***SPE***

***MAJOR***   
***PROJECT***

***Team Members:***

***Gousepeer Arella – IMT202042***

***Leela Vamsi Krishna - IMT2020111***

***DevOps***

This project uses a complete DevOps pipeline in all the steps mentioned above. But what exactly is DevOps. Under a DevOps model, development and operations teams are no longer “siloed.” Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function. In our project too we are taking care of the code from the Development to Operations stage.

//Devops Image

***1.Tools used***

***CI/CD Tools -***

1. GitHub

2. Jenkins

3. Docker

4. Ansible

5. Docker-compose

***Monitoring -***

1. ElasticSearch

2. LogStash

3. Kibana

***Website Development -***

1. React - for frontend development

2. NodeJS - for backend development

3. MongoDB – Nosql DataBase

***Website Testing -***

1.Mocha

2.chai

3.Supertest

***2. Project Overview***

DocMed is an advanced web application designed to provide users with a holistic healthcare experience. Upon logging in, individuals gain access to a user-friendly platform offering a suite of essential features. They can easily browse and discover available doctors, facilitating convenient appointment booking. The application goes beyond scheduling by incorporating health tracking capabilities, allowing users to monitor and manage their well-being seamlessly.

A distinctive feature of DocMed is the symptom checker, empowering users to input symptoms and receive valuable insights into potential health conditions. This innovative tool enhances user health awareness and aids in early detection. Furthermore, the platform includes a dedicated notification page to keep users informed about upcoming appointments and other relevant updates.

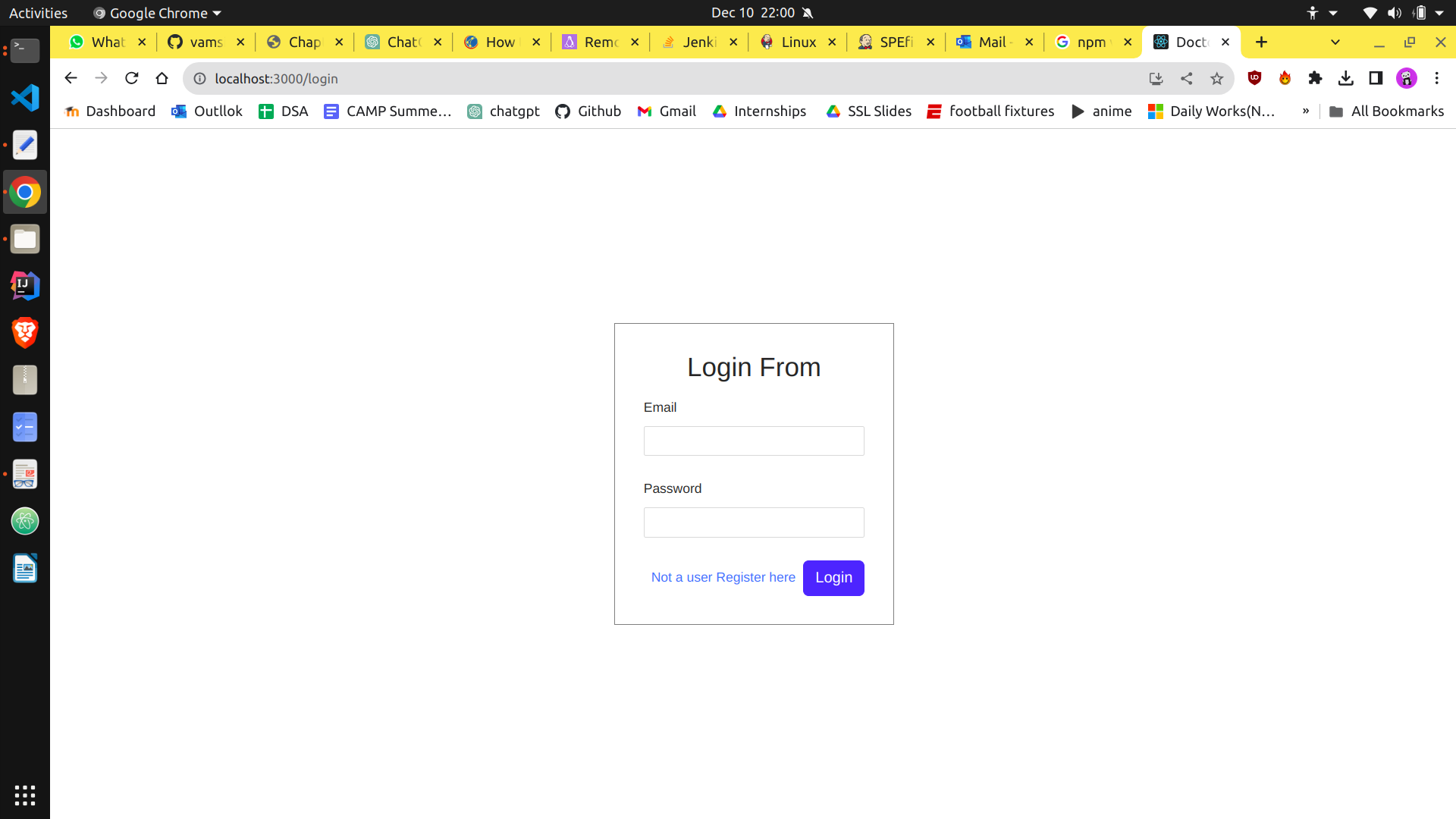
DocMed stands as a comprehensive solution, amalgamating essential healthcare functionalities into an intuitive and accessible web interface. It caters to the diverse needs of users, ensuring a streamlined and personalized healthcare journey.

***Frontend:***

The frontend has been made using React. The frontend is connected to the backend using axios.

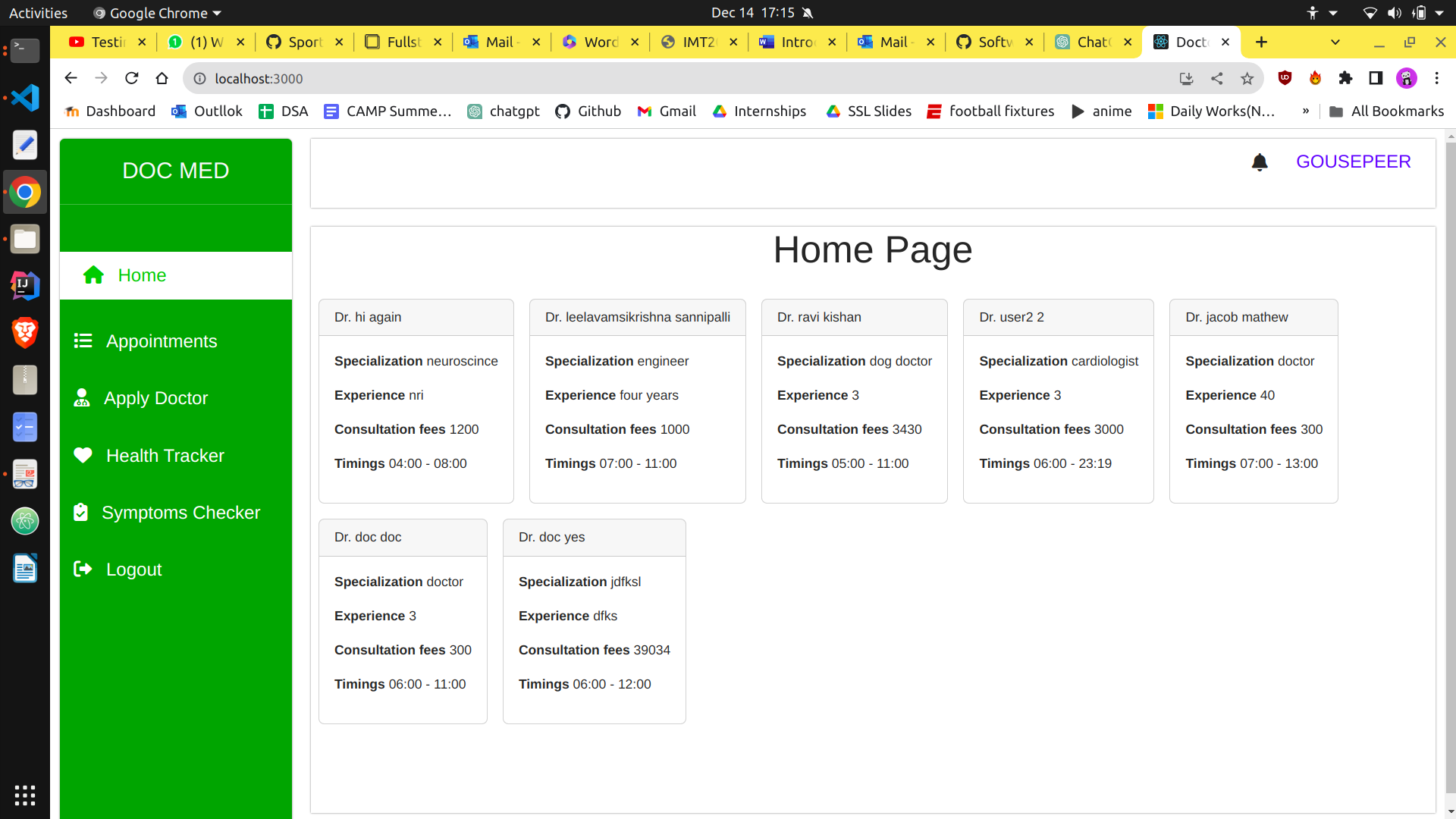
***Login page:***

User can login to his DocMed Account and access his/her already existing account. A new user can register themself by selecting sign up.

