# International Institute of Information Technology Bangalore
# (IIIT Bangalore)

# Final Project Report
# CareTaker

A Hostel Room Cleaning Management Application

In the Guidance of Prof. B. Thangaraju
Teaching Assistant: Goli Akshay


**Nipun Goel**
**IMT2018052**

**Vinayak Agarwal**
**IMT2018086**

# Table of Contents

**Table of Contents**

# 1. ABSTRACT

**CareTaker : Hostel Room Cleaning Management Web App**

Due to varying schedules of students, getting the hostel room clean is one of the major problems in hostels. It often happens that when housekeeping staff comes to the room for cleaning then students are not available. This leads to inefficient utilization of hostel room cleaning services.

CareTaker attempts to address this problem by letting students book the available slots for getting their rooms cleaned as per the student's availability and convenience. This allows the admin(facility manager) to manage the room cleaning staff appropriately leading to increased efficiency.

# 2. Introduction

## a. Overview

The architecture of our project is modularized in 3 tiers namely web server, application server and database server.

The specific frameworks and libraries which we are incorporating in our project are as follows:

- CareTaker Frontend is created using React.js and deployed on Netlify.
- CareTaker Backend Server is built on NodeJS.
- Express Framework is used to build rest apis to communicate between frontend and backend.
- MongoDB(NoSQL Database) Atlas is used as a database.
- CareTaker Backend is dockerized and deployed on Azure VM.

## b. Features

- **Student / User**
    - Can signup/signin on the application.
    - Can book a slot(30 min window) for their room cleaning as per the availability on the portal if they don't have an existing active booking.
    - Can view their existing active booking.
    - Can delete their existing booking if any.

- Can edit their room number and other personal details.

- **Admin**
  - Can signup/signin on the application.
  - Can add slots for cleaning which includes room number, date, time slot and number of housekeeping teams.
  - Can view all the active user bookings.
  - Can sort the view of all active user bookings based on room number or time.
  - Can filter the view of all active user bookings based on date.
  - Can download csv of all active bookings as per the sorting and the filter applied.
  - Can edit their personal details.

**c. Why Devops**

DevOps is an organizational approach that facilitates faster application development and easier maintenance of existing deployments by combining development (Dev) and operations (Ops) teams.

DevOps encourages the use of best practices, automation, and new tools to create shorter, more controllable iterations. It maximizes efficiency with automation and improves speed and stability of software development lifecycle.

## 3. System Configuration

**Operating System**

Ubuntu 20.04

**CPU**

4 cores processor

**RAM**

8 GB

**Project Technology Stack Used**

**i. Backend**

Express.js (version 4.x) and Node.js (version 16.x)

**ii. Frontend**

React.js (version 18.x)

iii. **Database**

MongoDB Atlas(version 5.x)

iv. **Cloud**

Azure

**Building Tools**

Node Package Manager (NPM)

**Devops Tools**

- **Source Control Management** - Github
- **Continuous Integration (CI)** - Jenkins and Github Actions
- **Containerization** - Docker
- **Continuous Deployment (CD)**- Ansible
- **Monitoring** - ELK Stack (Elastic Search, Logstash, Kibana)

## 4. Software Development Life Cycle

### a. Frontend

#### i. Installations

**Node JS and npm**

Node.js is an open-source and cross-platform JavaScript runtime environment. npm is the package manager for the Node JavaScript platform. It installs modules and intelligently handles dependency conflicts so that node can locate them.

Use the below commands to install nodejs and npm.

```
sudo apt install nodejs
sudo apt install npm
```

Checking if nodejs and npm installed or not

```
vinayak@vinayak-vivobook-asuslaptop:~$ node -v
v16.15.0
vinayak@vinayak-vivobook-asuslaptop:~$ npm -v
8.5.5
vinayak@vinayak-vivobook-asuslaptop:~$
```

**React**

React.js is the declarative JavaScript framework for creating dynamic client-side applications in HTML. React is based on components. Components let us split the UI into independent, reusable pieces, and think about each piece in isolation.

Use the below command to install react.

```
sudo npm install -g create-react-app
```

Then after installing react, we can create a new react app using the command below.

```
npx create-react-app my-app
cd my-app
npm start
```

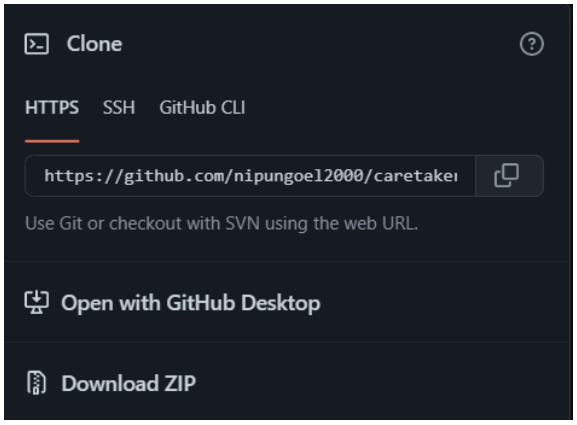
By default, a ReactJS app runs on port 3000. If the port is occupied then it needs to be changed or kill port 3000 process.

ii. **Source Control Management : Git/Github**

Source control Management is used for tracking the file changes history, source code etc. It helps us in many ways in keeping the running project in a structured and organized way. It allows multiple developers from the same team to work on a single project.

In this project, we'll be using Github as SCM Tool which is a cloud based version control system. Here we'll be focusing on the frontend part and the SCM setup, steps, workflow for caretaker-server is in the backend section under SCM.

**Github Repository Link:** https://github.com/nipungoel2000/caretakerClient.git




**Initializing your local project repo with git:**

```
git init
git remote add origin <github repo URL>
```

**Workflow:**

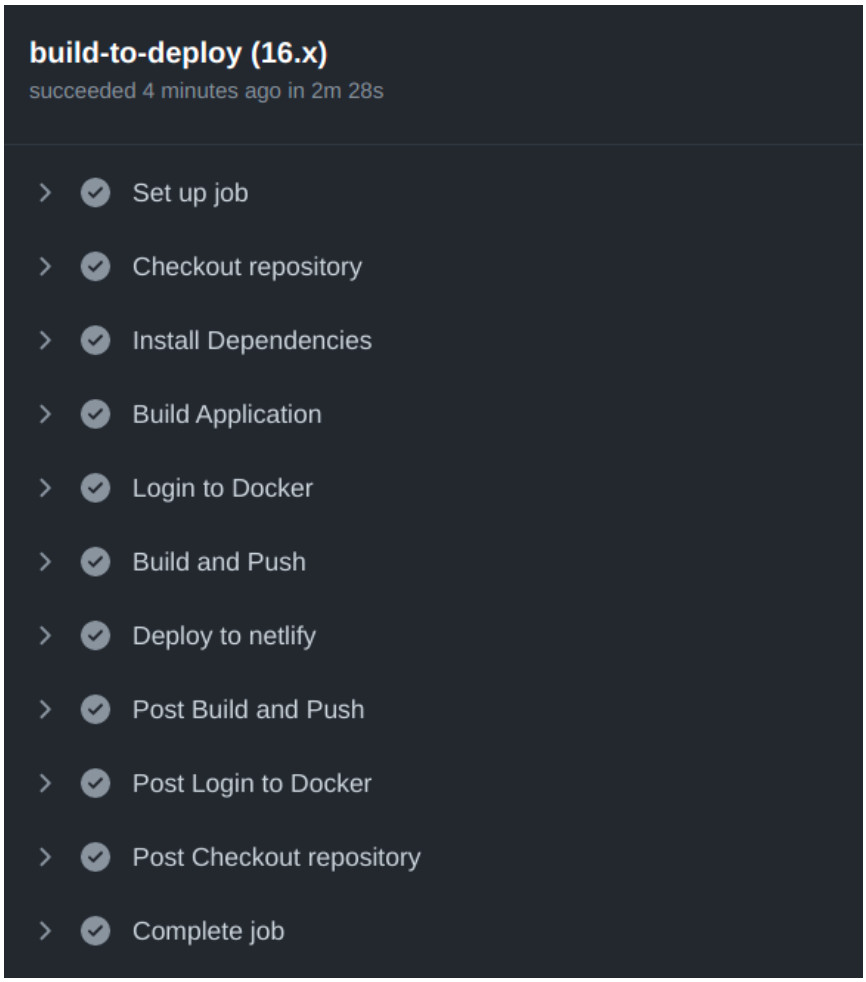
```
git add <files>
git commit -m <Your commit message>
git pull origin main
git push -u origin main
```

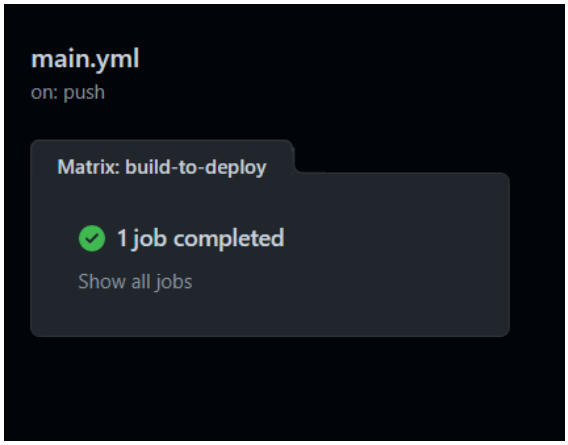
```
nipungoel11@DESKTOP-PT15ILM:/mnt/d/sem8/SPE/CareTaker/caretakerClient$ git pull
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 1), reused 4 (delta 1), pack-reused 0
Unpacking objects: 100% (4/4), 388 bytes | 2.00 KiB/s, done.
From https://github.com/nipungoel2000/caretakerClient
   eebd3e4..b0f5403  main       -> origin/main
Updating eebd3e4..b0f5403
Fast-forward
 netlify.toml      | 5 -----
 public/_redirects | 1 +
 2 files changed, 1 insertion(+), 5 deletions(-)
 delete mode 100644 netlify.toml
 create mode 100644 public/_redirects
```

### iii. Continuous Integration : Github actions

Continuous integration is a DevOps practice code changes are merged into a repository, after which automated builds and tests are run.

Before Github Actions, we had to use a CI/CD tool like Jenkins to trigger the build everytime code is pushed to the repository.
When too many contributors are working on a github project, we can take actions based on the events that take place like push etc. by using github actions. It is simpler and efficient to do this with the help of github actions.
CI/CD pipelines is one of the many workflows offered by github to automate work processes. Each workflow is written in its own file and follows yml syntax. File extension could be yml/yaml.
All the workflow yml files must be placed under the ".github/workflows" directory of your repository. Workflow is a collection of jobs and jobs can be triggered based on the trigger of an event.






Below is the screenshot of the yml file for the workflow using github actions. (main.yml)

```yaml
# This is a basic workflow to help you get started with Actions

name: Build

on:
    push:
        branches: [main]

jobs:
    build-to-deploy:
        runs-on: ubuntu-latest
        strategy:
            matrix:
                node-version: [16.x]
        steps:
            # Get code from repository
            - name: Checkout repository
              uses: actions/checkout@v2.3.3

            # Install all dependencies from a package-lock.json file
            - name: Install Dependencies
              run: npm ci


            # Build the Application for production
            - name: Build Application
              run: CI=false npm run build --prod

            # login to docker
            - name: Login to Docker
              uses: docker/login-action@v1
              with:
                username: ${{ secrets.DOCKER_USERNAME }}
                password: ${{ secrets.DOCKER_ACCESS_TOKEN }}

```

```yaml
      # login to docker
      - name: Login to Docker
        uses: docker/login-action@v1
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_ACCESS_TOKEN }}

      # build and push the docker image
      - name: Build and Push
        id: docker_build
        uses: docker/build-push-action@v2
        with:
          context: ./
          file: ./Dockerfile
          push: true
          tags: nipungoel2000/client:latest


      # Deploy to Netlify using our production secrets
      - name: Deploy to netlify
        uses: nwtgck/actions-netlify@v1.1
        with:
            publish-dir: './build'
            production-branch: main
            github-token: ${{ secrets.GITHUB_TOKEN }}
            deploy-message: 'Deploy from GitHub Actions'
            enable-pull-request-comment: false
            enable-commit-comment: true
            overwrites-pull-request-comment: true
        env:
            NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
            NETLIFY_SITE_ID: ${{ secrets.NETLIFY_SITE_ID }}
        timeout-minutes: 1
```

### iv. Build : npm

We are using Node Package Manager (npm) for build. To initialize the project we need to run the npm init command in the project directory.

```
vinayak@vinayak-vivobook-asuslaptop:~/caretaker/client$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (client)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/vinayak/caretaker/client/package.json:

{
  "name": "client",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes)
```

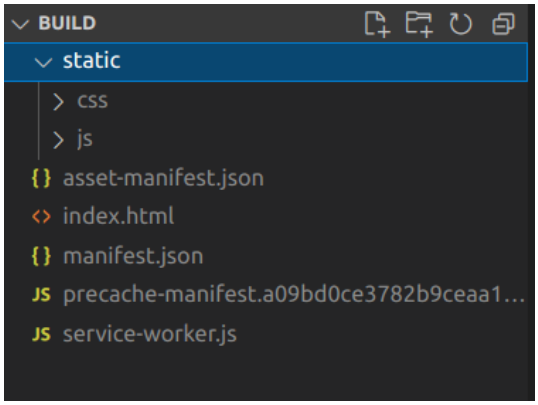
After that we write code for the project and finally do npm build which converts all the components and modules into compiled HTML, CSS and JavaScript files in a folder named build.

```
vinayak@vinayak-vivobook-asuslaptop:~/Semester8/SPE/caretaker/client$ npm run build --prod

> client@0.1.0 build
> react-scripts build

Creating an optimized production build...
Compiled with warnings.
```

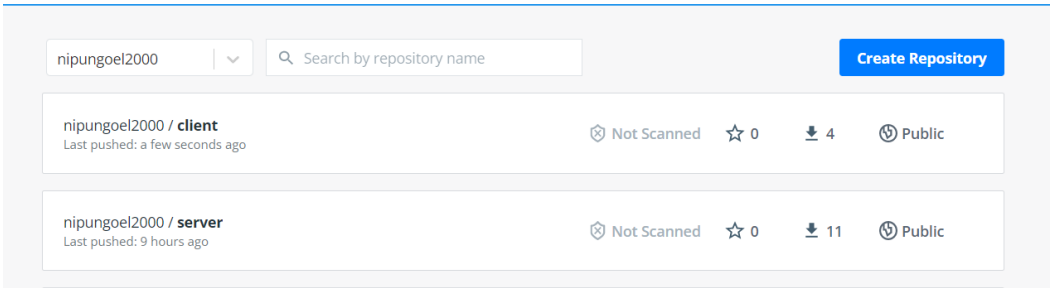
**--prod** flag minimizes and optimizes the project to a single file depending on type. It also ignores plugins/imports which have not been used but are included.



