**Software Process Selection and Project Plan**

**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** |

|  |  |  |
| --- | --- | --- |
| **Content** | **Totals** | **Obtained** |
| Software Process Selection | 25 | 25 |
| Project context analysis | 10 | 9 |
| Gantt chart | 25 | 15 |
| Development environment preparation | 20 | 20 |
| Deployment platform | 10 | 10 |
| Who did what | 3 | 3 |
| Review checklist | 2 | 2 |
| Overall formatting/template | 5 | 5 |
| Late submission penalty | -20 |  |
| **Total** | **100** | **89** |
| Review | 20 |  |
| **Grand Total** |  |  |

Risk Management 14/20

**Table of Contents**

[1. Introduction 2](#_gjdgxs)

[2. Software Process Selection 4](#_30j0zll)

[3. Gantt Chart 7](#_1fob9te)

[4. Potential Risks and Mitigation Strategies 8](#_281zdb4n2jqw)

[5. Development Environment Preparation 9](#_3znysh7)

[6. Deployment Platform 14](#_2et92p0)

[7. Who Did What? 15](#_tyjcwt)

[8. Review checklist 15](#_3dy6vkm)

# 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.

# Software Process Selection

**Agile:**

Pros:

1. Ability to respond rapidly to changing requirements since any change of direction can be discussed in the daily scrum meetings.
2. Teams are not too tightly bound by any stage or phase requirements and can creatively solve problems.
3. Often and more direct contact with the product owner through scrum meetings allowing for concepts, ideas, requirements and feedback to be communicated between the developers and the customer more effectively.
4. A potentially smaller time taken to get a minimum viable product to market owing to the principle of working software being the primary measure of progress.
5. Improved project management since development is divided into smaller goals rather than being monolithic.
6. Agile promotes cross-functional teams that include developers, designers, and stakeholders. Regular meetings and open communication foster teamwork.

Cons:

1. For very large projects that may be mission critical and have rigid requirements, agile may not be the best methodology since cross team coordination and the scaling up of agile practices can be challenging.
2. Additional features and requirements may be continually added to the project, potentially impacting timelines and budgets.
3. Working software is one of the principles of agile, however, for some organizations, it can be a drawback in situations where extensive documentation is necessary for compliance or regulatory reasons.
4. A highly involved customer may end up slowing down the development process if they are not very sure of their own needs or expectations.
5. The fast-paced, iterative nature of Agile can lead to burnout if not managed properly.

**Waterfall:**

Pros:

