National University
of computer and emerging sciences

Software Requirement and Design Specifications
Terrabuzz

Realized by
Saif Ul Islam (18K 0307)
Muhammad Hassan Zahid (18K 0208)
Tashik Moin Sheikh (18K 0142)

As part of the course
Software Design And Analysis - CS324

Work presented to
Ma'am Javeria Farooq

From the Department of Computer Science
Faculty of Computer And Emerging Sciences
National University Of Computing And Emerging Sciences
Date Of Delivery,

Contents

1 Introduction

1.1 Purpose Of Document

This report is generated as part of the *"Software Design And Analysis - CS324"* course at FAST NUCES, and is related to the SRS and SDS report as accompanied by the semester project.

The Software Requirement Specification (*SRS*) is meant to serve as a guide that details the exact nature, software, hardware, and the business requirements that lead to the development of a software project, in this case, this is for *Terrabuzz*. The guide provides an overview of the thought process, the nature of development, the workflow, the main conceiving ideas and driving force and list of business fulfilling points that the product tried to fill in the gaps for.

The Software Design Specification (*SDS*) entails the system architecture and the system design kept it mind while developing the product. How different components are interleaved with one another, and what dependencies exist, if any.

1.2 Intended Audience

This document is for anyone who would like to know about the decisions the developers took from a design and analysis perspective, and for future engineers who wish to understand the nature of the project, and how it became what it did, with emphasis on what ideas and what features. This report will talk about those lower level details, as well as the higher business level use cases that are part of the product.

1.3 Definition Of Terms, Acronyms And Abbreviations

This section provides a brief explanation to some of the key words used in the documentation. The format is "Topic - Description", the topic is highlighted as bold for clarity and for the sake of easy reading,

- **Term** - Description
- **API** - Application Programming Interface, a way of abstracting away the underlying logic of a component

- **Containerization** - Creating virtual environments which run with specific software packaged whenever they are 'started' or 'closed', meant to serve as isolated pieces of programs that deal with their own local state, instead of polluting the global system resources.

- **DevOps** - Development And Operations, when new changes pushed to code trigger events that either deal with the code itself, or use that code to automate the deployment process

1.4 Document Convention

The font is T1, Calligra, and the font size is 11pt for reading purposes.

2 Overall System Specification

2.1 Project Background

Terrabuzz was built around the prospect of using the idea of Social Media Applicatons to see what was lacking in the current market place. There are some ideas that we want to right now or in the future take Terrabuzz to, and that may mean is that we focus on things that differentiate it from the lies of Facebook, and Linkedin. While Facebook and Linkedin primarily focus on 'gaming' more and more reactions and comments, Terrabuzz wishes to emphasize on the quality of relations among the people, finding newer people that are similar to their interests, rather than just having tons of people whom you can't even relate to.

To address this challenge, we came up with the idea of *interests*, which help to group by your set of personal interests, along with the the interests of others, and see where both lie. That is, if you have a interest in #xyz, then the system designed can help you find similar people who have #xyz in their biography, or who post about #xyz, that's really the main idea and driving force behind this.

2.2 Project Scope

[This section will give an overview of project scope. This of project and will mention project boundaries and main functionalities that will be addressed in the system. Describe what the system will and will not do]

The system is limited obviously in ways that modern social media applications are not limited to. While building this project, we wanted to build something along the lines of a MVP, a *Minimum Viable Product*.

The main functionalities, however, are,

1. The ability to register
2. The ability to log in
3. The ability to send a forget password request
4. The ability to change your password via the forget password email generated
5. The ability to see a newsfeed
6. The ability to search for other users by their name
7. The ability to search for other users based on some interest in their profile
8. The ability to search for other users based o some interest mentioned in their posts

9. The ability to visit your own profile
10. The ability to visit the profile of others, and add them to your connection
11. The ability to visit your page of notifications
12. The ability to visit your settings page
13. The ability to change your email, your username, or your user handle, and your password
14. The ability to log out
15. The ability to visit the individual posts, including your own
16. The ability to like a post
17. The ability to add a comment to a post
18. The ability to publish a post
19. The ability to change your links of external social media portfolio
20. The ability to change your main profile information, such as your biography, your interests, your activities
21. The ability to choose your profile picture the first time you log in

Some of these contrasts are as follows,

1. This is a web application, not a mobile application.
2. Terrabuzz can help you 'search' for similar people, but not give you suggestions at the moment - as it has no suggestion system.
3. The system lets you choose from a predefined sets of images for the profile pictures, but offers you no option to upload pictures of your own.
4. The system has no progress in the sphere of privacy and advanced settings such as this, and is broken into two parts for displaying posts. One is the home page, where the posts of all users on the website are displayed, and one is /feed, where a logged in user sees only those posts by people he or she has *connected* to.
5. the application has not been built keeping in mind responsive design, though it is something we might to do later on

6. The publish page only gives you the capability to add textual content, and has no space for multimedia related features at the moment, unlike the most popular social media applications.

2.3 Not In Scope

This section will highlight/explicitly mention the functionalities (if any) that are not in the scope of current project. It does not mean that we will not want to work on these features in the future if we ever want to, just that this specific release (v1.1.0) did not keep in the following list of scope related ideas when in the implementation phase.

The below list is a collection of technical topics, and does not brush on the marketing and business aspects of the project,

1. We did not keep in mind the email verification aspect since the start of the project, and only very late did we realize this. It definitely exists as a module in the code itself, but needs to be tested with integration tests to make sure everything is working fine as expected.

2. You cannot delete posts or your profile. We have performed the CR part for posts, and CRU for profile, but we wanted to focus on the main feature aspects of the project - implementing the D part for both is just a matter of time, and can be done with some focus, because it becomes obvious that having finished the above list of details, we would be able to implement the *Delete* part as well in the future. Something that highlights and differentiates a project from others, we wanted to work on that aspects first and foremost, and even though we had this in our list of todos, the semester burden and the effort of carrying so much workload all together was admit-ably a bit too much for us.

3. Only two aspects of *Settings* work for the moment. We have to consider the other options as well, such as profile privacy, settings, advanced, how notifications are delivered, etc.

4. No indicator that a person has got new notifications. A user has to manually check if new notifications have arrived.

5. No tutorial guide for users who sign up to the application, like many online web applications, even apart from social media applications do

6. Interactions and sign up with other platforms - what if someone wants to connect their Terrabuzz account with Gmail or Facebook?

7. No way to see exactly the list of people who 'liked' your post

2.4 Project Objectives

The main objectives for this project were,

- Help people find better connections among their community, relating to whatever topic the

have their interests in

- This can help communities locally in the real world find quicker connections to people who face the same problem as them, and see what their perspective on that is, so people can easily collaborate on more things

- In a business environment, Terrabuzz can really be worked on give a platform that helps people collaborate on ideas, and as a much more advanced form of an online forum with more rooms for innovations, community involvement, engagement. The idea falls again with the connection network, graphs, and building features keeping in mind what will be useful and valuable to the end user, instead of just expanding itself as an infinite list of posts, such as Facebook and LinkedIn.

- Given a proper professional room for development and more diverse and talented team, Terrabuzz can definitely become something that can be put into production on the market level, even going as far as to incorporate good business models around it. It should be kept it mind that some social media applications simply did not have a business model to begin with. Twitter, WhatsApp, Facebook, had money invested in them before they even got their complete business model in practice. The idea was that if a social media came too directly and explicitly to the ad features and ideas, they would lose customers very faster. Social media applications are built around the idea of dominoes - if you lose a small percentage of your customers, the other customers start to leave as well.

- A much more feature rich editor to suit most people, from the professional environment, to the regular teenager user, to your local businesses. Terrabuzz wants to incorporate an editor that gives people much, much, more freedom to post content, and are not limited to poor text editors, like that in Facebook, which just has blocks of content. A caption with an optional picture. Those posts have no way of 'story telling', and are same for any group of users.

2.5 Stakeholders

From a business side of perspective, the following list of people are going to be the end users of the application. It can be a bad idea to expand too much of your global audience, and the other extreme of a bad idea is limiting to a very niche group of people - until and unless you really are building an application that addresses the need for those people, for example, a mobile application by McDonald's to help people with their McDonald's orders. It's too limited to a certain group of people, who's success is directly dependent on the performance of McDonald's to focus on it.

Some business stakeholders are,

- Mostly a younger generation of people who would liked to see what is going on their favorite communities with what kind of posts. This has broad perspectives for the different ranges of ages. A younger audience is interested in sharing jokes and small posts about their topics, a slightly more mature age is interested in engaging about discussions for their favorite posts

- A business suit of professionals who wish to expand their audience and find related people. While LinkedIn is more suited for this space and audience, LinkedIn offers little help when it comes to businesses who wish to broaden their network of people and grow with needed related audiences.

- Online influences, who wish to write, blog, post, share their opinions and ideas about the world. The editor envisioned for Terrabuzz might help them express their opinions more clearly.

- The developers who have some experience with the Agile method, and the MERN stack, with some basic database intuition about the project.

