**Low-Level Architecture and Data Models**

**P06: Open Source Backend In Rust**

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| **24100305** | **Abdul Wahab** |
| **24100121** | **Faraz Mansur Ahmad** |
| **24100012** | **Muhammad Saad** |
| **24100296** | **Moiz Raza Amir** |
| **24100085** | **Ahmed Mozammil Iqbal** |

**Table of Contents**

[1.](#_heading=h.gjdgxs) Introduction 3

[2.](#_heading=h.30j0zll) System Architecture 4

[2.1 Architecture Diagram—](#_heading=h.1fob9te)As it is in the prototype code 4

[2.2 Architecture Diagram—](#_heading=h.3znysh7)As it should-be 4

[3.](#_heading=h.2et92p0) Data Models 5

[4.](#_heading=h.tyjcwt) Tools and Technologies 6

[5.](#_heading=h.3dy6vkm) Who Did What? 7

[6.](#_heading=h.1t3h5sf) Review checklist 7

# Introduction

To provide context for our project, Backend as a Service (BaaS) solutions essentially abstract away the complexities of REST API such that the developer only needs to create the frontend and use the ready-made BaaS service’s methods to handle the backend. This increases a developer’s productivity as there is no need to write complex backend code as a result. Many known BaaS services come bundled with several available functionalities such as:

● Built-in REST API CRUD operations

● Out-of-the-box authentication

● File Storage

● OAuth Adapters

● Realtime Databases (useful for chat applications)

This makes BaaS solutions attractive for developers. There exist several BaaS services, such as ‘Firebase’ by Google. However, Firebase is closed source and any hosting of the database and other media is done by Google itself which some developers find problematic. Firebase also uses a proprietary data store called “Firestore” which makes data migration a hassle.

As such, there is a growing trend in self-hosting for reasons such as freedom and independence in hosting one’s own services, as well as having the ability to customize applications. Due to the increasing need of customizable services and providing transparency to users, Open Source projects are becoming popular. However, self-hosting open-source BaaS solutions can be tricky as there are several services that need to be configured for them to work securely and efficiently. Most of the existing BaaS solutions provide first-class support for usage as a service. However, they are hosted by the provider, and support for self-hosting in this domain is limited.

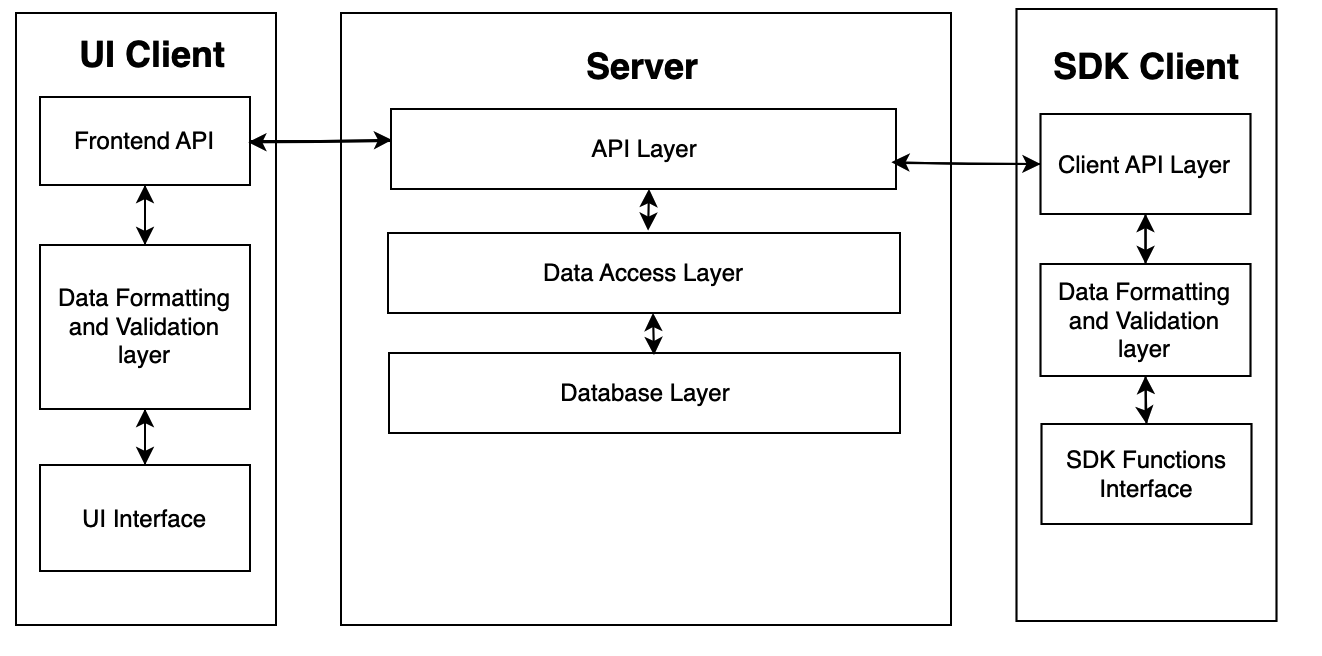
Hence we were motivated to create a lightweight backend similar to Firebase that is open source and can be self-hosted. [Pocketbase](https://pocketbase.io/) and [Supabase](https://supabase.com/) are close relatives of the idea, and are the references that will be used throughout the development of our project. The goal is to create a lightweight and fast backend while providing users well-made documentation and a clean UI to easily navigate our service. There is a high demand for efficient and less storage intensive backend solutions and we are choosing to address this need.

