

Table Hub Mobile App for Restaurant Table Management

Julian Bednarek¹
Hubert Szadkowski¹
Bartłomiej Sęczkowski¹
Jakub Raj¹
Diana Prośniewska¹

¹Lodz University of Technology, Poland

Supervisor: Bartosz Sakowicz (e-mail: bartosz.sakowicz@p.lodz.pl)

June 2025

Abstract

This report details the development of the "Table Hub Mobile" application, designed to manage restaurant table statuses. We have developed an application that allows users to view restaurant layouts, identify available and occupied tables, and report changes in table status. Our solution integrates real-time communication via WebSockets for immediate updates. Key features include location-based restaurant searching, filtering options by cuisine and rating, and an interactive table layout view for status changes. The prototype we developed utilizes a clean architecture with distinct layers for UI (Fragments, Composables), ViewModels, Services, and data Repositories, ensuring modularity and maintainability. Our testing involves unit tests for core logic and instrumental tests for UI components.

KEYWORDS: Android, Kotlin, Jetpack Compose, WebSocket, Table Management, Restaurant, Mobile Application, Hilt

1 Introduction

This document outlines the final report for the Table Hub Mobile application project. Our team aims to streamline the process of managing restaurant table availability, providing a user-friendly interface for both restaurant staff and customers to quickly ascertain and update table statuses.

1.1 Background Information

The traditional method of tracking restaurant table availability often relies on manual observation or rudimentary systems, leading to inefficiencies, inaccurate information, and potential loss of revenue due to mismanaged seating. Customers frequently face uncertainty regarding table availability, and staff can be overwhelmed during peak hours trying to coordinate seating. This problem scales with the size and popularity of the restaurant, as well as the complexity of its layout with multiple sections.

1.2 Problem Finding

Our detailed user research, including a survey, highlighted several key pain points in restaurant table management. The survey revealed that **over 74% of respondents (specifically 78% combining "often" and "very often" responses) indicated that their outings are spontaneous**, rather than pre-planned. This spontaneity frequently leads to difficulties, as approximately **74% of respondents reported problems finding a free table during peak hours**, a situation particularly prevalent on busy days like Fridays and Saturdays. This underscores a significant need for accessible, real-time information to enhance the dining experience and reduce frustration.

The specific problem we address is the need for a mobile application that provides accurate, real-time table status updates within a restaurant, allowing for efficient management and clear communication to both staff and customers. Existing solutions often lack the dynamic, real-time updates essential for addressing the spontaneity of diners and the challenges of peak hours.

2 Idea Finding

2.1 State of the Art

Current solutions for restaurant management often include reservation systems or POS (Point of Sale) integrations. While these can track bookings, they typically do not offer a dynamic, real-time visual representation of table occupancy or an intuitive way for staff to update statuses on the fly. Some larger restaurant chains might have proprietary systems, but a widely accessible and user-friendly mobile solution for independent restaurants is less common.

2.2 Innovative Ideas

We developed the following innovative ideas to address the stated problem, adding new value to better address the stated problem, possibly leading to more effective solutions:

- **Interactive Floor Plan:** A visual, interactive layout of the restaurant with clickable tables to quickly change their status.
- **Real-time WebSocket Communication:** Utilizing WebSockets for instant updates of table statuses across all connected clients, ensuring data consistency.
- **Location-Based Restaurant Discovery:** Enabling users to find nearby restaurants and view their table availability.
- **Filter Options:** Allowing users to filter restaurants based on various criteria like cuisine, rating, and minimum available seats.
- **Intuitive UI with Jetpack Compose:** Building the interface entirely with Jetpack Compose for a modern, reactive, and consistent user experience.

2.3 Main Idea Selection and Justification

Our team decided to focus on creating a mobile application with an interactive floor plan and real-time WebSocket communication as the core functionalities. We chose this approach because it directly tackles the real-time accuracy and user experience issues we identified. An interactive floor plan provides an immediate visual understanding of the restaurant's state, while WebSockets ensure that this visual representation is always up-to-date for all users. Our user research further indicated a strong potential for a user-driven solution, with **over 74% of respondents (30.1% 'Yes' and 44.7% 'Maybe') expressing willingness to share real-time information about available places, especially when incentivized by discounts, money, or gamification.** Our use of Jetpack Compose facilitates rapid UI development and a high-quality user interface.

3 Solution Implementation

The Table Hub Mobile application is an Android application developed in Kotlin, leveraging modern Android development practices, including Jetpack Compose for UI and Hilt for dependency injection. The application communicates with a backend service via WebSockets to manage restaurant and table data in real-time.