2.6 Operating Environment

Not much will be needed to operate the system. A regular and day to day computer with enough bandwidth to connect to the internet as with any application is just fine. The software itself operates on your regular Chrome, Mozilla, Edge browser, which exist on all operating systems and sometimes come pre packed with many new machines.

Thus, there is no,

- Specific version of any OS required - as all current Operating Systems support internet browsers

- A small enough bandwidth to help you connect to the internet is enough. Nothing too demanding

Terrabuzz only depends on the NPM package manager to setup the binaries for itself. A complete list of binaries will be mentioned later in this documentation.

2.7 System Constraints

2.7.1 Software Constraints

The software has not yet been tested to support multiple concurrent users at a single time, because it has not been deployed properly on a cloud platform, but for which a pipeline exists at the moment. Certain testing needs to be done to support this feature and aspect.

The software needs to deal with more upfront security issues, such as, checking if the email given even exists in the respective domain, thinking about how to deal with people who misuse the system, how to prevent spam and toxic users from jumping onto the platform, basically dealing with the community side of things, and making the perspective clear on how to maintain a healthy and growing community of users who don't abuse or exploit others - an entirely different social media pandemic and needs to be addressed

2.7.2 Hardware Constraints

The application is not supported on mobile at the moment, and only works as intended on desktop systems. Other than that, any decent laptop with a decent bandwidth is enough to run the web

application.

2.7.3 Cultural Constraints

It's not valuable to the users if we show them posts from a completely different part of the world. The application will not be interesting if the posts that the user sees does not even belong to their local area, because the average users wishes to relate, which is the hook that keeps them coming back to the platform. In the absence of such a mechanism, a cultural constraint would be to find and fetch relevant information for the user - that is, finding the right profiles, posts, and interests to keep the user on the platform.

2.7.4 Legal Constraints

Security, privacy, and data mining is a very huge legal challenge. If we put something like this in production, and Terrabuzz grows bigger, we seriously would have to answer what data we keep in our system, why we do that, and have a privacy policy ready for us to go for. We also have to answer if our data goes to any third party vendor, if it does, where is it stored, how secure it is, how easy it would be for a user to wipe their information for the application, and if the data collection we have and the recommendation system, if we introduce one to solve the problem in *Cultural Constraints*, if that system discriminates or prioritizes people, or if our system prevents people from getting the right exposure they should get for their posts.

2.7.5 Environment Constraints

There are no such environment constraints that can be thought of at the moment

2.7.6 User Constraints

The system wishes to target people starting in the age group from 12 to above. At the moment with release of v1.1.0, there is mostly an emphasis on textual interaction, and not so much with graphical interaction.

2.7.7 Off The Shelf Components

Off the shelf components effect the development process in how the project is actually structured and developed. There is a strict procedure about how to add a frontend interface to the application, and as well a procedure for how to add something to the backend side of things for that frontend interface. The file structure is designed and defined to contain a set and fixed idea of how to look at that project.

From the perspective of security, the application uses JWT, *JSON Web Tokens*, instead of using cookies on the server side, so it falls on the client to store the related session keys.

CORS is disabled on the application, that is, websites and web applications outside of the domain of the application itself cannot request web pages and resources from Terrabuzz.

Google API is used for sending emails to Gmail inboxes, and thus indirectly depends on it to send emails with the respective credentials.

The password is hashed, *encrypted* and *decrypted* to *Bcrypt*,

2.8 Assumptions & Dependencies

The list of assumptions at the moment are,

- A module for verifying the email address exists, but we did not manage the time to integrate this into the actual flow, so we're assuming that the email entered and given to us actually belongs to a user

The list of dependencies is split into three aspects. At the root of the project, we have dependencies needed to hold the project together. We then have a client side of dependencies, and a server side of dependencies.

2.8.1 Root Level

- **Eslint** - A linter that checks if all of the code is following a certain standard for styling. Follows the airbnb standard
- **Husky** - checks before any git commits, if all of the code edited is following the standards defined by Eslint. If not, it prevents the commit entirely.
- **Jest** - a JS testing library, useful for *Test Driven Development*

2.8.2 Client Level

- **ReactJS** - The frontend library we decided to go with for this project.

2.8.3 Server Level

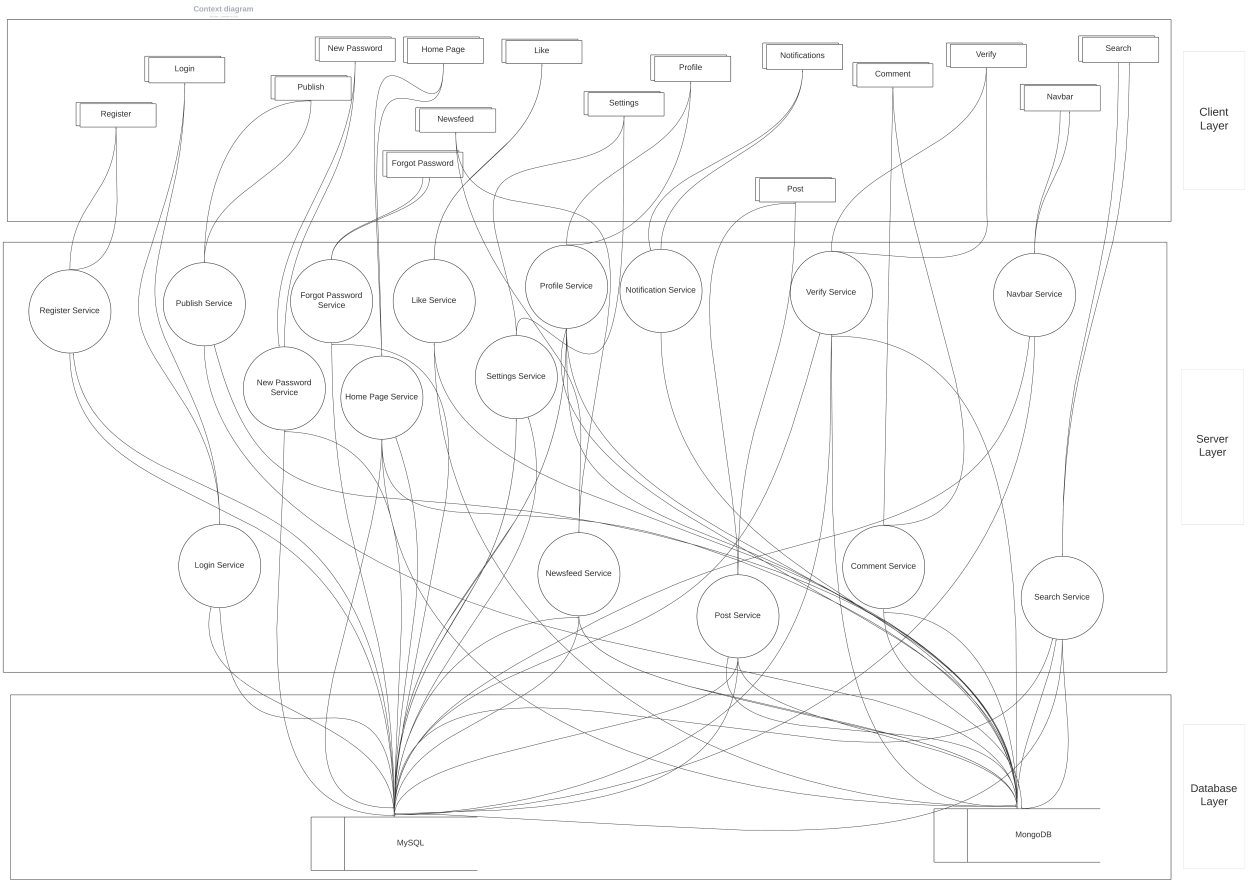
- **Bcrypt** - Need to hash, decry-pt, and encrypt passwords into the database
- **Cookie Parser** - For parsing cookies on the client level
- **CORS** - To prevent cross origin JavaScript files running on the Terrabuzz domain
- **Dotenv** - For incorporating the required security keys into the files
- **Express** - JS framework for dealing with NodeJS on a higher level with more functionalities
- **Googleapis** - List of google related services for sending emails to the respective gmail inbox

- **HTTP** - Express module for transferring data to HTTP modules with their status codes and JSON messages
- **Lodash** - to prevent a security flaw introduced by a regular expression used by the search bar
- **Mongoose** - A ORM, Object Relational Mapper, easier to work with MongoDB.
- **MySQL2** - A connector to connect to the My-SQL container to run our queries on top off
- **Neo4j-Driver** - A connector to connect to the Neo4j container to run *cipher* queries with
- **Nodemailer** - NodeJS npm module, used with googleapis, to send the emails via SMTP protocol
- **Redis** - A connector to connect to the Redis container to store and fetch cached data.

3 External Interface Requirement

[This section is intended to specify any requirements that ensure that the new system will connect properly to external components. Place a context diagram showing the external interfaces at a high level of abstraction.]

