Prepared by Spook

17 February, 2024

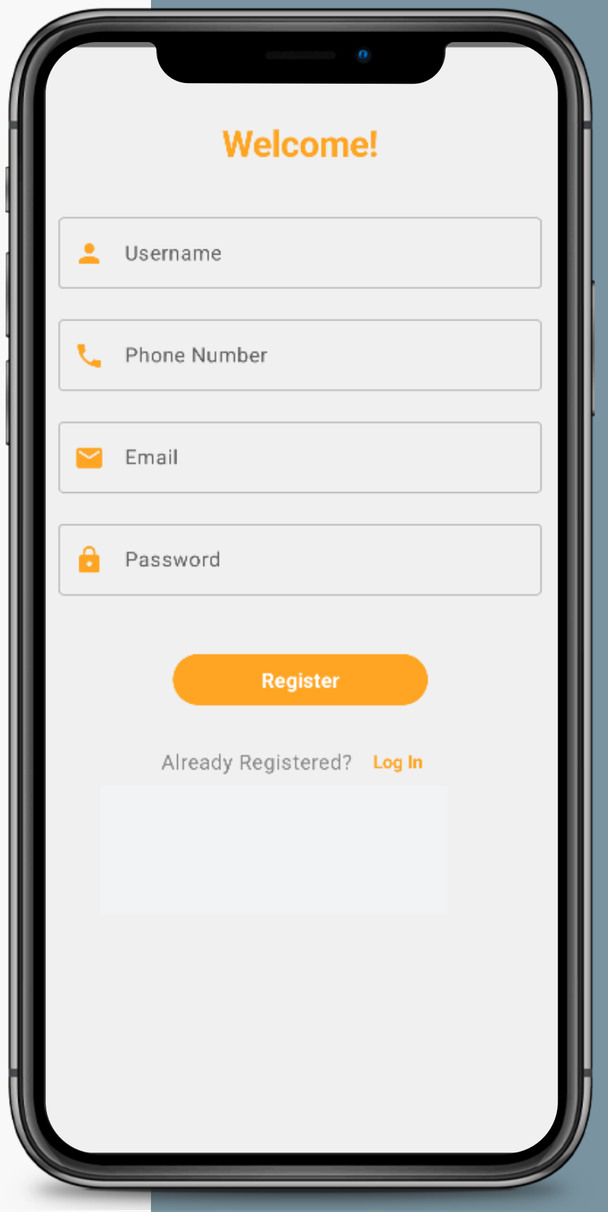
Gourgioti Zoi-8210028
Kounara Foteini-Andriani - 8210066
Nerantzakis Iasonas - 8210103
Rousi Katerina - 8210130

Our app

Our app is a reservation system for a variety of restaurants offered by Juicy. Users can explore available restaurants based on the selected day and the number of people. They can also view special offers, detailed store information, and their booking history through their profile. The app makes it easy to book a table at their favorite restaurants.

