Digital Egypt Pioneers Initiative - DEPI

Mahattaty Application

Documentation

NEXT38 _CAI1_SWD4_S10D

Name	Email
Abdelrahman Reda Mohamed	Abdelrahmanrmuhammed@gmail.com
Rana Esmail Zekery	Ranaesmailhelal@gmail.com
Aya Shawky Elshahat	ayashawky2441907@gmail.com
Roaa Emad-Eldin Muhammad	roaa20102002@gmail.com
Amr Anwar Yonus	amranwarx2@gmail.com
Doha Ibrahim Ahmed	doha.ibrahim2003@gmail.com

Instructors

Josph Gayed, Fady Sameh

Contents

Introduction

Goals

The primary goal of this project is to create a seamless Train Reservation Application using Flutter, integrated with the Riverpod state management and structured with Clean Architecture principles. We aimed to provide a robust solution that enhances user experience in train ticket booking and tracking while maintaining a scalable, maintainable codebase.

Goals Overview

- Provide an Intuitive User Experience: Make train booking simple, fast, and reliable for all users.
- Cross-Platform Availability: Ensure smooth functionality across iOS and Android devices using Flutter.
- Scalability and Maintainability: Build the app with clean architecture to support future growth and easy feature enhancements.

Requirements

Functional Requirements

ID	Requirement Statement
F1	The system should display a splash screen featuring the app logo, followed by three consecutive onboarding screens. These onboarding screens should only appear during the first launch of the application and introduce the app's features. After the splash screen, the app should navigate to the login screen if there is no active user session, or to the main screen if a user session is already active.
F2	The system must provide fields for entering an email and password, with buttons available for both logging in and creating a new account. When a user chooses to sign up, they must be prompted to provide a valid email address, a password that is at least 8 characters long (including one uppercase letter and one digit), a password confirmation, and a username. The system should support login options via social accounts, such as Google, and allow users to reset their password in case they forget it.

F3	The system should allow users to search for trains by selecting a departure station, an arrival station, a departure date, and a ticket type (either One Way or Round Trip). A "Search for Trains" button must be provided to initiate the search for available train tickets based on the selected criteria.
F4	After the user performs a train search, the system should display a list of available trains that match the search criteria. This list should include information such as the train name, type, departure time, arrival time, trip duration, and ticket price
F5	When a user selects a train, the system should display detailed information about the selected train, including the departure and arrival times, total price, available seat types, the number of available seats for each type, and the price for each seat type. Users should be able to choose their preferred seat type, and the system should update the total price accordingly. The system must include a "Confirm" button to finalize the booking process, allowing only one reservation per user for a specific train.
F6	The system must support multiple payment options, such as credit cards and e-wallets, to process ticket bookings. A "Confirm Payment" button should be included to allow users to finalize the payment and complete their booking.
F7	Once a ticket booking is confirmed and payment is successfully processed, users should be able to view the details of their tickets in the "My Tickets" section. Each ticket should include a QR code that can be scanned when entering the train station. The system should also allow users to filter their tickets by date, train type (Express, Ordinary, Touristic), and ticket state (Upcoming, Done).
F8	The system should provide users with the ability to view and manage their account settings. Users should be able to edit their username (but not their email) and update their password by entering a new password and confirming it. The system must ensure that the new password meets the security criteria and matches the confirmation. A "Save Changes" button should apply any updates, and a "Reset" button should reset the settings.
F9	The system should allow users to change the app’s language from a list of available options, with the selected language applying immediately across the app.
F10	The system should allow users to toggle between light and dark modes, with the selected appearance being applied instantly

F11	The system should provide Legal information about the application such as the Terms and Conditions, Privacy Policy, and Changes to the Service and they should be accessible, scrollable, and clearly displayed.
F12	The system should include a "Logout" option, and when selected, a confirmation dialog should appear with options to confirm or cancel the logout action.
F13	The system should display a list of train-related news articles on the Explore screen, showing a summary of each article, including the title and publication date. It should also display a list of the best available offers for train tickets, along with the timeframe for which the offer is valid. Users should be able to click on any news article to view its full details on a separate page.

Non - Functional Requirements

Name	Requirement Statement
Performance	• All screens must load within 2-3 seconds. • We should be sure that login screen are fast loading screens “no more than 2 seconds to load • The train search process must be completed within 5 seconds
Scalability	The system should be able to Support about 1000 users at the same time
Availability	The system must ensure a service availability of no less than 99%, with regular maintenance during off-peak hours.
Security	• Users should have access to their private information and payment information. • All user data and financial transactions must be encrypted using SSL/TSL protocols.
Compatibility	• The system must be compatible with Android and iOS platforms, supporting various screen sizes and resolutions. • The system must support functionality across different web browsers, including Chrome, Firefox, and Safari.
Usability	• The user interface must be intuitive, with clear steps and visual guidance for navigation. • The system must provide clear and informative error messages
Maintainability	• The system must be designed to allow database modifications and feature additions without affecting overall performance.