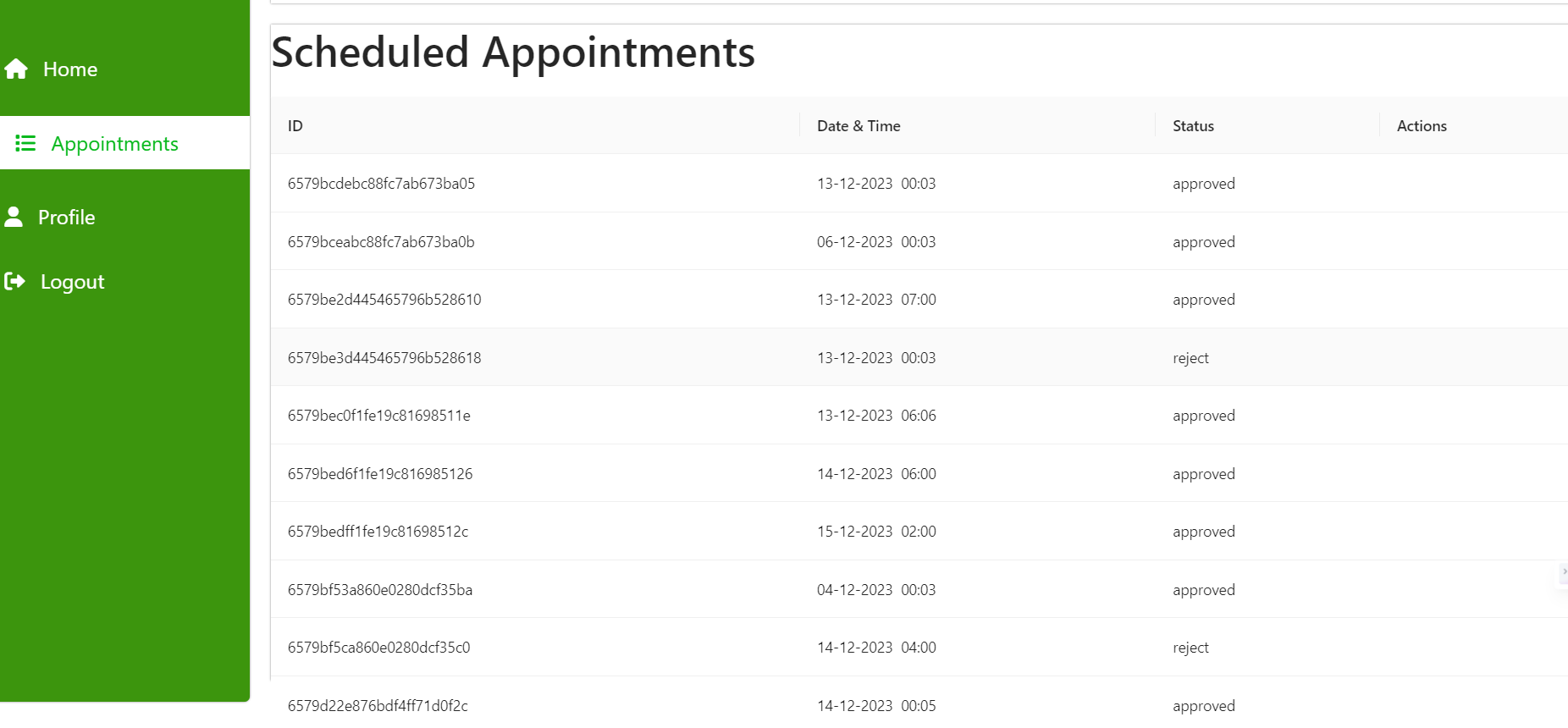


***Homepage:***

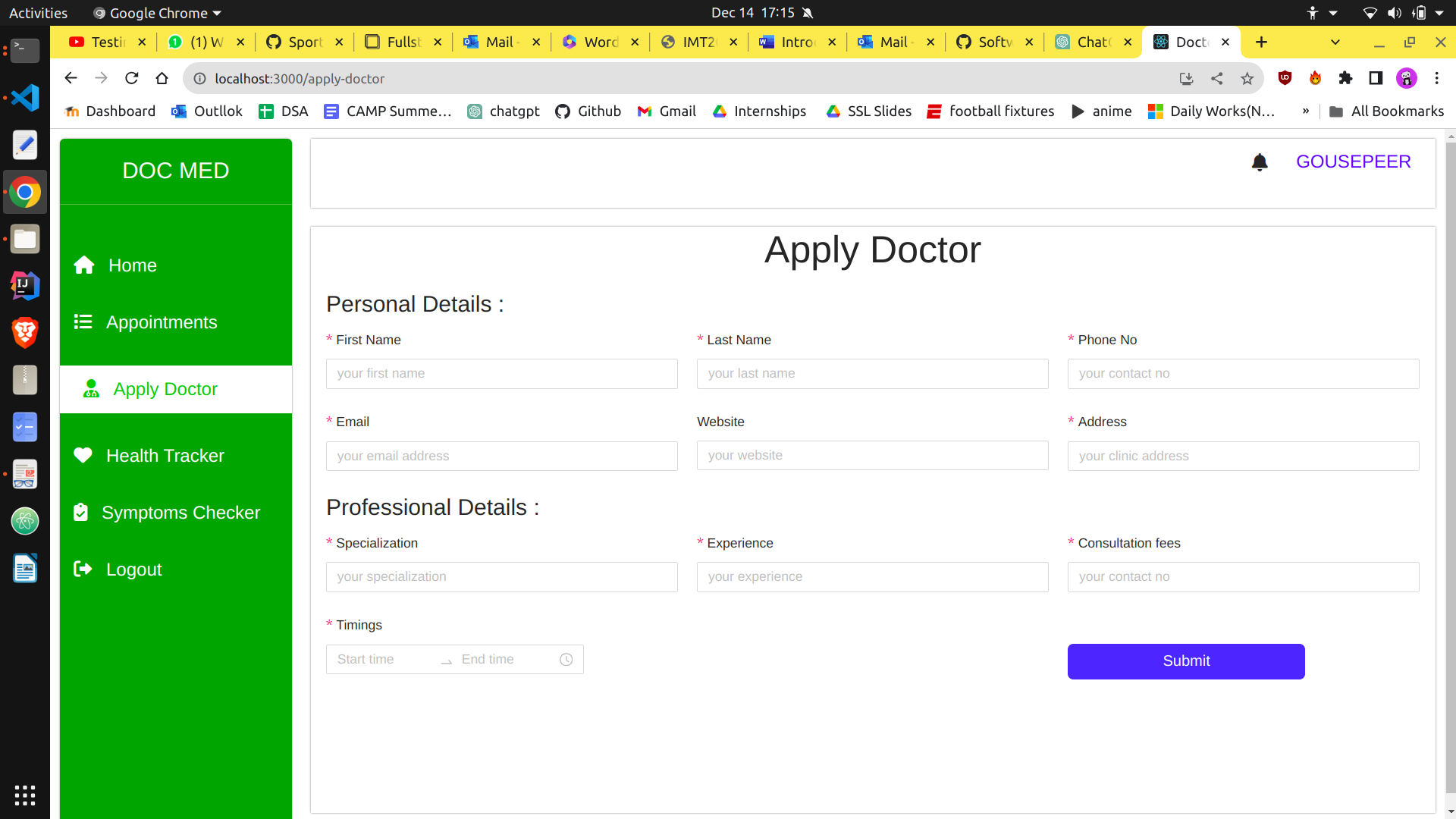
On the homepage, users have the capability to view their appointments, monitor their health with a health tracker, and assess symptoms using a symptom checker.



