτραπεί η είσοδος σε μια εφαρμογή ή υπηρεσία. Στη εργασία αυτή , η αυθεντικοποίηση γίνεται με τη χρήση της Firebase Authentication κάνοντας χρήση email και κωδικού πρόσβασης.

1.3.2.3 Data Storage (Αποθήκευση Δεδομένων)

Η αποθήκευση δεδομένων αναφέρεται στη διαδικασία προσθήκης και διατήρησης δεδομένων σε μια βάση δεδομένων. Συγκεκριμένα , το Firebase Firestore χρησιμοποιείται στην παρούσα εργασία για την αποθήκευση των δεδομένων των φοιτητικών ταυτοτήτων.

1.3.2.4 Διεπαφή Χρήστη (User Interface – UI)

Η διεπαφή χρήστη είναι το τμήμα της εφαρμογής με το οποίο αλληλοεπιδρά ο χρήστης. Το Jetpack Compose χρησιμοποιείται για τη δημιουργία μιας σύγχρονης διεπαφής χρήστη , που επιτρέπει την εύκολη και αποδοτική πλοήγηση στην εφαρμογή.

1.3.3 Μεθοδολογίες

1.3.3.1 Δηλωτικός Προγραμματισμός (Declarative Programming)

Ο δηλωτικός προγραμματισμός είναι μιας μέθοδος προγραμματισμού όπου ο προγραμματιστής περιγράφει το τελικό αποτέλεσμα που επιθυμεί χωρίς το τρόπο επίτευξης του αποτελέσματος αυτού. Το Jetpack Compose ακολουθεί αυτή τη μέθοδο προγραμματισμού, επιτρέποντας έτσι της εύκολη δημιουργία και διαχείριση διεπαφών χρήστη.

1.3.3.2 Ασύγχρονος προγραμματισμός (Asynchronous Programming)

Ο ασύγχρονος προγραμματισμός είναι μια μέθοδος προγραμματισμού που επιτρέπει την εκτέλεση λειτουργιών χωρίς να δεσμεύει/μπλοκάρει το κύριο νήμα (thread) της εφαρμογής. Το Kotlin Flow χρησιμοποιείται για την επεξεργασία των ασύγχρονων αυτών ροών δεδομένων, εξασφαλίζοντας έτσι ότι η εφαρμογή μπορεί ανταπεξέλθει στην επεξεργασία των δεδομένων σε πραγματικό χρόνο.

Με αυτές τις βασικές έννοιες και τεχνολογίες η πτυχιακή εργασία θέτει τα θεμέλια για την ανάπτυξη μιας σύγχρονης και λειτουργικής εφαρμογής για την Ηλεκτρονική Κάρτα Φοιτητή αξιοποιώντας όλες τις τελευταίες τεχνολογίες και εξελίξεις στον τομέα της ανάπτυξης εφαρμογών android , συμβαδίζοντας με την αγορά εργασίας και τις ταχύτατα αναπτυσσόμενες απαιτήσεις για κινητές συσκευές android.

1.4 Παραδοτέα της εργασίας

Τα παραδοτέα της εργασίας αποτελούνται από :

1. Την τεκμηρίωση της πτυχιακής εργασίας , μαζί με την βιβλιογραφία , τα αποτελέσματα, τον κώδικα της εφαρμογής καθώς και εικόνες από τη λειτουργία τις εφαρμογής.
2. Το αρχείο .zip που περιέχει τον λογισμικό που ανέπτυξα, συγκεκριμένα περιέχει ολόκληρο το κώδικα της εφαρμογής μαζί με όποια συνοδευτικά αρχεία.
3. Το αρχείο .apk που είναι το αρχείο εγκατάστασης της εφαρμογής για συσκευές με λειτουργικό android.
4. Ένα βίντεο επίδειξης της ορθής λειτουργίας της εφαρμογής , στο οποίο παρουσιάζονται αναλυτικά όλες οι λειτουργίες που έχω αναπτύξει.
5. Η βάση δεδομένων σε μορφή αρχείου .sql

1.5 Δομή της εργασίας

Στα επόμενα κεφάλαια θα παρουσιαστούν αναλυτικά όλες οι τεχνολογίες που έχουν χρησιμοποιηθεί μαζί με εικόνες για ευκολότερη κατανόηση, θα παρουσιαστούν Screenshots από το κώδικα που έχω αναπτύξει μαζί με αναλυτική επεξήγηση της κάθε εικόνας, καθώς και εικόνες από την λειτουργία της εφαρμογής.

Συγκεκριμένα:

Κεφάλαιο 2: Στο δεύτερο κεφάλαιο θα παρουσιάσω την γλώσσα προγραμματισμού Kotlin, τα Kotlin Coroutine Flows, θα αναλύσω με λεπτομέρεια την τεχνολογία του Jetpack Compose καθώς και την Firebase.

Κεφάλαιο 3: Στο τρίτο κεφάλαιο θα παρουσιάσω τη μεθοδολογία σχεδίασης, δηλαδή θα γίνει καταγραφή και ανάλυση των λειτουργικών προδιαγραφών, θα παρουσιαστεί η αρχιτεκτονική λύση με διάγραμμα UML, καθώς και εικόνες από τη πρωτότυπη σχεδίαση μέσω hi-fi story board.

Κεφάλαιο 4: Στο τέταρτο κεφάλαιο θα γίνει παρουσίαση της εφαρμογής σε λειτουργία με screenshots και επεξήγηση κάθε οθόνης, καθώς και παρουσίαση του κώδικα της εφαρμογής με αναλυτικό σχολιασμό.

Κεφάλαιο 5: Στο πέμπτο κεφάλαιο θα παρουσιαστούν μερικά από τα πιο σημαντικά σημεία ανάπτυξης της εφαρμογής.

Κεφάλαιο 6: Στο έκτο κεφάλαιο θα παρουσιάσω τα συμπεράσματα της πτυχιακής καθώς και μελλοντικές επεκτάσεις της εφαρμογής.

Κεφάλαιο 7: Στο έβδομο κεφάλαιο θα βρίσκονται οι βιβλιογραφικές πηγές που χρησιμοποιήθηκαν για την ανάπτυξη τόσο της εφαρμογής αλλά και της τεκμηρίωσης αυτής.

Κεφάλαιο 2ο

2 Επισκόπηση του χώρου

2.1 Γλώσσα προγραμματισμού Kotlin

Η Kotlin είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού υψηλού επιπέδου που είναι σχεδιασμένη να ενσωματώνει πλήρως τη Java. Αναπτύχθηκε πλήρως από τη JetBrains και έχει πάρει επιρροές από διάφορες γλώσσες προγραμματισμού όπως είναι η Java, Scala, C#, Groovy κλπ. και είναι πλήρως συμβατή με Java και Android, αποτελώντας την επίσημη γλώσσα του Android. Η Kotlin αξιοποιεί τις υπάρχουσες βιβλιοθήκες για την Εικονική Μηχανή της Java (JVM), για Android και web browser. Ένα πρόγραμμα Kotlin μπορεί να αναπτυχθεί σε περιβάλλον ανάπτυξης IntelliJ IDEA , Android Studio, Eclipse, NetBeans, Visual Studio Code κλπ. [8]

Ο κώδικας της Kotlin μπορεί να ενσωματωθεί σε κώδικα Java ενώ μπορεί να γίνει και το ακριβώς αντίθετο. Με λίγα λόγια η Java και Kotlin έχουν αλληλένδετους κώδικες που μπορούν να αντικαταστήσουν ο ένας τον άλλο χωρίς ιδιαίτερες προσαρμογές.

Επιπλέον:

Η Kotlin παρέχει ισχυρή υποστήριξη για ασφαλή διαχείριση null, κάτι που βοηθά στην αποφυγή σφαλμάτων που σχετίζονται με null pointer exceptions. Χρησιμοποιεί τύπους όπως το Nullable και το Non-Nullable, δίνοντας τη δυνατότητα στους προγραμματιστές να χειρίζονται πιο εύκολα και με ασφάλεια τις περιπτώσεις όπου μπορεί να υπάρχουν null τιμές.

Ο τρόπος γραφής της Kotlin είναι πιο περιεκτικός που σημαίνει ότι για να παράγεις το ίδιο αποτέλεσμα θα χρειαστείς λιγότερες γραμμές κώδικα σε σχέση με κώδικα με το ίδιο αποτέλεσμα αλλά σε Java. Διαθέτει όλες τις σύγχρονες τεχνικές και βιβλιοθήκες που πιθανόν να χρειαστεί κάποιος και είναι από τις πιο γρήγορα υιοθετούμενες γλώσσες προγραμματισμού , καθώς όλο και πιο πολλοί προγραμματιστές καθώς και εταιρίες όπως η Google , την χρησιμοποιούν για την ανάπτυξη των εφαρμογών τους.

2.2 Kotlin Flows

Τα Kotlin Flows είναι ένας τρόπος να διαχειρίζεσαι τα δεδομένα που λαμβάνονται σταδιακά , σε μια ροή. Δηλαδή επιτρέπουν τον ασύγχρονο χειρισμό των δεδομένων.

Τα Flows παράγουν δεδομένα μόνο όταν είναι απαραίτητο, κάτι το οποίο έχει ως αποτέλεσμα την εξοικονόμηση των πόρων του συστήματος. Δηλαδή η ροή των δεδομένων έρχεται σε εφαρμογή μόνο όταν υπάρχει κάποιο μέρος κώδικα που τη διαβάζει/χρησιμοποιεί.

Σου δίνουν επιπλέον τη δυνατότητα να διαχειριστείς ή και να μετασχηματίσεις τα δεδομένα που έρχονται, δηλαδή τη ροή , και να φιλτράρεις τα δεδομένα πριν τα αξιοποιήσεις κρατώντας μόνο τα ωφέλιμα για την εφαρμογή δεδομένα. Ενώ ταυτόχρονα προσφέρουν μεγάλη ευκολία στη διαχείριση των σφαλμάτων που μπορεί να προκύψουν στη ροή των δεδομένων.

Συμπερασματικά τα Kotlin Flows επιτρέπουν τον χειρισμό συνεχόμενων δεδομένων ή μεγάλων λιστών , χωρίς να επιβραδύνεις τη μνήμη της εφαρμογής. [9]

2.3 Jetpack Compose

Το **Jetpack Compose** είναι ένα μοντέρνο, δηλωτικό UI framework που αναπτύχθηκε από την Google για τη δημιουργία διεπαφών χρήστη για εφαρμογές Android. Σχεδιάστηκε για να αντικαταστήσει τις παραδοσιακές τεχνικές ανάπτυξης UI που βασίζονταν στο XML και να διευκολύνει τους προγραμματιστές να δημιουργούν δυναμικές και αποδοτικές εφαρμογές. Το Jetpack Compose είναι ανοιχτού κώδικα και βασίζεται στην Kotlin, μια γλώσσα προγραμματισμού που έχει ανέβει πολύ η δημοτικότητα της ανάμεσα στους Android Developers καθώς αποτελεί μια γλώσσα προγραμματισμού που προσφέρει μεγάλη ευκολία στη χρήση της αλλά και σαφήνεια, κάτι το οποίο βοηθάει τρομερά την καλύτερη και αποδοτικότερη ανάπτυξη των εφαρμογών. Ανακοινώθηκε για πρώτη φορά τον Μάιο του 2019 και έγινε επίσημα διαθέσιμο στο κοινό τον Ιούλιο του 2021, σηματοδοτώντας ένα νέο κεφάλαιο στην ανάπτυξη εφαρμογών για Android.[17]

Το τέλος της XML: Απλοποίηση της ανάπτυξης UI

Το Jetpack Compose εισάγει έναν νέο τρόπο σχεδιασμού των διεπαφών χρήστη, αντικαθιστώντας την παραδοσιακή χρήση του XML, που υπήρξε ο κύριος τρόπος δημιουργίας διεπαφών σε εφαρμογές Android για πολλά χρόνια. Αν και το XML πρόσφερε μια σταθερή βάση για την ανάπτυξη UI, παρουσίαζε σημαντικές προκλήσεις, όπως η ανάγκη για πολυπλοκότητα στον κώδικα και η ύπαρξη ατελείωτων γραμμών που έκαναν τη συντήρηση πιο δύσκολη. Το Jetpack Compose, από την άλλη πλευρά, προσφέρει μια πιο απλοποιημένη μορφή κώδικα, που μειώνει τον όγκο των γραμμών κώδικα και κάνει το UI development πιο κατανοητό και εύκολο στη διαχείριση.

Η δηλωτική φύση του Compose είναι αυτό που το καθιστά πραγματικά ξεχωριστό. Στον παραδοσιακό τρόπο ανάπτυξης, οι προγραμματιστές έπρεπε να ορίζουν με μεγάλη ακρίβεια πώς θα εμφανιστεί και θα λειτουργήσει κάθε στοιχείο του UI. Με το Compose, ο κώδικας επικεντρώνεται στο **αποτέλεσμα** και όχι στις διαδικασίες που χρειάζονται για

την επίτευξή του. Αυτός ο δηλωτικός τρόπος προγραμματισμού επιτρέπει στους προγραμματιστές να γράφουν λιγότερο κώδικα και να επικεντρώνονται περισσότερο στην λογική της εφαρμογής, αντί για τις λεπτομέρειες υλοποίησης του UI.

Η δύναμη της Kotlin

Ένας από τους κύριους λόγους για την επιλογή του Kotlin ως βάση για το Jetpack Compose είναι η εκφραστικότητα και η ευελιξία της γλώσσας. Η Kotlin έχει σχεδιαστεί για να μειώνει την πολυπλοκότητα που υπάρχει σε γλώσσες όπως η Java, κάνοντας τον κώδικα πιο καθαρό και εύκολα διαχειρίσιμο. Το Jetpack Compose αξιοποιεί πλήρως τις δυνατότητες της Kotlin για να προσφέρει ένα ευέλικτο περιβάλλον για την ανάπτυξη UI. Οι προγραμματιστές μπορούν να χρησιμοποιούν χαρακτηριστικά της Kotlin, όπως **higher-order functions**, **lambda expressions και coroutines flows** για να δημιουργούν UI που προσαρμόζεται δυναμικά στις αλλαγές δεδομένων. Αυτό σημαίνει ότι το UI μπορεί να ενημερώνεται σε πραγματικό χρόνο, καθώς αλλάζουν τα δεδομένα, χωρίς την ανάγκη για περίπλοκες διαδικασίες ανανέωσης της διεπαφής χρήστη.

Επιδόσεις και απόδοση

Ένα από τα κύρια πλεονεκτήματα του Jetpack Compose είναι η **αποδοτικότητα** και η **ταχύτητά του**. Ενώ οι παραδοσιακές προσεγγίσεις UI απαιτούσαν συχνά μεγάλο όγκο κώδικα και περιττές ενέργειες για την αλληλεπίδραση των χρηστών με την εφαρμογή, το Compose έχει σχεδιαστεί για να ελαχιστοποιεί αυτές τις υπερβολές και να προσφέρει πιο αποδοτική διαχείριση των πόρων. Αυτό σημαίνει ότι οι εφαρμογές που αναπτύσσονται με το Compose μπορούν να τρέχουν πιο γρήγορα και να ανταποκρίνονται καλύτερα στις ενέργειες του χρήστη, ειδικά σε σύγχρονες συσκευές Android με πιο προηγμένο υλικό.
Η αποδοτικότητα του Jetpack Compose δεν περιορίζεται μόνο στις συσκευές υψηλής τεχνολογίας. Οι εφαρμογές που αναπτύσσονται με Compose μπορούν να λειτουργούν άψογα και σε πιο παλιές συσκευές, καθώς η πλατφόρμα είναι σχεδιασμένη να προσαρμόζεται δυναμικά στις δυνατότητες του υλικού. Αυτό το κάνει ιδανικό για την ανάπτυξη εφαρμογών που πρέπει να λειτουργούν σε μια ποικιλία συσκευών, από τις τελευταίες κυκλοφορίες μέχρι τα μοντέλα προηγούμενων γενιών.

Συμβατότητα με άλλα Jetpack Libraries

Ένα από τα μεγαλύτερα πλεονεκτήματα του Jetpack Compose είναι η **συμβατότητά του με άλλες βιβλιοθήκες Jetpack**. Το οικοσύστημα Jetpack αποτελείται από μια σειρά εργαλείων και βιβλιοθηκών που έχουν σχεδιαστεί για να διευκολύνουν την ανάπτυξη εφαρμογών Android. Με τη χρήση του Compose, οι προγραμματιστές μπορούν εύκολα να ενσωματώσουν λειτουργίες όπως το **Navigation**, το **LiveData** και το **ViewModel**, που αποτελούν βασικά στοιχεία για τη σύγχρονη ανάπτυξη εφαρμογών.
Η ενσωμάτωση αυτών των εργαλείων γίνεται απρόσκοπτα στο Jetpack Compose, επιτρέποντας στους προγραμματιστές να χτίζουν εφαρμογές που είναι όχι μόνο γρήγορες και αποδοτικές, αλλά και εύκολες στη συντήρηση και την εξέλιξη. Η ευκολία

με την οποία το Compose συνεργάζεται με άλλα εργαλεία του οικοσυστήματος Jetpack, το καθιστά μια ολοκληρωμένη λύση για την ανάπτυξη εφαρμογών που απαιτούν σύνθετη λειτουργικότητα, διατηρώντας παράλληλα την απλότητα και τη σαφήνεια του κώδικα.

Εργαλεία και οικοσύστημα ανάπτυξης

Το Jetpack Compose συνοδεύεται από ένα πλήρες σετ εργαλείων ανάπτυξης που καθιστούν τη διαδικασία ανάπτυξης πιο αποδοτική και αποτελεσματική. Το Android Studio, το επίσημο IDE για την ανάπτυξη Android εφαρμογών, παρέχει **ισχυρή υποστήριξη για το Compose**, με ενσωματωμένες δυνατότητες όπως **Compose Preview**, που επιτρέπει στους προγραμματιστές να βλέπουν αλλαγές στο UI τους σε πραγματικό χρόνο, χωρίς να χρειάζεται να κάνουν compile την εφαρμογή τους.
Επιπλέον, το Compose προσφέρει δυνατότητες για **testing** που διευκολύνουν την ανάπτυξη σταθερών και αξιόπιστων εφαρμογών. Οι προγραμματιστές μπορούν να γράψουν unit tests για τα UI components, καθώς και να ελέγξουν τις αλληλεπιδράσεις μεταξύ του χρήστη και της εφαρμογής, εξασφαλίζοντας ότι το UI λειτουργεί όπως αναμένεται σε διαφορετικά σενάρια χρήσης.

Υιοθέτηση και μελλοντική πορεία

Από τη στιγμή που κυκλοφόρησε, το Jetpack Compose υιοθετήθηκε από πολλές κορυφαίες εφαρμογές, τόσο μικρές όσο και μεγάλες, στον κόσμο του Android. Η κοινότητα των προγραμματιστών έχει αποδεχθεί το Compose ως τη **νέα προτιμώμενη λύση** για τη δημιουργία μοντέρνων, αποδοτικών και εύκολα διαχειρίσιμων UI. Το γεγονός ότι προσφέρει μια ενιαία, σταθερή προσέγγιση για την ανάπτυξη UI, την καθιστά ιδανική επιλογή τόσο για νέους όσο και για έμπειρους προγραμματιστές.
Η Google συνεχίζει να επενδύει στην εξέλιξη του Jetpack Compose, με συχνές ενημερώσεις και βελτιώσεις που αφορούν τις δυνατότητες του framework. Καθώς οι τεχνολογικές απαιτήσεις των εφαρμογών αυξάνονται, το Compose έχει αποδείξει ότι είναι έτοιμο να προσαρμοστεί και να υποστηρίξει την ανάπτυξη του μέλλοντος. Με την ταχύτητα, την αποδοτικότητα και την απλότητά του, το Jetpack Compose αντιπροσωπεύει τη μελλοντική κατεύθυνση της ανάπτυξης UI για Android και θα συνεχίσει να είναι το κύριο εργαλείο για τους προγραμματιστές που θέλουν να δημιουργήσουν κορυφαίες εφαρμογές.

Συμπέρασμα

Το Jetpack Compose ήρθε για να αλλάξει , και το έχει καταφέρει, τον τρόπο με τον οποίο αναπτύσσονται τα UI σε εφαρμογές Android. Με τον δηλωτικό του χαρακτήρα, τη δύναμη της Kotlin και την ευκολία ενσωμάτωσης με το ευρύτερο οικοσύστημα Jetpack, αποτελεί για τους προγραμματιστές ένα εργαλείο που είναι ταυτόχρονα ισχυρό και απλό στη χρήση. Είναι προσαρμοσμένο στις απαιτήσεις των σύγχρονων εφαρμογών, επιτρέποντας τη δημιουργία αποδοτικών και φιλικών προς τον χρήστη διεπαφών, ενώ

προσφέρει την ευελιξία και την ταχύτητα που απαιτείται από τον σημερινό κόσμο της τεχνολογίας.