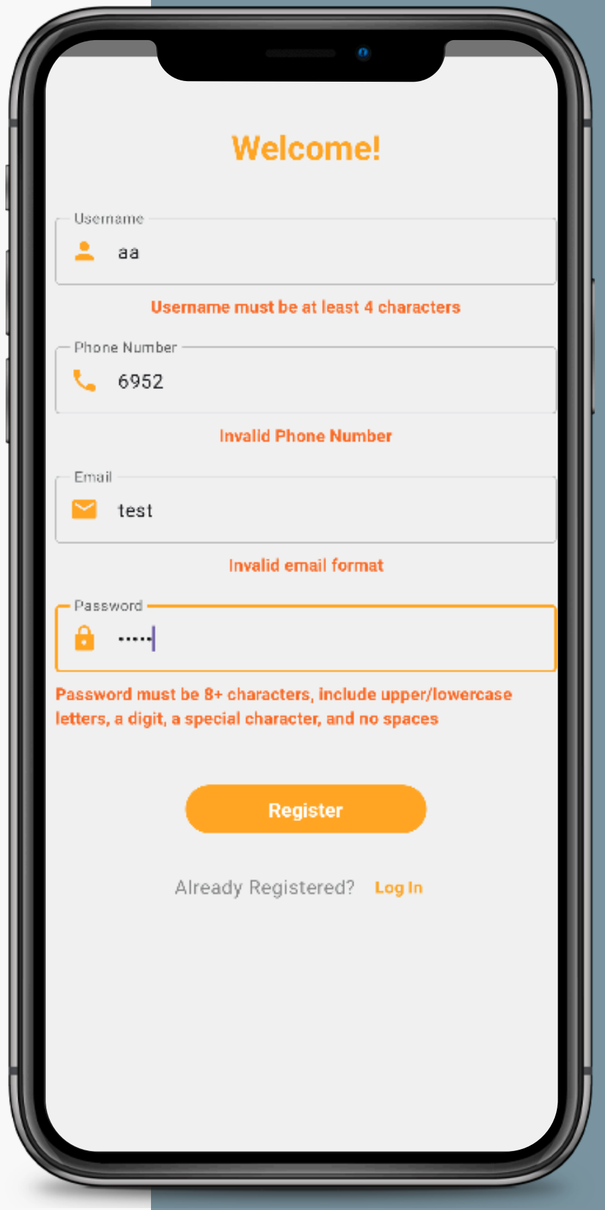
Register

- Implemented Field Requirements.
- User Feedback on Incorrect Field Data.
- Navigation to Login Page.



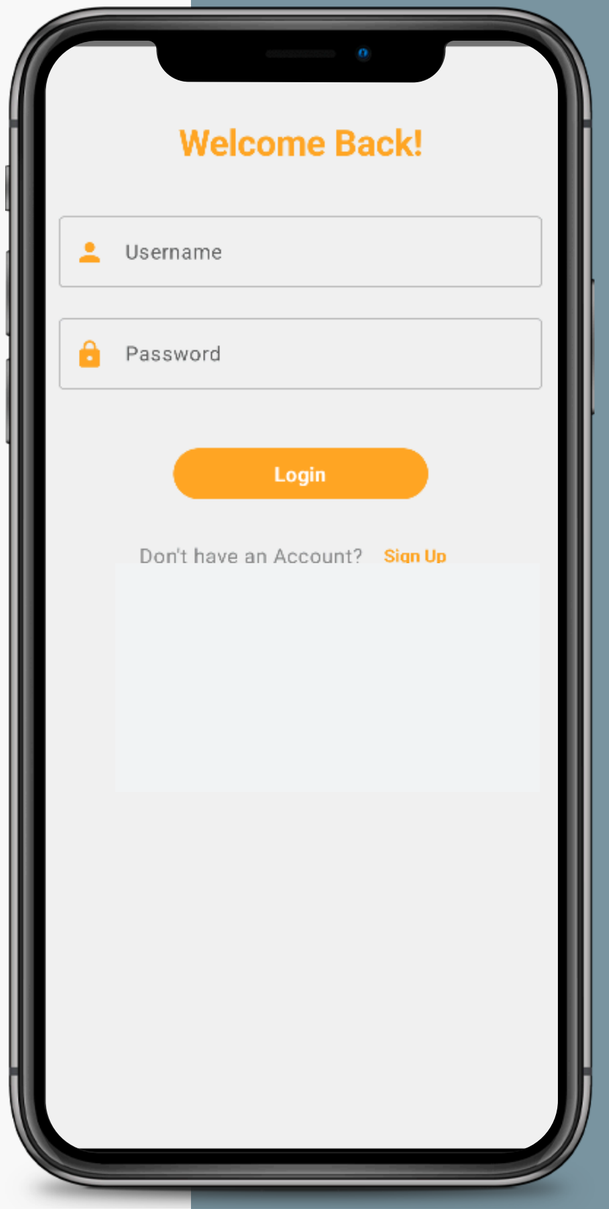
Register

- Implemented Field Requirements.
- User Feedback on Incorrect Field Data.
- Navigation to Login Page.



