



**FINAL SEMESTER ASSESSMENT (FSA)  
B.TECH. (CSE)  
VI SEMESTER**

**UE18CS355 – OBJECT ORIENTED ANALYSIS AND DESIGN  
WITH SOFTWARE ENGINEERING LABORATORY**

**PROJECT REPORT**

**ON**

**Code Management and Version  
Control**

SUBMITTED BY

**NAME**

**SRN**

- |                           |               |
|---------------------------|---------------|
| 1) Arjun Chengappa        | PES1201800119 |
| 2) Lavitra Kshitij Madan  | PES1201800137 |
| 3) Vishnu Charan Golugula | PES1201801230 |

**JANUARY – MAY 2021**

**DEPARTMENT OF COMPUTER SCIENCE &  
ENGINEERING**

**RR CAMPUS,  
BENGALURU – 560100, KARNATAKA, INDIA**

## TABLE OF CONTENTS

Sl.No	TOPIC	PAGE No
	ABSTRACT	1
1.	SOFTWARE REQUIREMENTS SPECIFICATION	2
2.	PROJECT PLAN	21
3.	DESIGN DIAGRAMS	27
4.	MODULE DESCRIPTION	35
5.	TEST CASES	36
6.	SCREEN SHOTS OF OUTPUT	46

# ABSTRACT

Code management and version control system is an important topic in the subject of software engineering. Presently, there are many softwares that help implement this set of functionalities, most prominent of them being GitHub. Code management, also known as Source code management (SCM), is used to track modifications to a source code repository. SCM tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple collaborators. This is a key necessity in the present times as it makes coding faster, error-free and more effective.

In this project, we have developed a system that is very close to GitHub and it is called CodeHub. It has been developed, implemented and tested using tools like Flask, Sqlalchemy, and many more. This project implements the four main important functionalities- Create repository, manage code (push, pull), collaboration (share and edit code) and view change history. These functionalities are the bare minimum that such a system should have and all of these features have been implemented in a simple way. This system also has security and privacy features such as authentication- login and signup feature for each account that is created.

In the course of this project, first a software requirements specification document was created based on the requirements that were needed for the project. This document has since then undergone one change due to lack of availability of time and resources for this project. After this document was made, the next phase was that of planning where a project plan was created that contains all the details, schedules, tools, resources, etc. required to finish the project. Following this, all the diagrams related to the system were made such as the architecture diagram, class diagram, state diagram, activity diagrams and the sequence diagrams to better understand the end-goals and to define a clear path to these goals. On completion of these diagrams, the system was developed, built and implemented using the various tools and technologies mentioned in the project plan and the system requirements specification documents and then the code was tested thoroughly after coming up with more than two dozen test cases that involved unit tests, integration tests and a system test. Finally, the project was deployed on Heroku and the project was successfully completed.

# CHAPTER 1

## SOFTWARE REQUIREMENTS SPECIFICATION

### Table of Contents

#### **Introduction**

Purpose  
Intended Audience and Reading Suggestions  
Product Scope  
References

#### **Overall Description**

Product Perspective  
Product Functions  
User Classes and Characteristics  
Operating Environment  
Design and Implementation Constraints  
Assumptions and Dependencies

#### **External Interface Requirements**

User Interfaces  
Software Interfaces  
Communications Interfaces  
Hardware Interfaces

#### **Analysis Models**

Use case diagram:

#### **System Features**

- **Description and Priority**
- **Stimulus/Response sequences**
- **Functional requirements**

User Authentication  
Create Repository  
Search  
Repository Management  
Issues  
Profile Management

Commits

### **Other Nonfunctional Requirements**

Performance Requirements

Safety Requirements

Security Requirements

Software Quality Attributes

Business Rules

### **Other Requirements**

**Appendix A: Glossary**

**Appendix B: Field Layouts**

**Appendix C: Requirement Traceability matrix**

## **Revision History**

<b>Name</b>	<b>Date</b>	<b>Reason For Changes</b>	<b>Version</b>
Suraj MS, R.Shrenik, Niranjan Bhaskar K, Adarsh Nair	06/02/21	Document Creation	1.0
Arjun, Lavitra, Vishnu	19/04/21	Lack of time and resources	1.1

# **1. Introduction**

## **1.1 Purpose**

The purpose of this document is to give a detailed description and the requirements for the Code Management and Version Control System. Illustrating the purpose and complete declaration for the development of this system, elaborating on its system constraints, interface and interactions with other external applications, this document is primarily intended to be used to propose to a customer for its approval.

## **1.2 Intended Audience and Reading Suggestions**

While the software requirement specification (SRS) document is written for a more general audience, this document is intended for software developers, project consultants, and team managers.

This document need not be read sequentially, users are encouraged to jump to any section they find relevant. Below is a brief overview of each part of the document.

### **1. Introduction**

This section offers a summary of the Code Management and Version Control System including relevant objectives and the product scope of the intended platform.

### **2. Overall Description**

This section includes the context and origin of the platform, a precise summary of the major functions that the product performs or lets the user perform. Additionally, a top-level data flow diagram of the Code Management and Version Control System is also included. Further, various user classes, its characteristics and its operating environment are elaborated. The section is concluded by describing any items or issues that will limit the options available to the developers and further elaborate upon assumptions and dependencies.

### **3. External Interface Requirements**

This module includes description of the logical characteristics of each of the user interfaces, software interfaces (including databases, operating systems, tools and libraries) of the system.

### **4. Analysis Models**

Includes a use case diagram to showcase the working of the system.

### **5. System Features**

This component illustrates the organization of the functional requirements of the product by highlighting system features and the major services provided.

### **6. Other Non-Functional Requirements**

This section elaborates on the performance, safety, and security requirements. Additionally, it also constitutes some quality characteristics and operating principles for the product.

### **7. Other Requirements**

Any other requirements apart from the ones specified above.

## 1.3 Product Scope

The Code Management and Version Control System is an internet code hosting platform for effective collaboration and distributed version control. The system manages and stores revisions of projects, while tracking contributions from the users. Although it is traditionally used for code, it could be used to manage any other type of file such as a word document or a final cut project. It can be thought of as a ledger containing all the changes done to the document by different collaborators.

### Features:

- Effective collaboration between users on the platform.
- Commits to hosted code / other data on the platform
- Issue tracking with labels (Urgent, Minor, Major).
- Incremental Search (Search-as-you-type) feature for finding users, repositories, issues etc.
- Email notifications.
- Task lists, document viewer, code viewer

## 1.4 References for the SRS creation

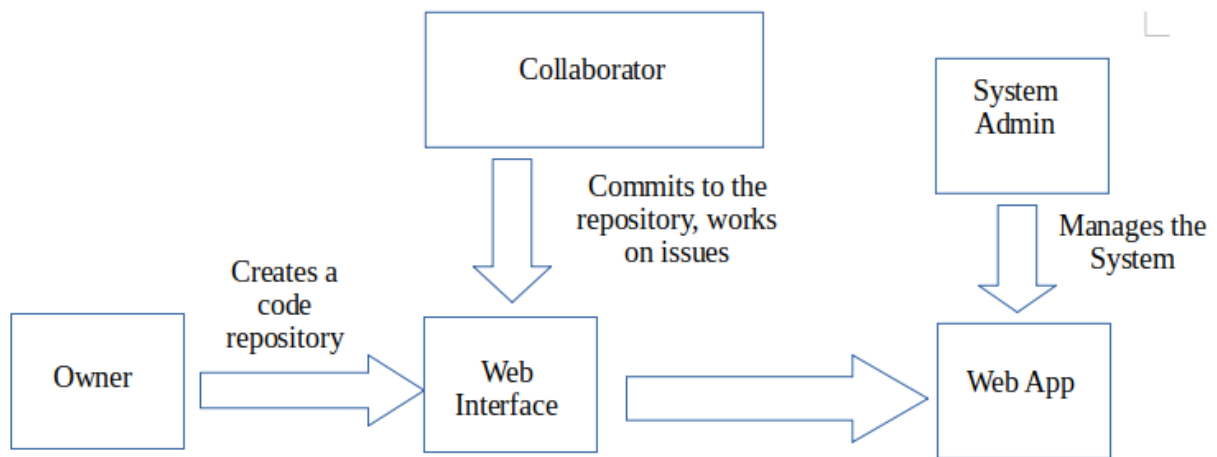
- IEEE Software Engineering Standards Committee, "IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications", October 20, 1998.
- Wieggers, Karl, Software Requirements (3rd Edition), Microsoft Press 2013.
- Standardized Statement Templates – beginning to write SRS
  - Link: [Coursera - Standardized Statement Templates](#)

# 2. Overall Description

## 2.1 Product Perspective

VCSs date back to the 70s. Version control systems records changes to a file or a set of files. A distributed VCS is one where individual users each have a copy of the history which lists all the changes done to the file or collection of files. A Code Management System works with the VCS, and provides additional features such as an intuitive interface allowing for easy collaboration, issue creation and tracking and controlling access to the set of files. The perspective of different sets of users are shown below in the diagram

The Product is a web application to help users (or different groups of them) maintain and manage the process of the development. It will also allow the users to assign levels of access for the files stored in the repository. There are mainly three user perspectives to the product. The Owner owns the repository and can modify the repository, allow collaborators, manage their access and has near unrestricted access to the files. The Collaborators can work on the issues filed on the repository. They can also commit to the repository subject to reviews to modify the origin repository. Normal users who aren't collaborators can either request the owner/collaborator for access or work on an existing issue. The System admin manages and maintains the web application and ensures the continual operation of the services provided by it.



*System Perspective Diagram*

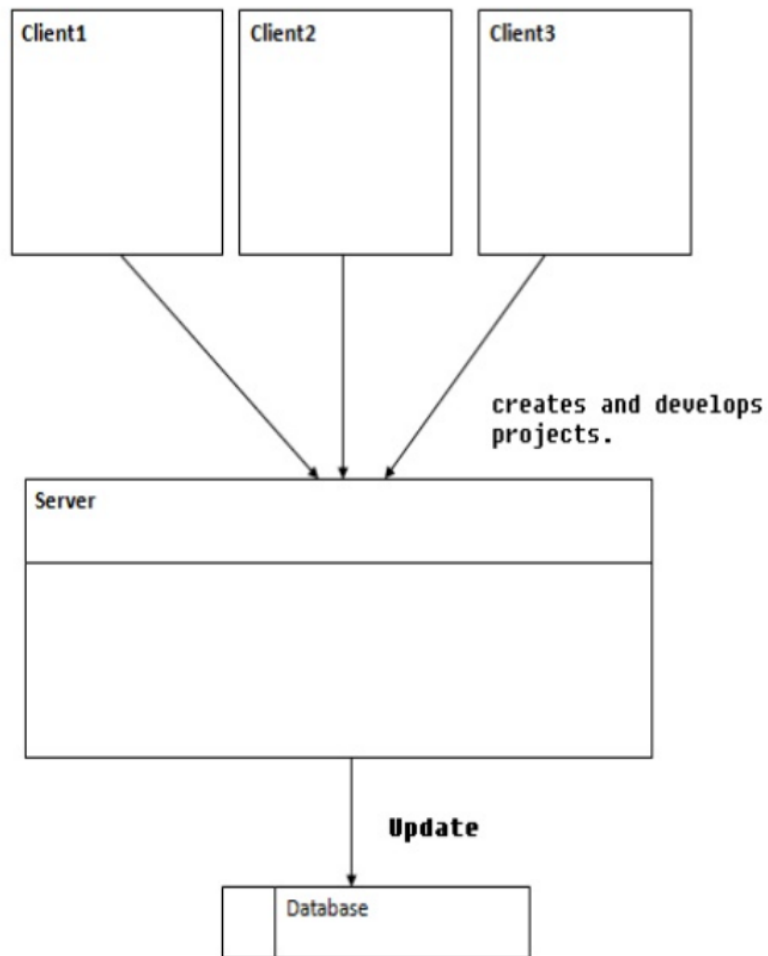
## 2.2 Product Functions

The product is meant to provide the below features to the users :-

- Hosting content (code, docs etc) on the web.
- User authentication and login.
- Search for different repositories which are designed to be public.
- Repository ownership and Access Control.
- Interface to post issues, review them with comments and tags (closed, pending, open etc)
- Maintain an insights section showcasing most active collaborators and other repo activity



**Data flow diagram**



### **2.3 User Classes and Characteristics**

User Classes	Frequency of Use	Functions Used	Privilege Level	Characteristics
--------------	------------------	----------------	-----------------	-----------------

Owner(s) of the repository.	Medium-High	<p>Has unrestricted access to the repository</p> <ul style="list-style-type: none"> <li>• managing access</li> <li>• Request approval for contributor addition, commits</li> <li>• Can comment.</li> <li>• issue creation, deletion, tracking</li> <li>• overall maintenance</li> </ul>	High	Their day-to-day interaction with the system would include approving commits and taking important design decisions wrt the project
Developers and designers	High	<ul style="list-style-type: none"> <li>• Committing changes</li> <li>• Bug tracking, fixing</li> </ul> <p>Based on the access provided to them by the owner, they will be able to make changes to the repository which will be reviewed by a senior dev or repo owner.</p>	Medium-High	Developers form the majority of the user base. Their day-to-day activities would include committing changes to the repo and requesting approvals.
New collaborators	Low	<ul style="list-style-type: none"> <li>• Request approval</li> <li>• Bug fixing (if allowed)</li> </ul>	Low	New users wishing to contribute to a repo have the option of fixing bugs and committing changes to a repository subject to approval.
System Administrator	Low-Medium	<ul style="list-style-type: none"> <li>• Website Maintenance and Security</li> <li>• Handling the application's server-side and client-side issues.</li> </ul>	High (wrt the system itself)	Sys-admins handle the end-to-end maintenance and security of the web application.