Το Jetpack Compose είναι σχεδιασμένο για να είναι αποδοτικό και να λειτουργεί γρήγορα σε σύγχρονες συσκευές Android. Με τη χρήση του Compose, οι εφαρμογές μπορούν να τρέχουν πιο ομαλά και να ανταποκρίνονται πιο γρήγορα στις αλληλεπιδράσεις του χρήστη.
Το Jetpack Compose αντιπροσωπεύει το μέλλον της ανάπτυξης UI για Android και έχει ήδη υιοθετηθεί από πολλές εφαρμογές, χάρη στα πολλά πλεονεκτήματα που προσφέρει. Είναι η νέα προτιμώμενη λύση για τη δημιουργία σύγχρονων, αποδοτικών και εύκολα διαχειρίσιμων UI.

2.4 Firebase

2.4.1 Firebase Authentication

Το Firebase Authentication είναι μια υπηρεσία που παρέχεται από τη Firebase η οποία προσφέρει στους προγραμματιστές τη δυνατότητα να προσθέσουν στα προγράμματα/εφαρμογές τους την λειτουργία της αυθεντικοποίησης (Authentication), το γνωστό Login/Signup που ξέρουμε όλοι και έχουμε δει σε σχεδόν όλες τις εφαρμογές. [29]
Η υπηρεσία της σύνδεσης/εγγραφής και γενικώς της αυθεντικοποίησης μπορεί να γίνει με διάφορους τρόπους καθώς η Firebase δίνει τη δυνατότητα για:

- Είσοδος με Email και κωδικό
- Είσοδος με Social Media (Google, Facebook, WhatsApp κλπ.)
- Είσοδος σαν επισκέπτης (Ανώνυμος)

Ο προγραμματιστής μπορεί να ενσωματώσει όποιον τρόπο θέλει και όσους θέλει για να επιτρέψει την είσοδο στην εφαρμογή του.
Πλεονέκτημα της Firebase είναι ότι ο καθένας μπορεί να τη χρησιμοποιήσει καθώς αποτελεί δωρεάν εργαλείο, δίνει την εύκολη επιλογή για τις υπηρεσίες αυθεντικοποίησης καθώς αναλαμβάνει την ασφαλή αποθήκευση των κωδικών των χρηστών και ο προγραμματιστής μπορεί να την ενσωματώσει μέσα σε μόλις λίγες γραμμές κώδικα.
Η Google έχει αγοράσει τη Firebase το 2014 και πλέον αποτελεί το πιο διαδεδομένο τρόπο για την υπηρεσία της αυθεντικοποίησης στις εφαρμογές Android και είναι πλήρως ενσωματωμένο και μέσα στο Android Studio, διευκολύνοντας ακόμα περισσότερο τη χρήση και την ενσωμάτωση της υπηρεσίας.
Με το Firebase Authentication, οι προγραμματιστές μπορούν να επικεντρωθούν στη δημιουργία των λειτουργιών της εφαρμογής τους, χωρίς να χρειάζεται να ασχολούνται με τα πολύπλοκα θέματα της ασφάλειας και της αυθεντικοποίησης χρηστών. Είναι ένα ισχυρό εργαλείο που μπορεί να κάνει τη διαδικασία σύνδεσης πιο ασφαλή και εύκολη για τους χρήστες.

2.4.2 Firebase Firestore

Το Firestore είναι μια υπηρεσία βάσης δεδομένων που προσφέρεται από το Firebase, σχεδιασμένη για την αποθήκευση, συγχρονισμό και διαχείριση δεδομένων σε πραγματικό χρόνο σε εφαρμογές Web και mobile. Είναι μια NoSQL βάση δεδομένων που επιτρέπει την αποθήκευση δεδομένων σε έγγραφα (documents), τα οποία οργανώνονται σε συλλογές (collections). [27]

Τα έγγραφα περιέχουν ζευγάρια κλειδιών και τιμών (key-value pairs), ενώ οι συλλογές είναι ομάδες εγγράφων. Η δομή αυτή επιτρέπει την αποθήκευση δεδομένων με τρόπο εύκολο στην κατανόηση και τη διαχείριση.

Έχεις επιπλέον τη δυνατότητα να εκτελείς περίπλοκα ερωτήματα (Queries) για την ανάκτηση των δεδομένων από τη βάση καθώς και τη δυνατότητα φιλτραρίσματος (Filtering), ταξινόμησης και συνδυασμού των αποτελεσμάτων του λαμβάνεις από τα ερωτήματα και διευκολύνει τη ανάκτηση της πληροφορίας.

Το Firestore επιτρέπει στους χρήστες την αδιάκοπή χρήση της εφαρμογής ακόμα και εκτός σύνδεσης δηλαδή ακόμα και με έλλειψη σύνδεσης με το διαδίκτυο. Τα δεδομένα αποθηκεύονται προσωρινά στη συσκευή του χρήστη και μεταφέρονται στη βάση δεδομένων μόλις αποκατασταθεί η σύνδεση. Έτσι εξασφαλίζει μια ομαλή εμπειρία χρήστη, ανεξάρτητα από τη διαθεσιμότητα του δικτύου.

Με το Firestore, οι προγραμματιστές μπορούν να δημιουργούν εφαρμογές που διαχειρίζονται δεδομένα με αποδοτικό και ασφαλή τρόπο, προσφέροντας ταυτόχρονα μια εξαιρετική εμπειρία χρήστη, χάρη στη δυνατότητα συγχρονισμού σε πραγματικό χρόνο και την υποστήριξη offline. Είναι ένα ευέλικτο και ισχυρό εργαλείο για την ανάπτυξη σύγχρονων εφαρμογών.

Εικόνα 6. Πανεπιστήμιο Πειραιώς

Κεφάλαιο 3ο

3 Μεθοδολογία Σχεδίασης

1. Καταγραφή και Ανάλυση Λειτουργικών Προδιαγραφών

Σκοπός: Να επιτρέψει στους φοιτητές να έχουν την ακαδημαϊκή τους ταυτότητα στην κινητή τους συσκευή χωρίς την ανάγκη της φυσικής κάρτας.

Χρήστες:

- **Φοιτητής:** Ο φοιτητής αποτελεί το κύριο χρήστη της εφαρμογής και μέσα από αυτήν θα έχει πρόσβαση στην ακαδημαϊκή του ταυτότητα.

Προϋποθέσεις:

- Ο χρήστης να έχει κάνει εγγραφή στην εφαρμογή μέσω του αντίστοιχου πεδίου.
- Ο χρήστης να έχει έγκυρα στοιχεία εισόδου σε περίπτωση που έχει πραγματοποιηθεί η εγγραφή.
- Ο χρήστης να έχει έγκυρα στοιχεία ταυτότητας (Αριθμό μητρώου, επώνυμο) για να μπορεί αναζητήσει τη ταυτότητα του.
- Η συσκευή πρέπει να είναι συνδεδεμένη στο διαδίκτυο για να μπορεί να γίνει η επικοινωνία με τη βάση δεδομένων της Firebase.

Βασική Ροή:

1. Ο φοιτητής ανοίγει την εφαρμογή στην κινητή του συσκευή
2. Ο φοιτητής βλέπει την **οθόνη έναρξης (Splash Screen)**, η οποία πραγματοποιεί έλεγχο για το αν ο χρήστης έχει ήδη συνδεθεί
    - Αν έχει πραγματοποιήσει σύνδεση, τότε πραγματοποιείται έλεγχος για το εάν ο χρήστης έχει αποθηκεύσει τη ακαδημαϊκή του ταυτότητα.
        - Εάν έχει αποθηκευμένη ταυτότητα τότε μεταφέρεται απευθείας στην **οθόνη προβολής** της ταυτότητας.
        - Εάν όχι, τότε μεταφέρεται στην κεντρική σελίδα την εφαρμογής, την **οθόνη αναζήτησης**.
    - Αν όχι, μεταφέρεται στη **σελίδα σύνδεσης**.

3. Ο φοιτητής εισάγει τα στοιχεία εισόδου (διαπιστευτήρια) του (email και κωδικό πρόσβασης).
4. Μετά την επιτυχημένη είσοδο, ο φοιτητής μεταφέρεται στην **οθόνη αναζήτησης** της ακαδημαϊκής ταυτότητας, όπου θα εισάγει το επώνυμο και τον αριθμό μητρώου του.
5. Ο φοιτητής επιλέγει πριν κάνει αναζήτηση εάν επιθυμεί να αποθηκευτεί η ακαδημαϊκή του ταυτότητα.
6. Η εφαρμογή πραγματοποιεί αναζήτηση με τα στοιχεία του φοιτητή στη βάση δεδομένων Firebase Firestore.
7. Αν βρεθεί ταυτότητα που ταιριάζει με τα στοιχεία που έχουν δοθεί, τότε εμφανίζονται στη **οθόνη προβολής** της ταυτότητας όλες οι πληροφορίες του φοιτητή.
    - Οι πληροφορίες περιλαμβάνουν όνομα , επώνυμο , αριθμό μητρώου , φωτογραφία , πανεπιστήμιο , ημερομηνία εγγραφής , δελτίου εισιτηρίου , αριθμό ταυτότητας και QR Code για εύκολο σκανάρισμα από την υπηρεσίες του πανεπιστημίου.

Εναλλακτική Ροή :

- Αν ο φοιτητής δεν μπορεί να συνδεθεί ή εισάγει λάθος δεδομένα τότε εμφανίζεται μήνυμα λάθους και ζητείται να τα εισάγει εκ νέου.

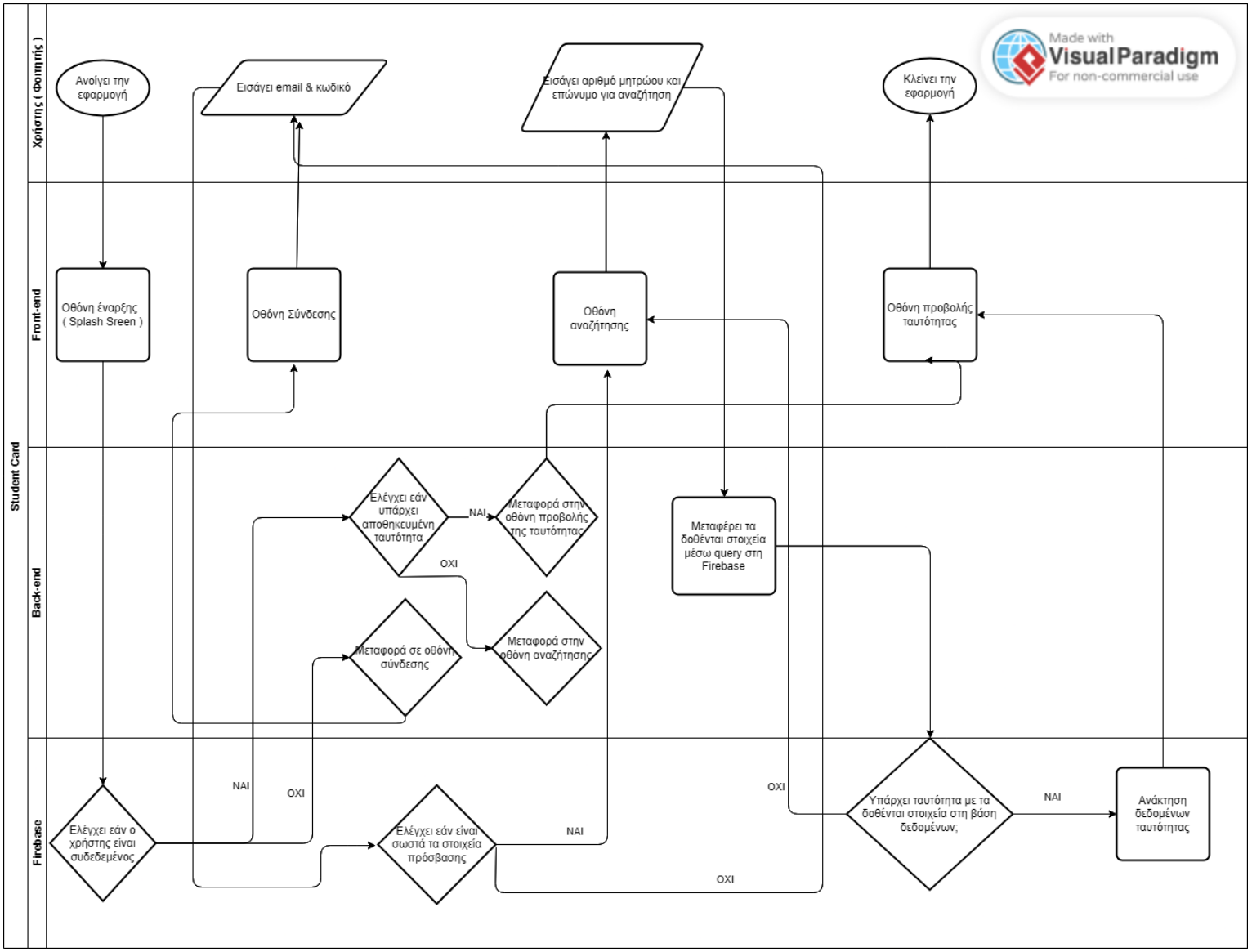
Απαιτήσεις Συστήματος :

- Κινητή συσκευή με λειτουργικό Android
- Firebase για την αυθεντικοποίηση χρηστών και την αποθήκευση δεδομένων

Πιθανά Προβλήματα :

- Αποτυχία σύνδεσης στη βάση δεδομένων λόγω έλλειψης δικτύου.
- Λανθασμένα στοιχεία από το φοιτητή κατά την είσοδο.
- Λανθασμένα στοιχεία από το φοιτητή κατά την αναζήτηση ταυτότητας.

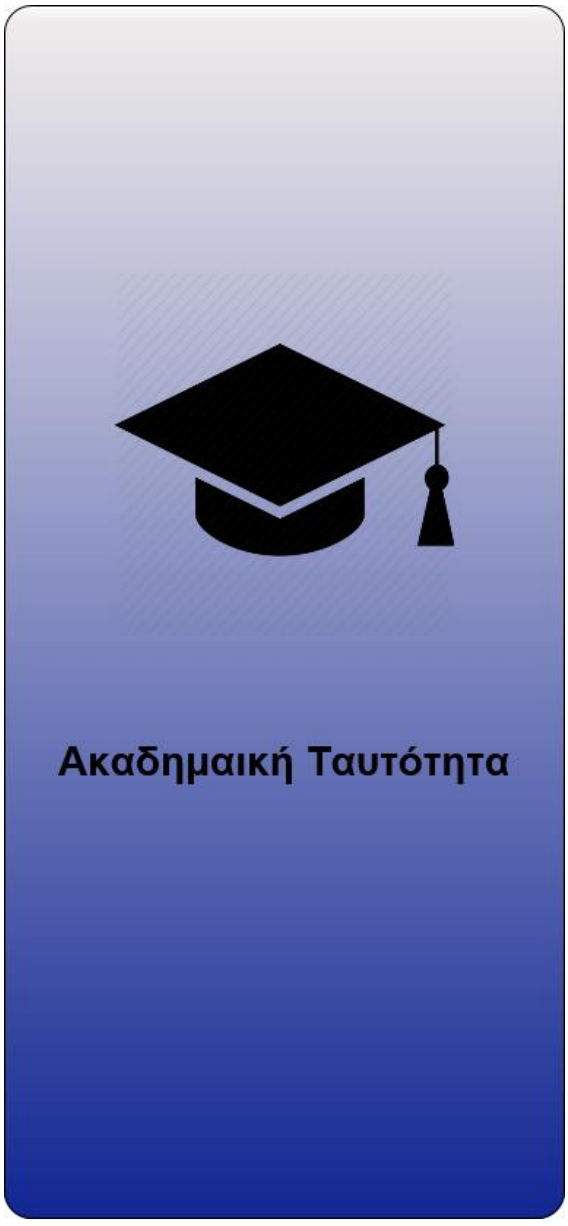
2. Αρχιτεκτονική Λύση με διάγραμμα UML Swimlane



3. Οθόνες της Λύσης μέσω hi-fi story board

Σε αυτή την ενότητα θα παρουσιάσω οθόνης της λύσης όπως της δημιούργησα σε μορφή hi-fi story board. Οι εικόνες αποτελούν τις πρωτότυπες ιδέες σχεδιασμού, οι οποίες αργότερα πέρασαν και στην τελική έκδοση της εφαρμογής.

3.1 Οθόνη Έναρξης



Εικόνα 1 Πρωτότυπη Οθόνη Έναρξης

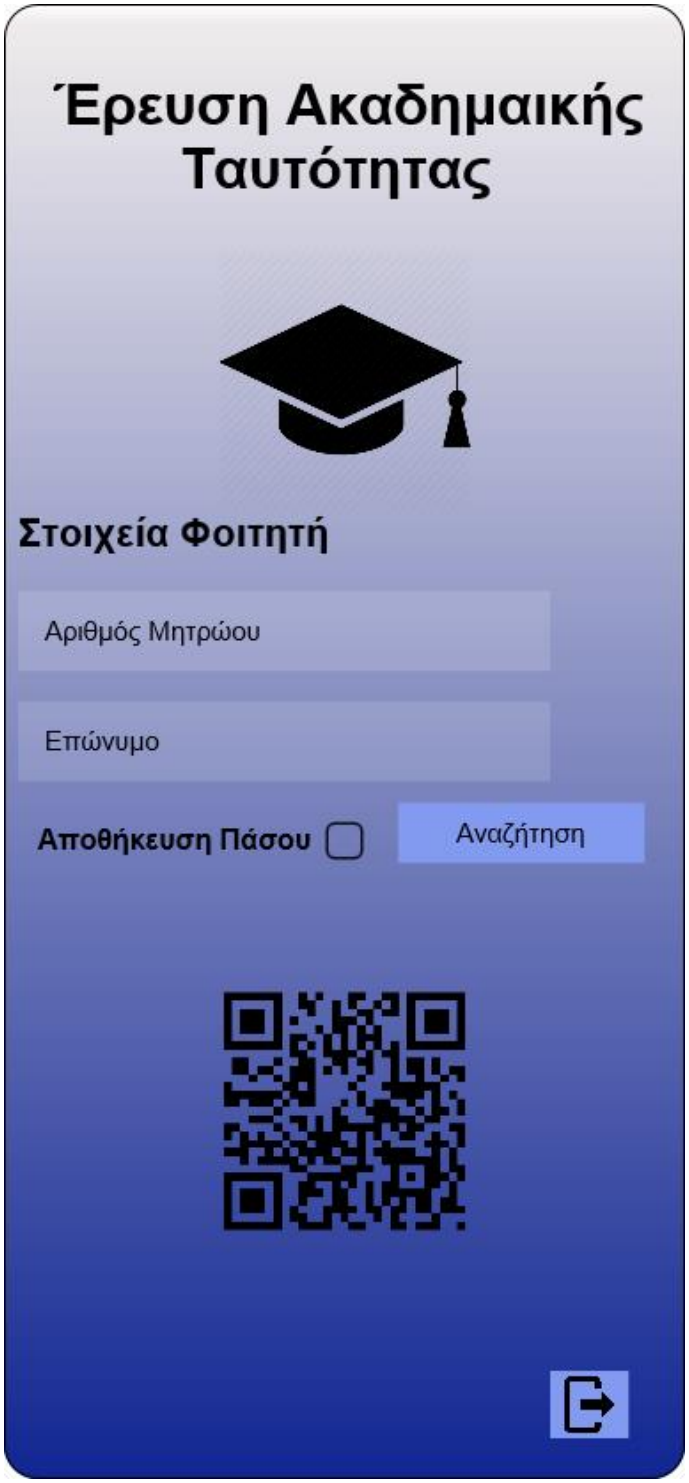
Η οθόνη αυτή αποτελεί την πρώτη επαφή του χρήστη με την εφαρμογή. Συγκεκριμένα, η οθόνη αυτή εμφανίζεται κάθε φόρα που ο χρήστης ανοίγει την εφαρμογή. Ταυτόχρονα στο background γίνονται και ορισμένοι έλεγχοι για την ομαλή λειτουργία της εφαρμογής.

3.2 Οθόνη Σύνδεσης

Εικόνα 2 Πρωτότυπη Οθόνη Σύνδεσης

Η οθόνη αυτή αποτελεί την οθόνη σύνδεσης του χρήστη στην εφαρμογή. Συγκεκριμένα, ο χρήστης πληκτρολογεί το όνομα χρήστη και το κωδικό του και πατάει είτε το κουμπί της εγγραφή για κάνει εγγραφή με τα δοθέντα στοιχεία, είτε πατάει το κουμπί είσοδος για να συνδεθεί με τα δοθέντα στοιχεία.

3.3 Οθόνη Αναζήτησης



Εικόνα 3 Πρωτότυπη Οθόνη Αναζήτησης

Η οθόνη αυτή αποτελεί την οθόνη αναζήτησης της ακαδημαϊκής ταυτότητας του χρήστη. Ο χρήστης θα πρέπει να εισάγει το αριθμό μητρώου και το επώνυμο του και να πατήσει το κουμπί

αναζήτηση. Επιπλέον έχει τη δυνατότητα να πατήσει να αποθηκευτεί το πάσο του και να μην χρειάζεται να εισάγει τα στοιχεία κάθε φορά που ανοίγει την εφαρμογή. Για να πραγματοποιήσει αναζήτηση μπορεί να πατήσει την εικόνα του QR και να κάνει αναζήτηση μέσω του QR της ταυτότητας. Το κουμπί κάτω δεξιά αποτελεί το κουμπί αποσύνδεσης.