### v. Containerization : Docker/Dockerhub

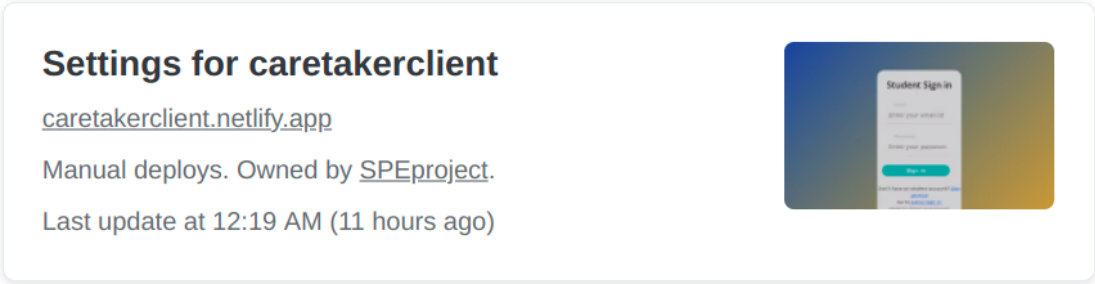
Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

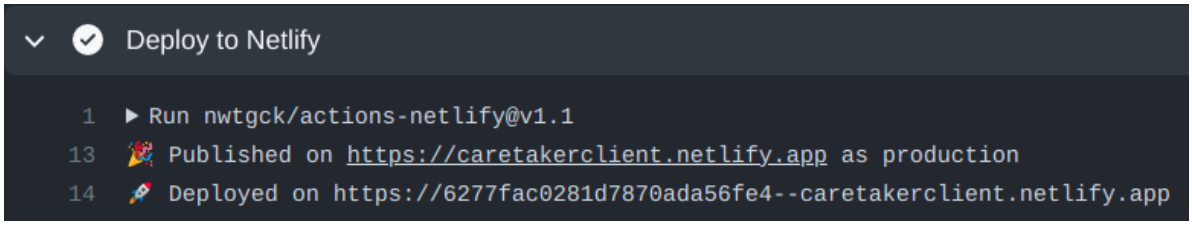
- We build a docker image of the frontend code of our project. This docker image is then pushed to dockerhub.
  https://hub.docker.com/r/nipungoel2000/client




### vi. Continuous Deployment : Netlify

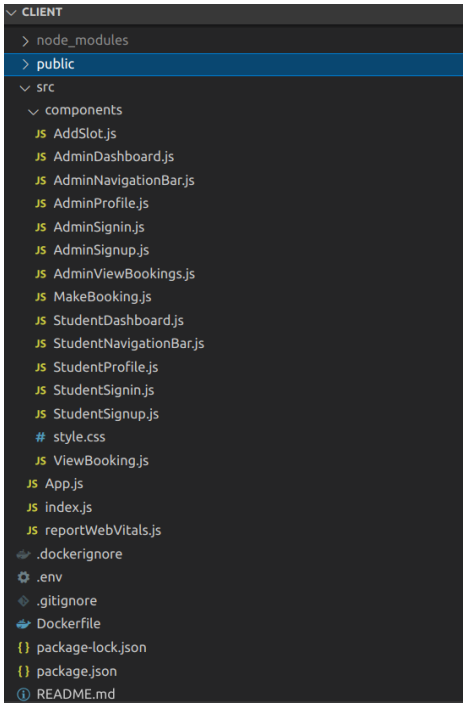
Netlify is a cloud based platform for automating web applications which offers dynamic functionality like serverless backend services and hosting for web applications. Continuous Deployment allows to automatically update development or production sites built with Netlify, as changes are pushed into the github repository.




Netlify provides a public URL for the web application.

## vii. Frontend Code Walkthrough




- Components are the building blocks for the react project. This folder contains a collection of UI components such as dashboard, Sign In, Signup, NavigationBar etc., that can be used in several files throughout the project. All the components are stored in the /**components** directory.
- All the routings are configured in **app.js**
- All the secret variables are stored in **.env**
- All the workflow files are stored in the **.github** directory.

**Making the API calls from frontend to backend:**

API calls are done using axios. Axios makes it easy to send asynchronous HTTP requests to REST endpoints and perform CRUD operations.

```
import axios from "axios";
```

```
let slot_data = {date:date_str,startTime:stime_str,endTime:etime_str,floor:floorNum,teams:numTeams};
await axios({
    url: serverURL+"slot/add",
    method: "POST",
    data: slot_data,
}).then((res) => {
    console.log(res);
    if(res.status==201)
    {
    alert("Slot added successfully");
    window.location="/addslot";
    }
    else
    {
    alert("Internal Server Error : " + res.status);
    }
})
.catch((err) => {
    console.log(err);
    if(err.response)
      alert(err.response.data.message);
    else
      alert("Internal Server Error");
})
```

**Managing Project Dependencies:**

A package. json is a JSON file that exists at the root of a project. It holds metadata relevant to the project and it is used for managing the project's dependencies ,version , scripts, etc.

```
"name": "client",
"version": "0.1.0",
"private": true,
"dependencies": {
  "@testing-library/jest-dom": "^5.16.3",
  "@testing-library/react": "^12.1.4",
  "@testing-library/user-event": "^13.5.0",
  "axios": "^0.26.1",
  "bootstrap": "^5.1.3",
  "dotenv": "^16.0.0",
  "js-file-download": "^0.4.12",
  "react": "^18.0.0",
  "react-bootstrap": "^2.3.0",
  "react-csv": "^2.2.2",
  "react-dom": "^18.0.0",
  "react-flatpickr": "^3.10.11",
  "react-router-dom": "^6.3.0",
  "react-scripts": "^2.1.8",
  "web-vitals": "^2.1.4"
},
▷ Debug
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

**Authenticating Users:**

To authenticate a user, a client application must send a JSON Web Token (JWT) in the authorization header of the HTTP request to the backend API. If the user is a validated

user then it can access the desired resources and if not then we are navigating it to the admin/student sign in page.

```
useEffect(() => {
  if(localStorage.getItem('token') && localStorage.getItem('role')==='admin'){
    navigate('/adminDashboard');
  }
  if(localStorage.getItem('token') && localStorage.getItem('role')==='student'){
    navigate('/studentDashboard');
  }
});
```

## b. Backend

### i. Installations

#### Nodejs and npm

To run node applications server, install nodejs and npm.

```
sudo apt install nodejs
sudo apt install npm
```

Checking if nodejs and npm installed or not

```
vinayak@vinayak-vivobook-asuslaptop:~$ node -v
v16.15.0
vinayak@vinayak-vivobook-asuslaptop:~$ npm -v
8.5.5
vinayak@vinayak-vivobook-asuslaptop:~$
```

#### Express

Express is a Node.js web application framework that offers a comprehensive collection of functionality for building web and mobile applications.

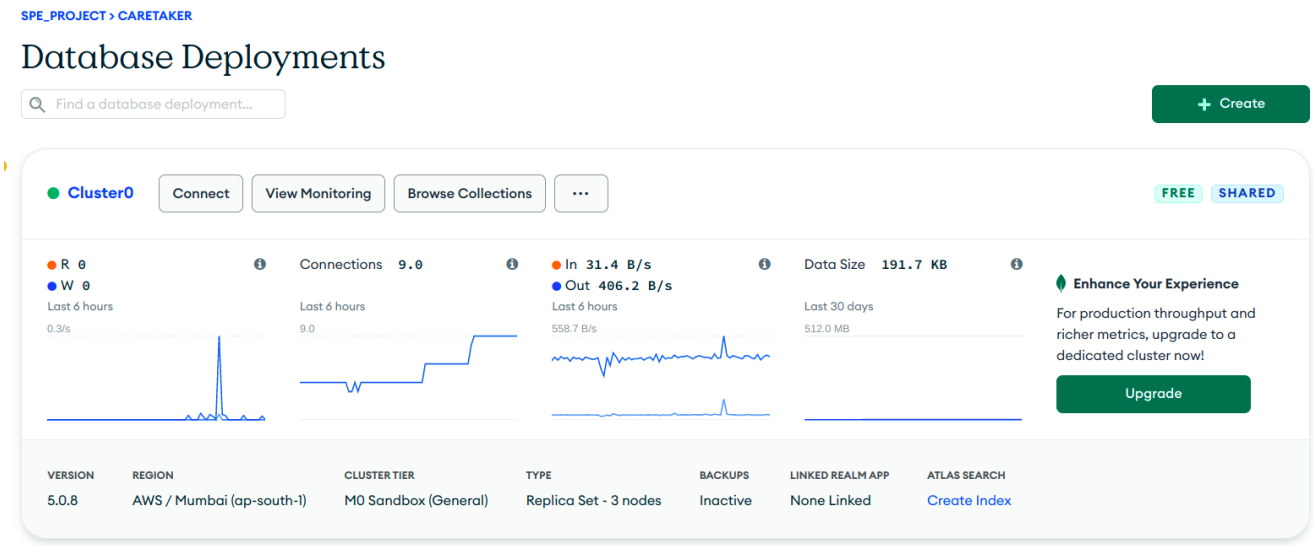
To install Express framework use the below command. Run this command in the server directory.

```
npm install express --save
```

#### MongoDB

MongoDB is an open-source document database that provides high performance, availability, and automatic scaling. It is a schemaless NoSQL based database.
We are using MongoDB atlas which is a cloud application data platform. We don't need to install it in our local system. We can simply sign up on MongoDB atlas and start using it. Below is the screenshot of the connection details of MongoDB. This is kept in the dot env file.

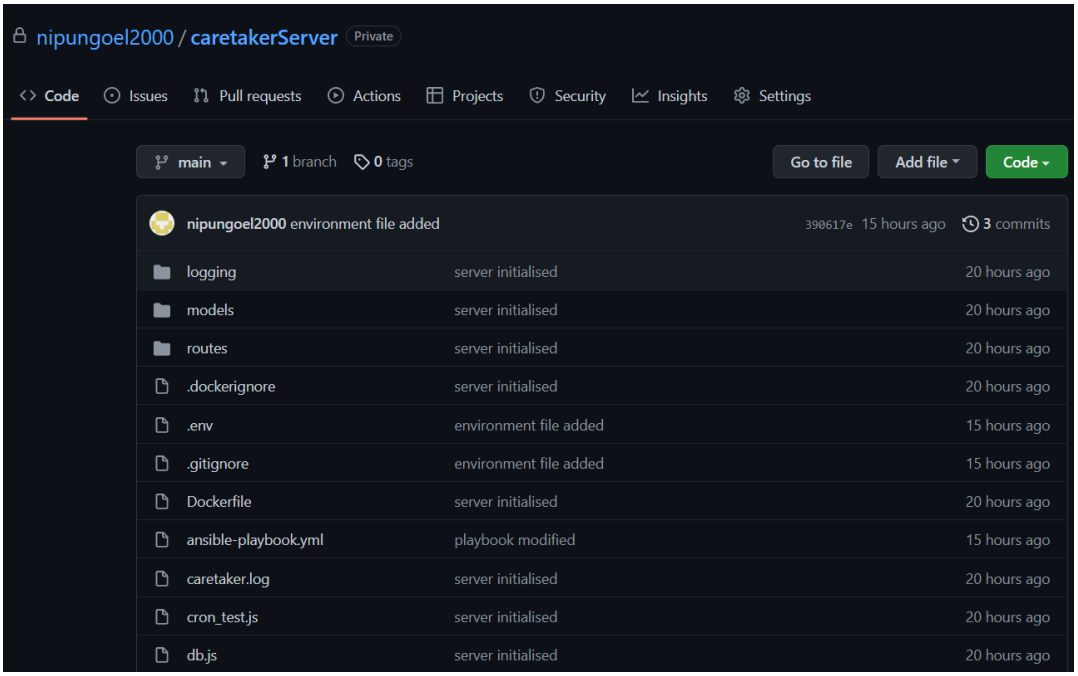
```
DB_URI = "mongodb+srv://admin:admin@cluster0.oi62x.mongodb.net/myFirstDatabase?retryWrites=true&w=majority"
```




## ii. Source Control Management : Git/Github

We are using github for version control in our backend.

**Github Repository Link:** https://github.com/nipungoel2000/caretakerServer.git




### iii. Continuous Integration : Jenkins

Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration purposes. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.

**Installation of Jenkins**

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

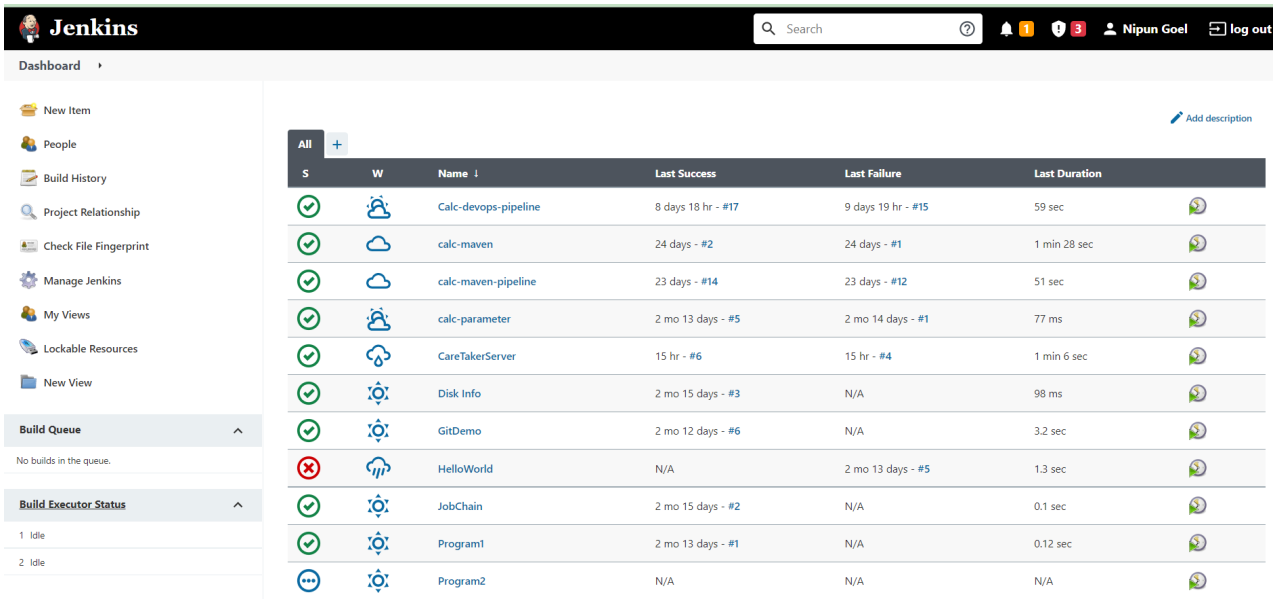
$ sudo sh -c 'echo deb https://pkg.jenkins.io/debian binary/ > /etc/apt/sources.list.d/jenkins.list'

$ sudo apt-get update

$ sudo apt-get install jenkins

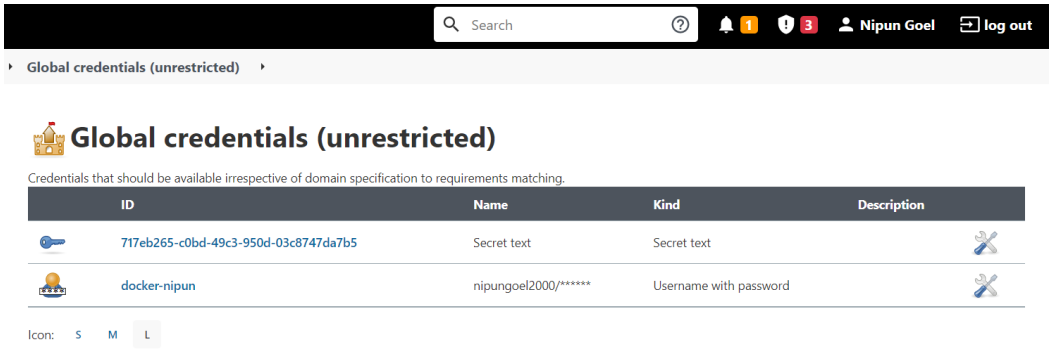
```
nipungoel11@DESKTOP-PT15ILM:~$ sudo service jenkins start
Correct java version found
 * Starting Jenkins Automation Server jenkins
nipungoel11@DESKTOP-PT15ILM:~$ sudo service jenkins status
Correct java version found
Jenkins Automation Server is running with the pid 270
```

Jenkins runs on the localhost at port 8080 by default.




## CI with Jenkins

- Install the following necessary plugins by going to Dashboard/manage Jenkins/ manage plugins/available :
    - Docker Plugin
    - Docker Pipeline
    - Git
    - Github Integration Plugin
    - Ansible Plugin
- Add your Docker Hub username, password in Jenkins by :
    - Going to manage jenkins/manage credentials/jenkins/Global Credentials/
    - click on add credentials and add your Docker Hub username, password, and give an ID to these credentials.




- Now in order to access the azure VM remote server via Jenkins type the following commands:

  $ sudo su jenkins

  $ ssh-keygen -t rsa

  $ ssh-copy-id REMOTEUSER@<REMOTE IP ADDRESS>
- Now, create a new job for Jenkins. Enter jenkins job name and choose pipeline as job functionality. Benefits of pipeline is that it is script based and each stage of pipeline script runs one after another.
- Add the below shown stages in the pipeline :
  - **Step 1: Git Clone**

    stage('Step 1. Pull source code from github') {

    steps {

    git branch: 'main', url: 'https://github.com/nipungoel2000/caretakerServer.git'

    }

    }
  - **Step2 : Install npm dependencies**

    stage('Step 2: Install dependencies'){

    steps{

    sh 'npm install'

    }

    }
  - **Step 3: npm test with Jest**

    stage('Step 3: Testing with Jest'){

    steps{

    sh 'npm run test'