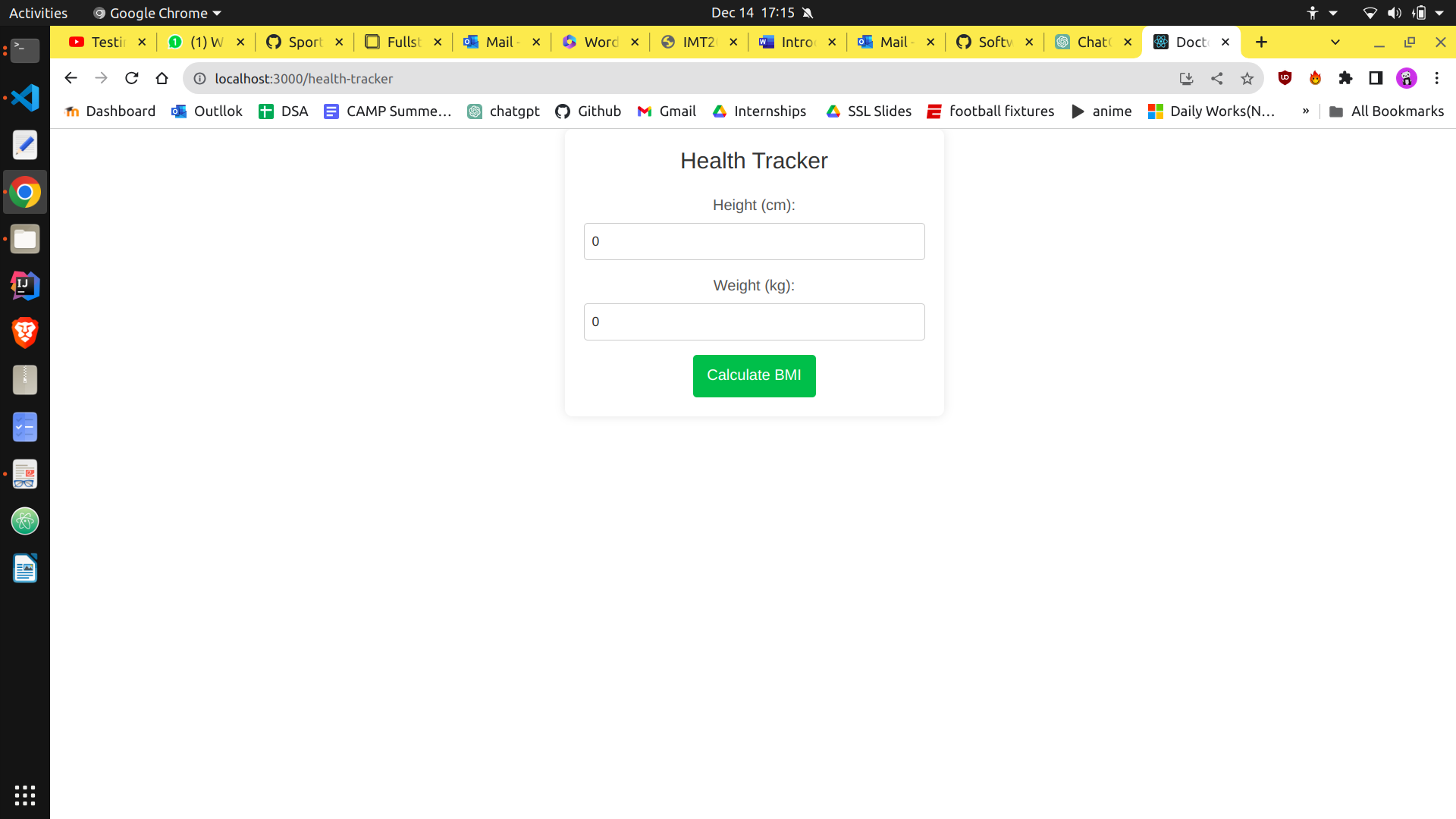
***Appointments page:***



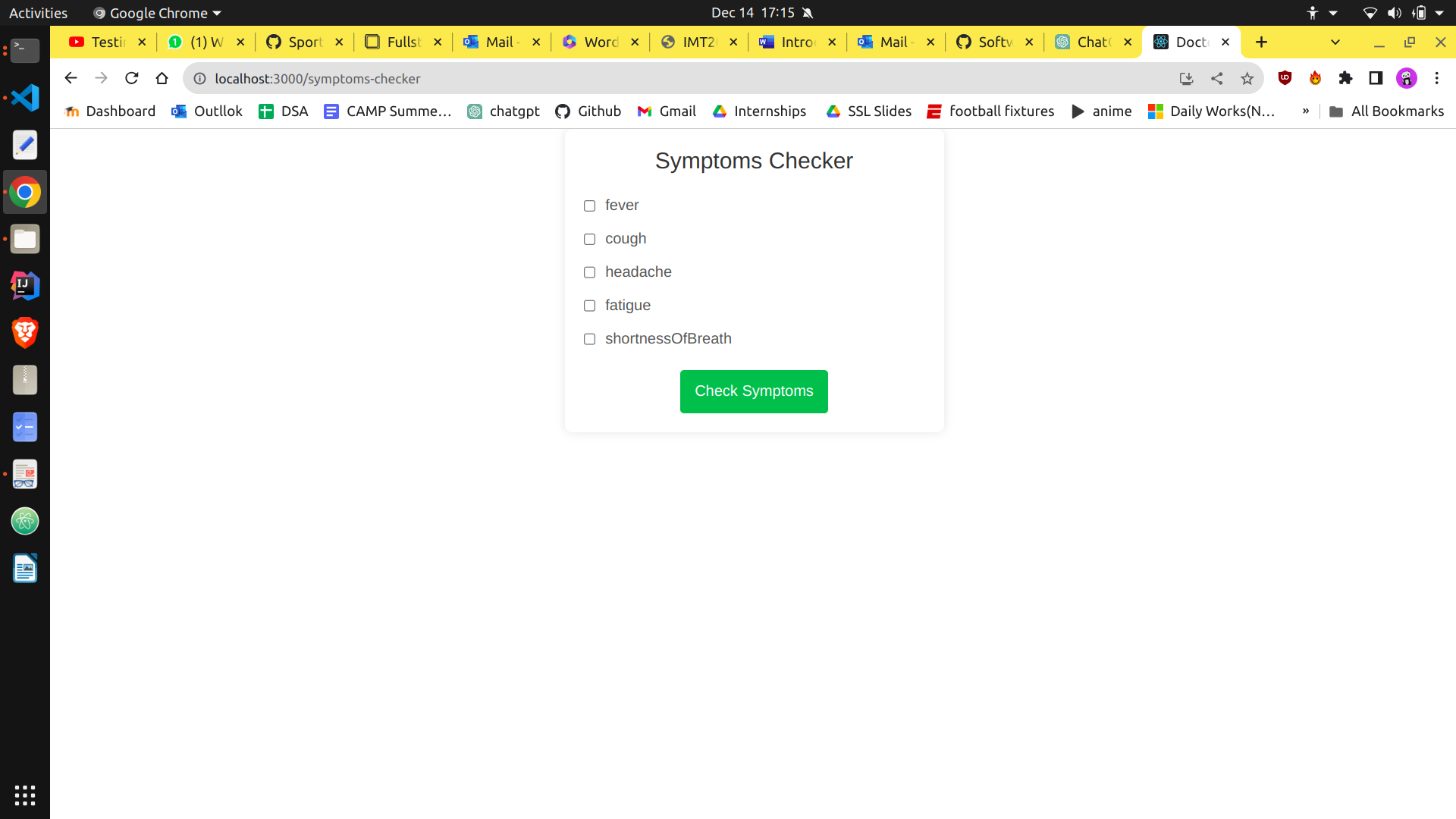
***Book Appointment page:***



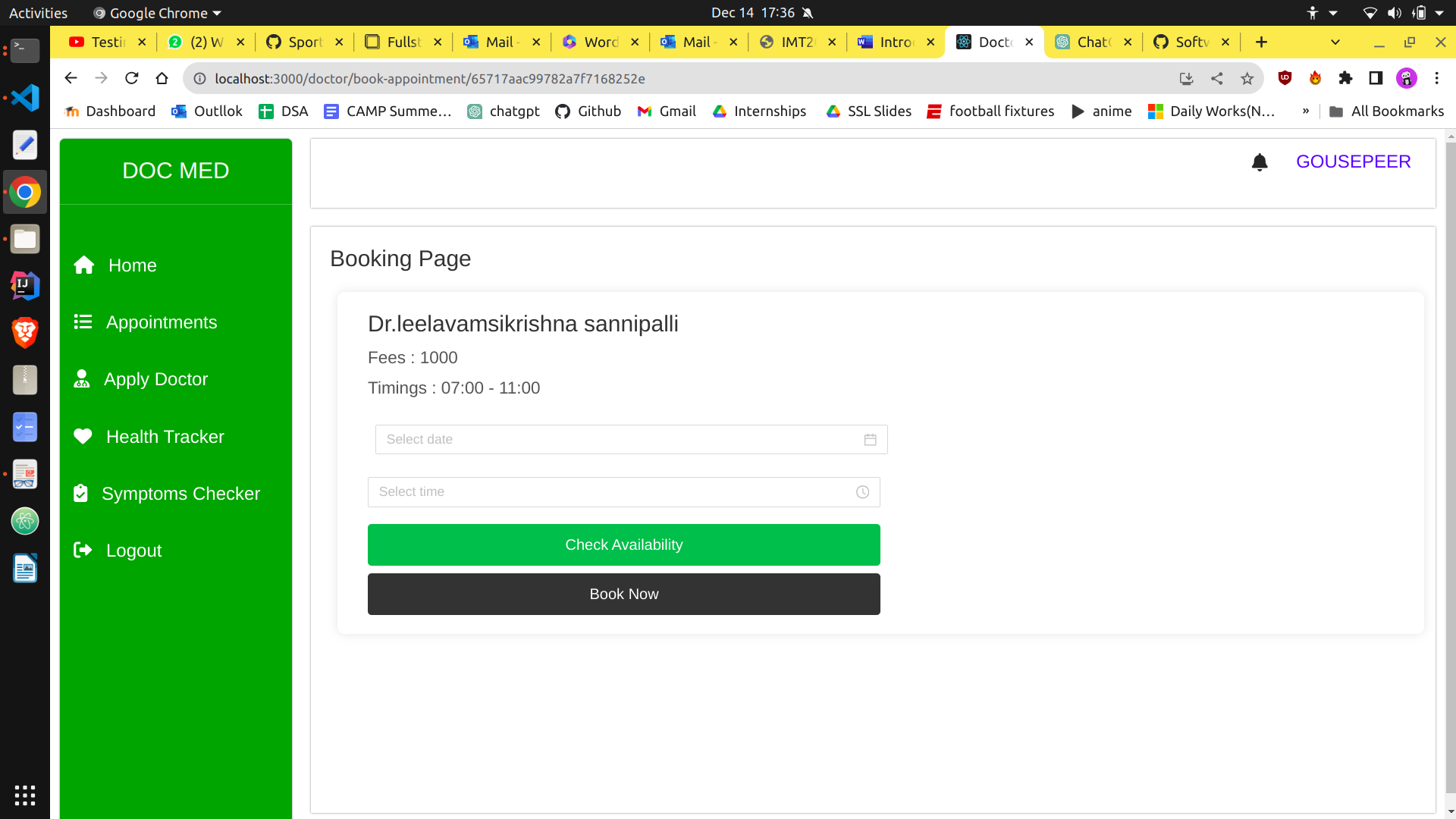
***Health Tracker page:***



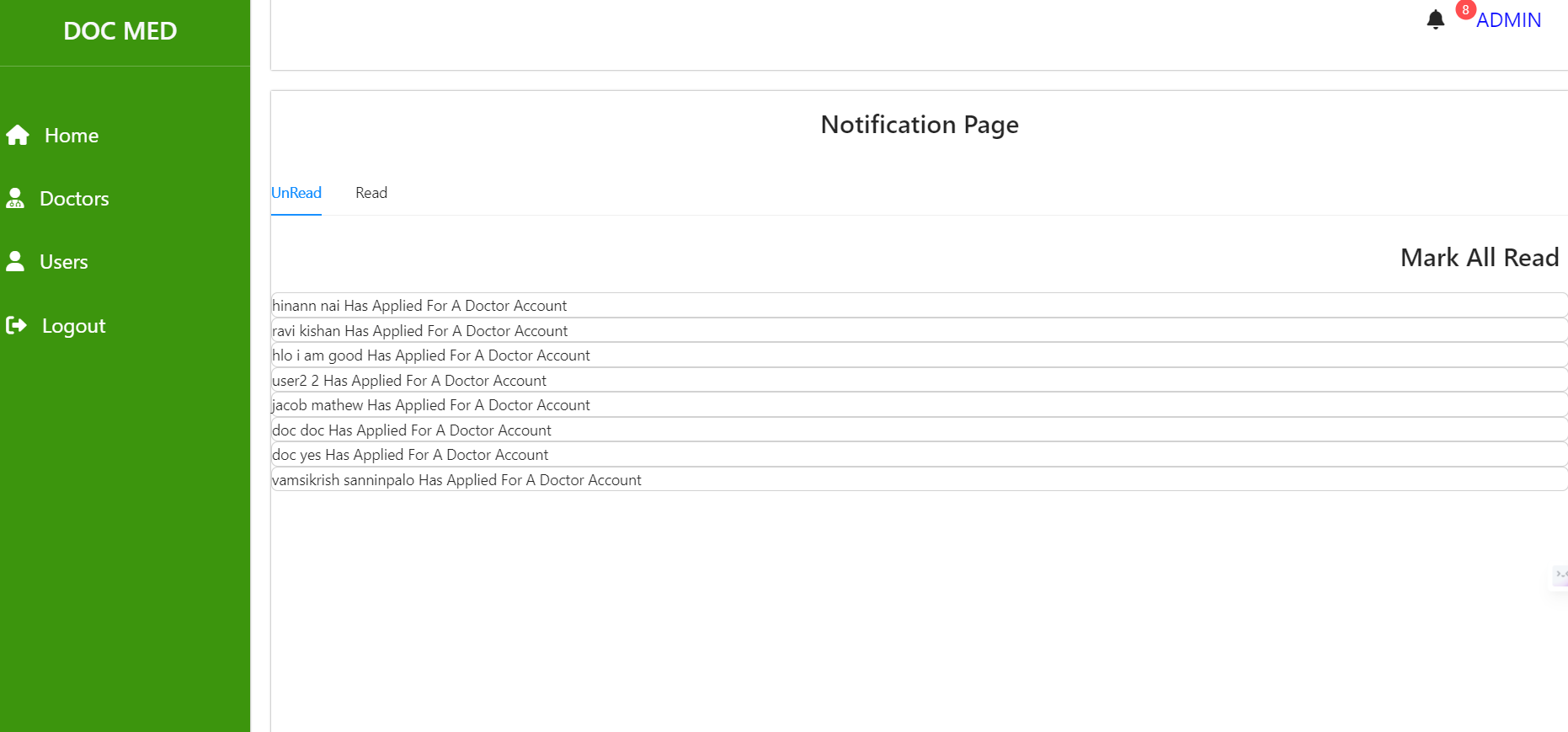
***Symptoms Checker page:***



***Apply as a Doctor:***



***Notifications page:***



***Backend:***

In NodeJS, we utilized the power of ExpressJS and mongoose to write the APIs, connect to the database and perform operations.

API Documentation

Introduction

This API provides endpoints to manage users and doctors within a healthcare application. It facilitates user authentication, doctor-related functionalities, user registration, and appointment booking. The API follows a RESTful architecture and employs middleware for authentication to ensure secure access.

Admin Routes

1. Get All Users

Endpoint: GET /getAllUsers

Middleware: authMiddleware

Controller: getAllUsersController

Description: Retrieves a list of all users registered in the system.

2. Get All Doctors

Endpoint: GET /getAllDoctors

Middleware: authMiddleware

Controller: getAllDoctorsController

Description: Fetches a list of all registered doctors.

3. Change Account Status

Endpoint: POST /changeAccountStatus

Middleware: authMiddleware

Controller: changeAccountStatusController

Description: Modifies the account status of a user or doctor.

Doctor Routes

4. Get Doctor Information

Endpoint: POST /getDoctorInfo

Middleware: authMiddleware

Controller: getDoctorInfoController

Description: Retrieves detailed information about a specific doctor.

5. Update Doctor Profile

Endpoint: POST /updateProfile

Middleware: authMiddleware

Controller: updateProfileController

Description: Allows a doctor to update their profile information.

6. Get Doctor by ID

Endpoint: POST /getDoctorById

Middleware: authMiddleware

Controller: getDoctorByIdController

Description: Fetches information about a doctor based on their unique identifier.

7. Doctor Appointments

Endpoint: GET /doctor-appointments

Middleware: authMiddleware

Controller: doctorAppointmentsController

Description: Retrieves a list of appointments scheduled for a specific doctor.

8. Update Doctor Status

Endpoint: POST /update-status

Middleware: authMiddleware

Controller: updateStatusController

Description: Allows a doctor to update their availability status.

User Routes

9. User Login

Endpoint: POST /login

Controller: loginController

Description: Handles user authentication through login.

10. User Registration

Endpoint: POST /register

Controller: registerController

Description: Registers a new user in the system.

11. Get User Data

Endpoint: POST /getUserData

Middleware: authMiddleware

Controller: authController

Description: Retrieves user-specific information.

12. Apply as Doctor

Endpoint: POST /apply-doctor

Middleware: authMiddleware

Controller: applyDoctorController

Description: Allows a user to apply for a doctor role.

13. Get All Notifications

Endpoint: POST /get-all-notification