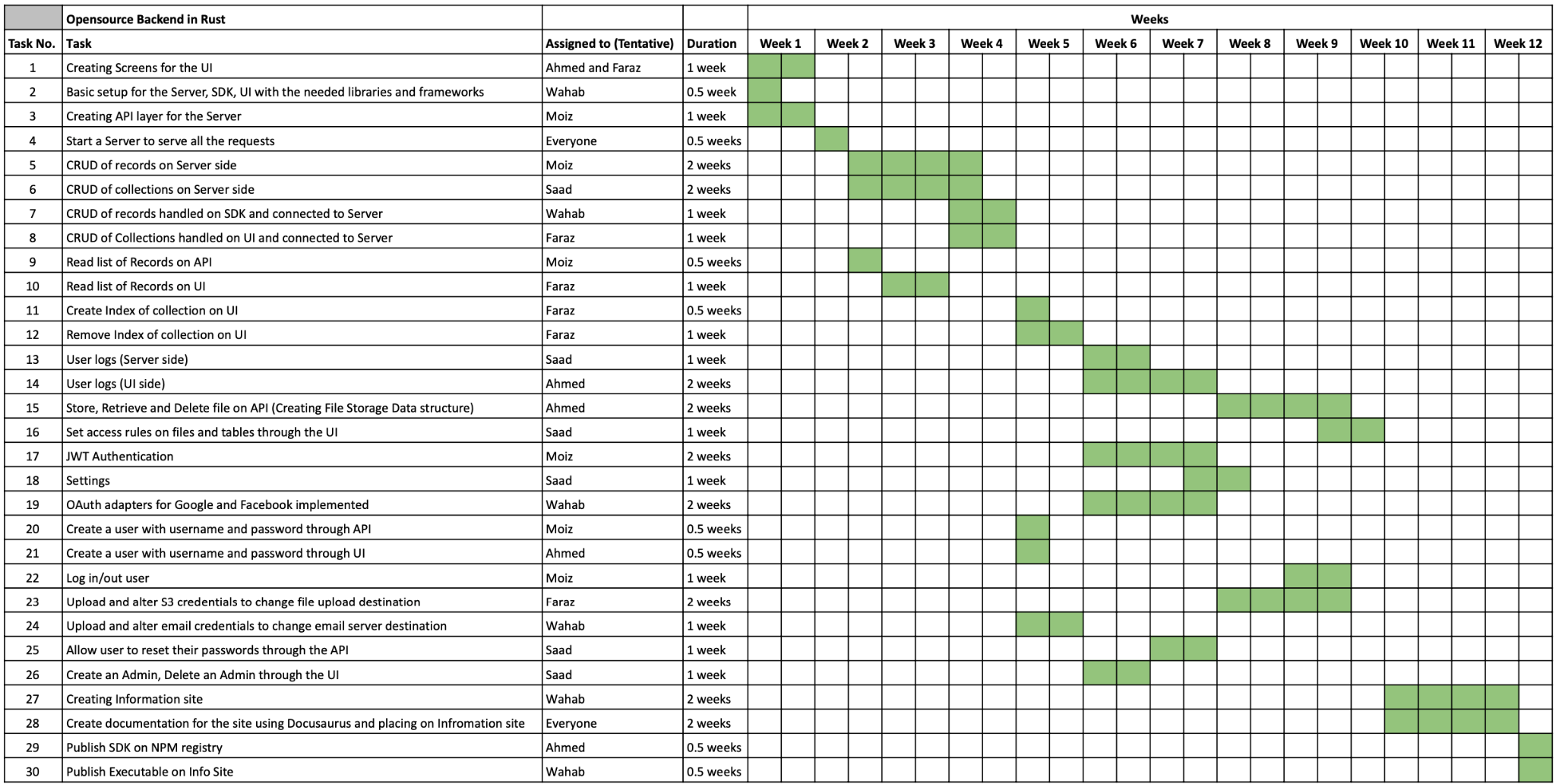
1. A heavy focus in waterfall is on comprehensive documentation, this can lead to a solid understanding of the project's scope and objectives, which allows for minimal scope creep and a predictable project timeline.
2. The Waterfall model has well defined milestones and phases, making it easier to assess progress at different stages of the project.
3. Software releases are less frequent and thus each release can be more carefully vetted and of a higher quality compared to the fast paced releases of agile.
4. A good amount of time being spent in the earlier stages like the requirements stage allows for early detection of potential issues.
5. Mission critical systems greatly benefit from a waterfall development methodology owing to comprehensive documentation and the stability of development.

Cons:

1. Owing to its rigidity, accommodating changes once a project has started may prove to be a time consuming and difficult process.
2. The entire project must be completed before any deliverables are ready for testing or client review resulting in a potentially slower time to market.
3. Producing extensive documentation can be very time consuming and costly, especially for small projects where documentation may outweigh the development effort.
4. The earlier stages of a waterfall project are immensely crucial, any major mistakes or oversights during these stages can lead to critical issues down the line.

We will adopt an Agile development methodology in our project. From a project context analysis point of view, our potential loss due to defects/bugs is low to medium since our system is not mission critical and bugs may not present too big a loss to users [So, any mission critical system cannot use your API]. The developer’s skill levels range from medium to high, the rate of requirement of change is high, the team size is low, the team is highly adaptive to change, there is high pressure to develop prototypes and early releases, business staff’s commitment to work extensively with development team is high, developer’s experience with similar systems is medium and the availability of reusable components is high. All these aforementioned points lead us to follow Agile principles since they fulfill our and the project's requirements most completely.

# Gantt Chart

[Details of final integration and testing are missing]

Milestones

Completion of screens (Week 1)

Completion of CRUD (Week 4)

Completion of Authentication (Week 7)

Completion of Rules (Week 10)

Completion of File Storage (Week 9)

Completion of UI for all required use cases (Week 10)

Completion of SDK for all required use cases (Week 12)

Completion of Documentation and Info Site (Week 11)

Working Executable produced of the rust server (Week 12)