There are no external components to this project. Most of the components are internally a part of the software, and which is coded out, without the need for more external devices and machines. The application is based on a Graphical User Interface for the user to interact with by the use of buttons, fields, and graphical annotations to guide the user.



3.1 Hardware Interfaces

[Describe the characteristics of each interface between the software and hardware components of the system. This description might include the supported device types, the nature of the data and control interactions between the software and the hardware.]

The supported device types are desktops, mobile, and web applications, even though the styling

and the order of the content might be messed up in all except desktop, it is completely possible to use the project outside of the desktop domain. Testing for the user interface needs to be done.

3.2 Software Interfaces

[Describe the connections between this system and other external software components (identified by name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify and describe the purpose of the data items or messages exchanged among the software components. Describe the services needed and the nature of the inter-component communications. Identify data that will be shared across software components.]

The system depends on a list of external packages, listed and updated with the use of the NPM package manager. The technologies used can be classified into three different components depending on the level of their usage. For example,

- Frontend
- Backend
- Deployment

The front-end is connected to the back-end via a list of functions described as the functional components that abstract away the HTTP protocols required to pass data back and from the back-end. The back-end system uses those HTTP protocols and uses them to route to the appropriate service defined for it. Each functionality has an existing controller that takes care of the API to connect, receive, and send data to the backend. This effect takes place whenever a page loads to retrieve the context of the information. Thus, each call to the backend bundles up and finishes the database work on the server side, to return everything as a JSON back to the client side to store it in relevant states of the information to give the needed results.

To address the *'Describe the services needed and the nature of the inter-component communications.'*, we have two views offered by the application. This is for the logged in and logged out sessions that a use can see. For example, whenever a user is logged out, he or she is visited by the home page that lists down all the posts that exist on the system, and a person who is logged on sees a box that allows them to make a post, or use the navbar to navigate to the other pages. The posts that this logged in user sees are only from the people that the particular user has 'followed' or 'connected' with.

This is accomplished by the use of tokens on the client side for the purposes of *authentication* and *authorization*.

Now, to answer the *'Identify data that will be shared across software components'*, we can say that most, if not all services and front end side requests, are primarily based on the MySQL and MongoDB database. Each controller service on the server's side will make sure that the user requesting the service is logged in, to maintain a session, and to not let users reach an undesired location - for

example, no logged out user can publish any posts.

3.3 Communications Interfaces

[Describe the requirements associated with any communication functions the system will use, including email, web browser, network communications standards or protocols, electronic forms, and so on. Define any pertinent message formatting. Specify communication security or encryption issues, data transfer rates, and synchronization mechanisms.]

The requirements for sending emails exist such that the user has given and entered an email that definitely exists on the database on Google for their GMail service, however, this is an assumption that no user will ever really want to make tons of fake emails just to be registered. There is no real benefit, except maybe overloading the database with many UPDATE requests. However, a simple rate on the server and the cloud hosted solution for the deployed instance should be enough to check that a user is not making too many requests at any given moment.

Some security issues on the database definitely did arise.

- For the search component, a regular expression is used to get the relevent search request. We to had to use the *lodash* package to escape any sequences that might send our regular expression function into an infinite over drive by an malicious intended bots or hackers wanting to DDOS the web application

- At mulitple times, we had to make sure we sent in sanitized input to the SQL queries we were making, to avoid an SQL injection into our database

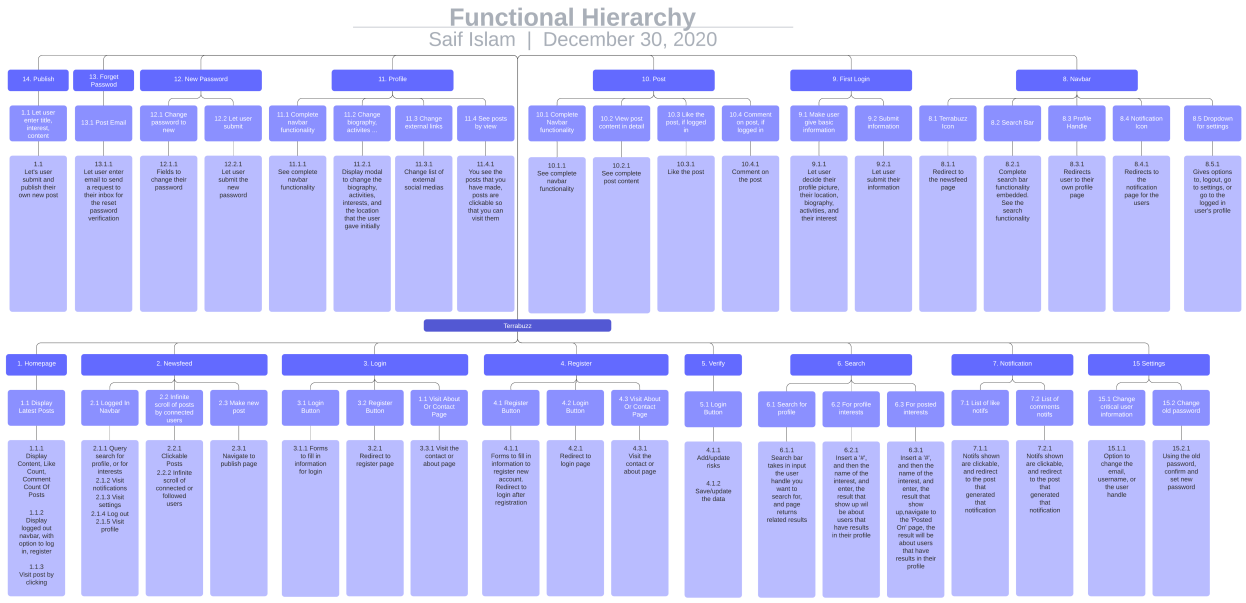
We have not yet considered synchronization mechanisms, because we did not plan or test out how our application would behave if multiple users were signed in together at a single moment.

We noted that some parts of our application were making heavier performance hits to the Relational Database that we had, because we had too many read queries for posts. This would not be great, and it came to us later that we should something like a *Redis* type of database, that we can cache the most trending and related posts, to serve them up quickly.

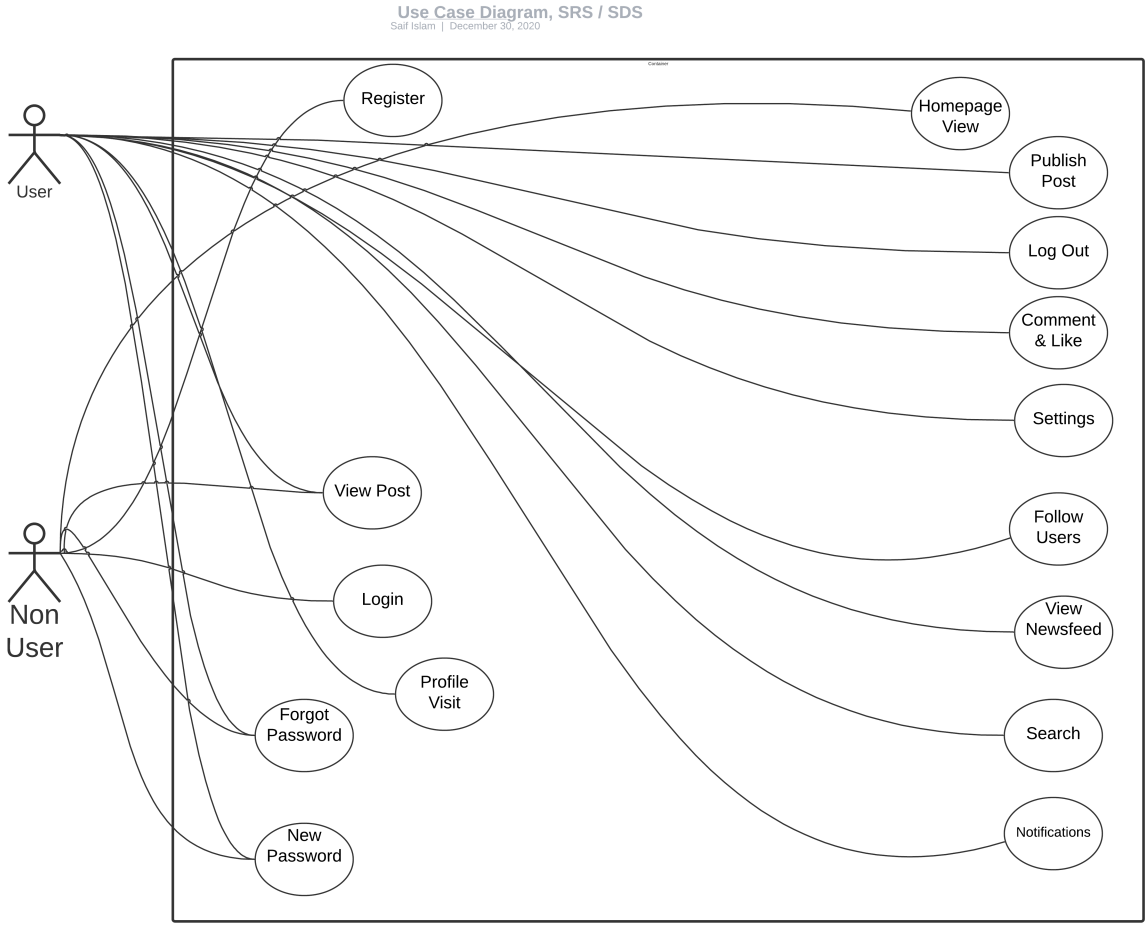
4 Functional Requirements

4.1 Functional Hierarchy

The attached figure lists down the related functional hierarchy as used in the project