Login

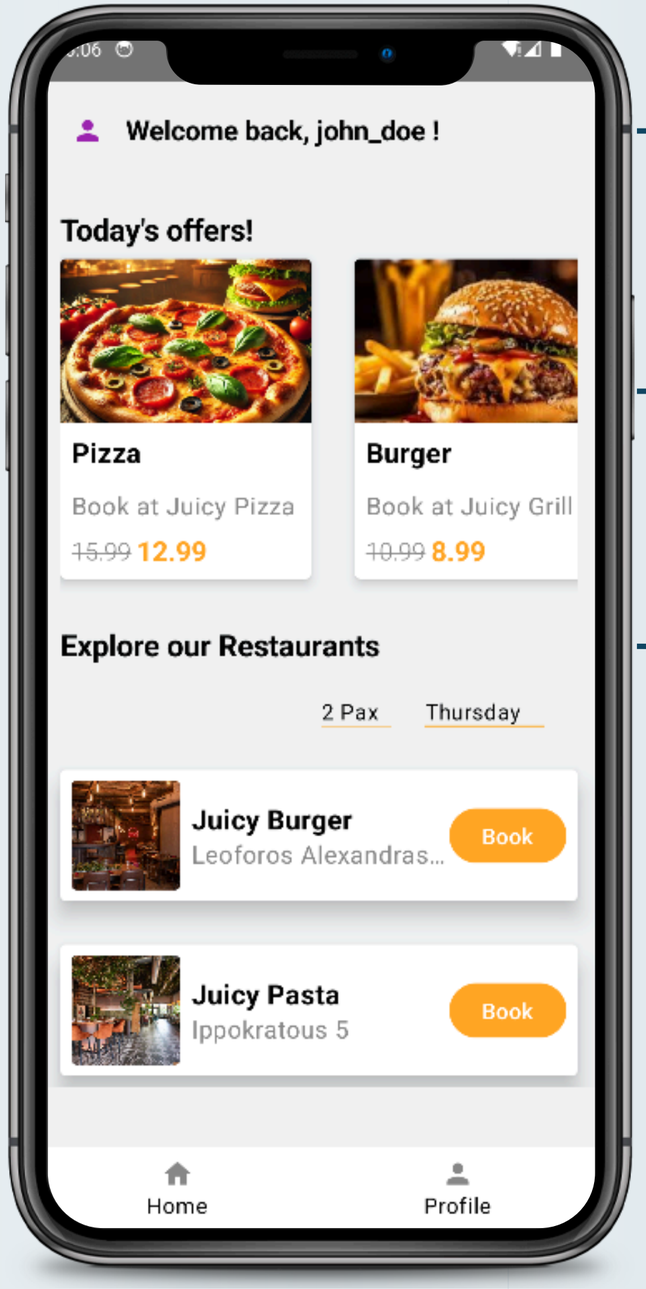
- Appears on Launch.
- User Authentication.
- User Feedback on Incorrect Credentials.
- User Feedback on Empty Fileds.
- Navigation to Register Page.



Homepage

- **Dynamic Filtering:** Availability matching by syncing data from tables stores and slots based on the selected date and pax. (MutableState, LaunchedEffect)

- **Performance Optimization:** Loading images and updating UI state in real time.



Header

Personalized welcome message.

Today's Offers

Dynamic discounts linking to specific restaurants for quick booking.

Restaurant List

Filterable list of available restaurants based on the number of people and selected date, with an instant booking option.

UI/UX Design

Box

- Fine control with align, RoundedCornerShape, CardElevation.
- Layering interactive overlays with modifier.

Cards

- Dynamic content layouts (Row, Column).
- Managing overflows (clip(), zIndex()).

Material 3

- Dynamic Colors.
- Elevation.

