**High level Architecture**

**P11 : Blood donating system**

**<team member names & ids>**

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| 22100082 | Muhammad Saad |
| 22100221 | Ahmad Aslam |
| 22100224 | Muhammad Bilal |

**Table of Contents**

1. Introduction 3
2. System Architecture 4
   1. Architecture Diagram 4
   2. Architecture Description 6
   3. Justification of the Architecture 7
3. Risk Management 9
   1. Potential Risks and Mitigation Strategies 9
4. Tools and Technologies 11
5. Hardware Requirements 13
6. Who Did What? 14
7. Review checklist 14

# Introduction

Blood Donating System is an Android, iOS and web-based app that provides a platform to blood donors and receivers for successful blood donation. It is going to make blood donation accessible to the one in need. The potential users of the app are blood donors, NGOs and blood recipients.

The overall objectives of the app are:

* Expedite the process of blood donation.
* Encourage NGOs to provide blood donors to the system.
* Through a system of ratings, points and rewards, encourage more and more donors to use the app.
* Provide an easy communication between blood donors and blood receivers.
* Make blood donation easily accessible to the one in need.

In the app, a recipient is going to ask for a donation of blood. The request will be pushed to a newsfeed where a donor will willingly respond to the request. A chat module will open between the donor and the receiver where they can communicate regarding the blood donation. On successful blood donation, the donor will be rewarded with points.

The app is going to be android, iOS and web-based. The main technologies that we are going to use for our app are Ruby on Rails, React and React Native.

# System Architecture

## Architecture Diagram

# Picture 1

## Architecture Description

**Presentation Layer:**

Presentation layer is the top most layer of our layered architectural pattern. In our case, it consists of our blood donation web/app user interface (UI). So, our Users (receiver, donor, NGO, admin) will directly interact or communicate with this layer. But user interactions will be handled by business and persistence layer. For example, in case of, donor wants to connect to NGO, it will interact with our UI, and this interaction will be forwarded to Business layer to logically deal with donor. Afterwards, business layer will interact with persistence layer in order to read/write data to database.

We will implement this layer in our project using react/react native.

**Business Layer:**

Business layer is the second layer of our layered architectural pattern. In our case, it contains some basic functionalities such as feedback management and donor management etc.

We implement the logic of system in this layer. For example, if donor wants to join NGO, it will be selected on basis of its health reports. So, logic of donor selection will be implemented in this layer.

**Persistence Layer:**

Persistence layer is the third layer of our layered architectural pattern. In our case, it contains user repositories and data access objects etc. This layer will access data from our database using repositories and data access objects and will pass it to business layer above it. For example, if donor wants to view his profile, it will interact with UI, this interaction will be forwarded to business layer and then to persistence layer, persistence layer will then read data of donor from database. Persistence layer will send this read data to business layer and from there it will be sent to presentation layer, where data will be shown to user.

**Database:**

Database is the part of layered architectural pattern, where all user data is stored. Persistence layer access data from database using data access objects and repositories to handle user interactions.

We will use PostgreSQL as our Database for our project.

## Justification of the Architecture

We will use layered architectural pattern in our project, pros and cons of using this architecture in our system are discussed below.

**Pros**:

* **Ease of development:** Every team member will not need full knowledge of project stack. Every member can work on the skillset, it is comfortable with. For example, if some member is good at react native, it can work on presentation layer building front end. Similarly, if a team member is good at backend, it can work on persistence and database layer.
* **Maintainability:** since layers are separated by their functionalities. So, our code will be modular and changing code will be easy and would not impact other layers.
* **Ease of testing:** we can easily test each layer by making mock components at each layer.
* **Reusable code:** we will be able to reuse code with in same layer. For example, we can reuse react components within same layer.

**Cons:**

* **Performance issues:** Since each request needs to go through multiple layers in order to fulfill user requirement. So, Performance issues may arise, if application gets complicated.
* We may be **unable to achieve clean separation of layers** as system get complicated, which is drawback of layered architecture.
* May be Harder for our team to ensure **consistency** with layered architecture.

**Justification:**

1. We will be using layered architecture in our project, since there is possibility that some members may not have full knowledge and expertise in project stack, which can affect the overall progress of our project. As layered architecture separates the layers i.e., presentation layer, business layer and persistence layer. So, members can work on the layers they are comfortable with depending upon their skillset. For example, if some team member is good at backend, he can work on persistence and database layer. Also, team member can work in different layers, if they are comfortable with overall project stack.
2. We can check the **progress of our project**, by looking at how much has been implemented in each layer.
3. As in layered architecture, code will be modular and maintainable in each layer, so making changes to any layer of code will not be an issue and will not impact other layers and overall system. For example, if we want to implement full functionality, we will start by implementing smaller functions and then merging them would not be an issue since code is **modular and maintainable**. Also, we can add new features without compromising existing ones.
4. When we merge smaller functions into bigger functionalities, there are chances **of performance issues**. But as code will be modular in layered architecture, so we can make changes to smaller functions to reduce overhead, so that we do not run into performance issues when we merge them later on.
5. As layered architecture have separate database, so we can make database secure using different built in **security measures** such as middleware and SQL injection measure to handle security of system.