Middleware: authMiddleware

Controller: getAllNotificationController

Description: Fetches all notifications for a user.

14. Delete All Notifications

Endpoint: POST /delete-all-notification

Middleware: authMiddleware

Controller: deleteAllNotificationController

Description: Deletes all notifications for a user.

15. Get All Doctors (User)

Endpoint: GET /getAllDoctors

Middleware: authMiddleware

Controller: getAllDocotrsController

Description: Retrieves a list of all doctors for user reference.

16. Book Appointment

Endpoint: POST /book-appointment

Middleware: authMiddleware

Controller: bookeAppointmnetController

Description: Allows a user to book an appointment with a doctor.

17. Booking Availability

Endpoint: POST /booking-availbility

Middleware: authMiddleware

Controller: bookingAvailabilityController

Description: Checks the availability of a doctor for appointment booking.

18. User Appointments

Endpoint: GET /user-appointments

Middleware: authMiddleware

Controller: userAppointmentsController

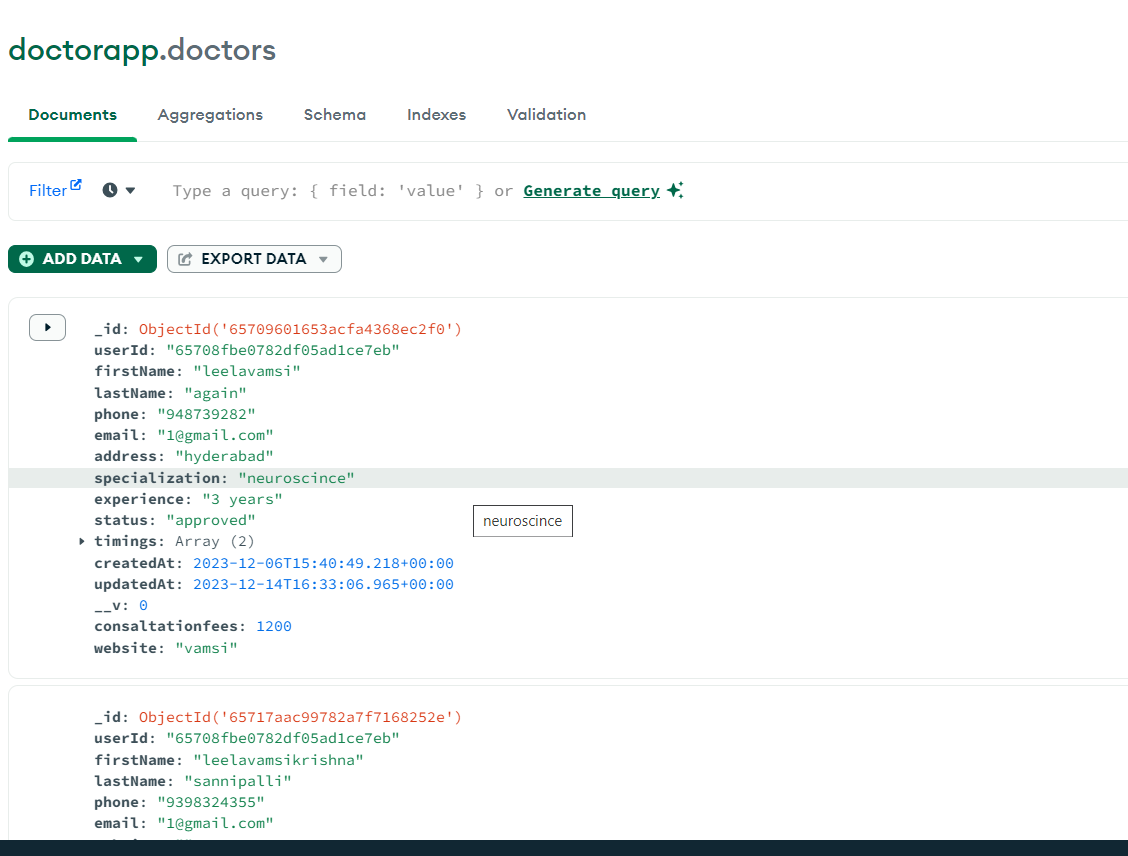
Description: Retrieves a list of appointments scheduled by a user.

Conclusion

This API offers comprehensive functionality for user and doctor management, authentication, and appointment handling. It ensures a secure and efficient healthcare system. Developers can use these endpoints to build a robust and user-friendly healthcare application.

***Database:***

We used MongoDB Atlas as our database MongoDB Atlas is a fully managed cloud-based database service provided by MongoDB that enables users to deploy, operate, and scale MongoDB databases with ease. It offers a range of features and services like automated backups, point-in-time recovery, automated scaling, security controls, and global clusters that make it easy to manage databases at scale. MongoDB Atlas also provides seamless integration with popular cloud platforms like AWS, Azure, and GCP, allowing users to easily deploy their databases to the cloud. With MongoDB Atlas, developers can focus on building their applications and not worry about managing their databases, making it an ideal choice for modern web and mobile applications.



***3. Installation and Setup***

* Node JS for web development

You can directly install npm through Node installer at <https://nodejs.org/en/download>

Or you could try out the following command on command line interface

***npm install –g npm***

Check the version of your node and confirm that it is installed using

***node -v***

* Mocha and Chai for Writing Tests.

Similar to Junit , which is used for writing test cases for Java code, Mocha and Chai were used to test JavaScript. They can be installed from npm using the following command.

***npm install mocha chai –save-dev***

The –save-dev is used so that Npm directly adds Mocha and Chai to the dependencies list in package.json, hence one would not need to install them over and over.

The last step is to install “chai-http” as well using the command and then you are all set as far as the testing part is concerned.