```
    }
  }
```

- **Step 4: Build docker image**

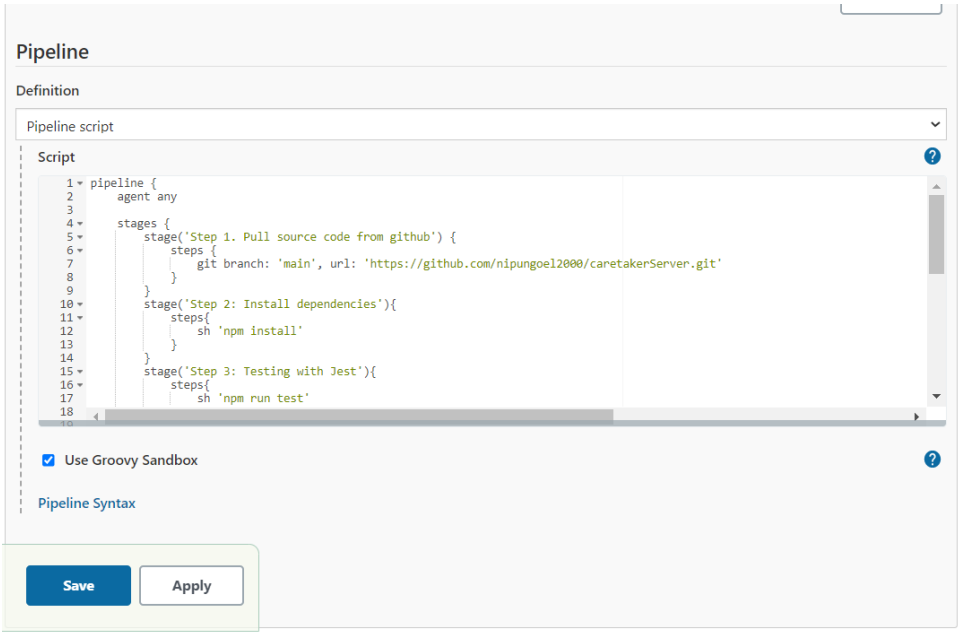
```
stage('Step 4: Build Docker Image') {
  steps{
    script {
      imageName=docker.build "nipungoel2000/server"
    }
  }
}
```

- **Step 5: Push docker image to dockerhub**

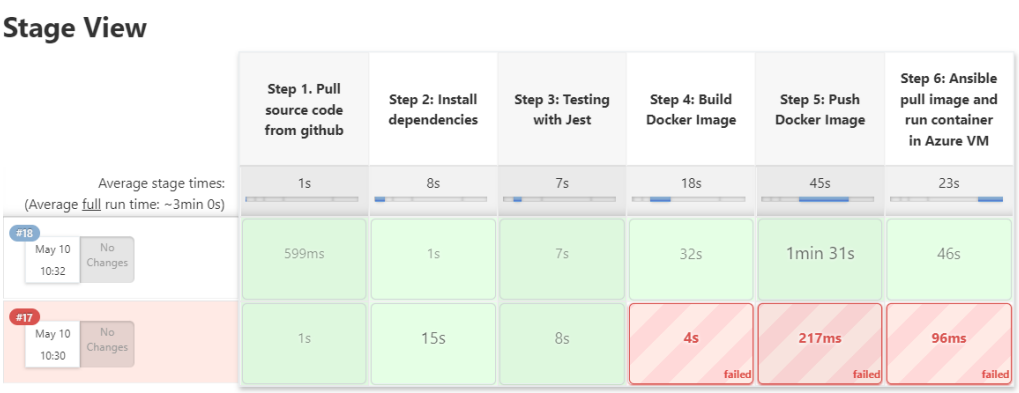
```
stage('Step 5: Push Docker Image') {
  steps {
    script{
      docker.withRegistry('','docker-nipun'){
      imageName.push()
      }
    }
  }
}
```

- **Step 6: Pull Image on Azure VM via Ansible**

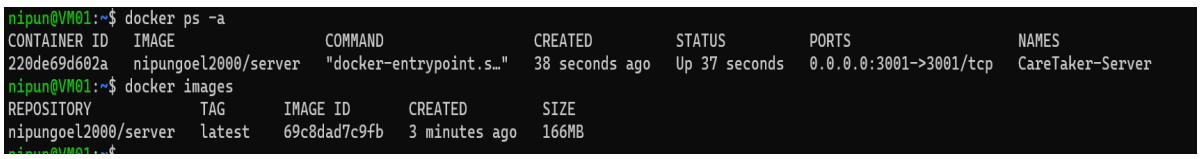
```
stage('Step 6: Ansible pull image and run container in Azure VM') {
  steps {
    ansiblePlaybook becomeUser: null, colorized: true, disableHostKeyChecking: true,
installation: 'Ansible', inventory: 'inventoryFile', playbook: 'ansible-playbook.yml', sudoUser:
null
    }
  }
```




- Click on build now to execute the pipeline. Result of each stage of the pipeline and the console output after the build completes can be seen as below:




- After the successful build, you will be able to see the docker image on the Azure VM(managed nodes). Also, since there is a docker run command in the pipeline, you will also be able to see a docker container with backend server running.



### iv. Testing : Jest

Testing of backend node js server code is automatically triggered after the build process through the pipeline using Jest.

Jest is an Opensource JavaScript Testing Framework for creating, and running tests.

#### Installation of Jest

Jest and supertest can be installed using npm. Use the below to install jest.

```
npm install --save-dev jest supertest
```

#### Configuring NPM script

To run tests from the command line - firstly open the package.json file and then configure a script named test for running Jest.

```
"scripts": {
  "test": "NODE_OPTIONS='--experimental-vm-modules' jest",
  "dev": "nodemon index.js"
},
```

#### Writing test cases

Create a file index.test.js

All the test cases are written in this file.

Test cases are written as follows.

```
// Tested
describe("post request /admin/signin", () => {
    test("Status Code must be 201 and type must be json list", async () => {
        const response = await request(app).post("/admin/signin").send({
            email: "test_admin@gmail.com",
            password: "Test@123"
        })
        expect(response.statusCode).toBe(201);
        expect(response.type).toBe("application/json");
    })
    test("Status Code must be 401 and type must be json list", async () => {
        const response = await request(app).post("/admin/signin").send({
            email: "test_admin@gmail.com",
            password: "testpassword"
        })
        expect(response.statusCode).toBe(401);
        expect(response.type).toBe("application/json");
    })
})
```

**Running test cases**

Test cases are run using the npm run test command.

```
vinayak@vinayak-vivobook-asuslaptop:~/Semester8/SPE/caretaker/server$ npm run test

> server@1.0.0 test
> NODE_OPTIONS='--experimental-vm-modules' jest
```

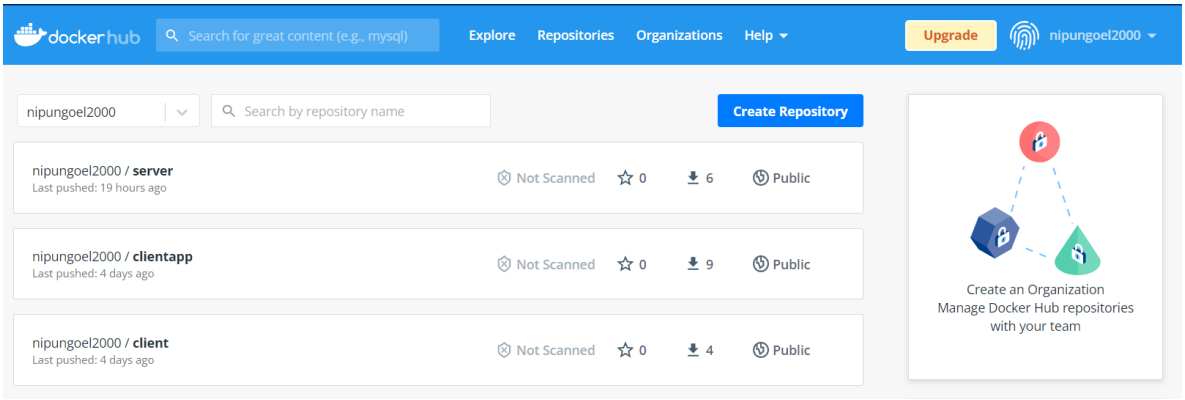
```
Test Suites: 1 passed, 1 total
Tests:       11 passed, 11 total
Snapshots:   0 total
Time:        3.251 s, estimated 4 s
Ran all test suites.
```

v. **Containerization : Docker/Dockerhub**

Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

- We create a docker image of the backend code of our project after npm build and npm run test. This docker image is then pushed to dockerhub.

- This push procedure will be done after every build which would include removing the previously built docker image and replacing it with the latest built on docker hub. The pushed images can be found at the following link : https://hub.docker.com/r/nipungoel2000/server




**Installing Docker :**

$ sudo apt install docker.io

$ sudo systemctl start docker

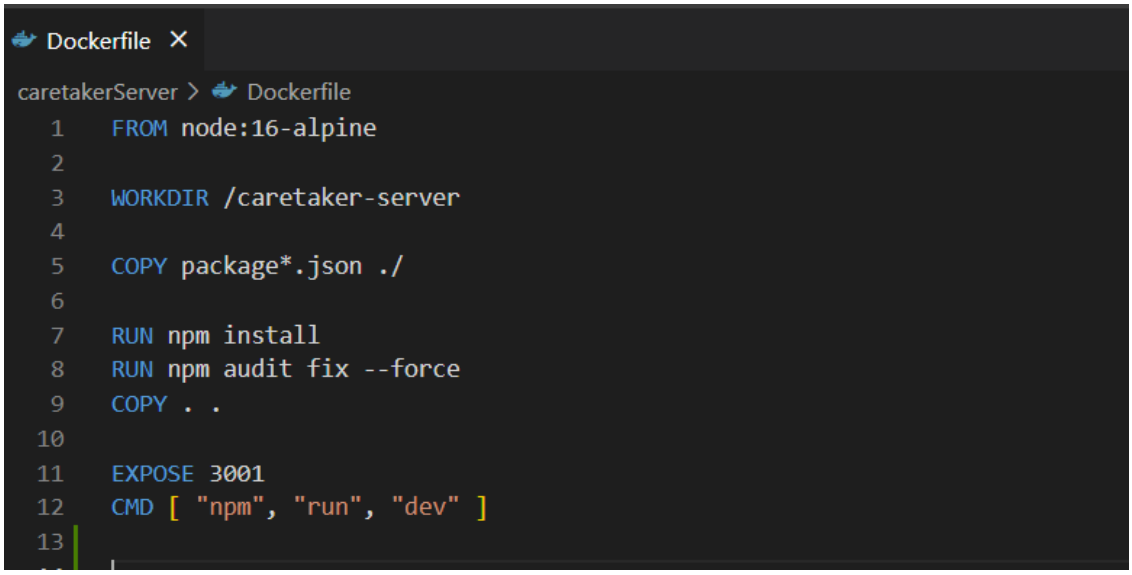
$ sudo systemctl status docker

Jenkins should be added to the docker group so that jenkins can use docker for building docker images. To do so below command should be used:

$sudo usermod -aG docker jenkins

**DockerFile**

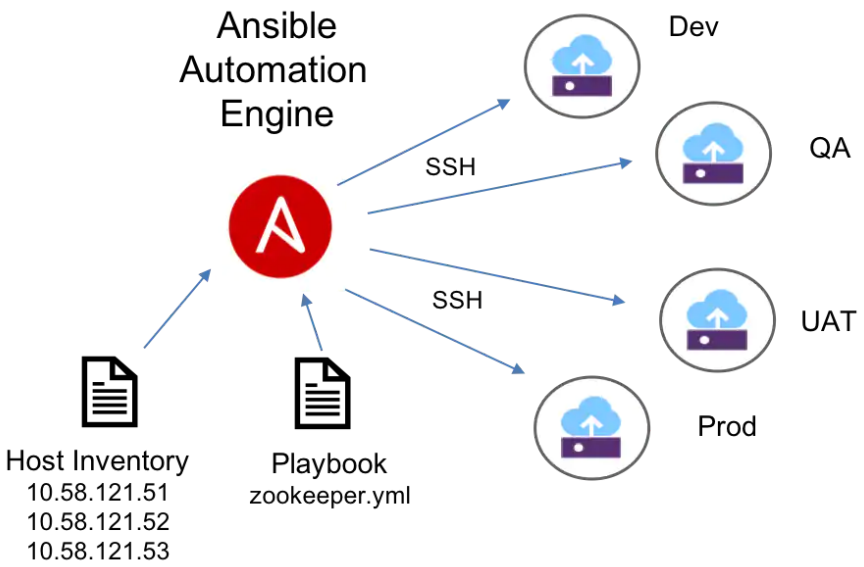
Dockerfile is essentially the build instructions to build the image. In the dockerfile we mention all the commands in succession which need to be run in order to create an image.




**Command to build docker image :** docker build -t <image-name> .

### vi. Continuous Deployment : Ansible

- Ansible is an open source IT configuration management, deployment, and orchestration tool. It empowers DevOps teams to define their infrastructure as a code in a simple and declarative manner.
- It uses no agents and no additional custom security infrastructure, so it's easy to deploy and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allows you to describe your automation jobs in a way that approaches plain English.
- It can be used to push anything to multiple ”managed hosts”. Ansible is installed and runs from the control node. A control node houses all the copies of your Ansible project files and configuration information.
- Managed hosts are systems to which the application/infrastructure as code is pushed.
- Ansible acts as a central distributor from the control node to all the managed hosts. The managed hosts are listed in the inventory file.




We are using Ansible to fetch and run the docker image of the server pushed onto the docker hub onto our azure virtual machine.

**Installing Ansible :**

$ sudo apt update

$ sudo apt install ansible