Jetpack Compose

- State Management (MutableState)
- Side Effects (controlled updates, LaunchedEffect, remember)

LazyColumn/LazyList

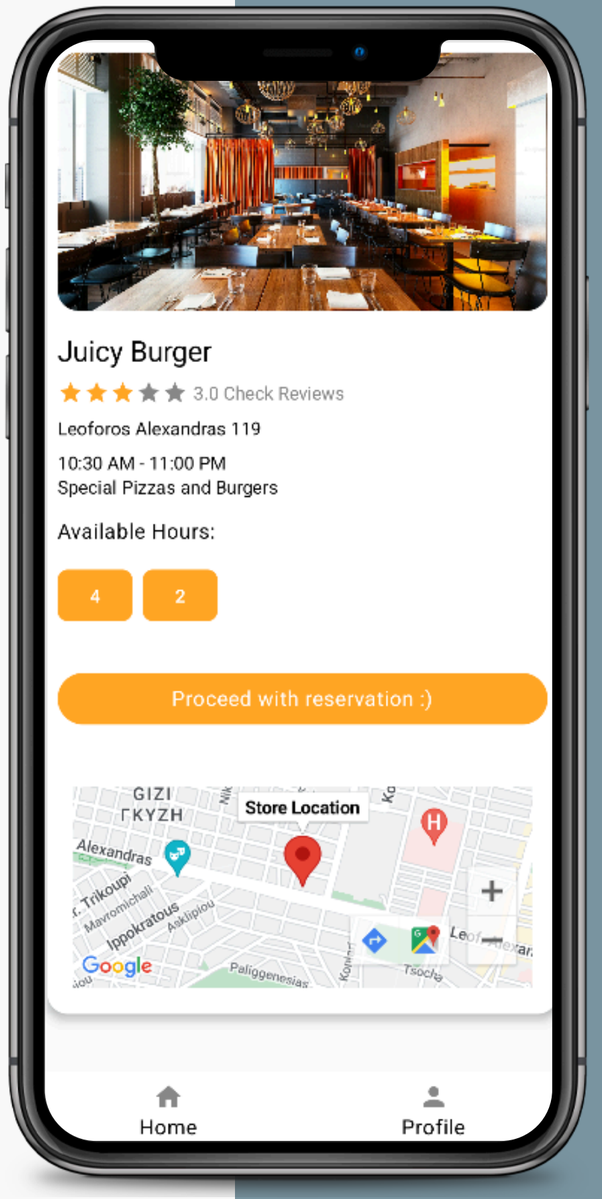
- Custom itemsIndexed() for tailored layouts.
- Handling keys for stable, reusable list items.
- Grid-style customization (LazyListScope, LayoutModifier).

Stores

- From homepage .
- Specific information for the store selected in the Homepage.
- Store table information: location,name,hours,description.
- Slots table information: available slots.
- Review table information: reviews for the store.