## 4. Potential Risks and Mitigation Strategies

|  |  |  |
| --- | --- | --- |
| **Sr.** | **Risk Description** | **Mitigation Strategy** |
|  | Unauthenticated users might try to access the api endpoints. | Usee industry standard Auth protocols like OAuth to ensure a robust authentication mechanism. |
|  | Access to sensitive user data during data leaks. | Secure storage through hashing of highly sensitive data to mitigate damage from leaks. |
|  | Injection attacks which can cause 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. |
| 4. | Authenticated users might try to access the data for which they are unauthorized | Use robust filtering layer allows role based authorisation giving the user fine grained access control on who has access to some data. |
| 5. | Storing user data might bring about legal and regulatory challenges, especially if data crosses borders. | Users are responsible for their own data as the server is self-hosted. [Any standard to be followed for cross border data access]? GDPR? |
| 6. | Third-party libraries or dependencies used might contain vulnerabilities. | Use well-audited libraries which regularly receive security updates. |
| 7. | As user demand grows, the backend might struggle to cope with scalability issues. | The architecture to be horizontally scalable from the outset. All the bottlenecks are identified and addressed during stress tests. |
| 8. | Loss of user data due to server failures or other issues. | Allow the user to easily create a backup for all the data stored in the database. |
| 9. | Users might incorrectly configure the backend leading to vulnerabilities. [should the users allowed create vulnerability in the backed? or even admin?] | Provide clear documentation, to guide users through the configuration process. |
| 10. | Without proper logging, malicious activities might go unnoticed. | Implement comprehensive logging and monitoring so users can clearly see all the requests being made to their server. |

# 

# 5. Development Environment Preparation

The following technologies will be used to develop the prototype:

**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.

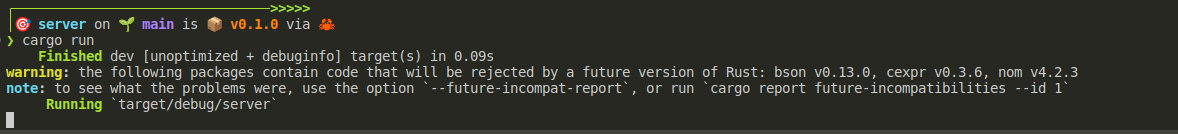
**Javascript ES11:** An interpreted programming language for the web.

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

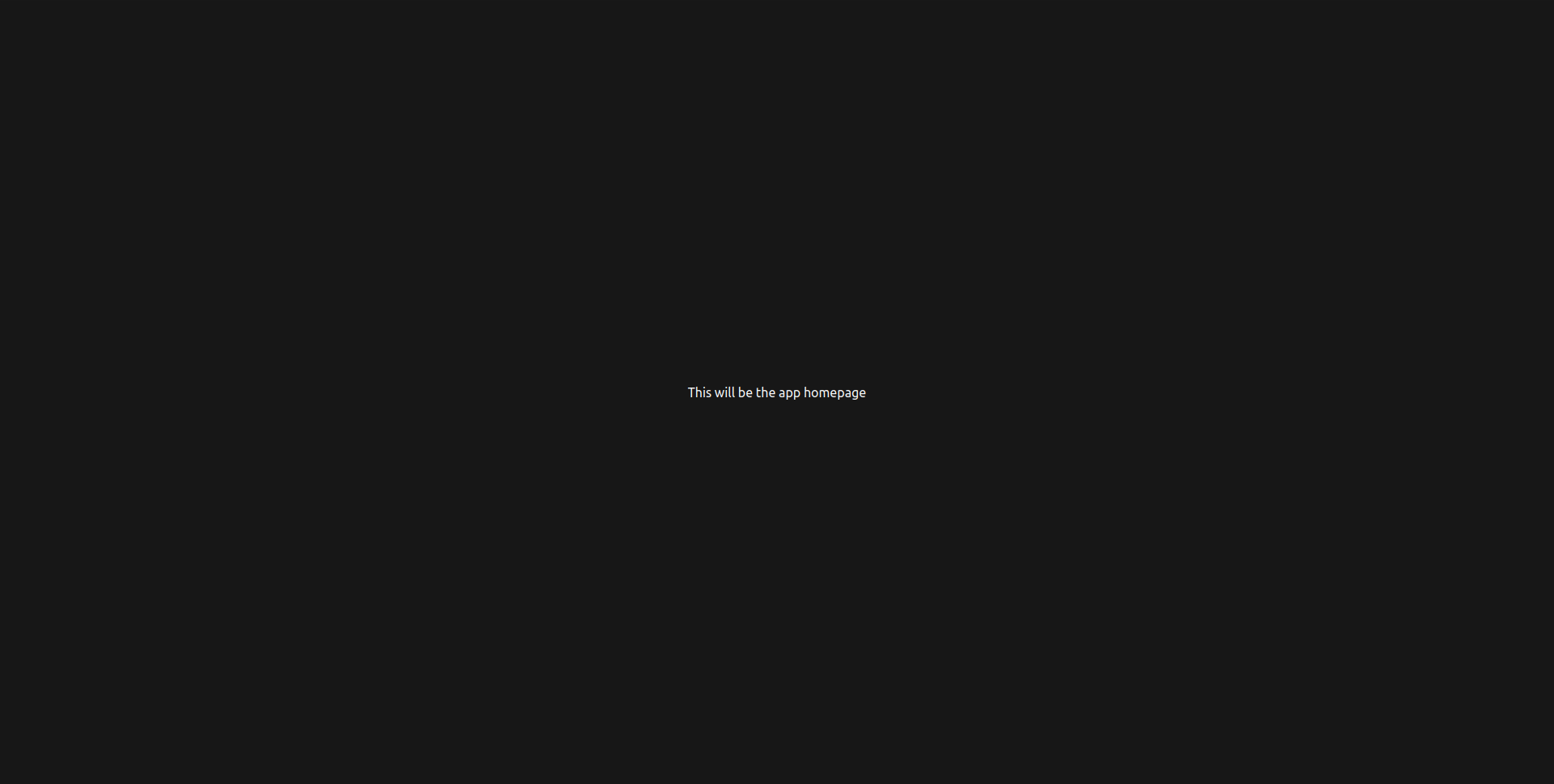
Additional technologies may be added as needed.

