KoTwin: PlaceSpotter

Μαρδάς Αλέξιος Ερρίκος – 8210088

Νάκου Αικατερίνη Αναστασία - 8210102

Μάθημα: Ανάπτυξη & Σχεδίαση Κινητών Εφαρμογών

Διδάσκων: Φραγκούλης Μάριος

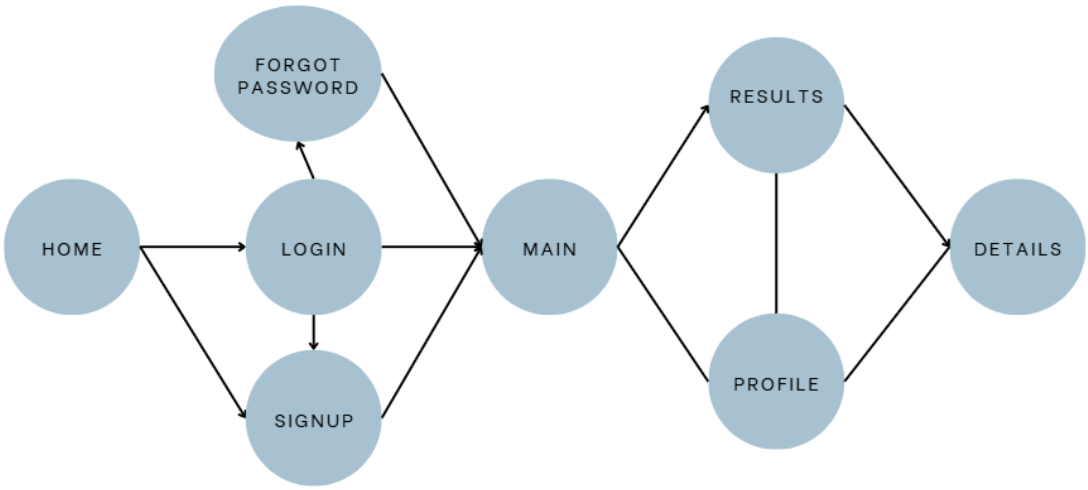
Αθήνα, Ιανουάριος 2025

Περιγραφή:

Η *εφαρμογή* αποτελεί ένα εργαλείο για τους ταξιδιώτες οι οποίοι ενδιαφέρονται να εξερευνήσουν σημεία ενδιαφέροντος στον προορισμό της επιλογής τους. Τέτοια σημεία ενδιαφέροντος θα μπορούσαν να είναι αξιοθέατα, μουσεία, εστιατόρια, μπαρ και καφετέριες.

Για τη δημιουργία αυτού του εργαλείου χρησιμοποιήθηκε το Google Places API. Για την εξατομίκευση της εφαρμογής στους χρήστες μας δημιουργήθηκε μια βάση που αποθηκεύει τα credentials του χρήστη και τα αγαπημένα του μέρη με τη βοήθεια του *Room*.

Ένα απλό overview του τρόπους με τον οποίο ο χρήστης αλληλοεπιδρά με τις οθόνες τις εφαρμογής μας είναι το εξής:



Προδιαγραφές:

1. Η εφαρμογή μας αποτελείται από 8 λειτουργικές οθόνες που αλληλοεπιδρούν με τον χρήστη.
2. Χρησιμοποιούνται όλα τα κύρια γραφικά στοιχεία (λίστες, κουμπιά, κείμενο, εικόνες, βίντεο) και η κατάλληλη διάταξη αυτών εκτός των βίντεο. Για παράδειγμα, χρησιμοποιούνται 2 κουμπιά το ένα κάτω από το άλλο στο *HomeScreen*, ώστε να μπορέσει ο χρήστης είτε να συνδεθεί είτε να δημιουργήσει έναν νέο λογαριασμό. Στα *ProfileScreen* εμφανίζονται οι εικόνες για κάθε Favorite Place. Η χρήση λίστας γίνεται, για παράδειγμα, στο *ResultsScreen*, στο οποίο για κάθε *query* του χρήστη εμφανίζεται μια λίστα από Places. Κείμενο υπάρχει στο *DetailsScreen*, στο οποίο φαίνονται για το επιλεγμένο Place περαιτέρω πληροφορίες. Τέλος, δεν υπάρχει χρήση των βίντεο, διότι θεωρήσαμε ότι δεν ταιριάζει με τη φιλοσοφία της εφαρμογής μας.
3. Σε κάθε ViewModel μας χρησιμοποιούνται διάφορα state δεδομένα εκτός του *HomeViewModel*. Για παράδειγμα, στη *LoginScreen* χρησιμοποιούνται state δεδομένα για username, password, passwordVisible και error message.
4. Η εφαρμογή μας χρησιμοποιεί αντικειμενοστρεφή προγραμματισμό σε διάφορα σημεία. Για παράδειγμα, από τα Screen αρχεία μας καλούνται Object των αντίστοιχων ViewModel αρχείων μας για τη διαχείριση των εκάστοτε intent του χρήστη. Ένα άλλο δείγμα αντικειμενοστρέφειας είναι η ύπαρξη των data κλάσεων μας, η οποίες προσδιορίζουν ένα αντικείμενο User και Favorite, καθώς και το object Place που ανάλογα με τα δεδομένα που δέχεται από την κλήση του API ορίζει τα γνωρίσματα των Place αντικειμένων.
5. Η εφαρμογή χρησιμοποιεί τη βάση δεδομένων Room, ένα wrapper της SQLite, για τη διαχείριση των δεδομένων της εφαρμογής. Η βάση αυτή περιέχει δύο πίνακες τους User και τα Favorites
6. Η εφαρμογή λαμβάνει δεδομένα από το διαδίκτυο μέσω της σύνδεσης με το Google Places API.
7. Η εφαρμογή αξιοποιεί το συγχρονισμό (concurrency) με τη χρήση coroutines για την εύρυθμη λειτουργία και απόδοση της εφαρμογής. Αυτό μπορεί να φανεί σε όλες τις μεθόδους που συνδέονται είτε με τη βάση είτε με το διαδίκτυο.
8. Η πλοήγηση της εφαρμογής από οθόνη σε οθόνη υλοποιείται μέσω του αρχείου *GuideNavGraph,* το οποίο είναι υπεύθυνο για τη σωστή σύνδεση μεταξύ των οθονών. Η πλοήγηση με κατάλληλη ένδειξη της οθόνης στην οποία βρίσκεται ο χρήστης και δυνατότητα επιστροφής στο προηγούμενο βήμα γίνεται μέσω του navbar μας, το οποίο έχει ένα back button, την πρόσβαση στο Home και τη πρόσβαση στο Profile.

Βασικές Προκλήσεις/Καινοτομίες:

Η πρώτη πρόκληση της εφαρμογής μας ήταν η σωστή διαχείριση των ViewModel μας. Συγκεκριμένα, καθώς τα ViewModel χρειάζεται να επικοινωνούν με τη βάση για τη σωστή διαχείριση των Screens, κάθε ViewModel χρειάζεται ένα δικό του Factory, που να διαχειρίζεται σωστά τη δημιουργία του κάθε φορά που η εφαρμογή μας χτίζεται.

```
object AppViewModelProvider {
    val Factory = viewModelFactory { this: InitializerViewModelFactoryBui...

        // Initializer for ForgotPasswordViewModel
        initializer { this: CreationExtras
            ForgotPasswordViewModel(guideApplication().container.usersRepository)
        }

        // Initializer for LoginViewModel
        initializer { this: CreationExtras
            LoginViewModel(guideApplication().container.usersRepository)
        }
        // Initializer for SignUpViewModel
        initializer { this: CreationExtras
            SignUpViewModel(guideApplication().container.usersRepository)
        }

        // Initializer for MainViewModel
        initializer { this: CreationExtras
           MainViewModel()
        }

        // Initializer for HomeViewModel
        initializer { this: CreationExtras
            HomeViewModel()
        }

    }
```