		<ul style="list-style-type: none"> <li>• Bug fixes (if any)</li> <li>• Overseeing day to day operations</li> </ul>		
--	--	--	--	--

The most important user classes include the developers and collaborators interacting with the system on a day-to-day basis from both a technical perspective and a business perspective. The system in place allowing for seamless usage requires constant maintenance without which the user base would decline.

## 2.4 Operating Environment

**Environment:** Cross-browser responsive web application. Will run on any desktop OS that comes with a web browser.

**Hardware:** Not Applicable. Use online hosting services.

**Version:** 1.0

**Other dependencies:** None

## 2.5 Design and Implementation Constraints

- **Regulatory policies:**
  - Keeping content on website civil (Should not host abusive / unlawful content on website)
  - Overloading servers with extremely high number of requests may lead to the user getting banned from the platform.
- **Potential Hardware and Software problems:**
  - Software requirements: A web browser
  - May not work on mobile operating systems
  - Multiple users pushing code into the same branch simultaneously can result in merge conflicts.
  - Upload of files through the web application is limited by the client browser.
- **Security considerations:**
  - Use HTTP for the website.
  - Encryption of user passwords.
- **Others:**
  - The system might be limited by the number of active servers.

## 2.6 Assumptions and Dependencies

- Developers have experience creating basic user interfaces on the web and working with databases (relational or non-relational).
- Familiarity with implementing basic security protocols on a website.

### 3. External Interface Requirements

#### 3.1 User Interface

- **Home Page**
  - First screen that the user is able to view on the website.
  - Has a Nav bar containing buttons depending on whether or not the user has signed in.
- **Login**
  - Contains a form with two inputs: Username and Password.
  - Shows the respective errors when an invalid login is attempted.
  - Redirects to homepage upon successful login
- **Signup**
  - Contains a form with 4 inputs:
    - Name
    - Username
    - Email
    - Password
    - Profile picture
  - Necessary validations are implemented for all inputs.
  - Shows the respective errors when an invalid signup is attempted.
  - Logs the user in and redirects to homepage upon successful signup
- **Repositories**
  - Displays the repositories that are owned by the user or those in which the user is a collaborator.
  - The repository name, owner name and a brief description of the repository is displayed
- **Create Repository**
  - Allow users to create repositories.
  - Displays a form with 3 inputs at first:
    - Repository name
    - Repository description
    - Number of collaborators
  - Does the necessary validation upon form submission.
  - Redirects to another form if no discrepancies are found. Users can add the collaborators in this form.
  - Repository is created on submission of the form and the user is redirected to the Manage Repository page of the repository.
  - The names of the collaborators that don't exist is displayed on top of the redirected page
- **Manage Repository**
  - Users can add collaborators, upload files and create folders in the repository using this page.
  - Only the owners are allowed to add collaborators to a repository.

- **Code Viewer**
  - In-app code viewer to view source code
  - Change history of the file is displayed towards the right of the display screen.
- **User Settings**
  - Profile
    - User has the option of viewing and editing his profile details including his name, email, bio, connected accounts and visible elements of his/her account.
  - Account Settings
    - User has the option of changing his/her username, exporting account data and deleting and migrating the account.
  - Account security
    - This section covers the security aspects of the account on the system (passwords)
  - Notifications
    - Here, the user has the ability to choose the types of notifications (email, SMS etc) he/she receives.

### 3.2 Software Interfaces

- Flask web-servers can be used and information will be accepted through HTML forms and using various JS methods the basic login authentication will be conducted.
- A relational will be used to store various information about the repository, files, commits, etc. from the user and the collaborator's through the GUI's input.
- The frontend of the website can be created using CSS and Bootstrap4.

### 3.3 Communications Interfaces

The product makes use of only one standard for communication:

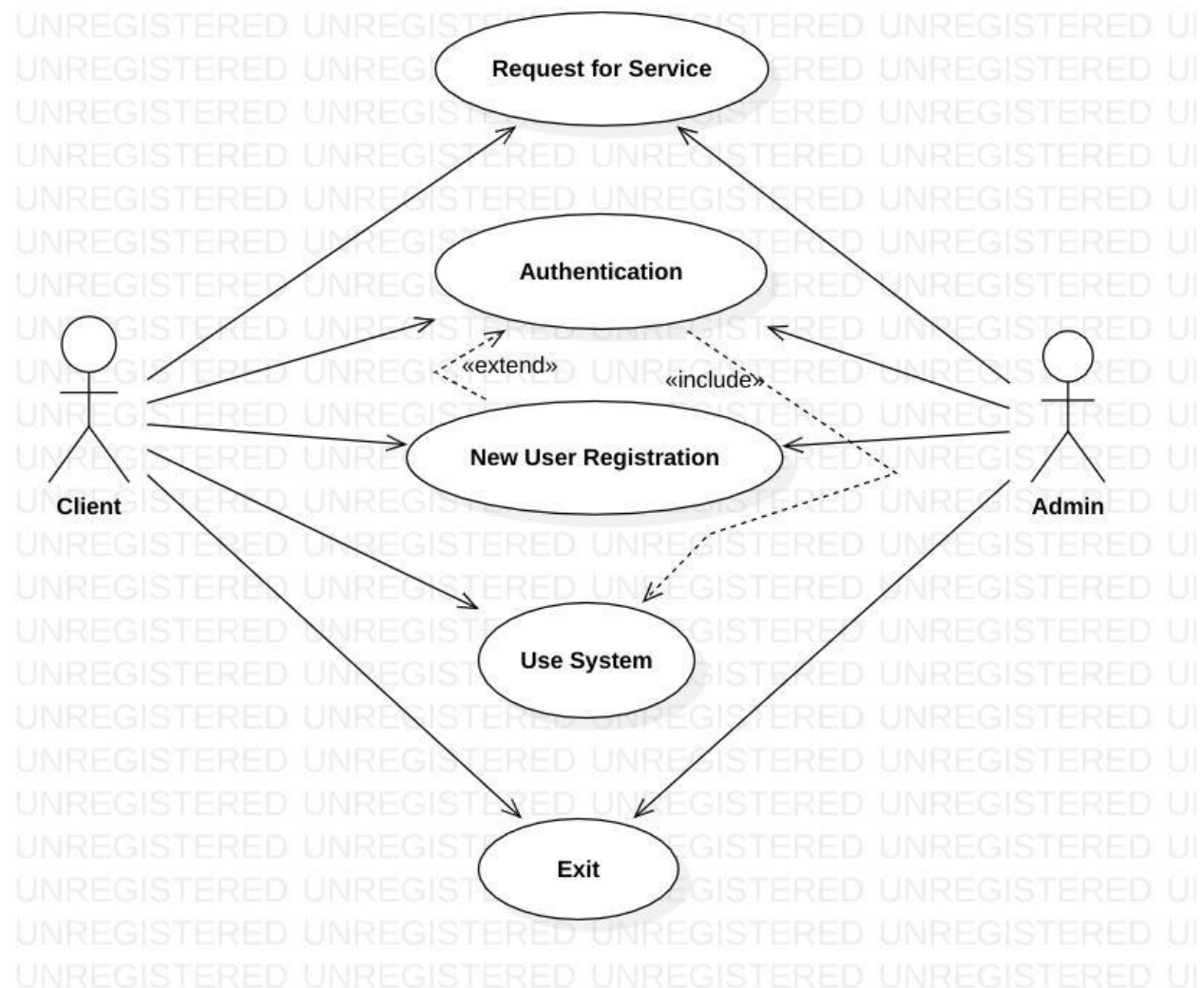
- **HTTP protocol:** For web interface access and data transfer. It needs port 80 to facilitate communication at the server side.

### 3.4 Hardware Interfaces

- a) **Server side:** A Cloud based web server with port 80 open for the web application.
- b) **Client side:** A computer with a working NIC card and a connection to the Internet.

## 4. Analysis Models

Use case diagrams:



Services include repository creation, management, profile management, collaboration and commits etc.

## 5. System Features

### 5.1 User Authentication

#### 5.1.1 Description and Priority

To gain access to the platform and its features, it is required that you must be authenticated. You supply or confirm credentials that are unique to you thereby proving that you are who you claim to be. The system supports the following method of authentication:

- Username and Password only

### 5.1.2 Stimulus/Response Sequences

- Registered User -> Redirect to login section -> Login with username, password -> Home page
- Unregistered User -> Redirect to registration section -> User is prompted to enter details -> Validate details -> Home page

### 5.1.3 Functional Requirements

**REQ-1:** Working login / signup page on the website

**REQ-2:** Username based verification for registration and login.

## 5.2 Create Repository

### 5.2.1 Description and Priority

Repository creation is vital to a code management and version control system. Repositories are essentially containers with the concerned user's project files and each file's revision history. The ownership of a repository is confined to a single user. But a maximum of 10 collaborators can be added at a time.

### 5.2.2 Stimulus/Response Sequences

- Upon choosing to create a repository:
  - **Configure a new repository:** This allows the user to create an empty repository with the concerned user as the owner of the repository with access modification abilities.
- Commits to the repo are pushed to the database and the changes will be reflected in the user's repository.

### 5.2.3 Functional Requirements

**REQ-1:** "Create repository" page should be functional

**REQ-2:** Repository should have a valid name (Duplicate names aren't allowed)

## 5.3 Search

### 5.3.1 Description and Priority

The search feature is essential for users to facilitate navigating the entire system with ease. The feature allows users to perform the following queries:

- Search for repositories and users.
- Search for matches in the files and the issues section within repositories.

### 5.3.2 Stimulus/Response Sequences

- The search field is universally accessible. The entered term(s) will be searched across the existing repositories and users to look for matches.
- Upon reception of keywords in the search field, the client sends requests to the search provider (HubSpot, Google Custom Search etc) to get corresponding responses as keyword matches (as mentioned above).
- The provider checks the database for matching keywords and further, returns the desired values to the client.
- Search results are displayed to the client.
- No results are returned if no matches are found.

### 5.3.3 Functional Requirements

**REQ-1:** Only english characters are allowed.

**REQ-2:** Scope of search can be specified if required. (Providers handle this automatically)

## 5.4 Repository Management

### 5.6.1 Description and Priority

The Repository Management feature allows users to configure the repo settings.

- **Access Control:** Can only be performed by the owner. It allows the owner to designate various privileges to collaborators as well.
- **Basic operations & features:** Renaming, deleting or sharing repository details.
- Viewing files in the repo.

### 5.6.2 Stimulus/Response sequences

- The user can view repositories on his dashboard and via search.
- Clicking on one of the repository's hyperlinks redirects the user into the repository-specific overview page.
- Redirect web app to view and modify repo settings.

### 5.6.3 Functional Requirements:

- **REQ-1:** The User should have access to view or make changes to the repository.

## 5.5 Issues

### 5.8.1 Description and Priority

Issues are a great way to keep track of the features, enhancements, and bugs pertaining to a repo. The users can collect user feedback, report software bugs,



and organize tasks Moreover, the author/contributor can also link an issue to a commit. Issues can be tagged “closed” or “pending”.

### 5.8.2 Stimulus/Response sequence

- An “issues” tab is made available in the repository page where users can do either of the following operations:
  - **File a new issue:** This essentially allows the user to create a new issue concerned with the repository along with an issue description.
  - **Comments:** This feature allows other users to broadcast their views concerned with the issue.
  - **Issue assignment:** This allows the collaborators (with read/write access) of the repository to assign the issue to a specific user, so that he/she can work on resolving the issue pertaining to the repository.
  - **Close resolved issue:** Say, the issue has been resolved by fellow developers and thereby we no longer need to keep the issue open. This option allows the member to close the issue.

### 5.8.3 Functional Requirements

**REQ-1:** The repository should exist.

**REQ-2:** The issue can be opened by any contributor, but can be closed only by the users with privileges and can be assigned only by the issue creator and the privileged users.

## 5.6 Profile Management

### 5.10.1 Description and Priority

Profile Management is a feature used to organize and describe the user’s profile. Profile management is a core feature necessary to provide users full access on what information they are providing on the website. The main features are specified as follows :-

- **Account Settings and Security :-** Provides a method for the user to change username, password, enable Two-Factor Authentication(2FA) and delete the account if the user chooses to do so.
- **Repositories :-** Lists the repositories where the user has either owner/collaborator privileges.
- **Profile Details :-** Provides a summary of the user’s profile.