**Wahab Environments:**

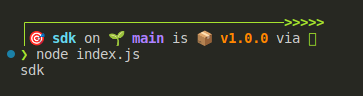
1. **Server**



1. **Admin UI**

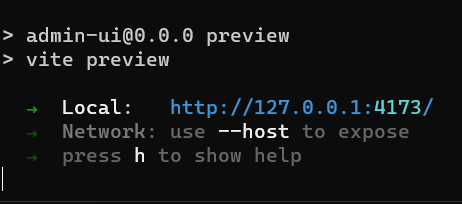
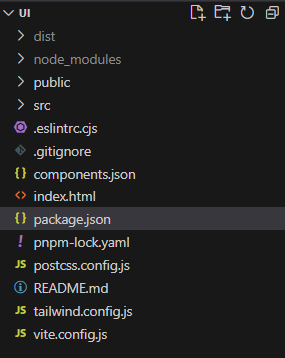
****

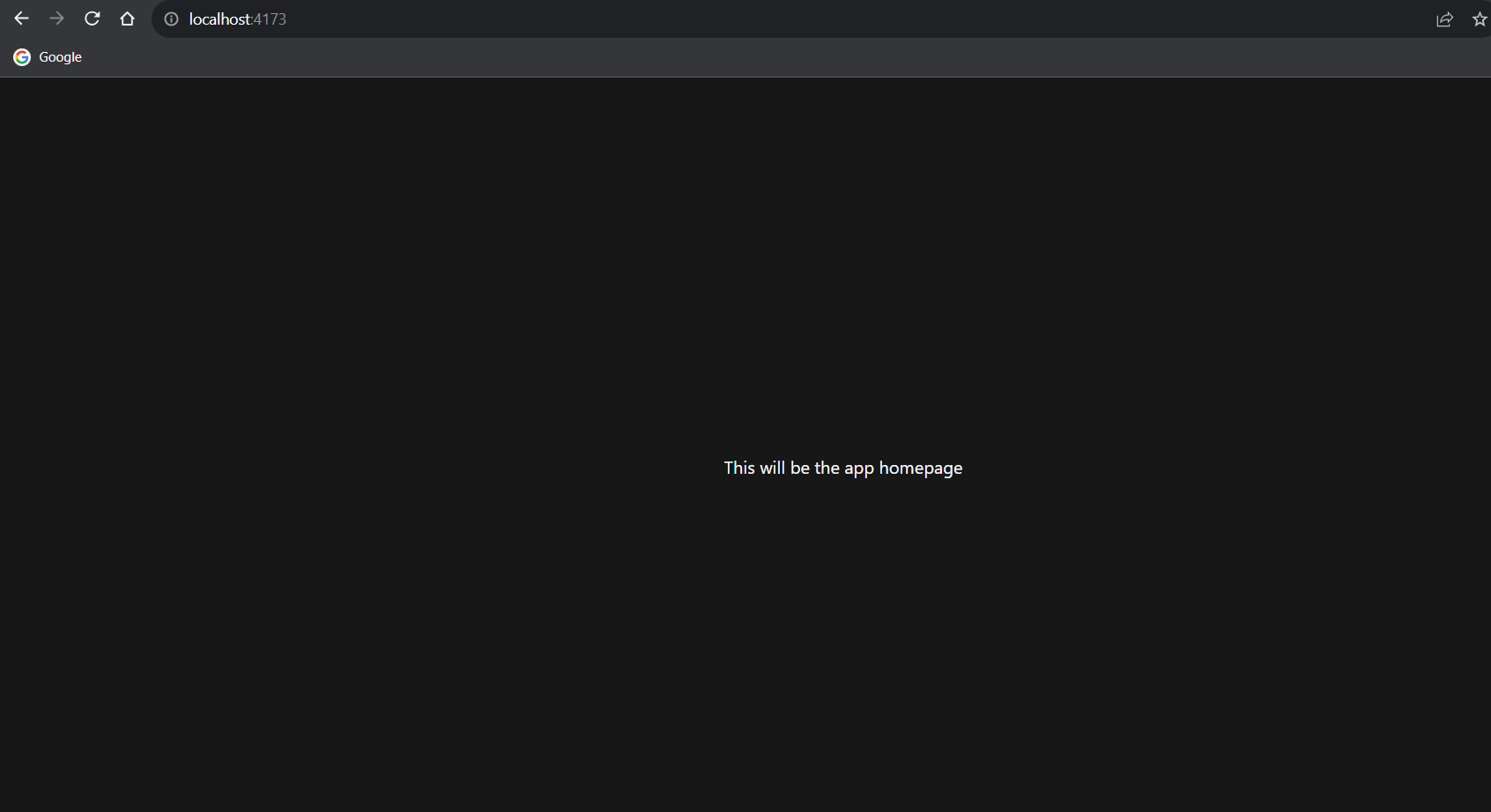
1. **SDK**

****

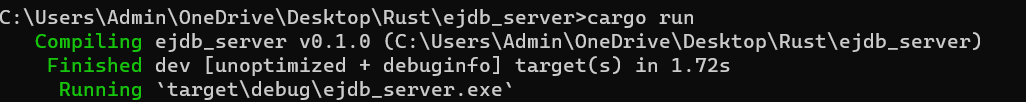
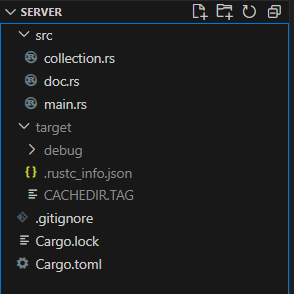
**M.Saad Environments:**