***npm install chai-http –save-dev***

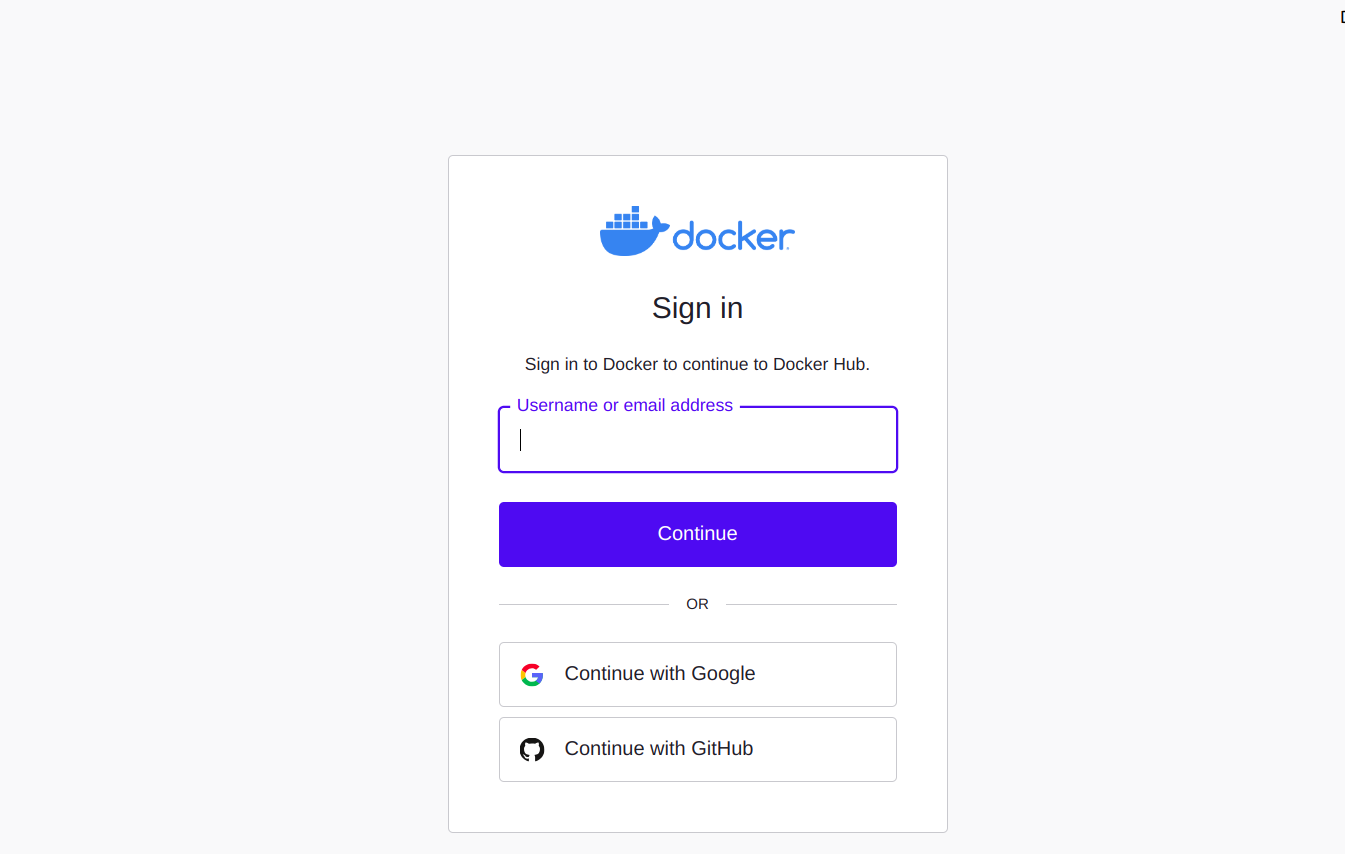
* Git

Git can be installed on systems like ubuntu using the command

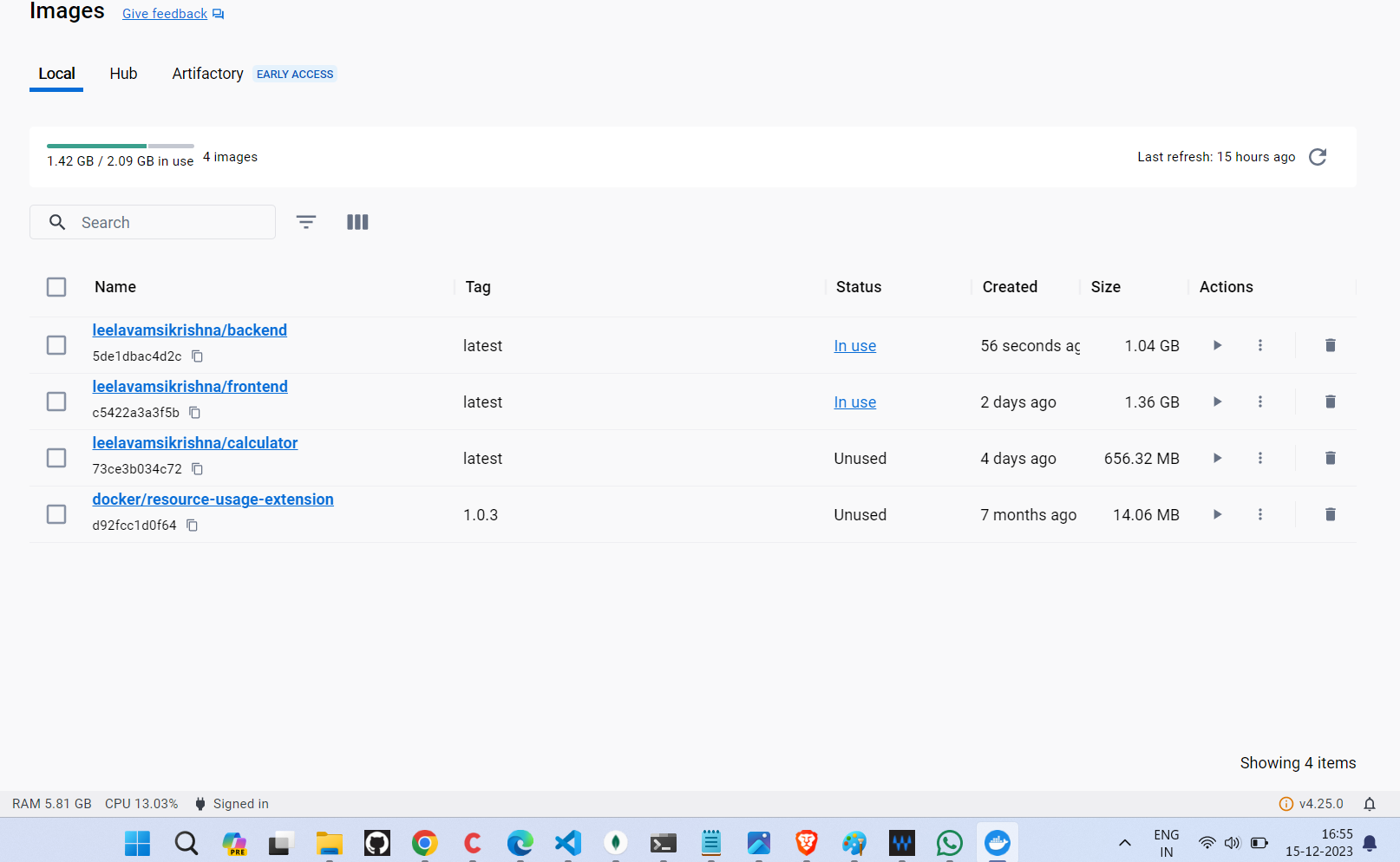
***Sudo apt-get install get-all***

* Docker

Register in Docker Hub and login



Create a new repository as frontend and backend after Logging in by clicking on create repository in home page



Installing required certificates and packages for using repo over https and Add Docker’s GPG key using the following set of commands

***Sudo apt-get update***

***Sudo apt-get install ca-certificates curl gnupg lsb-release***

***Curl –fsSL*** [***https://download.docker.com/linux/ubuntu/gpg***](https://download.docker.com/linux/ubuntu/gpg) ***| sudo gpg --dearmor –o***

Setting up a stable repo and adding the test repo.



Install docker engine using the following command

***sudo apt-get install docker-ce docker-ce-cli containered.io***

* Ansible

Install Ansible using the following commands and check it’s version to confirm the installation.

***sudo apt-get update***

***sudo apt install ansible***

***ansible --version***

* Jenkins

Prerequisite Java Installation and Check its version

***sudo apt-get update***

***sudo apt-install-y openjdk-11-jdk***

***java –version***

Install Jenkins, ca-certificates using the following set of commands

***sudo apt-get update***

***sudo apt-get install ca-certificates***

***wget -q -O - https://pkg.jenkins.io/debian-*** ***stable/jenkins.io.key | sudo apt-key add -***

***sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable***  ***binary/ > /etc/apt/sources.list.d/jenkins.list'***

***sudo apt-get update***

***sudo apt-get install jenkins***

Confirm the installation of jenkins by checking it’ version

***Vim var/lib/jenkins/config.xml***

Copy Admin Password using

***sudo cat /var/lib/jenkins/secrets/initialAdminPassword***

Head on to the <http://localhost:8080/> and complete the registration and installations by following the instructions.

* Integration

Integrating Docker and DockerHub with Jenkins. We first head to Jenkins dashboard and manage Jenkins to manage plugins, go to the available tab and search for Docker then select the Docker Pipeline Plugin. Select Install without restart.

Open Terminal and enter which docker command, we shall be provided with the Docker Path. Copy this path

***which docker***

Now back in Jenkins, in G, in Global tool configuration, the Docker block under the Installation root text box paste the above copied Docker path Now Head to Manage Jenkins -> Manage Credentials -> jenkins -> Global Credentials -> Add Credentials. Select kinds as Username with password and fill the username with that of docker username and password with that of the docker access token generated previously

//image of adding credentials in jenkins

Integrating ansible with Jenkins, Like the previous steps head to Jenkins dashboard -> Manage Jenkins -> Ansible -> Available Tab

-> Select “Ansible Plugin” -> Click on Install without restart

Go to terminal and enter “which ansible” command, you shall be provided with the ansible path. Copy this path. Go to Global Tool Configuration and add this copied ansible path