```
nipungoel11@DESKTOP-PT15ILM:/mnt/d/sem8/SPE/CareTaker$ ansible --version
ansible 2.9.6
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/nipungoel11/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, Nov 26 2021, 20:14:08) [GCC 9.3.0]
```

**Inventory :**

The Ansible inventory file defines the hosts and groups of hosts upon which commands, modules, and tasks in a playbook operate.

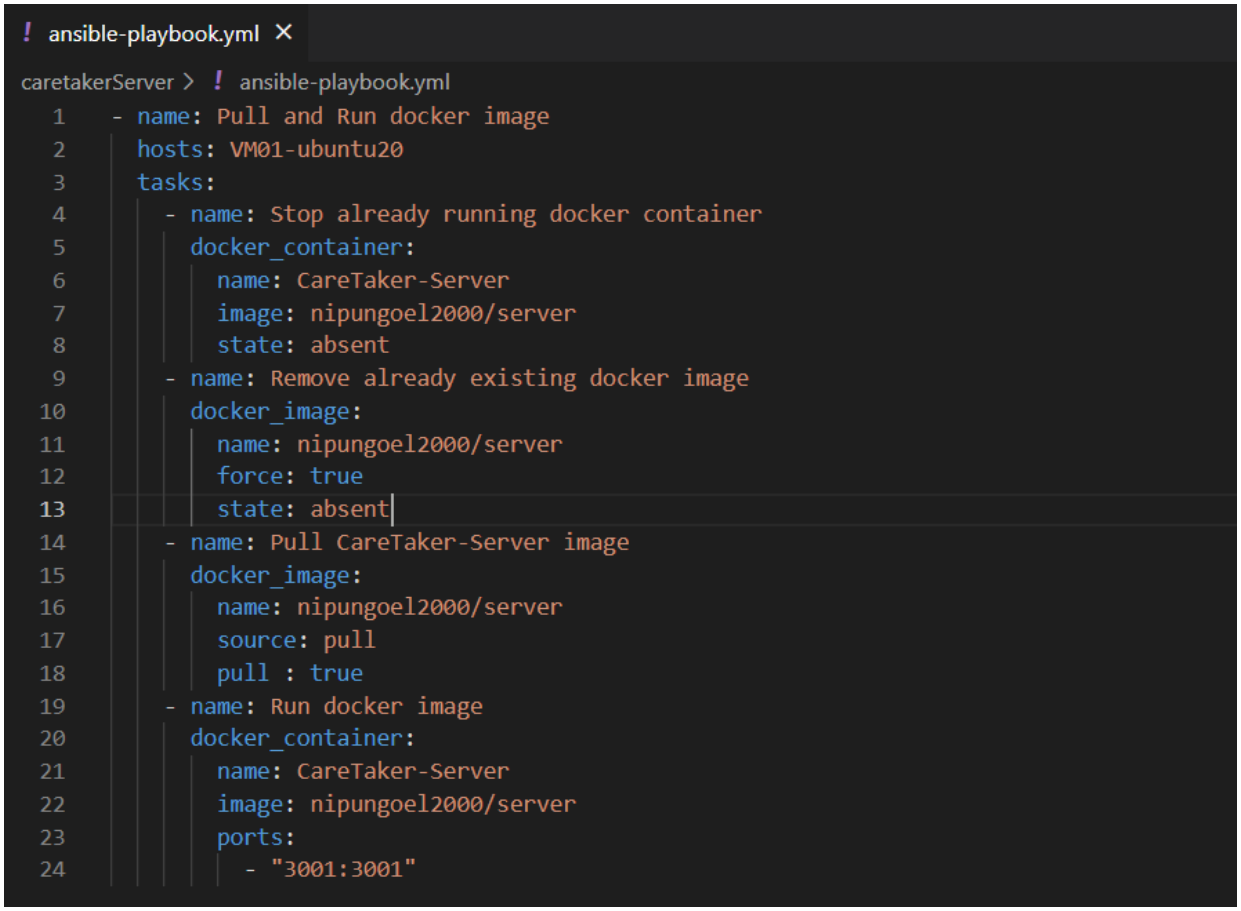
inventoryFile

caretakerServer > inventoryFile

```
1  [VM01-ubuntu20]
2  20.213.240.131 ansible_user=nipun
3
```

**Playbook :**

Ordered lists of tasks, saved so you can run those tasks in that order repeatedly. Playbooks can include variables as well as tasks. Playbooks are written in YAML and are easy to read, write, share and understand.




### vii. Nginx Reverse proxy in Azure VM

A reverse proxy listens for client web requests and forwards them directly to your app or website and delivers the server's response back to the client in real-time.

Nginx is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache.

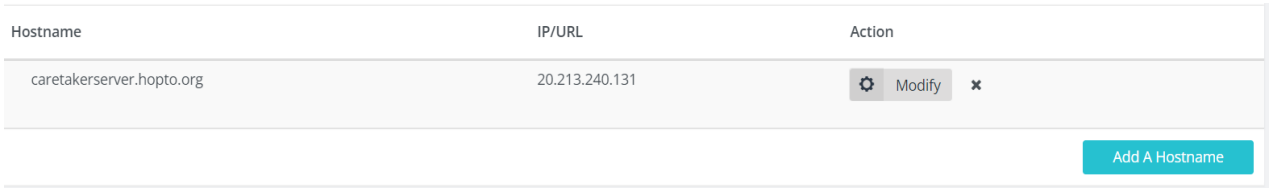
**Setting up NGINX reverse proxy:**

```
sudo apt update
sudo apt install nginx
```

After installing NGINX, the configuration files can be found in the /etc/nginx folder.

**Getting the domain name :**

Domain name was taken from noip.com for our backend hosted on Azure VM.




**Certbot for SSL Certificate :**

The certbot obtains free HTTPS certificates for our application and also handles the renewal process automatically. It also redirects all unencrypted HTTP connections to HTTPS.

We need to install certbot and also a plugin for our NGINX server.

```
sudo apt install certbot
sudo apt install python3-certbot-nginx
```

After installing the above packages, we can obtain our certificates.

```
sudo certbot --nginx -d caretakerserver.hopto.org
```

This automatically generates the certificate in the NGINX.conf.

**Our Config file :**

All the requests on caretakerserver.hopto.org are redirected to 127.0.0.1:3001 in our Azure VM.

```
server {
        root /var/www/html;

        server_name caretakerserver.hopto.org;

        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                #try_files $uri $uri/ =404;
                proxy_pass http://127.0.0.1:3001;
        }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/caretakerserver.hopto.org/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/caretakerserver.hopto.org/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}
```

```
server {
    if ($host = caretakerserver.hopto.org) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80 default_server;
        listen [::]:80 default_server;

        server_name caretakerserver.hopto.org;
    return 404; # managed by Certbot

}
```

A softlink of the above config file is also created in the sites-available folder and then nginx service is restarted.

```
ln -s ../sites-available/* .
systemctl restart nginx
```

## viii. Monitoring

### Installing ELK stack

First clone the repository using the below command in terminal

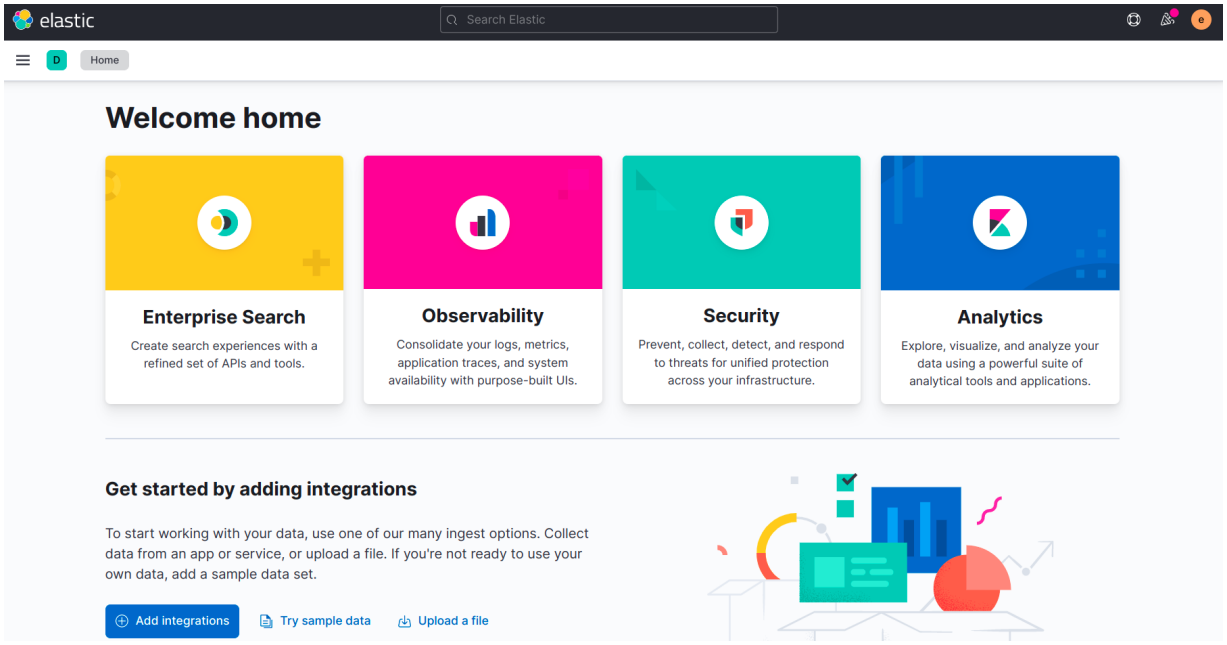
$ git clone https://github.com/deviantony/docker-elk.git

Now, to run the stack run the command

$ cd docker-elk

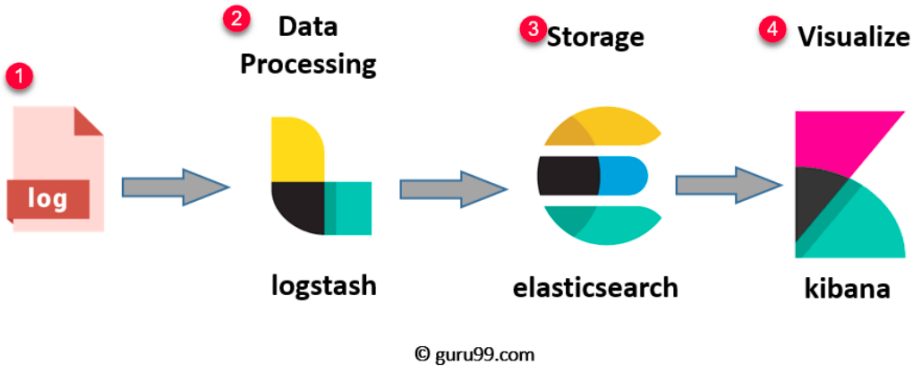
$ docker-compose up -d

Kibana server starts at port:5601 and Elastic search starts at port:9200




The ELK Stack is a log management platform made up of three open-source Elasticsearch, Logstash, and Kibana products.

Elasticsearch indexes and saves the information that Logstash collects and parses. After Kibana creates visualizations of the data.




For logging we are using winston which is a logging library. First install it by using the below command.

```
npm install winston --save
```

After that we have created a folder named logging in our project and added a file logconfig.js containing the winston configuration for the project.

logConfig.js M

home > vinayak > Semester8 > SPE > caretaker > caretakerServer > logging > logConfig.js > ...

```
const winston = require('winston');

const logConfig = {
    'transports': [
        new winston.transports.File({
            filename: './caretaker.log'
        })
    ],
    format: winston.format.combine(