1. **Admin UI**

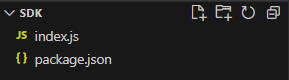




1. **Server**

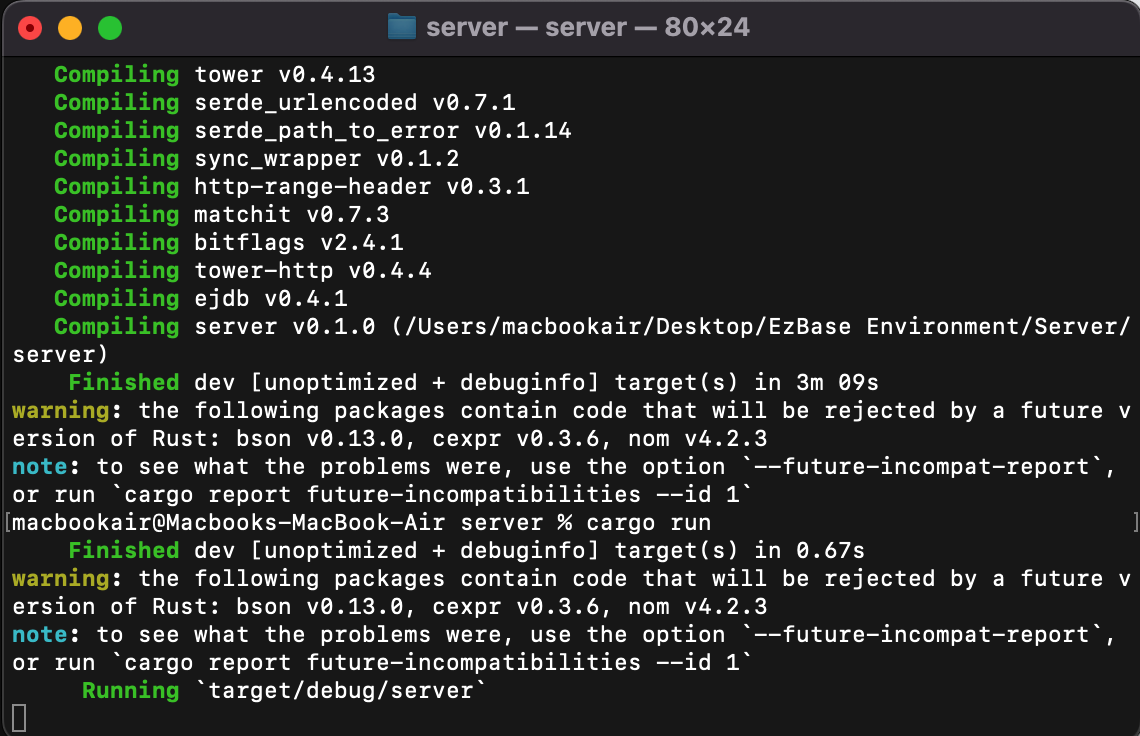
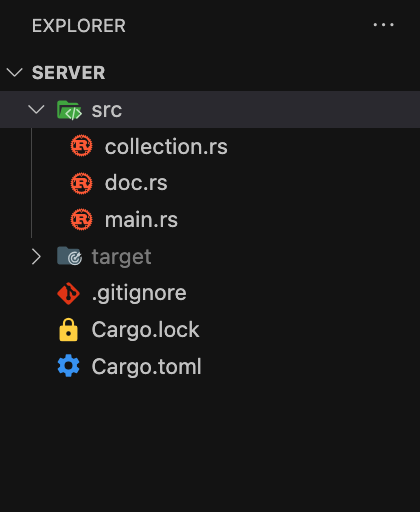


