International Institute of Information Technology, Bangalore
(IIIT Bangalore)

Software Production Engineering Project Report

SocialConnect - Social Platform for Hobbyist

Under the Guidance of Prof. B. Thangaraju

Teaching Assistant: Harsh Shah

Jay Parekh
MT2022052

Jayant Mukundam
MT2022054

TABLE OF CONTENTS

1. Abstract

The project is a comprehensive online platform that brings together users with diverse interests and allows them to participate in a wide range of events, as well as create their own. The platform offers a user-friendly interface that enables users to easily search and discover events filtered by their location and hobbies. Users can also create and promote their own events, inviting others to participate and collaborate. The platform provides tools for event organizers to manage their events.By providing a space for individuals to discover, create, and engage with events, the platform aims to facilitate the exchange of knowledge, ideas, and experiences among like-minded individuals, ultimately promoting personal growth and social interaction.

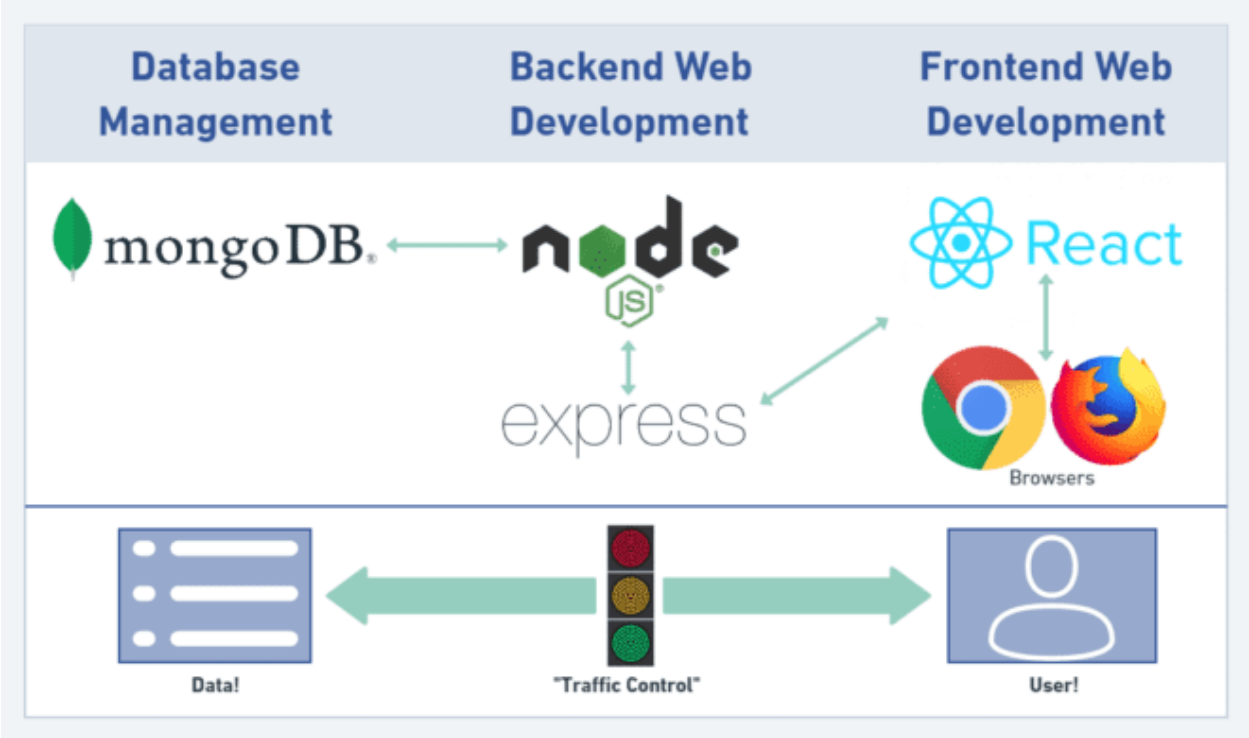
2. Introduction

2.1. Overview

SocialConnect is a platform that enables its users to connect with people who share similar hobbies.

TechStack used :

- ➔ MongoDB
- ➔ ExpressJS
- ➔ ReactJS
- ➔ NodeJS



2.2. Features

A. **Login/Register**
   Users have to create their SocialConnect Account before exploring the web application.

B. **Home page**
   Shows all the events to the user based on their hobbies & location. Users can participate in any event.

C. **Add hobbies**
   A list of hobbies along with their brief description. Users can add/delete their hobbies.

D. **Create Event**
   Users can create an event for a specific hobby. They must provide a registration deadline, event date and minimum participation they are looking for along with other required details.

E. **Your Events**
   A list of events that was created by the user. They can observe the current count of participants & can delete the event

F. **Participated Events**
   A list of events which the user has participated in. A user can also participate in an event that they have created. Also, they can opt out of any of the participating events.

3. System Configuration

3.1. Host System Configuration

Operating System: 20.04.4 LTS
CPU and RAM: 8 core processor with 8 GB RAM
Kernel Version: Linux version 5.13.0-40-generic

3.2. Project Technology Stack

Frontend: React (HTML, JavaScript, Bootstrap)
Backend: Express (NodeJS, JavaScript)
Database: MongoDB (No-SQL)
Cloud: MongoDB Atlas
Build Tool: npm

3.3. DevOps Tools

Source Control Management: Git/Github
Continuous Integration: Jenkins
Containerization: Docker/Docker Hub
Container Orchestration: Kubernetes
Testing: Supertest (Backend)
Continuous Deployment: Ansible
Logger: Winston
Monitoring: Elastic Search, Kibana

4. Software Development Life Cycle

4.1. Installations

React

React is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta and a community of individual developers and companies.

Update local before installing:

```
sudo apt-get update
```

Keep the local packages and softwares updated.

