Chapter 1

The Implementation of our App

1 Introduction

This chapter is dedicated to the practical realization of our mobile application. It provides a detailed overview of the technical environment in which the project was developed and tested, including both hardware and software components.

We will begin by presenting the development and testing equipment used, followed by the tools and technologies adopted throughout the implementation process. In addition, we will present the application's key interfaces and explain how they interact.

Finally, this chapter will showcase selected code snippets and logic responsible for handling certain events and functionalities in the application — offering a clear view of the internal mechanisms that drive the user experience.

2 Work Environment

2.1 Hardware Environment

For development and testing, the following hardware was used:

- **Development Machines:**
  - **HP EliteBook 840 G4** – Intel Core i5-7200U, 16GB DDR4 RAM (2444MHz), 256GB SSD.
  - **Dell Laptop 1** – Intel Core i5 9th Gen, 16GB RAM.
  - **Dell Laptop 2** – Intel Core i5 6th Gen, 16GB RAM.

- **Test Devices:**
  - Samsung Galaxy S21
  - Samsung Galaxy S21 FE
  - Redmi Note 8 Pro

2.2 Software Environment

2.2.1 Technologies Used

It presents the main technologies adopted, such as the programming language, development platform, target mobile environment, and the chosen database management system.

2.2.1.1 Presentation

For the development of our mobile application, we adopted a set of modern tools and technologies to ensure the quality, performance, and maintainability of the project. Below is an overview of the main components of our software environment:

- **Programming Language:** We used **Dart**, through the **Flutter** framework, to develop a cross-platform mobile application (Android and iOS) from a single codebase.
- **Development Environments:** We primarily used **Visual Studio Code** and **Android Studio** as our code editors and development environments, which are widely recognized and powerful for mobile app development.
- **Backend:** We chose the **Laravel** PHP framework for creating the backend API. It provided us with a clear MVC structure, enhanced security, and easy management of routes and controllers.
- **Database:** The application uses a **MySQL** database, which we designed and hosted on a **Hostinger** server. The backend files were also deployed on this server to handle API communication with the mobile app.
- **Firebase:** Integrated for backend services, particularly **authentication**. We used **Google Console** to enable patients to log in using their Google accounts.
- **Testing Tools:** We used **Postman** to test various API requests and endpoints before integrating them into the mobile app.

- **Version Control:** Our source code was managed using **Git** and hosted on **GitHub**, which facilitated team collaboration and version tracking.
- **Design Tools:** We used **Figma** to design the user interface of the app and create user flow diagrams. For database modeling and class diagrams, we used **dbdiagram.io**.
- **Additional Tools:** *[This section is reserved for any additional tools or technologies we may have used during the development process.]*

2.2.1.2 Database Management System Used

In this project, special focus was given to designing and implementing the database using Laravel, which provides an efficient system for migrations, models, and seeders. This section describes the workflow, tools, and structure adopted for managing data throughout the application development.

The database schema was initially designed using **dbdiagram.io**, a web-based tool that allows for clear visualization of tables, their fields, and relationships. This step served as a blueprint, ensuring a well-structured and relational database design before implementation.

After finalizing the design, the schema was implemented using Laravel's **migration system**. Each table was defined in a separate migration file, outlining columns, data types, primary and foreign keys, and relevant constraints. Laravel's migration mechanism supports version control for the database, allowing for easy maintenance and scalability over time.

Following the migration setup, a dedicated **Eloquent model** was created for each table. These models act as an abstraction layer between the application and the database, handling operations such as data insertion, updates, deletion, and retrieval. Laravel automatically maps models to their corresponding tables and supports defining inter-model relationships (e.g., `hasMany`, `belongsTo`, `hasOne`, `belongsToMany`) that mirror the relational schema.

To initialize the database with essential application data, **seeders** were created. Seeders insert predefined data into the tables, making them particularly useful during development, testing, or on first-time deployments.

Throughout the development process, the database was managed and inspected using **phpMyAdmin**, a popular web-based tool for administering MySQL databases. phpMyAdmin facilitated direct interaction with the database for purposes such as data verification, relationship checks, and query testing, complementing Laravel's command-line and code-driven operations.