3.4 Οθόνη Προβολής Ταυτότητας

Εικόνα 4 Πρωτότυπη Οθόνη Προβολής Ταυτότητας

Η συγκεκριμένα οθόνη αποτελεί την οθόνη προβολής της ταυτότητας. Συγκεκριμένα, μετά από την επιτυχημένη αναζήτηση στην προηγούμενη οθόνη, όλα τα δεδομένα της ακαδημαϊκής ταυτότητας μεταφέρονται εδώ και προβάλλονται στην αντίστοιχη θέση τους.

3.5 Οθόνη Προβολής QR Ταυτότητας

Εικόνα 5 Πρωτότυπη Οθόνη Προβολής QR Ταυτότητας

Η συγκεκριμένα οθόνη αποτελεί συνέχεια της οθόνης προβολής ταυτότητας. Συγκεκριμένα όταν ο χρήστης πατήσει το κουμπί κάτω δεξιά τότε ανοίγει το παράθυρο με το QR κωδικό της ταυτότητας που περιέχει την αριθμό της ταυτότητας.

4 Παρουσίαση Εφαρμογής και Κώδικα

4.1 Παρουσίαση Λειτουργίας Εφαρμογής και Επεξήγηση

Στο συγκεκριμένο κεφάλαιο θα παρουσιάσω την εφαρμογή σε πραγματική λειτουργία παραθέτοντας εικόνες της λειτουργίας της καθώς και επεξήγηση για κάθε οθόνη και τις λειτουργίες της. Όλα τα screenshot προέρχονται από τη λειτουργία της εφαρμογής από τη δικιά μου συσκευή android και συγκεκριμένα το Realme GT Neo 2.

Οθόνη Έναρξης

Η συγκεκριμένη οθόνη αποτελεί την οθόνη έναρξης της εφαρμογής, δηλαδή όταν ο χρήστης ανοίξει την εφαρμογή στο κινητό του τηλέφωνο τότε θα εμφανιστεί αυτή η οθόνη η οποία μετά από μερικά δευτερόλεπτα θα φύγει και ο χρήστης θα μεταφερθεί στην επόμενη οθόνη. Κατά τη διάρκεια αυτής τη οθόνης, εκτελούνται στο παρασκήνιο ορισμένες λειτουργίες/έλεγχοι, συγκεκριμένα μόλις εμφανιστεί η οθόνη ελέγχει εάν ο χρήστης είναι συνδεδεμένος, εάν ναι τότε προχωράει στο επόμενο έλεγχο, εάν όχι τότε ο χρήστης

μεταφέρεται στην **Οθόνη Σύνδεσης** της εφαρμογής. Το επόμενο στάδιο ελέγχου, εφόσον είναι συνδεδεμένος έχει να κάνει με το εάν ο χρήστης σε προηγούμενη αναζήτηση του επέλεξε να αποθηκεύσει την ακαδημαϊκή του ταυτότητα. Εφόσον είναι αποθηκευμένη τότε η εφαρμογή μεταφέρει το χρήστη απευθείας στη **Οθόνη Προβολής Ταυτότητας,** εάν δεν υπάρχει αποθηκευμένη ταυτότητα τότε μεταφέρεται στη **Οθόνη Αναζήτησης.**

Οθόνη Σύνδεσης

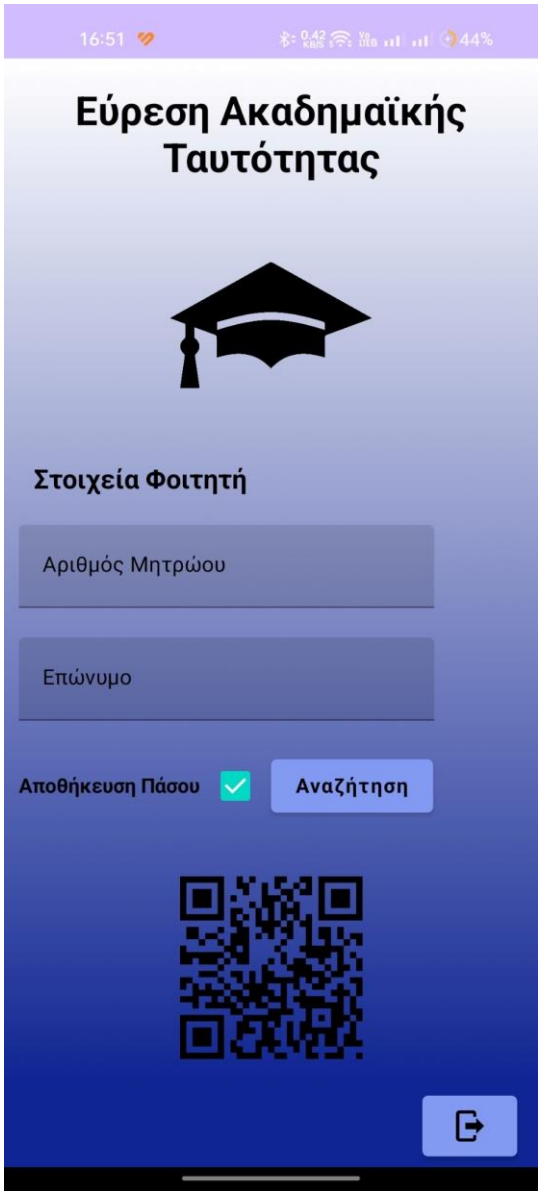
Η συγκεκριμένα οθόνη αποτελεί την οθόνη σύνδεσης του χρήστη. Ο χρήστης θα πρέπει να εισάγει τα στοιχεία του και να πατήσει το κουμπί είσοδος, έπειτα η εφαρμογή θα ελέγξει μέσω της Firebase εάν τα στοιχεία που πληκτρολογήθηκαν είναι ορθά, εφόσον είναι τότε ο μεταφέρεται στην επόμενη οθόνη, την **οθόνη αναζήτησης,** εάν τα στοιχεία δεν είναι σωστά τότε εμφανίζεται ένα μήνυμα σφάλματος ώστε να προβεί σε διορθώσεις. Επιπλέον εάν ο χρήστης δεν διαθέτει λογαριασμό τότε μπορεί να εισάγει τα επιθυμητά στοιχεία εισόδου και να πατήσει Εγγραφή. Για τη πραγματοποίηση της εγγραφής θα πρέπει να ισχύουν κάποιες προϋποθέσεις. Συγκεκριμένα:

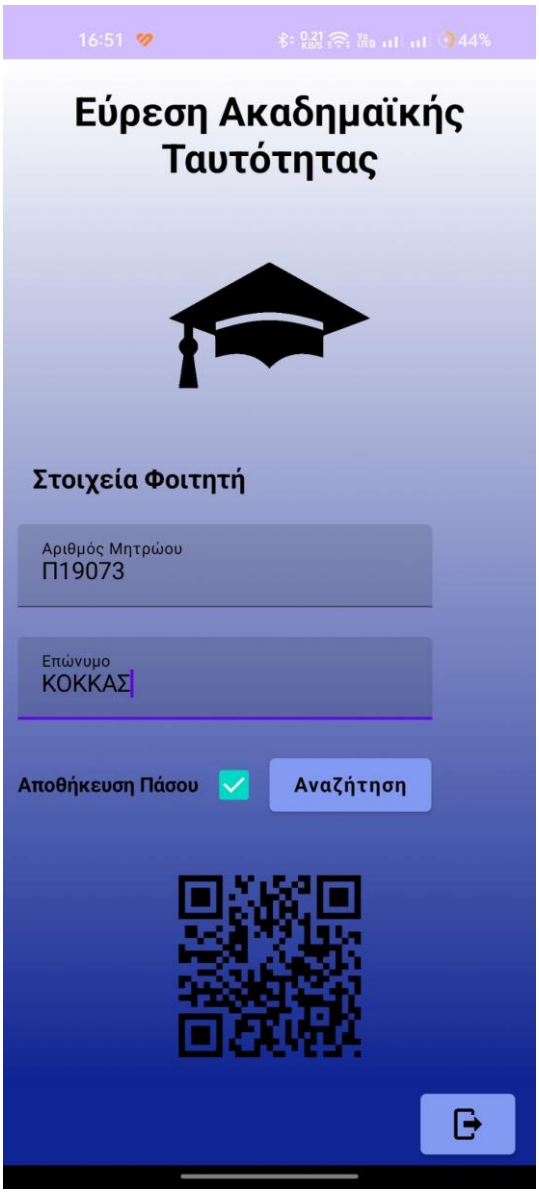
- Τα πεδία όνομα χρήστη και κωδικός να μην είναι κενά
- Το όνομα χρήστη να είναι της μορφής email δηλαδή της μορφής xxxx@xxxx.xxx

- Ο κωδικός πρόσβασης να περιέχει τουλάχιστον 6 ψηφία/γράμματα

Εφόσον όλα τα παραπάνω τηρούνται τότε η εφαρμογή κάνει εγγραφή το χρήστη με τα στοιχεία αυτά στη Firebase και έπειτα ο χρήστης μπορεί να τα χρησιμοποιήσει για να συνδεθεί και να ακολουθηθεί η διαδικασία που περιεγράφηκε πιο πάνω.

Οθόνη Αναζήτησης





Στην συγκεκριμένη οθόνη ο χρήστης κάνει αναζήτηση με τα στοιχεία του, για την ακαδημαϊκή του ταυτότητα. Συγκεκριμένα ο χρήστης θα πληκτρολογήσει τον αριθμό μητρώου του και το επώνυμο του (το επώνυμο θα πρέπει να είναι με κεφαλαία γράμματα), έπειτα έχει την επιλογή εάν θέλει το πάσο του να αποθηκευτεί ή όχι, εφόσον επιλέξει να αποθηκευτεί τότε την επόμενη φορά που θα ανοίξει την εφαρμογή η ακαδημαϊκή του ταυτότητα θα εμφανιστεί αυτόματα. Ο έλεγχος για το αν έχει αποθηκεύσει το πάσο του γίνεται κατά την έναρξη την εφαρμογή στο παρασκήνιο της οθόνης έναρξης.

Αφού πληκτρολογήσει τα στοιχεία του και επιλέξει εάν θέλει να αποθηκευτεί η ταυτότητα ή όχι τότε θα πατήσει το κουμπί «Αναζήτηση», η εφαρμογή τότε θα στείλει το query στη firebase όπου θα γίνει η αναζήτηση μεταξύ των καταχωρημένων εντός της ταυτοτήτων και εάν υπάρξει αντιστοίχιση τότε ο χρήστης μεταφέρεται στην **οθόνη προβολής ταυτότητας,** όπου εκεί θα εμφανιστούν όλα τα στοιχεία της ταυτότητας που αντιστοιχεί στα στοιχεία που έδωσε.

Υπάρχει επιπλέον η δυνατότητα ο χρήστης να πατήσει το σήμα του QR, το οποίο θα ανοίξει τη κάμερα (θα χρειαστεί να δοθεί η κατάλληλη άδεια) για να μπορέσει να σκανάρει το κωδικό QR της ταυτότητας. Αφού σκαναριστεί ο κωδικός τότε ακολουθείτε η ίδια διαδικασία που περιεγράφηκε πριν.

Εάν ο χρήστης το επιθυμεί μπορεί να πραγματοποιήσει αποσύνδεση από την εφαρμογή πατώντας το κουμπί αποσύνδεσης κάτω δεξιά.

Οθόνη Προβολής Ταυτότητας

Η συγκεκριμένη οθόνη αποτελεί την τελική οθόνη της εφαρμογής. Συγκεκριμένα, είναι η οθόνη προβολής της ταυτότητας του χρήστη. Ο χρήστης εδώ θα μπορεί να δει όλες τις πληροφορίες που υπάρχει και στην φυσική ακαδημαϊκή ταυτότητα, δηλαδή φωτογραφία, ονοματεπώνυμο, ημερομηνία εγγραφής, δελτίο εισιτηρίου, αριθμός μητρώου, αριθμός ταυτότητα και διεύθυνση. Επιπλέον πατώντας το κουμπί κάτω δεξιά έχει τη δυνατότητα να προβάλει το QR Code της ταυτότητας που περιέχει τον αριθμό ταυτότητας ακριβώς όπως και το φυσικό αντίτυπο.

Σε περίπτωση που θέλει να κάνει νέα αναζήτηση μπορεί να πατήσει το κουμπί επιστροφής στην πάνω αριστερή γωνία και θα επιστρέψει στην **οθόνη αναζήτησης.**

4.2 Παρουσίαση Κώδικα Εφαρμογής και Επεξήγηση

Στο παρακάτω κεφάλαιο θα παραθέσω το κώδικα που αποτελεί την εφαρμογή καθώς και θα τον αναλύσω με κάθε λεπτομέρεια, παρουσιάζοντας όλες τις μεθόδους, τις μεταβλητές και τα Activities, τη χρήση όλων και πως συνδυάζονται μεταξύ τους, την χρήση της Firebase ως βάση δεδομένων , την ενσωμάτωση του Jetpack Compose καθώς και τη χρήση Flow.

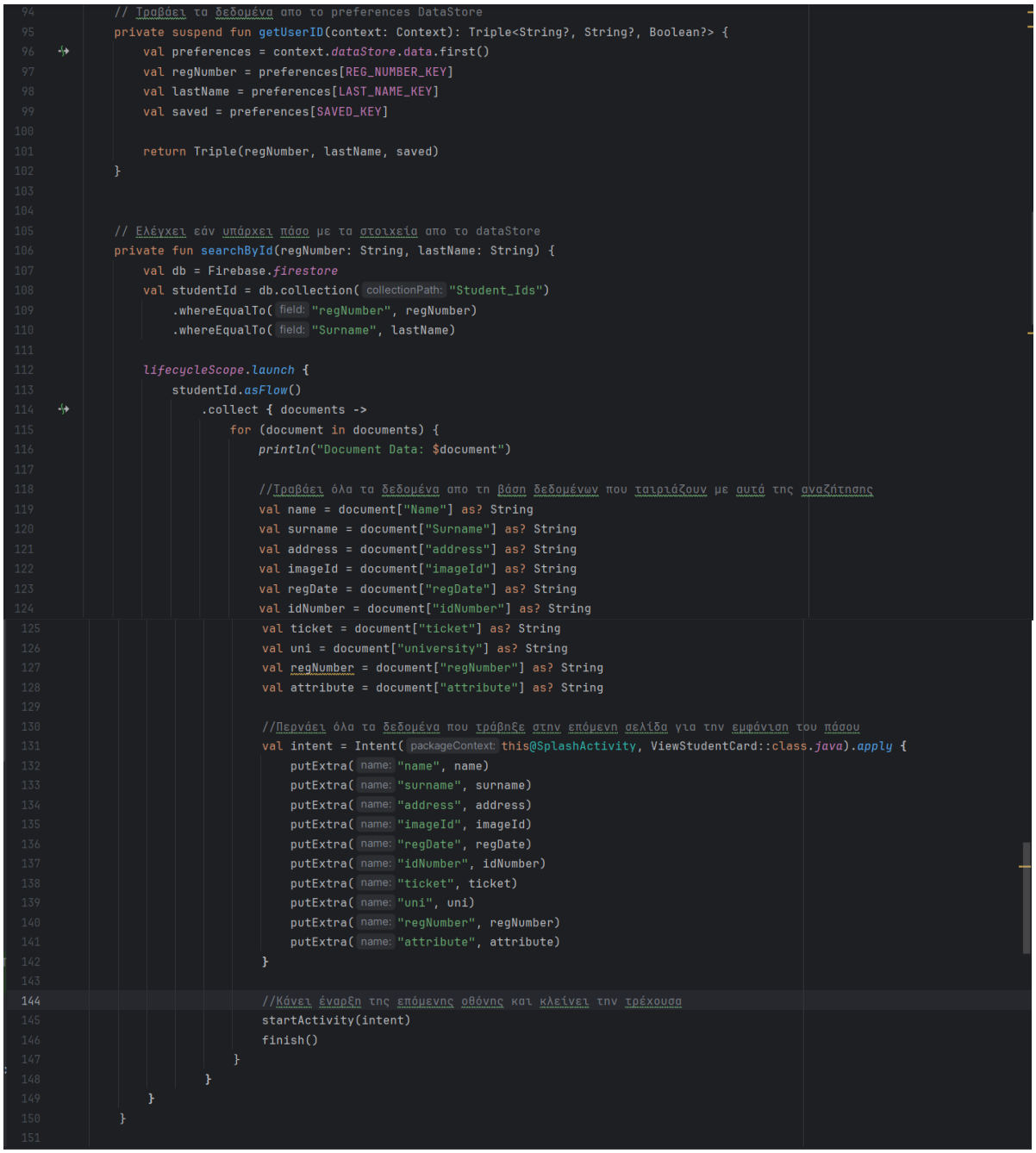
Ο κώδικας θα προβληθεί μέσα από στιγμιότυπα (Screenshots) και θα σχολιαστεί κάτω από κάθε εικόνα τι κάνει το κάθε κομμάτι κώδικα.

Θα προβάλλω πρώτα το κώδικα από το UI μαζί με την προεπισκόπηση του και μετά το κώδικα με τις λειτουργίες της κάθε οθόνης.

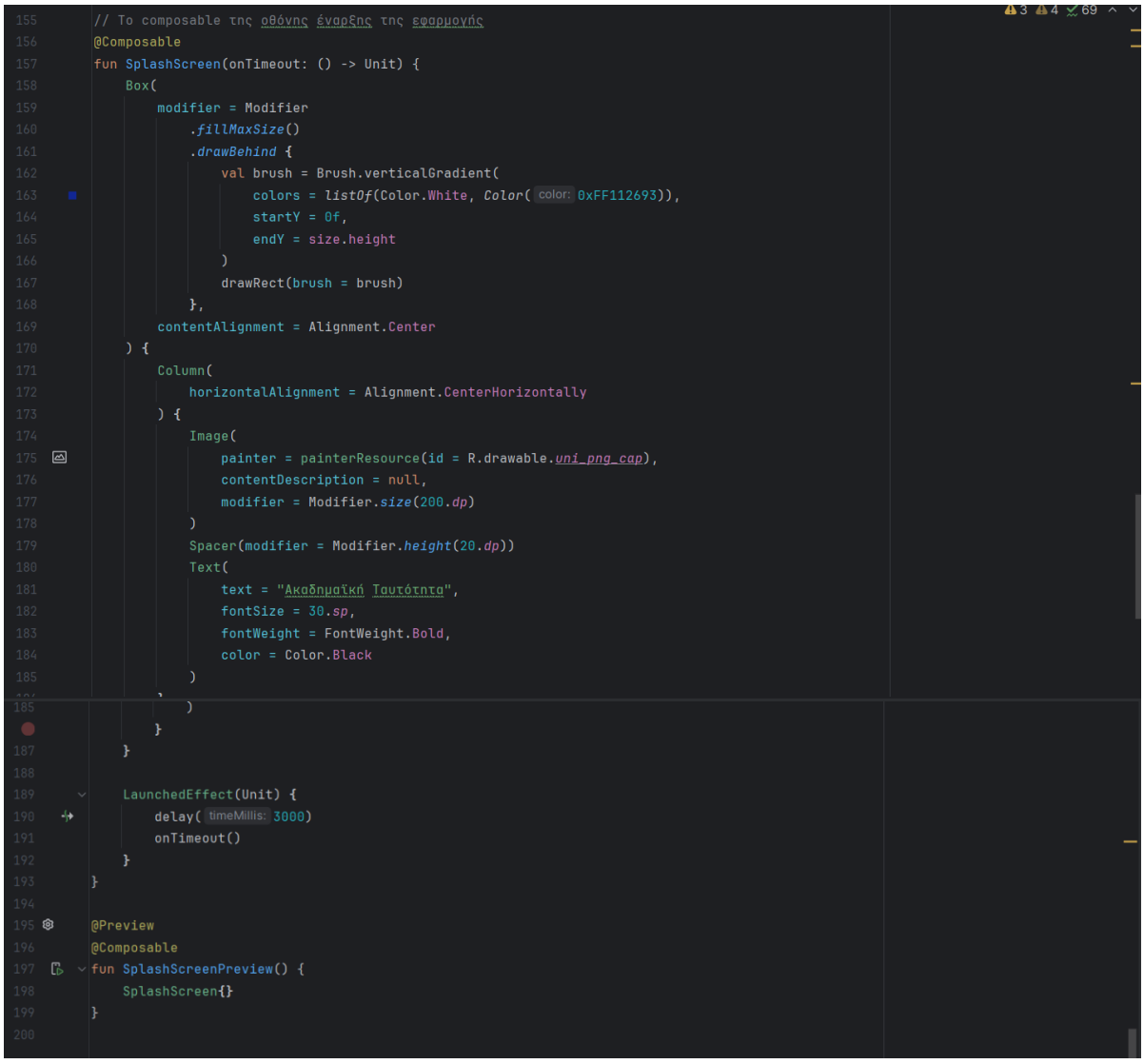
Κάθε κομμάτι αρχείο κώδικα θα έχει και τον αντίστοιχο σχολιασμό που θα εξηγεί τι κάνει η κάθε μέθοδος με λεπτομέρεια.