3.1 Technical Details

The application's architecture follows a MVVM (Model-View-ViewModel) pattern, integrated with a repository pattern for data management and services for business logic and network communication.

3.1.1 Core Components

- **Activities:** MainActivity serves as the single activity host for the application's fragments.
- **Fragments:**
 - **LogInFragment:** Handles user authentication.
 - **MainViewFragment:** Displays the main map view with restaurants and allows navigation to other features.
 - **ReportViewFragment:** Shows a list of restaurants for reporting purposes.
 - **RestaurantLayoutFragment:** Displays the interactive layout of a selected restaurant's tables.
 - **SettingsFragment:** Provides settings options.

- **ViewModels:**
 - **MainViewModel:** Manages UI-related data for the main map view, including restaurants and user location, and handles fetching user location.
 - **ReportViewModel:** ViewModel for the report view.
- **Services:**
 - **TablesService** (interface) and **TablesServiceImplementation:** Define and implement the core logic for table status updates, interacting with the **WebSocketService**.
 - **WebSocketService:** Manages WebSocket connections and communication with the backend server for real-time data exchange. It handles connecting, subscribing to updates, and sending table status changes.
 - **MockTablesService:** A mock implementation of **TablesService** for testing and development purposes.
- **Repositories:**
 - **IRestaurantsRepository** (interface) and **RestaurantsRepositoryImpl:** Abstract and implement the data layer for managing restaurant information, including processing initial data and table status changes.
- **Models:** Data classes representing core entities like **Location**, **Position**, **Restaurant**, **Section**, **Table**, and **TableStatus**. WebSocket-specific models (e.g., **WebSocketMessage**, **MessageType**, **TableUpdateRequest**) are also defined.
- **Dependency Injection:** Hilt is used for dependency injection, making the application components easily testable and managing their lifecycle. Modules like **ClientModule**, **MessageProcessorModule**, and **RepositoryModule** provide instances of WebSocket clients, message processors, and repositories.
- **UI (Jetpack Compose):** The user interface is built declaratively using Jetpack Compose. Examples include **MainLoginView**, **MainMapView**, **RestaurantLayoutMainView**, and various reusable composables for buttons, text fields, and pop-ups.

3.1.2 Key Features and Implementations

- **Location Services:** The **MainViewModel** requests and fetches the user's last known location using **LocationServices.getFusedLocationProviderClient**. Necessary permissions (**ACCESS_FINE_LOCATION**, **ACCESS_COARSE_LOCATION**) are declared in **AndroidManifest.xml** and handled at runtime.
- **Map Integration:** The **MapboxMapWrapper** composable integrates Mapbox for displaying restaurants on a map. Custom markers are created to show the number of available tables.
- **Real-time Communication:** The **WebSocketService** establishes and manages a STOMP (Simple Text Oriented Messaging Protocol) WebSocket connection to <ws://192.168.18.35:8080/ws>. It subscribes to **/topic/restaurant/status** for initial status and **/user/queue/updateTableStatus** for table updates, and sends updates to **/app/updateTableStatus**. The **WSMessageRelay** acts as a central hub for WebSocket messages.
- **Data Processing:** The **WebsocketMessageProcessorImpl** observes the **WSMessageRelay** and processes incoming WebSocket messages based on their **MessageType** (e.g., **QUERY_RESTAURANTS_RESPONSE**, **SUBSCRIBE_RESTAURANT_UPDATES_RESPONSE**, **TABLE_STATUS_CHANGED_EVENT**), updating the **RestaurantsRepositoryImpl** accordingly.
- **Table Status Update:** Within **RestaurantLayoutFragment**, users can select a section and then tap on individual tables to change their status (**AVAILABLE**, **OCCUPIED**, **UNKNOWN**) via an **AlertDialog**. These updates are then sent to the backend through the **TablesServiceImplementation**.
- **User Interface Styling:** The app defines its color scheme in **colors.xml** and **Color.kt**, and themes in **themes.xml** and **Theme.kt**, providing a consistent visual identity. String resources for localization are available in **strings.xml** and **strings-pl.xml**.