4.2 Use Cases


Use Case Diagram, SRS / SDS
Saif Islam | December 30, 2020
User
Non User
Register
View Post
Login
Profile Visit
Forgot Password
New Password
Homepage View
Publish Post
Log Out
Comment & Like
Settings
Follow Users
View Newsfeed
Search
Notifications

4.2.1 Register

Use Case Description to sign up on the website	Provides a non user the functionality
Use Case Name	Register
Primary Actor	Non User
Other Actors	None
Stakeholders	Non User | Developer
Relationships - Includes	Registration Service
Pre-conditions * *	That the user is not logged in / that the email for that account does not already exist
Flow Of Actions 1. Actor filling in the forms 2. submitting via the register button	
Alternative & Exceptional Flows	If the non user gives an email / user handle already used, they are asked to fill the form again
Post-conditions	Login is possible after the register

4.2.2 Login

Use Case Description	Helps the non user login
Use Case Name	Login
Primary Actor	Non user
Other Actors	None
Stakeholders	Non User | Developer
Relationships - Includes	Login Service
Pre-conditions *	There is an email belonging to the filled account
Flow Of Actions	Filling in the password and the email
Alternative & Exceptional Flows 1. Ask non user to enter the fields again	If the login credentials are not correct
Post-conditions 1. If logging in for the first time 2. Else	Log the visit in Redirect to the first login Redirect to the news-feed page

4.2.3 Forgot Password

Use Case Description	Sends a reset email to the email entered
Use Case Name	Forgot Password
Primary Actor	User
Other Actors	None
Stakeholders	Developer | Gmail
Relationships - Includes	Forget password service
Pre-conditions - That the email entered actually belongs to the email	
Flow Of Actions 1. Users enter their email 2. Email is sent to their inbox	
Alternative & Exceptional Flows 1. That the email entered does not belong to any user in the database	
Post-conditions *	That an email is sent to their inbox

4.2.4 New Password

Use Case Description	A new password page with the valid secret link
Use Case Name	New Password
Primary Actor	User
Other Actors	Non-User
Stakeholders	Developer | User
Relationships - Includes	 New Password Service
Pre-conditions *	 That the link used is valid
Flow Of Actions 1.	 User visits the link via the email in their inbox
Alternative & Exceptional Flows 1.	 That the link used is not valid anymore
Post-conditions *	 That the password has been changed

4.2.5 Homepage View

Use Case Description	View of all posts for the logge
Use Case Name	Homepage
Primary Actor	Non user
Other Actors	None
Stakeholders	Developer | Non u
Relationships - Includes	Homepage Servic
Pre-conditions *	That the user is logg
Flow Of Actions 1. None	
Alternative & Exceptional Flows 1. That the user is logged in, if so, redirect to newsfeed [1ex] heightPost-conditions *	None

4.2.6 View Newsfeed

Use Case Description	View of the page when user is logged in
Use Case Name	View Newsfeed
Primary Actor	User
Other Actors	None
Stakeholders	Developer | User
Relationships	
- Includes	Newsfeed service
Pre-conditions	
*	That the user is logged in
Flow Of Actions	
1.	The user infinite scrolls for the posts and profiles they have connected to
Alternative & Exceptional Flows	
1.	That the user uses the navbar functionality
2.	That the user clicks on anyone of the posts
Post-conditions	
*	Visiting any of the routes available on the page

4.2.7 Search

Use Case Description	Search functionality in the navbar
Use Case Name	Search
Primary Actor	User
Other Actors	None
Stakeholders	Developer | User
Relationships	
- Includes	Search by posts
- Includes	Search by interest
- Includes	Search for profile
Pre-conditions	
*	That the user is logged in
Flow Of Actions	
1.	Input is entered
2.	One of two buttons is selected
3.	A search is made by hitting the return button
Alternative & Exceptional Flows	
1.	None
Post-conditions	
*	None

4.2.8 Profile Visit

Use Case Description	Profile view for each user
Use Case Name	Profile Visit
Primary Actor	User | Non-User
Other Actors	None
Stakeholders	Developer | User | Non-User
Relationships - Includes - Includes - Includes	 Get Posts Service Get profile information Edit profile information
Pre-conditions *	 None required
Flow Of Actions 1. Visiting the link 2. If not the same as logged in user 3. Else Show optiont to edit profile information	 Give option to connect to user
Alternative & Exceptional Flows 1.	 None
Post-conditions *	 None

4.2.9 Publish Post

Use Case Description	Publish a personal post
Use Case Name	Publish Post
Primary Actor	User
Other Actors	None
Stakeholders	Developer | User
Relationships - Includes	 Publish Post Service
Pre-conditions *	 User is logged in
Flow Of Actions 1. 2. 3.	 Enter the interest Enter the title Enter the content
Alternative & Exceptional Flows 1.	 All the fields have not been entered - throw an error
Post-conditions *	 That the post is successfully published

4.2.10 Log Out

Use Case Description	Logs logged in user out
Use Case Name	Log Out
Primary Actor	User
Other Actors	None
Stakeholders	Developer | User
Relationships - Includes	Log Out Service
Pre-conditions *	The user is logged in
Flow Of Actions 1. User goes to dropdown menu on the navbar 2. User goes to log out button on the dropdown 3. User clicks on the log out button	
Alternative & Exceptional Flows 1.	None
Post-conditions *	None

4.2.11 Settings

Use Case Description	Give option to the user to change the user details
Use Case Name	Settings
Primary Actor	User
Other Actors	None
Stakeholders	Developer | User
Relationships	
- Includes	Change Password service
- Includes	Change User Information service
Pre-conditions	
*	The user is logged in
Flow Of Actions	
1. User clicks on the settings option	Redirect to the settings page
2. If needed to change profile information	Let user change user information
3. If user needs to change password	Redirect and change to forget password and set new passwor
Alternative & Exceptional Flows	
1.	Go back to the previous options
2.	The user handle or email being changed to already belongs to some
Post-conditions	
*	The information is changed or updated

4.2.12 Notifications

Use Case Description	Display list of comments and likes for the user
Use Case Name	Notifications
Primary Actor	User
Other Actors	None
Stakeholders	Developer | User
Relationships - Includes Likes	Comments
Pre-conditions *	The user is logged in
Flow Of Actions 1. User visits the notification page 2. If User wants to see the relevant post	Sees list of comment and like notifications Send user to post on click which generated the notification
Alternative & Exceptional Flows 1.	None
Post-conditions *	None

4.2.13 Follow Users

Use Case Description	Allows users to follow other users and see their post on Newsfeed
Use Case Name	Follow Users
Primary Actor	User
Other Actors	None
Stakeholders	Developer | User
Relationships - Includes - Excludes	 Profile Newsfeed
Pre-conditions *	 The user is logged in
Flow Of Actions 1. User visits other profile, except their own	 clicks on the connect button
Alternative & Exceptional Flows 1. If the user already follows a user	 Display a cross button instead of the connect button
Post-conditions *	 None

4.2.14 Comment And Like

Use Case Description	Comments And Likes on Posts
Use Case Name	Comment And Like
Primary Actor	User
Other Actors	None
Stakeholders	Developer | Users
Relationships - Includes	Notifications
Pre-conditions *	User is on the post page
Flow Of Actions 1. User enters comment and clicks on 'submit 2. User likes a post	
Alternative & Exceptional Flows 1. 2. * *	 If commented, post a comment notification If liked, send a like notification heightPost-conditions If commented, post a comment notification If liked, send a like notification height

4.2.15 View Post

Use Case Description	Get content for the post
Use Case Name	View Post
Primary Actor	User
Other Actors	None
Stakeholders	Developer | User
Relationships - Excludes	Comments And Likes
Pre-conditions * *	If need to use Comments And Likes, user must be logged in If user is logged out, retrieve the count of Comments And Likes
Flow Of Actions 1. User visits the page for each Post	
Alternative & Exceptional Flows 1.	None
Post-conditions *	None

5 Non-Functional Requirements

5.1 Performance Requirements

For a social media application, the main criteria is the smoothness and delivery of content at real time and maintaining a consistent view against many users about some information. This can include seeing the same number of comments, the same number of likes. The high performance criteria here in this case are,

- Very smooth UX experience and good delivery of content with speed without too much lag
- Reliability in terms of the security measure taken to make sure no privacy credentials are released to the malicious hackers with miss intent.
- High availability of the application is a huge driving force. Being down for a few hours has a huge impact on the consumer base.