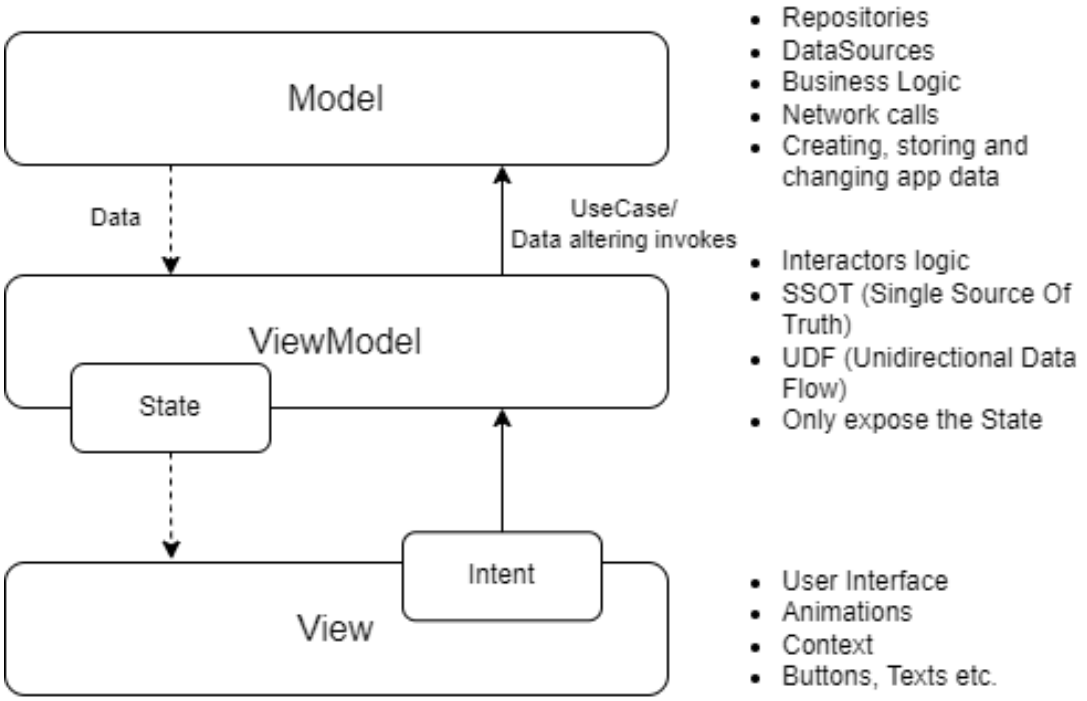
Η δεύτερη πρόκληση που αντιμετωπίσαμε είναι η διαχείριση του χρήστη κατά τη χρήση της εφαρμογής του. Συγκεκριμένα, καθώς η εφαρμογή προσαρμόζεται στις προτιμήσεις του χρήστη, χρειάζεται πάντα να αποθηκεύουμε και να αναγνωρίζουμε ποιος είναι ο χρήστης της εφαρμογής. Αυτό το πρόβλημα το λύσαμε μέσω του Navigation αρχείου μας. Συγκεκριμένα, κατά την είσοδο του χρήστη στην εφαρμογή, μέσω του LoginScreen, SignUpScreen ή ForgotPasswordScreen, αποθηκεύουμε το κλειδί του χρήστη το οποίο έπειτα το περνάμε σαν παράμετρο σε όλες τις σελίδες μας μέσω του navigation μας.

```
// Profile Screen
composable(route = "${ProfileDestination.route}/{userId}",
    arguments = listOf(navArgument( name: "userId") { type = NavType.IntType })
) { this: AnimatedContentScope  backStackEntry ->
    val userId = backStackEntry.arguments?.getInt( key: "userId") ?: 0
    val profileViewModel: ProfileViewModel = viewModel(
        factory = ProfileViewModelFactory(userId, userRepository, favoritesRepository)
    )
    ProfileScreen(
        navigateBack = { navController.navigateUp() },
        navigateToMain = {navController.navigate( route: "${MainDestination.route}/$userId") },
        viewModel = profileViewModel
    )
}
```

Η τελευταία πρόκληση που αντιμετωπίσαμε σχετίζεται με το API Call μας. Συγκεκριμένα, ενώ καταφέραμε επιτυχώς να δεχόμαστε όλες τις παραμέτρους που επιστρέφει το API Call, όπως το name και το rating των Place, η παράμετρος Reviews μας επιστρέφει σε κάθε κλήση Null. Αυτό πιθανότατα οφείλεται σε κάποιο πρόβλημα του API που χρησιμοποιήθκε (Text Search).