Unlike Pocketbase which uses Go and Supabase which uses Typescript, we will be writing our backend in Rust: a fast, systems programming language that performs orders of magnitudes better than both Go and Typescript in benchmarks. Rust is also known for its robust error handling mechanisms and type-safety that make software safe from expensive errors such as “null pointer exceptions” and allows for a great degree of compiler level optimization. For these reasons we felt that it was appropriate to program our project in Rust.

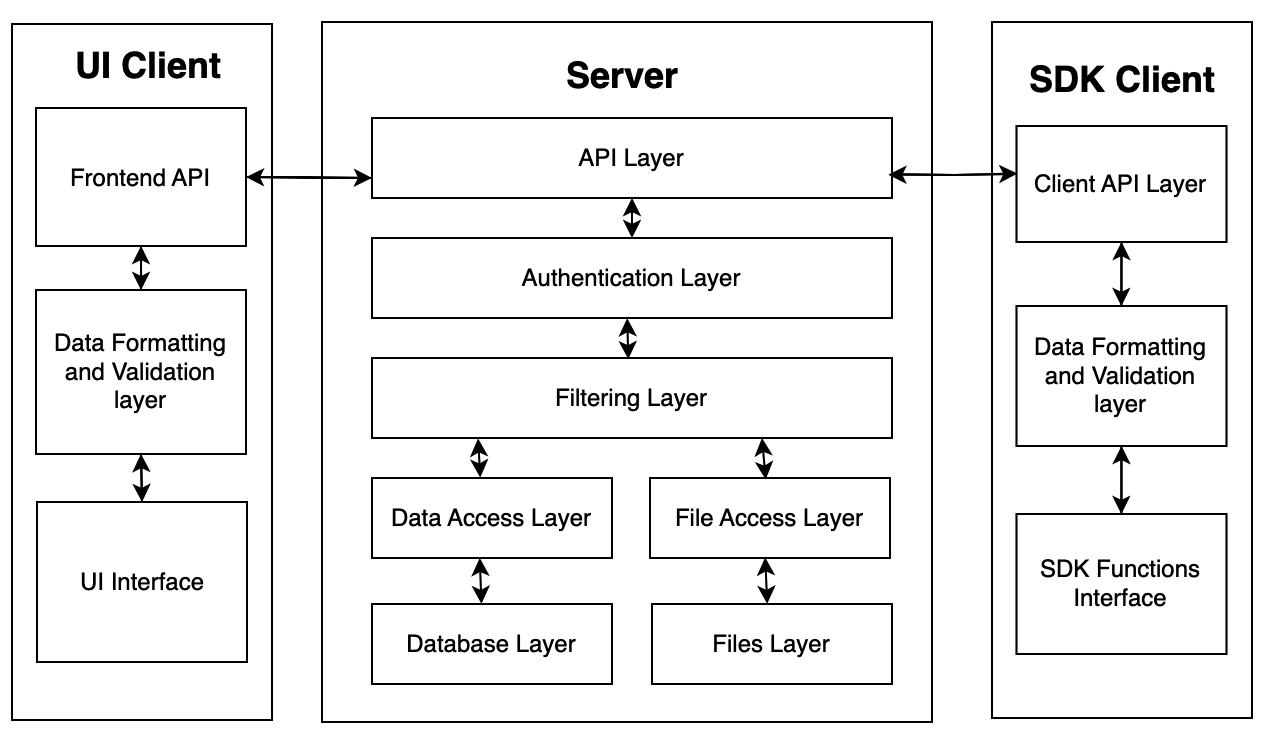
The potential users will mainly be developers. However when developers deploy our service as a backend for their software, System Admins will be able to use our provided User Interface to make any edits.