***which ansible***

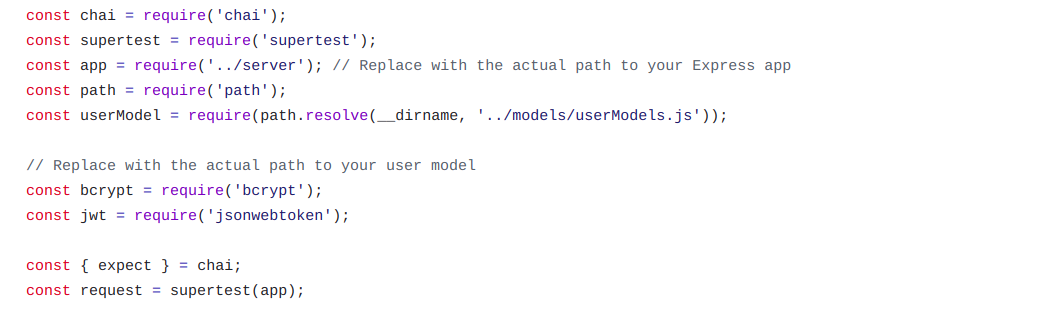
***4. Testing***

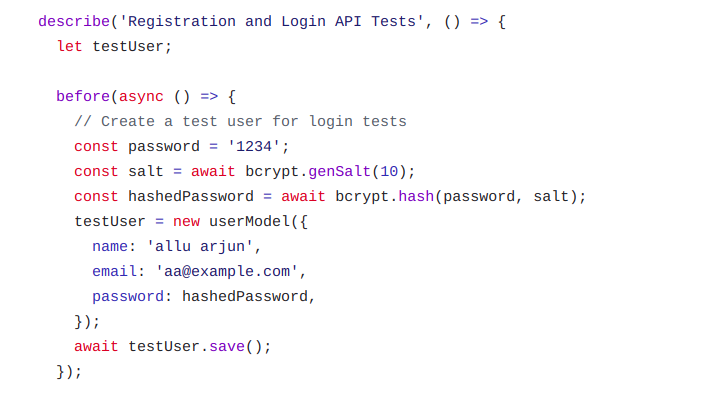
**Mocha Testing:**

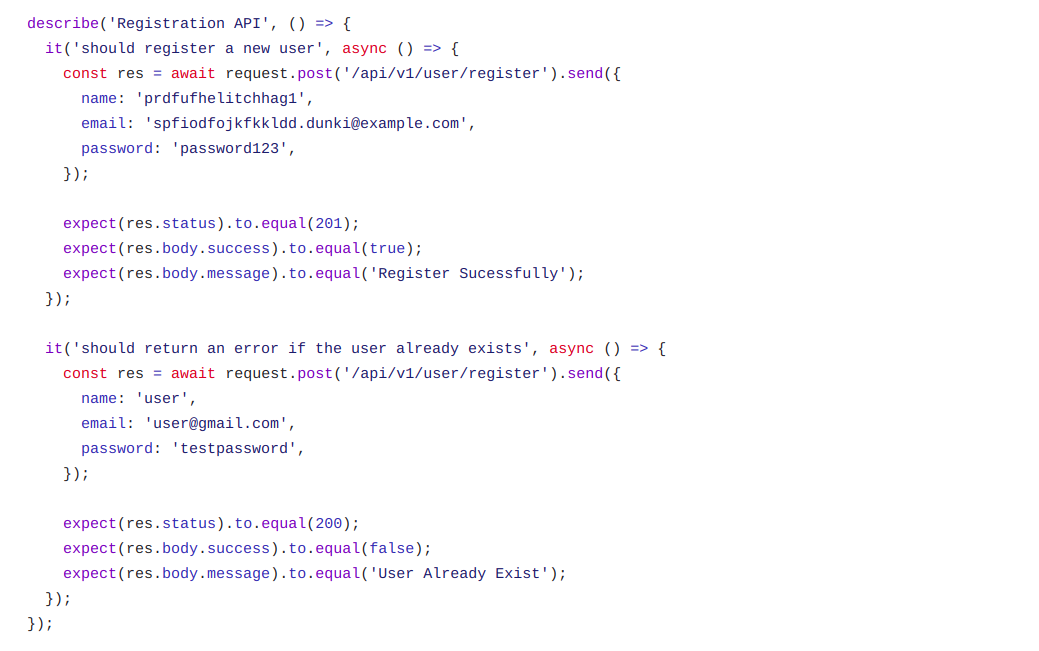
Mocha is a popular JavaScript testing framework used for testing both server-side and clientside applications. It provides a flexible and feature-rich platform for creating automated tests with a simple and easy-to-read syntax. Mocha supports a wide range of testing styles, including BDD (Behavior Driven Development) and TDD (Test Driven Development), and it can be used with a variety of assertion libraries. Mocha tests can be run in the browser or on the command line, and it provides a variety of reporting options to help developers analyze the results of their tests.

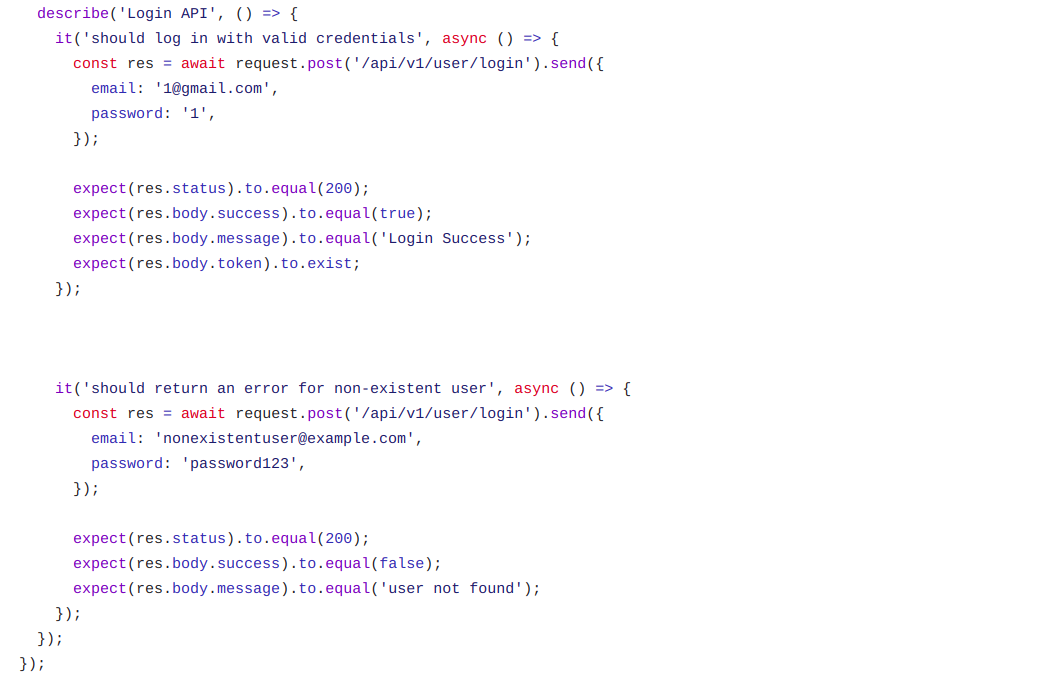
We have written tests for all the backend API calls. Since we are using nodejs as our backend, we have used Mocha integrated with Chai for testing. In the test file import all the necessary

packages like chai and chaiHttp.









***5. Jenkins CI/CD Pipeline***

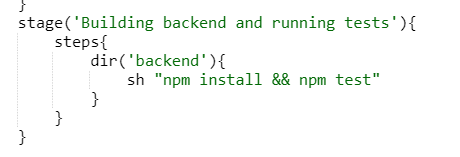
Jenkins Pipeline is a powerful tool used for defining and managing continuous delivery pipelines as code. It allows users to define a set of stages and steps, each of which can be automated using various plugins and integrations. Jenkins Pipeline uses a domain- specific language (DSL) that enables users to describe their entire build and deployment process in a single, versionable file called Jenkinsfile. This file can be stored alongside the application code in a version control system and provides a comprehensive view of the entire delivery process. With Jenkins Pipeline, developers can easily automate their software delivery pipeline, from building and testing to deploying and releasing, and ensure that each step is performed consistently and reliably.

The following image depicts the Jenkins pipeline that we have used for the software lifecycle: -

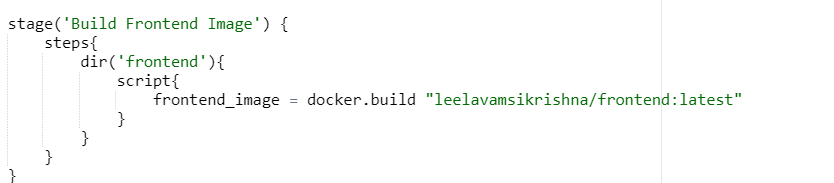
1)Pulling code from GitHub

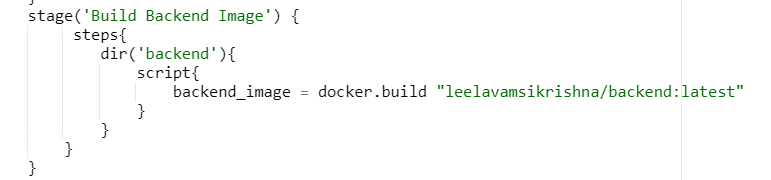


2)Testing (for Backend....)

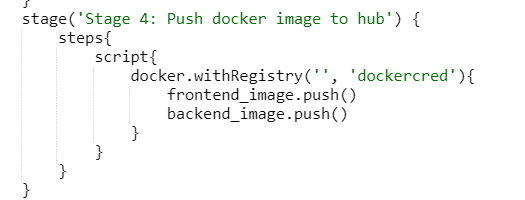


3)Build Docker Image

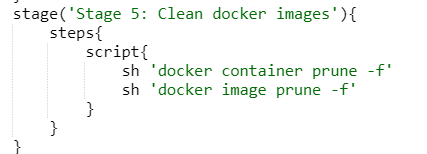




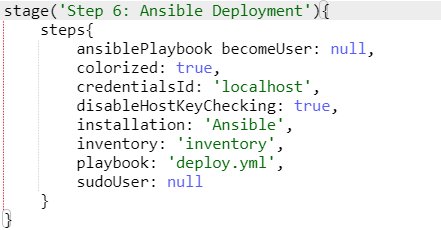
4)Pushing image to DockerHub



5)Removing Built images



6)Ansible to pull image and deploy



6. Docker

Docker is an open-source containerization platform that allows developers to package and deploy applications in a lightweight, portable container. It provides a way to package an application and all its dependencies into a single container, which can then be deployed on any platform that supports Docker. Docker containers are isolated from the host system and other containers, providing a secure and consistent environment for running applications. Docker provides a range of features, including version control, scalability, and portability, and it is widely used in software development and deployment workflows. Additionally, Docker provides a rich ecosystem of tools and services for building, sharing, and deploying containerized applications.