### 5.10.2 Stimulus/Response Sequences

- On the profile page, we have a summary of the profile being displayed.
- To use the other features, the user goes to the edit profile wizard, where the following tabs are present :-

- **Account Settings and Security**:- This tab is used to rename, change password, enable Two-Factor Authentication and delete the account.
- **Repositories** :- This tab shows a list of all the repositories where the user is listed as a collaborator or owner.

## 5.7 Commits

### 5.11.1 Description and Priority

Commits are snapshots of a repository at particular instances of time. Commits have the following attributes associated with them:

- **Timestamp of creation**: When the changes were made.
- **Commit author**: The user who created the commit.

These records are made available in the repository and can be very useful to track changes. A commit records changes made to a particular repo in this case.

### 5.11.2 Stimulus/Response Sequences

- The owner or the collaborators can push code into the repository in the Manage Repository page.
- Every push is noted down and tracked by the server. Upon viewing the code, the change history can also be seen.
- Assuming that there was no conflict with merging the commit into the repo, the commit is pushed.
- Say there was a conflict, the merge process is aborted and an error is shown.

### 5.11.3 Functional Requirements

**REQ-1**: The user trying to commit changes must have read & write access to the repository concerned.

**REQ-2**: Absence of merge conflicts essential for successful merge.

## 5.8 Other Nonfunctional Requirements

### 5.9 Performance Requirements

- Response times on the website should not take >3s ideally (ignoring client side constraints) for active CRUD operations.
- Higher speeds can be achieved with better compression, optimized images and CSS and better hosting services.

## 5.10 Safety Requirements

It is recommended that developers keep in mind the following before and while designing the system:

- **Strengthening Access Controls:**
  - Follow the least privilege model (subject should be given only those privileges needed for it to complete its task)
  - Restrict the creation of repositories to prevent users from exposing organization information in public repositories.
  - Revoke access for all inactive users who are no longer part of the collaborators.
  - Ensure users do not share GitHub accounts or passwords.
  - Ensure every contributor uses two-factor authentication on their account.
- **Storing credentials in GitHub files:**
  - Leaking secrets to repositories, either via code, config files, or commit messages, provides a gateway for attacks.
  - Initiate removal of the repository once it is realized that sensitive info has been leaked.
- **Enabling security alerts (optional feature)**

## 5.11 Security Requirements

- **Password Protection**
  - The passwords stored in the server database must be encrypted to ensure that no security breach occurs in case of a leak.

## 5.12 Software Quality Attributes

- Databases should remain consistent in case of an error.
- Optional:
  - Test-driven development approach with unit test execution and approval cycles.
  - Avoid inline and embedded styles during the creation of the web app to avoid context switching between HTML and CSS parsers.
  - Use browser-native JSON methods. Avoid XML for better application performance.
  - Style guide to be followed : [JS Style Guide](#)
- A web-based platform offers medium-high portability considering widespread internet access.

## 5.13 Business Rules

- **Users**

- They retain ultimate administrative control over their User Accounts and the Content within it.
- Functions include account creation, repository creation and modification, user access control and profile management.
- **Owner of the repository**
  - The owner of a repository has ultimate administrative control over it.
- **System Administrator**
  - On the organizational side, the final authority on matters concerning feature approval and deployment lies with the system admin subject to technical and legal constraints.
  - Functions include improving the website, its performance, UI changes and changes to the database.

**Domain requirements:** Domain requirements are the requirements which are characteristic of a particular category or domain of projects.

- **Security Audits:** Since the code management system is offered as a SaaS, it is an essential requirement that the providers (us) undergo security audits
- **Integrated Applications and functions within the code management system**
- **For a real-world project with real-world use cases (Not required in this case):**
  - **Physical device security for devices on the backend network.**
  - **High availability:** Availability records act as a credit score for service vendors (us).
  - **Multiple, secure, disaster-tolerant data centers**

## 6. Other Requirements

The following requirements are relevant only for applications with a large user-base. For the purposes of the OOADSE Lab project, satisfying the minimum requirements to keep the website running should be enough.

- **Database requirements:**
  - SQLite database. No dedicated machines required.
- **Reuse policies:**
  - Information scraped, obtained via the API or otherwise can be used for the following:
    - Make sure you understand what constitutes open source software and what doesn't.
    - Scraping refers to extracting information from the Service via an automated process, such as a bot or webcrawler. Scraping does not refer to the collection of information through our API.
    - Information obtained must not be used for spamming purposes, including for the purposes of sending unsolicited emails to users or selling User Personal Information to recruiters, headhunters, and job boards.

- **Privacy:**
  - Misuse of User Personal Information is prohibited.
  - Any person, entity, or service collecting any user personal Information from the system will only use that info provided it is secured, authorized and has active feedback to complaints and removal requests.
- **Excessive Bandwidth Usage:**
  - The system's bandwidth limitations vary based on the features used. If it is determined that the bandwidth usage of a user is significantly excessive in relation to other users of similar features, the system administrator has the right to suspend the account, throttle file hosting and limit activity.

## Appendix A: Glossary

- **VCS: Version Control System**
  - Version control is a class of systems responsible for managing changes to computer programs, documents, large web sites, or other collections of information.
- **SaaS**
  - **Software as a service** is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. It is sometimes referred to as "on-demand software"
- **SSH**
  - SSH or **Secure Shell** is a cryptographic network protocol for operating network services securely over an unsecured network.
- **NIC**
  - A **network interface controller** is a computer hardware component that connects a computer to a computer network
- **2FA**
  - 2-factor authentication is an extra layer of security used to make sure that people trying to gain access to an online account are who they say they are. First, a user will enter their username and a password. Then, instead of immediately gaining access, they will be required to provide another piece of information/
- **REQ:**
  - Requirement

## Appendix B: Field Layouts

[Link to spreadsheet](#)

- **User registration:**

1	User Registration				
2	Field	Length	Datatype	Description	Is Mandatory
3	Username	25	String	Identity of the user on the VCS	Y
4	Email Address	NA	alphanumeric		Y
5	Password	15 characters OR 8 (alphanumeric)	alphanumeric		Y
6	Email preferences	NA	Boolean	User has the option to opt for alerts, promotions and offers	N
7	Captcha	NA	image	Puzzle to prevent bots from trying to register on the system	Y
8	Profession	NA	Boolean (Checkboxes)	Current Employee Designation	N
9	Programming Experience	2	numeric	User has the option to specify programming experience in years for personalized recommendations	N
10	Interests	20	List of strings	List of tools and frameworks the user is familiar with. Added to user profile and for personalized content	N
11	Verification code	5	numeric	Randomly Generated Two-Factor Code	Y
12					

- **User profile information:**

User Profile Info			
Fields	Data Type	Length	Description
Account ID	Alphanumeric	16	Unique ID that identifies the user across the system
Account Name	String	30	Username
Profile Photo	Image	NA	
Preferred Language	Boolean (Checkbox)	NA	
Repositories	Numeric	NA	List of repositories created, starred, forked or collaborated with
Archived Repositories	Numeric	NA	List of repositories archived
Bio	String	255	
Followers	List of Strings	NA	List of followers
Following	List of Strings	NA	List of users being followed
Starred	List	NA	List of Starred repositories
Contributions	List of strings	NA	List of contributions made to personal and non-personal public and/or private repositories.

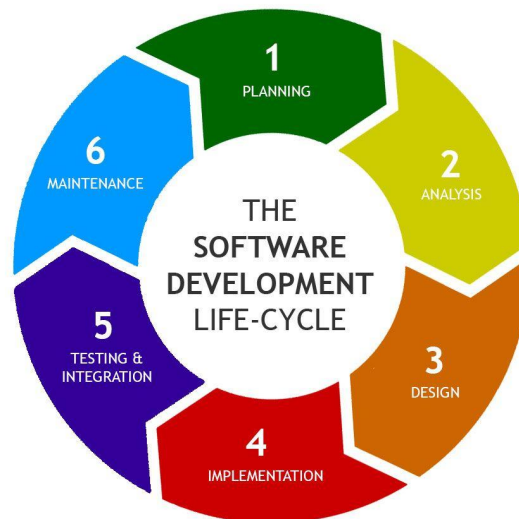
## Appendix C: Requirement Traceability Matrix

Sl. No	Requirement ID	Brief Description of Requirement	Architecture Reference	Design Reference	Code File Reference	Test Case ID	Test Case Status
1	R01	Correct Login	1,2,3	3.4.2,3.5.2	app.py@login	UT_2	Pass
2	R02	Incorrect Login	1,2,3	3.4.2,3.5.2	app.py@login	UT_3	Pass
3	R03	Register into the system	1,2,3	3.4.2,3.5.2	app.py@sign_up	UT_1	Pass
4	R04	Create Repository	1,2,3	3.4.3,3.5.3	app.py@create_repository	IT_1	Pass
5	R05	Manage your repository	1,2,3	3.4.3,3.5.3	app.py@manage	IT_2	Pass
6	R06	Managing accesses to the repository(Owner)	1,2,3	3.4.3,3.5.3	app.py@add_contributor	UT_17	Pass
7	R07	Commit; without conflict	1,2,3	3.4.1,3.5.1	app.py@upload_file	UT_9	Pass

## CHAPTER 2

### PROJECT PLAN

## Lifecycle



### 1) Requirements Gathering and Analysis

- **Hybrid Model** can be used here since the project consists of components which are dependent of previous phases of development (Repository Management and Profile Management are dependent on User Auth and access control) as well as components that can be worked on independently (Comments section, Search, Database creation)
- The end user requires a web app for a code management system with features like sharing files, managing repositories, committing changes, and tracking issues.
- Inputs by the developer team should include feature approval and modification, development time specifications, tech stack constraints etc.
- Upon approval, the [SRS Document](#) is created. Once the final SRS document is approved, the project moves on to the design phase.

### 2) Design

- Feature and functional requirements are specified and finalized.

- Mockups of the intended platform are created by the design team and are reviewed by developers, product managers and a small test set of customers. Use prototyping tools such as Bootstrap Studio to create design mockups.
- The test set of customers further give their feedback which are added onto the design according to the needs of the majority.
- These design ideas are constantly improved upon through various numbers of phases.

### **3) Implementation & Testing**

- Provide updates to product managers.
- A test-driven approach is followed where a feature or a part of it is created, tested, reviewed and deployed.
- Frequent updates are provided and they are tested for the same using tools such as Postman to verify that each of the API calls are responded to as intended.

### **4) Deployment**

- Deployment in software or web development means to push changes or updates from one deployment environment to another.
- After the web application is production-ready, you want the ability to make changes to the web application in real-time. Such environments are called production environments, or deployment environments.
- The initial development environments will essentially be set up as a local environment. Such environments are called development or staging environments. These are typically non-production-ready.

### **5) Maintenance**

- The Maintenance Phase occurs once the system is operational.
- It includes implementation of changes that software might undergo over a period of time, or implementation of new requirements after the software is deployed at the customer location.