Install NodeJS and NPM:

Inorder to run React, Node environment shall be installed before starting,

```
jay@jay-IdeaPad-L340-15IRH-Gaming:~$ sudo apt-get install nodejs
Reading package lists... Done
```

```
jay@jay-IdeaPad-L340-15IRH-Gaming:~$ node -v
v19.1.0
jay@jay-IdeaPad-L340-15IRH-Gaming:~$ npm -v
8.19.3
```

React App

```
jay@jay-IdeaPad-L340-15IRH-Gaming:~$ npx create-react-app socialconnect
```

```
jay@jay-IdeaPad-L340-15IRH-Gaming:~/Desktop/Social Platform For Hobbyist/client$ npm start
npm start

> client@0.1.0 start
> react-scripts start
```

Express

Express.js, or simply Express, is a back end web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

Express App

Initializing node project with the default config:

```
npm init -y
```

-y automates the config with node default values i.e without going through the interactive process.

Running the Node Application locally:

```
node server.js
```

Running the Node Application locally with nodemon:

Nodemon automatically restarts the node application whenever a file change in the directory is detected.

4.2 Testing

Supertest provides a high-level abstraction for testing HTTP, while still allowing you to drop down to the lower-level API provided by superagent.
Here is a snippet of how to write tests with Supertest, in this case, we are testing the /users/ API. Similarly we have tested the /hobbies and /events APIs.

```
describe('Login api', () => {
  describe('Give a username and password', () => {
    test('Correct username and password responds with status code 200', async () => {
      const user = {
        username: 'jayantmukundam',
        password: 'jayant',
      }
      await api.post('/api/auth/login').send(user).expect(200)
    })
  }, 30000)
  describe('Give a username and password', () => {
    test('Incorrect username or password responds with status code 400', async () => {
      const user = {
        username: 'jayantmukundam2811',
        password: 'mukundam',
      }
      await api.post('/api/auth/login').send(user).expect(400)
    })
  }, 30000)
```

Note that these files must have the extension .test.js in order for Jest to pick up these files as testing files. After this you can run the tests by running the command ***npm run test***

```
PASS src/tests/user_api.test.js
  Login api
    Give a username and password
      ✓ Correct username and password responds with status code 200 (164 ms)
      ✓ Incorrect username or password responds with status code 400 (51 ms)
      ✓ Username doesn't exist responds with status code 400 (65 ms)
  Create user API
    Provide user details
      ✓ Correct user details responds with status code 200 (183 ms)
      ✓ Duplicate username responds with status code 400 (55 ms)
```

For backend testing we have created a testing database on MongoDB Atlas which has its own URI separate from the production database. This ensures that testing doesn't interfere with the production database at any point in time and we can do operations like deleting entire entries from collections in the testing database in order to maintain consistency between tests.

4.3 Source Control Management(SCM)

Source Control Management is used for tracking the file change history, source code, etc. It helps us in many ways in keeping the running project in a structured and organized way.
Repository Link: https://github.com/jayparekh691/SPE_major_project
The frontend is created in the client/ directory and the backend is created in the server/ directory.

Initializing the project:

git init
git remote add origin https://github.com/jayparekh691/SPE_major_project.git

Workflow:

git add <files>
git commit -m "commit message"
git pull origin master
git push origin master

The code is first pulled before making a push to make sure that our project is in the latest stage and to avoid merge conflicts. For the above, the pull and push is done on the “master” branch.

Working on a feature/issue:

git checkout master
git checkout -b "<your_branch_name>"

After creating a branch, required changes are done and a pull request to the main is created.

git add <files>
git commit -m "commit message"
git pull origin master
git push origin master

Then merge the pull request from Github.

4.4. Containerization with Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker builds images automatically by reading the instructions from a Dockerfile -- a text file that contains all commands, in order, needed to build a given image. A Dockerfile adheres to a specific format and set of instructions which you can find at Dockerfile reference.

Frontend Dockerfile

client > Dockerfile

```
# Dockerfile for React client

# Use node's alpine variant to save on space and version number is 17 (latest)
FROM node:17-alpine

# Create the folder "app" under the /usr/src path and set it to be the working directory
# for the further COPY, RUN and CMD instructions
WORKDIR /usr/src/app

# Copy the package.json and package-lock.json files to the "app" folder
COPY package*.json ./

# Install the dependencies mentioned in package.json
RUN npm install

# Copy the local files to the "app" folder
COPY . .

# Expose port 3000 on the host machine to the container for listening to external connections
EXPOSE 3000

# Start the React applications
CMD ["npm", "start"]
```

Backend Dockerfile

server > Dockerfile

```
# Dockerfile for Express Backend

# Use node's alpine variant to save on space and version number is 17 (latest)
FROM node:17-alpine

# Create the folder "app" under the /usr/src path and set it to be the working directory
# for the further COPY, RUN and CMD instructions
WORKDIR /usr/src/app

# Copy the package.json and package-lock.json files to the "app" folder
COPY ./package*.json ./

# Install the dependencies mentioned in package.json
RUN npm install

# Copy the local files to the "app" folder
COPY . .

# Expose port 3001 on the host machine to the container for listening to external connections
EXPOSE 3001

CMD ["npm", "start"]
```

Running docker build using this Dockerfile as the source creates the required Docker image that is ready to run our application.

We need to build the Docker image and push it to Docker Hub which requires logging in to the DockerHub account as well.
This will be covered in the Continuous Integration section.

4.5 Kubernetes :

Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management. Originally designed by Google, the project is now maintained by the Cloud Native Computing Foundation.

The name Kubernetes originates from Greek, meaning 'helmsman' or 'pilot'. Kubernetes is often abbreviated as K8s, counting the eight letters between the K and the s (a numeronym).

Kubernetes works with containerd. Its suitability for running and managing large cloud-native workloads has led to widespread adoption of it in the data center. There are multiple distributions of this platform – from ISVs as well as hosted-on cloud offerings from all the major public cloud vendors.

On the host machine, follow the instruction of this link
https://kubernetes.io/docs/tasks/tools/

After installation, you should be able to run the following and start minikube and see kubernetes version information.