3.1.3 UML Diagrams

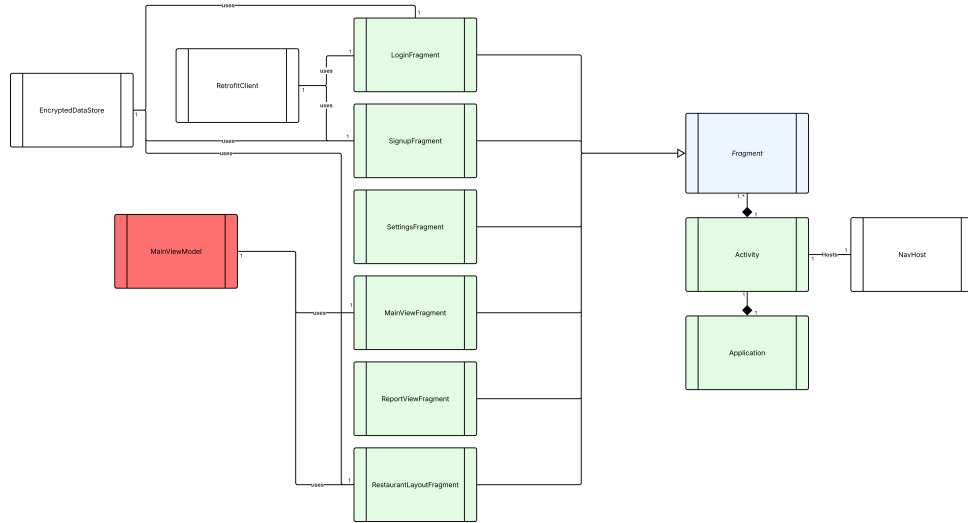


Figure 1: UI Component Diagram

This diagram (Figure 1) illustrates the hierarchy and interaction of the main UI components, showing how fragments and activities are structured and how they relate to the overall application flow. It also depicts the connection to external components like `MainViewComponent` and `LoginComponent`.

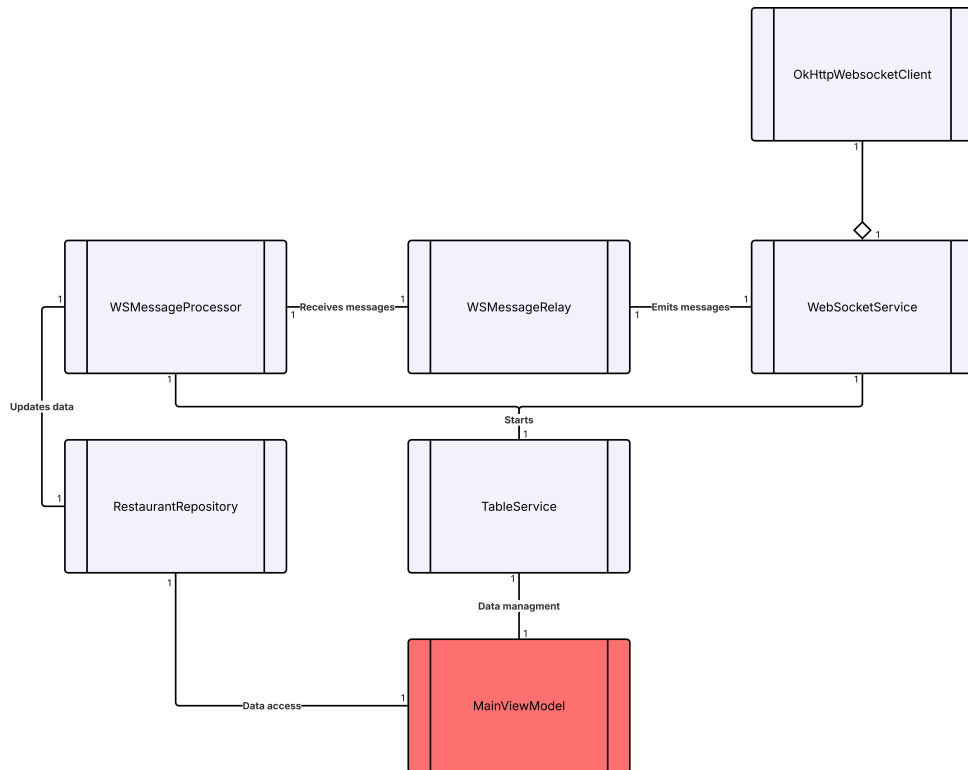


Figure 2: Service and Repository Interaction Diagram

As seen in Figure 2, this diagram details the relationships between the WebSocket communication components (`OkHttpClient`, `WebSocketService`), the message processing logic (`WSMessagesRelay`, `WSMessagesProcessor`), and the data layer (`RestaurantRepository`). It highlights how the `MainViewViewModel` accesses data and how `TableService` initiates data management.