        winston.format.timestamp({
            format: 'YYYY-MM-DD HH:mm:ss'
        }),
        //${info.level}: ${info.label}:
        winston.format.printf(info => `${[info.timestamp]}, ${info.level}, ${info.message}`),
    )
};
const logger = winston.createLogger(logConfig);
module.exports = logger;
```

After configuring it, we can use it in any file by importing the logger.

Below is the screenshot of the usage of logger in userstudent.js file.

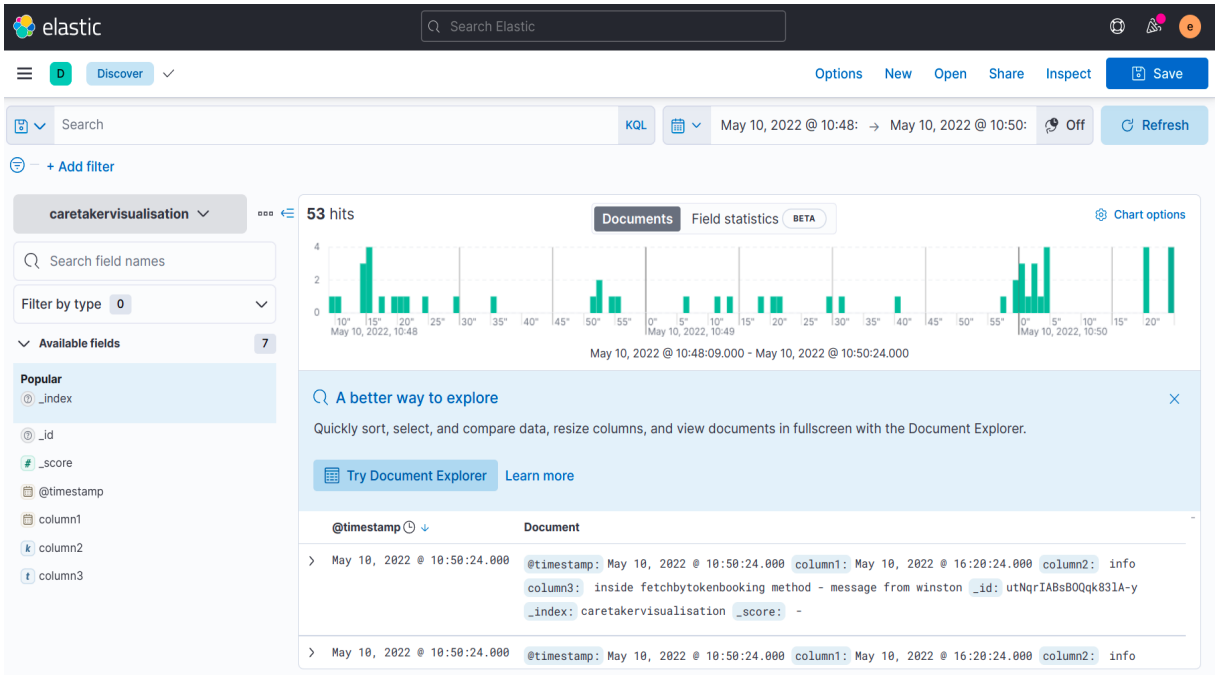
```
const savedStudent = await newStudentUser.save();
console.log("savedStudent" + `${savedStudent}`);
logger.info("student user signed up successfully - message from winston");
return res.status(201).send({message : "User created successfully" });
```

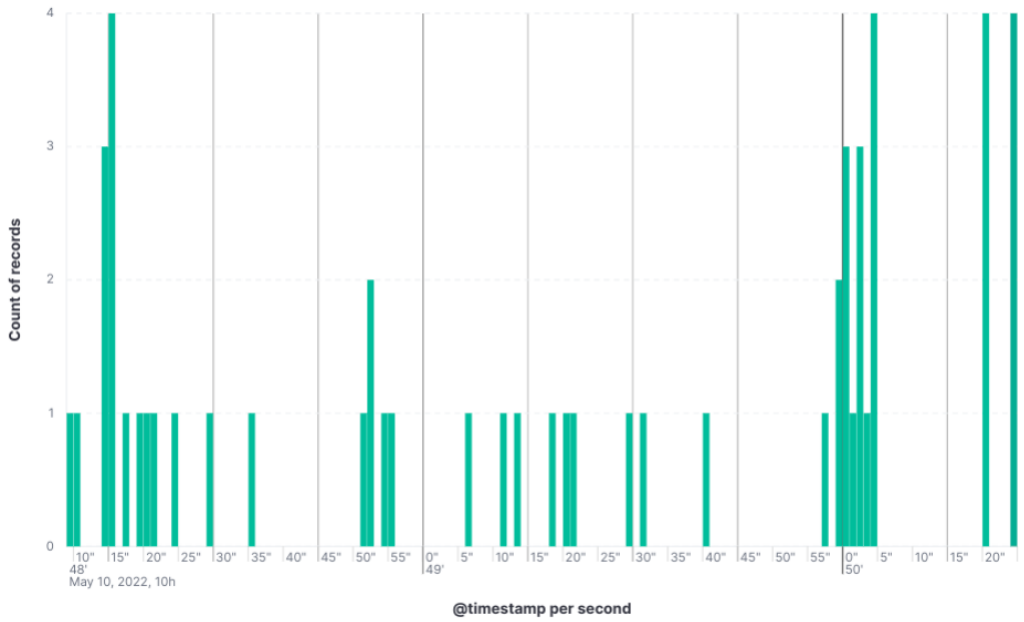
Follow the steps below for Continuous monitoring using ELK stack:-

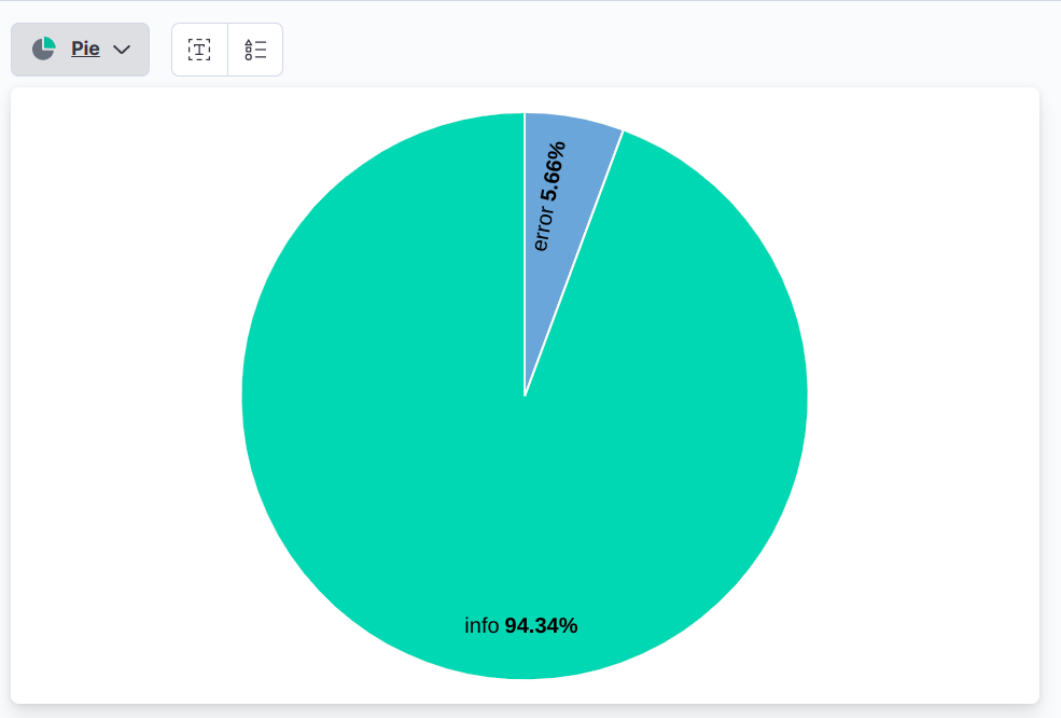
(a) Open the link [http://localhost:5601](http://localhost:5601)

(b) Then using the log file generated “caretaker.log” to elasticsearch for continuous monitoring.

## Visualizations using ELK stack



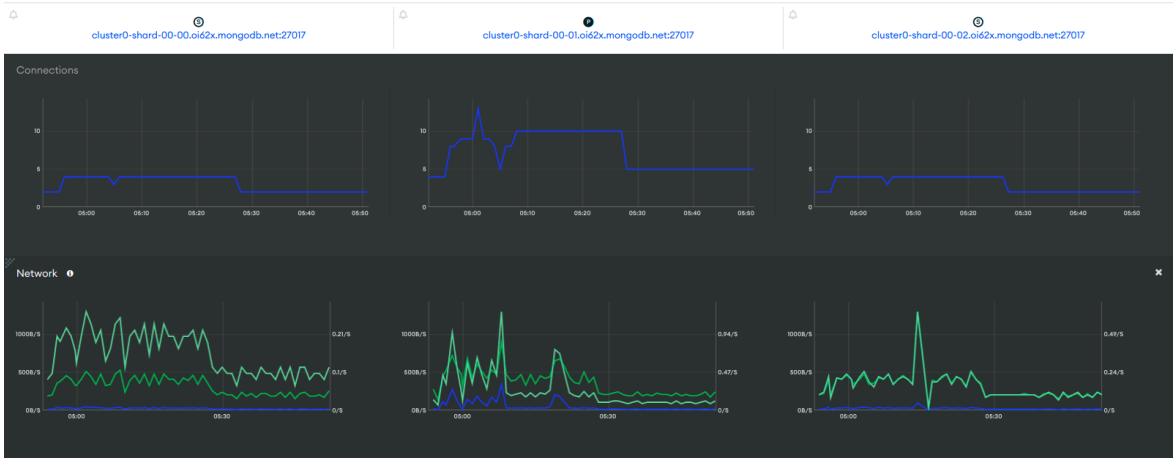






(pie chart visualization in kibana)

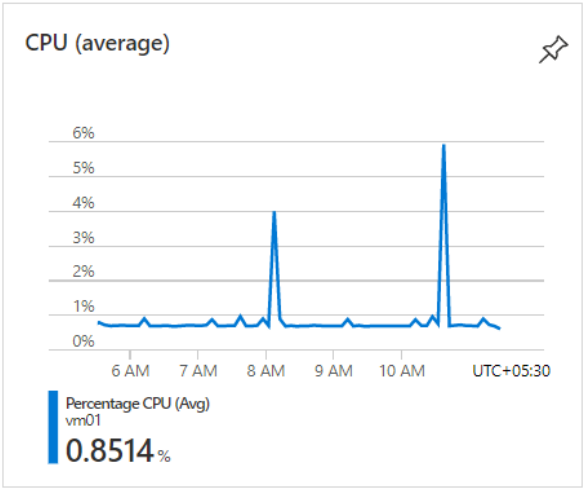
## Mongodb monitoring

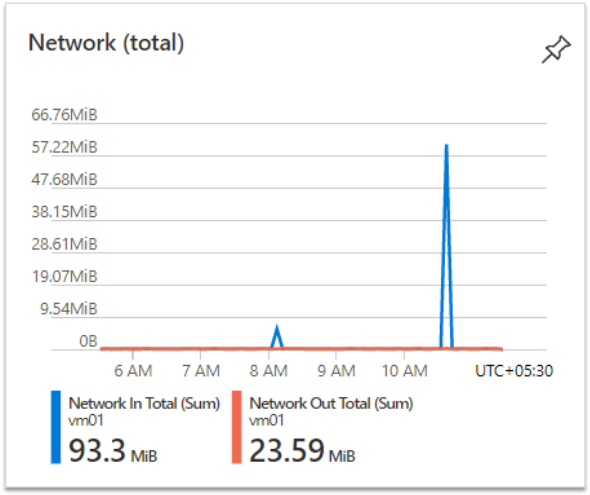
Key Metrics like Connections, Network etc., are monitored using Mongodb monitoring.

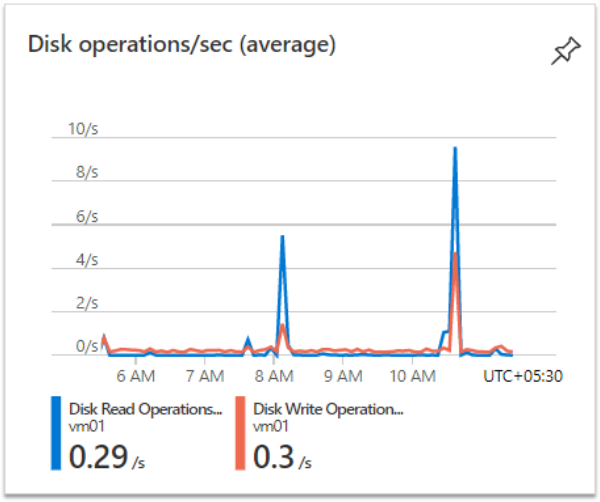


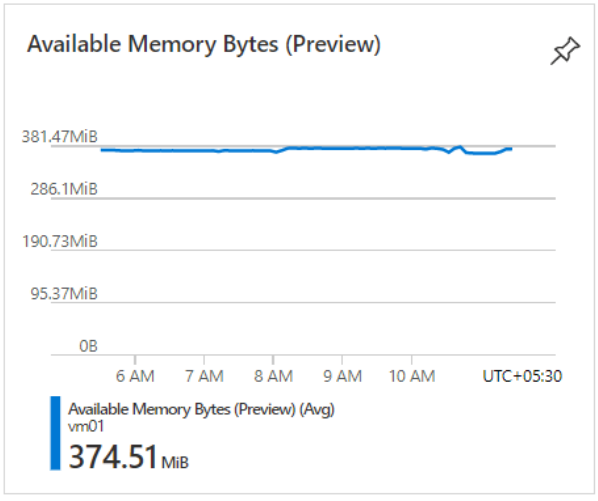
**Monitoring Key metrics with Azure**

Key Metrics like CPU Usage, network, disk operations, available memory etc., are monitored using Azure VM monitoring.

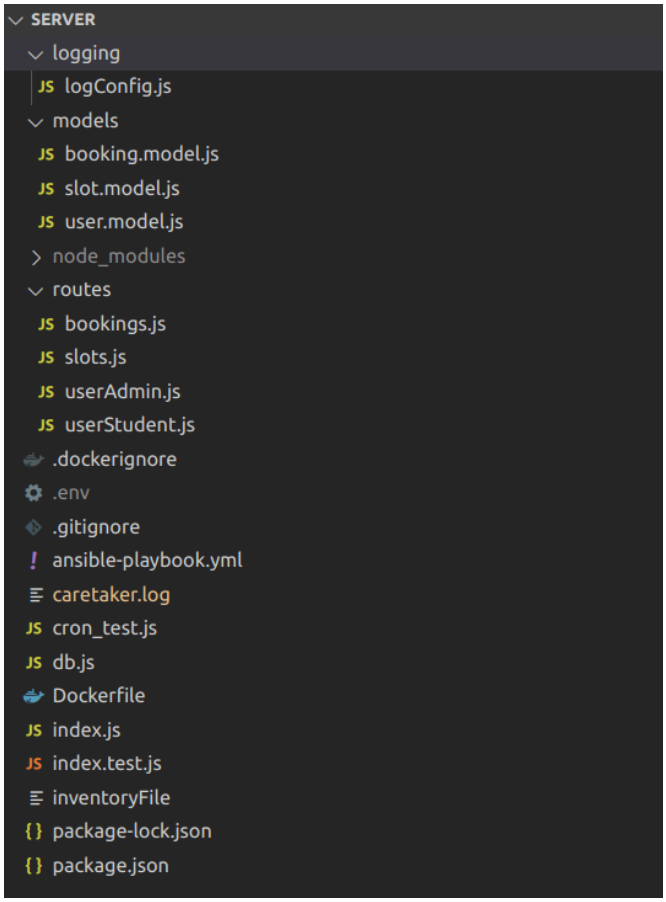



## ix. Backend Code Walkthrough




- All the routes are stored in the **/routes** directory.
- All the database schemas for booking, slot and user are stored in **/models** directory.
- logConfig file is stored in **/logging** directory.
- Dockerfile, inventory file, ansible playbook file are stored in the root project directory.
- **db.js** which makes the connection with the database is stored in the root project directory.
- All the secret variables are stored in the **.env** file.

### Routes

Api endpoints in the backend are written with router.post/router.get/router.delete methods. These routes are stored in the **/routes** directory.

```
//req.body(token)
router.post("/fetchbytoken",async (req,res) =>
{
    logger.info("inside fetchbytokenbooking method - message from winston");
    try{
        const token=req.body.token;
        jwt.verify(token,process.env.JWTPRIVATEKEY,async (err,decodedToken) =>{
            if(err){
                logger.error("unauthorized access inside fetchbytokenbooking method - message from winston");
                res.status(400).send({message:err});
            }
            else{
                // console.log(decodedToken);
                let userdata=await userModel.findOne({_id:decodedToken._id});
                const email=userdata.email;
                let bookingData = await bookingModel.findOne({email:email, status:"active"});
                let bookings=[];
                bookings.push(bookingData);
                res.status(201).send({bookingData:bookings});
            }
        })
    }
    catch(err){
        logger.error("error inside fetchbytokenbooking method - message from winston");
        res.status(500).send({message:err});
    }
});
```

### Models

Schemas are written in /models directory. Below is an example of user schema.

Then the collection can be created with the mongoose.model method which takes 2 parameters: Collection name and collection schema. The method returns the mongoose object.

```
const userSchema = new Schema(
    {
        firstName: {type: String, required: true},
        lastName: {type: String, required: true},
        email: {type: String, required:true, unique: true},
        password: {type: String, required: true},
        type: {type: String,required: true},
        roomNum: {type: Number,default:null,required:false}
    },
    {timestamps: true}
);

userSchema.methods.generateAuthToken = function () {
    const token = jwt.sign({ _id: this._id }, process.env.JWTPRIVATEKEY, {
        expiresIn: "7d",
    });
    return token;
};
const userModel = mongoose.model("user",userSchema,"UsersData");
```

## 5. API Documentation

### a. Student signup :

Endpoint : https://caretakerserver.hopto.org/student/signup

HTTP method : POST

Description : It will add a new user to the database.

Request body :

```
{
  "firstName": "string",
  "lastName": "string",
  "email":"string",
  "password":"string"
}
```

Response body :

```
{
   "message": "string"
}
```

### b. Student sign in :

Endpoint : https://caretakerserver.hopto.org/student/signin

HTTP method : POST

Description : It will authenticate a user.

Request body :

```
{
  "email": "string",
  "password": "string"
}
```

Response body :

```
{
  "data": "string",
  "message": "string"
}
```

### c. Get student data :

Endpoint : https://caretakerserver.hopto.org/student/getdata

HTTP method : POST

Description :To fetch the student details.

Request body :

```
{
  "token": "string"
}
```

Response body :

```
{
    "data": {
        "_id": "ObjectId",
        "firstName": "string",
        "lastName": "string",
        "email": "string",
        "password": "string",
        "type": "string",
        "roomNum": "Int32",
        "createdAt": "Date",
        "updatedAt": "Date",
        "__v": "Int32"
    }
}
```

d. Get student name :

Endpoint : https://caretakerserver.hopto.org/student/getname

HTTP method : POST

Description :To fetch the student name..

Request body :

```
{
  "token": "string"
}
```

Response body :

```
{
  "name": "string"
}
```

e. Update student data :

Endpoint : https://caretakerserver.hopto.org/student/updatedata

HTTP method : POST

Description :To update the student details.

Request body :

```
{
  "token": "string",
  "newdata": {
    "firstName": "string",
    "lastName": "string",
    "roomNum": "Int32"
  }
}
```

Response body :

```
{
    "message": "string"
}
```

f. Admin signup :

Endpoint : https://caretakerserver.hopto.org/admin/signup

HTTP method : POST

Description : It will add a new admin to the database.

Request body :

```
{
  "firstName": "string",
  "lastName": "string",
  "email":"string",
  "password":"string"
}
```

Response body :

```
{
    "message": "string"
}
```

g. Admin sign in :

Endpoint : https://caretakerserver.hopto.org/admin/signin

HTTP method : POST

Description : It will authenticate admin details.

Request body :

```
{
  "email": "string",
  "password": "string"
}
```

Response body :

```
{
  "data": "string",
  "message": "string"
}
```

h. Get admin data :

Endpoint : https://caretakerserver.hopto.org/admin/getdata

HTTP method : POST

Description :To fetch the admin details.

Request body :

```
{
  "token": "string"
}
```

Response body :