# Risk Management

## Potential Risks and Mitigation Strategies

|  |  |  |
| --- | --- | --- |
| **Sr.** | **Risk Description** | **Mitigation Strategy** |
|  | **Servers Unavailability risk:**  In case of high traffic, servers can become unavailable and user can get delayed response. | In such situation, owner will be suggested to move to high performing servers to handle heavy traffic, and make system available all time. |
|  | **Operational Risk:**  In case of an issue in working application, it can crash. | Team will identify problem and will work on solving it. It will make application work again within time constraint. |
|  | **Reusable component risk:**  There is risk of faults in reusable components of application, which can make certain application features unavailable. | Our team will identify the fault in component. It will work on resolving it and making it functional for future. |
|  | **Team members Unavailability Risk:**  There is possibility of team members unavailability, that can disturb the normal flow of activities done by team. | Available team will reorganize the tasks, and will do missing work. |
|  | **Requirement Change Risk:**  Change in requirement can happen in future, depending upon customer need. | Team will be notified of change in requirements. It will re-define time constraint for updated requirements.  Also, owner will be notified of a cost change. |
|  | **Team Skill Risk:**  In case of team skillset, there are chances that some team members will not be familiar with tool and technologies project requires. | Team will hold tutorials to get familiarize all team members with the tools and technologies project requires. |
|  | **Security Risk:**  There is risk of data breaching of application users. | Team will identify security issues and will suggest owner to move to more secure system. |
|  | **Performance Risk:**  There is the risk of low performance of application. | In such situation, team will make amendments in tools and technologies it used, for high performance of application. |
|  | **Database risk:**  There is risk of low database performance i.e., it cannot perform expected number of transactions. | In such situation, team will identify the problem and owner will be suggested to move to high performing database. |
|  | **System Feature Risk:**  There is risk that some system features stop working due to unexpected programmatic issue. | Team will get feedback from end users and will resolve the issues. |

# Tools and Technologies

**Tech stack:** React Js, React Native, Ruby on Rails, PostgreSQL

**React Native:**

For **frontend** we will be using React Native. React native is an open source UI software framework created by Facebook. It allows developers to develop applications for Android, iOs and web. Difference between react Js and react native is that, react Js is library and react native is a framework which use native components to develop responsive mobile applications.

Advantages of using react native for frontend development are:

* + It is really famous and backed by a very large community which increases the availability of solutions we might face while development.
  + Its uses Modular and intuitive architecture which allows developers to create independent reusable component. This approach contributes in clean and reusable code, enhanced flexibility and saving time and money.
  + It allows developers to create an app with cross platform support.

**Version:** we will be using latest react native version 0.66

**Ruby on Rails:**

For **backend** we will be using Ruby on Rails. Ruby on Rails or Rails is a server-side web application framework written in Ruby. Ruby is highly used for the social app development.

Advantages of using react native for frontend development are:

* + Ruby on Rails is a 100% free and open-source framework. From a developer’s point of view, it is easy to use, and with the add on advantage of tons of gems, the developer can save plenty of time and effort.
  + The framework is by default, installed and enabled with some security measures. When you use Ruby in Rails, you are following a secure development process.
  + It allows developers to make changes easily and quickly. Since, we will be using agile methodology incorporation of changes is really important for us.

**Version:** we will be using current stable version  3.0. 2.

**PostgreSQL:**

We will be using PostgreSQL as **Database.** PostgreSQL is a powerful, open source object-relational database system. It is really mature database which provides following advantages:

* + High speed
  + Secure
  + reliability

**Version:** we will be using current version 13.3.

**Styling and Responsiveness:**

To enhance the usability of the app, responsiveness is really important. Bootstrap is CSS framework which provides grid layout for easier incorporation of responsiveness in the app. For better styling and responsiveness we will be using Bootstrap 5 along with HTML5 and CSS(2.1) .

**Design:**

For designing the screens and prototypes we will be using **Figma** version 102.9.

**Testing:**

For testing the backend API’s we will be using **Postman** (8.

**Coding Environment:**

We will be using **vsCode**(v.1.61) as Integrated Development Environment.

**Version Control:**

To track progress and version control, we will be using **GitHub.**

**Task Management:**

We will be using **Trello** for task management.

# Hardware Requirements

**Development Machines:**

Mobile app development using react native is an intensive task for development machines because react native development require the use of memory heavy tools like Xcode and Android Studio. On the other hand backend development with ruby on rails does not require a powerful machine. Following are the basic hardware requirement for the development machine.

* + 8 GB RAM
  + 2,8 Hz 4-core Core i5 processor
  + 256 SSD
  + 1920 x 1080 panel

**Deployment Machines:**

The app will be deployed on a server with minimum specifications of:

* + 64gb disk space
  + 8gb RAM
  + Supporting 64 bit operating system.

# Who Did What?

|  |  |
| --- | --- |
| **Name of the Team Member** | **Tasks done** |
| Saad Azam | Hardware requirements, risk management |
| Ahmad Aslam | Introduction, Architecture diagram, Architecture description |
| Muhammad Bilal | Architecture justification, 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)** |
| Architecture description, Architecture diagram, Architecture justification. | Saad Azam |
| Risk management Architecture diagram, Architecture justification. | Muhammad Bilal |
| Hardware requirements, Architecture diagram, Architecture description. | Ahmad Aslam |