**System Requirements**

**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. Introduction 2](#_heading=h.gjdgxs)

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

[3. Functional Requirements 5](#_heading=h.1fob9te)

[4. Non-functional Requirements / Quality Attributes 6](#_heading=h.3znysh7)

[5. Security Requirements 7](#_heading=h.tyjcwt)

[6. Who Did What? 10](#_heading=h.3dy6vkm)

[7. Review checklist 10](#_heading=h.1t3h5sf)

# 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 softwares, System Admins will be able to use our provided User Interface to make any edits.

# System Actors

|  |  |
| --- | --- |
| **Actor Name** | **Description** |
| Developers | Developers are the primary users of our BaaS solution. They will interact with the system extensively to create the frontend of their applications and use the BaaS service's methods to handle the backend such as utilizing:   * REST API CRUD operations * Authentication * File Storage * OAuth Adapters * Realtime Databases (optional) |
| Admins | Admins will be responsible for configuring and managing the BaaS system. Hence, their responsibilities, interactions, and system privileges with the system will include, but not limited to:   * User Management * System Configuration & Customization * Security Management * Data Management * Scaling & Resource Management * Backup & Recovery * Troubleshooting & Support |

# Functional Requirements

Below are the functional requirements for the system:

|  |  |
| --- | --- |
| **Requirements** | |
| **Sr#** | **Requirement** |
| 1 | As an admin, I want a clean and usable GUI to interact with the system. |
| 2 | As a developer, I need control over all of my data. |
| 3 | As a developer, I need the system to be easily self-hostable. |
| 4 | As a developer, I need a modular and expansive SDK to use the system. |
| 6 | As a developer, I need excellent documentation to consult when necessary. |
| 7 | As a developer, I need good logging mechanisms to allow analysis and optimization |
| 8 | As a developer, I need cross-platform support to be able to host my solution anywhere, preferably freely. |
| 9 | As a developer, I need a safe and easy to use authentication system. |
| 10 | As a developer, I need several OAuth mechanisms out of the box. |
| 11 | As a developer, I need to be able to create, read, update and delete records in the database. |
| 12 | As a developer, I need to be able to upload files to the system. |
| 13 | As a developer, I need to be able to access and modify all of my uploaded files. |

# Non-functional Requirements / Quality Attributes

Below are the non-functional requirements for the system:

|  |  |
| --- | --- |
| **Sr#** | **Requirements** |
| 1 | The system should not utilize more than 1 GB of memory at any time during its execution. |
| 2 | The system should be available 99% of the time. |
| 3 | The system's user interface (UI) should be intuitive and require minimum training for end-users. |
| 4 | The system should be able to handle concurrent requests from 1000 users. |
| 5 | The system should be able to handle 32000 database writes in 1 second. |
| 6 | The system should be cross-platform, supporting Windows, macOS, and Linux. |

# Security Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr#** | **Security Risks** | **Potential Losses** | **Controls** |
| 1 | Identification and Authentication Failures | Data loss, Sensitive data leak, Loss of user trust leading to financial and business loss, Hostile takeover of systems leading to infrastructure loss | Industry standard Auth protocols like OAuth to ensure a robust authentication mechanism. Thorough validation and testing of authentication logic to prevent breaches.  Secure storage through hashing of highly sensitive data to mitigate damage from leaks. Data like user roles which the system relies on should not be accessible by unauthorized users |
| 2 | Injection | Data leaks, Data loss, Data tampering leading to financial and business loss | All input data should be properly validated and sanitized, validation and sanitization should not be limited to the client side but should be extended to the server side in order to ensure maximum coverage. Parameterized and prepared queries should be used as much as possible. Untrusted user data should be treated with extreme caution |
| 3 | Vulnerable and Outdated Components | Data leaks, Increasing vectors for attack, Loss of user trust, Financial loss and business loss owing to an unreliable system | The system architecture consists of multiple components which have cross component dependencies, these components should be updated as soon as security patches are released and verified. Components with strict version related dependencies to other components may break if they are not updated in lock-step. This issue is particularly prevalent in Javascript libraries and so there should be a process in place to identify external vulnerabilities as well as upgrade, downgrade or even remove possibly damaging components. |

# Who Did What?

|  |  |
| --- | --- |
| **Name of the Team Member** | **Tasks done** |
| Muhammad Saad | Security Requirements |
| Abdul Wahab | Functional Requirements |
| Ahmed Mozammil Iqbal | Introduction |
| Faraz Mansur Ahmad | System Actors |
| Moiz Raza Amir | Non Functional Requirements |

# 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)** |
| Introduction | Saad and Moiz |
| Actors | Wahab |
| Functional Requirements | Faraz and Ahmed |
| Non-functional requirements | Wahab |
| Security Requirements | Faraz |