	• Regular security and feature updates must be planned and implem

Implementation

Architecture Pattern

This project follows a **modular architecture** with a clear separation of concerns to ensure maintainability, scalability, and testability. The core architecture pattern used is **MVVM (Model-View-View Model)**, along with some components of **Clean Architecture** principles to manage the business logic and data flow effectively.

1. Model-View-View Model (MVVM) Pattern

The **MVVM** pattern is applied to ensure a clean separation between the UI, business logic, and data layers, making the codebase easy to maintain and extend. This pattern is particularly useful in a mobile development framework like Flutter, allowing for a highly responsive and modularized design.

- **Model (M)**:
  - The **Model** represents the data layer of the application, including the structure of data and how it's fetched or stored. The models are responsible for handling data from APIs (e.g., Firestore, .NET backend) and converting it into a format that the View Model can understand.
  - Example: The News and Train classes represent data models that hold information like news articles or train schedules.
- **View (V)**:
  - The **View** refers to the user interface (UI) of the application, where the data is displayed. In Flutter, this is represented by the widgets that users interact with. The View listens to changes in the View Model and updates itself accordingly.
  - Example: Flutter widgets display lists of news articles or train schedules, based on the data processed by the View Model.
- **View Model (VM)**:
  - The **View Model** acts as a mediator between the Model and the View. It holds all the presentation logic and transforms the Model data into a form that the View can use. The View Model uses state management tools like **Riverpod** to ensure efficient state updates.
  - Example: In the **train booking module**, the View Model manages fetching train data, handling seat selections, and responding to user input in the View (e.g., updating the UI when a seat is selected).

2. Clean Architecture

In addition to MVVM, **Clean Architecture** principles are applied to maintain a separation between layers, ensuring that business logic is independent of the UI and data layers. This allows the project to be flexible for future modifications, making it easier to change technologies or replace components (e.g., switching data sources).

- **Presentation Layer**:
  - Contains the **View Model** and UI components. Handles user input, displays data, and reacts to changes in state. This layer communicates with the **Domain Layer** to fetch data and make updates.
- **Domain Layer**:
  - Contains the core business logic. It is independent of the frameworks and manages operations like train booking, seat selection, or news voting. This layer includes use cases and domain models.
- **Data Layer**:
  - Manages all data-related operations. This includes interacting with APIs (e.g., fetching news articles, train schedules) and local databases (e.g., caching news articles for offline use). The data layer is completely abstracted from the UI.

3. State Management with Riverpod

For efficient state management, **Riverpod** is used throughout the application to manage application states (like booking progress, news list status) and enable reactive UI updates. Riverpod helps in maintaining the state independently of the UI and allows sharing states across different widgets.

- **Scoped State Management**: Each module (news, train booking, onboarding) maintains its own independent state, ensuring that changes in one part of the app do not affect others.
- **Listeners and Providers**: The View listens to providers from Riverpod, and whenever the View Model updates the state, the UI automatically reflects these changes.

4. Modular Approach

The app is divided into distinct modules (auth, news, train booking, onboarding, settings), each encapsulating its specific functionalities. Each module follows the **MVVM** pattern internally, ensuring a consistent architecture across the app.

- **Module Separation**: Each module (e.g., auth, news, train booking) is treated as a separate feature with its own responsibilities, View Model, and UI. This modular approach allows independent development and testing of each feature, ensuring that one module can be modified or updated without affecting the others.

5. Backend Integration

The backend is powered by **Firebase** using **Firestore** as the main database to store and retrieve data. The app communicates with Firestore to fetch and store information like user authentication data, train schedules, and news articles.

- **Firestore as Backend**:

  - **Firestore** is used for real-time database management. The app interacts with Firestore to retrieve real-time data for news, train schedules, and user details. Firestore ensures that the app stays up to date with any changes in the backend data.

- **Shared Preferences**:
  - The app uses **Shared Preferences** for lightweight local storage, such as tracking user preferences and application states. For example, it stores a flag to check whether the onboarding process has been completed, ensuring that the onboarding screens are only shown on the first app launch.

RESTful Services

While Firestore handles real-time data, RESTful API principles are still used in the app's interaction with Firebase. Data is fetched and updated through asynchronous **HTTP requests** to Firebase services, ensuring smooth communication between the client and server layers.

- **Data Serialization**: All data fetched from Firestore is serialized into Dart models before being used within the application. This ensures that data is consistently structured and easily manageable.

Caching

To provide a better user experience, especially when network access is unavailable, the app implements **data caching**. This is particularly important for the news module, where news articles are cached locally using **Fire store's offline persistence** capabilities.

- **Offline Access**: Fire store's built-in offline persistence allows the app to function even when the device is offline. Users can view previously loaded news articles and other cached data while offline.
- **Shared Preferences Caching**: Shared Preferences is also used for storing small pieces of persistent data such as user preferences, which are retained across app sessions.

Conclusion

The combination of **MVVM**, **Clean Architecture**, and **Riverpod** for state management ensures that the app is maintainable, scalable, and modular. This architectural approach allows for easy integration of new features, smooth user interaction, and a robust backend communication flow. Each module is developed independently, following the same pattern, making it easy to extend and update the application in the future.