Kubernetes Backend yaml file:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: socialconnect-backend-deployment
  namespace: socialconnect
spec:
  selector:
    matchLabels:
      app: socialconnect-backend
  replicas: 1
  template:
    metadata:
      labels:
        app: socialconnect-backend
    spec:
      containers:
        - name: socialconnect-backend
          image: jayparekhiiitb/spe_major_project-server
          resources:
            limits:
              memory: '128Mi'
              cpu: '500m'
          ports:
            - name: http
              containerPort: 3001
          envFrom:
            - secretRef:
                name: socialconnect-backend-secret
          volumeMounts:
            - name: karaf-conf-storage
              mountPath: '/usr/src/app/src/logs'
        # Use hostPath here
      volumes:
        - name: karaf-conf-storage
          hostPath:
            path: '/tmp/data'
---
apiVersion: v1
kind: Service
metadata:
  name: socialconnect-backend-service
  namespace: socialconnect
spec:
  selector:
    app: socialconnect-backend
  ports:
    - name: http
      port: 3001
      targetPort: 3001
```

Kubernetes Frontend yaml file:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: socialconnect-frontend-deployment
  namespace: socialconnect
spec:
  selector:
    matchLabels:
      app: socialconnect-frontend
  template:
    metadata:
      labels:
        app: socialconnect-frontend
    spec:
      containers:
        - name: socialconnect-frontend
          image: jayparekhiiitb/spe_major_project-client
          # resources:
          #   limits:
          #     memory: "128Mi"
          #     cpu: "500m"
          ports:
            - containerPort: 3000
          # envFrom:
          #   - configMapRef:
          #       name: socialconnect-frontend-configmap
---
apiVersion: v1
kind: Service
metadata:
  name: socialconnect-frontend-service
  namespace: socialconnect
spec:
  type: LoadBalancer
  selector:
    app: socialconnect-frontend
  ports:
    - name: http
      port: 3000
      targetPort: 3000
      nodePort: 30000
```

Kubernetes Frontend ConfigMap file :

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: socialconnect-frontend-configmap
  namespace: socialconnect
data:
  REACT_APP_FRONTEND_URL: socialconnect-frontend-service
  REACT_APP_BACKEND_URL: socialconnect-backend-service
```

Kubernetes Secret File:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: socialconnect-backend-secret
  namespace: socialconnect
type: Opaque
data:
  PORT: MzAwMQ==
  MONGODB_URI: bW9uZ29kYitzcnY6Ly9qYXlwYXJla2g6cm9jazEyM0FAcHJvZHVjdGlvbi4wNzRieHJhLm1vbmdvZGIubmV0Lz9yZXRyeVdyaXRlcz10cnVlJnc9bWFqb3JpdHk=
```

Kubernetes Ingress File:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: socialconnect-ingress
  namespace: socialconnect
  labels:
    name: socialconnect-ingress
spec:
  rules:
    - host: socialconnect.com
      http:
        paths:
          - pathType: Prefix
            path: '/'
            backend:
              service:
                name: socialconnect-frontend-service
                port:
                  number: 3000
          - pathType: Prefix
            path: '/api'
            backend:
              service:
                name: socialconnect-backend-service
                port:
                  number: 3001
```

4.6. Ansible

Ansible is an open-source automation tool, or platform, used for IT tasks such as configuration management, application deployment, intraservice orchestration and provisioning.

Ansible is mainly used to perform a lot of tasks that otherwise are time-consuming, complex, repetitive, and can make a lot of errors or issues.

Note: We are going to be pulling the Docker Hub image to the host system for Ansible deployment.

Creating Inventory file

The inventory file is used to specify the list of managed hosts/server machines. The inventory file looks as,

```
[demoArc]
client1 ansible_host=172.16.138.223 ansible_connection=ssh ansible_user=jay ansible_sudo_pass=********
```

Configuring OpenSSH Server

Install openssh-server on the host machine and because it is the Jenkins user that is going to be doing the pulling of Docker image, we need to SSH from the Jenkins user to the user that is specified in the inventory file.

```
apt-get install openssh-server service ssh restart

su jenkins
ssh-keygen -t rsa
ssh-copy-id <host-username>@<host-ip-address> sudo chmod 666
```

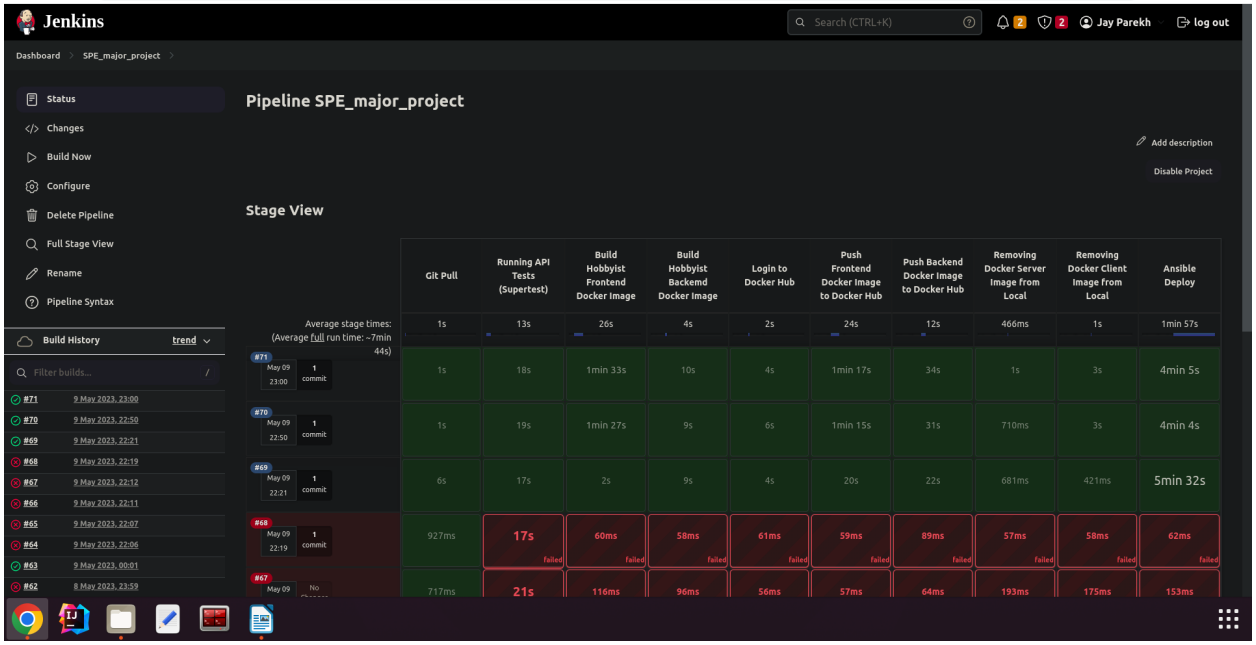
The chmod command is required so that the Jenkins user has access to the Docker socket for performing the docker build and push operations.

Because we use MongoDB Atlas for storing our database, we require the MongoDB URI in order to access the database and this URI should be kept secret. However we have to send this URI in a .env type of file to the backend container, how to do this and not leak the URI in any way? This is where the kubernetes-secret.yaml file is used.

4.7. Continuous Integration: Jenkins

Jenkins is an open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat.

Create a Jenkins Pipeline project,



Script