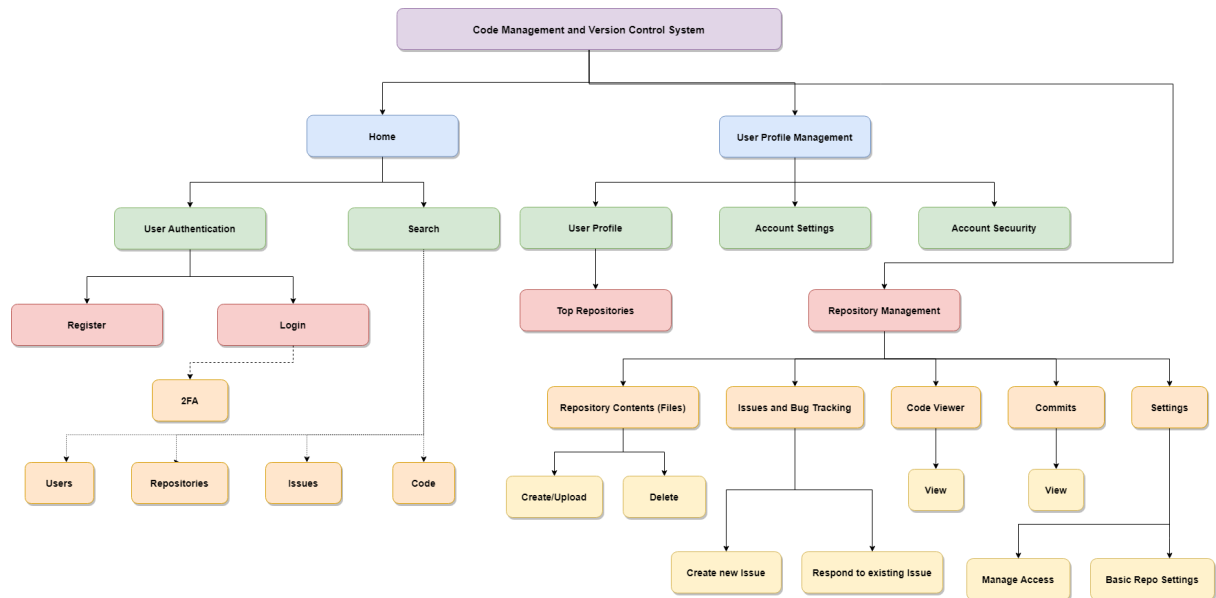


# Tools

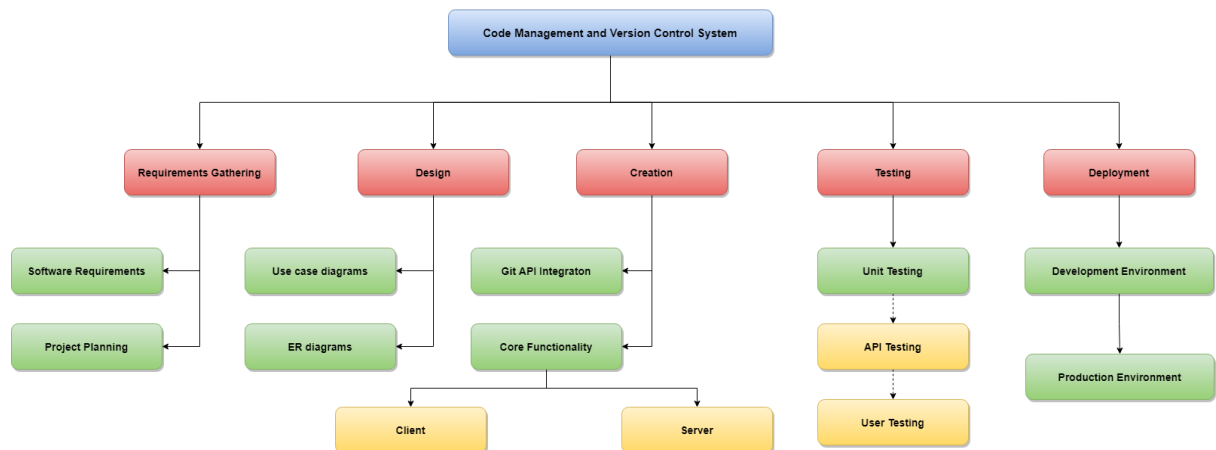
Project Phase	Tools Required
Analysis and Planning	Draw.io, StarUML, Google Sheets, Google Docs
Detailed Design	Google Sheets, Google Docs, StarUML, Bootstrap Studio
Code Review	Atom
Implementation	Atom, Flask, HTML, CSS, JavaScript, Bootstrap, Cryptography.Fernet, SQLAlchemy
Testing	Postman, Google Chrome
Version Control	Git

# Work Breakdown Structure

## Functionality WBS



## Workflow WBS



**Determine all the deliverables and categorise them as reuse/build components and justify the same.**

**Deliverables:**

Build	Reuse
User Authentication	Search bar
Repository Management	Code viewer
Profile Management	Comments section
CRUD operations on database	

**Reusable components:**

- The search bar is universally accessible throughout the website and developers will only need to use a vendor such as HubSpot or Google to implement the search functionality once on the landing page. This component can be reused across the website
- The code viewer need not be implemented from scratch. Editing existing implementations of web-based source code viewers to view files across the code management system will suffice.
- The comments section will again be common to the “issues” section and the “commit” section and can thus be reused.

All other components of the web app involving user authentication, repository-specific settings, implementing commits to the repo resulting in updation of the database and managing user profiles will need to be built from scratch.

## Do a rough estimate of effort required to accomplish each task in days.

Assuming a deadline after 12 weeks, given below is a rough estimate of the effort required to accomplish each task in days:

1. Planning and Learning Phase
  - a. Basic idea of a code management and version control system. **[4 days]**
  - b. Deciding on the tech stack to be used to build the platform. **[4 days]**
  - c. Going through documentations for the tech stack chosen and the required JavaScript libraries. **[7 days]**
  - d. Deciding on the overall application design. **[3 days]**
2. Implement the register and login pages vital for user authentication along with placeholders for the remaining components. **[7 days]**
3. Implement the user dashboard which has a list of the user's top repositories along with a search bar at the top of the page allowing users to search for public code, issues, repositories and other users on the platform. **[7 days]**
4. Implement the repository-specific dashboard (Create and delete files, commit list, and repository-specific settings). (excluding issues and bug tracking) **[7 days]**
5. Fully implement issues (along with the Q&A section where fellow developers can discuss on the concerned issues) and bug tracking under the repository-specific dashboard. **[7 days]**
6. Implement the code viewer allowing users to view the code contained in a file under the repositories. **[4 days]**
7. Implement user-specific settings. **[4 days]**
8. Testing **[2 days]**
9. Fixing critical bugs (if encountered upon testing). **[3 days]**
10. Deployment **[1 day]**

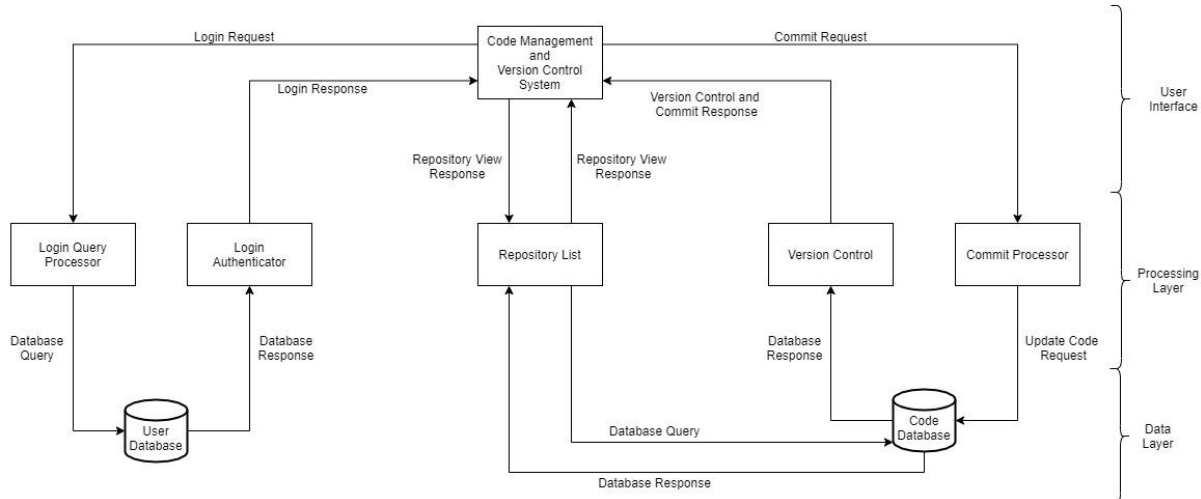
## Create the Gantt chart for scheduling the previously defined tasks.

[Code Management and Version Control System Gantt Chart](#)

# CHAPTER 3

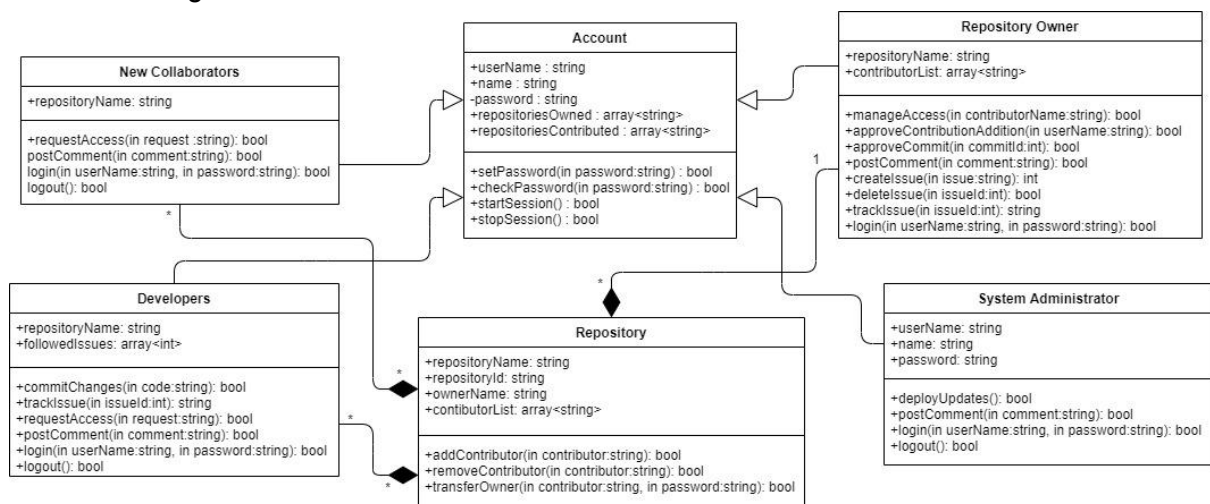
## DIAGRAMS

### 3.1 Architecture Diagram



This diagram depicts the overall architecture used in this project. It shows the relationships and connections between the logical components of the system. As seen in the diagram, there are three parts of the system, namely, user interface, processing layer and the data layer. The user interface is where the users can interact with the proposed system. They will have a choice to login, then see their repository list, version control and make commits. These operations happen in the processing layer. All the data required will be divided into 2 databases- User database and the code database. These databases constitute the data layer. Communication between these layers is the key to success in this project as it will let the functioning to be split across layers for seamless functioning and modularity of the system.

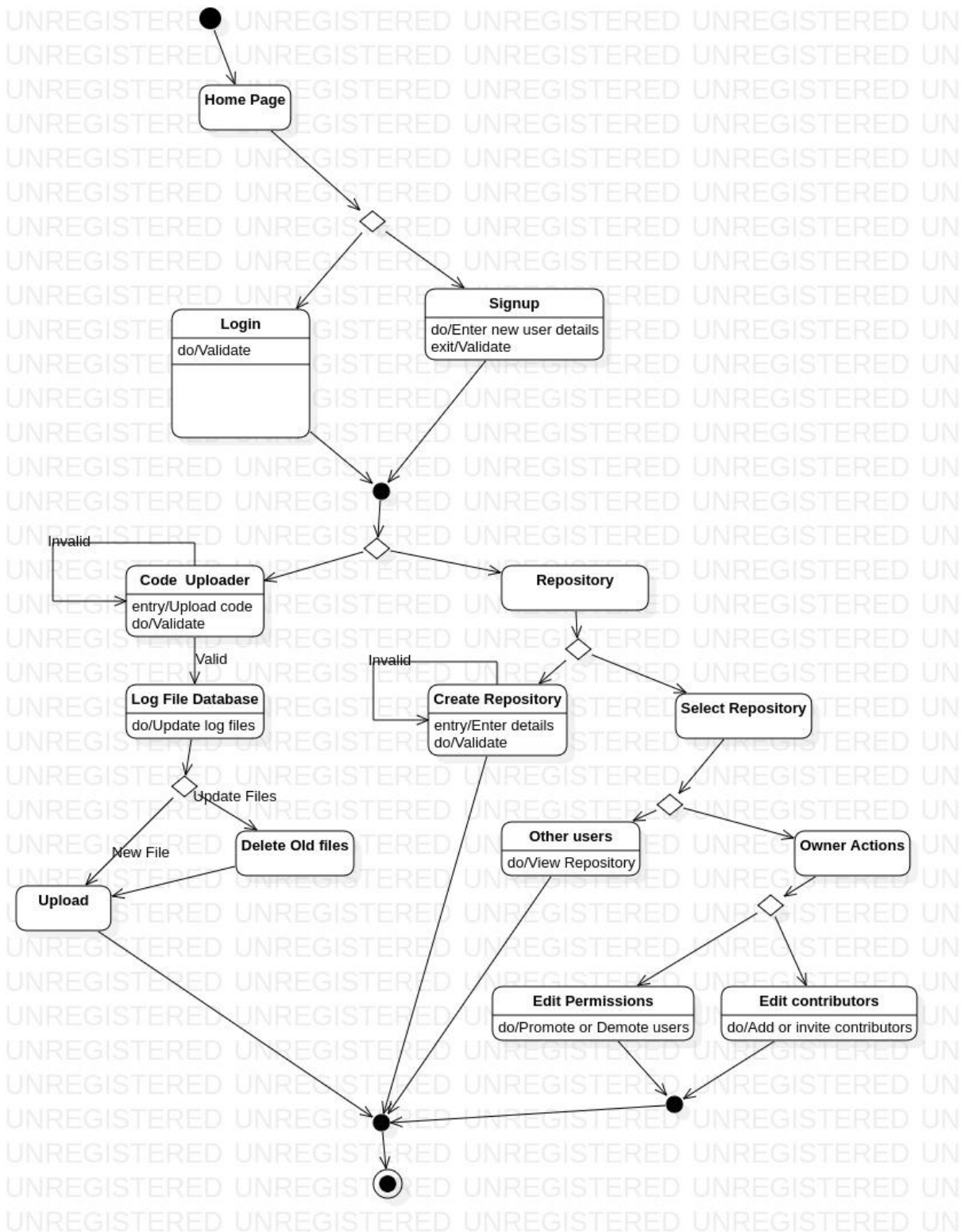
### 3.2 Class Diagram



This diagram, known as the class diagram, explains the system using object oriented techniques, i.e., it has classes and these classes can communicate with each other. In the

case of this system, there are six classes used, namely- New Collaborators, Account, Repository Owner, Developers, Repository and System Administrator. Each class has some attributes and behaviours (functions) with it. For example, the Account class has userName, name, password, repositoriesOwned and repositoriesContributed as its attributes where password is a private attribute and the other attributes are all public. The type of these attributes is given next to the attribute name, followed by a colon. The behaviours associated with that class are setPassword, checkPassword, startSession and stopSession. These functions may or may not have parameters, like the first two of these functions have a parameter and the last two do not have. All the functions have a return type that is given after the respective function name, followed by a colon. The relationships between the classes is defined by the type of relation and the multiplicity involved in the relationship.