Although backend authentication **(using Firebase And Laravel)** was not the focus of this section, the system does implement strict user authentication and role-based access control. Each user is assigned a specific role upon registration (e.g., doctor, clinic, admin, or general user), and strict permission logic ensures that a user of one type cannot access another role's interface or perform unauthorized actions. For example, a doctor account cannot log in as a normal user or admin, and vice versa.

The authentication system integrates **Firebase Authentication** (via email/password and Google Sign-In), managed through the **Google Cloud Console**. Upon successful authentication on the client side (using Flutter), the user receives a Firebase token, which is then sent to the backend. On the Laravel side, we verify this token using Firebase's Admin SDK and issue a Laravel session token using **Laravel Sanctum**.

Sanctum provides a simple and secure way to manage API tokens and session authentication. We implemented custom middleware that checks the user's Firebase UID, retrieves their corresponding role from the database, and enforces access rules based on that role. Unauthorized access attempts result in a 403 Forbidden error, with clear JSON responses for frontend handling.

The following figures illustrate two key components of the system's security architecture. The first diagram explains how authentication is handled using Firebase and Laravel Sanctum, ensuring only verified users can access the system. The second diagram focuses on how user roles and permissions are enforced to control access to specific resources within the application.

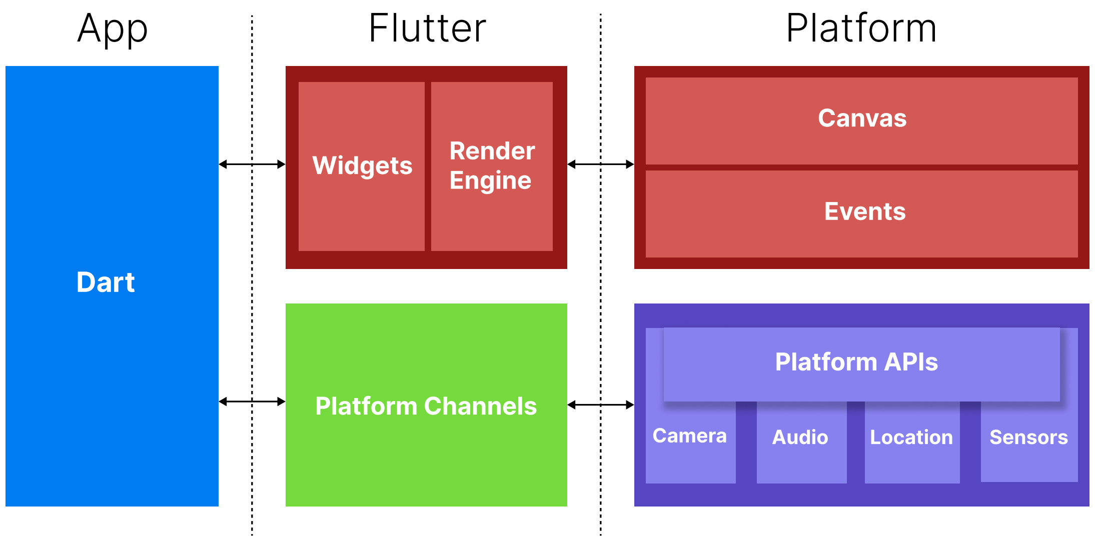

Figure 1.1: Authentication Flow: Firebase Authentication integrated with Laravel Sanctum