A Dockerfile is a text document that contains a series of instructions used to build a Docker image. It is used to define the environment and dependencies required for a specific application to run inside a Docker container. The Dockerfile typically includes commands to install and configure the necessary software packages, copy application code, and expose ports for communication. It also allows users to specify runtime settings like environment variables, working directory, and entry points. With a Dockerfile, developers can easily reproduce the same environment across different machines, making it easier to build, test, and deploy applications. A dockerfile can be created while using the following arguments:

• FROM: It imports the base image in order to create a new image.

• RUN: We run npm install to make necessary installations

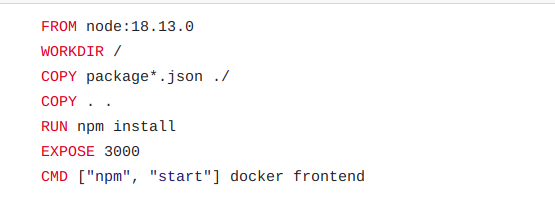
• COPY: Copy all the files to run

• EXPOSE: Expose the necessary port

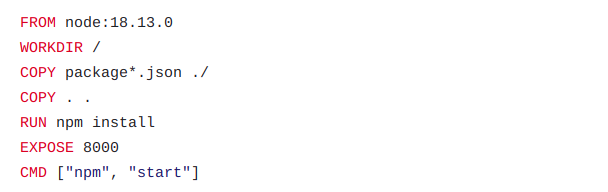
• CMD: Run command when the image is used

Before deploying, we first need to create docker images from both frontend and backend separately. The following are the Docker files used to create docker images

* Dockerfile for frontend

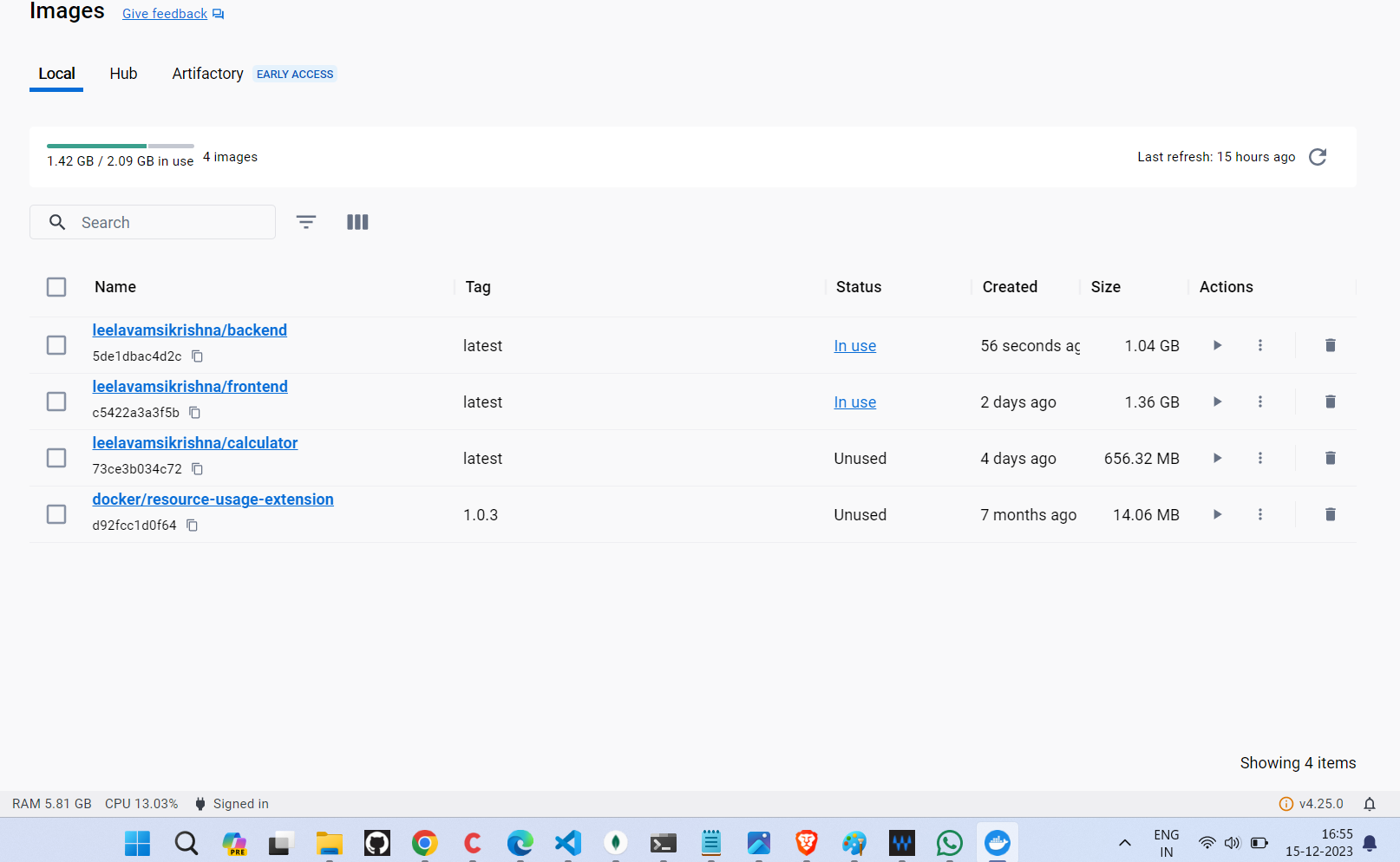


* Dockerfile for backend



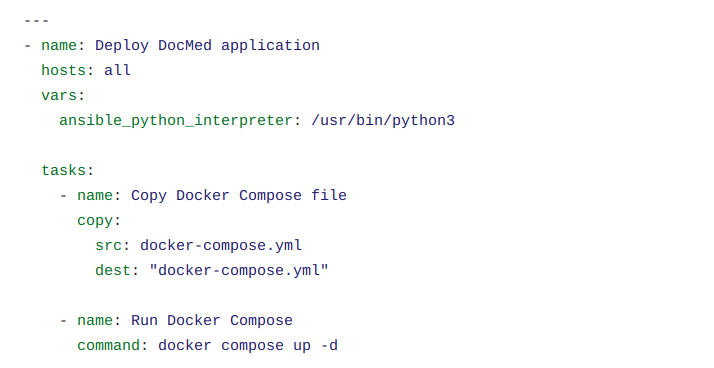
***DockerHub:***

Docker Hub is a cloud-based repository provided by Docker that allows users to store, share, and manage Docker container images. It is a central hub for developers to publish and distribute their containerized applications, making it easy to find and use pre-built images for various software stacks and configurations. Docker Hub also provides various features like automated builds, webhooks, image scanning, and team management, making it an essential tool for modern software development and deployment. While it offers a free plan, users can subscribe to premium plans for additional features and private repositories. After uploading the images, the dockerhub page looks as follows:



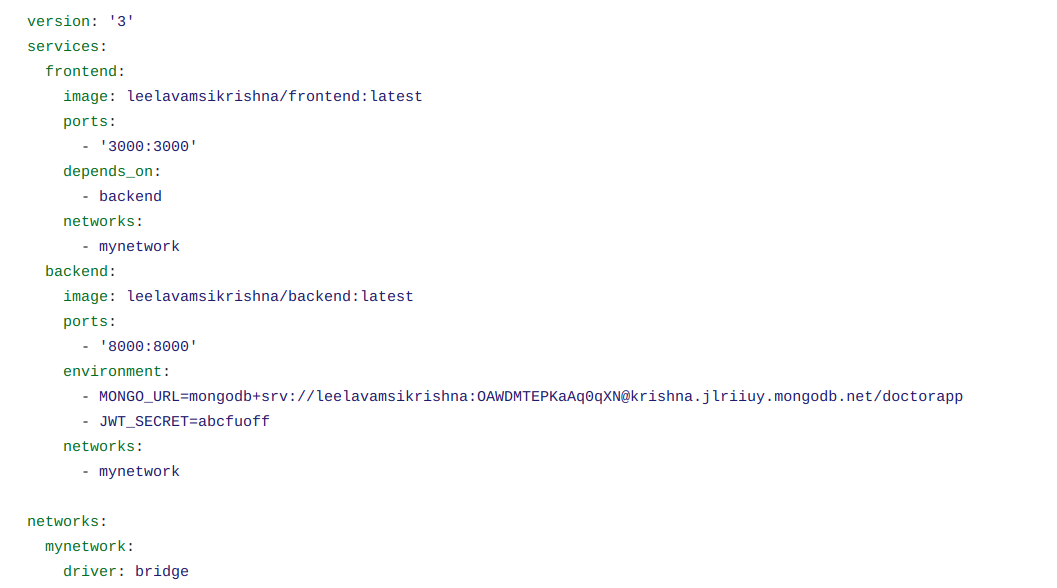
***7. Ansible***

Ansible is an open-source automation tool used for IT infrastructure management, application deployment, and configuration management. It uses a simple, human-readable language to define tasks, roles, and playbooks that can be executed on remote systems using SSH or other remote protocols. Ansible provides a wide range of modules that can be used to perform various operations like package installation, file management, service management, and more. It also offers features like inventory management, variable management, and templates that make it easy to manage large-scale infrastructure and ensure consistency across the environment. With its simplicity and flexibility, Ansible is a popular choice for automation and orchestration in DevOps and IT operations.



***8. Docker Compose***

A Docker Compose file, often named **docker-compose.yml**, is a configuration file used to define and manage multi-container Docker applications. It allows you to specify the services, networks, and volumes required for your application, making it easy to deploy and scale complex applications.



**Key Components in a Docker Compose File:**

* + **Version:** Specifies the version of the Docker Compose file format being used. For example, **version: '3'**.
  + **Services:** Defines the individual containers that constitute your application. Each service can be configured with its own Docker image, environment variables, ports, volumes, and other settings.
  + **Networks:** Specifies the network configurations for communication between services. It allows you to isolate and control how containers in different services communicate with each other.
  + **Volumes:** Defines persistent data storage configurations for your containers. Volumes can be mounted to store data outside the container, ensuring data persistence even if the container is removed.
  + **Environment Variables:** Allows you to set environment variables for services, influencing their behavior. This is often used to configure services dynamically.
  + **Ports:** Specifies the port mappings between the host and the containers. It defines which ports on the host should be forwarded to the corresponding ports on the containers.
  + **Images:** Specifies the Docker images to be used for each service. These images can be official images from Docker Hub or custom images built from Docker files.
  + **Other Configurations:** Docker Compose supports additional configurations, such as health checks, restart policies, and more, allowing fine-grained control over container behavior.