### 3.3 State Diagram



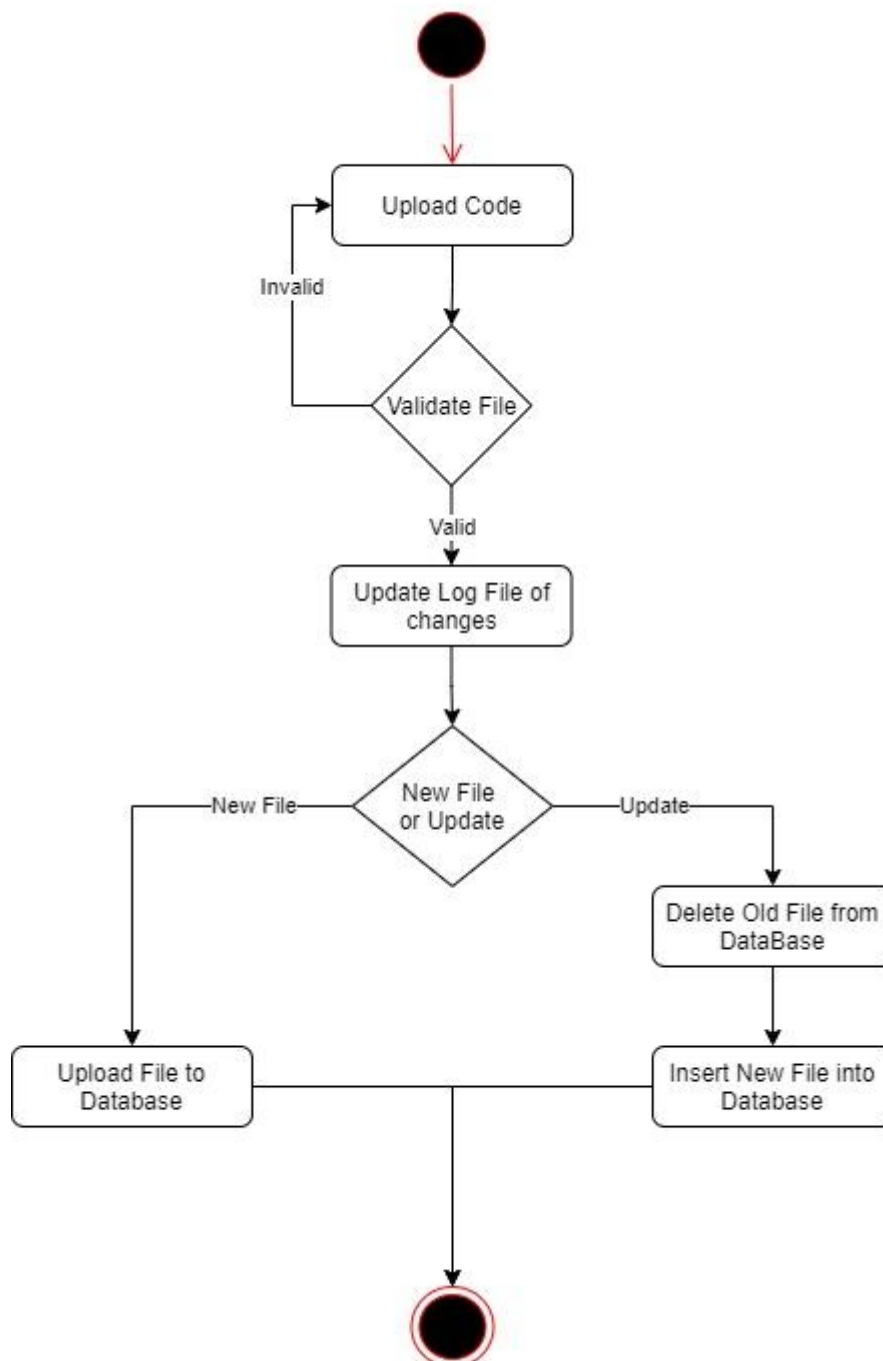
In the above state diagram the user sees the homepage and either performs a login or signup function based on whether he is new or not to the website. After the users logs in the user can click on repositories to either create a repository or select a repository. If the user chooses the select repository then the user has to get permission from the owner and the owner can perform actions to edit permissions and collaborators.

If the code has to be uploaded to the database then all the log cases will also be updated parallelly in the database. Once that is done it either creates a new file in the database if

there is no existing file else if there are existing files then the old files will be replaced by the new files in the database.

### 3.4 Activity Diagram

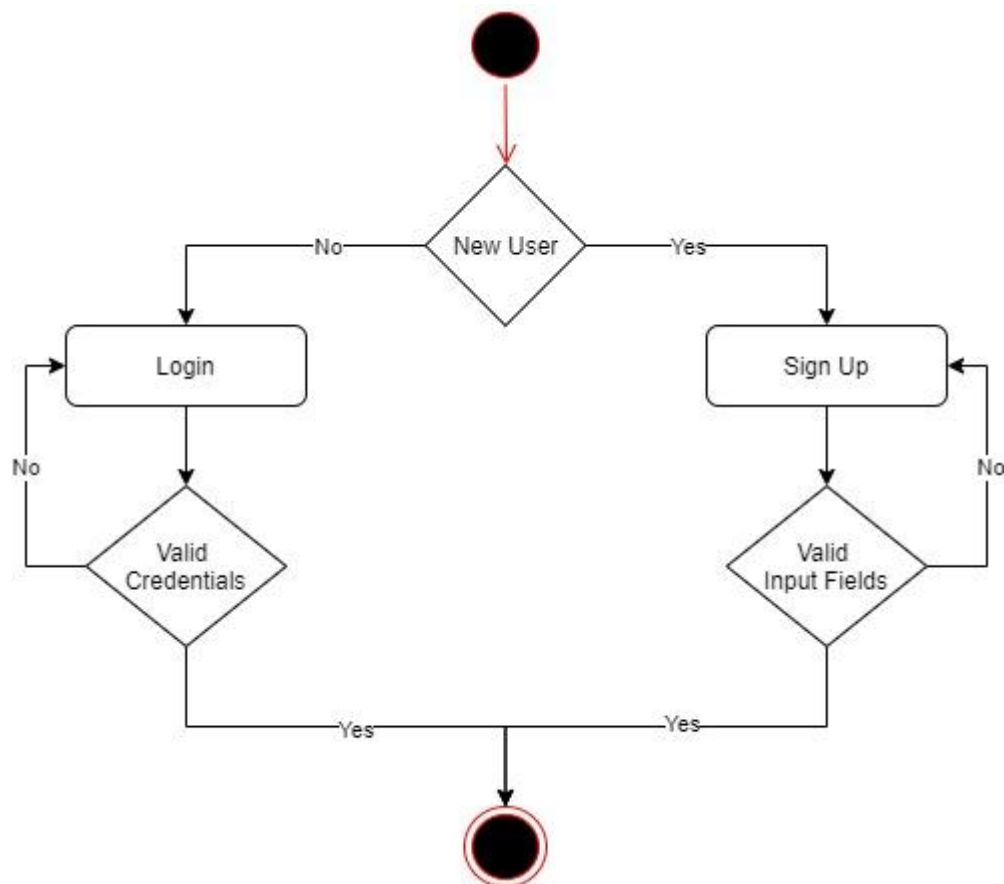
#### 3.4.1 Manage Code



The above activity diagram depicts uploading of code files into the database. Firstly the user uploads the code and the validator checks if it is valid or not. If it is invalid then it goes back to the previous step and if it is valid then it updates the log files and applies the changes needed. If it is a new file then it uploads the new file into the database. If it is an existing file then it deletes the older files from the database and inserts the new file into the database.



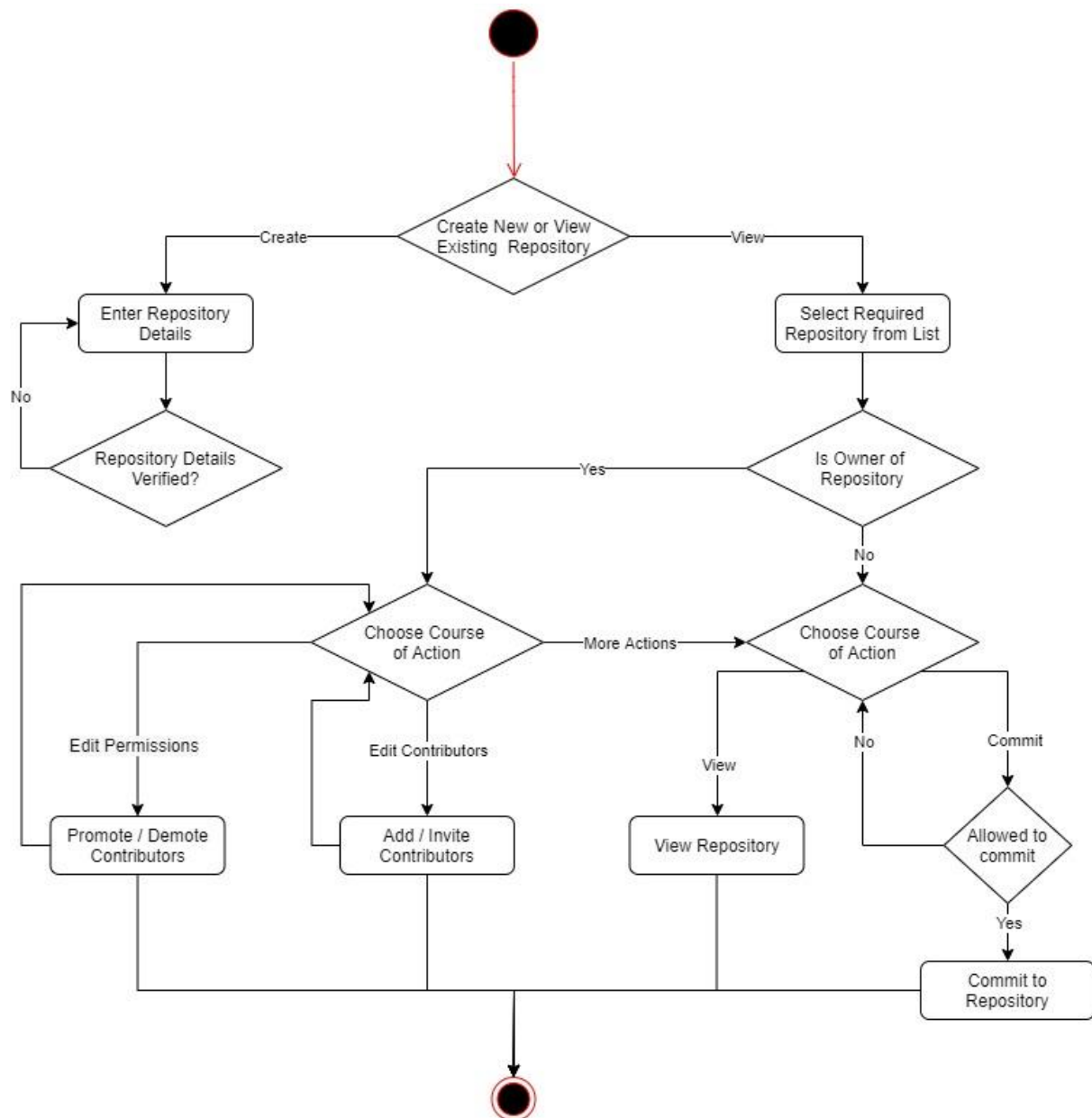
### 3.4.2 Authentication



In the above activity diagram first the system checks if the user is existing or not. If it is a new user then the system asks the user to sign up. Once the user gives the credentials his account gets successfully created, in case the user gives credentials that do not match the system credentials it asks them to re-enter.

If the user is existing then the system goes to the login page and the user is expected to give the user credentials. Once the user credentials are met by the system's requirement then the user is sent to the home page.