```
{
    "data": {
        "_id": "ObjectId",
        "firstName": "string",
        "lastName": "string",
        "email": "string",
        "password": "string",
        "type": "string",
        "roomNum": "Int32",
        "createdAt": "Date",
        "updatedAt": "Date",
        "__v": "Int32"
    }
}
```


i. Get admin name :

Endpoint : https://caretakerserver.hopto.org/admin/getname

HTTP method : POST

Description :To fetch the admin name.

Request body :


```
{
  "token": "string"
}
```


Response body :


```
{
  "name": "string"
}
```


j. Update admin data :

Endpoint : https://caretakerserver.hopto.org/admin/updatedata

HTTP method : POST

Description :To update the admin details.

Request body :

```
{
  "token": "string",
  "newdata": {
    "firstName": "string",
    "lastName": "string"
  }
}
```

Response body :

```
{
    "message": "string"
}
```

k. Book a slot :

Endpoint : https://caretakerserver.hopto.org/booking/make

HTTP method : POST

Description :To book a slot for room cleaning.

Request body :

```
{
  "token": "string",
  "startTime": "string",
  "endTime": "string",
  "date": "string"
}
```

Response body :

```
{
    "message": "string"
}
```

l. Fetch a student's booking :

Endpoint : https://caretakerserver.hopto.org/booking/fetchbytoken

HTTP method : POST

Description :To fetch a student's active booking.

Request body :

```
{
  "token": "string"
}
```


Response body :


```
{
  "bookingData": [
    {
      "_id": "ObjectId",
      "email": "string",
      "roomNum": "Int32",
      "date": "string",
      "startTime": "string",
      "endTime": "string",
      "status": "string",
      "createdAt": "string",
      "updatedAt": "string",
      "__v": "Int32"
    }
  ]
}
```


m. Delete a booking :

Endpoint : https://caretakerserver.hopto.org/booking/delete/:id

HTTP method : DELETE

Description :To delete a booking by booking id .

Parameter : id

Request body :


```
{
  "token": "string"
}
```


Response body :

```
{
  "deletedBooking": {
    "acknowledged": "boolean",
    "deletedCount": "Int32"
  }
}
```


n. Fetch all active bookings :

Endpoint : https://caretakerserver.hopto.org/booking/fetchall

HTTP method : POST

Description : To fetch all active bookings in custom sorted order and filter by date.

Request body :


```
{
  "sortby": "string",
  "date": "string"
}
```


Response body :


```
{
  "bookingsLst": [
    {
      "BookingId": "Objectid",
      "email": "string",
      "roomNum": "Int32",
      "date": "string",
      "startTime": "string",
      "endTime": "string"
    }
  ]
}
```


o. Add a new slot

Endpoint : https://caretakerserver.hopto.org/slot/add

HTTP method : POST

Description : To add a new slot(duration 30 mins) which is then made available to students.

Request body :

```
{
  "date": "string",
  "startTime": "string",
  "endTime": "string",
  "floor": "Int32",
  "teams": "Int32"
}
```

Response body :

```
{
    "message": "string"
}
```

p. Fetch available slots

Endpoint : https://caretakerserver.hopto.org/slot/fetch

HTTP method : POST

Description : To fetch all the available slots for a user based on date and the floor number.

Request body :

```
{
  "token": "string",
  "date": "string"
}
```

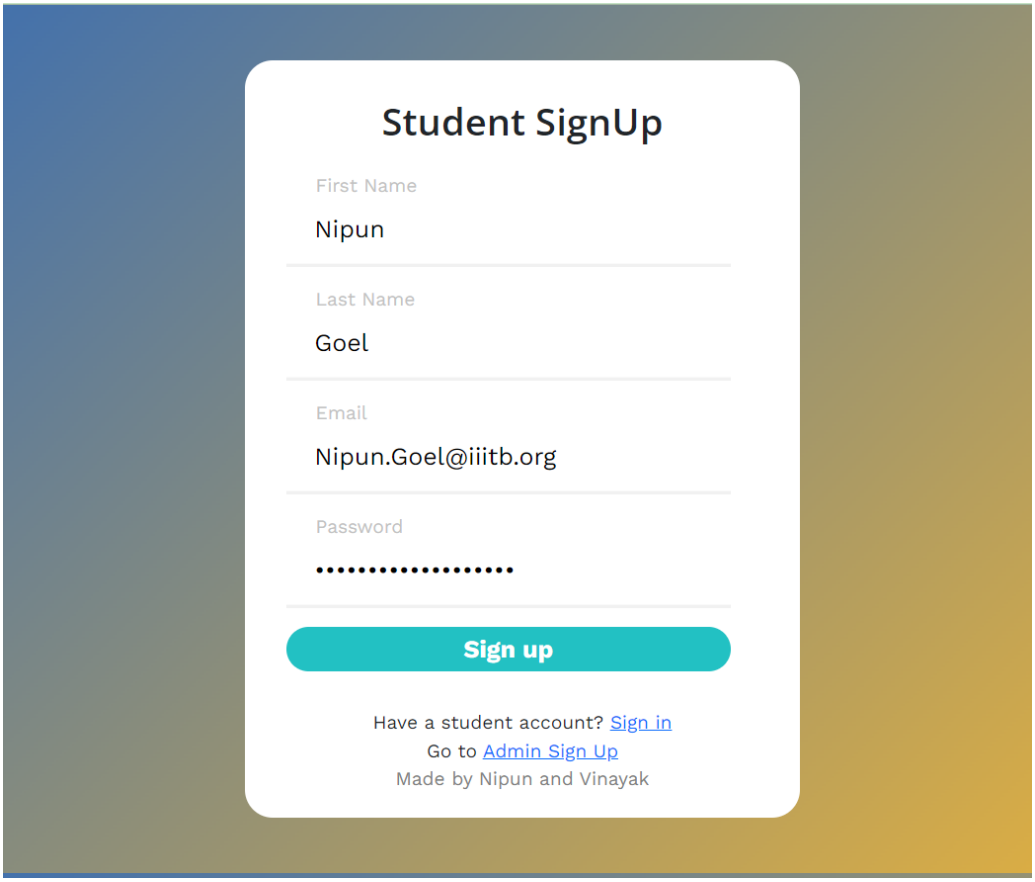
Response body :