Αρχιτεκτονικός Σχεδιασμός:

Η εφαρμογή ακολουθεί το μοτίβο αρχιτεκτονικής MVVM (Model-View-ViewModel) για τον διαχωρισμό των ευθυνών της εφαρμογής. Η εν λόγω αρχιτεκτονική απεικονίζεται στο παρακάτω διάγραμμα:



- **Model**: Διαχειρίζεται τα δεδομένα της εφαρμογής, περιλαμβάνοντας τη βάση δεδομένων και τις κλήσεις στο API για την ανάκτηση πληροφοριών

(όπως λεπτομέρειες τόπων ή κριτικές). Στο Model περιλαμβάνονται τα network και τα data classes.

- **ViewModel**: Λειτουργεί ως ενδιάμεσος μεταξύ του UI και των δεδομένων, χειρίζεται τη λογική της εφαρμογής, αποθηκεύει τα δεδομένα του UI (state) και διασφαλίζει ότι η εφαρμογή παραμένει ευέλικτη και εύκολα συντηρήσιμη. Μέσω του ViewModel, τα intents του χρήστη, που εκφράζονται από την αλληλεπίδραση του με το View Layer πυροδοτούν τις δέουσες αλλαγές στο Model Layer. Το state που παρατηρείται από το ViewModel, προσαρμόζεται κατάλληλα στις αλλαγές που λαμβάνουν χώρα στο Model Layer και προκαλεί το recomposition του View Layer, όπου πρέπει, ούτως ώστε να απεικονίζονται κατάλληλα στον χρήστη αυτές οι αλλαγές. Συνεπώς, χρησιμοποιείται το Unidirectional Data Flow (UDF) που εξασφαλίζει υψηλή αξιοπιστία στη λειτουργία της εφαρμογής.
- **View**: Το UI της εφαρμογής, που περιλαμβάνει τις οθόνες (screens) και τα UI components που υλοποιούνται με Jetpack Compose. Κάθε οθόνη αντιπροσωπεύει μια λειτουργικότητα της εφαρμογής (π.χ., αναζήτηση, προβολή λεπτομερειών τόπου, διαχείριση προφίλ χρήστη).

Παρακάτω παρατίθενται εν συντομία τα αρχεία της εφαρμογής, η βασική τους λειτουργικότητα καθώς και το layer στο οποίο ανήκουν.

1. **Κλάσεις πακέτου com.example.guide.data:**
   Το σύνολο των κλάσεων αυτού του πακέτου εντάσσονται στο Model Layer και σχετίζονται με τη δημιουργία της βάσης δεδομένων και την αλληλεπίδραση με αυτήν. Συγκεκριμένα:
   - Favorite: Η κλάση που ορίζει την αναπαράσταση του πίνακα “favorites” στο Room Database.
   - FavoriteDao: Η διεπαφή που ορίζει τις μεθόδους που πρέπει να υλοποιηθούν για την αλληλεπίδραση με τον πίνακα favorites.
   - FavoritesRepository: Η διεπαφή που ορίζει τις μεθόδους που θα καλούν αυτές της FavoriteDao για πραγματική υλοποίηση των επιθυμητών λειτουργιών.
   - OfflineFavoritesRepository: Η κλάση που υλοποιεί τις μεθόδους της FavoritesRepository, αντιστοιχίζοντας τις με αυτές τις FavoriteDao.
   - User, UserDao, UserRepository και OfflineUsersRepository: Υλοποιούν την ίδια ακριβώς λειτουργικότητα με τις παραπάνω κλάσεις με τα παρεμφερή ονόματα, εστιάζοντας στον πίνακα “users” της βάσης δεδομένων και την αλληλεπίδραση με αυτόν.
   - GuideDatabase: Αρχικοποιεί το instance της βάσης δεδομένων.

- AppContainer: Αρχικοποιεί τα Repository objects που χρησιμοποιούνται για την επικοινωνία με τη βάση οπουδήποτε αυτό κρίνεται σκόπιμο στην εφαρμογή.
- FakeUsersRepository και FakeFavoritesRepository: Dummy classes που χρησιμοποιούνται απλά για την τροφοδότηση των preview οθονών με κάποια δεδομένα. Υπάρχουν καθαρά για σχεδιαστικούς σκοπούς και δεν συνεισφέρουν κάπως στην λειτουργικότητα.