These are very important factors to consider, only after we put our project into deployment and production. Unfortunately, we have not gotten the time to do so.

5.2 Safety Requirements

There is room for a group of people to misuse the system to encourage inappropriate action towards communities by igniting discussions that spiral into hate and disgust. These have real impacts on the social lives of people, and so far at this scope, we have not been able to address this issue.

5.3 Security Requirements

Some criteria can be,

- Not storing passwords as simple plain text
- Keeping secrets out of the main repository
- Queries made to the database MUST be sanitized
- No internal logging of the systems should be view-able by a user.
- Root access to the databases should be protected by a a hashed password
- A rate limiter to make sure no user makes too many requests
- In forget password, the hash used to visit the new password page MUST be valid and only very recent

5.4 User Documentation

At the moment, there is no such thing, but we can plan on delivering a tutorial on how to use the platform for new users unfamiliar with the platform to get them started quickly.

SDS

6 System Architecture

6.1 External Links To Diagrams

Some of the diagrams don't exactly fit right, or are to big to view in detail. To view these diagrams outside of this document, visit the links provided below to view a detailed, HD version for each of them for a much more screen-fit resolution,

- Activity Diagram - here
- Class Diagram - here
- Component Level Diagram, Production - here
- Component Level Diagram, Development - here
- Context Diagram - here
- Deployment Diagram, Production - here
- Deployment Diagram, Development - here
- Functional Hierarchy - here
- Sequence Diagram - here
- State Chart Diagram - here
- Use Case Diagram - here

6.2 System Design Philosophy

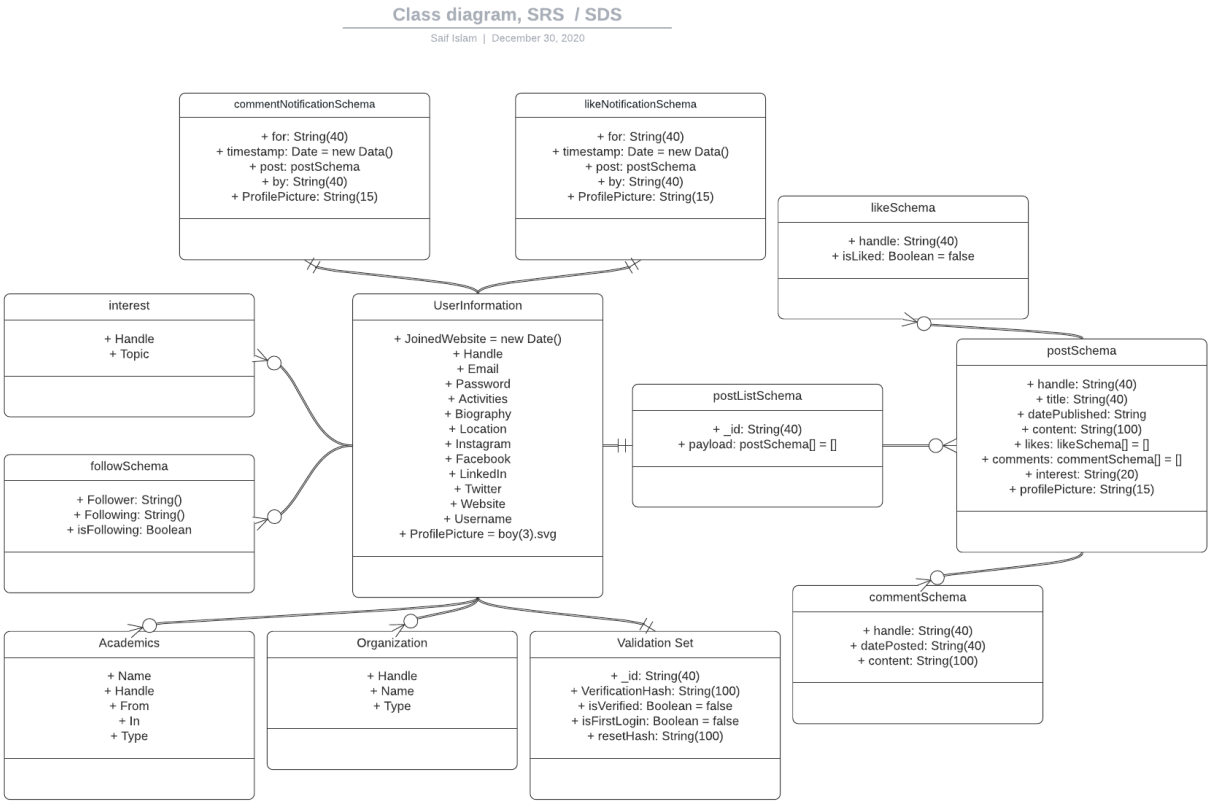
Each of the units designed is designed to follow the SOLID level of principles, to whatever degree we could, so the use cases defined above are self contained and have been tried to not depend so much on other units as much as possible.

Different components do exists separately. For example, on the client side, each page is broken down into different components, where each component has a separate individual set of styling, and each client side view makes a single request to the backend when triggered on viewing.

The system is decomposed into how the data flows betweent the two main systems, and how the web server acts as an intermediary to transfer the information. A client side requested is routed by a router service in the *routes* section, and each route is assigned a specific controller function that deals with the lower database side of functionality, by sending and running queries on the database, and using the Mongoose Schemas that it has to to save data on the MongoDB data.

Different classes have been made to deal with those set of information

- Academics, MySQL, a list of academic related user information
- Connection, Helper Class, helper class for server side service
- folowSchema, MongoDB, list of users following or followed by a connection
- likeNotificationSchema, MongoDB, like details needed to generate a like notification
- likeSchema, MongoDB, like count helper class
- postSchema, MongoDB, post information
- PostEntry, Helper Schema, helper class for server side service
- ValidationSet, MongoDB, set of information regarding to validate user
- commentNotificationSchema, MongoDB, comment details needed to generate a comment notification
- CommentEntry, Helper Class, helper class for server side service
- FeedEntry, Helper Class, helper class for server side service
- Organization, MySQL, organization information for a user
- commentSchema, MongoDB, required information to store a comment
- postListSchema, MongoDB, required information to store a list of posts
- UserInformation, MySQL, needed user information stored within user information



7 System Level Architecture

The system architecture is split into different containers, each with their own set of responsiblities. For example, the *web-server* denoted in the diagram below is responsible for setting up the routes for the backend and frontend to communicate.

The backend itself is contained within the *backend* container, which includes of all the responsibilities assigned within the
server folder. This means that controllers, routes, and the middleware are all pointed and mounted on a specific address. Whenn running on localhost, this means that they are at the /api/ point, while all of the client side responsibilities are at the / point.

Each page within the client side, is contained within a App.js, which basically takes care of the routes, and provides context for the login, for the navbar, and for posts, and redirects to specific pages based on whether the user is logged out or logged in. These pages are named in the below given pages as app components.

Each of the database containers, when running on the development environment, lets Docker Compose allocate specific ports for them along with the volume mapping needed for storing and directing their data and configuration files. Each of the databases, though not all of them have been

properly integrated at the moment, exist as containers with a specific use case and intention for the system.