1. **SDK**

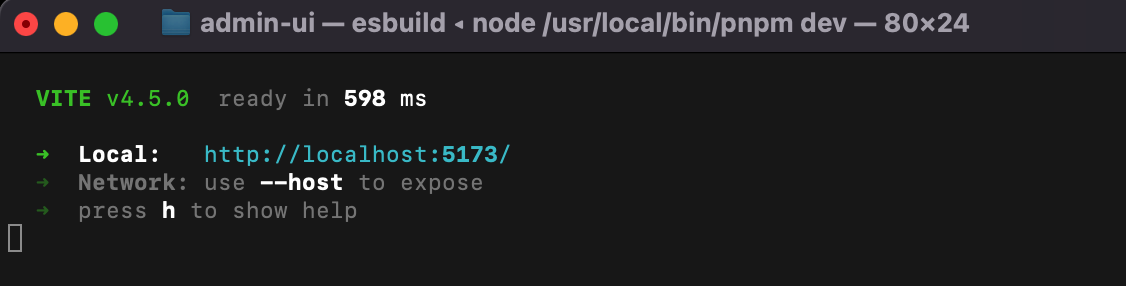
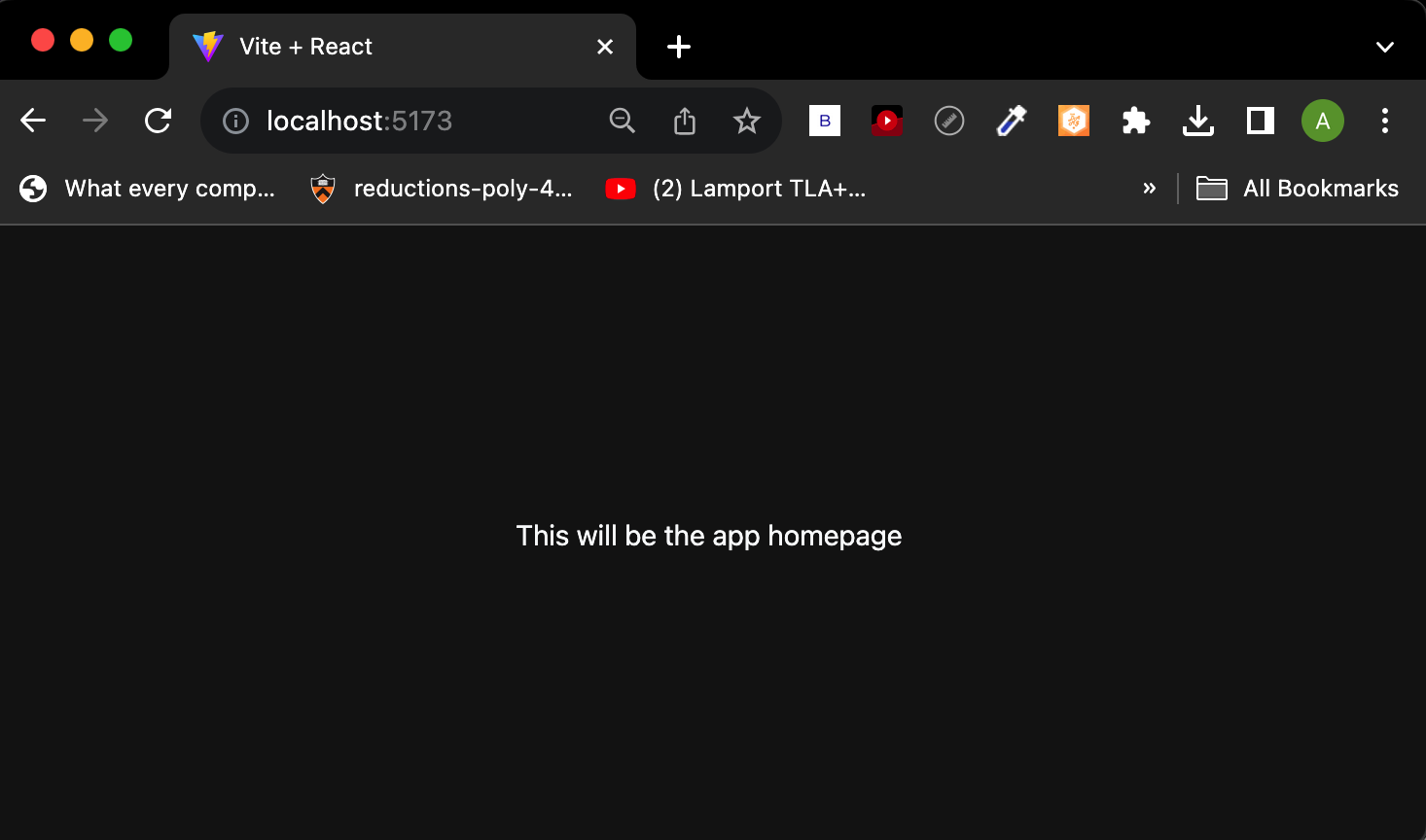
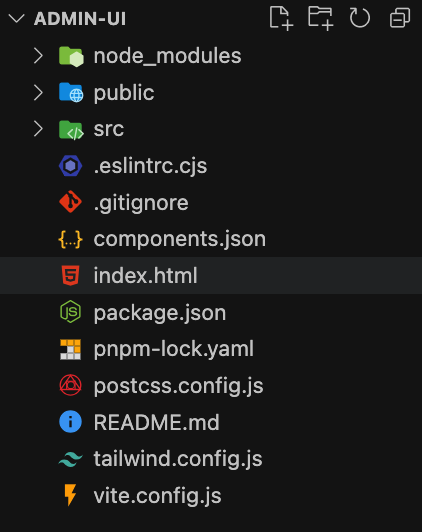