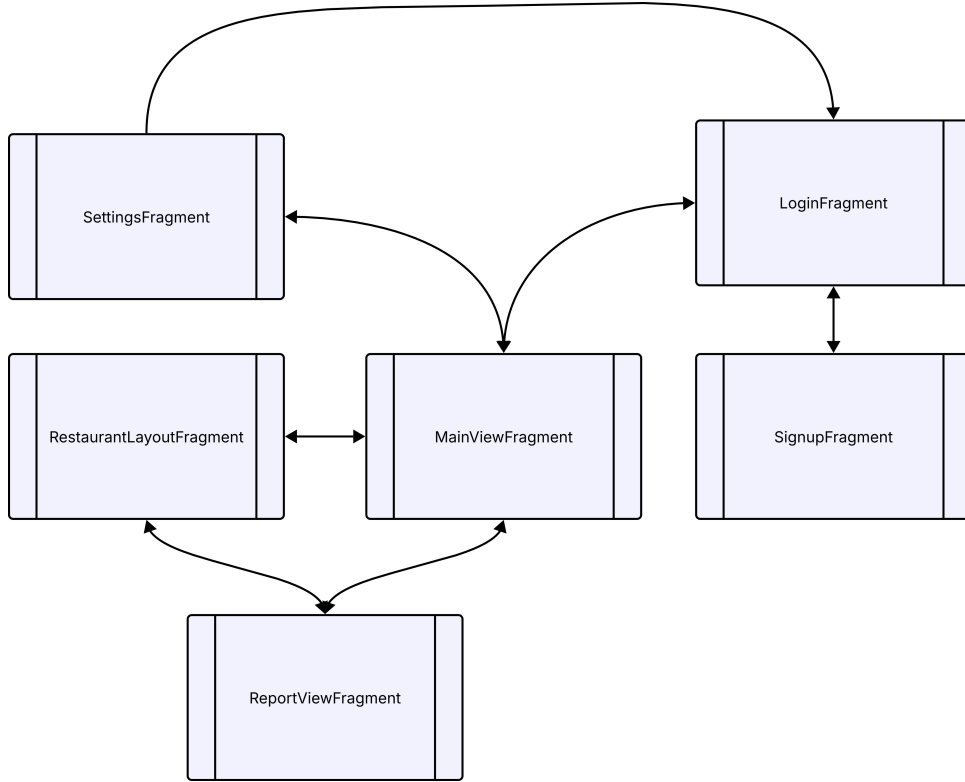


Figure 3: Navigation Graph Diagram

The navigation graph (Figure 3) visually represents the flow between different fragments in the application. It shows the possible transitions between `LoginFragment`, `SignupFragment`, `MainViewFragment`, `SettingsFragment`, `ReportViewFragment`, and `RestaurantLayoutFragment`.

4 Ways of Verification

The solution's effectiveness can be verified through a combination of unit testing, integration testing, and user acceptance testing.

- **Unit Tests:** We test individual components, especially business logic in ViewModels and data handling in Repositories, in isolation. For example, `ExampleUnitTest.kt` demonstrates a basic unit test for an addition function.
- **Integration Tests:** We perform testing of the interaction between different layers, such as a View-Model interacting with a Repository, or the `TablesServiceImpl` with the `WebSocketService`, to ensure proper data flow and communication.
- **UI Tests (Instrumented Tests):** Using AndroidX Test and Espresso (or Compose testing APIs), we test the UI components and navigation flows on an actual device or emulator. `ExampleInstrumentedTest.kt` shows a basic instrumented test verifying the app's package name.
- **User Acceptance Testing (UAT):** Real users (restaurant staff and customers) would interact with the application in a live environment to validate its usability, functionality, and ability to solve the problem efficiently. Key metrics would include the time taken to update table statuses, accuracy of reported availability, and overall user satisfaction.

5 Conclusions and Perspectives

The Table Hub Mobile application offers a robust and intuitive solution for real-time restaurant table management. Its modular architecture, leveraging Kotlin, Jetpack Compose, and Hilt, promotes maintainability and scalability. The integration of WebSockets ensures that table status updates are instant and synchronized across all users.

5.1 Strengths

- Real-time data synchronization.
- User-friendly and interactive UI for table management.
- Clear separation of concerns through MVVM and Repository patterns.
- Leverages modern Android development tools.

5.2 Weaknesses

- **Fixed WebSocket URL:** The application currently relies on a hardcoded WebSocket server URL (`ws://192.168.18.35:8080/ws`). This limits flexibility and would need to be made configurable for broader deployment.
- **Basic WebSocket Error Handling:** Error handling for WebSocket communication could be improved. Implementing retry mechanisms or more robust user notifications for connection issues would enhance reliability.

5.3 Potential Follow-up

- **Backend Integration:** Develop the full backend service to complement the mobile application, ensuring secure and scalable data storage.
- **User Authentication and Profiles:** Implement a complete user authentication system with different roles (e.g., staff, customer) and personalized profiles.
- **Reservation System Integration:** Extend functionality to include table reservation capabilities directly within the app.
- **Notifications:** Implement push notifications for significant events, such as a requested table becoming available.
- **Admin Panel:** Create a web-based admin panel for restaurant owners to manage their layouts, sections, and tables.
- **Improved Location Accuracy:** Integrate more advanced location services or manual location input for better accuracy.