```
pipeline {
    // Declare variables that will be used by the later stages
    environment {
        DOCKERHUB_REGISTRY = "jayparekhiiitb/spe_major_project"
        DOCKERHUB_CREDENTIALS = credentials('dockerhub-id')
    }
    agent any
    stages {

        stage('Git Pull') {
            steps {
                // credentials are required because its a private repository
                git url: 'https://github.com/jayparekh691/SPE_major_project.git'
            }
        }
        stage ('Running API Tests (Supertest)') {
                steps {
                    sh '''
                        cd server
                        npm ci
                        npm run test
                    '''
                }
            }
        stage('Build Hobbyist Frontend Docker Image') {
            steps {
                sh "docker build -t $DOCKERHUB_REGISTRY-client:latest client/"
              }
        }

        stage('Build Hobbyist Backemd Docker Image') {
            steps {
                sh "docker build -t $DOCKERHUB_REGISTRY-server:latest server/"
              }
        }
```

Running Docker containers after build:

```
jay@jay-IdeaPad-L340-15IRH-Gaming: ~
jay@jay-IdeaPad-L340-15IRH-Gaming: ~ 200x24
jay@jay-IdeaPad-L340-15IRH-Gaming:~$ docker ps
CONTAINER ID   IMAGE                                                    COMMAND                  CREATED         STATUS         PORTS
                                                                NAMES
3011ec45adb0   gcr.io/k8s-minikube/kicbase-builds:v0.0.38-1680381266-16207   "/usr/local/bin/entr…"   3 minutes ago   Up 3 minutes   127.0.0.1:49
->5000/tcp, 127.0.0.1:49154->8443/tcp, 127.0.0.1:49153->32443/tcp   minikube
jay@jay-IdeaPad-L340-15IRH-Gaming:~$ kubectl get pods -n socialconnect
NAME                                           READY   STATUS    RESTARTS   AGE