**Ahmed Mozammil Iqbal Environments:**

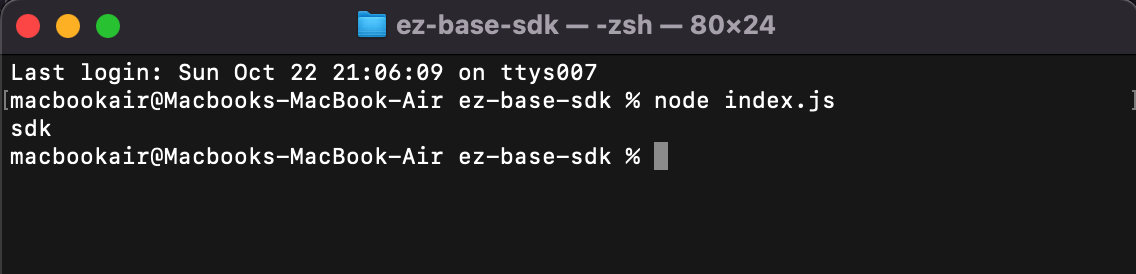
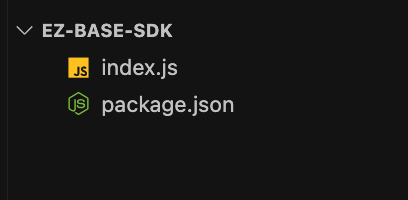
1. **Server**