4.2.1 SplashScreen.kt

```kotlin
package com.spyros.studentcard

import ...

//Η οθόνη έναρξης της εφαρμογής (Splash Screen )

class SplashActivity : ComponentActivity() {

    //Τα κλειδιά αποθήκευσης των δεδομένω στο DataStore
    private val SAVED_KEY = booleanPreferencesKey( name: "saved")
    private val REG_NUMBER_KEY = stringPreferencesKey( name: "regNumber")
    private val LAST_NAME_KEY = stringPreferencesKey( name: "lastName")

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            SplashScreen {
                //Κατα την έναρξη της οθόνης, ελέγχει εάν ο χρήστης είναι συνδεδεμένος ή όχι
                checkAuthenticationStatus()
            }
        }


        // Γίνεται έλεγχος εάν ο χρήστης έχει αποθηκευτεί το πάσο του
        lifecycleScope.launch {
            val (regNumber, lastName, saved) = getUserID( context: this@SplashActivity)
            Log.d( tag: "MainPage",  msg: "Retrieved data - regNumber: $regNumber, lastName: $lastName, saved: $saved")

            if (saved == true) {
                regNumber?.let { reg ->
                    lastName?.let { last ->
                        Log.d( tag: "MainPage",  msg: "Performing search with - regNumber: $reg, lastName: $last")
                        searchById(reg, last)
                    }
                }
            }
        }


    }

    // Έλεγχος εάν είναι συνδεδεμένος ο χρήστης ή όχι
    private fun checkAuthenticationStatus() {
        val user = Firebase.auth.currentUser
        if (user != null) {
            // Ο χρήστης είναι συνδεδεμένος , μεταφορά στην σελίδα αναζήτησης
            val intent = Intent( packageContext: this, MainPage::class.java)
            startActivity(intent)
            finish()
        } else {
            // Ο χρήστης δεν είναι συνδεδεμένος , μεταφορά στην σελίδα εισόδου
            val intent = Intent( packageContext: this, LoginActivity::class.java)
            startActivity(intent)
            finish()
        }
    }
```



```
    // Τραβάει τα δεδομένα απο το preferences DataStore
    private suspend fun getUserID(context: Context): Triple<String?, String?, Boolean?> {
        val preferences = context.dataStore.data.first()
        val regNumber = preferences[REG_NUMBER_KEY]
        val lastName = preferences[LAST_NAME_KEY]
        val saved = preferences[SAVED_KEY]

        return Triple(regNumber, lastName, saved)
    }


    // Ελέγχει εάν υπάρχει πάσο με τα στοιχεία απο το dataStore
    private fun searchById(regNumber: String, lastName: String) {
        val db = Firebase.firestore
        val studentId = db.collection( collectionPath: "Student_Ids")
            .whereEqualTo( field: "regNumber", regNumber)
            .whereEqualTo( field: "Surname", lastName)

        lifecycleScope.launch {
            studentId.asFlow()
                .collect { documents ->
                    for (document in documents) {
                        println("Document Data: $document")

                        //Τραβάει όλα τα δεδομένα απο τη βάση δεδομένων που ταιριάζουν με αυτά της αναζήτησης
                        val name = document["Name"] as? String
                        val surname = document["Surname"] as? String
                        val address = document["address"] as? String
                        val imageId = document["imageId"] as? String
                        val regDate = document["regDate"] as? String
                        val idNumber = document["idNumber"] as? String
                        val ticket = document["ticket"] as? String
                        val uni = document["university"] as? String
                        val regNumber = document["regNumber"] as? String
                        val attribute = document["attribute"] as? String

                        //Περνάει όλα τα δεδομένα που τράβηξε στην επόμενη σελίδα για την εμφάνιση του πάσου
                        val intent = Intent( packageContext: this@SplashActivity, ViewStudentCard::class.java).apply {
                            putExtra( name: "name", name)
                            putExtra( name: "surname", surname)
                            putExtra( name: "address", address)
                            putExtra( name: "imageId", imageId)
                            putExtra( name: "regDate", regDate)
                            putExtra( name: "idNumber", idNumber)
                            putExtra( name: "ticket", ticket)
                            putExtra( name: "uni", uni)
                            putExtra( name: "regNumber", regNumber)
                            putExtra( name: "attribute", attribute)
                        }

                        //Κάνει έναρξη της επόμενης οθόνης και κλείνει την τρέχουσα
                        startActivity(intent)
                        finish()
                    }
                }
        }
    }

```



```
// Το composable της οθόνης έναρξης της εφαρμογής
@Composable
fun SplashScreen(onTimeout: () -> Unit) {
    Box(
        modifier = Modifier
            .fillMaxSize()
            .drawBehind {
                val brush = Brush.verticalGradient(
                    colors = listOf(Color.White, Color( color: 0xFF112693)),
                    startY = 0f,
                    endY = size.height
                )
                drawRect(brush = brush)
            },
        contentAlignment = Alignment.Center
    ) {
        Column(
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Image(
                painter = painterResource(id = R.drawable.uni_png_cap),
                contentDescription = null,
                modifier = Modifier.size(200.dp)
            )
            Spacer(modifier = Modifier.height(20.dp))
            Text(
                text = "Ακαδημαϊκή Ταυτότητα",
                fontSize = 30.sp,
                fontWeight = FontWeight.Bold,
                color = Color.Black
            )
        }
    }

    LaunchedEffect(Unit) {
        delay( timeMillis: 3000)
        onTimeout()
    }
}

@Preview
@Composable
fun SplashScreenPreview() {
    SplashScreen{}
}
```

Η συγκεκριμένα κλάση (class) αποτελείτε από μια Composable συνάρτηση που αφορά το UI και από διάφορες συναρτήσεις που κάνουν ορισμένους ελέγχους και τραβάνε δεδομένα από τη βάση δεδομένων.

Ο σκοπός της είναι η εμφάνιση μιας μικρής και σύντομης οθόνης έναρξης για τους χρήστες, κάθε φορά που ανοίγουν την εφαρμογή. Αυτό το κομμάτι το αναλαμβάνει η Composable συνάρτηση SplashScreen().

Εκτός από την εμφάνιση της οθόνης έναρξης η κλάση κάνει και μερικούς απαραίτητους ελέγχους ώστε να μπορεί να αποφασίσει σε ποια οθόνη θα ξεκινήσει.

Συγκεκριμένα, κατά την έναρξη της οθόνης γίνεται έλεγχος για το εάν ο χρήστης έχει πραγματοποιήσει προηγουμένως σύνδεση, αυτό επιτυγχάνεται με τη χρήση της συνάρτησης checkAuthenticationStatus(). Η συγκεκριμένη συνάρτηση «τραβάει» από τη βάση δεδομένων και συγκεκριμένα το Firebase Authentication τα στοιχεία του τρέχοντος χρήστη, εάν ο χρήστης δεν είναι null δηλαδή είναι συνδεδεμένος τότε τον μεταφέρει με τη χρήση του Intent στην κεντρική σελίδα MainPage, ενώ εάν ο χρήστης είναι null τότε το μεταφέρει στη σελίδα σύνδεσης LoginActivity για να πραγματοποιήσει είσοδο.

Μετά τον έλεγχο για την αυθεντικοποίηση του χρήστη, κάνει επίσης έλεγχο για το αν υπάρχουν αποθηκευμένα δεδομένα στο Preferences DataStore. Ο έλεγχος πραγματοποιείτε χρησιμοποιώντας τη συνάρτηση getUserID η οποία τραβάει στις μεταβλητές regNumber , lastName , saved τα κλειδιά REG_NUMBER_KEY , LAST_NAME_KEY , SAVED_KEY αντίστοιχα και κοιτώντας εάν το κλειδί SAVED_KEY περιέχει την τιμή true δηλαδή εάν υπάρχει αποθηκευμένο

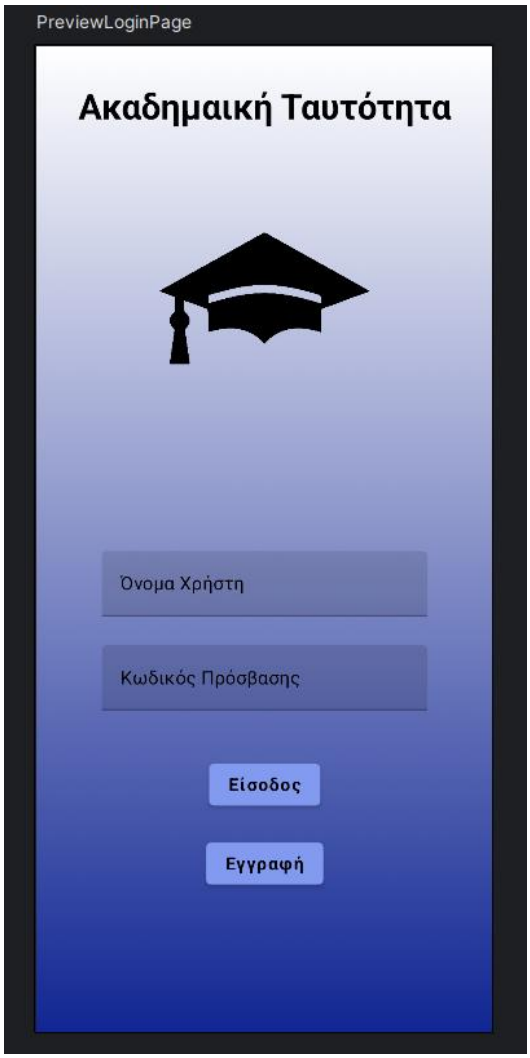
πάσο, εάν όχι, δηλαδή η τιμή είναι false ή null και δεν υπάρχει αποθηκευμένο πάσο, τότε συνεχίζει κανονικά τη ροή και μεταφέρει το χρήστη είτε στην οθόνη σύνδεσης εάν δεν είναι συνδεδεμένος είτε στην οθόνη αναζήτησης πάσου εάν είναι συνδεδεμένος, εάν η τιμή όμως είναι true, δηλαδή ο χρήστης έχει αποθηκεύσει κάποιο πάσο και τα κλειδιά περιέχουν τις βασικές τιμές που απαιτούνται για την αναζήτηση του πάσου, τότε καλεί τη συνάρτηση searchById στην οποία δίνει τις μεταβλητές regNumber και lastName που περιέχουν τον αριθμό μητρώου και το επώνυμου του φοιτητή αντίστοιχα.

Η συνάρτηση αυτή μετά παίρνει τις μεταβλητές αυτές και ψάχνει στο Firestore στο Collection με όνομα “Student_Ids” στο οποίο είναι αποθηκευμένα τα φοιτητικά πάσα και ελέγχει αν τα πεδία regNumber και Surname ταιριάζουν με τις μεταβλητές regNumber και lastName αντίστοιχα. Εάν ναι τότε τραβάει από το Collection όλα τα δεδομένα από την ακαδημαική ταυτότητα και τα βάζει στο intent και κάνει έναρξη της επόμενης οθόνης που είναι η προβολή του πάσου.

4.2.2 LoginPage.kt

```kotlin
package com.spyros.studentcard.composeAssets

import ...

@Composable
/*
Composable για την οθόνη του Login Page. Συγκεκριμένα περιέχει δύο πεδία στα οποία
θα εισάγει ο χρήστης τα στοιχεία εισόδου του και δύο κουμπιά τα οποία πραγματοποιούν
την είσοδο ή την εγγραφή του χρήστη.
 */
fun LoginPage(
    onLogin: (String, String) -> Unit,
    onRegister: (String, String) -> Unit
) {
    Box(modifier = Modifier
        .fillMaxSize()
        .drawBehind {
            val brush = Brush.verticalGradient(
                colors = listOf(Color.White, Color( color: 0xFF112693)),
                startY = 0f,
                endY = size.height
            )
            drawRect(brush = brush)
        }
    ) {
        Column(
            modifier = Modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Text(
                text = "Ακαδημαϊκή Ταυτότητα",
                modifier = Modifier
                    .padding(15.dp)
                    .padding(top = 20.dp),
                fontSize = 30.sp,
                color = Color.Black,
                fontWeight = FontWeight.Bold,
            )

            Spacer(modifier = Modifier.height(30.dp))

            Image(
                painter = painterResource(id = R.drawable.uni_png_cap),
                contentDescription = null,
                modifier = Modifier.size(200.dp)
            )

            Spacer(modifier = Modifier.height(120.dp))

            var username by rememberSaveable { mutableStateOf( value: "") }
            var password by rememberSaveable { mutableStateOf( value: "") }

            TextField(
                value = username,
                onValueChange = { username = it },
                label = { Text( text: "Όνομα Χρήστη") },
                textStyle = TextStyle(fontSize = 18.sp),

                colors = TextFieldDefaults.textFieldColors(
                    focusedLabelColor = Color.Black,
```

```
                    unfocusedLabelColor = Color.Black
                ),
                keyboardOptions = KeyboardOptions.Default.copy(
                    imeAction = ImeAction.Next
                )
            )
            Spacer(modifier = Modifier.height(25.dp))

            TextField(
                value = password,
                onValueChange = { password = it },
                label = { Text( text: "Κωδικός Πρόσβασης") },
                textStyle = TextStyle(fontSize = 18.sp),
                colors = TextFieldDefaults.textFieldColors(
                    focusedLabelColor = Color.Black,
                    unfocusedLabelColor = Color.Black
                ),
                visualTransformation = PasswordVisualTransformation(), // Hide password input
                keyboardOptions = KeyboardOptions(
                    keyboardType = KeyboardType.Password,
                    imeAction = ImeAction.Done
                )
            )

            Spacer(modifier = Modifier.height(40.dp))

            Button(
                onClick = { onLogin(username, password) },
                colors = ButtonDefaults.buttonColors(backgroundColor = Color( color: 0xff829aef))
            ) {
                Text(text = "Είσοδος",
                        fontWeight = FontWeight.Bold,
                    fontSize = 15.sp,)
            }
            Spacer(modifier = Modifier.height(20.dp))

            Button(
                onClick = { onRegister(username, password) },
                colors = ButtonDefaults.buttonColors(backgroundColor = Color( color: 0xff829aef))
            ) {
                Text(text = "Εγγραφή",
                    fontWeight = FontWeight.Bold,
                    fontSize = 15.sp)
            }
        }
    }
}

@Preview
@Composable
fun PreviewLoginPage() {
    LoginPage(
        onLogin = { _, _ -> },
        onRegister = { _, _ -> }
    )
}
```



Το LoginPage.kt αποτελεί το Composable κομμάτι της σελίδας εισόδου/εγγραφής, δηλαδή αποτελεί το αρχείο με το σχεδιασμό του UI.

Συγκεκριμένα έχουμε μέσα σε ένα Box, ένα Column που περιέχει το τίτλο, την εικόνα, τα δύο πεδία εισόδου για το όνομα χρήστη και για τον κωδικό , καθώς και δύο κουμπιά, ένα για την είσοδο και ένα για την εγγραφή.

Τα κουμπιά αυτά παίρνουν τα στοιχεία που έχει πληκτρολογήσει ο χρήστης στα πεδία και τα βάζουν στις μεταβλητές username και password.

Εάν πατηθεί το κουμπί εισόδου τότε εκτελείται η συνάρτηση onLogin η οποία παίρνει το username και το password σε μορφή String και το στέλνει στο LoginActivity μαζί με τις μεταβλητές όπου εκεί γίνεται η διαδικασία της σύνδεσης.

Εάν πατηθεί το κουμπί εγγραφής τότε εκτελείται η συνάρτηση onRegister η οποία ακολουθεί την ίδια διαδικασία με την onLogin με την εξαίρεση ότι τώρα αναλαμβάνει την διαδικασία της εγγραφής του χρήστη.

4.2.3 LoginActivity

```kotlin
package com.spyros.studentcard

import ...

class LoginActivity : ComponentActivity() {

    private val auth = Firebase.auth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            /* Χρήσιμοποιεί τις μεθόδους onLogin και onRegister μαζι με τα δεδομένα που σταλθηκαν απο το
                composable για να κάνει σύνδεση ή εγγραφή ο χρήστης
             */
            LoginPage(
                onLogin = { username, password -> handleLogin(username, password) },
                onRegister = { username, password -> handleRegister(username, password) }
            )
        }
    }

    // Η συνάρτηση αυτή διαχειρίζεται τη σύνδεση του χρήστη.
    private fun handleLogin(username: String, password: String) {
        // Εάν τα πεδία είναι συμπληρωμένα σωστά τότε προσπαθεί να πραγματοποιήσει την σύνδεση
        if (fieldsCheck(username, password)) {
            auth.signInWithEmailAndPassword(username, password)
                .addOnCompleteListener(this) { task ->
                    if (task.isSuccessful) {
                        Log.d( tag: "LoginActivity",  msg: "signInWithEmail:success")
                        val user = auth.currentUser
                        if (user != null) {
                            Log.d( tag: "LoginActivity",  msg: "current user is $user")
                            val intent = Intent( packageContext: this, MainPage::class.java)
                            startActivity(intent)
                            finish()
                        }
                    } else {
                        Log.w( tag: "LoginActivity",  msg: "signInWithEmail:failure", task.exception)
                        Toast.makeText(baseContext,  text: "Αποτυχία Σύνδεσης. Προσπαθήστε Ξανά",
                            Toast.LENGTH_SHORT).show()
                    }
                }
        }
    }


    // Η συνάρτηση αυτή διαχειρίζεται την εγγραφή του χρήστη.
    private fun handleRegister(username: String, password: String) {
        // Εάν τα πεδία είναι συμπληρωμένα σωστά τότε προσπαθεί να πραγματοποιήσει την εγγραφή
        if (fieldsCheck(username, password)) {
            auth.createUserWithEmailAndPassword(username, password)
                .addOnCompleteListener(this) { task ->
                    if (task.isSuccessful) {
                        Log.d( tag: "LoginActivity",  msg: "createUserWithEmail:success")
                        Toast.makeText(baseContext,  text: "Επιτυχής εγγραφή. Τώρα μπορείτε να συνδεθείτε!", Toast.LENGTH_SHORT).show()
                    } else {
                        Log.w( tag: "LoginActivity",  msg: "createUserWithEmail:failure", task.exception)
                        Toast.makeText(baseContext,  text: "Αποτυχία Εγγραφής. Προσπαθήστε Ξανά",
                            Toast.LENGTH_SHORT).show()
                    }
                }
```

```
        }
    }

    //Η συνάρτηση αυτή ελέγχει ότι τα πεδία είναι συμπληρωμένα σωστά
    private fun fieldsCheck(username: String, password: String): Boolean {
        var valid = true
        if (username.isEmpty()) {
            Toast.makeText( context: this,  text: "Το username δεν μπορεί να είναι κενό", Toast.LENGTH_SHORT).show()
            valid = false
        }
        if (password.isEmpty()) {
            Toast.makeText( context: this,  text: "Ο κωδικός δεν μπορεί να είναι κενός", Toast.LENGTH_SHORT).show()
            valid = false
        }
        if (password.length < 6) {
            Toast.makeText( context: this,  text: "Ο κωδικός πρέπει να αποτελείτε απο τουλάχιστον 6 ψηφία", Toast.LENGTH_SHORT).show()
            valid = false
        }
        if (!Patterns.EMAIL_ADDRESS.matcher(username).matches()) {
            Toast.makeText( context: this,  text: "Παρακαλώ είσάγετε μια έγκυρη διεύθυνση email", Toast.LENGTH_SHORT).show()
            valid = false
        }
        return valid
    }
}
```

Το activity αυτό αποτελεί το «σκελετό» της οθόνης σύνδεσης, δηλαδή περιέχει τις απαραίτητες συναρτήσεις για την εκτέλεση της σύνδεσης ή της εγγραφής.

Συγκεκριμένα στην onCreate ορίζει το UI, το οποίο είναι το LoginPage() μέσα από το οποίο λαμβάνει από τις συναρτήσεις onLogin και onRegister το username και το password και ανάλογα με το ποια συνάρτηση έδωσε δεδομένα, δηλαδή ποιο κουμπί έχει πατηθεί στο UI, τότε πάει είτε στη συνάρτηση handleLogin που χειρίζεται τη σύνδεση του χρήστη είτε στη συνάρτηση handleRegister που χειρίζεται την εγγραφή του χρήστη .

Και οι δύο συναρτήσεις πριν ξεκινήσουν την διαδικασία την σύνδεσης ή της εγγραφής αντίστοιχα, παίρνουν το username και το password που έχουν δοθεί και τα περνάνε από τη συνάρτηση fieldsCheck η οποία κάνει τους απαραίτητους ελέγχους και επιστρέφει τη τιμή valid αν όλα τα στοιχεία ήταν οκ. Συγκεκριμένα ελέγχει εάν κάποια τιμή είναι άδεια/null , εάν ο κωδικός έχει λιγότερο από 6 ψηφία και εάν το email έχει τη σωστή μορφή. Εφόσον όλα είναι εντάξει επιστρέφεται η τιμή Boolean Valid που θα είναι true, σε αντίθετη περίπτωση που κάτι δεν είναι σωστό τότε επιστρέφει false και εμφανίζεται το αντίστοιχο μήνυμα στο χρήστη για να προβεί στις απαραίτητες διορθώσεις.