Appendix

App Views

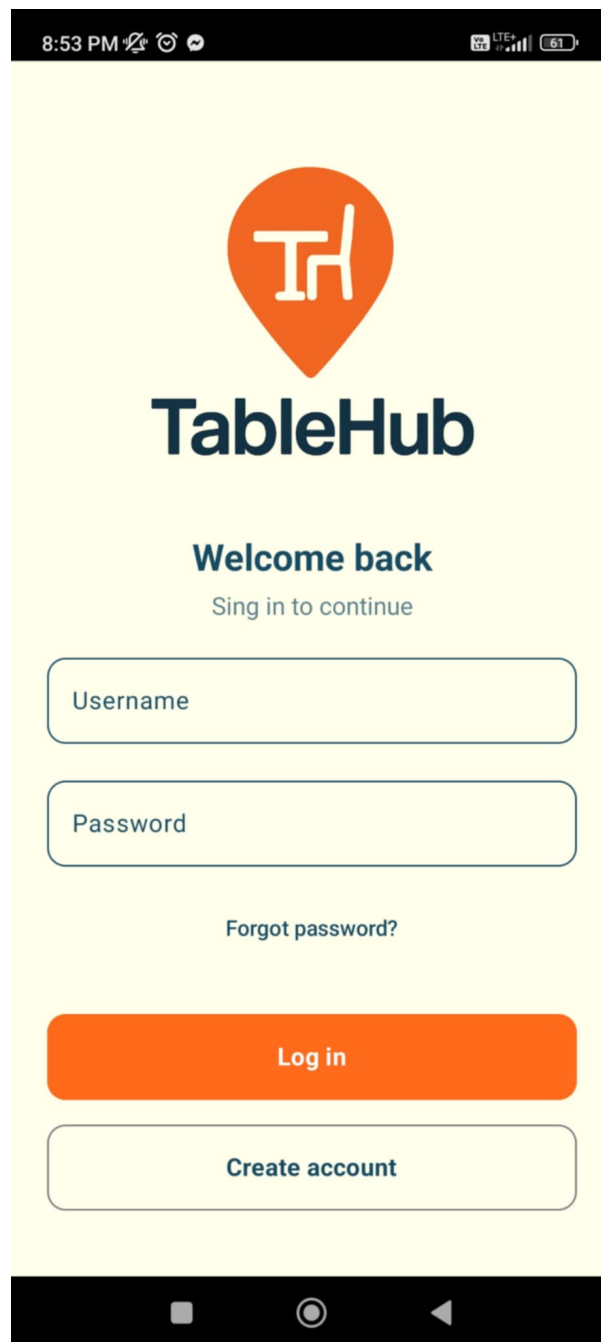
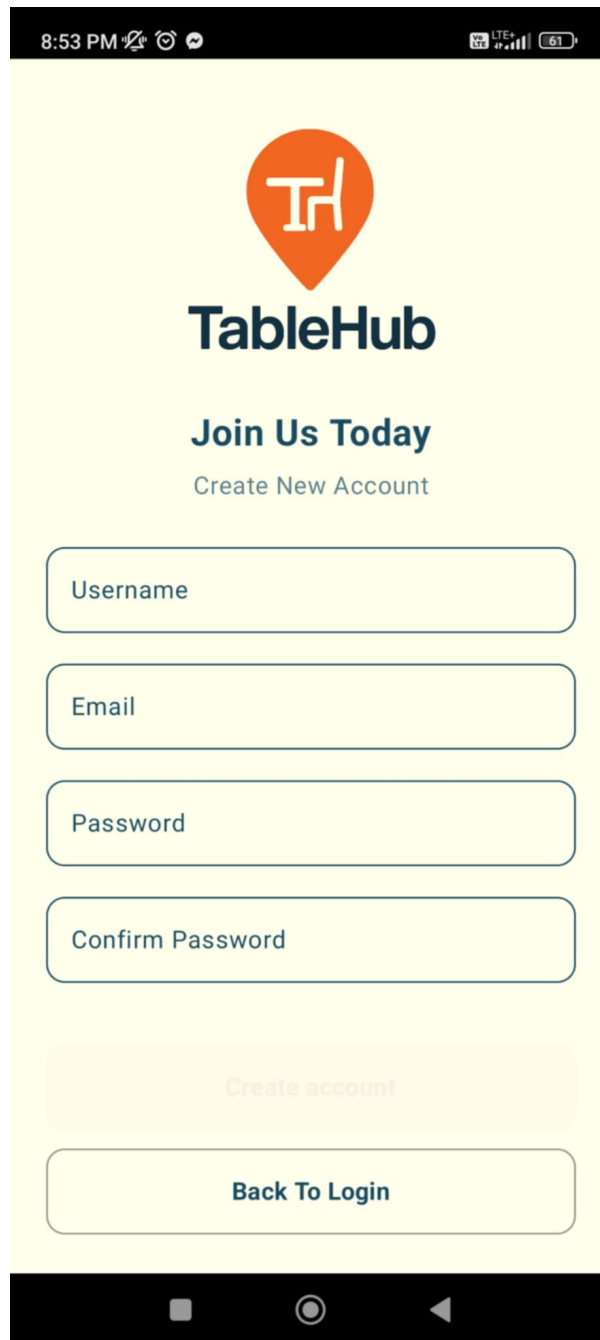