socialconnect-backend-deployment-bf574649-dsjrt      1/1     Running   0          4m41s
socialconnect-frontend-deployment-7cb6b6577d-j7shf   1/1     Running   0          4m41s
jay@jay-IdeaPad-L340-15IRH-Gaming:~$
```

Monitoring - ELK Stack

"ELK" is the acronym for three open source projects: Elasticsearch, Logstash, and Kibana. ELK stack gives us the ability to aggregate logs from all the systems and applications, analyze these logs, and create visualizations for application and infrastructure monitoring, faster troubleshooting, security analytics, and more.

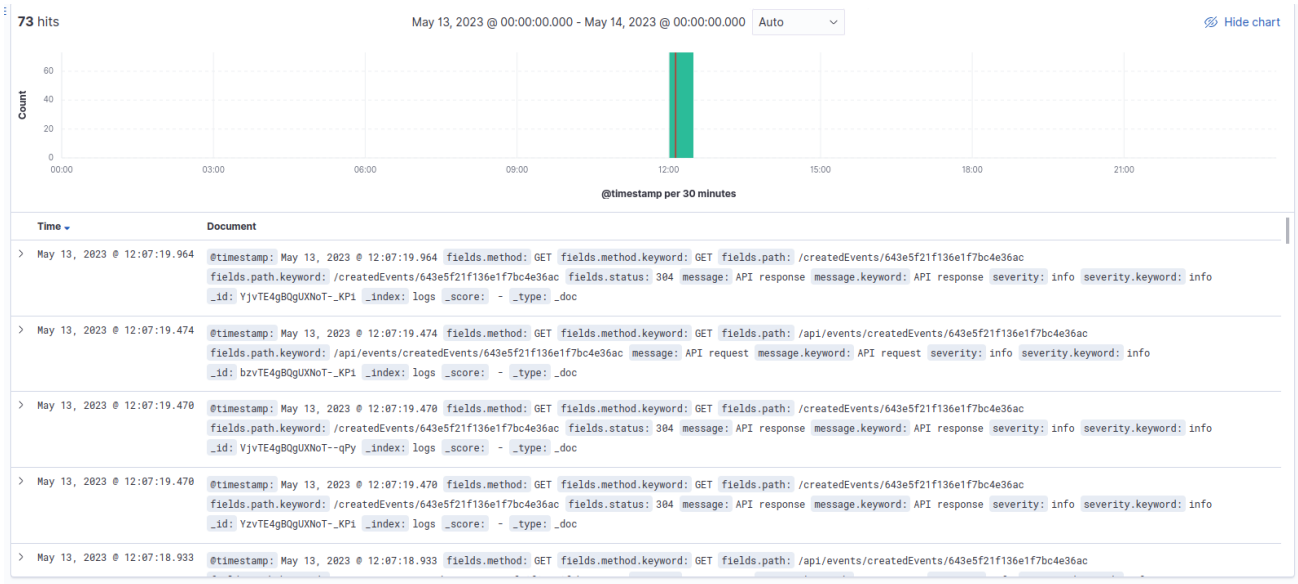
To use the ELK stack we first require the logs that are generated from the application. Because we have used Docker volumes for persisting the logs as specified in the Docker Compose file, we can get the logs at the path /home/server/logs/access.log.

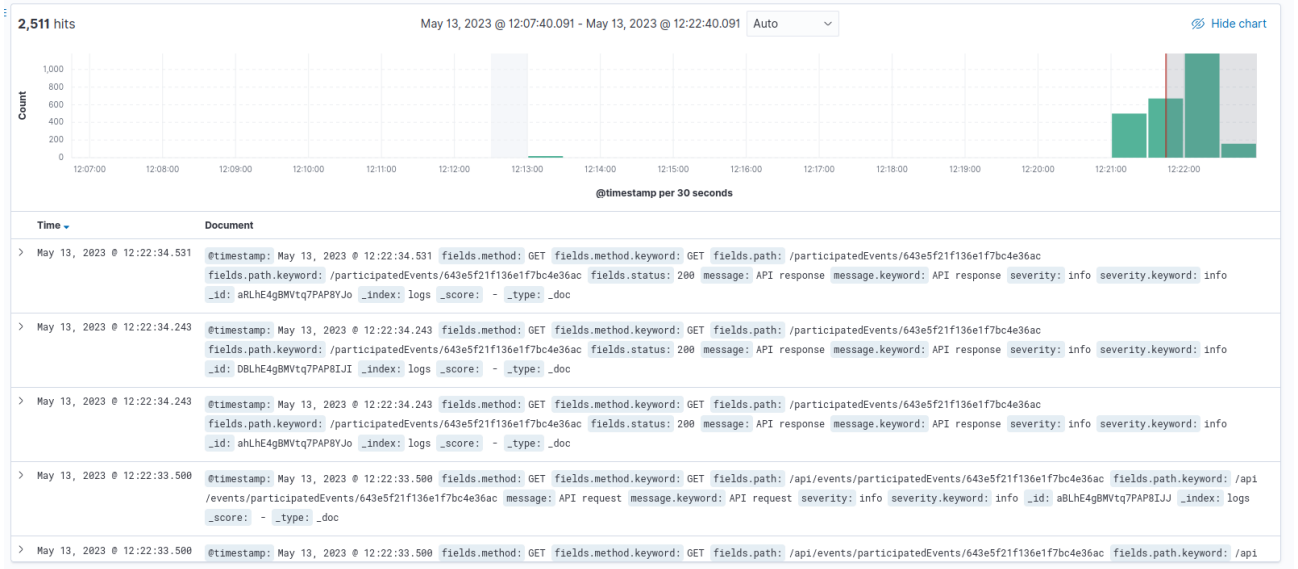
Now we can upload this log file into our Elastic Cloud cluster.

We have automated our ELK log analysis. Follow the instructions in this article to automate ELK : https://drigger.medium.com/automate-mern-using-elk-8059038459c2

To generate Logs we use Winston :

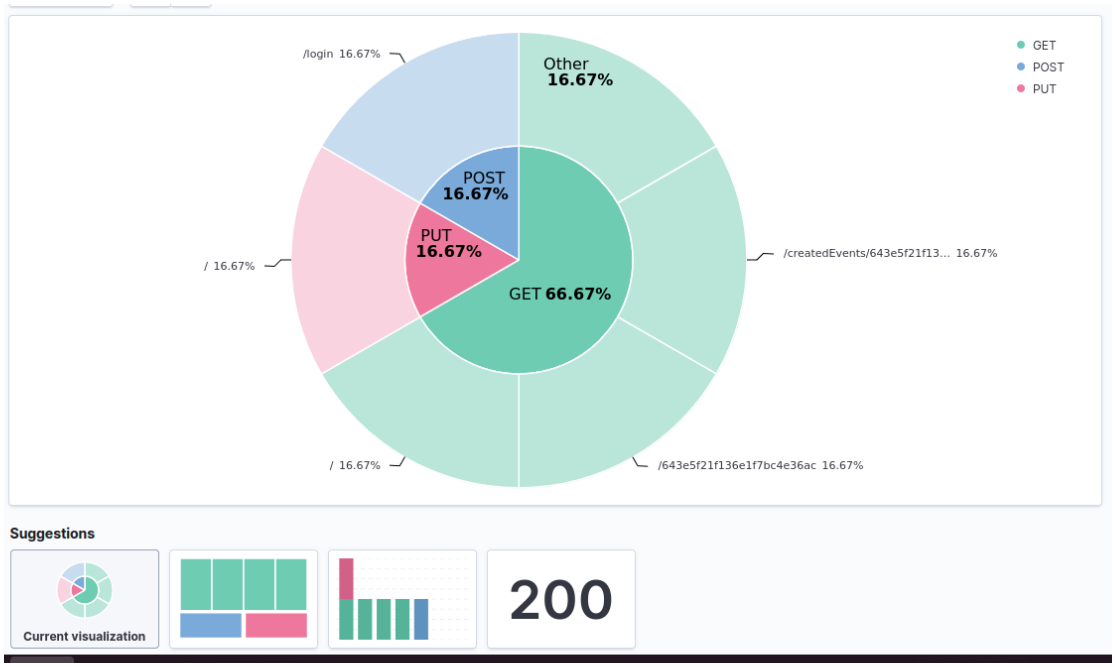
1. Time Series View

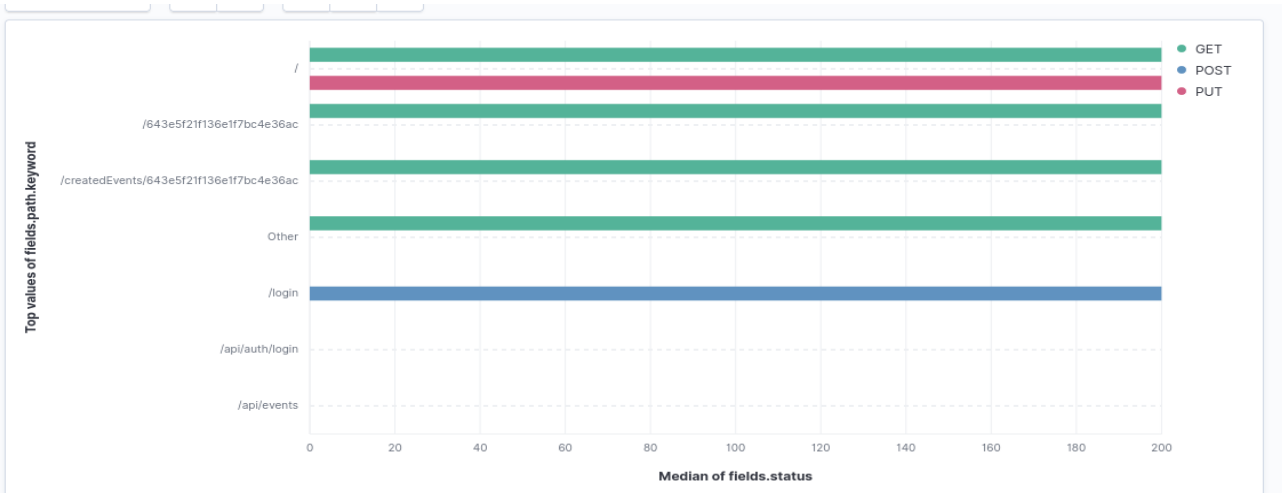


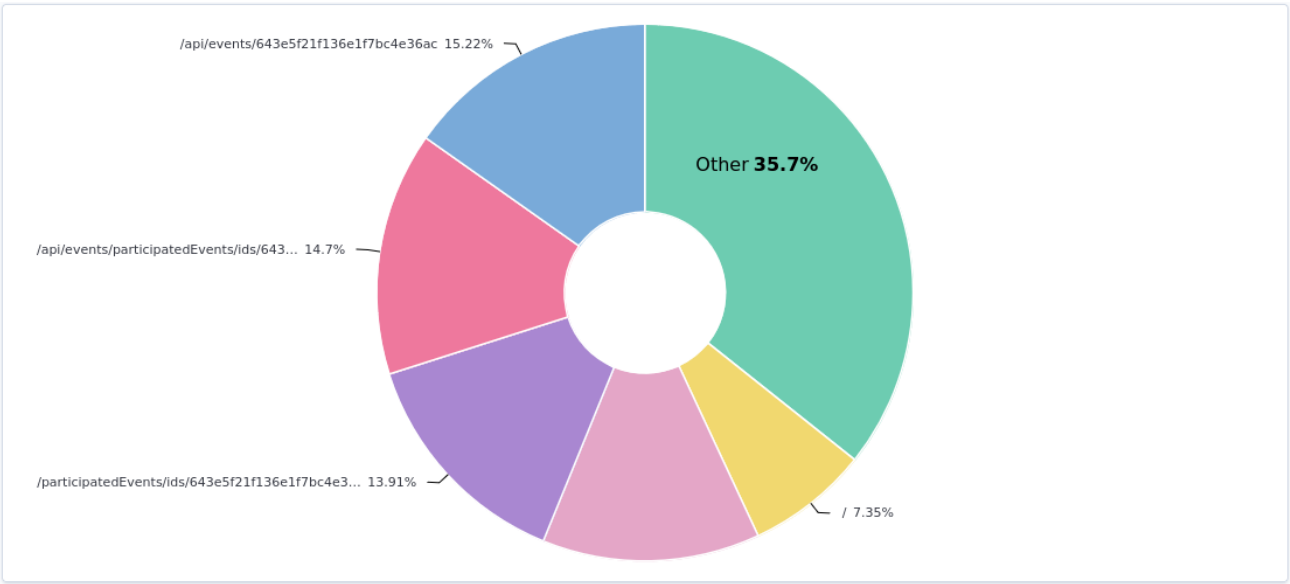


2. Visualizing http request_field

3. Visualizing http requests for each API




GET
POST
PUT
/
/643e5f21f136e1f7bc4e36ac
/createdEvents/643e5f21f136e1f7bc4e36ac
Other
/login
/api/auth/login
/api/events
Top values of fields.path.keyword
0
20
40
60
80
100
120
140
160
180
200
Median of fields.status


/api/events/643e5f21f136e1f7bc4e36ac 15.22%
Other 35.7%
/api/events/participatedEvents/ids/643... 14.7%
/participatedEvents/ids/643e5f21f136e1f7bc4e3... 13.91%
/ 7.35%

5. Experimental Setup

5.1 Functional Requirements

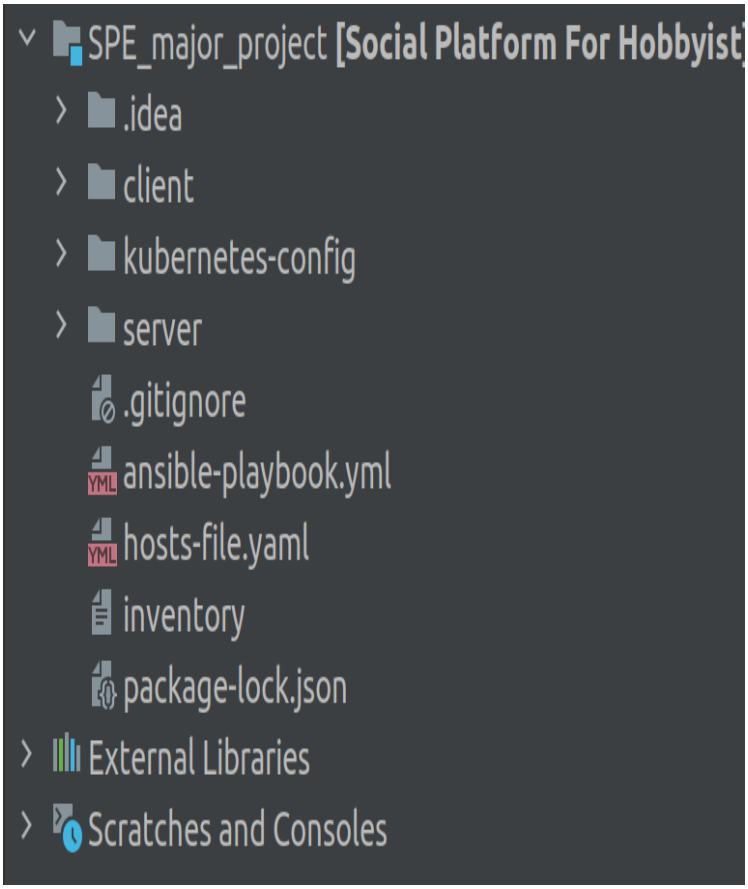
- ➔ Users can register and create their account.