Εφόσον όλα περάσουν από τους απαραίτητους ελέγχους και τα δεδομένα είναι σωστά δηλαδή η συνάρτηση fieldsCheck επιστρέψει τη μεταβλητή Valid με τιμή True, τότε προχωράνε κανονικά στην εκτέλεση τους.

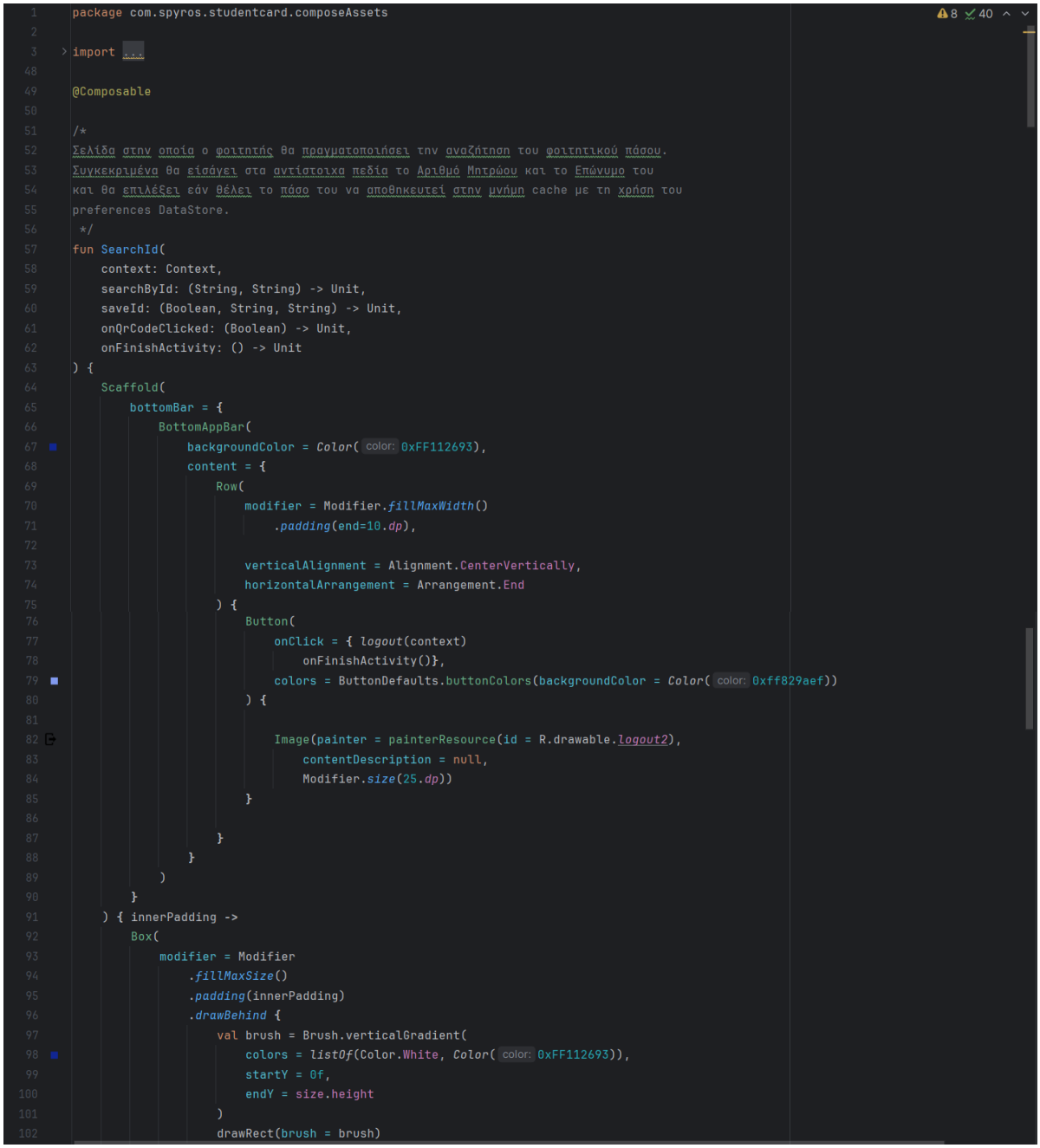
Η handleLogin κάνοντας χρήση της μεθόδου σύνδεσης της Firebase Authentication ελέγχει ένα τα στοιχεία που δόθηκαν είναι έγκυρα δηλαδή εάν ανήκουν σε κάποιον λογαριασμό, εάν ναι τότε προχωράει στη σύνδεση και μεταφέρει το χρήστη στην επόμενη οθόνη (Activity) η οποία είναι και η κεντρική οθόνη. Σε περίπτωση αποτυχίας εμφανίζεται ένα Toast Message όπου ενημερώνει το χρήστη ότι δεν ήταν εφικτή η σύνδεση.

Η handleRegister κάνοντας χρήση της Firebase Authentication για την εγγραφή κάποιου χρήστη, παίρνει τα στοιχεία και δημιουργεί στη βάση δεδομένων έναν χρήστη με τα δοσμένα στοιχεία

σύνδεσης, τα οποία θα τα χρησιμοποιήσει για να συνδεθεί. Εάν ή εγγραφή είναι επιτυχής τότε εμφανίζει ένα μήνυμα ότι η εγγραφή ήταν επιτυχής και ότι μπορεί να πραγματοποιήσει σύνδεση. Σε περίπτωση αποτυχίας εμφανίζεται μήνυμα που ενημερώνει το χρήστη ότι η εγγραφή του απέτυχε και να προσπαθήσει ξανά.

Εφόσον έχει ολοκληρωθεί η εγγραφή, ο χρήστης μπορεί να ακολουθήσει τη διαδικασία της εισόδου, η οποία θα τον μεταφέρει στην κεντρική σελίδα όπου θα μπορεί να αναζητήσει το πάσο του.

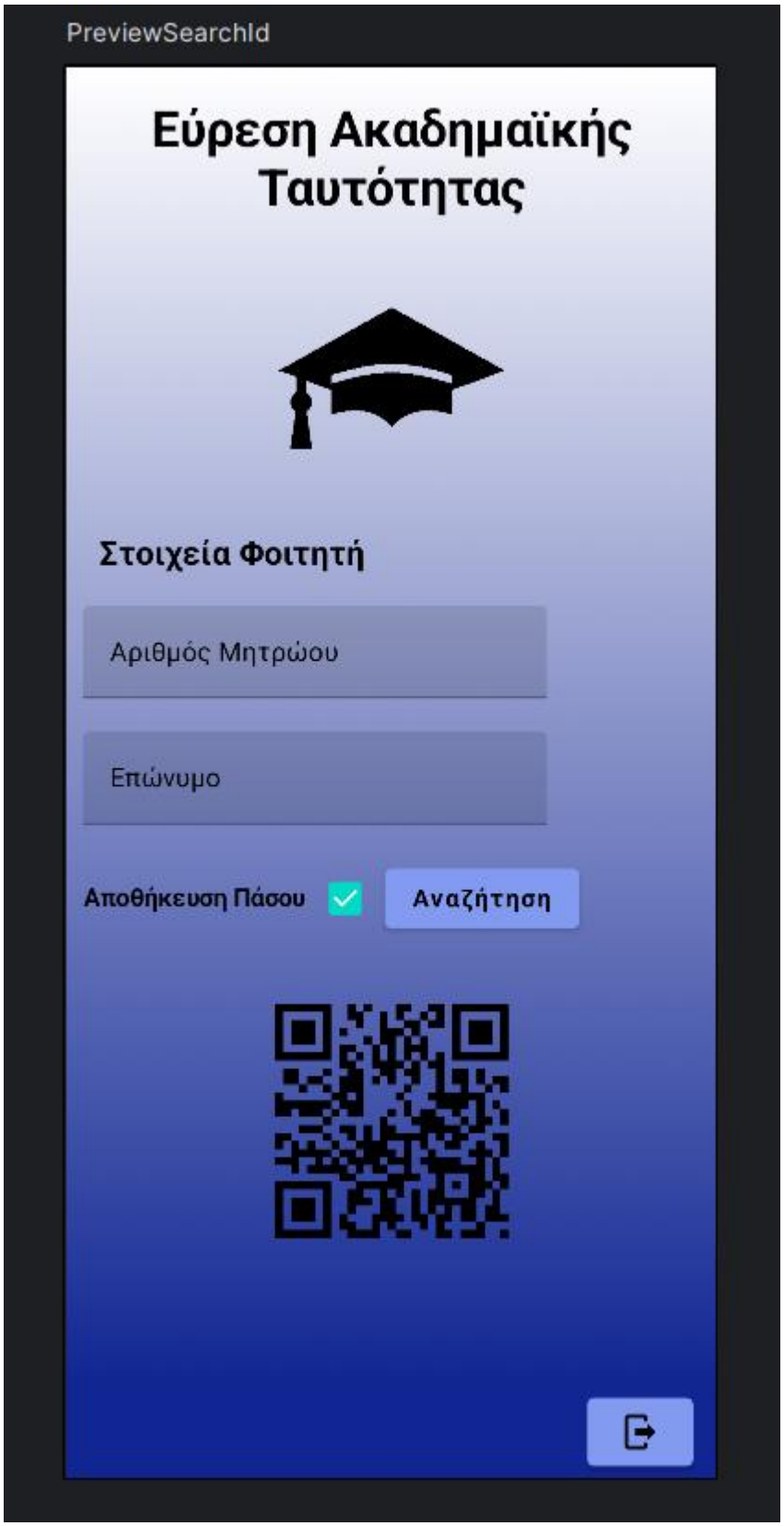
4.2.4 SearchID.kt



```kotlin
package com.spyros.studentcard.composeAssets

import ...

@Composable

/*
Σελίδα στην οποία ο φοιτητής θα πραγματοποιήσει την αναζήτηση του φοιτητικού πάσου.
Συγκεκριμένα θα είσάγει στα αντίστοιχα πεδία το Αριθμό Μητρώου και το Επώνυμο του
και θα επιλέξει εάν θέλει το πάσο του να αποθηκευτεί στην μνήμη cache με τη χρήση του
preferences DataStore.
 */
fun SearchId(
    context: Context,
    searchById: (String, String) -> Unit,
    saveId: (Boolean, String, String) -> Unit,
    onQrCodeClicked: (Boolean) -> Unit,
    onFinishActivity: () -> Unit
) {
    Scaffold(
        bottomBar = {
            BottomAppBar(
                backgroundColor = Color( color: 0xFF112693),
                content = {
                    Row(
                        modifier = Modifier.fillMaxWidth()
                            .padding(end=10.dp),

                        verticalAlignment = Alignment.CenterVertically,
                        horizontalArrangement = Arrangement.End
                    ) {
                        Button(
                            onClick = { logout(context)
                                onFinishActivity()},
                            colors = ButtonDefaults.buttonColors(backgroundColor = Color( color: 0xff829aef))
                        ) {

                            Image(painter = painterResource(id = R.drawable.logout2),
                                contentDescription = null,
                                Modifier.size(25.dp))
                        }

                    }
                }
            )
        }
    ) { innerPadding ->
        Box(
            modifier = Modifier
                .fillMaxSize()
                .padding(innerPadding)
                .drawBehind {
                    val brush = Brush.verticalGradient(
                        colors = listOf(Color.White, Color( color: 0xFF112693)),
                        startY = 0f,
                        endY = size.height
                    )
                    drawRect(brush = brush)
```

```
            }
        ) {
            Column(modifier = Modifier.fillMaxSize()) {
                Text(
                    text = "Εύρεση Ακαδημαϊκής Ταυτότητας",
                    fontSize = 30.sp,
                    textAlign = TextAlign.Center,
                    fontWeight = FontWeight.Bold,
                    modifier = Modifier.padding(top = 20.dp)
                )
                Spacer(modifier = Modifier.padding(10.dp))

                Column(modifier = Modifier.fillMaxSize()) {
                    Image(
                        painter = painterResource(id = R.drawable.uni_png_cap),
                        contentDescription = null,
                        modifier = Modifier
                            .size(150.dp)
                            .align(Alignment.CenterHorizontally)
                    )

                    Spacer(modifier = Modifier.padding(10.dp))
                    Text(
                        text = "Στοιχεία Φοιτητή",
                        fontSize = 20.sp,
                        textAlign = TextAlign.Left,
                        fontWeight = FontWeight.Bold,
                        modifier = Modifier.padding(start = 20.dp, bottom = 20.dp, end = 5.dp)
                    )

                    var regNumber by rememberSaveable { mutableStateOf( value: "") }
                    var lastName by rememberSaveable { mutableStateOf( value: "") }
                    var checked by remember { mutableStateOf( value: true) }

                    TextField(
                        value = regNumber,
                        onValueChange = { regNumber = it },
                        label = { Text( text: "Αριθμός Μητρώου") },
                        textStyle = TextStyle(fontSize = 18.sp),
                        colors = TextFieldDefaults.textFieldColors(
                            focusedLabelColor = Color.Black,
                            unfocusedLabelColor = Color.Black
                        ),
                        keyboardOptions = KeyboardOptions.Default.copy(
                            imeAction = ImeAction.Next
                        ),
                        modifier = Modifier.padding(bottom = 20.dp, start = 10.dp)
                    )

                    TextField(
                        value = lastName,
                        onValueChange = { lastName = it },
                        label = { Text( text: "Επώνυμο") },
                        textStyle = TextStyle(fontSize = 18.sp),
```

```
                        colors = TextFieldDefaults.textFieldColors(
                            focusedLabelColor = Color.Black,
                            unfocusedLabelColor = Color.Black
                        ),
                        keyboardOptions = KeyboardOptions.Default.copy(
                            imeAction = ImeAction.Send
                        ),
                        modifier = Modifier.padding(start = 10.dp)
                    )

                    Spacer(modifier = Modifier.padding(10.dp))
                    Row(verticalAlignment = Alignment.CenterVertically) {
                        Text(
                            text: "Αποθήκευση Πάσου",
                            fontWeight = FontWeight.Bold,
                            fontSize = 15.sp,
                            modifier = Modifier.padding(start = 10.dp)
                        )
                        Checkbox(
                            checked = checked,
                            onCheckedChange = { checked = it }
                        )
                        Button(
                            onClick = {
                                searchById(regNumber, lastName)
                                saveId(checked, regNumber, lastName)
                            },
                            colors = ButtonDefaults.buttonColors(backgroundColor = Color( color: 0xff829aef))
                        ) {
                            Text(
                                text = "Αναζήτηση",
                                fontSize = 15.sp,
                                fontWeight = FontWeight.Bold
                            )
                        }
                    }
                    Spacer(modifier = Modifier.padding(10.dp))

                    Image(
                        painter = painterResource(id = R.drawable.qr_logo),
                        contentDescription = null,
                        modifier = Modifier
                            .size(180.dp)
                            .align(Alignment.CenterHorizontally)
                            .clickable {
                                onQrCodeClicked(true)
                            }
                    )
                }
            }
        }
    }
}

//Συνάρτηση για τη διαχείριση της αποσύνδεσης και μεταφορά του χρήστη στην αρχική σελίδα.
fun logout(context: Context) {
    val auth = FirebaseAuth.getInstance()
    auth.signOut()
    val intent = Intent(context, LoginActivity::class.java)
    context.startActivity(intent)

}

@Preview
@Composable
fun PreviewSearchId() {
    SearchId(searchById = { _, _ -> }, saveId = { _, _, _ -> }, onQrCodeClicked = {}, context = LocalContext.current, onFinishActivity
}
```


PreviewSearchId
Εύρεση Ακαδημαϊκής Ταυτότητας
Στοιχεία Φοιτητή
Αριθμός Μητρώου
Επώνυμο
Αποθήκευση Πάσου
Αναζήτηση

Το searchID.kt αποτελεί το Composable αρχείο για το UI της MainPage. Περιέχει μερικές συναρτήσεις οι οποίες εκτελούνται απευθείας από αρχείο αυτό χωρίς να μεταφερθεί η πληροφορία στο MainPage activity.

Αποτελείτε από ένα Box component που περιέχει μέσα όλα τα υπόλοιπα κομμάτια του UI. Έγινε έτσι με σκοπό τη προσθήκη του background χρώματος. Συγκεκριμένα εντός του box υπάρχουν μερικά rows και columns που στοιχίζουν ανάλογα τα text , image , TextField , Button, Checkbox.

Τα δύο TextField που υπάρχουν έχουν σκοπό ο χρήστης να πληκτρολογήσει στο ένα το Επώνυμο και στο άλλο τον Αριθμό Μητρώου του πάσου που θέλει να εμφανίσει.

Το CheckBox βρίσκεται εκεί για να μπορεί να επιλέξει ο χρήστης εάν θέλει να αποθηκεύσει το πάσο ή όχι.

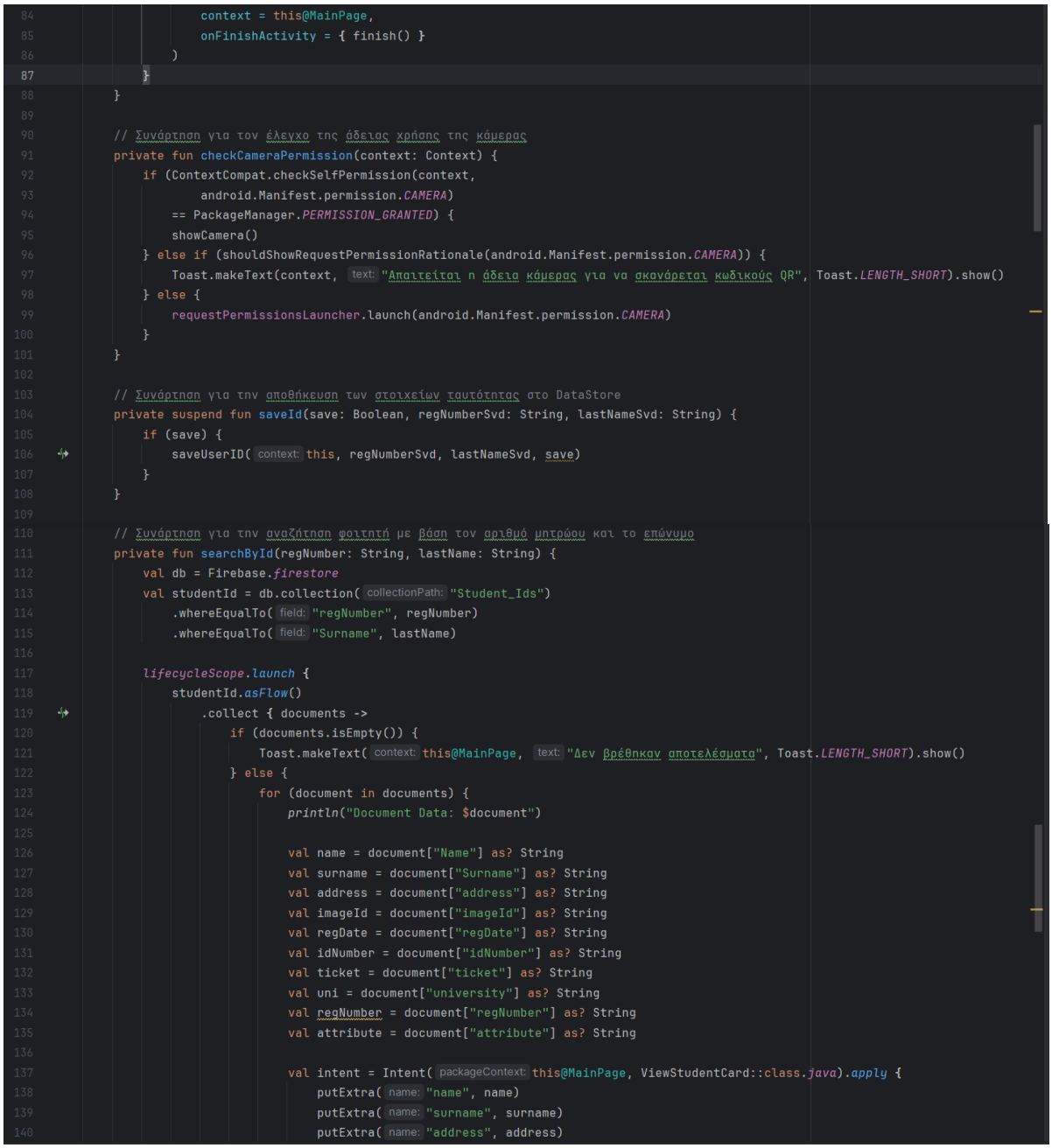
Το Button για την Αναζήτηση δίνει τη εντολή για την έναρξη της αναζήτησης εντός της βάσης δεδομένων για τα στοιχεία που δόθηκαν και εφόσον βρεθεί αποτέλεσμα μεταφέρονται στη οθόνη προβολής της ακαδημαϊκής ταυτότητας.

Η εικόνα με το σχήμα ενός QR Code έχει γίνει clickable και όταν πατηθεί ανοίγει η κάμερα σε λειτουργία για σκανάρισμα ενός κωδικού QR από κάποιο φοιτητικό πάσο. Εφόσον σκαναριστεί κάποιο πάσο και ο κωδικός του βρεθεί στη βάση δεδομένων τότε μεταφέρεται ο χρήστης στην οθόνη προβολής της ταυτότητας.