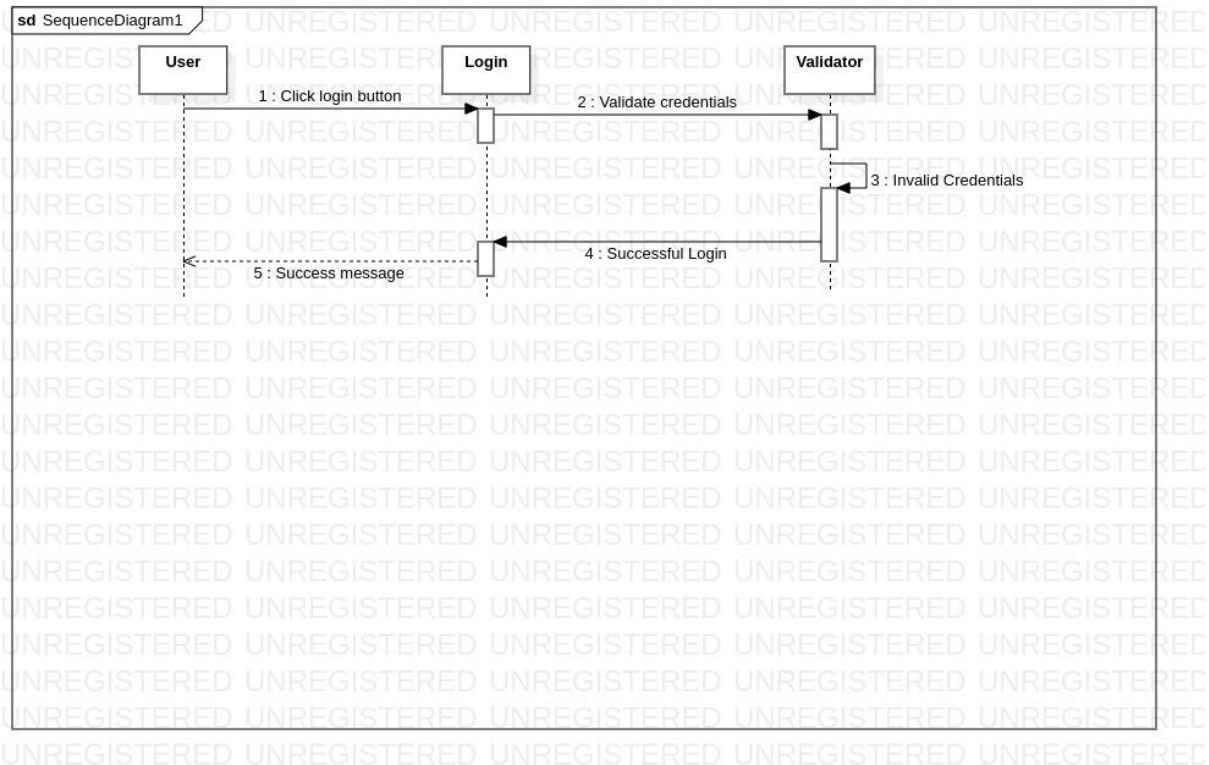
### 3.4.3 Repository activity and management



In the above activity diagram, if there is no repository then it creates a new repository and asks for the repository details. The system then asks for the verification of the details. If the repository already exists then the user views the existing repository. From the repository list the user can choose upon choice. If the user is the owner of the repository then he can choose a course of action and add members or invite collaborators. The user can also edit permissions by promoting or demoting the collaborators. Alternate actions are that he can view the repository and commit to the repository. If the user is not the owner of the repository then he can view the repository or he can commit to the repository and once the permission is granted by the admin of the repository the user is allowed to commit to the repository.

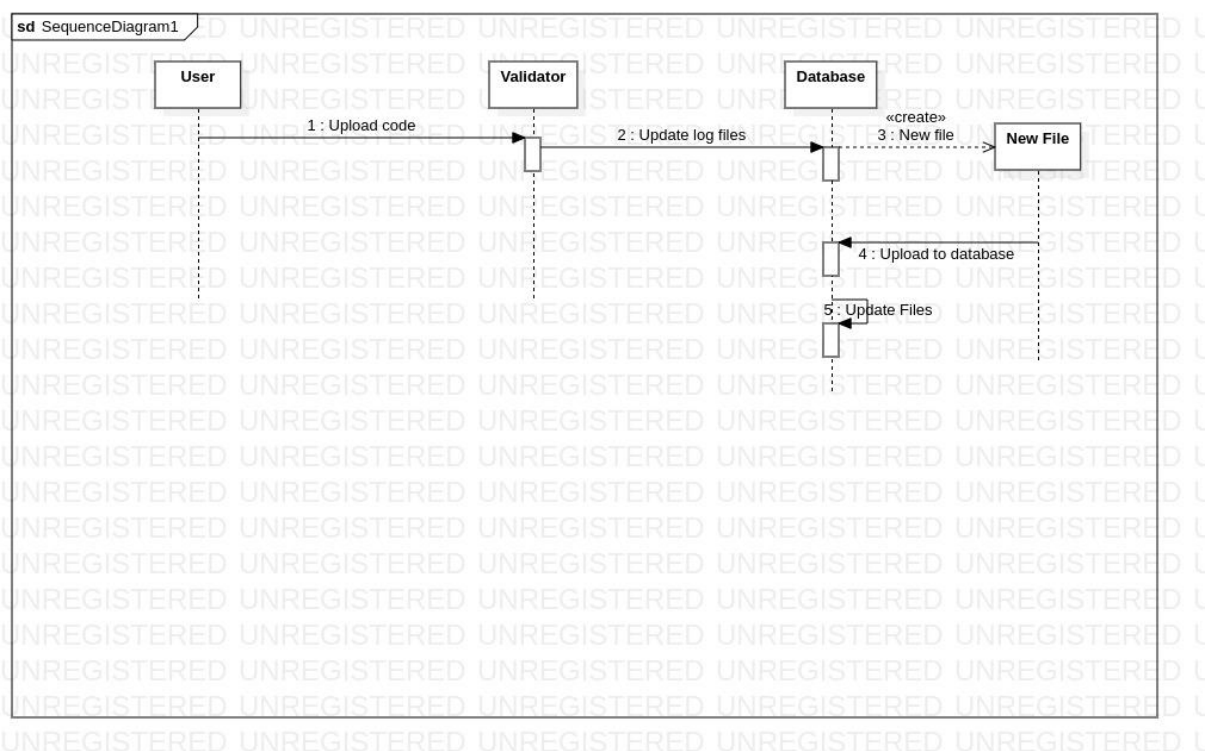
### 3.5 Sequence Diagrams

#### 3.5.1 Authentication



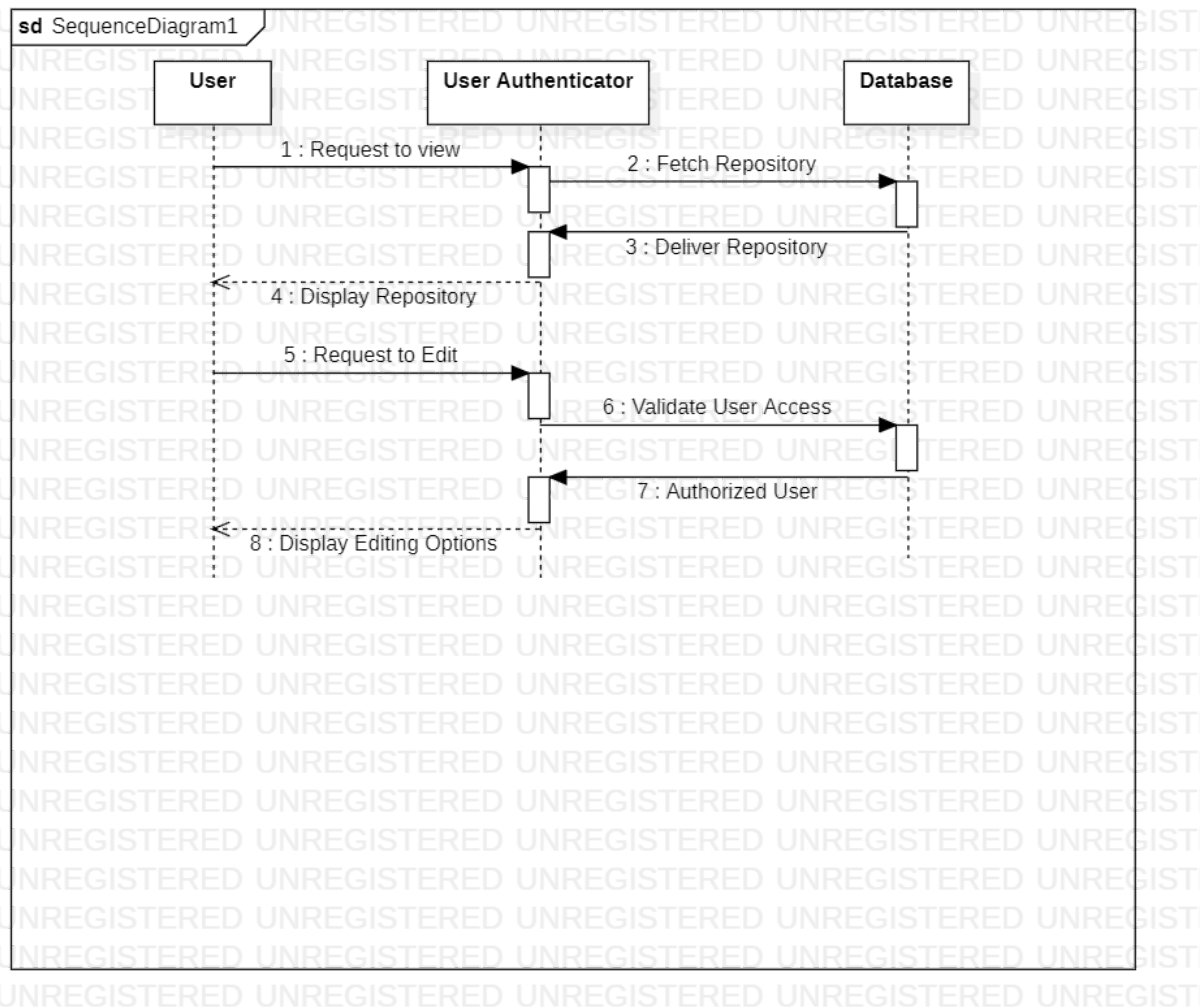
In the above diagram the user clicks on the login button to login to the website. The validator then checks all the credentials of the user. If the user has given incorrect credentials then the user has to re-enter the credentials. If the user credentials are correct then the validators send a successful login message and a success message is displayed on the user's screen.

### 3.5.2 Manage Code



In the above diagram the user uploads the code and the validator updates the log files into the database. There is a new file created in the database if none exists and it stores it in the database. If there is an existing file it just updates these files into the database.

### 3.5.3 Repository management



In the above diagram the user requests for a user authentication for viewing the repository, once he gets the permission the system then fetches the files from the database. Then the user will be able to view the displayed repository that he requested from the database. The user then requests permission to edit the files. Then the system validates the user's access from the database and authorizes permission to the user and then displays the editing options to the user.

## **CHAPTER 4**

### **MODULE DESCRIPTION**

#### **4.1 Create Repository**

Using create repository the user can create a new repository. Once the user creates a new repository he fills in the repository name, description for the given repository and the number of collaborators to the repository. If the collaborators are less than 0 then it throws an error saying that the number of collaborators must be 1 or more. If the collaborators are greater than 11 then the system sends an error message saying that the number of collaborators cannot be greater than 10. So once the user specifies the number of collaborators he then gets the names of those who are contributing to the repository. All this happens only when the user logs in to his account otherwise this functionality doesn't work.

#### **4.2 Manage Code (Push, Pull)**

The code can be pushed onto an existing repository or a new repository based on whether there is an existing repository or not. Once the code is pushed into a new repository it can be viewed and also the user can pull the repository. If this is pushed into an existing repository then there is a timestamp created as to when the original was posted and when the new updated files have been uploaded into the database. There is a regular maintenance check if log files are updated and all the code is up to date. This is only functional when the user creates an account and has a repository or creates a new repository.

#### **4.3 Collaboration (Share and edit code)**

Collaborators can be added both during and after creation of a repository. Only the owner has permission to add collaborators to the repository. All the collaborators are allowed to edit the code and push new code. All changes are recorded in the logfiles of the repository. No user has access to the log files and hence it can not be tampered with.

#### **4.4 View Change History**

Whenever a code is edited or a new code is pushed to a repository the log file of the repository is updated and the timestamp and user doing the change is recorded. This change history is always shown on the view page of all files. Only the logs of that particular file are viewed.