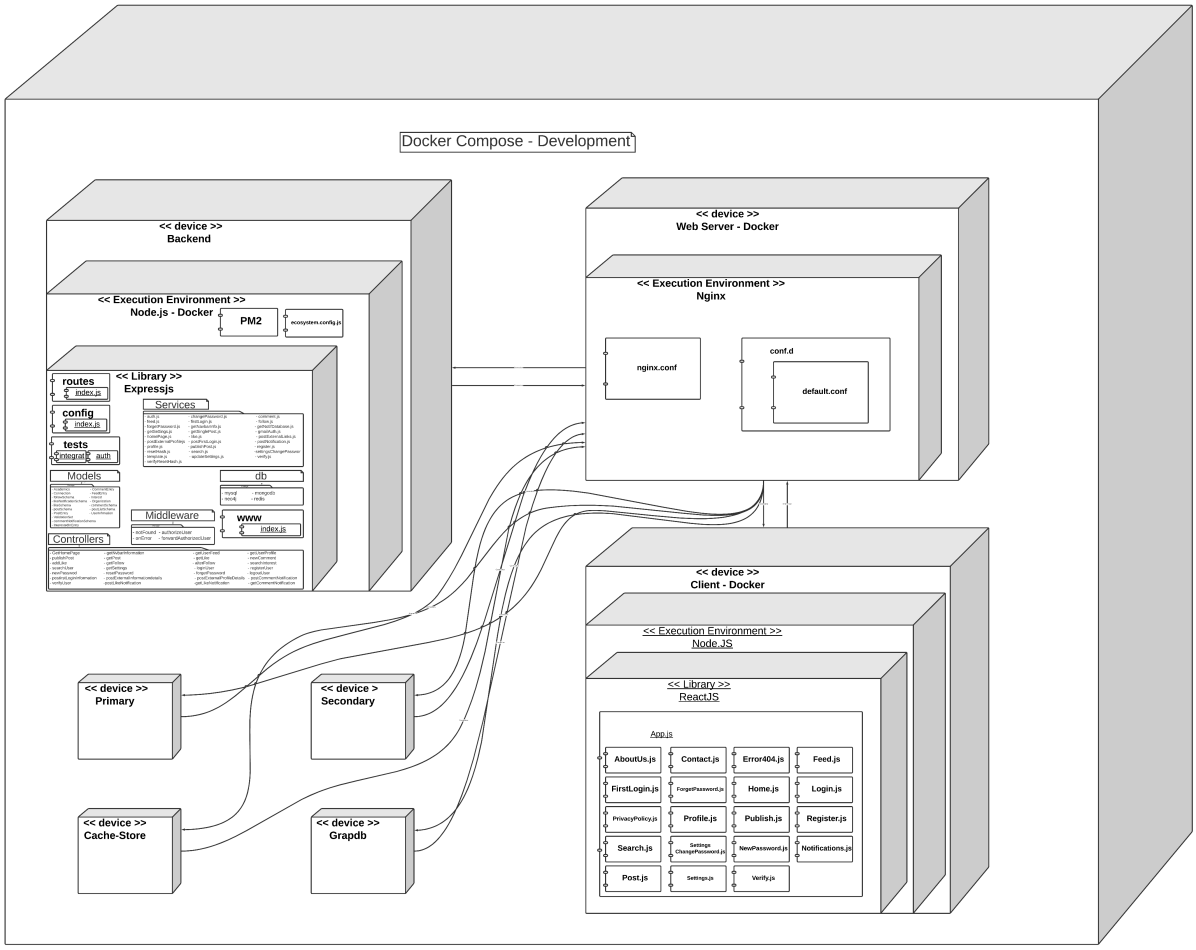
When running in production how ever, the backend is supposed to be deployed to a *Virtual machine* hosted on the Azure Cloud Service, while the frontend is to be deployed on the Vercel service, which provides an interface for the client side of things. The two services intercommunicate with each other as components pull up information by GET and POST HTTP protocols by and from the backend web-server hosted on the Azure Virtual Machine. The databases also exist as containers on the Virtual Machine itself, so all of the responsibility, instead of maintaining the frontend on the VM, is allocated to the VM.

The diagram below shows the decomposition of different containers into different components and services and pages. While the interaction of sending, GET and POST information is indicated by the arrow symbols between the different containers.

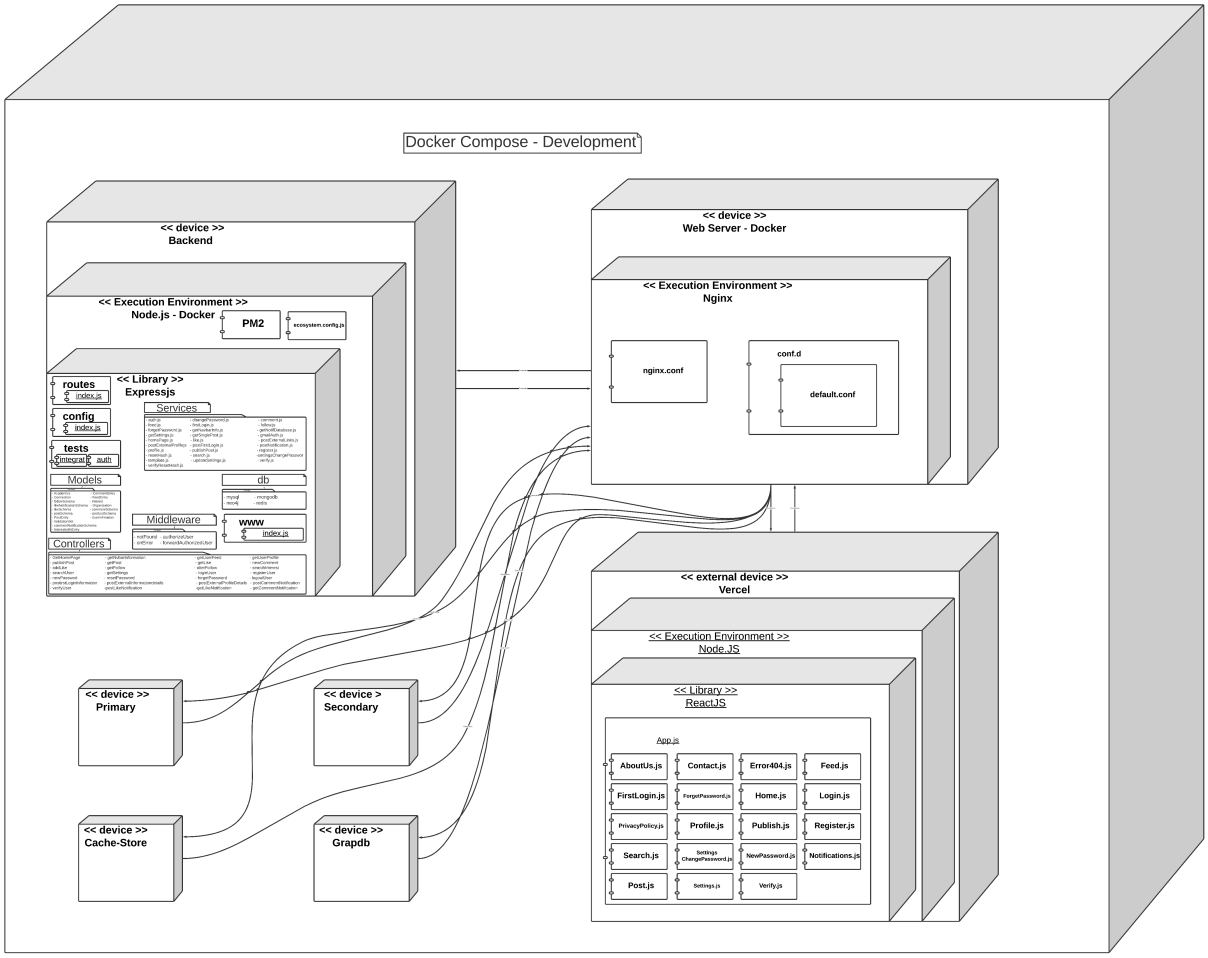
All of services are maintained as monolith applications, instead of the microservice architecture, so we see little interaction with external systems and diagrams, since everything is contained within the *Docker Compose* environment.

The error handling is done within multiple levels. Both the client side and the server side have their own list of services, which take care of what kind of error to propagate. An error on the client side is shown by an alert, which is popped up from the client side level, which comes from the controller level of the server, which comes from the service side of the server.

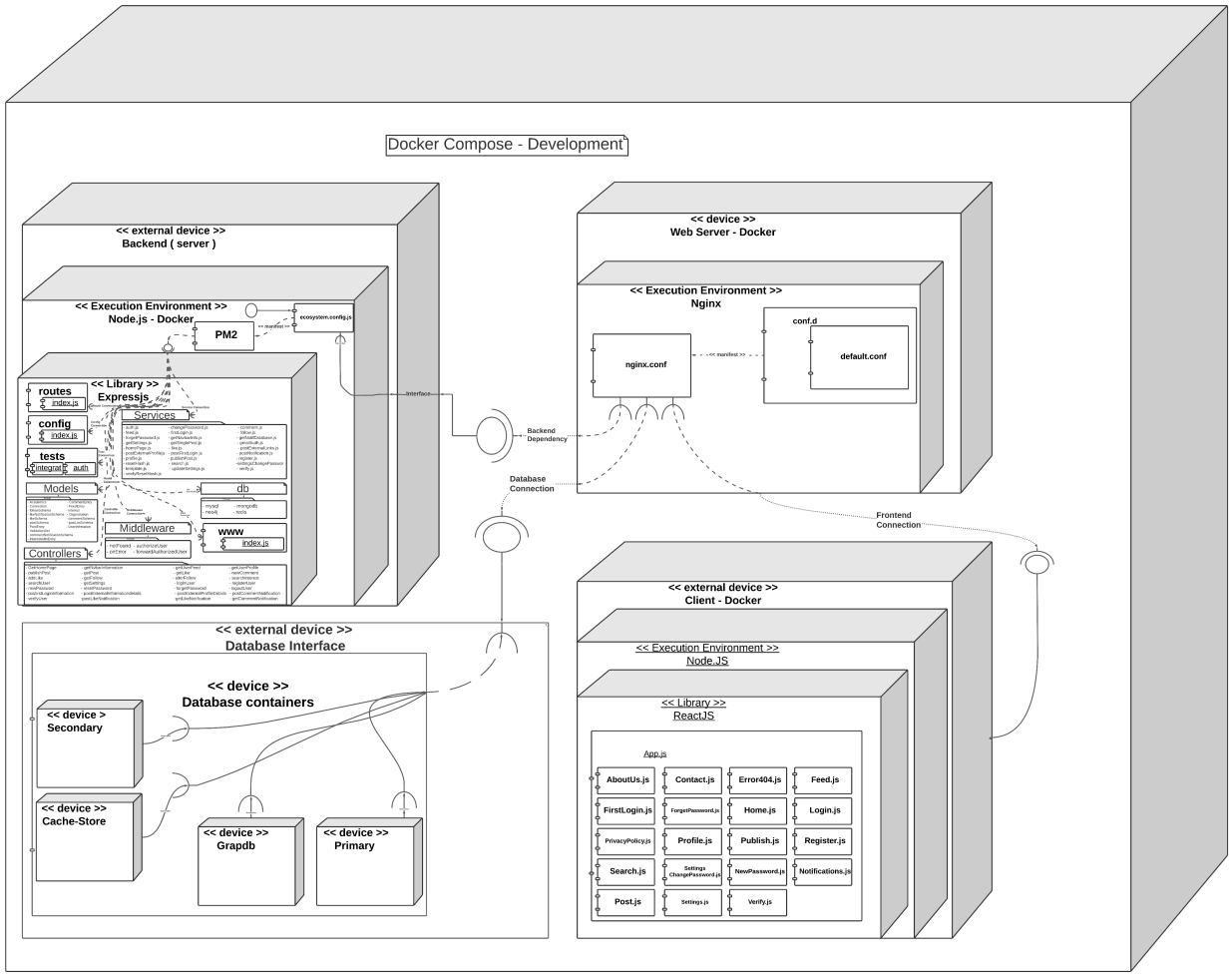
7.1 System Level Architecture - Deployment