2. **Κλάσεις πακέτου com.example.guide.network:**

   Το σύνολο των κλάσεων αυτού του πακέτου εντάσσονται, επίσης, στο Model layer και υλοποιούν την σύνδεση, μέσω της βιβλιοθήκης Retrofit, με το Google Places Text Search API. Στην κατηγορία αυτή, έμμεσα εντάσσεται και το object AppConfig μέσα στο οποίο ο χρήστης πρέπει να εισάγει το API key του.

   - GuideApiService: Παρέχει τη διεπαφή για τη σύνδεση με το API, μέσω της βιβλιοθήκης Retrofit.
   - PlacesResponse: Μοντελοποιεί το response του Google Places API σε Serializable Kotlin Objects.

3. **Κλάσεις πακέτου com.example.guide.ui.theme:**
   Το σύνολο των κλάσεων αυτού του πακέτου καθορίζει το theme της εφαρμογής και εντάσσονται στο View layer.

4. **Κλάσεις πακέτων com.example.guide.ui.screens και com.example.guide.ui.home:**
   Περιλαμβάνουν το σύνολο των κλάσεων που υλοποιούν τις οθόνες και τα ViewModel τους. Οι κλάσεις αυτών των πακέτων είναι μοιρασμένες, με άλλες να ανήκουν στο View layer (οι οθόνες) και οι υπόλοιπες αποτελούν το ViewModel layer της εφαρμογής.
   - HomeScreen και HomeViewModel: Υλοποιούν την πρώτη οθόνη που συναντάει ο χρήστης της εφαρμογής καθώς και τη λογική της.
   - LoginScreen και LoginViewModel: Υλοποιούν την οθόνη που ο χρήστης συναντάει για την είσοδό του στην εφαρμογή καθώς και την λογική της.
   - SignUpScreen και SignUpViewModel: Υλοποιούν την οθόνη που χρησιμοποιείται για την εγγραφή του χρήστη στην εφαρμογή και την λογική της.
   - ForgotPasswordScreen και ForgotPasswordViewModel: Υλοποιούν την οθόνη που ο χρήσης χρησιμοποιεί για να αλλάξει τον κωδικό του, καθώς και την λογική της.

- MainScreen και MainViewModel: Υλοποιούν την οθόνη που ο χρήστης χρησιμοποιεί για να αναζητήσει τοποθεσίες καθως και την λογική της.
- SearchResultsScreen και SearchResultsViewModel: Υλοποιούν την οθόνη που εμφανίζει τα αποτελέσματα στον χρήστη καθώς και την λογική της.
- DetailsScreen και DetailsViewModel: Υλοποιούν την οθόνη που εμφανίζει τις λεπτομέρειες κάθε μέρους, καθώς και την λογική της.
- ProfileScreen και ProfileScreenViewModel: Υλοποιούν την οθόνη που εμφανίζει το προφίλ κάθε χρήστη καθώς και την λογική της.
- SharedPlaceViewModel: Υποστηρίζει τη σύνδεση μεταξύ της SearchResultsScreen και του DetailsScreen.

5. **Κλάσεις πακέτου com.example.guide.navigation:**

   Δεν ανήκουν σε κάποιο συγκεκριμένο layer, αλλά λειτουργούν ως ο ενορχηστρωτής συντονίζοντας όλα τα υπόλοιπα:

   - NavigationDestination: Ορίζει τη διεπαφή που θα υλοποιεί κάθε προορισμός του GuideNavGraph.
   - GuideNavGraph: Χειρίζεται το σύνολο της πλοήγησης της εφαρμογής και της αλληλεπίδρασης μεταξύ των οθονών
6. **Λοιπές κλάσεις πακέτου com.example.guide.ui:**
   Περιλαμβάνουν τις κλάσεις που σχετίζονται με τη δημιουργία των ViewModel κάθε οθόνης:
   - AppViewModelProvider: Δημιουργεί το σύνολο των ViewModels της εφαρμογής που δεν χρειάζονται περαιτέρω παραμέτρους.
   - DetailsViewModelFactory: Αρχικοποιεί το ViewModel που χρησιμοποιείται από το DetailsScreen.
   - ProfileViewModelFactory: Αρχικοποιεί το ViewModel που χρησιμοποιείται από το ProfileScreen.
   - SearchResultsViewModelFactory: Αρχικοποιεί το ViewModel που χρησιμοποιείται από το SearchResultsScreen.
7. **Λοιπές κλάσεις του πακέτου com.example.guide:**

   Χρησιμοποιούνται για την έναρξη της λειτουργίας της εφαρμογής υλοποιώντας το MainActivity.