Figure 4: Login Screen

This view provides user authentication, allowing existing users to log in and offering options for new account creation or password recovery.



The image shows a mobile application sign-up screen for 'TableHub'. At the top, there is a status bar with the time '8:53 PM', signal strength, and battery level '61%'. Below the status bar is the TableHub logo, which consists of an orange location pin icon with the letters 'TH' inside. The text 'TableHub' is displayed in a bold, dark blue font. Below the logo, the text 'Join Us Today' is written in a bold, dark blue font, followed by 'Create New Account' in a smaller, lighter blue font. There are four input fields for registration: 'Username', 'Email', 'Password', and 'Confirm Password'. Each field is a rounded rectangle with a thin blue border. Below these fields is a large, light orange button with the text 'Create account' in a lighter orange font. At the bottom of the form is a rounded rectangle button with a thin blue border and the text 'Back To Login' in a bold, dark blue font. The entire form is set against a light yellow background. At the very bottom, there is a black navigation bar with three white icons: a square, a circle, and a triangle.

8:53 PM 61%



TableHub

Join Us Today
Create New Account

Username

Email

Password

Confirm Password

Create account

Back To Login

Figure 5: Sign Up Screen

This view facilitates new user registration, capturing necessary details for account creation.

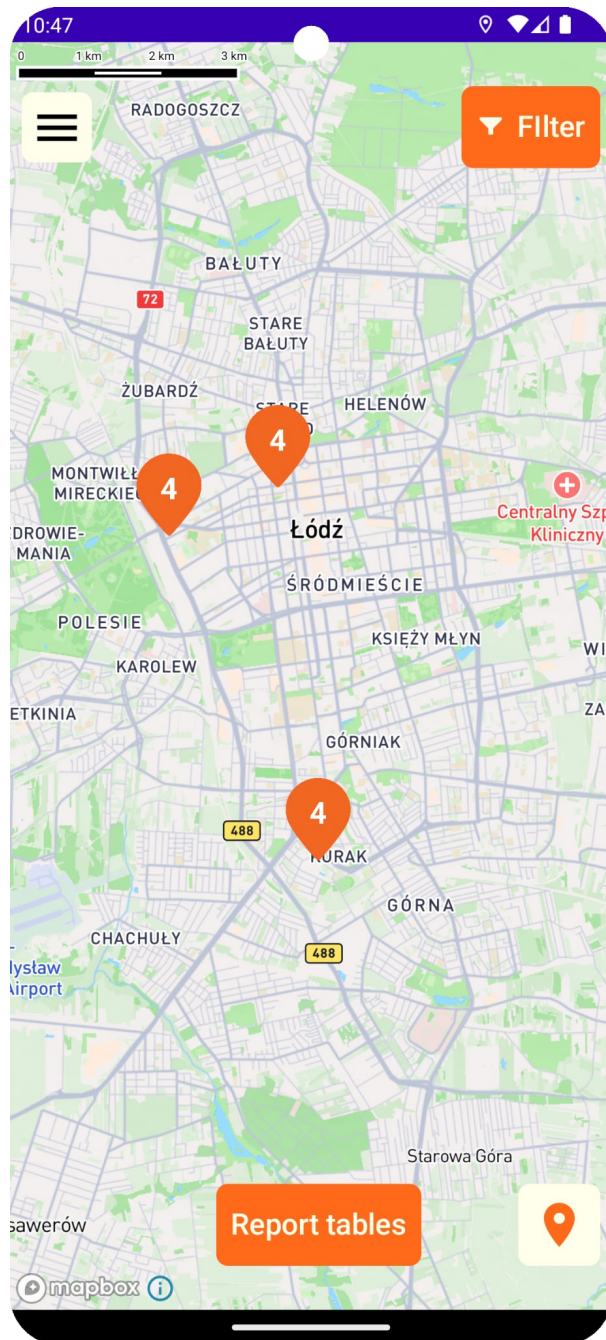


Figure 6: Map View with Restaurants

This map-based interface enables users to discover nearby restaurants, with visual markers indicating table availability and access to filtering options.