```
{
  "availableSlots": [
    {
      "value": "string"
    }
  ]
}
```

# 6. Result and Discussion

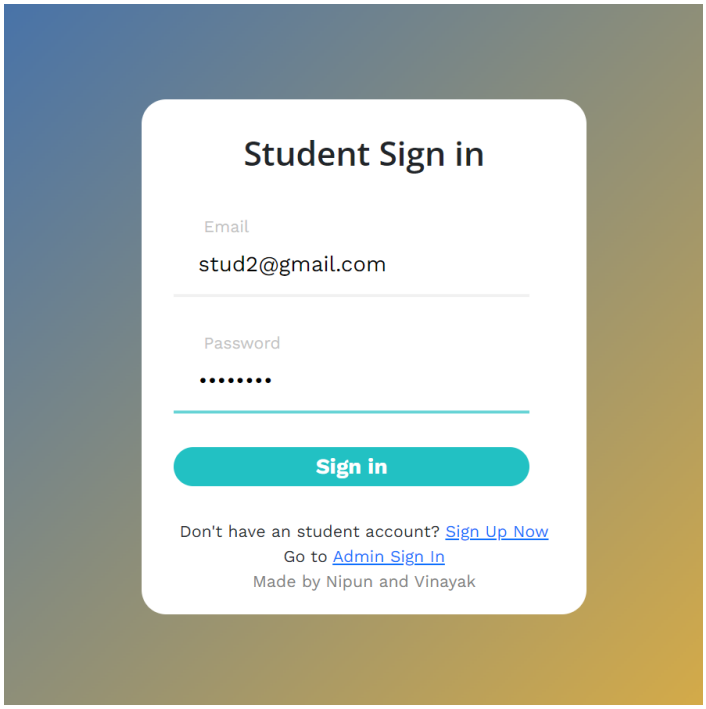
## a. Student Sign Up Page

Student signup requires the student user to provide First Name, Last Name, Email, and Password. In the backend, we have regex limits on email and password to ensure that users enter valid email and password that contain at least one number, one uppercase character and also have length restrictions on password.

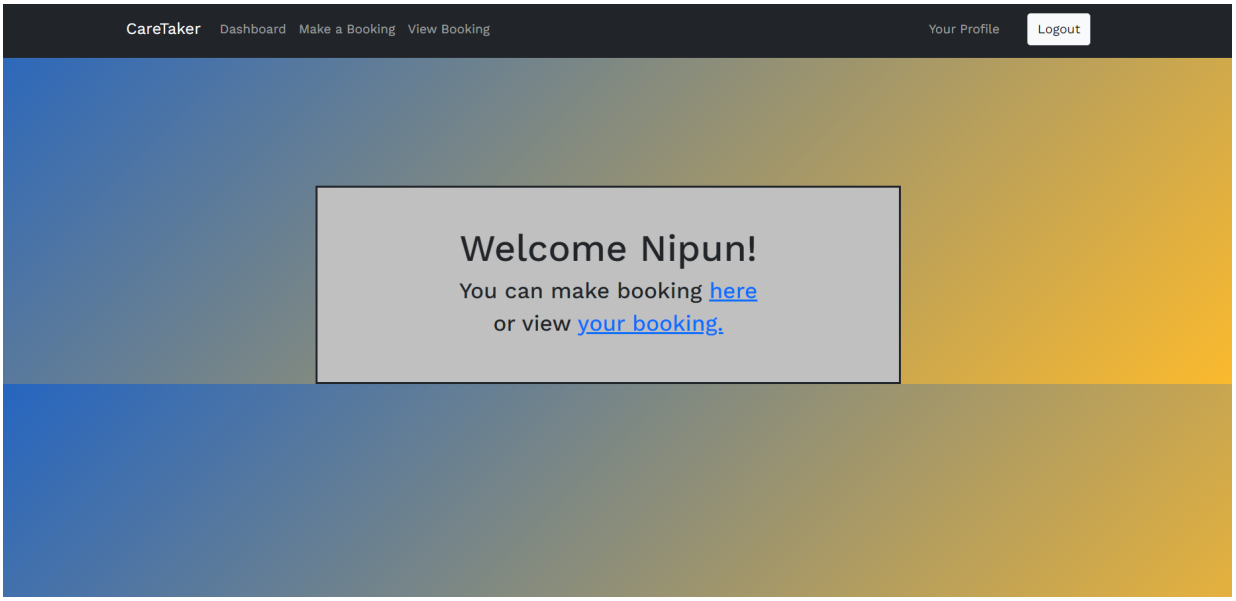


## b. Student Sign In Page

Registered student users can login by providing valid email and password.



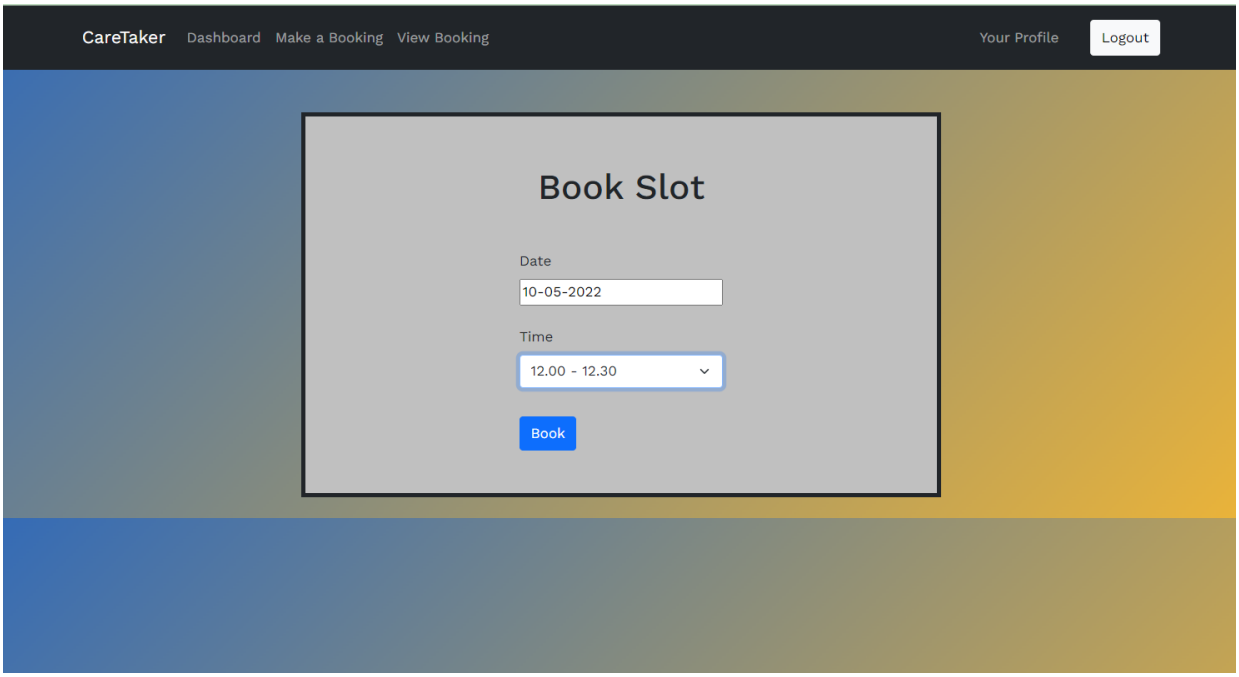

## c. Student Dashboard



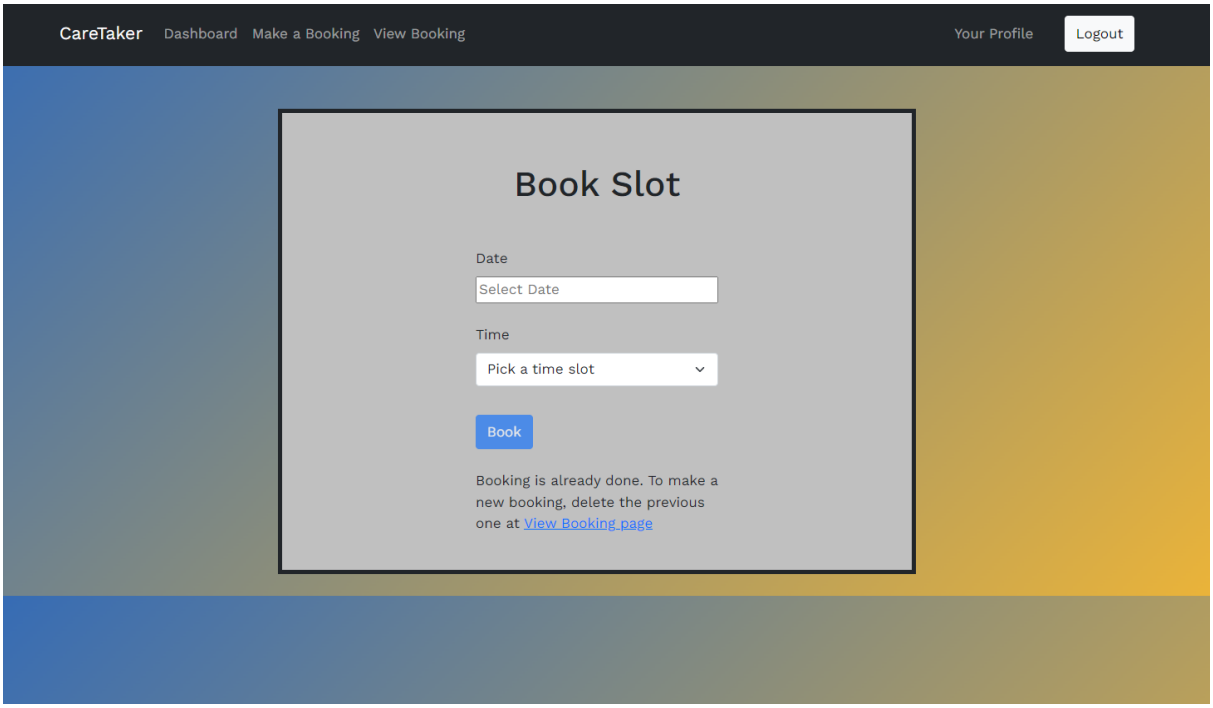
After successful login, the Student dashboard page is displayed. It has a navigation bar which has buttons for navigating to the dashboard, make a booking, view booking, your profile pages and logout button. This navbar is present in other pages as well in student pages.

## d. Student Make Booking Page

Student users can book the slot on the make booking page. Users can choose the date and time slot available for booking on that date and after pressing the book button booking is confirmed and the user is navigated to view the booking page.

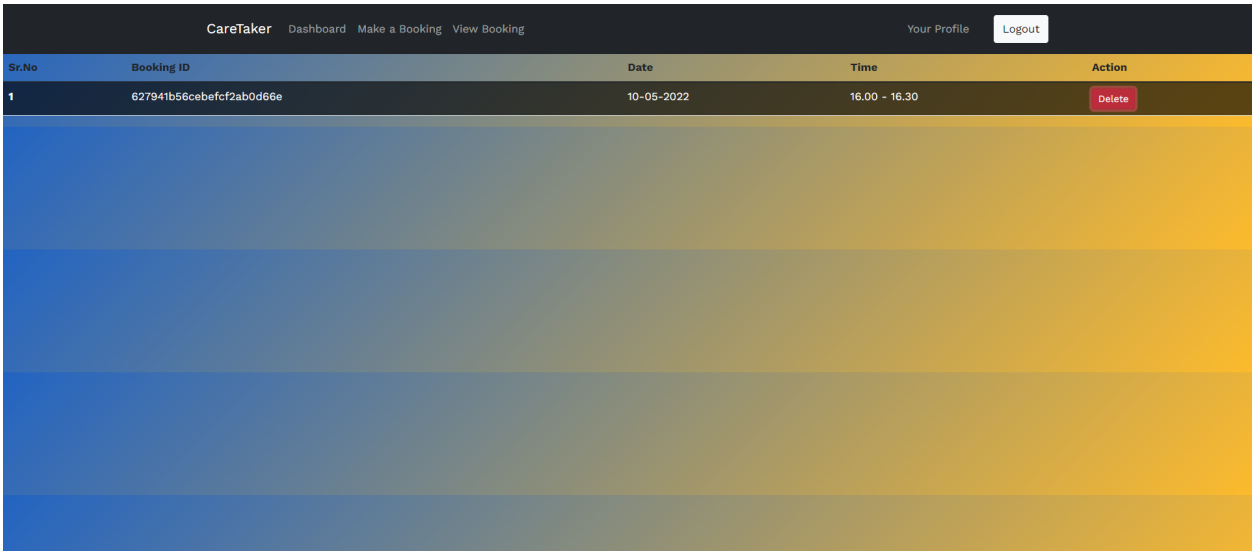



If the user has already an active booking then it will not allow him to book another slot. In that case the book button will be disabled. If the user has to create a new booking then has to delete existing booking.



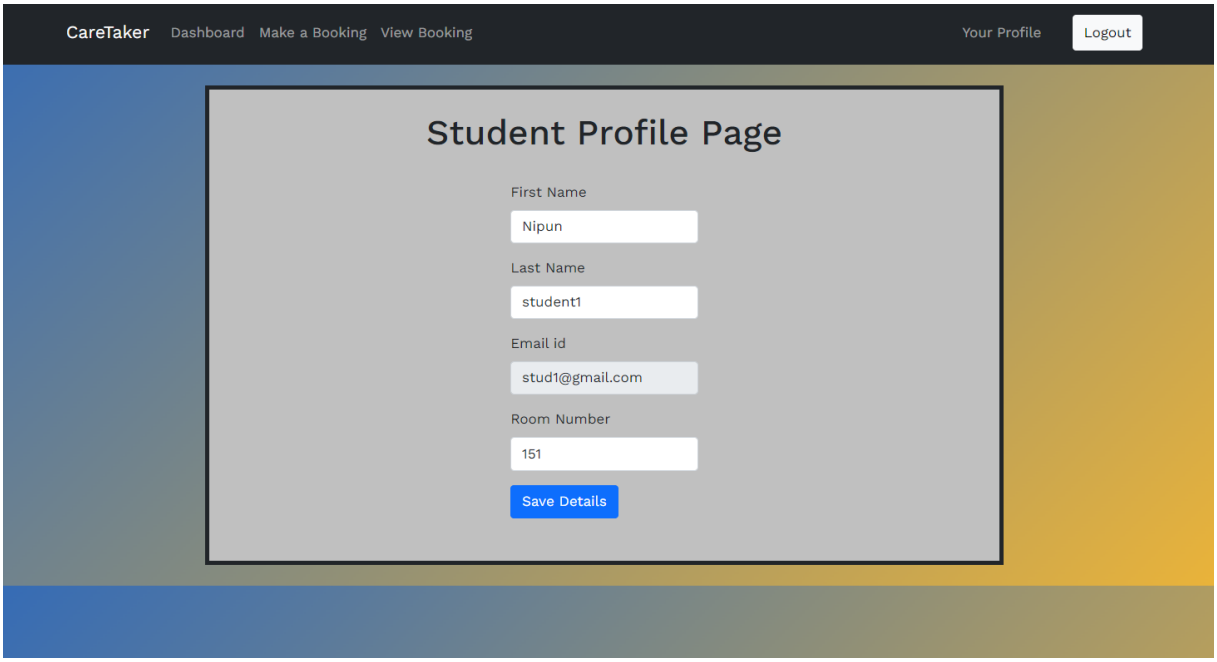

## e. Student View Booking Page

Student users can view the bookings on the view booking page. At a time only utmost one active booking is present for a user. Student users can also delete the booking by clicking the delete button present for that booking.



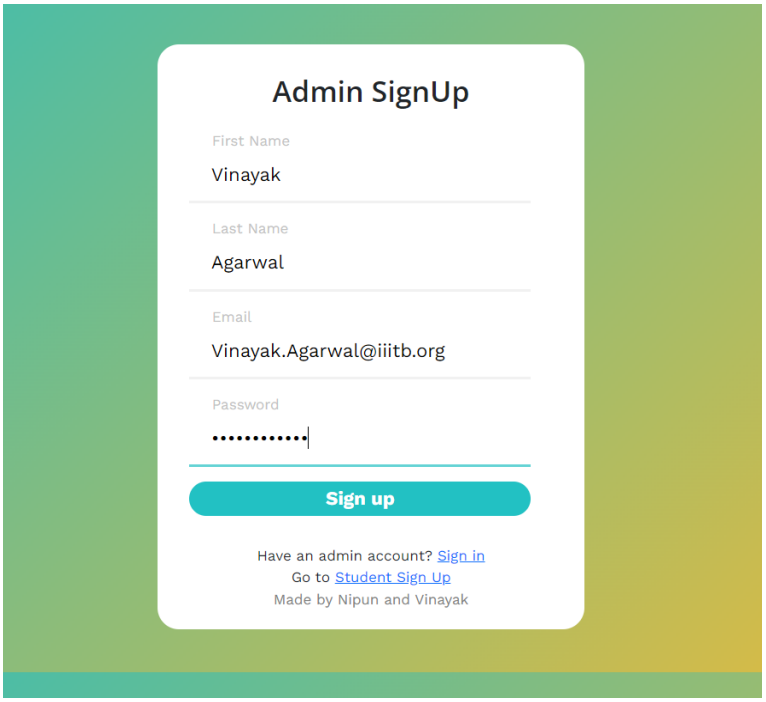
## f. Student Profile Page

Student user can see his own profile page which contains the user details like First Name, Last Name, Email id and Room Number. Users can also edit profiles and then click the save details button.

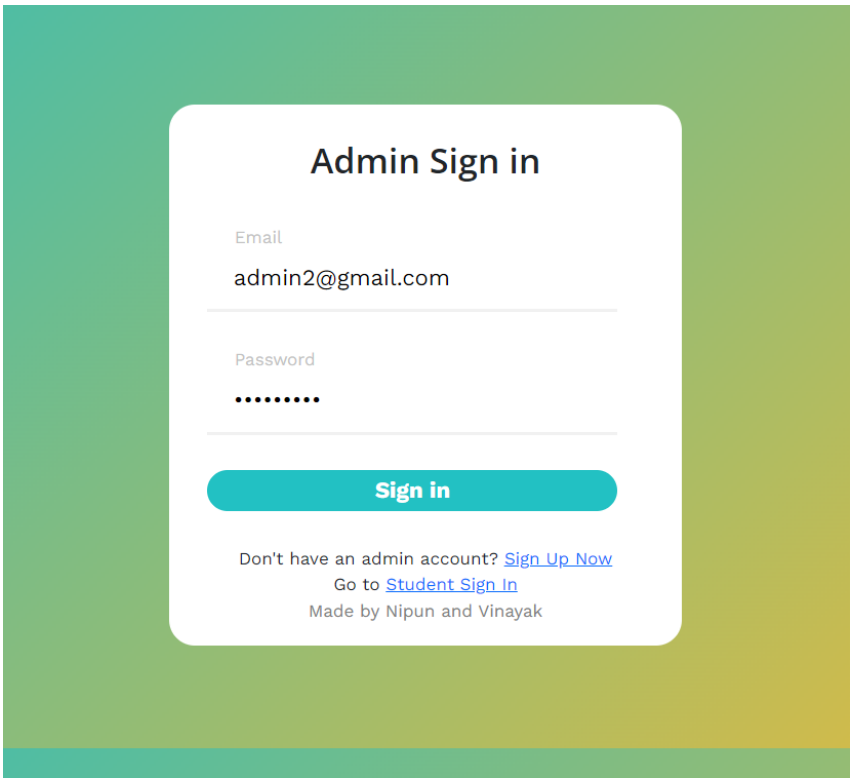



## g. Admin Sign Up Page

Admin signup requires the admin user to provide First Name, Last Name, Email, and Password. In the backend, we have regex limits on email and password to ensure that users enter valid email and password that contain at least one number, one uppercase character and also have length restrictions on password.

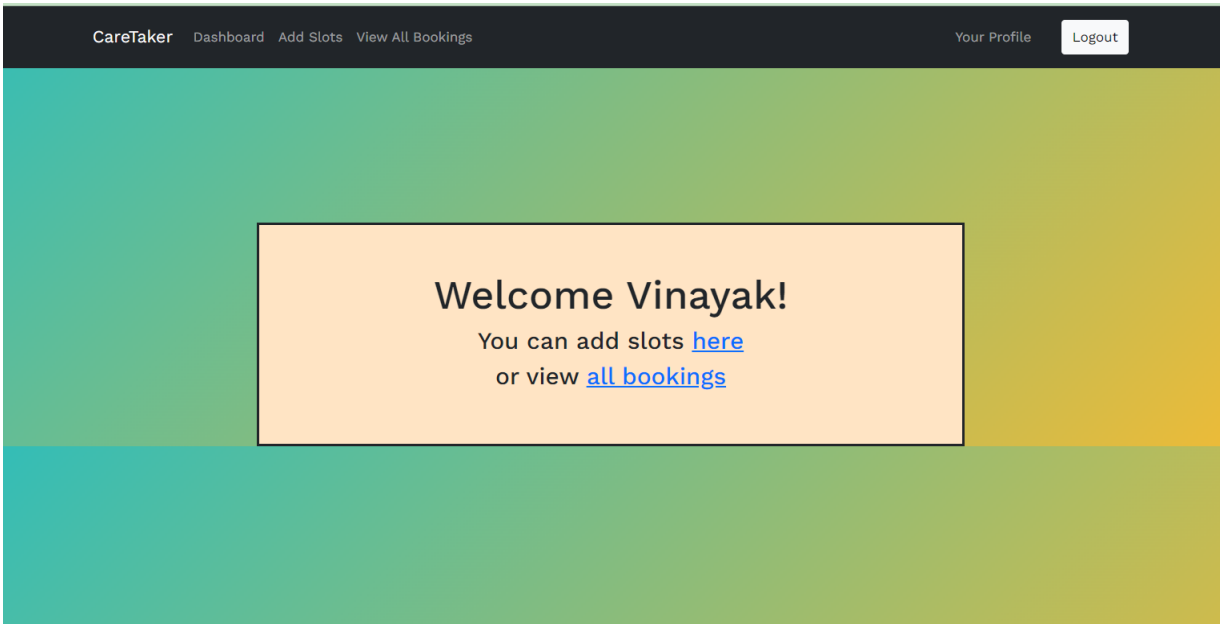



## h. Admin Sign In Page




Registered admin users can login by providing valid email and password.

## i. Admin Dashboard

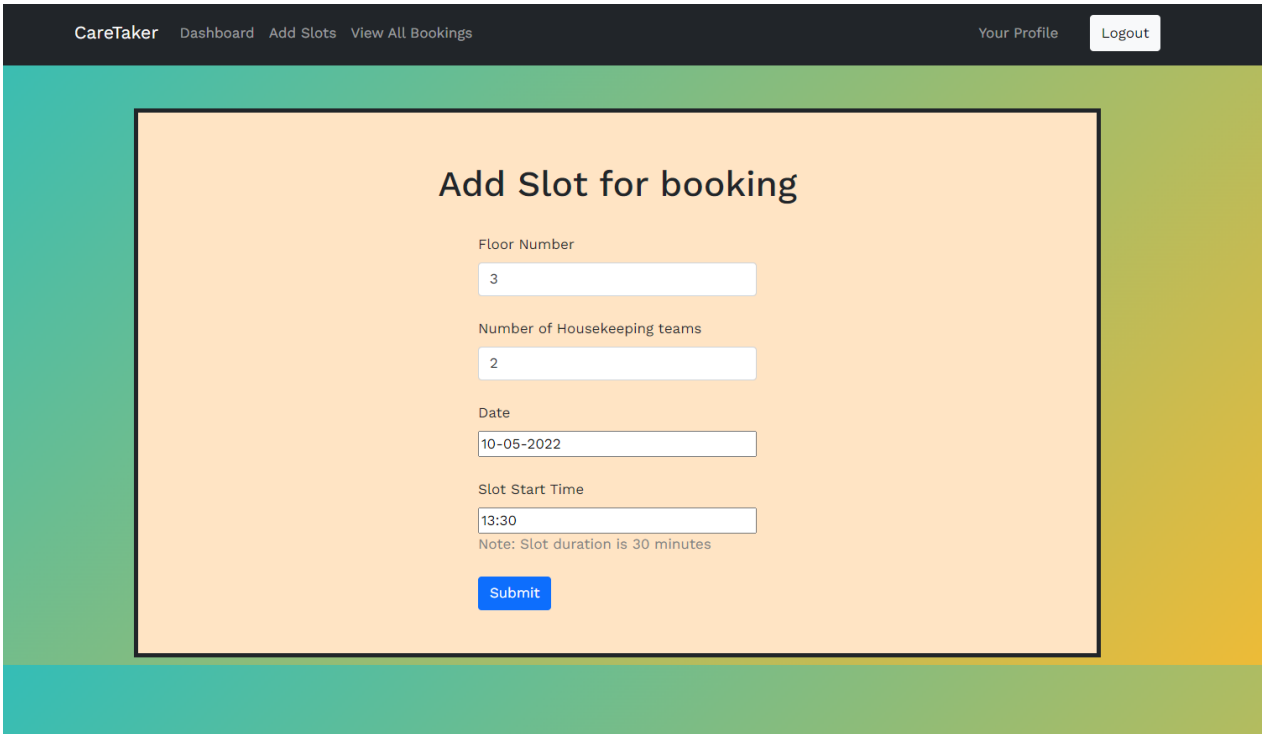



After successful login, the Admin dashboard page is displayed. It has a navigation bar which has buttons for navigating to the dashboard, add slots, view all bookings, your profile pages and logout button. This navbar is present in other pages as well in admin pages.

## j. Admin Add Slots Page

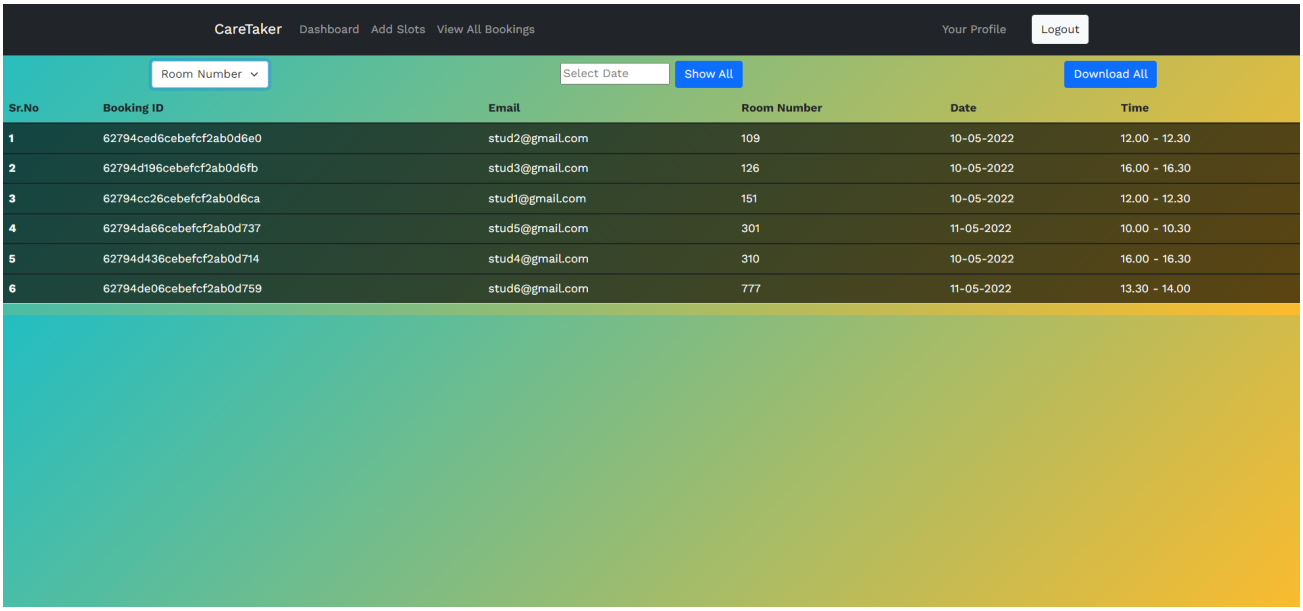
On this page, the admin can add a slot by providing a valid floor number between 1 to 7, number of housekeeping teams, date and slot start time which is then made available to student users.

Note: Slot is of duration of 30mins.



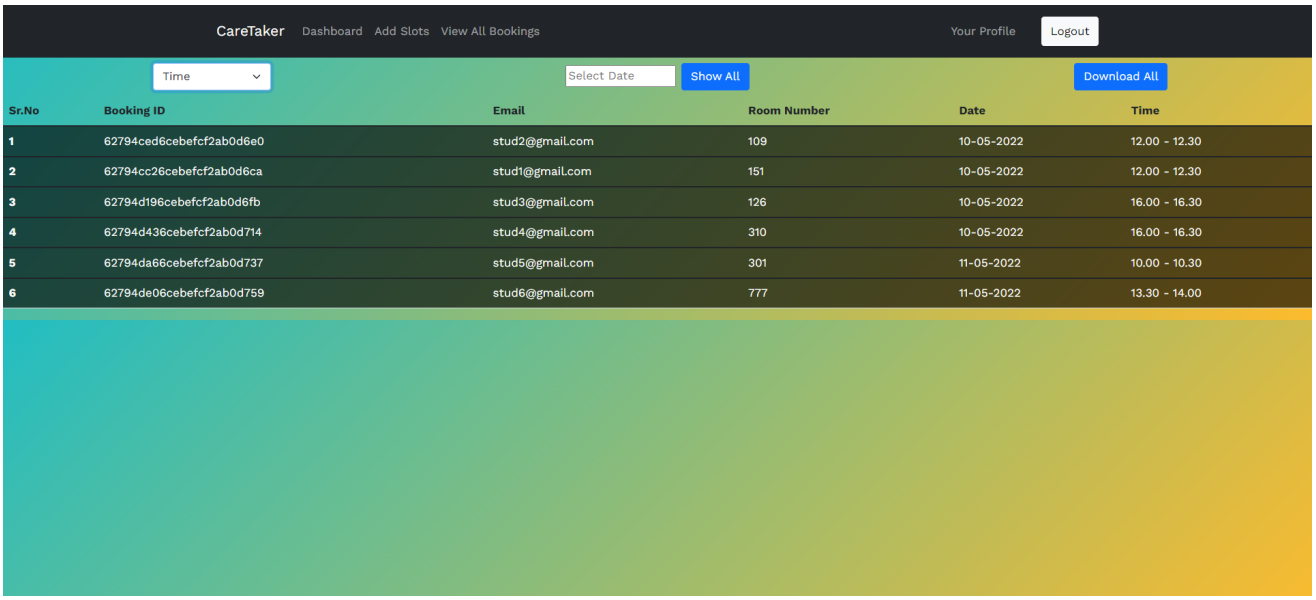

## k. Admin View All Bookings Page

Admin can view and download(in csv format) all the active user bookings.



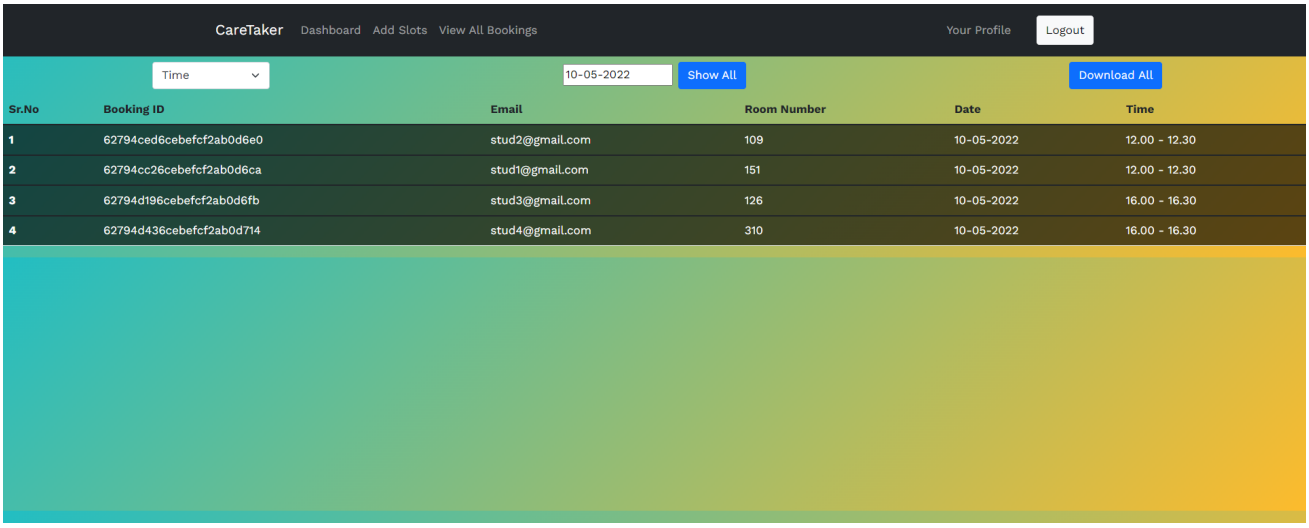

| Sr.No | Booking ID | Email | Room Number | Date | Time |
|---|---|---|---|---|---|
| 1 | 62794ced6cebefcf2ab0d6e0 | stud2@gmail.com | 109 | 10-05-2022 | 12.00 - 12.30 |
| 2 | 62794d196cebefcf2ab0d6fb | stud3@gmail.com | 126 | 10-05-2022 | 16.00 - 16.30 |
| 3 | 62794cc26cebefcf2ab0d6ca | stud1@gmail.com | 151 | 10-05-2022 | 12.00 - 12.30 |
| 4 | 62794da66cebefcf2ab0d737 | stud5@gmail.com | 301 | 11-05-2022 | 10.00 - 10.30 |
| 5 | 62794d436cebefcf2ab0d714 | stud4@gmail.com | 310 | 10-05-2022 | 16.00 - 16.30 |