Το Button με το σήμα της αποσύνδεσης πραγματοποιεί την αποσύνδεση του χρήστη από την εφαρμογή με τη βοήθεια της συνάρτησης logout() ή οποία παίρνει το instance από το Firebase Authentication και πραγματοποιεί αποσύνδεση με την εντολή auth.singOut() και μετά μεταφέρει το χρήστη στην οθόνη σύνδεσης.

4.2.5 MainPage

```
package com.spyros.studentcard

import ...

class MainPage : ComponentActivity() {

    // Μεταβλητή για την αποθήκευση του αποτελέσματος του QR code
    private val qrResult = mutableStateOf( value: "")

    // Συνάρτηση για την εμφάνιση της κάμερας και τη σάρωση του QR code
    private fun showCamera(){
        val options = ScanOptions()
        options.setDesiredBarcodeFormats(ScanOptions.QR_CODE)
        options.setPrompt("Σκανάρετε ένα QR")
        options.setCameraId(0)
        options.setBeepEnabled(true)
        options.setOrientationLocked(true)

        barCodeLauncher.launch(options)
    }

    //Για τα αποτελέσματα της σάρωσης του QR code
    private val barCodeLauncher = registerForActivityResult(ScanContract()){
            result ->
        if (result.contents == null){
            Toast.makeText( context: this,  text: "Ακυρώθηκε", Toast.LENGTH_SHORT).show()
        } else {
            qrResult.value = result.contents
            Toast.makeText( context: this, result.contents, Toast.LENGTH_SHORT).show()
            Log.d( tag: "MainPage",  msg: "QR Result: ${qrResult.value}")
            searchByQr(qrResult.value)
        }
    }

    // Έλεγχος άδειας χρήσης κάμερας. Εάν δεν έχει δωθεί η άδεια τότε τη ζητάει.
    private val requestPermissionsLauncher = registerForActivityResult(ActivityResultContracts.RequestPermission()) {
            isGranted: Boolean ->
        if (isGranted) {
            showCamera()
        }
    }

    // Κλειδιά για την αποθήκευση των preferences στο DataStore
    private val SAVED_KEY = booleanPreferencesKey( name: "saved")
    private val REG_NUMBER_KEY = stringPreferencesKey( name: "regNumber")
    private val LAST_NAME_KEY = stringPreferencesKey( name: "lastName")

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            SearchId(
                searchById = { regNumber, lastName -> searchById(regNumber, lastName) },
                saveId = { save, regNumberSvd, lastNameSvd ->
                    lifecycleScope.launch {
                        saveId(save, regNumberSvd, lastNameSvd)
                    }
                },
                onQrCodeClicked = { clicked ->
                    if (clicked) {
                        checkCameraPermission( context: this@MainPage)
                    }
                },
```



```
                context = this@MainPage,
                onFinishActivity = { finish() }
            )
        }
    }

    // Συνάρτηση για τον έλεγχο της άδειας χρήσης της κάμερας
    private fun checkCameraPermission(context: Context) {
        if (ContextCompat.checkSelfPermission(context,
                android.Manifest.permission.CAMERA)
            == PackageManager.PERMISSION_GRANTED) {
            showCamera()
        } else if (shouldShowRequestPermissionRationale(android.Manifest.permission.CAMERA)) {
            Toast.makeText(context, text: "Απαιτείται η άδεια κάμερας για να σκανάρεται κωδικούς QR", Toast.LENGTH_SHORT).show()
        } else {
            requestPermissionsLauncher.launch(android.Manifest.permission.CAMERA)
        }
    }

    // Συνάρτηση για την αποθήκευση των στοιχείων ταυτότητας στο DataStore
    private suspend fun saveId(save: Boolean, regNumberSvd: String, lastNameSvd: String) {
        if (save) {
            saveUserID( context: this, regNumberSvd, lastNameSvd, save)
        }
    }

    // Συνάρτηση για την αναζήτηση φοιτητή με βάση τον αριθμό μητρώου και το επώνυμο
    private fun searchById(regNumber: String, lastName: String) {
        val db = Firebase.firestore
        val studentId = db.collection( collectionPath: "Student_Ids")
            .whereEqualTo( field: "regNumber", regNumber)
            .whereEqualTo( field: "Surname", lastName)

        lifecycleScope.launch {
            studentId.asFlow()
                .collect { documents ->
                    if (documents.isEmpty()) {
                        Toast.makeText( context: this@MainPage, text: "Δεν βρέθηκαν αποτελέσματα", Toast.LENGTH_SHORT).show()
                    } else {
                        for (document in documents) {
                            println("Document Data: $document")

                            val name = document["Name"] as? String
                            val surname = document["Surname"] as? String
                            val address = document["address"] as? String
                            val imageId = document["imageId"] as? String
                            val regDate = document["regDate"] as? String
                            val idNumber = document["idNumber"] as? String
                            val ticket = document["ticket"] as? String
                            val uni = document["university"] as? String
                            val regNumber = document["regNumber"] as? String
                            val attribute = document["attribute"] as? String

                            val intent = Intent( packageContext: this@MainPage, ViewStudentCard::class.java).apply {
                                putExtra( name: "name", name)
                                putExtra( name: "surname", surname)
                                putExtra( name: "address", address)
```

```
                                putExtra( name: "imageId", imageId)
                                putExtra( name: "regDate", regDate)
                                putExtra( name: "idNumber", idNumber)
                                putExtra( name: "ticket", ticket)
                                putExtra( name: "uni", uni)
                                putExtra( name: "regNumber", regNumber)
                                putExtra( name: "attribute", attribute)
                            }

                            startActivity(intent)
                            finish()
                        }
                    }
                }
            }
        }

        // Συνάρτηση για την αναζήτηση φοιτητή με βάση το αποτέλεσμα του QR code
        private fun searchByQr(qrResult: String) {
            Log.d( tag: "MainPage",  msg: "searchByQr called with qrResult: $qrResult")
            val db = Firebase.firestore
            val studentId = db.collection( collectionPath: "Student_Ids")
                .whereEqualTo( field: "idNumber", qrResult)

            lifecycleScope.launch {
                studentId.asFlow()
                    .collect { documents ->
                        if (documents.isEmpty()) {
                            Toast.makeText( context: this@MainPage,  text: "Δεν βρέθηκαν αποτελέσματα", Toast.LENGTH_SHORT).show()
                        } else {
                            for (document in documents) {
                                println("Document Data: $document")
                                val surname = document["Surname"] as? String
                                val regNumber = document["regNumber"] as? String

                                if (surname != null && regNumber != null) {
                                    searchById(regNumber, surname)
                                } else {
                                    Log.d( tag: "MainPage",  msg: "No matching document found for qrResult: $qrResult")
                                }
                            }
                        }
                    }
            }
        }

        // Συνάρτηση για την αποθήκευση των στοιχείων ταυτότητας στο DataStore
        private suspend fun saveUserID(context: Context, regNumber: String, lastName: String, save: Boolean) {
            context.dataStore.edit { preferences ->
                preferences[REG_NUMBER_KEY] = regNumber
                preferences[LAST_NAME_KEY] = lastName
                preferences[SAVED_KEY] = save
            }
        }
    }
```

Το MainPage Activity αποτελεί την κύρια οθόνη της εφαρμογής, για UI έχει οριστεί το searchID.kt Composable το οποίο επιστρέψει ορισμένες συναρτήσεις ανάλογα με τις επιλογές του χρήστη δηλαδή ανάλογα τι κουμπί έχει πατήσει.

Το searchById παίρνει το regNumber και το lastName που έχει δώσει ο χρήστη στα πεδία Input του UI και τα δίνει στη συνάρτηση searchById, η οποία παίρνει τις μεταβλητές αυτές και κάνει αναζήτηση στο Firestore μέσα στο Collection «Student_Ids», το οποίο περιέχει τις ακαδημαϊκές ταυτότητες, και έπειτα αναζητεί για το document στο οποίο τα fields “regNumber” και “Surname” ταιριάζουν με αυτά τα οποία εισήγαγε ο χρήστης. Στην περίπτωση που βρεθεί document που ταιριάζει στα κριτήρια αναζήτησης τότε τραβάει τη ροή των δεδομένων και βάζει σε αντίστοιχες μεταβλητές τύπου nullable String όλα τα δεδομένα της ακαδημαϊκής ταυτότητας. Τις μεταβλητές αυτές ύστερα τις βάζει στο putExtra του intent και μετά κάνει έναρξη της επόμενης οθόνης ViewStudentCard στη οποία θα προβληθούν όλα τα δεδομένα.

Το saveId λαμβάνει από το searchId τη μεταβλητή save καθώς και το regNumberSvd , lastNameSvd τα οποία κρατούν τις πληροφορίες για το εάν ο χρήστης πάτησε να αποθηκευτεί το πάσο, τον αριθμό μητρώου και το επώνυμο αντίστοιχα, έπειτα μεταφέρονται στην saveId συνάρτηση του activity. Η συνάρτηση αυτή ελέγχει αν το save που είναι τύπου Boolean είναι true, εάν ναι τότε μεταφέρει τα regNumberSvd και lastNameSvd στην συνάρτηση saveUserID, η οποία παίρνει τις μεταβλητές και τις αποθηκεύει στο Preferences DataStore. Ο σκοπός είναι κατά την εκκίνηση της εφαρμογής να γίνει έλεγχος για το αν είναι αποθηκευμένα στοιχεία στο DataStore και αν το save είναι true τότε να γίνει αυτόματα αναζήτηση για την ταυτότητα κατά την έναρξη της εφαρμογής.

Η onQrCodeClicked είναι μια συνάρτηση που ενεργοποιείται όταν ο χρήστης πατάει την εικόνα με το εικονίδιο του QR το οποίο έχει γίνει Clickable. Σκοπός της είναι η ενεργοποίηση της κάμερας και το σκανάρισμα ενός QR κωδικού και η αναζήτηση με το αποτέλεσμα στη βάση δεδομένων για την εύρεση της ακαδημαϊκής ταυτότητας που ταιριάζει με αυτά τα στοιχεία.

Αρχικά ξεκινάει ελέγχοντας εάν υπάρχουν οι κατάλληλες άδειες, συγκεκριμένα η άδεια για την ενεργοποίηση της κάμερας. Εάν δεν έχουν δοθεί η κατάλληλες άδειες τότε εμφανίζει ένα pop-up παράθυρο που ζητάει τις άδειες, εάν δοθούν τότε προχωράει και ανοίγει τη κάμερα, εάν όχι τότε δεν κάνει τίποτα και την επόμενη φορά που θα πατηθεί το κουμπί θα εμφανιστεί ένα Toast Message που θα λέει ότι απαιτείται η άδεια της κάμερας για να λειτουργήσει το QR Scanner.

Για τη λειτουργία του σκάνερ χρησιμοποιήθηκε η βιβλιοθήκη ZXing που παρέχει ολοκληρωμένη και εύκολη λειτουργία για QR Scanner.

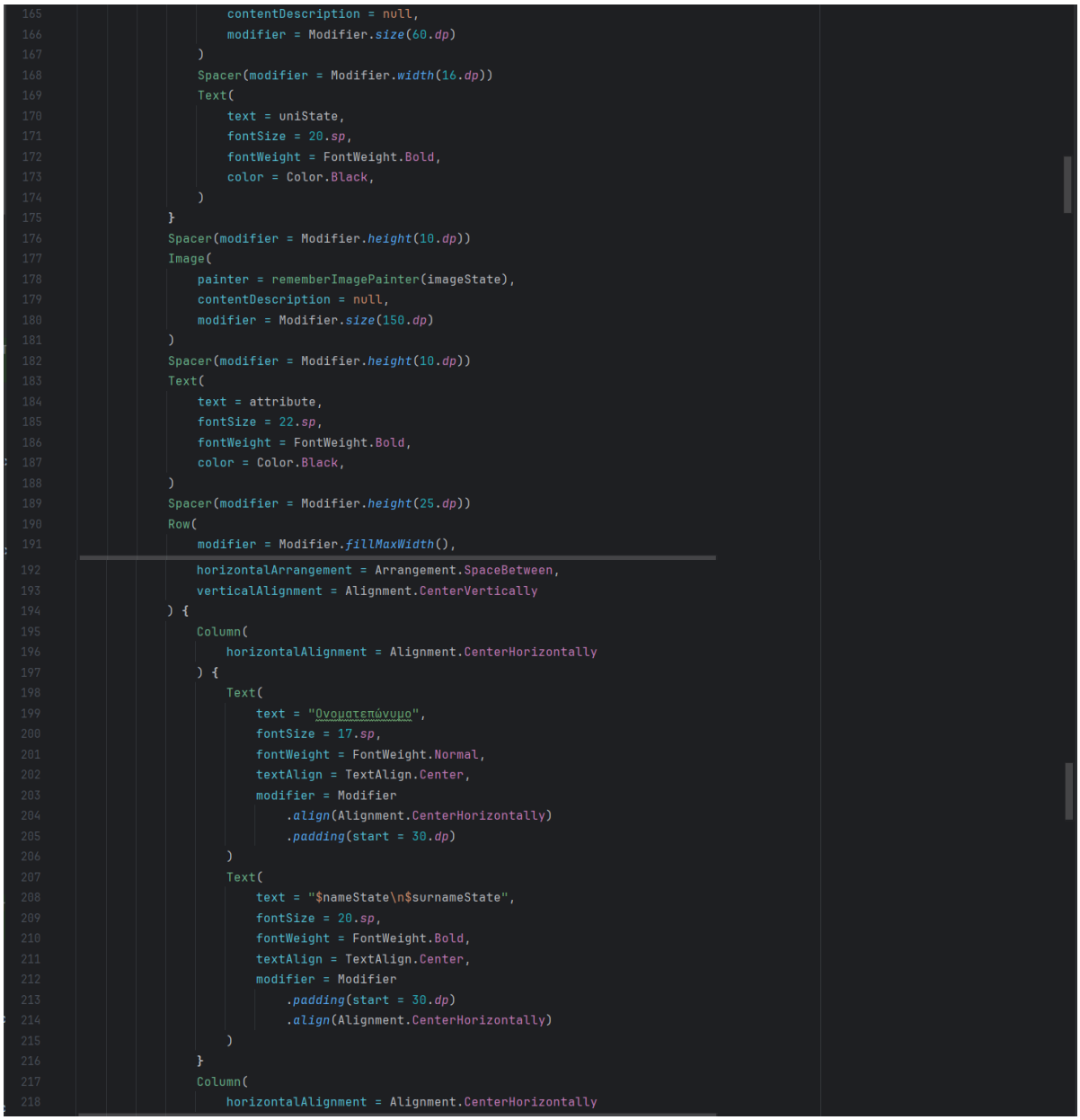
Η μέθοδος showCamera() ενεργοποιείται εφόσον έχει δοθεί η απαραίτητη άδεια τη πρόσβαση στη κάμερα. Ανοίγει τη κάμερα με σκοπό να σκανάρει έναν κωδικό QR, αφού βρεθεί κάποιος κωδικός τότε μέσω της μεταβλητής barCodeLauncher διαχειρίζεται το αποτέλεσμα. Η μεταβλητή αυτή, ελέγχει εάν το αποτέλεσμα είναι null δηλαδή ο χρήστης ακύρωσε την αναζήτηση, τότε εμφανίζει ένα Toast Message που λέει ακυρώθηκε και κλείνει τη κάμερα. Εάν το αποτέλεσμα δεν είναι null δηλαδή ο χρήστης σκάναρε έναν κωδικό τότε εμφανίζει το αποτέλεσμα σε ένα Toast Message και δίνει το αποτέλεσμα στην μέθοδο searchByQr η οποίο το αξιοποιεί για να βρει εάν τα στοιχεία αντιστοιχούν σε κάποια ακαδημαϊκή ταυτότητα.

Η λειτουργία της searchByQr είναι παρόμοια με την λειτουργία της searchById με τη διαφορά ότι κάνει αναζήτηση στη βάση δεδομένων και συνεπώς στο document , με βάση τον αριθμό ταυτότητας και όχι με βάση τον αριθμό μητρώου και το επώνυμο. Αυτό γίνεται διότι στις ακαδημαϊκές ταυτότητες το QR που έχουν πάνω περιέχει τον αριθμό της ταυτότητας χωρίς άλλα στοιχεία του φοιτητή.

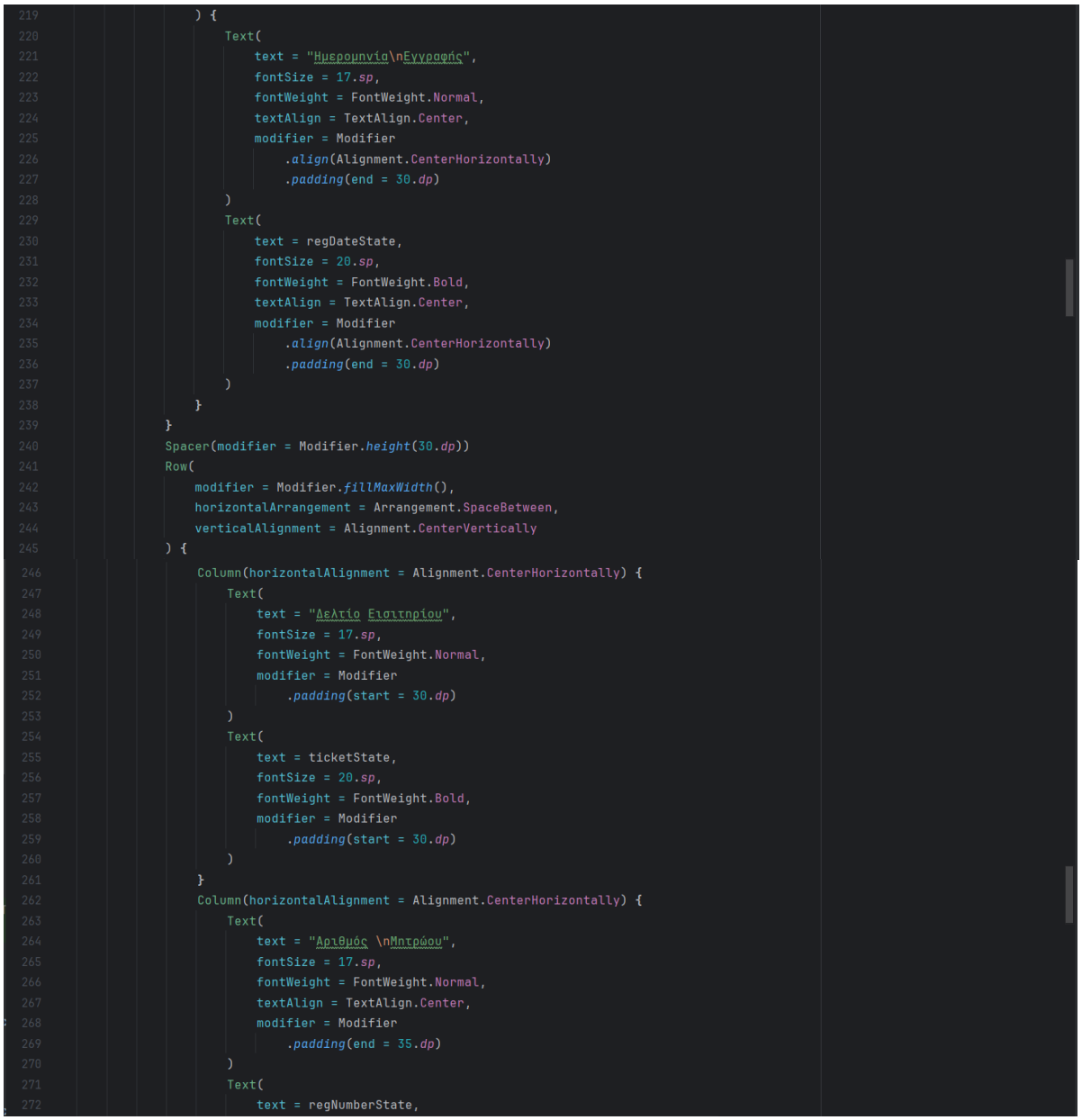
4.2.6 StudentCard.kt

```kotlin
package com.spyros.studentcard.composeAssets

import ...

val Context.dataStore by preferencesDataStore(name = "settings")


@OptIn(ExperimentalCoilApi::class)
@Composable
fun StudentCard(
    name: String,
    surname: String,
    address: String,
    imageId: String,
    regDate: String,
    idNumber: String,
    ticket: String,
    uni: String,
    uniImage: Int,
    regNumber: String,
    attribute: String,
    context: Context,
    returnBtn: (Boolean, Context) -> Unit
) {
    var showDialog by remember { mutableStateOf( value: false) }
    val qrCodeBitmap = remember { mutableStateOf(generateQRCode(idNumber)) }

/*
Δημιουργεία των MutableStateFlows για την διαχείριση των καταστάσεων των μεταβλητών
που κρατάνε τα στοιχεια του πάσου. Οι τιμές των flows συλλέγονται ως State χρησιμοποιώντας την
συνάρτηση collectAsState, επιτρέποντας την αυτόματη ενημέρωση του UI κάθε φορά που αλλάζουν
οι καταστάσεις.

    val uniFlow = MutableStateFlow(uni)
    val uniState by uniFlow.collectAsState()

    val nameFlow = MutableStateFlow(name)
    val nameState by nameFlow.collectAsState()

    val surnameFlow = MutableStateFlow(surname)
    val surnameState by surnameFlow.collectAsState()

    val regDateFlow = MutableStateFlow(regDate)
    val regDateState by regDateFlow.collectAsState()

    val ticketFlow = MutableStateFlow(ticket)
    val ticketState by ticketFlow.collectAsState()

    val imageFlow = MutableStateFlow(imageId)
    val imageState by imageFlow.collectAsState()

    val idNumberFlow = MutableStateFlow(idNumber)
    val idNumberState by idNumberFlow.collectAsState()

    val addressFlow = MutableStateFlow(address)
    val addressState by addressFlow.collectAsState()

    val regNumberFlow = MutableStateFlow(regNumber)
    val regNumberState by regNumberFlow.collectAsState()

    var returnClicked by remember { mutableStateOf( value: true) }
```

```
105
106     // Εμφάνιση του QR code μέσω του alertDialog Composable όταν Ο χρήστης πατάει το Floating Action Button
107     if (showDialog) {
108         AlertDialog(onDismiss = { showDialog = false }, qrCodeBitmap = qrCodeBitmap.value , idNumber = idNumberState)
109     }
110
111     Box(
112         modifier = Modifier
113             .fillMaxSize()
114             .drawBehind {
115                 val brush = Brush.verticalGradient(
116                     colors = listOf(Color.White, Color( color: 0xFF112673)),
117                     startY = 0f,
118                     endY = size.height
119                 )
120                 drawRect(brush = brush)
121             }
122     ) {
123         Button(
124             onClick = {
125                 returnClicked = true
126                 returnBtn(returnClicked, context)
127             },
128             colors = androidx.compose.material.ButtonDefaults.buttonColors(
129                 backgroundColor = Color.Transparent
130
131             ),
132             elevation = null,
133             modifier = Modifier
134                 .border(0.dp, Color.Transparent)
136         ) {
137             Image(
138                 painter = painterResource(id = R.drawable.back_arrow),
139                 contentDescription = null,
140                 Modifier.size(30.dp)
141             )
142         }
143         Column(
144             modifier = Modifier.fillMaxSize(),
145             horizontalAlignment = Alignment.CenterHorizontally
146         ) {
147             Spacer(modifier = Modifier.height(5.dp))
148             Text(
149                 text = "Academic ID \nΑκαδημαϊκή Ταυτότητα",
150                 textAlign = TextAlign.Center,
151                 fontSize = 26.sp,
152                 fontWeight = FontWeight.Bold,
153                 color = Color.Black,
154                 modifier = Modifier.padding(top = 10.dp)
155             )
156             Spacer(modifier = Modifier.height(20.dp))
157
158             // Προβολή όλων των στοιχείων του πάσου που τράβηκε απο τα Flow state
159
160             Row(
161                 verticalAlignment = Alignment.CenterVertically,
162             ) {
163                 Image(
164                     painter = painterResource(id = uniImage),
```