- ➔ Users can add or remove hobbies.
- ➔ Users can participate in an event held in their location & have the same hobby as theirs.
- ➔ Users can view the events they have participated in.
- ➔ Users can create an event specifying the hobby, registration deadline, event date and minimum number of participation they are looking for.
- ➔ Users can delete the events created by them. Subsequently, the event will be removed from all the other users' UI who had participated in that event.

5.2 Non functional Requirements

- ➔ Portability: To ensure portability, Docker images have been built for the frontend and backend.
- ➔ Scalability: The database is created on MongoDB Atlas and in case of high traffic, Atlas will automatically handle this via scaling and thus scalability is achieved.
- ➔ User Friendly: The website has to be user friendly and error messages should pop up when relevant.
- ➔ Performance: The performance of the website should not degrade in case of multiple users.

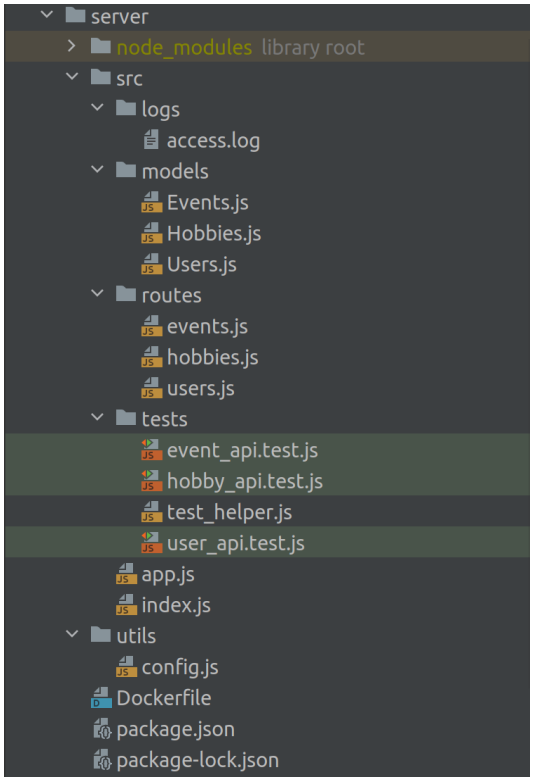
5.3 Code Walkthrough

The folder structure that we have followed includes the client-side *client/* and the server-side *server/* with subfolders within them to ensure proper flow of data in the Website.



Directory structure

5.3.1 Backend



Directory Structure for Backend

We have incorporated MVC architecture (Model View Controller) in the project.

Model:

- ➔ It is known as the lowest level which means it is responsible for maintaining data.
- ➔ The Model is actually connected to the database so anything you do with data - adding or retrieving data is done in the Model component.
- ➔ It responds to the controller requests because the controller never talks to the database by itself. The Model talks to the database back and forth and then it gives the needed data to the controller.

Here are the Models that we have used in our project.

1. User Model