Agile Development Plan

This project follows Agile methodology, dividing development into multiple short sprints. Each sprint focuses on delivering a specific module in 3-4 days. The development plan is as follows:

Sprint 1: Authentication Module (4-5 days)

Objectives:

- Implement user authentication including registration, login, and password reset.
- Ensure proper session management and error handling for incorrect credentials.

Tasks:

- Day 1-2: Design and implement the user interface with multiple languages and animations
- Day 3-4: Set up user authentication and integrate backend services.
- Day 4-5: Add password reset functionality and test the authentication flow.

Deliverables:

- Fully functional authentication system that includes:
  - User registration
  - Login
  - Password reset

Sprint 2: Onboarding Module (3-4 days)

Objectives:

- Create an onboarding system that is shown only on the first launch of the app
- Create the splash screen UI that will display every time the app starts.

Tasks:

- Day 1-2: Design and implement onboarding screens with multiple languages and animations
- Day 3-4:
  - Implement logic to ensure the splash screen is shown on every launch.
  - Ensure that the onboarding process only appears on the first launch by using Shared Preferences.
  - Conduct user testing to verify that the onboarding and splash screen functionality works as intended.

Deliverables:

- Functional onboarding system that:
  - Displays only on first launch
  - Smoothly transitions between screens
  - Stores a flag in Shared Preferences to track first launch
- A splash screen that:
  - is shown every time the app launches.

- Provides a seamless user experience as the app transitions to the main interface.

Sprint 3: Train Booking Module (6 days)

Objectives:

- Implement the train booking system including search, seat selection, and confirmation.

Tasks:

- Day 1-2:
  - Build the user interface for searching available train schedules based on criteria such as date, time, and destination.
  - Integrate backend services to fetch train data, ensuring that users receive real-time information.
- Day 3:
  - Develop custom components to display best offers and promotions on train bookings, enhancing the user experience.
  - Ensure the interface is intuitive and visually appealing.
- Day 4-5:
  - Implement seat selection functionality, allowing users to choose specific seats from a seating chart.
  - Incorporate booking confirmation logic, where users can review their selection before finalizing the purchase.
- Day 6:
  - Test the entire booking process with mock data or real APIs to ensure functionality, accuracy, and a seamless user experience.
  - Conduct user testing to gather feedback and make necessary adjustments.

Deliverables:

- A fully functional train booking system that includes:
  - A user-friendly interface for searching and filtering available train schedules.
  - Custom components to showcase the best offers and promotions.
  - An intuitive seat selection process with a clear seating chart.
  - A booking confirmation feature that allows users to review and finalize their selections.
  - Integration with backend services to ensure real-time data fetching and booking accuracy.

Sprint 4: News Module (5 days)

Objectives:

- Implement the train booking system including search, seat selection, and confirmation. Develop a dynamic news module that displays news articles and implements caching for offline access.

Tasks:

- Day 1-2:
  - Set up the user interface for displaying news articles, including essential elements such as title, description, image, and publication date.
  - Integrate backend services to fetch news data from Firestore or an external API, ensuring real-time updates
- Day 3:
  - Implement pagination or lazy loading to enhance performance when displaying many articles.
  - Develop custom components for displaying article details, ensuring a visually appealing layout.
- Day 4-5:
  - Conduct thorough testing to ensure all functionalities work correctly and efficiently.
  - Gather user feedback to refine the interface and interaction flow.

Deliverables:

- A fully functional news module that includes:
  - A user-friendly interface for displaying news articles with relevant details.
  - Efficient pagination or lazy loading for performance optimization.
  - Real-time data fetching capabilities from the backend.
  - Caching functionality to enable offline access to previously view articles.

Sprint 5: Settings Module (3 days)

Objectives:

- Provide users with the ability to modify app settings and manage their profiles.

Tasks:

- Day 1-2:
  - Design the settings page UI, including options for account management and preferences.

- Day 3:
  - Integrate backend settings persistence and test on multiple devices

Deliverables:

- A functional settings page that:
  - Allows users to update their profiles and manage settings
  - Integrates with the backend for persistence

Sprint 6: Testing, Integration, and Cleanup (2 days)

Objectives:

- Perform final integration of all modules and conduct comprehensive testing to ensure app stability

Tasks:

- Day 1:
  - Conduct end-to-end testing of each module to identify any bugs or issues.
- Day 2:
  - Finalize integration, resolve

Future Improvements

- Future Features**:**
  - Additional login options (e.g., Facebook, Twitter).
  - More payment methods.
  - User profile customization (profile pictures and travel preferences).
  - Push notifications for ticket confirmations and promotions.
  - In-app chat support for customer assistance.
  - Real-time train tracking features.
  - Loyalty rewards program for frequent travelers.
  - Social sharing options for travel plans.
  - **Extended Multilingual Support**: Beyond Arabic and English, potentially adding more languages to cater to a wider audience.
  - Enhanced explore features with personalized offers.
  - Review and rating system for user feedback

Use case Diagram