# System Architecture

## Architecture Diagram—As it is in the prototype code



## Architecture Diagram—As it should-be



**UI Client:** The UI client serves as an administrative side panel crucial for interacting with the server. Its primary functionalities include executing standard CRUD operations, configuring system settings, and managing the creation/viewing of collections.

**Server:** The REST API acts as the backbone of the project, handling essential functionalities such as CRUD operations, authentication, and file management. It encapsulates both user and system data while managing the implementation of user-defined business logic.

* **API Layer:** Responsible for managing incoming requests and routing them to the appropriate endpoints within the server.
* **Authentication Layer:** Ensures secure access to the system and its functionalities.
* **Filtering Layer:** Manages and processes data based on specified filters or conditions.
* **Data Access Layer:** Responsible for interfacing with the database and managing data retrieval and storage.
* **Database Layer:** Handles all interactions with the database, including storing and retrieving structured data.
* **File Access Layer:** Manages interactions related to file handling within the system.
* **Files Layer:** Specifically handles file-related operations and storage within the system.

**SDK:** The SDK comprises various components essential for interfacing with the server's functionalities.

* **Client API:** Provides an interface for the UI client to communicate with the server's API endpoints.
* **Data Formatting:** Handles the transformation and formatting of data to ensure compatibility between the server and client.
* **SDK Functions:** Houses a collection of functions enabling various interactions and operations between the UI client and the server**.**

**Architecture Justification:**

* **Maintainability:** The layered server architecture promotes a modular design, which is highly beneficial for this BaaS system. This modularity aligns with the goal of creating a lightweight and customizable backend. Developers can also work on different components independently, allowing for easier development, testing, and maintenance. This modularity is particularly important for an open-source project, as it enables the community to contribute to specific components without disrupting the entire system. This accomplishes the following non-functional requirements (NFRs) as were described in the Requirements Specification document:
  + The system should be available 99% of the time.
  + The system should not utilize more than 1 GB of memory at any time during its execution.
  + The system's user interface (UI) should have at most 15 pages with direct buttons to perform tasks like CRUD for example. And the UI should require less than one week training for end-users to use proficiently.
* **Reusability:** The architecture promotes reusability through multiple ways. The SDK and UI can be utilized across various projects, allowing developers to easily integrate the BaaS functionalities into their applications without the need to rewrite code. Thus they serve as a reusable and standardized interface, streamlining the development process for different applications. This is because the frontend is based on a JS framework, making the BaaS versatile as it allows for the code to be easily adapted for different applications and development purposes and/or platforms and compatibility with different hardware. Furthermore, the architecture consists of basic sub-layers throughout the layers such Data Validation and Formatting in SDK and UI, Authentication and Filtering in Server, which can be used for any future additions and services.
* **Extensibility: T**he modularity of the architecture supports extensibility by enabling the addition of new features or components to the system. The reason for this is that the current architecture includes the foundations of a BaaS system and then some. For example, if there is a need to introduce additional authentication methods or expand CRUD operations in the future, the admin can extend the server layer without affecting the SDK or UI Client. This flexibility ensures that the system can evolve to meet changing requirements and accommodate new functionalities without major overhauls.
* **Separation of concerns:** The architecture effectively separates concerns between the SDK, Server, and UI Client. Each layer has a well-defined responsibility, reducing interdependencies and making the codebase more understandable and maintainable. The SDK and UI abstract away the complexities of the server, allowing developers to focus on the frontend. The server layer handles API, authentication, and filtering concerns, while the UI Client manages the presentation layer. This separation enhances code readability, facilitates easier testing, and simplifies future modifications.