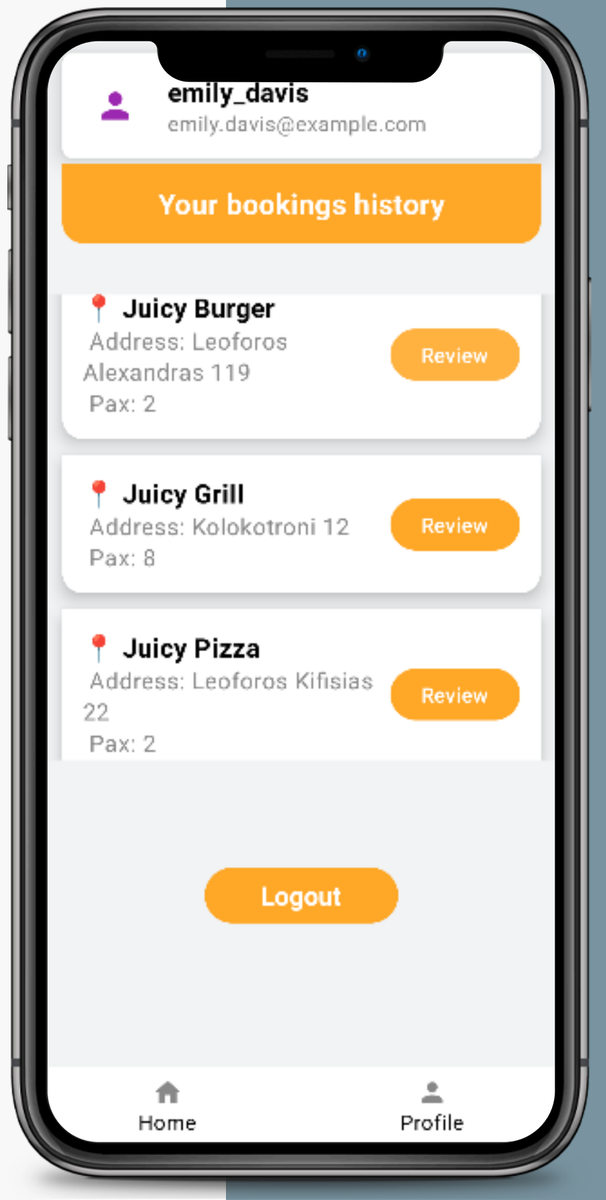
Google Maps API

- Gets the location of the store.
- Location from string ⟶ longitude,latitude (Geocoder).
- Passes (longitude,latitude) to Google API.



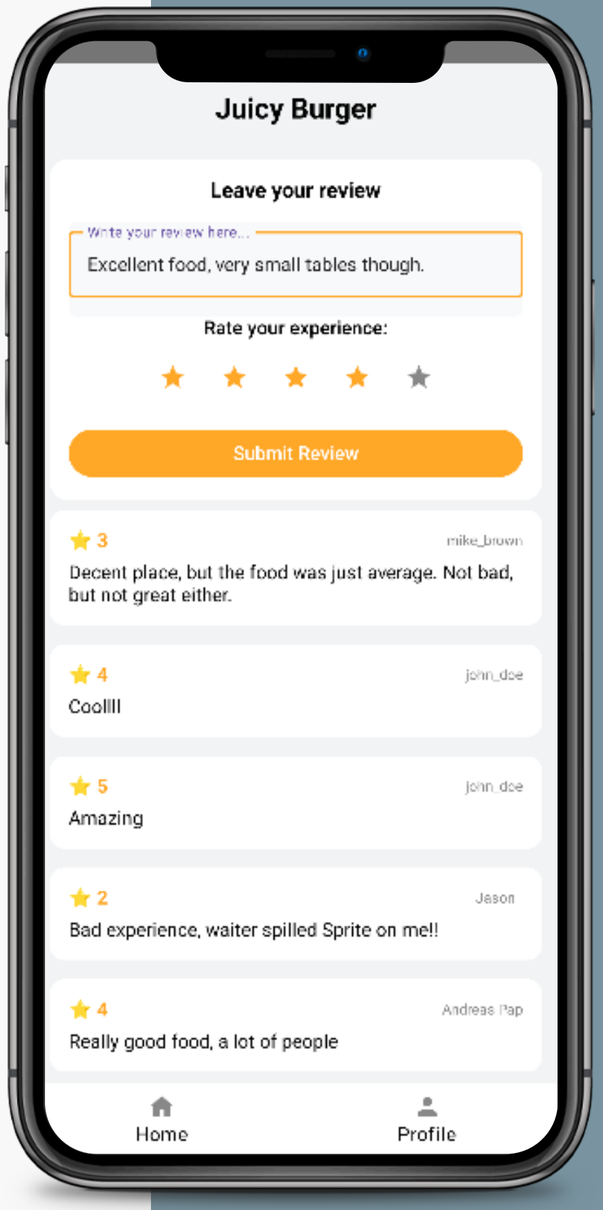
Profile

- From all the pages.
- Information about your bookings.
- User table information: username,email.
- Bookings table information: bookings made by user,number of people.
- Store table information: name,location.