## CHAPTER 5

### TEST CASES

Test case ID	Name of module	Test case description	Pre-conditions	Test steps	Test data	Expected results	Actual result	Test Result
Create Repository:								
UT_1	User registration module	Test the sign up functionality	Access to chrome browser	1. Click on signup on the website 2. Enter user credentials 3. Enter Submit	Username: test email: abc@gmail.com password: abc12345	Sign up successful with home page shown	Sign up Successful with home page displayed	Pass
UT_2	User Login module	To test the login functionality	Access to chrome browser	1. Existing user 2. User name 3. Password	Username: test email: abc@gmail.com password: cab1345	Unsuccessful login with a try again message	Unsuccessful login with a try again message	Pass
UT_3	Invalid Password	To test the for valid passwords- should have both alphabets and numbers	Access to chrome browser	1. Click on signup on the website 2. Enter user credentials with password as abc 3. Enter Submit	Username: test email: abc@gmail.com password: abc	Enter valid password with alphabets and numbers	Enter valid password with alphabets and numbers	Pass
UT_4	Email verification	Entering a valid email id for functionality	Access to the website	1. Click on email and enter an address during submission	Username: test email: abc121 password: abc12345	Enter a valid email address	Invalid email id, enter another address	Pass
IT_1	Create new repository	Creating and upload a new repository	User login to be successful	1. Press create repository 2. enter repository name 3. Enter valid description 4. Enter no. of contributors 5. Click Submit button	Enter details for the repository Repo Name: Codehub Description: OOADSE Lab Number of	Created successfully and redirected to the manage webpage	Created successfully and redirected to the manage webpage	Pass

					contributors: 0			
UT_5	No of Contributors for the repository	Entering a valid number for the no. of contributors for the repository	1. User login to be successful 2. After pressing the create repository button	1. enter repository name 2. Enter valid description 3. Enter no. of contributors as a number greater than 10 4. Click Submit button	Enter details for the repository Repo Name: Codehub2 Description: OOADSE Lab Number of contributors: 11	no of contributors > 10 enter valid number Not created successfully	no of contributors > 10 enter valid number Not Created Successfully	Pass
UT_6	Repository name with a slash (/)	Entering a name with slash in repository name while creating a repository	1. User login to be successful 2. After pressing the create repository button	1. enter repository name with a slash in between 2. Enter valid description 3. Enter no. of contributors as a number lesser than 10 4. Click Submit button	Enter details for the repository Repo Name: Code/hub Description: OOADSE Lab Number of contributors: 5	Created successfully and redirected to the manage webpage	URL not found is displayed	Fail
UT_7	Existing repository	Creating repository with an existing name	1. User login to be successful 2. After pressing the create repository button	1. enter repository name 2. Enter valid description 3. Enter no. of contributors 4. Click Submit button	Repo Name: Codehub Description: OOADSE Lab Number of contributors: 3	Repository name is unavailable	Repository name is unavailable	Pass
Manage Code:								
IT_2	Manage repository	Managing files in existing repository	Repository must be created	1. Click on repositories in Nav bar 2. Click on any pre-existing repository	None	Should be redirected to the "Manage" page of the repository	Was redirected to "Manage" page of the repository	Pass

UT_8	Push existing repository_ success	Uploading a new code/file into the repository	Repository must be created	1. Click on Upload file. 2. Select a new file to upload 3. Click on upload	New file to be uploaded	1. Should be redirected to the "Manage" page of the directory in the repository 2. The new file should be visible in the directory it was uploaded in	1. Was redirected to the "Mange" page of the directory in the repository 2. The file uploaded was visible in its respective directory	Pass
UT_9	Push existing repository_ success	Uploading existing code/file into the repository	1. Repository must be created 2. File must already be uploaded to the directory	1. Click on Upload File 2. Select an already existing file to upload 3. Click on upload	Existing File to be uploaded	1. Should be redirected to the "Manage" page of the directory in the repository 2. The new file should be visible in the directory it was uploaded in 3. When Clicked on the file it should show an entry with timestamp saying that it was edited by the current user	1. Was redirected to the "Mange" page of the directory in the repository 2. The file uploaded was visible in its respective directory 3. When clicked on the file it shows that the file was edited by the current user	Pass
UT_10	Push to existing repository_ no upload	Clicking upload without uploading a file	Repository must be created	1. Click on Upload file. 2. Do not select a new file to upload 3. Click on upload	None	1. Please choose a file to upload 2. Go back to give a choice to	It redirects to an error page called "IsADirectoryError"	Fail



						the user		
UT_11	Pull repository_success	Downloading a file from the repository	1. Repository must be created 2. File should already exist in the Repository	1. Click on the File to be pulled 2. On the "View" webpage, click on the download button beneath the code	None	1. Should be redirected to the "View" webpage of the file. 2. On clicking download, it should download the file into the users system	1. Was redirected to the "View" webpage of the file 2. Was asked if file was to be downloaded.	Pass
IT_3	View Code	Viewing existing code in the repository	1. Repository must be created 2. File must already be uploaded to the directory 3. Repository must have code	1. Go to the repository 2. Click on a file that has been uploaded	None	1. Should be redirected to "View" webpage of the file. 2. Left side should display the code/file. 3. Right side will show the Change History	1. Was redirected to the "View" webpage of the file 2. Left side displayed the code in the file. 3. Right side showed the change history of the file	Pass
UT_12	View HTML file	Viewing an html file uploaded in repository	1. Repository must be created 2. Repository must have HTML file	1. Go to the repository 2. Upload a HTML file 3. View the file	HTML file	1. Should be redirected to the "Manage" webpage once its uploaded 2. Should show the code in the HTML file while viewing it.	1. Was redirected to the "Manage" webpage after the file was uploaded 2. The webpage of the HTML file was displayed in its "View" page instead of	Fail

							its code	
UT_13	Maintainance repository	Maintaining files in the repository	1. Three accounts must be created 2. Repository must be created	1. Create a repository from one account 2. Upload the same file from other two accounts.	Any file	The view page of the file should display all three log entries	The View page of the file displays that the file was created by the first user and two entries from the other two.	Pass
UT_14	View Code_empty	Empty code uploaded in the repository	Repository must be created	1. Click on upload file 2. Upload an empty file 3. Click on the file	Empty file to be uploaded	1. Should upload the file successfully 2. When clicked on the file, it should display an empty left side and a log entry stating that the file was created by the user	1. The file was uploaded successfully. 2. When clicked on the file, the left side was empty and the right side displays a log entry stating that the user created the file.	Pass
IT_4	Logout	Logout user from session	User must be logged into the webapp	1. Click on Log Out	None	1. Should Logout the user 2. Should redirect user to Home page	1. User was successfully logged out 2. User was redirected to Home page	

Collaboration:

UT_15	Add contributor successful - 1	Add contributor while creating repository that does not exist	User 1 and user 2 should exist	<ol style="list-style-type: none"> <li>1. Create a user</li> <li>2. Create another user</li> <li>3. Click on create repository for user 1</li> <li>4. Enter repo name, description, number of contributors = 1</li> <li>5. Add the username of user 2</li> <li>6. Click on create</li> </ol>	Username: test email: abc@gmail.com password: abc12345  Username: test2 email: abc2@gmail.com password: abc12345  Repository name: Codehub Description: OOADSE Contributors: 1 Add Contributor name as test2	Created Successfully	Created Successfully	Pass
UT_16	Add contributor unsuccessful - 1	Add contributor who doesn't exist while creating repository that does not exist	User 1 should be created	<ol style="list-style-type: none"> <li>1. Create a user</li> <li>2. Click on create repository for user 1</li> <li>3. Enter repo name, description, number of contributors = 1</li> <li>4. Add the username of a user who doesn't exist</li> <li>5. Click on create</li> </ol>	Username: test email: abc@gmail.com password: abc12345  Repository name: Codehub Description: OOADSE Contributors: 1 Add Contributor name as test2	Creates repository but doesn't add contributor Displays a message saying Invalid contributors: test2	Creates repository but doesn't add contributor Displays a message saying Invalid contributor s: test2	Pass

UT_17	Add contributor successful - 2	Add contributor while creating repository that exists	User 1 and user 2 should exist	<ol style="list-style-type: none"> <li>1. Create a user</li> <li>2. Create another user</li> <li>3. Click on create repository for user 1</li> <li>4. Enter repo name, description, number of contributors = 0</li> <li>5. Click on create</li> <li>6. Click on Add contributor</li> <li>7. Enter number of contributors as 1</li> <li>8. Enter user name as user2</li> <li>9. Click on add</li> </ol>	<p>Username: test email: abc@gmail.com password: abc12345</p> <p>Username: test2 email: abc2@gmail.com password: abc12345</p> <p>Repository name: Codehub Description: OOADSE Contributors: 1 Add Contributor name as test2</p>	Added successfully	Added successfully	Pass
UT_18	Add contributor unsuccessful - 2	Add contributor who doesn't exist while creating repository that exists	User 1 should be created	<ol style="list-style-type: none"> <li>1. Create a user</li> <li>2. Click on create repository for user 1</li> <li>3. Enter repo name, description, number of contributors = 0</li> <li>4. Click on create</li> <li>5. Click on Add contributor</li> <li>6. Enter number of contributors as 1</li> <li>7. Enter user name of a user who doesn't</li> </ol>	<p>Username: test email: abc@gmail.com password: abc12345</p> <p>Repository name: Codehub Description: OOADSE Contributors: 1 Add Contributor name as test2</p>	Displays a message saying Invalid contributors : test2	Displays a message saying Invalid contributors: test2	Pass

				exist 8. Click on add				
UT_19	Downloading a file - contributor	Non-owner user download	1. Two users should be created 2. User 2 should be a contributor to the repository created by User 1 3. Repository should have a file	1. Create a user 2. Create another user 3. Click on create repository for user 1 4. Enter repo name, description, number of contributors = 1 5. Add the username of user 2 6. Click on create 7. Upload code 8. Log in as user 2 9. Click on the code and download	Username: test email: abc@gmail.com password: abc12345  Username: test2 email: abc2@gmail.com password: abc12345  Repository name: Codehub Description: OOADSE Contributors: 1 Add Contributor name as test2  Upload file: any	Successful	Successful	Pass
UT_20	Editing a file - 1	Contributor edits a file created by owner	1. Two users should be created 2. User 2 should be a contributor to the repository created by User 1 3. Repository should	1. Create a user 2. Create another user 3. Click on create repository for user 1 4. Enter repo name, description, number of contributors = 1 5. Add the username of user 2 6. Click on create	Username: test email: abc@gmail.com password: abc12345  Username: test2 email: abc2@gmail.com password: abc12345  Repository name: Codehub	Shows Created by test with timestamp and Updated by test2 with timestamp	Shows Created by test with timestamp and Updated by test2 with timestamp	Pass

			have a file	7. Upload code 8. Log in as user 2 9. Upload the new file with same name	Description : OOADSE Contributors: 1 Add Contributor name as test2  Upload file: any			
UT_21	Editing a file - 2	Owner edits a file created by contributor	1. Two users should be created 2. User 2 should be a contributor or to the repository created by User 1 3. Repository should have a file	1. Create a user 2. Create another user 3. Click on create repository for user 1 4. Enter repository name, description, number of contributors = 1 5. Add the username of user 2 6. Click on create 7. Upload code as user 2 8. Log in as user 1 9. Upload the new file with same name	Username: test email: abc@gmail.com password: abc12345  Username: test2 email: abc2@gmail.com password: abc12345  Repository name: Codehub Description : OOADSE Contributors: 1 Add Contributor name as test2  Upload file: any	Shows Created by test2 with timestamp and Updated by test with timestamp	Shows Created by test2 with timestamp and Updated by test with timestamp	Pass