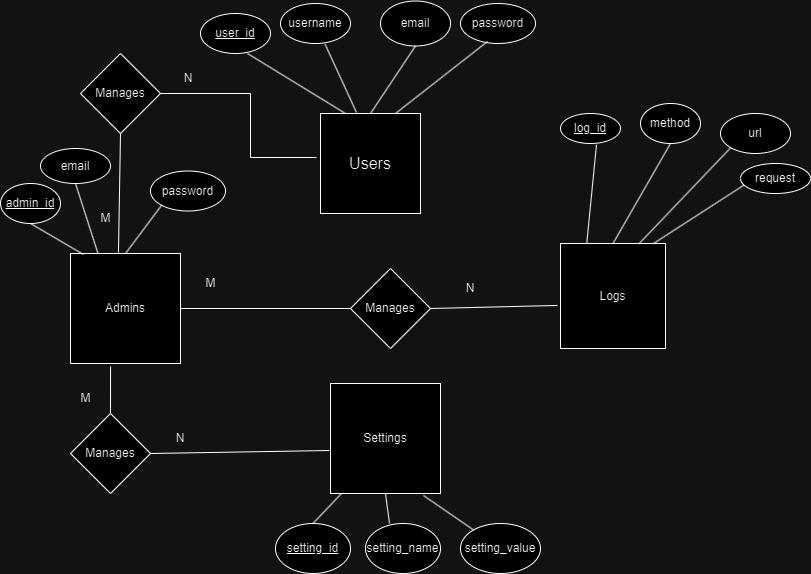
Together, these architectural decisions reinforce and extend the following foundational layers of a BaaS\*:

* **Foundation Layer:** At the foundation layer, the core components of the application are established. This layer forms the basis upon which the entire system is constructed. The architectural decisions for this layer include the choice of data storage mechanisms, and data backup strategies. The justification for this layer's existence lies in the need to establish a robust and reliable infrastructure to support the application's operations.
* **Applications Layer:** This is primarily handling user requests such as logins and various application-specific functions. The architectural decisions in this layer include defining the APIs and endpoints to handle user interactions.
* **Connection Layer:** This layer enables the application servers to access the internet. It involves network communication, data exchange with external services, and managing connections to ensure data flows seamlessly between the application and external resources. The architectural decisions here revolve around network protocols, security mechanisms, and optimizing data transfer.

\*(Reference: https://radixweb.com/blog/backend-as-a-service-baas)

# Data Models

The data model has four entities Admins,Users, Logs and Settings. Additional tables are made by the user and it depends how each developer makes the data model. Admin manages all the models in the server, performs crud operations and changes permissions. Settings model is used to keep track of the general settings set by admin for example SMTP email address and S3 configuration. Logs keep track of all API requests made to the server. Users table is used to authenticate every user interacting or making requests to the server.



# Tools and Technologies

**Rust 1.73:** A memory-safe systems programming language.

**ReactJS 18:** A Javascript DOM manipulation library.

**EJDB 2.0:** An embedded NoSQL database.

**Docusaurus >= 2.0:** A ReactJS Static Site Generator for documentation.

**Javascript ES11:** An interpreted programming language for the web.

**Axum >= 0.6:** A lightweight Rust library for implementing REST APIs.

**Cargo >= 0.75.1:** A package manager and build system for Rust.

**Visual Studio Code 2023:** Source-code editor developed for Windows, Linux and macOS

**PNPM >= 8.11:** A fast and space effecient package manager for the JavaScript programming language**.**

**NPM Registry CLI Version >= 8.0:** The free npm Registry is the center of JavaScript code sharing, and with more than two million packages, the largest software registry in the world**.** This will be where we host our SDK. It is more of a platform than a tool or technology but we have listed it anyway since we’re utilizing the CLI for deployment.

**Git >= 2.4:** Used for project source control.

**Zoom >= 5.0:** Used for collaboration in development as well as for meetings.

**Whatsapp >= 2.0:** Used for development collaboration and discussions.

# Who Did What?

|  |  |
| --- | --- |
| **Name of the Team Member** | **Tasks done** |
| Muhammad Saad Zia | Introduction, Tools and Technologies |
| Ahmed Mozammil Iqbal | Prototype Architecture diagram and Updated Architecture Diagram |
| Moiz Raza Amir | Data Model |
| Abdul Wahab | System component description |
| Faraz Mansur Ahmad | Tools and Technologies |

# Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

|  |  |
| --- | --- |
| **Section** **Title** | **Reviewer Name(s)** |
| Data model | Wahab |
| Architecture Diagram | M.Saad |
| Tools and Technologies | Moiz Raza Amir |
| Introduction | Ahmed Mozammil Iqbal |
| System component description | Faraz Mansur Ahmad |