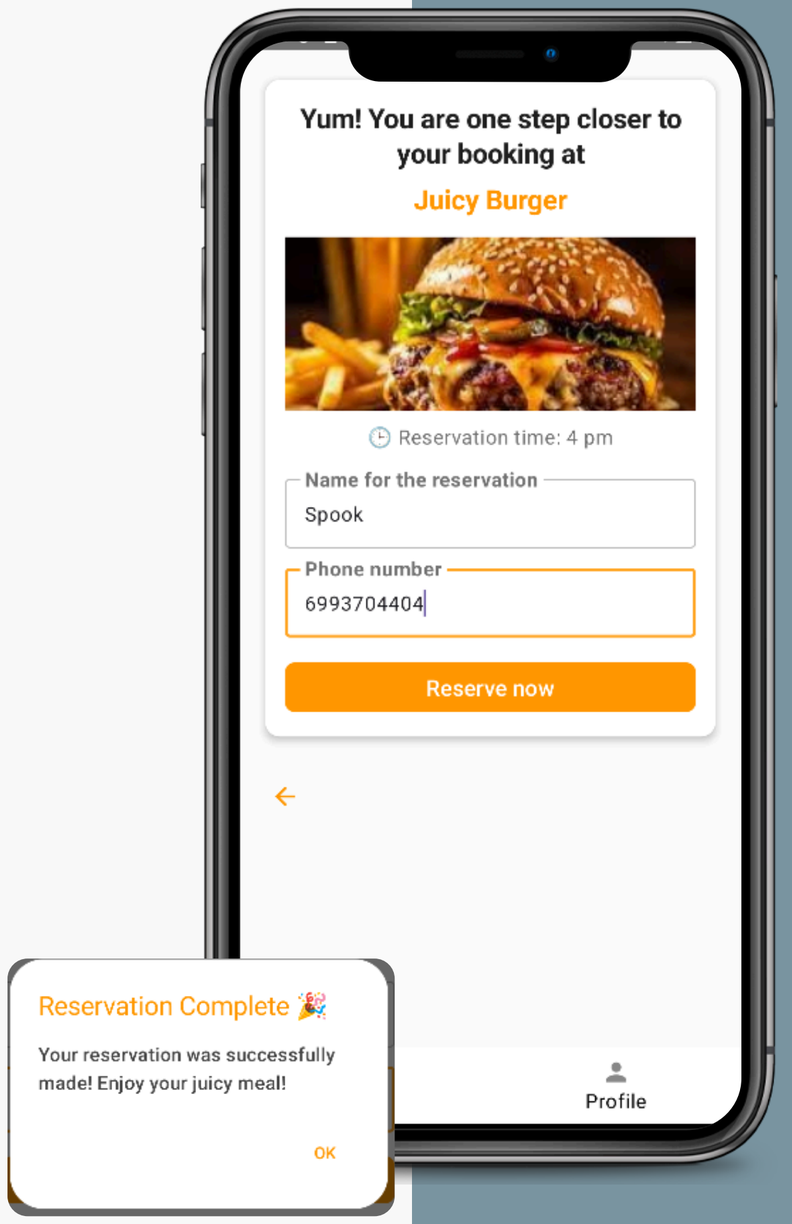
Reviews

- Navigation from Store and Profile Page.
- Users can write a comment.
- See comments and their author.
- Algorithm that takes the reviews(stars) and produces the store rating.



Booking

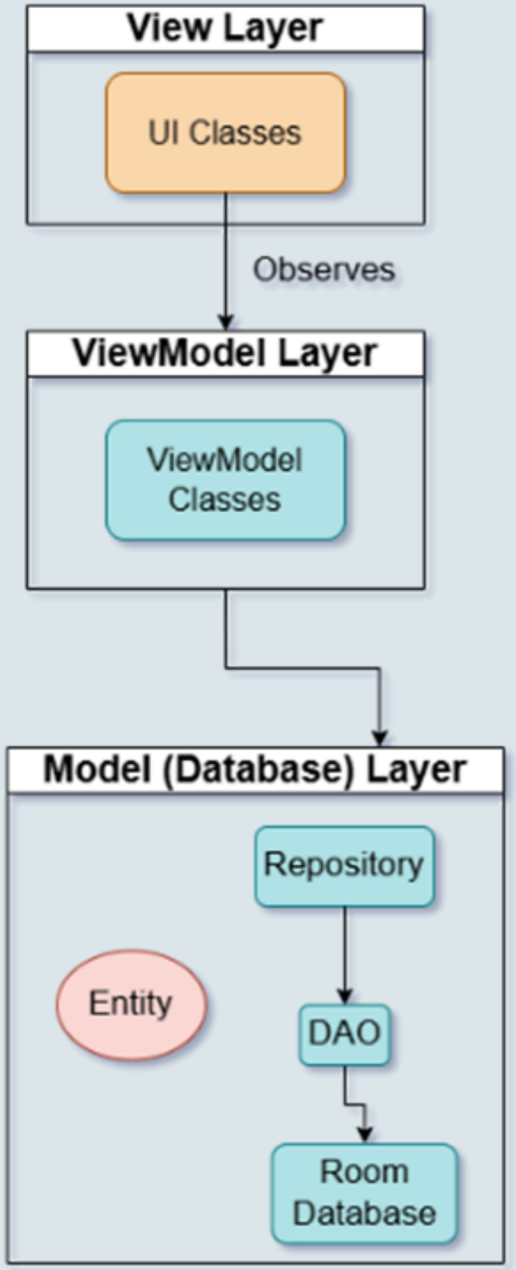
- Gets data from the Store Screen.
- Final step before booking is made.
- Verifies user details.
- Auto fill if user is logged in.
- Navigates to User Profile Screen.
- Ability to navigate back.