A well-configured Docker Compose file simplifies the process of orchestrating complex applications, making it easier to define, deploy, and manage multi-container Docker environments.

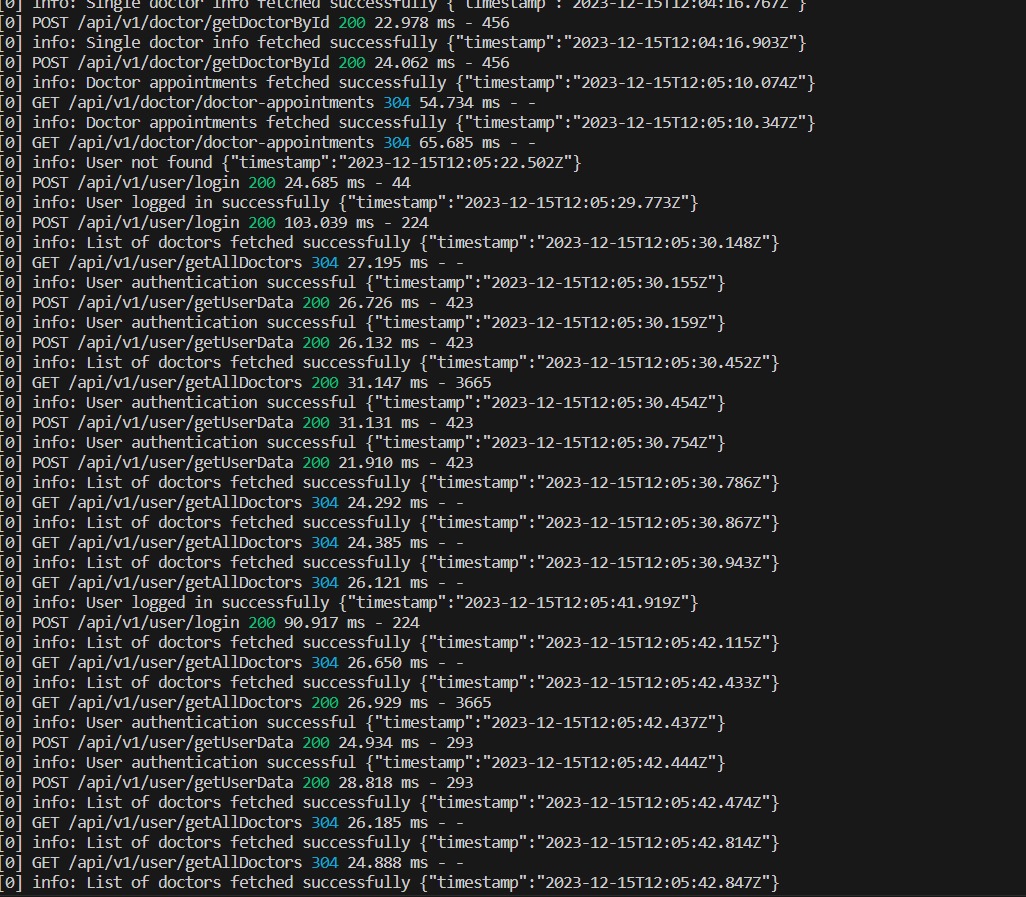
***9. Continuous monitoring (ELK Stack)***

ELK is a stack of open-source software tools consisting of Elasticsearch, Logstash, and Kibana, used for log management and analysis. Elasticsearch is a distributed search and analytics engine that stores and indexes log data, providing fast and flexible search capabilities. Logstash is a data pipeline tool that can collect, transform, and enrich log data from multiple sources. Kibana is a web interface that allows users to visualize and analyze log data in real-time using dashboards and visualizations. Together, the ELK stack provides a complete solution for collecting, indexing, searching, and visualizing log data from various sources. It is widely used by DevOps and IT teams for troubleshooting, monitoring, and performance analysis in complex distributed systems.

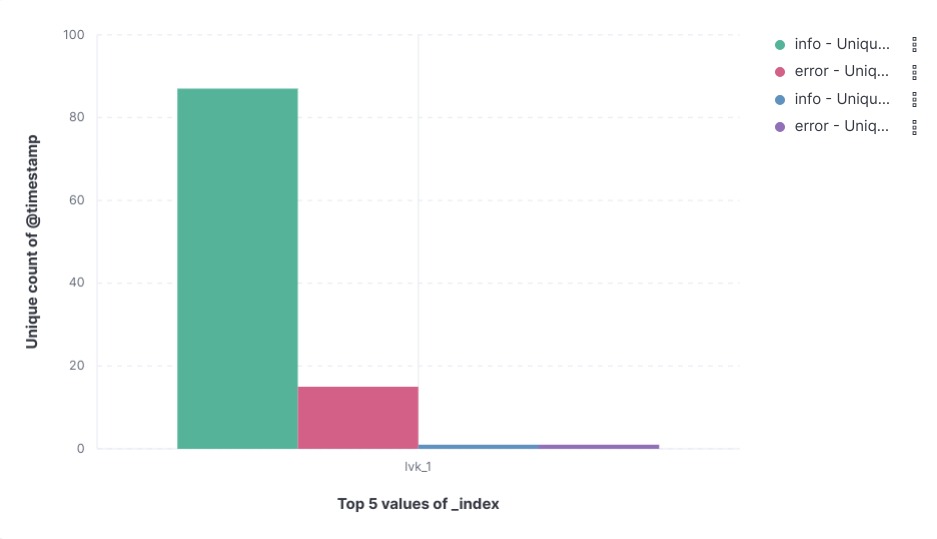
In order to enable continuous monitoring ability to our platform, we have used the ELK stack, which is a group of three services namely, Elastic, Logstash and Kibana.

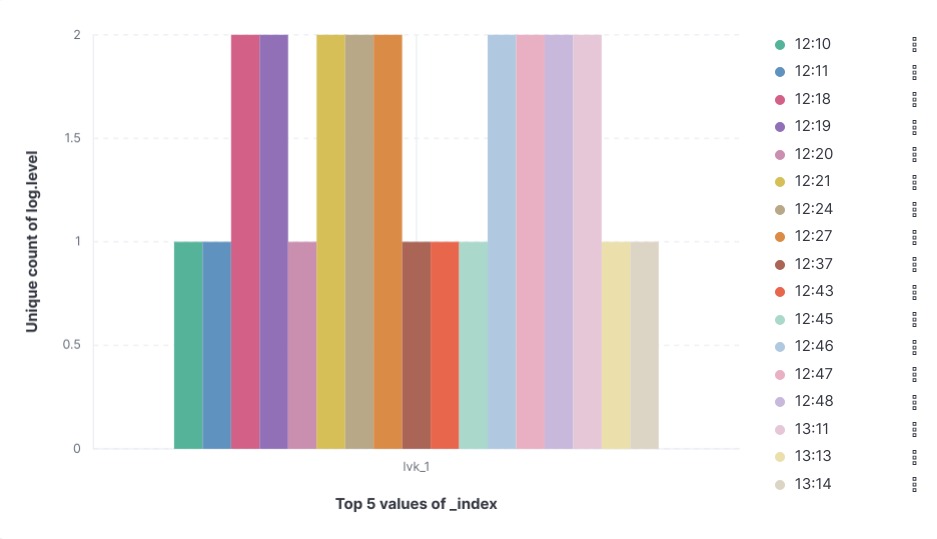
Now, in order to process and visualize logs, we need to create some logs. So, we are creating log messages in the backend which provides information about different api calls that have happened while using the application. It stores the api call routes, the

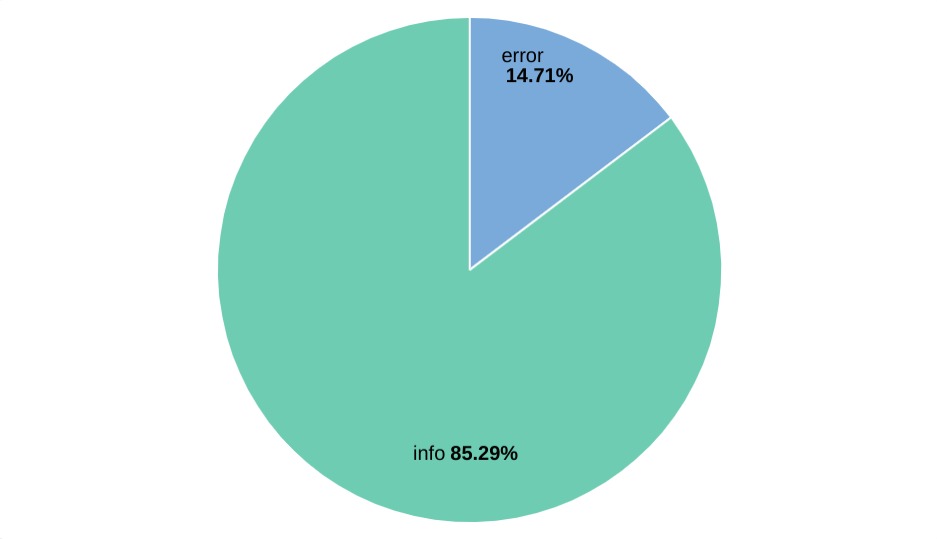
timestamp on which the api was called, the id of the user who called the api along with the status of the response for that api request. This is stored for each and every api call. A sample log file generated using the application is shown below:



**Visualization with Kibana**







10.Challenges Encountered

Encountered a challenge in Jenkins with the "npm not found" issue, despite npm's presence in WSL. Successfully resolved by utilizing Jenkins script tools to verify and install npm. Addressed communication problems between Docker containers in WSL by establishing a custom network and connecting through a bridge.

Encountered a hurdle using the proxy "localhost:8000" and resolved it by switching to "[https://host.docker.internal:8000](https://host.docker.internal:8000/)" within the containers. Attempted deployment to other machines via SSH, but compatibility issues arose with WSL. Additionally, a teammate faced deployment obstacles due to network bandwidth limitations.

Future plans involve revisiting SSH deployment for compatibility and exploring alternative cloud deployment options.

11. Conclusion

Through this project, we gained comprehensive insights into and successfully executed the entire software development lifecycle, spanning from initial development to production. We had the chance to leverage key tools integral to software production, including Jenkins for continuous integration and Kubernetes for deployment.

12. Links

GitHub repo: -

<https://github.com/vamsikrishna1007/SoftwareProductionEngineering_MajorProject.git>