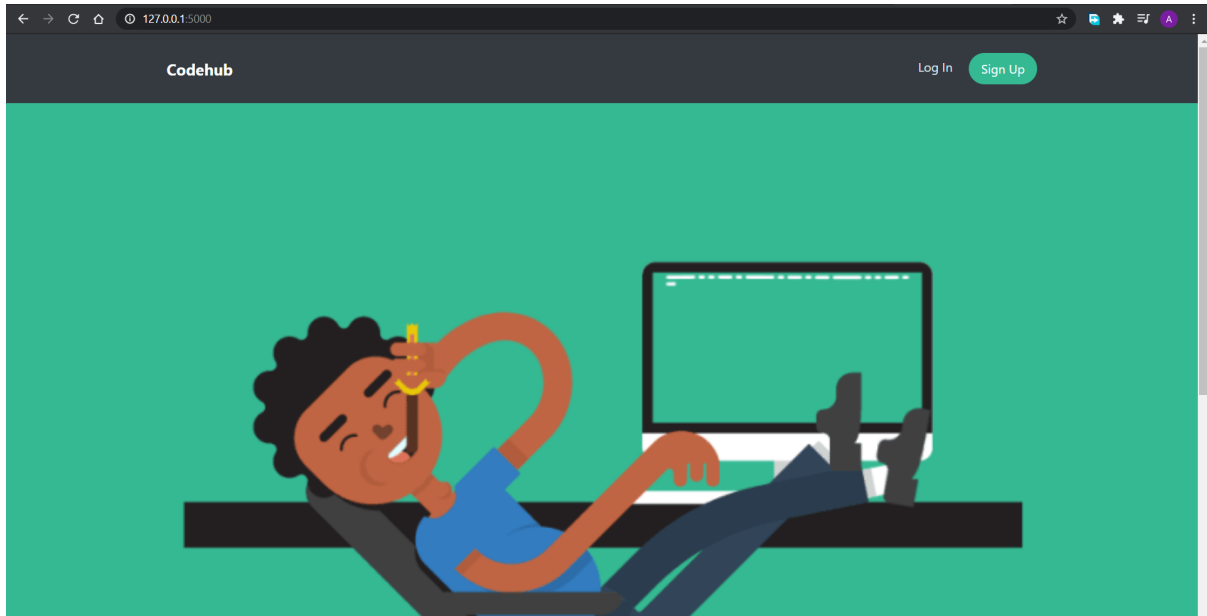
ST_1	Adding a contributor - non owner account	Trying to add a new contributor by a user who is not the owner of the repository	1. Two users should be created 2. User 2 should be a contributor to the repository created by User 1	1. Create a user 2. Create another user 3. Click on create repository for user 1 4. Enter repo name, description, number of contributors = 1 5. Add the username of user 2 6. Click on create 7. Login as user2 and there will be no option to add contributors	Username: test email: abc@gmail.com password: abc12345  Username: test2 email: abc2@gmail.com password: abc12345  Repository name: Codehub Description: OOADSE Contributors: 1 Add Contributor name as test2	No option to add contributors	No option to add contributors	Pass
------	--	--	---	---	---	-------------------------------	-------------------------------	------

## CHAPTER 6

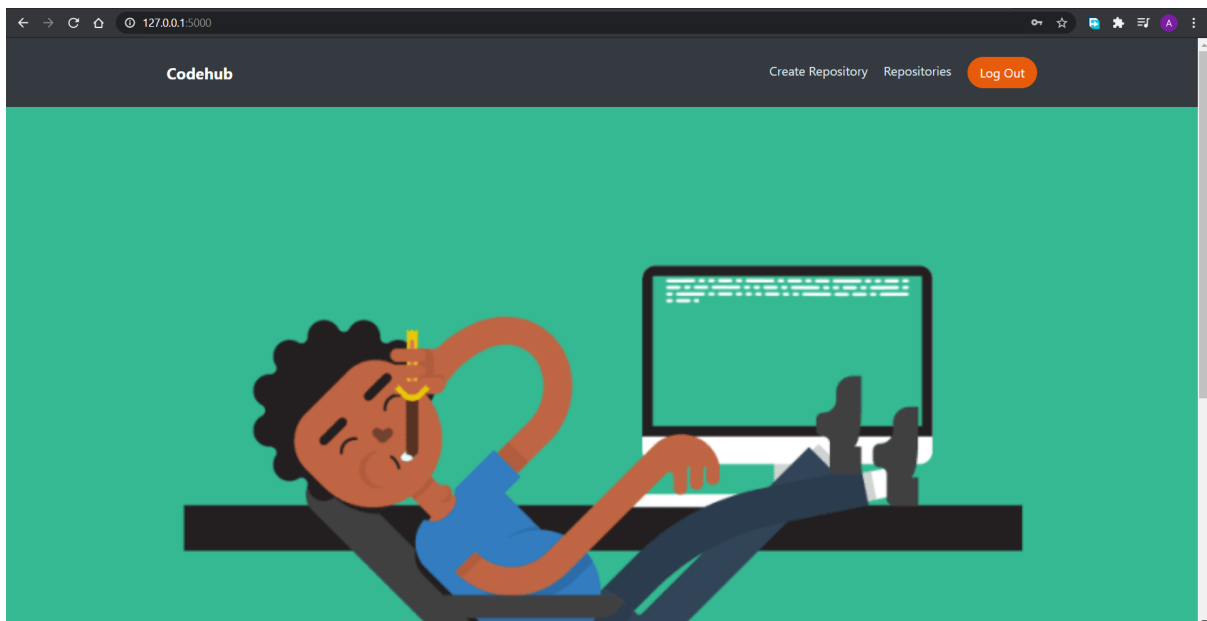
### OUTPUT SCREENSHOTS

#### 1. HOME PAGE

##### a. Before Login

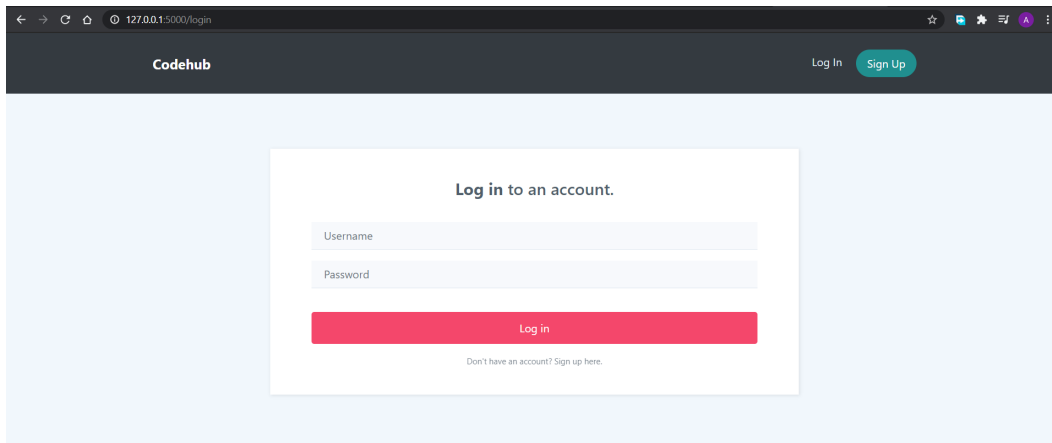


##### b. After Login



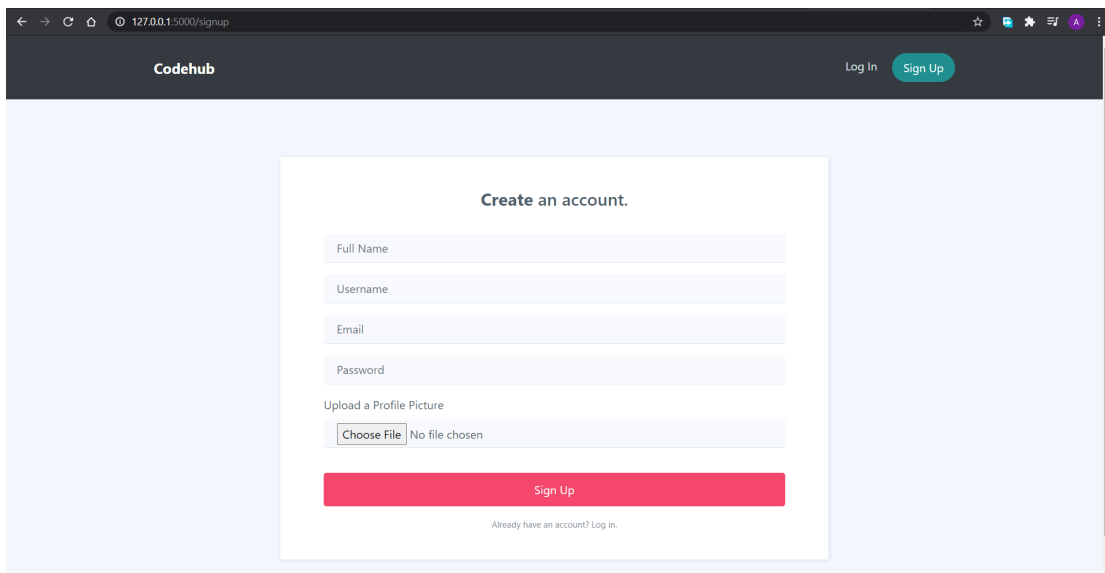


## 2. LOGIN PAGE



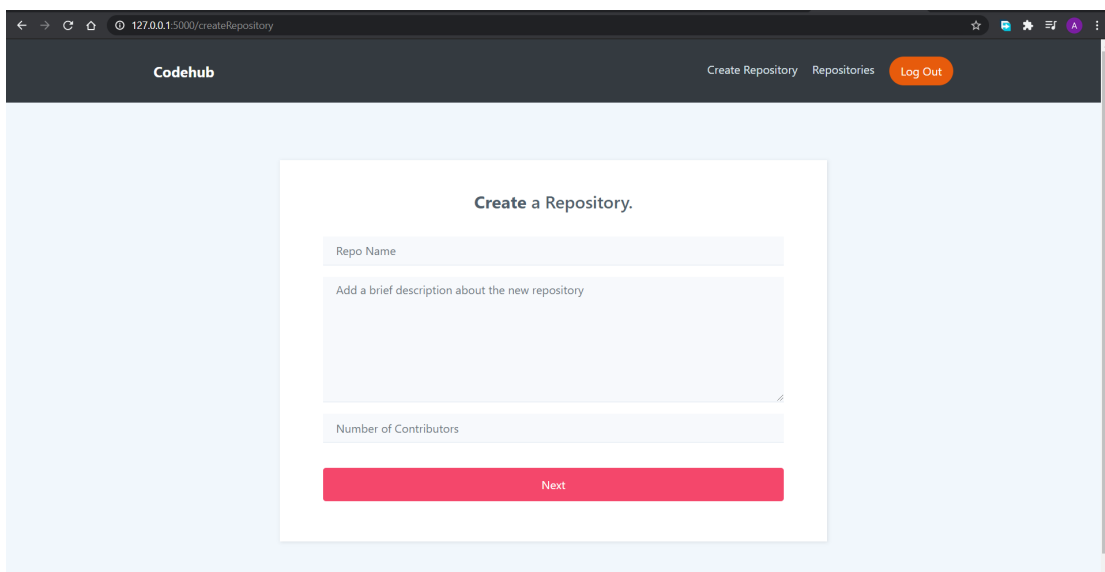
The screenshot shows a web browser window with the URL `127.0.0.1:5000/login`. The page has a dark header with the 'Codehub' logo on the left and 'Log In' and 'Sign Up' buttons on the right. The main content area is light blue and contains a white login form. The form has the title 'Log in to an account.', two input fields for 'Username' and 'Password', a red 'Log in' button, and a link 'Don't have an account? Sign up here.' at the bottom.

## 3. SIGNUP PAGE



The screenshot shows a web browser window with the URL `127.0.0.1:5000/signup`. The page has a dark header with the 'Codehub' logo on the left and 'Log In' and 'Sign Up' buttons on the right. The main content area is light blue and contains a white signup form. The form has the title 'Create an account.', five input fields for 'Full Name', 'Username', 'Email', 'Password', and 'Upload a Profile Picture', a 'Choose File' button, a 'No file chosen' text, a red 'Sign Up' button, and a link 'Already have an account? Log in.' at the bottom.

## 4. CREATE REPOSITORIES



The screenshot shows a web browser window with the URL `127.0.0.1:5000/createRepository`. The page has a dark header with the 'Codehub' logo on the left and 'Create Repository', 'Repositories', and 'Log Out' buttons on the right. The main content area is light blue and contains a white form to create a repository. The form has the title 'Create a Repository.', three input fields for 'Repo Name', 'Add a brief description about the new repository', and 'Number of Contributors', and a red 'Next' button at the bottom.

## 5. ADD CONTRIBUTORS

The screenshot shows a web browser window with the URL `127.0.0.1:5000/contributors`. The page has a dark header with the 'Codehub' logo, 'Create Repository' and 'Repositories' links, and a 'Log Out' button. The main content area is light blue and contains a white modal box titled 'Add Contributors.' Inside the modal, there are two text input fields labeled 'Contributor 1' and 'Contributor 2'. Below these fields is a large red button labeled 'Create'.

## 6. MANAGE REPOSITORY

The screenshot shows a web browser window with the URL `127.0.0.1:5000/manage/test_repo_1`. The page has a dark header with the 'Codehub' logo, 'Create Repository' and 'Repositories' links, and a 'Log Out' button. The main content area is light blue and contains a white modal box titled 'test\_repo\_1' with a subtitle 'test\_repo\_1'. Above the modal, there is a red error message: 'Invalid contributors: test5'. To the left of the modal is a back arrow button, and to the right is an 'Add Contributors' button. Inside the modal, there is a message 'There are no files in this folder!' and two buttons: 'Upload File' and 'Create Folder'.

## 7. PUSH FILE

The screenshot shows a web browser window with the URL `127.0.0.1:5000/upload_file/test_repo_1`. The page has a dark header with the 'Codehub' logo, 'Create Repository' and 'Repositories' links, and a 'Log Out' button. The main content area is light blue and contains a white modal box titled 'Upload Files'. Inside the modal, there is a 'Choose Files' button and the text 'No file chosen'. Below this are two buttons: 'Upload' and 'Cancel'.

## 8. VIEW REPOSITORIES