Architecture

MVVM (Model - View - ViewModel)

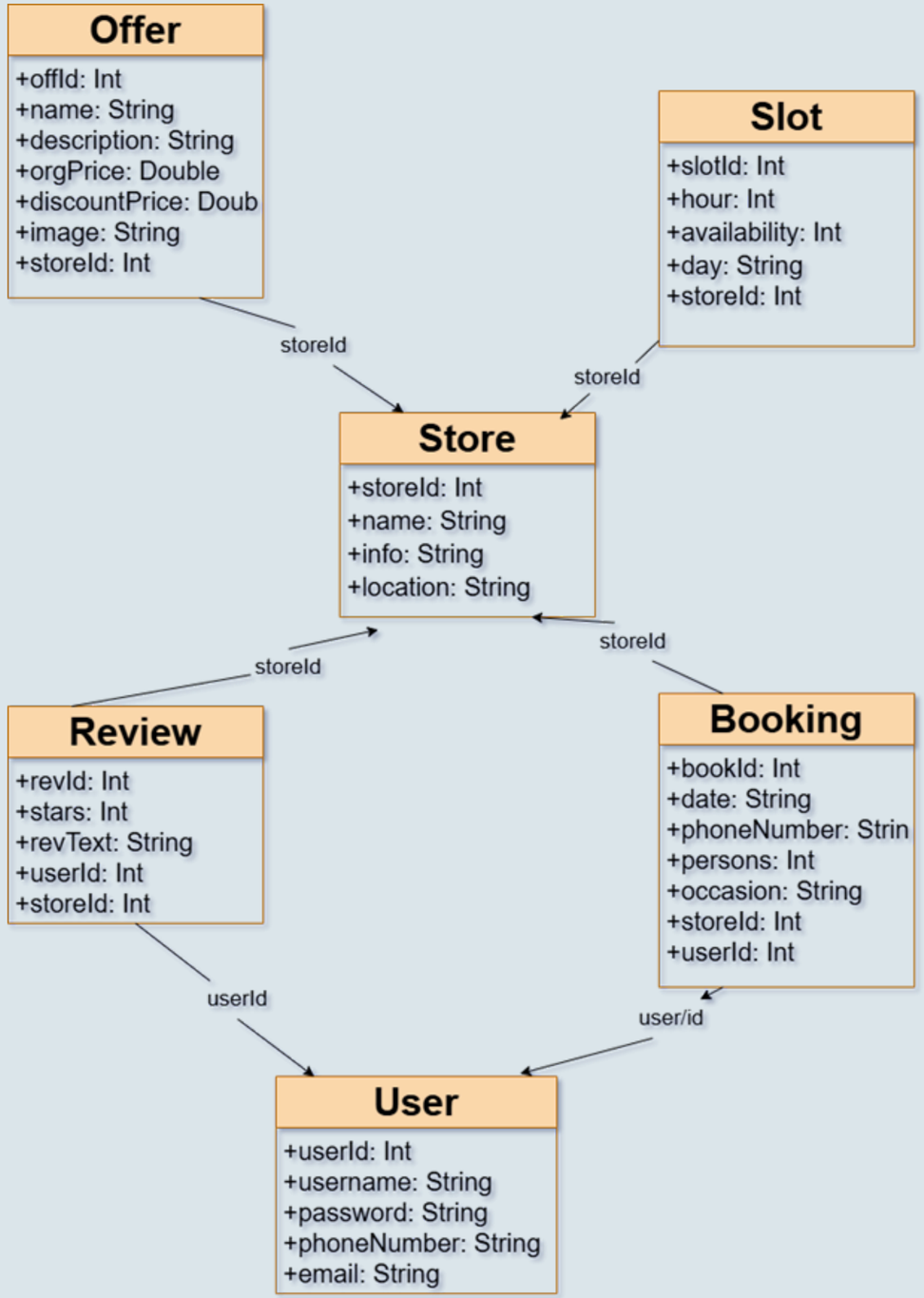
- Model: Data handling and business logic.
- ViewModel: Middle layer that implements DAO functions and holds UI data.
- View: UI classes (Activities, Composables) and display of VM data.