****

1. **Admin-ui**

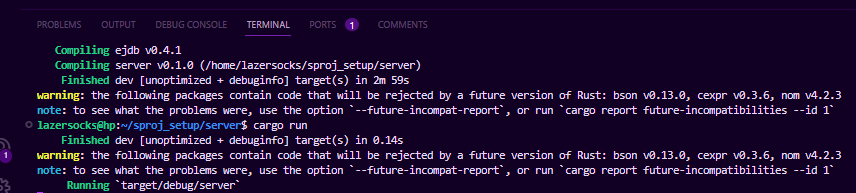
****

1. **SDK**

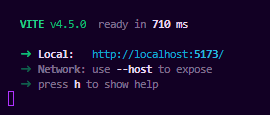


**Moiz Raza Environments:**

1. **Server**

****

1. **Admin ui**

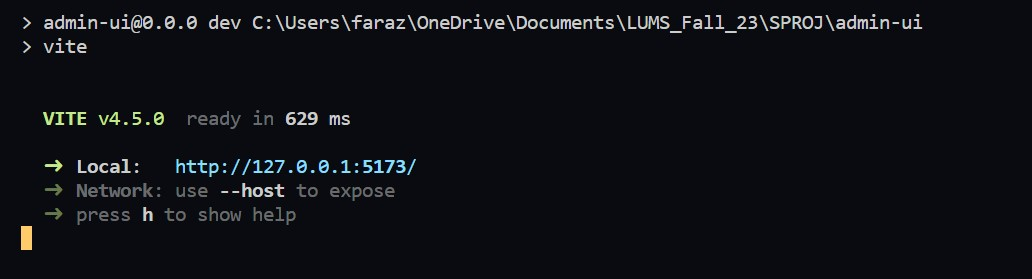
****

1. **Sdk**

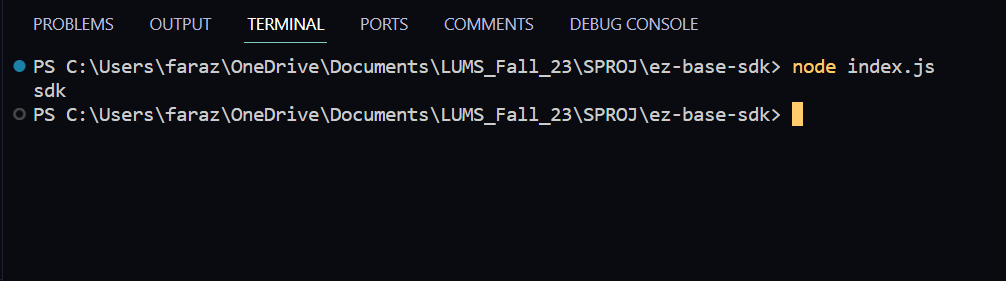
****

**Faraz Mansur Ahmad Environment:**

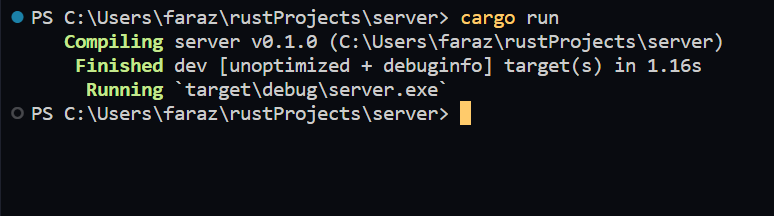
1. **Admin UI:**

****

1. **SDK:**

****

1. **Server:**

****

# 6. Deployment Platform

We will use **docusauraus**  to deploy our Documentation website.

The website for the application itself will be deployed on **vercel.**

Clients can choose to deploy their BaaS client owned server on any deployment tool.

# 7. Who Did What?

|  |  |
| --- | --- |
| **Name of the Team Member** | **Tasks done** |
| Saad | Section 1, Section 2 |
| Ahmed | Section 3 |
| Faraz | Section 6 |
| Moiz | Section 4 |
| Wahab | Section 5 |

# 8. 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)** |
| Section 3 | M.Saad |
| Section 6 | Ahmed Mozammil Iqbal |
| Section 1 | Wahab |
| Section 2 | Moiz |
| Section 5 | Faraz |