Οδηγίες χρήσης και εγκατάστασης:

Για την εγκατάσταση της εφαρμογής μας ο χρήστης θα πρέπει να έχει πρώτα εγκατεστημένο το Android Studio.

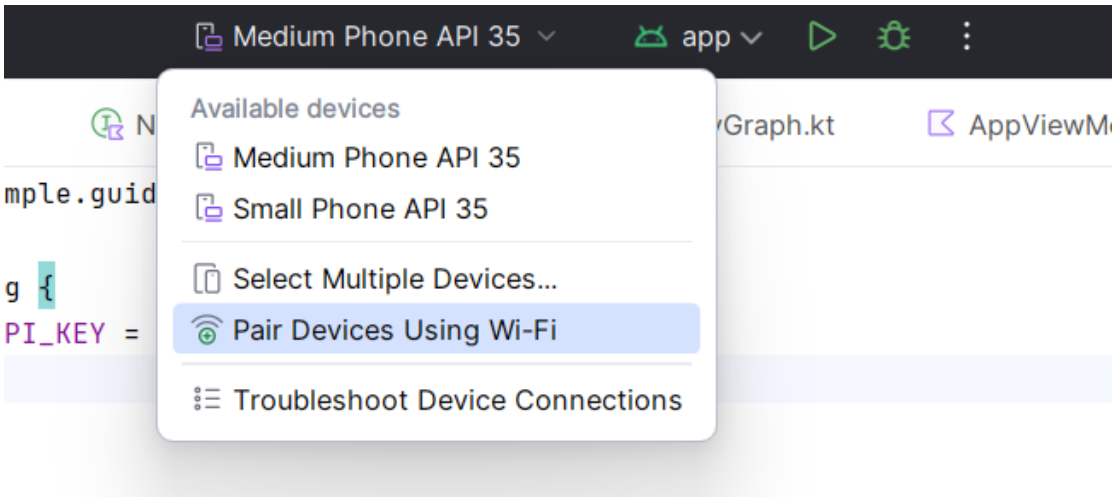
Έπειτα ο κώδικας της εφαρμογής είναι διαθέσιμος στο Github Repository: https://github.com/KaterinaNakou2003/guide .

Έχοντας κάνει clone το παραπάνω repository, ο χρήστης πρέπει να θέσει το δικό του Google Places API key στο αρχείο AppConfig του φακέλου *com.example.com/ui,* ώστε να μπορέσει να συνδεθεί με το API και να λάβει δεδομένα από αυτό.

```
package com.example.guide

object AppConfig {
    const val API_KEY = "Your API key"
}
```

Το τελευταίο βήμα στην εγκατάσταση της εφαρμογής είναι ο χρήστης να κάνει Build και run μέσω του Android Studio την εφαρμογή είτε στον ενσωματωμένο emulator, είτε στο κινητό του. Για να συμβεί το δευτέρο ο χρήστης πρέπει να κάνει *Pair Devices Using Wi-Fi*.



Εάν αυτή η επιλογή δεν είναι διαθέσιμη από αυτό το σημείο ο χρήστης μπορεί να το βρει ακολουθώντας το επόμενο μονοπάτι:

View -> Tool Windows -> Device Manager .

Περισσότερες πληροφορίες για τη συγκεκριμένη διαδικασία βρίσκονται στη επίσημη σελίδα των Android Developers: https://developer.android.com/studio/run/device .

Έτσι, ολοκληρώθηκε η διαδικασία εγκατάστασης της εφαρμογής.

Για τη χρήση της εφαρμογής ο χρήστης αρκεί να φτιάξει έναν λογαριασμό με τα credentials του.