| 6 | 62794de06cebefcf2ab0d759 | stud6@gmail.com | 777 | 11-05-2022 | 13.30 - 14.00 |

Active bookings can also be viewed in sorted order by room number or time whichever is chosen by the admin.




| Sr.No | Booking ID | Email | Room Number | Date | Time |
|---|---|---|---|---|---|
| 1 | 62794ced6cebefcf2ab0d6e0 | stud2@gmail.com | 109 | 10-05-2022 | 12.00 - 12.30 |
| 2 | 62794cc26cebefcf2ab0d6ca | stud1@gmail.com | 151 | 10-05-2022 | 12.00 - 12.30 |
| 3 | 62794d196cebefcf2ab0d6fb | stud3@gmail.com | 126 | 10-05-2022 | 16.00 - 16.30 |
| 4 | 62794d436cebefcf2ab0d714 | stud4@gmail.com | 310 | 10-05-2022 | 16.00 - 16.30 |
| 5 | 62794da66cebefcf2ab0d737 | stud5@gmail.com | 301 | 11-05-2022 | 10.00 - 10.30 |
| 6 | 62794de06cebefcf2ab0d759 | stud6@gmail.com | 777 | 11-05-2022 | 13.30 - 14.00 |

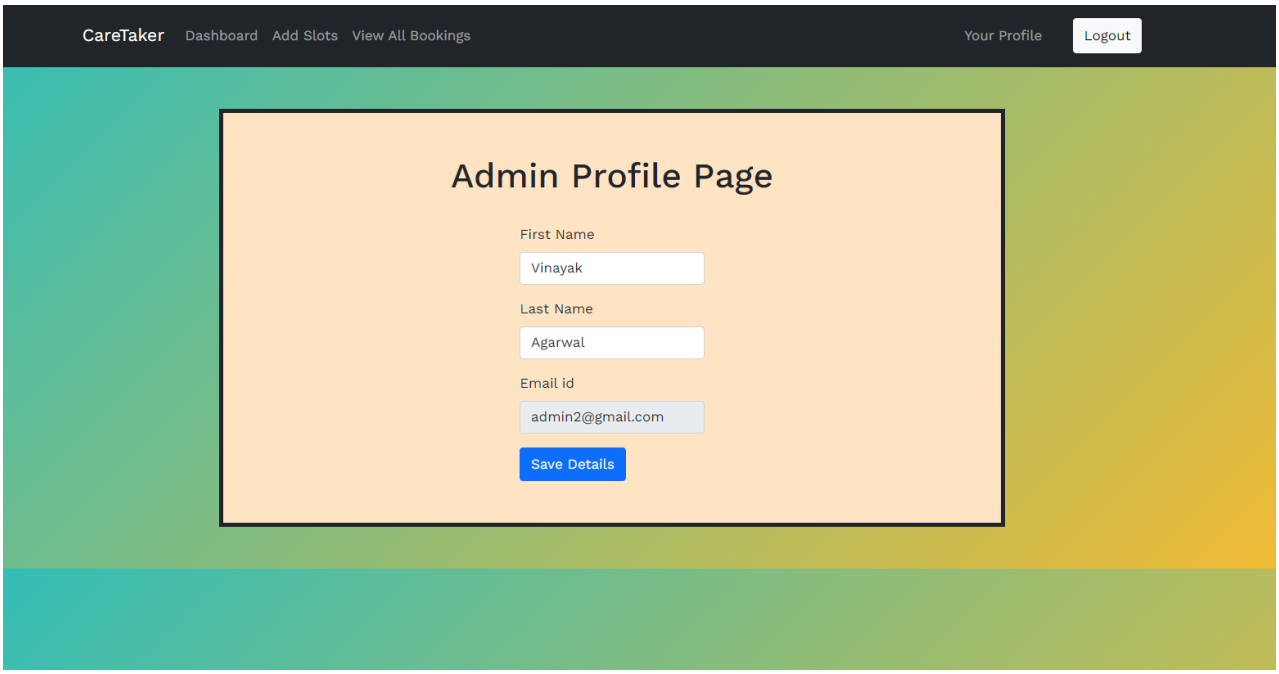
There is also an option to filter the active user bookings by date. This allows the admin to view/download bookings for a particular date.




| Sr.No | Booking ID | Email | Room Number | Date | Time |
|---|---|---|---|---|---|
| 1 | 62794ced6cebefcf2ab0d6e0 | stud2@gmail.com | 109 | 10-05-2022 | 12.00 - 12.30 |
| 2 | 62794cc26cebefcf2ab0d6ca | stud1@gmail.com | 151 | 10-05-2022 | 12.00 - 12.30 |
| 3 | 62794d196cebefcf2ab0d6fb | stud3@gmail.com | 126 | 10-05-2022 | 16.00 - 16.30 |
| 4 | 62794d436cebefcf2ab0d714 | stud4@gmail.com | 310 | 10-05-2022 | 16.00 - 16.30 |

### I. Admin Profile Page

Admin users can see their own profile page which contains the user details like First Name, Last Name, Email id and Room Number. Users can also edit profiles and then click the save details button.




## 7. Scope of Future Work

Following more advanced features can be implemented in the future:

- Login via gmail or iiitb accounts.
- Feedback option to students after the room cleaning slot.
- Notification feature to student users when a slot for their floor is added.

## 8. Challenges Faced

- Initially, we deployed our frontend and backend on "web app for containers" powered by Azure by creating docker images for both. But the continuous deployment option in Azure was not working here.

- We decided to move our frontend code to netlify. But netlify does not support docker, so we deployed the npm production build for frontend directly on netlify using github actions.
- We ran our backend container on Azure VM. To run it on https protocol, we had to set up **SSL certificate** in our VM using **nginx reverse proxy** and **certbot** which was a huge challenge.

## 9. Conclusion

We built a web application for hostel room cleaning management called "**CareTaker**". It allows admin to add slots for room cleaning which can later be booked by students as per their availability leading to efficient use of hostel room cleaning services. This entire application was automated using DevOps tools like Github Actions and Jenkins for continuous integration , Ansible for configuration management and deployment, Docker for containerization, Jest for testing, Winston for logging, ELK stack for monitoring, Netlify and Azure VM Instance for deployment, and Azure for monitoring.

## 10. Project Links

- Web application link https://caretakerclient.netlify.app
- Server DNS link https://caretakerserver.hopto.org/
- Github Main repository link of project https://github.com/nipungoel2000/SPE_project
- Github repository link of client https://github.com/nipungoel2000/caretakerClient
- Github repository link of server https://github.com/nipungoel2000/caretakerServer

## 11. References


- https://reactjs.org/tutorial/tutorial.html
- https://react-bootstrap.github.io/
- https://www.w3schools.com/nodejs/
- https://expressjs.com/en/4x/api.html
- https://www.mongodb.com/docs/manual/

- https://docs.docker.com/
- https://docs.ansible.com/ansible
- https://www.elastic.co/guide/index.html
- Reverse proxy with NGINX and letsencrypt tutorial - The Digital Life (the-digital-life.com)
- https://www.noip.com/