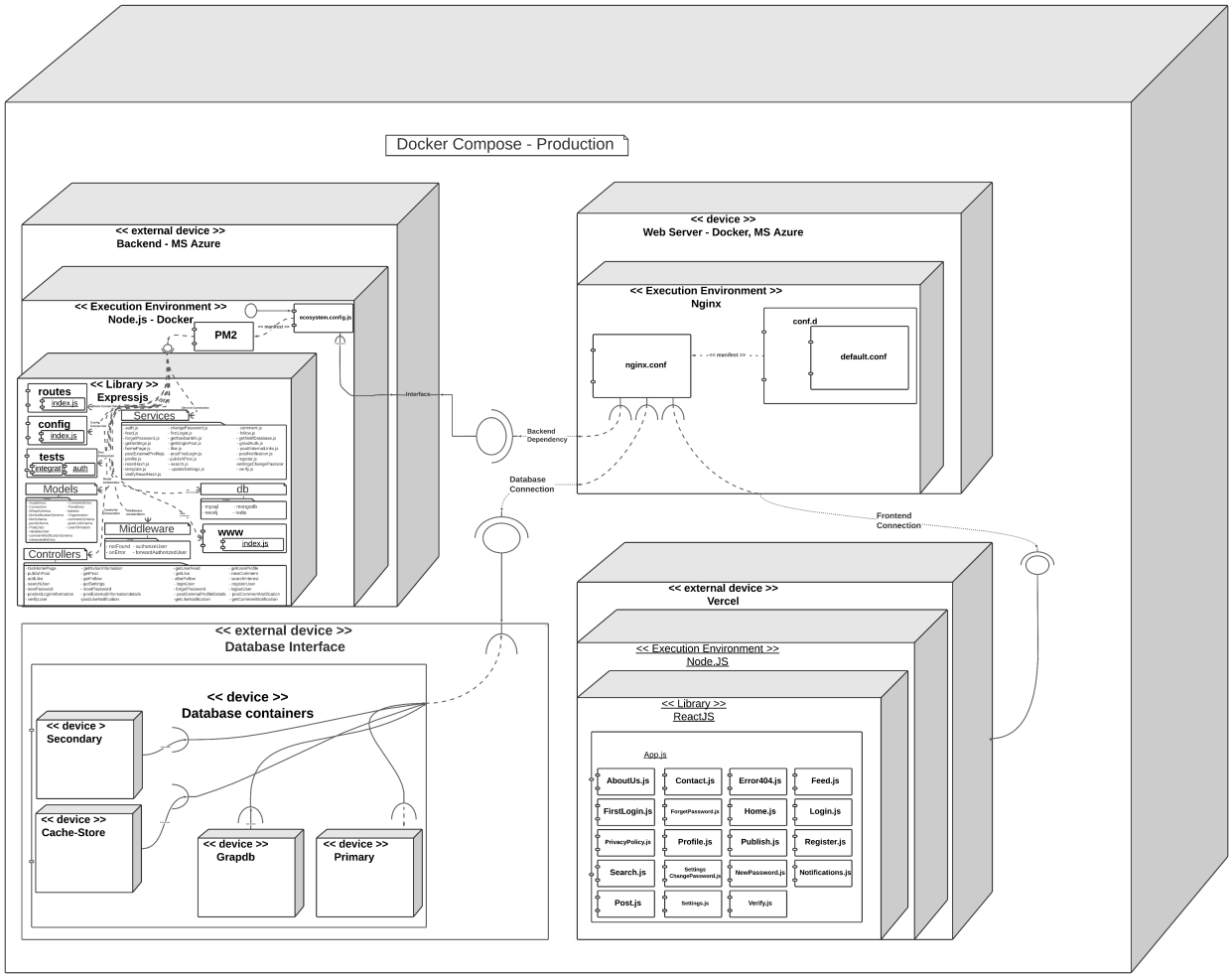
7.2 System Level Architecture - Production