Figure 7: Restaurant Information Pop-up on Map View

This pop-up presents concise details for a selected restaurant from the map, including address, cuisine, rating, and available tables, with options for further action.

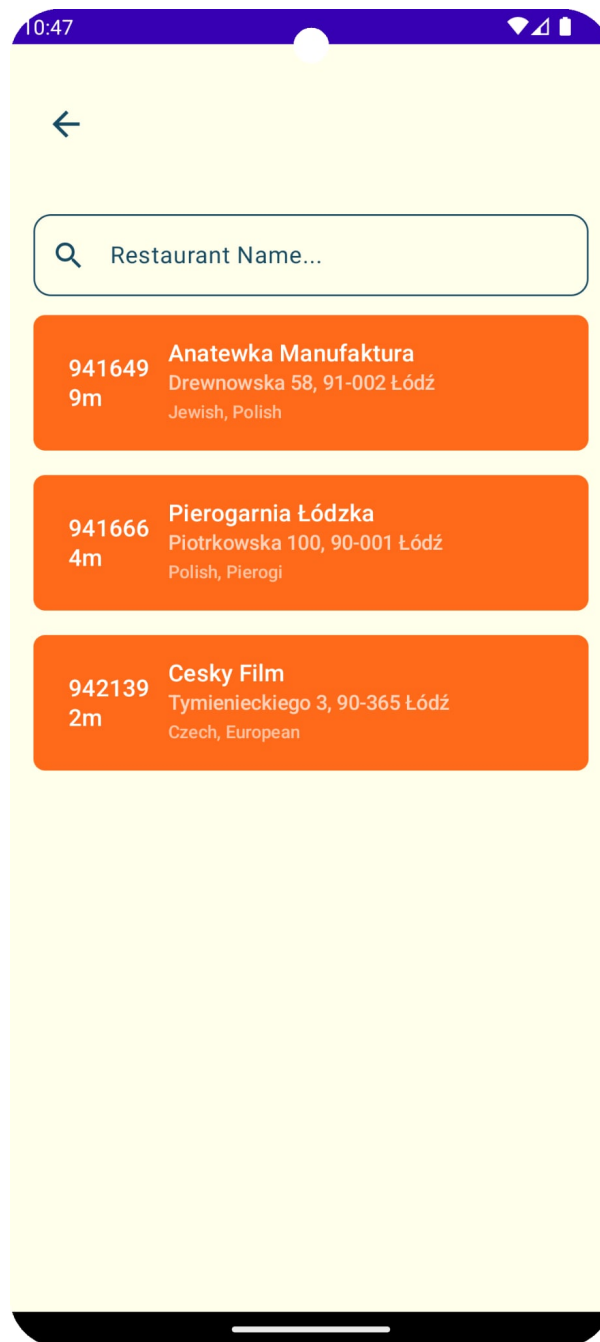


Figure 8: Restaurant Search and List View

This interface allows users to search for restaurants by name and browse results in a list format, displaying key information like name, address, distance, and cuisine.