```
const UserSchema = new mongoose.Schema({
  name: { type: String, required: true },
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  address: { type: String, required: true },
  district: { type: String, uppercase: true, required: true },
  state: { type: String, uppercase: true, required: true },
  gender: { type: String, required: true },
  mobilenumber: { type: String, required: true },
  participatedEvents: [{ type: mongoose.Schema.Types.ObjectId, ref: 'events' }],
  hobbies: [
    { type: mongoose.Schema.Types.ObjectId, ref: 'hobbies', default: null },
  ],
})
```

2. Event Model

```
const EventSchema = new mongoose.Schema({
  eventname: { type: String, required: true, unique: true },
  hobbyname: { type: String, required: true },
  registrationDate: { type: String, required: true },
  location: { type: String, required: true },
  district: { type: String, uppercase: true, required: true },
  state: { type: String, uppercase: true, required: true },
  description: { type: String, required: true },
  eventDate: { type: String, required: true },
  minParticipation: { type: Number, required: true },
  participants: [{ type: mongoose.Schema.Types.ObjectId, ref: 'users' }],
  userOwner: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'users',
    required: true,
  },
})
```

3. Hobby Model

```
const HobbySchema = new mongoose.Schema({
    hobbyName: { type: String, required: true, unique: true },
    hobbyDescription: {type: String, required:true},
    userList: [{ type: mongoose.Schema.Types.ObjectId, ref: 'users' }],
    image:{type:String}

})
```

View:

- ➔ Data representation is done by the View component.
- ➔ It actually generates UI or user interface for the user.
- ➔ So in web applications when you think of the View component just think of it as the HTML/CSS part.
- ➔ Views are created by the data which is collected by the model component but this data isn't taken directly from the Model and instead obtained through the controller.
- ➔ View only speaks to the Controller.

Controller:

- ➔ It's known as the main man because the controller is the component that enables the interconnection between the View and the Model, so it acts as an intermediary.
- ➔ The Controller doesn't have to worry about handling data logic, it just tells the Model what to do.
- ➔ After receiving data from the Model, the Controller processes it and sends the information to the View.

Express & NodeJS do all the functional programming and will be used to write the Business Tier.
This tier represents the Application Server that acts as the bridge of communication between the Client and Database. This tier will serve the React components to the user's device and accept HTTP requests from the user and follow with the appropriate Response.

Here is an overview of the Controllers in our project.

1. User Controller