7.3 Component Level Diagram - Development

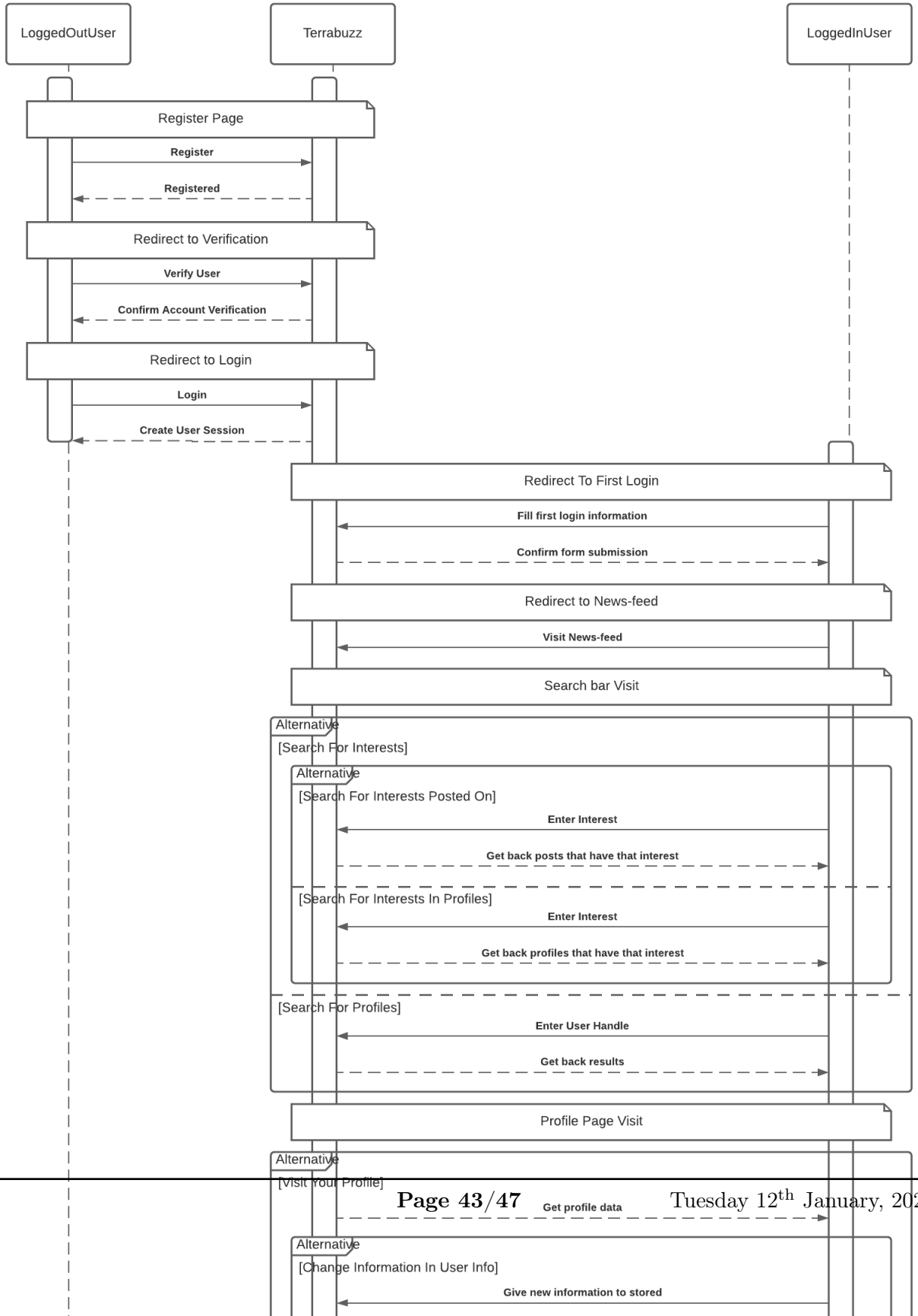


7.4 Component Level Diagram - Production

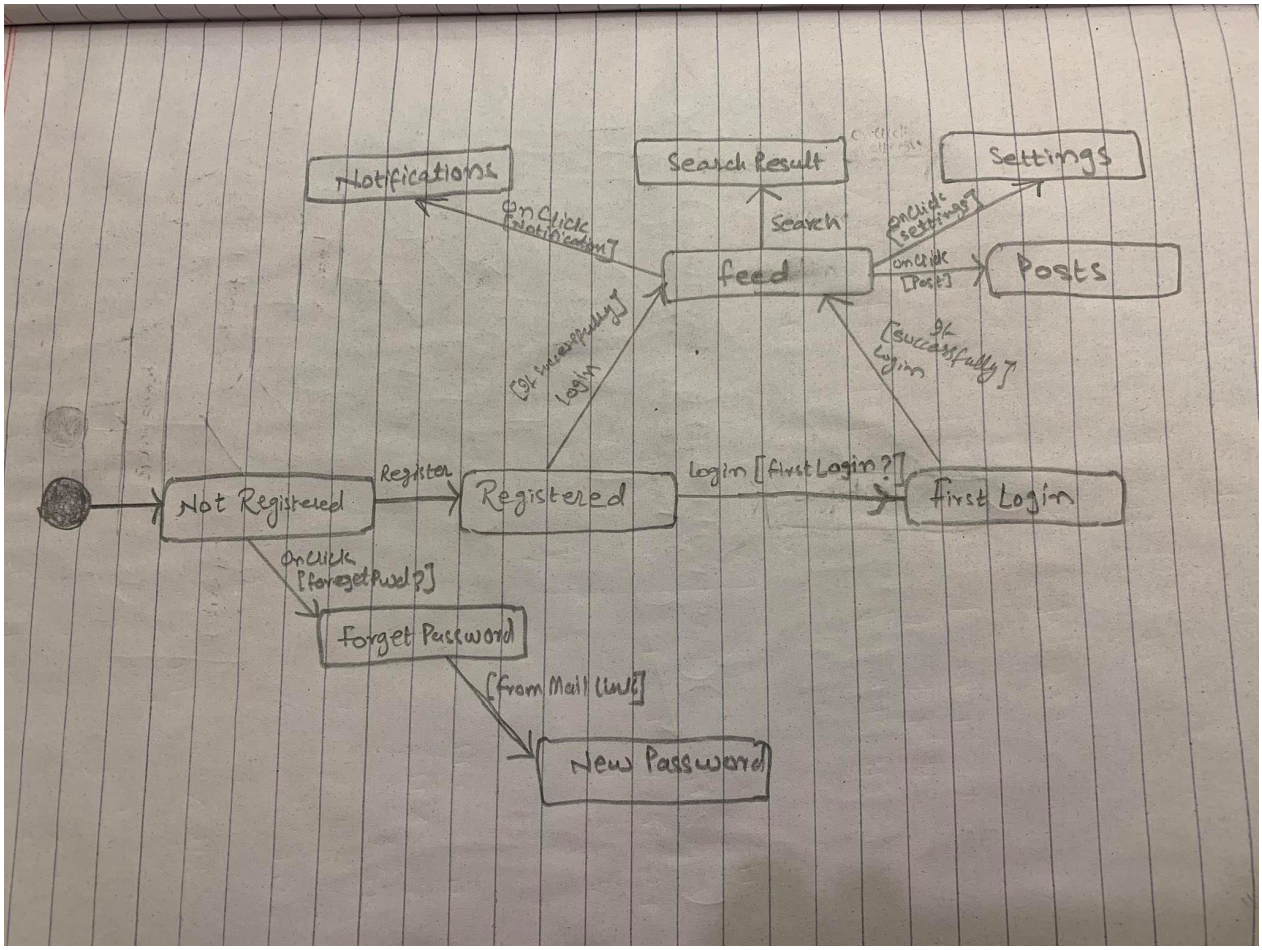


8 Application Design

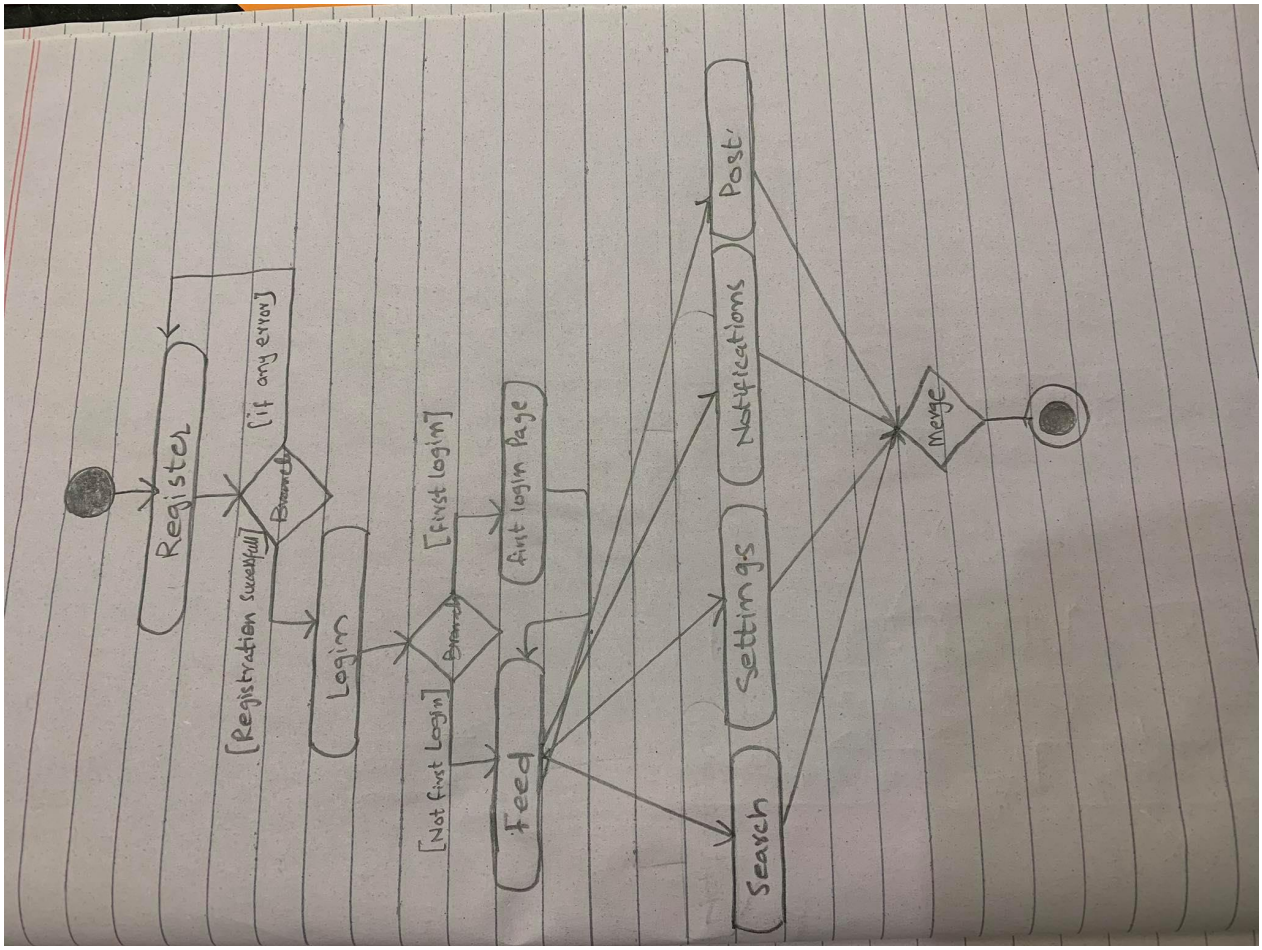
8.1 Sequence Diagram



8.2 State Transition Diagram



8.3 Activity Diagram



9 References

[This section should provide a complete list of all documents referenced at specific point in time. Each document should be identified by title, report number (if applicable), date, and publishing organization. Specify the sources from which the references can be obtained. (This section is like the bibliography in a published book).]

- The original required project proposal for *Software Design And Analysis* is here.
- The original project proposal submitted is here.
- The GitHub repository as an origin of truth for the project is here.
- The relevant issues to solve the units for the project are listed here with their closing pull request.
- The list of pull request made to the project are listed here.
- The final ERD decided and designed for the application is here

The rest of the documents that we used were little, and mostly just helper guides from Stack Overflow, and on GitHub itself.

10 Appendices

The only detail to be noted is that the root level application, the client application, and the server application, all have *secrets* that are used files. They are credentials and security measures in each case, and are too sensitive to be exposed to the general public, thus, they have been kept locally and safely with us on a private Notion repository. The application cannot work without it.

Other than this, every detail has been mentioned above.