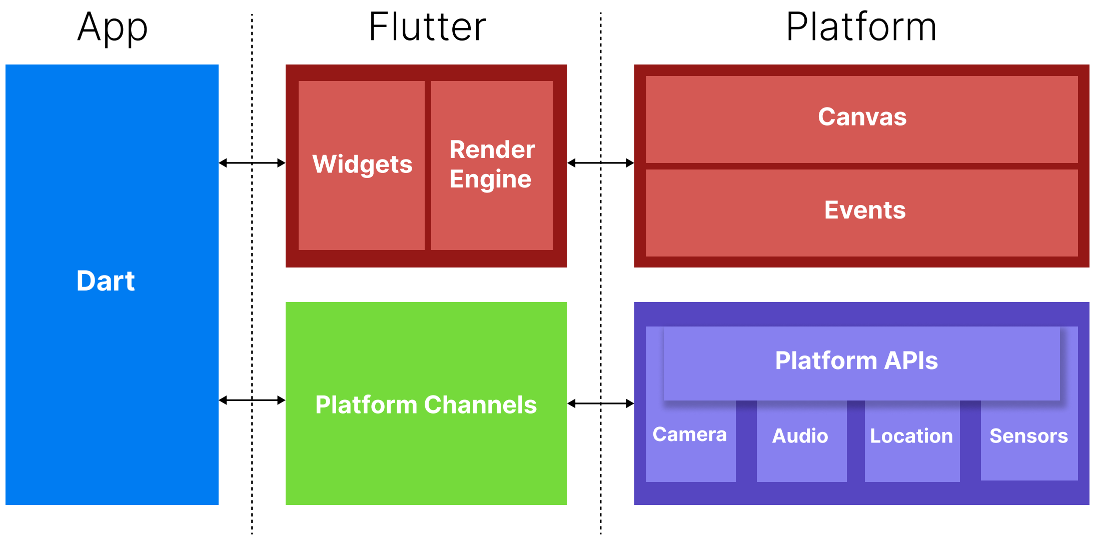

Figure 1.2: Role-Based Access Control: Enforcing Permissions for Users, Doctors, Clinics, and Admins

With a secure authentication and access control system in place, the structure of our database becomes the backbone for managing and organizing user-related data. To illustrate how different types of users and their interactions are handled within the system, we now highlight the most important tables—such as `users`, `clinics`, `doctors`, and `appointments`. These tables represent the core entities and relationships that power the application's functionality.