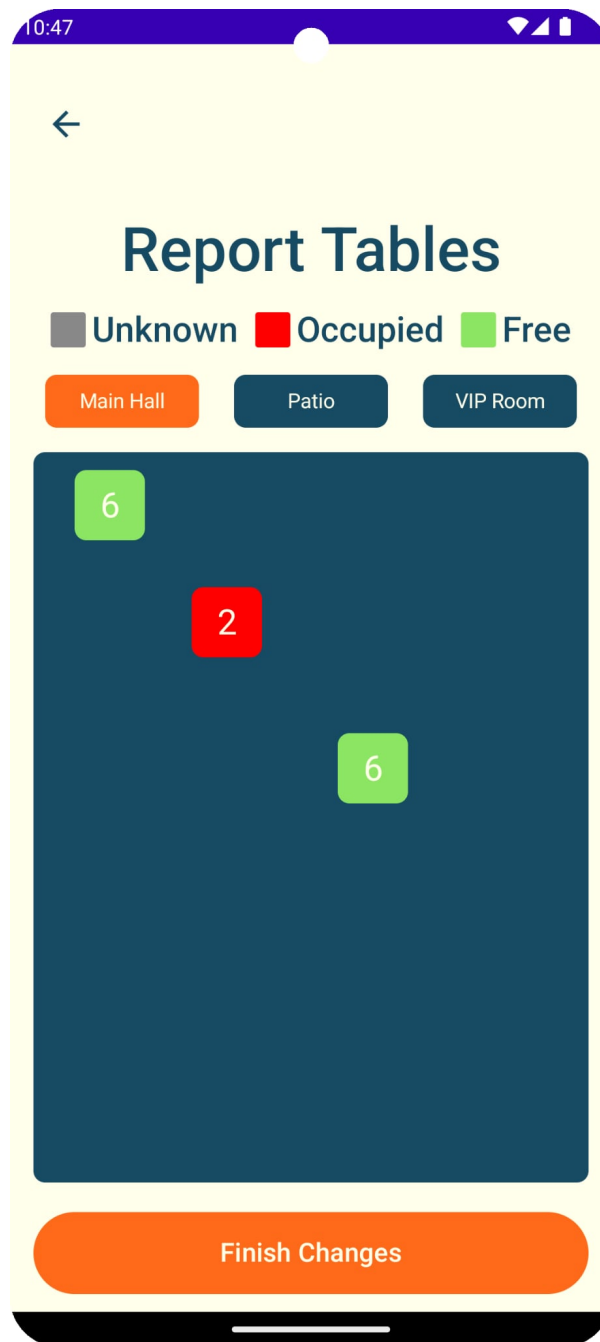


Figure 9: Interactive Table Layout View

This interactive view presents a visual floor plan of a restaurant section, distinguishing tables by their current status (e.g., free, occupied) and allowing users to initiate status changes.

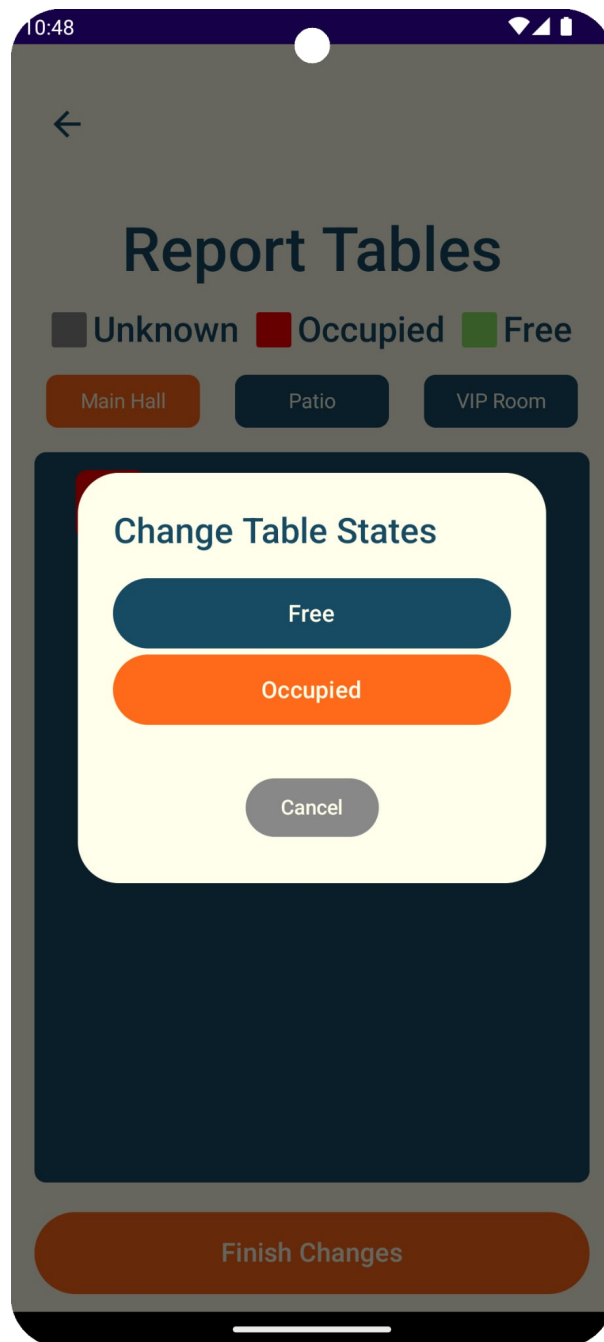


Figure 10: Change Table States Dialog

This dialog allows users to update the status of a selected table (e.g., to 'Free' or 'Occupied'), facilitating real-time status reporting.

Code Repository

<https://github.com/Table-Hub-TUL>

Acknowledgment

We would like to express our sincere gratitude to our supervisor, Bartosz Sakowicz, for their invaluable guidance and support throughout this project. Their expertise and encouragement were instrumental in bringing Table Hub Mobile to fruition.