Room Database

Library for SQLite Abstraction

- DAO Classes: SQL Queries.
- Entity Classes: One per table.
- AppDatabase file (Initialize and prepopulate).



Example structure for the ‘Store’ Entity

StoreData.kt

Declares store_table as an Entity

```kotlin
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "store_table")
data class Store(
    @PrimaryKey(autoGenerate = true)
    val storeId: Int = 0,
    val name: String,
    val info: String,
    val location: String,
)
```

StoreDAO.kt

Implements SQL Queries

```kotlin
@Dao
interface StoreDao {
    @Insert
    suspend fun insert(store: Store)

    @Query("SELECT * FROM store_table")
    fun getAllStores(): Flow<List<Store>>

    @Query("DELETE FROM store_table")
    suspend fun deleteAllStores()

    @Query("SELECT name FROM store_table WHERE storeId = :storeId")
    suspend fun getStoreNameById(storeId: kotlin.String): String?

    @Query("SELECT location FROM store_table WHERE storeId = :storeId")
    suspend fun getLocationById(storeId: Int): String?
}
```

Example structure for the ‘Store’ Entity

StoreViewModel.kt

Functions and UI data

```
fun fetchStoreName(storeId: Int) {
    viewModelScope.launch {
        val name = storeDao.getStoreNameById(storeId.toString())
        _storeNames.update { current ->
            current.toMutableMap().apply {
                this[storeId] = name
            }
        }
    }
}
```

```
Text(
    text = viewModel.storeName,
    style = MaterialTheme.typography.titleLarge.copy(fontWeight = FontWeight.Bold),
    color = Color( color: 0xFFFF9800),
    textAlign = TextAlign.Center
)
```

Homepage.kt

UI and data collections

Jetpack Navigation

How to call?

```
navController.navigate(Screen.Stores.withArgs(it.name, selectedDay, selectedPer
```

Passing Arguments

```
composable(
    route = Screen.Stores.route + "/{name}/{selectedDay}/{selectedPersons}",
    arguments = listOf(

        navArgument( name: "name") {
            type = NavType.StringType
            nullable = true
        },

        navArgument( name: "selectedDay") {
            type = NavType.StringType
            nullable = true
        },

        navArgument( name: "selectedPersons") {
            type = NavType.StringType
            nullable = true
        }
    )
) { entry ->
    val name = entry.arguments?.getString( key: "name") ?: "Juicy Grill"
    val selectedDay = entry.arguments?.getString( key: "selectedDay") ?: "Monday"
    val persons = entry.arguments?.getString( key: "selectedPersons") ?: "2"
    StoreNavigation(userId, userViewModel, storeViewModel, bookingViewModel, reviewViewModel,
}
```

Navigation Bar

```
BottomNavBar(
    onHomeClick = {
        navController.navigate(Screen.HomePage.route) {
        popUpTo(Screen.HomePage.route) { inclusive = true } // Avoids stacking multiple Homepages
    } },
    onProfileClick = { navController.navigate(Screen.ProfileScreen.withArgs(userId.toString())) }
)
```

Thank you!