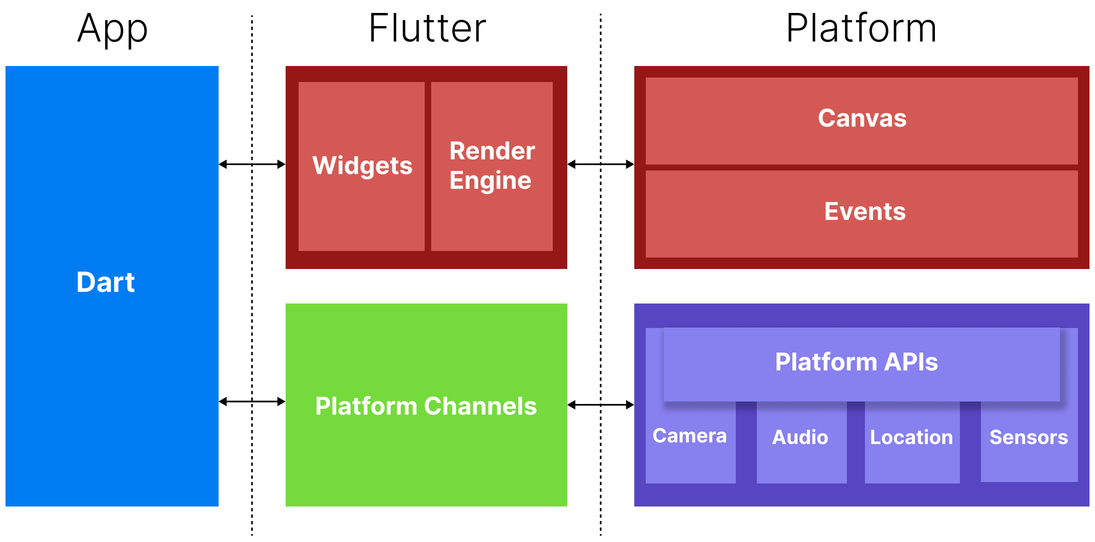

Figure 1.3: Users Table Schema

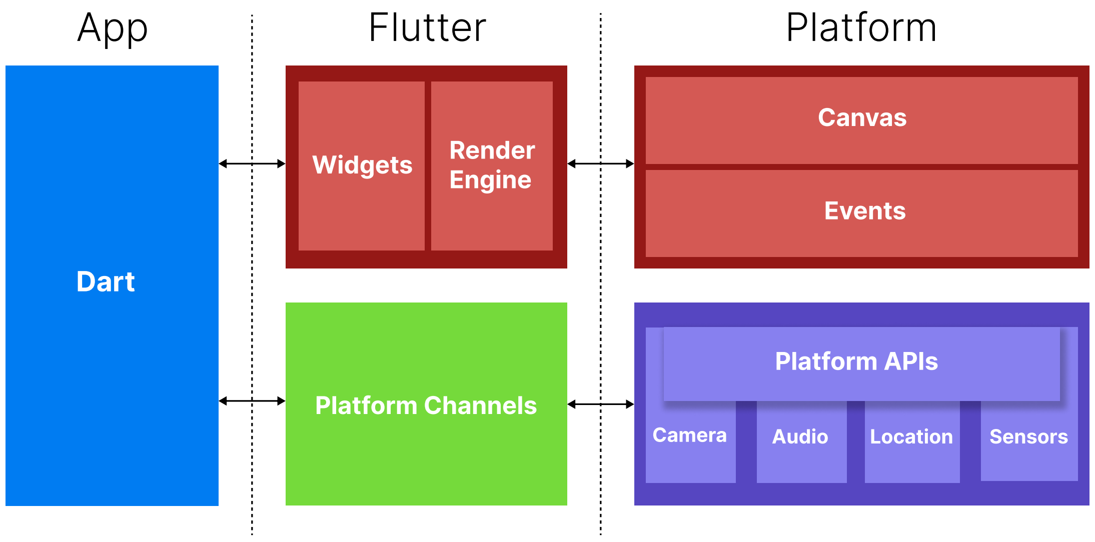

Figure 1.4: Clinics Table Schema

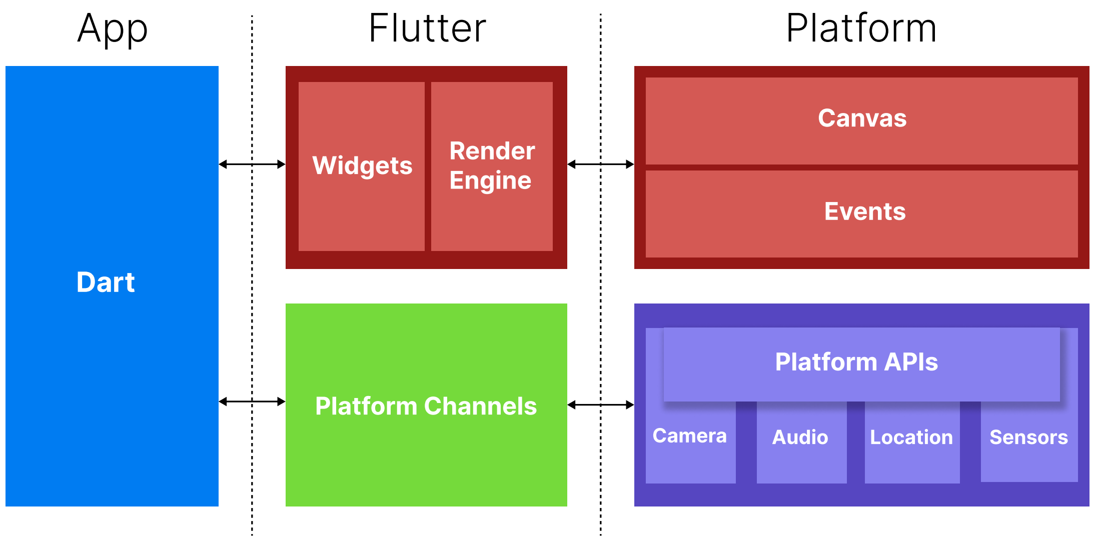

Figure 1.5: Doctors Table Schema

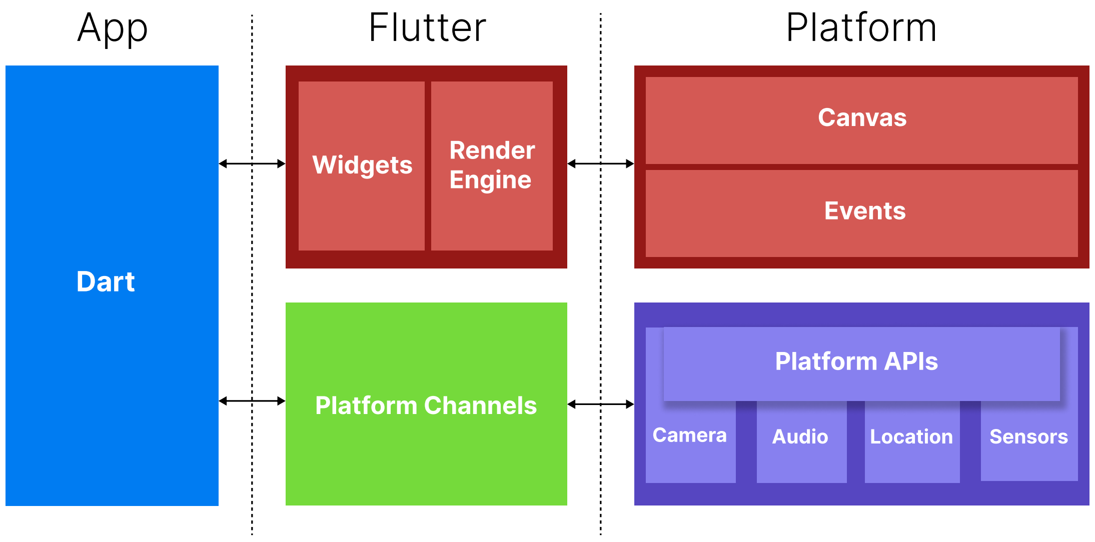

Figure 1.6: Appointments Table Schema

Finally, the complete database structure and its initial data were deployed using the following Artisan command:

Listing 1.1: Laravel command to run migrations and seeders

```
php artisan migrate --seed
```

This command runs all migration files to create the defined schema and immediately executes the seeders to populate the tables with required data. It ensures the database is fully prepared for integration with the rest of the application.

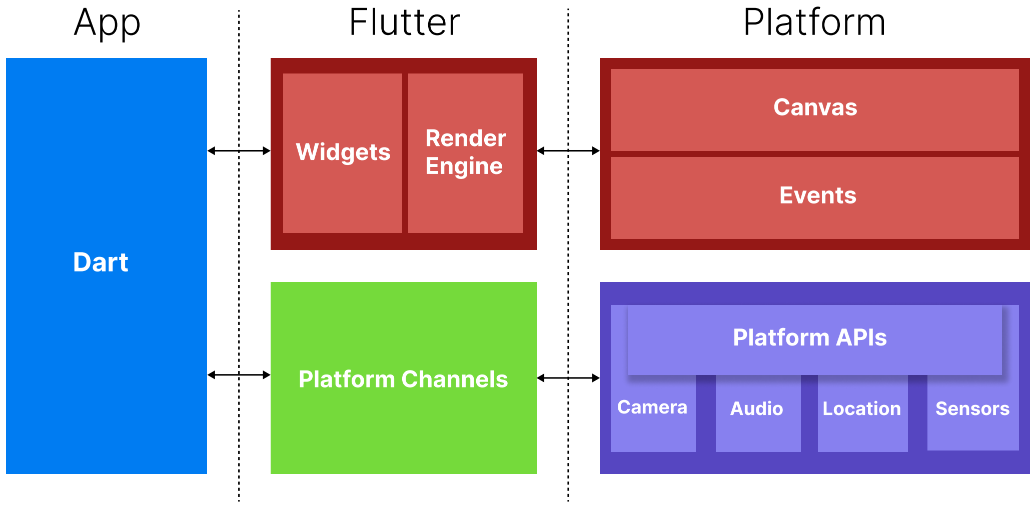

Figure 1.7: Laravel MVC Architecture: Model-View-Controller Structure