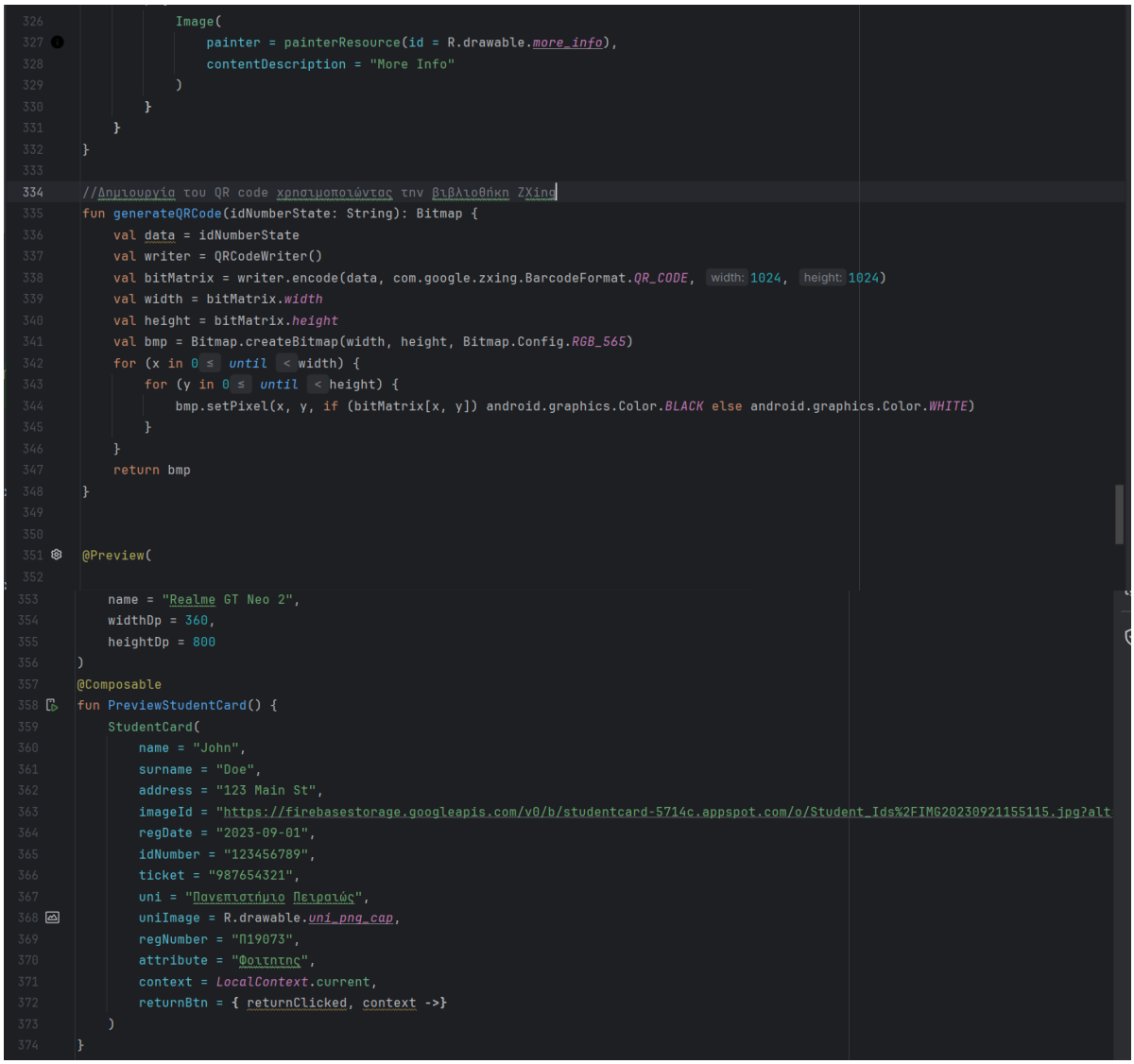


```
                contentDescription = null,
                modifier = Modifier.size(60.dp)
            )
            Spacer(modifier = Modifier.width(16.dp))
            Text(
                text = uniState,
                fontSize = 20.sp,
                fontWeight = FontWeight.Bold,
                color = Color.Black,
            )
        }
        Spacer(modifier = Modifier.height(10.dp))
        Image(
            painter = rememberImagePainter(imageState),
            contentDescription = null,
            modifier = Modifier.size(150.dp)
        )
        Spacer(modifier = Modifier.height(10.dp))
        Text(
            text = attribute,
            fontSize = 22.sp,
            fontWeight = FontWeight.Bold,
            color = Color.Black,
        )
        Spacer(modifier = Modifier.height(25.dp))
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceBetween,
            verticalAlignment = Alignment.CenterVertically
        ) {
            Column(
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                Text(
                    text = "Ονοματεπώνυμο",
                    fontSize = 17.sp,
                    fontWeight = FontWeight.Normal,
                    textAlign = TextAlign.Center,
                    modifier = Modifier
                        .align(Alignment.CenterHorizontally)
                        .padding(start = 30.dp)
                )
                Text(
                    text = "$nameState\n$surnameState",
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Bold,
                    textAlign = TextAlign.Center,
                    modifier = Modifier
                        .padding(start = 30.dp)
                        .align(Alignment.CenterHorizontally)
                )
            }
            Column(
                horizontalAlignment = Alignment.CenterHorizontally
```



```
            ) {
                Text(
                    text = "Ημερομηνία\nΕγγραφής",
                    fontSize = 17.sp,
                    fontWeight = FontWeight.Normal,
                    textAlign = TextAlign.Center,
                    modifier = Modifier
                        .align(Alignment.CenterHorizontally)
                        .padding(end = 30.dp)
                )
                Text(
                    text = regDateState,
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Bold,
                    textAlign = TextAlign.Center,
                    modifier = Modifier
                        .align(Alignment.CenterHorizontally)
                        .padding(end = 30.dp)
                )
            }
        }
        Spacer(modifier = Modifier.height(30.dp))
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceBetween,
            verticalAlignment = Alignment.CenterVertically
        ) {
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
                Text(
                    text = "Δελτίο Εισιτηρίου",
                    fontSize = 17.sp,
                    fontWeight = FontWeight.Normal,
                    modifier = Modifier
                        .padding(start = 30.dp)
                )
                Text(
                    text = ticketState,
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Bold,
                    modifier = Modifier
                        .padding(start = 30.dp)
                )
            }
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
                Text(
                    text = "Αριθμός \nΜητρώου",
                    fontSize = 17.sp,
                    fontWeight = FontWeight.Normal,
                    textAlign = TextAlign.Center,
                    modifier = Modifier
                        .padding(end = 35.dp)
                )
                Text(
                    text = regNumberState,
```

```
                        fontSize = 20.sp,
                        fontWeight = FontWeight.Bold,
                        modifier = Modifier
                            .padding(end = 35.dp)
                    )
                }
            }
            Spacer(modifier = Modifier.height(20.dp))
            Column(
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                Text(
                    text = "Αριθμός \nΤαυτότητας",
                    fontSize = 17.sp,
                    textAlign = TextAlign.Center,
                    fontWeight = FontWeight.Normal,
                )
                Text(
                    text = idNumberState,
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Bold,
                )
            }
            Spacer(modifier = Modifier.height(20.dp))
            Column(
                horizontalAlignment = Alignment.CenterHorizontally
            ) {
                Text(
                    text = "Διεύθυνση",
                    fontSize = 17.sp,
                    fontWeight = FontWeight.Normal,
                )
                Text(
                    text = addressState,
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Bold,
                    modifier = Modifier.padding(start = 10.dp)
                )
            }
        }


        //Εμφάνιση του QR του αριθμού Ταυτότητας

        FloatingActionButton(
            onClick = { showDialog = true },
            backgroundColor = Color.LightGray,
            contentColor = Color.White,
            modifier = Modifier
                .align(Alignment.BottomEnd)
                .padding(24.dp)
                .size(45.dp)
        ) {
```



```
                Image(
                    painter = painterResource(id = R.drawable.more_info),
                    contentDescription = "More Info"
                )
            }
        }
    }

    //Δημιουργία του QR code χρησιμοποιώντας την βιβλιοθήκη ZXing
    fun generateQRCode(idNumberState: String): Bitmap {
        val data = idNumberState
        val writer = QRCodeWriter()
        val bitMatrix = writer.encode(data, com.google.zxing.BarcodeFormat.QR_CODE, width: 1024, height: 1024)
        val width = bitMatrix.width
        val height = bitMatrix.height
        val bmp = Bitmap.createBitmap(width, height, Bitmap.Config.RGB_565)
        for (x in 0 ≤ until < width) {
            for (y in 0 ≤ until < height) {
                bmp.setPixel(x, y, if (bitMatrix[x, y]) android.graphics.Color.BLACK else android.graphics.Color.WHITE)
            }
        }
        return bmp
    }


    @Preview(

        name = "Realme GT Neo 2",
        widthDp = 360,
        heightDp = 800
    )
    @Composable
    fun PreviewStudentCard() {
        StudentCard(
            name = "John",
            surname = "Doe",
            address = "123 Main St",
            imageId = "https://firebasestorage.googleapis.com/v0/b/studentcard-5714c.appspot.com/o/Student_Ids%2FIMG20230921155115.jpg?alt
            regDate = "2023-09-01",
            idNumber = "123456789",
            ticket = "987654321",
            uni = "Πανεπιστήμιο Πειραιώς",
            uniImage = R.drawable.uni_png_cap,
            regNumber = "Π19073",
            attribute = "Φοιτητης",
            context = LocalContext.current,
            returnBtn = { returnClicked, context ->}
        )
    }
```


PreviewStudentCard - Realme GT Neo 2
Academic ID
Ακαδημαική Ταυτότητα
Πανεπιστήμιο Πειραιώς
Φοιτητης
Ονοματεπώνυμο
Spyros
Kokkas
Ημερομηνία Εγγραφής
2019-09-01
Δελτίο Εισιτηρίου
09/26
Αριθμός Μητρώου
Π19073
Αριθμός Ταυτότητας
123456789
Διεύθυνση
Kifissias Leof. 1

Το αρχείο StudentCard.kt αποτελεί το Composable UI για την προβολή των στοιχείων της ακαδημαϊκής ταυτότητας των φοιτητών. Εδώ θα εμφανιστούν όλες οι απαραίτητες πληροφορίες τους που σχετίζονται με το πανεπιστήμιο. Αποτελείτε από 2 Images τα οποία αλλάζουν ανάλογα με 1. Το πανεπιστήμιο του φοιτητή και 2. Την φωτογραφία της ταυτότητας που υπάρχει στη βάση δεδομένων. Η πρώτη φωτογραφία περιέχει το λογότυπο του πανεπιστημίου του φοιτητή και δίπλα ακριβώς περιέχει το όνομα του πανεπιστημίου το οποίο αλλάζει ανάλογα με το πανεπιστήμιο φοίτησης του χρήστη. Υπάρχουν πεδία για το ονοματεπώνυμο, την ημερομηνία εγγραφής , το δελτίο εισιτηρίου , τον αριθμό μητρώου, τον αριθμό ταυτότητας και τη διεύθυνση του φοιτητή.

Τα πεδία αυτά λαμβάνουν τα δεδομένα τους μέσα από τη βάση δεδομένων. Για την διαχείριση των δεδομένων που εμφανίζονται χρησιμοποίησα το Flow της Kotlin που επιτρέπει τη διαχείριση των δεδομένων ασύγχρονα. Συγκεκριμένα χρησιμοποίησα το StateFlow που αποτελεί μια ειδική μορφή του Flow το οποίο διασφαλίζει ότι το UI ενημερώνεται άμεσα με κάθε αλλαγή.

Κάθε πληροφορία που εμφανίζεται στο UI αποθηκεύεται σε ξεχωριστά MutableStateFlow τα οποία αρχικοποιούνται με τιμές από τη βάση δεδομένων. Χάρη στη χρήση του StateFlow, η UI ανανεώνεται αυτόματα όποτε αλλάζει η τιμή σε οποιοδήποτε από τα StateFlow. Για παράδειγμα, αν αλλάξει το πανεπιστήμιο του φοιτητή στη βάση δεδομένων, η εικόνα και το όνομα του πανεπιστημίου στο StudentCard θα ενημερωθούν αμέσως χωρίς να χρειάζεται να κάνεις κάτι επιπλέον.

Το StateFlow παρέχει ένα ασφαλές τρόπο διαχείρισης και ενημέρωσης της κατάστασης στην UI. Διασφαλίζει ότι η UI παραμένει ενημερωμένη με τις τελευταίες αλλαγές στα δεδομένα.

Στην αριστερή κορυφή της οθόνης υπάρχει και ένα κουμπί με σύμβολο το βέλος της επιστροφής, αυτό το κουμπί εφόσον πατηθεί επιστρέφει το χρήστη στην οθόνη αναζήτησης και επιπλέον εφόσον έχει αποθηκευτεί πάσο, το σβήνει από το DataStore που σημαίνει ότι με την επόμενη έναρξη της εφαρμογής θα χρειαστεί να γίνει ξανά αναζήτηση.

Κάτω δεξιά υπάρχει ένα κυκλικό κουμπί το οποίο εμφανίζει ένα alert Dialog το οποίο περιέχει το QR Code της ταυτότητας, όπως και η φυσική πλαστική ταυτότητα. Το QR δημιουργείτε από την generateQRCode μέθοδο η οποία επιστρέφει ένα bitmap. Παίρνει τον αριθμό της ταυτότητας και με τη βοήθεια της βιβλιοθήκης ZXing δημιουργεί ένα bitmap που περιέχει το κωδικό QR τον οποίο λαμβάνει το AlertDialog.kt και το προβάλλει.

4.2.7 AlertDialog.kt

```kotlin
package com.spyros.studentcard.composeAssets

import ...

@Composable
// Alert Dialog για την εμφάνιση του QR Code της Ακαδημαϊκής Ταυτότητας
fun AlertDialog(onDismiss: () -> Unit, qrCodeBitmap: Bitmap, idNumber: Any) {
    androidx.compose.material3.AlertDialog(
        onDismissRequest = onDismiss,
        confirmButton = { },
        modifier = Modifier.height(350.dp), // Το μέγεθος του παραθύρου
        title = {
            Row(
                verticalAlignment = Alignment.CenterVertically
            ) {
                Icon(imageVector = Icons.Default.QrCodeScanner, contentDescription = "QR Code")
                Text(
                    text = idNumber.toString(),
                    fontSize = 20.sp,
                    fontWeight = FontWeight.Bold
                )
            }
        },
        text = {
            Column(
                modifier = Modifier.fillMaxSize(),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            ) {
                Image(
                    bitmap = qrCodeBitmap.asImageBitmap(),
                    contentDescription = idNumber.toString(),
                    modifier = Modifier // Το μέγεθος του QR
                        .width(512.dp)
                        .height(512.dp)
                        .padding(10.dp)
                )
            }
        }
    )
}

@Preview
@Composable
fun AlertDialogPreview() {
    val qrCodeBitmap = Bitmap.createBitmap( width: 100,  height: 100, Bitmap.Config.ARGB_8888)
    AlertDialog(onDismiss = {}, qrCodeBitmap = qrCodeBitmap, idNumber = "12843254323")

}
```

Το AlertDialog.kt αποτελεί το Composable πάνω στο οποίο θα αποτυπωθεί το QR Code της ακαδημαϊκής ταυτότητας. Συγκεκριμένα, λαμβάνει το bitmap του qr code που δημιουργήθηκε με την έναρξη την οθόνης στο Composable StudentCard.kt και το εμφανίζει ως ένα Alert Dialog συγκεκριμένου μεγέθους ώστε να μπορεί να εμφανιστεί το QR σε καλό βαθμό για να μπορεί να σκαναριστεί εφόσον απαιτηθεί.

4.2.8 ViewStudentCard

```kotlin
package com.spyros.studentcard

import ...

class ViewStudentCard : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Λήψει των στοιχείων του φοιτητή απο τη προηγούμενη οθόνη
        val name = intent.getStringExtra( name: "name") ?: ""
        val surname = intent.getStringExtra( name: "surname") ?: ""
        val address = intent.getStringExtra( name: "address") ?: ""
        val imageId = intent.getStringExtra( name: "imageId") ?: ""
        val regDate = intent.getStringExtra( name: "regDate") ?: ""
        val idNumber = intent.getStringExtra( name: "idNumber") ?: ""
        val ticket = intent.getStringExtra( name: "ticket") ?: ""
        val uni = intent.getStringExtra( name: "uni") ?: ""
        val regNumber = intent.getStringExtra( name: "regNumber") ?: ""
        val attribute = intent.getStringExtra( name: "attribute") ?: ""

        val uniImage = addUniImage(uni)

        val storage = Firebase.storage
        val storageRef = storage.getReferenceFromUrl(imageId)

        storageRef.downloadUrl.addOnSuccessListener { uri ->
            val imageUrl = uri.toString()
            setContent {
                StudentCardTheme {
                    StudentCard(
                        // Μεταφορά των στοιχείων του φοιτητή στην οθόνη προβολής
                        name = name,
                        surname = surname,
                        address = address,
                        imageId = imageUrl,
                        regDate = regDate,
                        idNumber = idNumber,
                        ticket = ticket,
                        uni = uni,
                        uniImage = uniImage,
                        regNumber = regNumber,
                        attribute = attribute,
                        context = this@ViewStudentCard,
                        returnBtn = { returnBtn , context ->
                            if (returnBtn) {
                                returnBtnFun(context)
                            }
                        }
                    )
                }
            }
        }.addOnFailureListener {
            Toast.makeText( context: this,  text: "Σφάλμα φόρτωσης εικόνας", Toast.LENGTH_SHORT).show()
        }
    }

    // Προσθήκη της κατάλληλης εικόνα με βάση το πανεπιστήμιο του φοιτητή
    private fun addUniImage(uni: String): Int {
        return if (uni == "Πανεπιστήμιο Πειραιώς") {
```

```
                R.drawable.unipi_logo
            } else {
                R.drawable.uni_png_cap
            }
        }

        // Κουμπί επιστροφής στην οθόνη αναζήτησης με ταυτόχρονή διαγραφή των preferences απο το DataStore
        // Δηλαδή το αποθηκευμένο πάσο διαγράφεται.
        private fun returnBtnFun(context: Context) {
            CoroutineScope(Dispatchers.IO).launch {
                context.dataStore.edit { preferences ->
                    preferences.clear()
                }
            }
            val intent = Intent( packageContext: this, MainPage::class.java)
            startActivity(intent)
            finish()
        }
    }

```

Η ViewStudentCard αποτελεί το λειτουργικό κομμάτι για την προβολή της ακαδημαϊκής ταυτότητας. Συγκεκριμένα, μέσα από το Intent παίρνει όλες τις πληροφορίες της ταυτότητας από τη προηγούμενη οθόνη της αναζήτησης και τις μεταφέρει στη StudentCard οθόνη μέσα από το setContent. Οι πληροφορίες που λαμβάνει από τη προηγούμενη οθόνη είναι το όνομα, το επώνυμο, την ημερομηνία εγγραφής, τον αριθμό μητρώου, το δελτίο εισιτηρίου, το όνομα του πανεπιστημίου, τον αριθμό ταυτότητας και την ιδιότητα του. Επιπλέον λαμβάνει το link (σύνδεσμο) για την εικόνα του φοιτητή στη βάση, το οποίο σημαίνει ότι απαιτείται σύνδεση στο διαδίκτυο για να την αποθηκεύσει. Η εικόνα του πανεπιστημίου εντοπίζεται τοπικά εντός της συσκευής από τη μέθοδο addUniImage() με βάση το όνομα του πανεπιστημίου, προσωρινά και για τους σκοπούς ανάπτυξης της εφαρμογής έχει προστεθεί μόνο το Πανεπιστήμιο Πειραιώς και η εικόνα του στη εφαρμογή, σε μελλοντική αναβάθμιση θα προστεθούν και τα υπόλοιπα πανεπιστήμια.

Επιπλέον υπάρχει και η μέθοδος returnBtnFun() η οποία διαχειρίζεται το κουμπί της επιστροφής που όπως αναφέρθηκε στην οθόνη προβολής υπάρχει πάνω αριστερά. Συγκεκριμένα όταν το κουμπί της επιστροφής πατηθεί τότε στο setContent το returnBtn λαμβάνει τη returnBtn μεταβλητή τύπου Boolean και ελέγχει εάν είναι true, εφόσον είναι προχωράει στη μέθοδο returnBtnFun() η οποία σβήνει τις αποθηκευμένες τιμές εντός του Preferences DataStore και έπειτα μεταφέρει το χρήστη στην οθόνη αναζήτησης κλείνοντας τη τρέχουσα οθόνη.

4.2.9 DataStoreSingleton

```
1  package com.spyros.studentcard
2
3
4  import ...
8
9  // Μετατρέπει το Preferences DataStore σε singleton για την αποθήκευση των στοιχείων του χρήστη
10 // και την αποφυγεί σφαλμάτων διπλότυπων DataStore
11
12 val Context.dataStore: DataStore<Preferences> by preferencesDataStore(name = "settings")
```

Σκοπός του αρχείου αυτού είναι να δημιουργήσει ένα **Singleton** για το Preferences DataStore χρησιμοποιώντας το by delegate, ώστε να υπάρχει μόνο μία μοναδική παρουσία του DataStore κατά τη διάρκεια ζωής της εφαρμογής, διασφαλίζοντας έτσι την ορθή και αποδοτική διαχείριση των δεδομένων της εφαρμογής.

Δημιουργήθηκε ως μέσο Debugging διότι σε κάθε αποθήκευση δημιουργούταν και ένα καινούργιο DataStore που είχε ως αποτέλεσμα όταν η εφαρμογή προσπαθούσε να ανακτήσει τα δεδομένα να τερματίζει η εφαρμογή (Crash) διότι υπήρχαν πάνω από μία εγγραφές DataStore με το ίδιο όνομα.

4.2.10 Flow.kt

```kotlin
package com.spyros.studentcard.firestore

import ...


// Η συνάρτηση αυτή χρησιμοποιεί τη μέθοδο asFlow για να μετατρέψει ένα Firestore Query σε Kotlin Flow
// με την δυνατότητα να βλέπουμε τις αλλαγές σε πραγματικό χρόνο
fun Query.asFlow(): Flow<List<Map<String, Any>>> = callbackFlow {
    val listenerRegistration = addSnapshotListener { snapshot, error ->
        if (error != null) {
            close(error)
            return@addSnapshotListener
        }
        val documents = snapshot?.documents?.map { it.data.orEmpty() } ?: emptyList()
        trySend(documents).isSuccess
    }
    awaitClose { listenerRegistration.remove() }
}
```

Σκοπός του συγκεκριμένου αρχείου είναι να μετατρέψει ένα Firestore Query σε ένα Kotlin Flow, επιτρέποντας την παρακολούθηση των αλλαγών στα δεδομένα της Firestore σε πραγματικό χρόνο. Χρησιμοποιώντας το callbackFlow, ο SnapshotListener προσθέτει τα νέα δεδομένα στο Flow, τα οποία μπορούν να προβληθούν άμεσα σε άλλα μέρη της εφαρμογής. Αν παρουσιαστεί κάποιο σφάλμα, σταματάει και το Flow κλείνει με αναφορά του σφάλματος.

Κεφάλαιο 5ο

5 Σημαντικά σημεία ανάπτυξης

Θα παρουσιάσω μερικά από τα πιο σημαντικά σημεία της ανάπτυξης της εφαρμογής

1. Πλοήγηση εντός της εφαρμογής

Για την πλοήγηση ανάμεσα στις διαφορετικές οθόνες της εφαρμογής, χρησιμοποίησα το παραδοσιακό σύστημα Intent του Android framework. Κάθε Activity που ανοίγει περιέχει Composable UI στοιχεία, τα οποία συνδυάζουν το μοντέρνο Jetpack Compose με τις ήδη υπάρχουσες λειτουργίες του Android, όπως ο έλεγχος ταυτότητας και η μετάβαση μέσω Intent.

2. Σύνδεση με Firebase

Η σύνδεση με το Firebase έγινε ακολουθώντας τις οδηγίες του Android Studio και του Firebase Dashboard. Αρχικά, πρόσθεσα τα απαραίτητα dependencies για το Firebase Authentication και το Firestore στο αρχείο build.gradle της εφαρμογής και έκανα συγχρονισμό του Gradle. Στη συνέχεια, δημιούργησα ένα νέο project στο Firebase console, το οποίο ονόμασα σύμφωνα με το project του Android Studio. Μέσω των οδηγιών σύνδεσης, κατάφερα να ενσωματώσω την εφαρμογή με το Firebase project, συνδέοντας τα δύο projects μεταξύ τους (Android και Firebase). Μετά τη σύνδεση, η αρχικοποίηση του Firebase γίνεται σε κάθε Activity ή Composable όπου χρειάζεται πρόσβαση σε Firebase υπηρεσίες, όπως η αυθεντικοποίηση ή η αποθήκευση δεδομένων στο Firestore.

3. Singleton Desing Pattern για το DataStore

Στην εφαρμογή, χρησιμοποίησα το **Singleton Design Pattern** για τη διαχείριση του Preferences DataStore. Το Singleton διασφαλίζει ότι υπάρχει μόνο μία και μοναδική παρουσία του DataStore καθ' όλη τη διάρκεια λειτουργίας της εφαρμογής, γεγονός που αποτρέπει τη δημιουργία πολλαπλών περιπτώσεων του DataStore και τη σύγκρουση των δεδομένων (conflict of data). Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη όταν θέλουμε να διατηρούμε σταθερή την κατάσταση της εφαρμογής και να αποφεύγουμε σφάλματα που μπορεί να οδηγήσουν σε τερματισμό (crashes).
Κατά την ανάπτυξη, προέκυψε το πρόβλημα ότι κάθε αποθήκευση δεδομένων δημιουργούσε ένα νέο instance του DataStore, το οποίο οδηγούσε σε πολλαπλές εγγραφές και ασυνέπειες στα δεδομένα. Με την υλοποίηση του Singleton,

εξασφάλισα ότι μόνο ένα instance του DataStore θα υπάρχει καθ' όλη τη διάρκεια χρήσης, κάτι που βελτίωσε την αποδοτικότητα της εφαρμογής και απέτρεψε τα σφάλματα.
Η χρήση του Singleton pattern βοήθησε επίσης στη διαχείριση της μνήμης και στην εξασφάλιση ότι τα δεδομένα είναι πάντα συγχρονισμένα σε όλα τα μέρη της εφαρμογής που χρειάζονται πρόσβαση σε αυτά. Αυτή η προσέγγιση είναι σημαντική για εφαρμογές που χρησιμοποιούν προτιμήσεις χρήστη ή άλλα επίμονα δεδομένα.

4. Real-Time Data Sync με Flow

Στην εφαρμογή, χρησιμοποίησα το **Kotlin Flow** για τη διαχείριση και τον συγχρονισμό δεδομένων σε πραγματικό χρόνο από το Firestore. Το Flow είναι ένα ισχυρό εργαλείο για την ασύγχρονη διαχείριση δεδομένων, καθώς επιτρέπει τη συλλογή των δεδομένων σταδιακά και την αντίδραση σε αλλαγές σε πραγματικό χρόνο. Για παράδειγμα, όταν γίνεται μια αλλαγή σε ένα Document στο Firestore, η εφαρμογή λαμβάνει άμεσα την ενημέρωση και το UI ανανεώνεται αυτόματα, χάρη στη χρήση του **StateFlow**.
Χρησιμοποιώντας το callbackFlow, μπορώ να μετατρέψω ένα **Firestore Query** σε Kotlin Flow, επιτρέποντας την παρακολούθηση αλλαγών σε πραγματικό χρόνο. Το Flow είναι ιδανικό για την παρακολούθηση δεδομένων που ενημερώνονται συχνά, καθώς η χρήση του εξασφαλίζει ότι το UI παραμένει συγχρονισμένο με τα πιο πρόσφατα δεδομένα χωρίς την ανάγκη χειροκίνητης ανανέωσης.
Αυτή η προσέγγιση βοηθά στη βελτίωση της εμπειρίας χρήστη, καθώς οι αλλαγές στα δεδομένα εμφανίζονται άμεσα στην οθόνη, δημιουργώντας μια πιο δυναμική και διαδραστική εφαρμογή. Επιπλέον, η διαχείριση σφαλμάτων γίνεται πιο αποτελεσματικά, καθώς το Flow παρέχει έναν ενσωματωμένο μηχανισμό για τον χειρισμό τους.

5. QR Code Scanner με χρήση της βιβλιοθήκης ZXing

Για τη λειτουργία σκαναρίσματος των QR Codes στην εφαρμογή, χρησιμοποίησα τη βιβλιοθήκη ZXing. Αυτή η βιβλιοθήκη είναι ευρέως γνωστή για την αξιοπιστία της και προσφέρει έναν εύκολο τρόπο για την ενσωμάτωση λειτουργιών σκαναρίσματος barcodes και QR codes σε εφαρμογές Android. Η υλοποίηση έγινε με τη χρήση του Jetpack Compose, δημιουργώντας ένα Composable στοιχείο για το UI και συνδέοντας τη λειτουργία της κάμερας με την ενέργεια σκαναρίσματος.
Όταν ο χρήστης πατάει στο QR εικονίδιο, ανοίγει η κάμερα της συσκευής μέσω της μεθόδου showCamera(), και ξεκινά το σκανάρισμα ενός QR κωδικού.
Χρησιμοποιώντας την μεταβλητή barCodeLauncher, διαχειρίζομαι το αποτέλεσμα του

σκαναρίσματος, και αν ο κωδικός είναι έγκυρος, τον αναζητώ στη βάση δεδομένων (Firestore) για να βρω τα αντίστοιχα στοιχεία του φοιτητικού πάσου.
Αυτή η λειτουργία προσφέρει ευκολία στους χρήστες, επιτρέποντας τη γρήγορη ανάκτηση δεδομένων μέσω του σκαναρίσματος του φοιτητικού πάσου, χωρίς να χρειάζεται χειροκίνητη εισαγωγή στοιχείων. Επιπλέον, η βιβλιοθήκη ZXing παρέχει δυνατότητες διαχείρισης σφαλμάτων, όπως αποτυχίες σκαναρίσματος ή έλλειψη άδειας για την κάμερα, καθιστώντας τη λύση αυτή ιδιαίτερα ολοκληρωμένη.

6 Συμπεράσματα και μελλόντικες επεκτάσεις

Η πτυχιακή αυτή εργασία είχε ως σκοπό την ανάπτυξη μιας εφαρμογής για κινητές συσκευές όπου θα δίνει τη δυνατότητα στους φοιτητές να έχουν την ακαδημαϊκή τους ταυτότητα μαζί τους οποιαδήποτε στιγμή χωρίς να είναι απαραίτητο το πλαστικό αντίγραφο της. Η εφαρμογή που ανέπτυξα παρέχει μια λειτουργική λύση, έτσι ώστε ο φοιτητής να έχει πρόσβαση κάθε στιγμή στα στοιχεία της ακαδημαϊκής του ταυτότητας, μειώνοντας έτσι της ανάγκη για την πλαστική κάρτα. Η εφαρμογή δίνει επιπλέον μια λύση για το θέμα των χαμένων ή και αργοπορημένων ταυτοτήτων που πολλοί φοιτητές δεν έχουν πρόσβαση στις ταυτότητες τους , και με την εφαρμογή αυτή θα μπορούν να έχουν ελεύθερη πρόσβαση στις υπηρεσίες του πανεπιστημίου χωρίς την ανάγκη της πλαστικής κάρτας.

Για την ανάπτυξη της εφαρμογής ήταν απαραίτητη η χρήση της γλώσσας προγραμματισμού Kotlin, του Framework Jetpack Compose καθώς και η χρήση της Coroutines Βιβλιοθήκης Flow της Kotlin. Έχοντας μηδενική εμπειρία με όλα τα παραπάνω έπρεπε να μελετήσω και να εμβαθύνω όλες αυτές τις νέες τεχνολογίες καθώς ήταν απαραίτητες για την ανάπτυξη της εφαρμογής αλλά και για την ακαδημαϊκή μου εκπαίδευση. Συγκεκριμένα, εφόσον τα παραπάνω αποτελούν τις νεότερες τεχνολογίες στο χώρο της ανάπτυξης εφαρμογών Android ήταν προς το καλύτερο μου συμφέρον να μάθω να τις χειρίζομαι ορθά και με σωστές πρακτικές κώδικα.

Συνεχίζοντας, η πτυχιακή αυτή κατάφερε μέσω της εφαρμογής να καλύψει πλήρως τους στόχους που τέθηκαν. Συγκεκριμένα κατάφερε να ικανοποιήσει πλήρως τις ανάγκες των φοιτητών για μια ψηφιακή ακαδημαϊκή λύση που θα συνεισφέρει στην άμεση και εύκολη πρόσβαση στη ακαδημαϊκή τους ταυτότητα. Η χρήση των τελευταίων τεχνολογιών στην ανάπτυξη εφαρμογών Android συνέβαλε στην ανάπτυξη μιας λειτουργικής, μοντέρνας και αξιόπιστης εφαρμογής.

Κατά την ανάπτυξη της εφαρμογής αντιμετωπίστηκαν ορισμένες δυσκολίες όπως η εκμάθηση των καινούργιων τεχνολογιών καθώς και οι αντιμετώπιση των bugs που προέκυψαν στη διάρκεια της ανάπτυξης. Η διαδικασία της μάθησης των νέων τεχνολογιών πήρε αρκετές ημέρες/εβδομάδες με πολλές προσπάθειες και πολλές διορθώσεις λαθών, όμως το τελικό αποτέλεσμα δείχνει ότι η επένδυση σε αυτές απέδωσε και το τελικό προϊόν είναι μια ποιοτική και άρτια εφαρμογή.

Μελλοντικά θα μπορούσαν να υπάρξουν ορισμένες προσθήκες στην εφαρμογή ώστε να γίνει ακόμα πιο προσιτή και δυναμική, συγκεκριμένα μελλοντικά θα μπορούσε να υπάρξει και ειδικό τμήμα εντός της εφαρμογής ώστε ο φοιτητής να δηλώσει τυχόν απώλεια της ταυτότητας ή φθορά και να μπορεί να κάνει απευθείας αίτηση για νέα ταυτότητα, έχοντας την επιλογή να διαλέξει εάν επιθυμεί μόνο ψηφιακό ή και φυσικό αντίτυπο. Εφόσον τα στοιχεία τη ταυτότητα ενημερώνονται δυναμικά, με την έκδοση της νέα ταυτότητας θα εμφανιστούν και στην εφαρμογή όλα τα νέα στοιχεία.

7 Βιβλιογραφικές Πηγές

Βιβλιογραφία

[1] Wikipedia, Android EL https://el.wikipedia.org/wiki/Android#

[2] Wikipedia, Android (operating system) EN
https://en.wikipedia.org/wiki/Android_%28operating_system%29

[3] Wikipedia, Android Studio EL https://el.wikipedia.org/wiki/Android_Studio

[4] Wikipedia, Android Studio EN https://en.wikipedia.org/wiki/Android_Studio

[5] Developer android, Android Studio EN https://developer.android.com/studio/intro

[6] android fandom, Android Studio EN https://android.fandom.com/wiki/Android_Studio

[7] kotlinlang.org, Kotlin https://kotlinlang.org/docs/home.html

[8] Wikipedia, Kotlin (Γλώσσα προγραμματισμού)
https://el.wikipedia.org/wiki/Kotlin_(%CE%B3%CE%BB%CF%8E%CF%83%CF%83%CE%B1_%CF%80%CF%81%CE%BF%CE%B3%CF%81%CE%B1%CE%BC%CE%BC%CE%B1%CF%84%CE%B9%CF%83%CE%BC%CE%BF%CF%8D)

[9] kotlinlang.org, Kotlin for android https://kotlinlang.org/docs/android-overview.html

[10] kotlinlang.org, Asynchronous Flow https://kotlinlang.org/docs/flow.html

[11] developer. android , Kotlin Flows on Android https://developer.android.com/kotlin/flow

[12] medium.com , Understanding Kotlin Flow in Android: A Comprehensive Guide
https://medium.com/@jaisavi25/understanding-kotlin-flow-in-android-a-comprehensive-guide-7531f5f8fcf0

[13] medium.com , Kotlin Flow Fundamentals: A Step-by-Step Guide
https://medium.com/@android-world/kotlin-flow-fundamentals-a-step-by-step-guide-fbe9f4068134

[14] baeldung.com, Introduction to the Kotlin Flow Class
https://www.baeldung.com/kotlin/flow-intro

[15] developer.android , StateFlow and SharedFlow
https://developer.android.com/kotlin/flow/stateflow-and-sharedflow

[16] decode.agency , Kotlin flows in Jetpack Compose: the ultimate guide
https://decode.agency/article/kotlin-flows-guide/

[17] Wikipedia, Jetpack Compose https://en.wikipedia.org/wiki/Jetpack_Compose

[18] developer. android , Compose https://developer.android.com/compose

[19] jetpackcompose.net , Jetpack Compose Cookbook
https://www.jetpackcompose.net/jetpack-compose-cookbook

[20] jetpackcompose.net, Jetpack Compose Basics
https://www.jetpackcompose.net/home/categories/basics

[21] YouTube , Philipp Lackner The Jetpack Compose Beginner Crash Course for 2023 💻 (Android Studio Tutorial) https://www.youtube.com/watch?v=6_wK_Ud8--0

[21] tutorialwing.com , Implement Android QR Code Scanner Using ZXing Library in Kotlin
https://tutorialwing.com/implement-android-qr-code-scanner-using-zxing-library-in-kotlin/

[22] YouTube , Step-By-Step Guide: Creating a QR Code Reader App Using Kotlin and ZXing
https://www.youtube.com/watch?v=5SHpuDgdnMY&t

[23] YouTube, Philipp Lackner Creating Your First Jetpack Compose App - Android Jetpack Compose - Part 1 https://www.youtube.com/watch?v=cDabx3SjuOY

[24] YouTube, Philipp Lackner Rows, Columns & Basic Sizing - Android Jetpack Compose - Part 2
https://www.youtube.com/watch?v=rHKeRWK3zL4

[25] YouTube, Philipp Lackner Modifiers - Android Jetpack Compose - Part 3
https://www.youtube.com/watch?v=XCuC_p3E0qo

[26] YouTube, Philipp Lackner Preferences Datastore in 10min (SharedPreferences deprecated)
https://www.youtube.com/watch?v=McnVx7l5awk

[27] firebase.google.com, Cloud Firestore https://firebase.google.com/docs/firestore/

[28] Wikipedia, Firebase https://en.wikipedia.org/wiki/Firebase

[29] firebase.google.com, Firebase Authentication https://firebase.google.com/docs/auth

[30] dummies.com, Use Case Description https://www.dummies.com/article/business-careers-money/business/general-business/how-to-create-use-case-description-for-your-business-analysis-report-162468/

[31] online.visual-paradigm.com, Online Swimlane Diagram Tool https://online.visual-paradigm.com/diagrams/features/flowchart-tool/swimlane-diagram-tool/