```
// USER LOGIN

router.post('/login', async (req ... , res :Response<ResBody, LocalsObj> ) => {
  // console.log('In login server')
  const { username, password } = req.body
  const user = await UserModel.findOne({ username })
  // console.log(user)
  if (!user) {
    res.status(400)
    res.json({ message: 'User dont exist' })
    return res
  }
  const isPasswordValid = await bcrypt.compare(password, user.password)
  if (!isPasswordValid) {
    res.status(400)
    res.json({ message: 'Username or Password is incorrect' })
    return res
  }
  const token = jwt.sign({ id: user._id }, 'secret')
  res.json({ token, userID: user._id }).status(200)
})
```

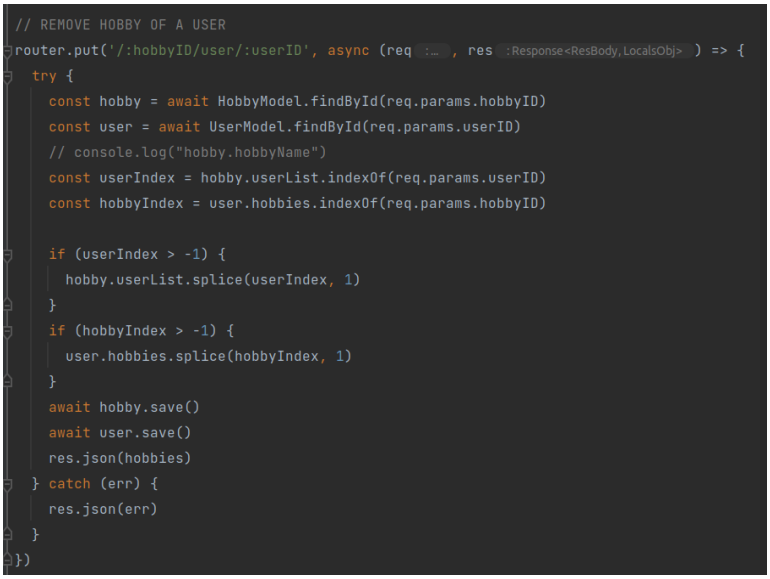
Create a new user

2. Event Controller

```
router.get('/createdEvents/:userID', async (req, res) => {
  try {
    const id = await UserModel.findById(req.params.userID)
    const dist = id.district
    const hobbies = id.hobbies
    let response = await EventModel.find({ userOwner: req.params.userID })
    response = response.map((e, i) => {
      console.log(e)
      let d = e._doc
      d.image = 'image'
      return d
    })
    let result = []
    for (let index = 0; index < response.length; index++) {
      for (let index1 = 0; index1 < hobbies.length; index1++) {
        let hobby = await HobbyModel.findById(hobbies[index1])
        if (response[index].hobbyname === hobby.hobbyName) {
          // console.log(hobby.image)
          response[index].image = hobby.image
          // console.log(response[index].image)
          result.push(response[index])
          break
        }
      }
    }
    console.log(result)
    res.json(response)
  } catch (err) {
    res.status(400)
    res.json(err.message)
  }
})
```

Fetch all the events based on user's location & hobbies

3. Hobby Controller



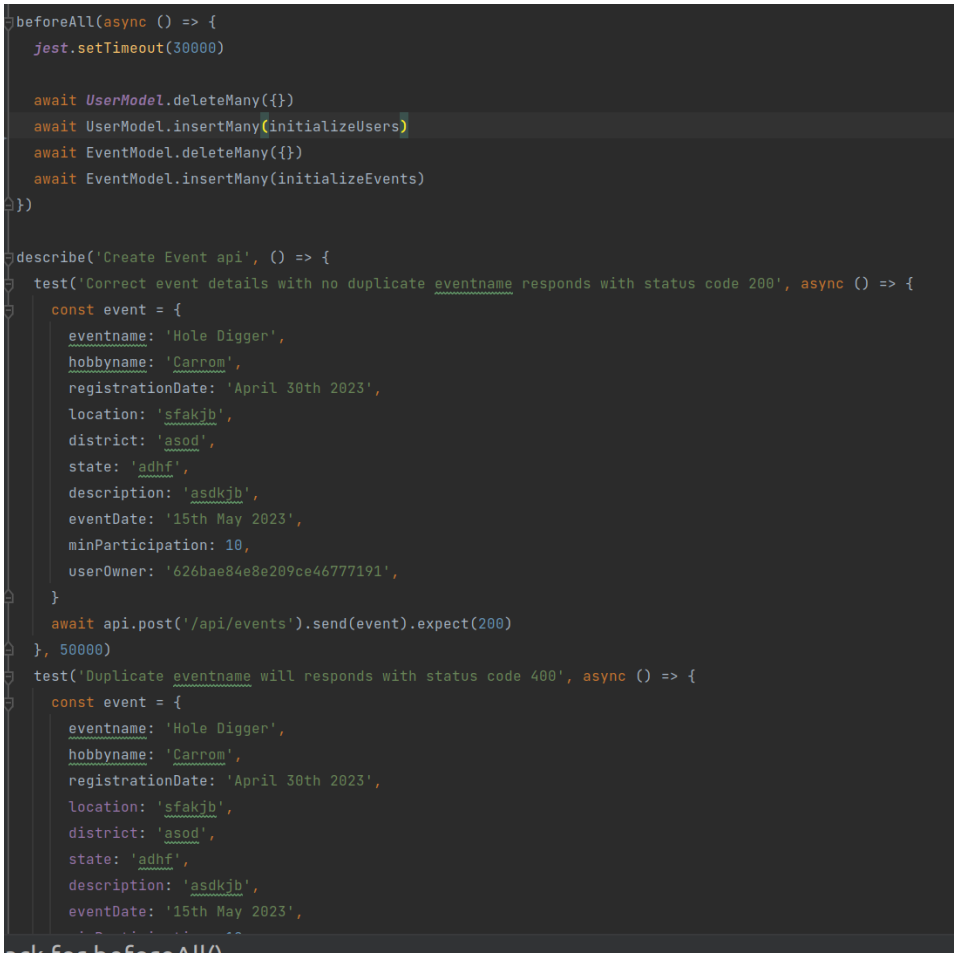
```
// REMOVE HOBBY OF A USER
router.put('/:hobbyID/user/:userID', async (req , res ) => {
  try {
    const hobby = await HobbyModel.findById(req.params.hobbyID)
    const user = await UserModel.findById(req.params.userID)
    // console.log("hobby.hobbyName")
    const userIndex = hobby.userList.indexOf(req.params.userID)
    const hobbyIndex = user.hobbies.indexOf(req.params.hobbyID)

    if (userIndex > -1) {
      hobby.userList.splice(userIndex, 1)
    }
    if (hobbyIndex > -1) {
      user.hobbies.splice(hobbyIndex, 1)
    }
    await hobby.save()
    await user.save()
    res.json(hobbies)
  } catch (err) {
    res.json(err)
  }
})
```

Remove hobby of a user

utils directory
Contains config file to support controller's functionalities.

tests directory
Contains the supertest test files as described in the SDLC section.

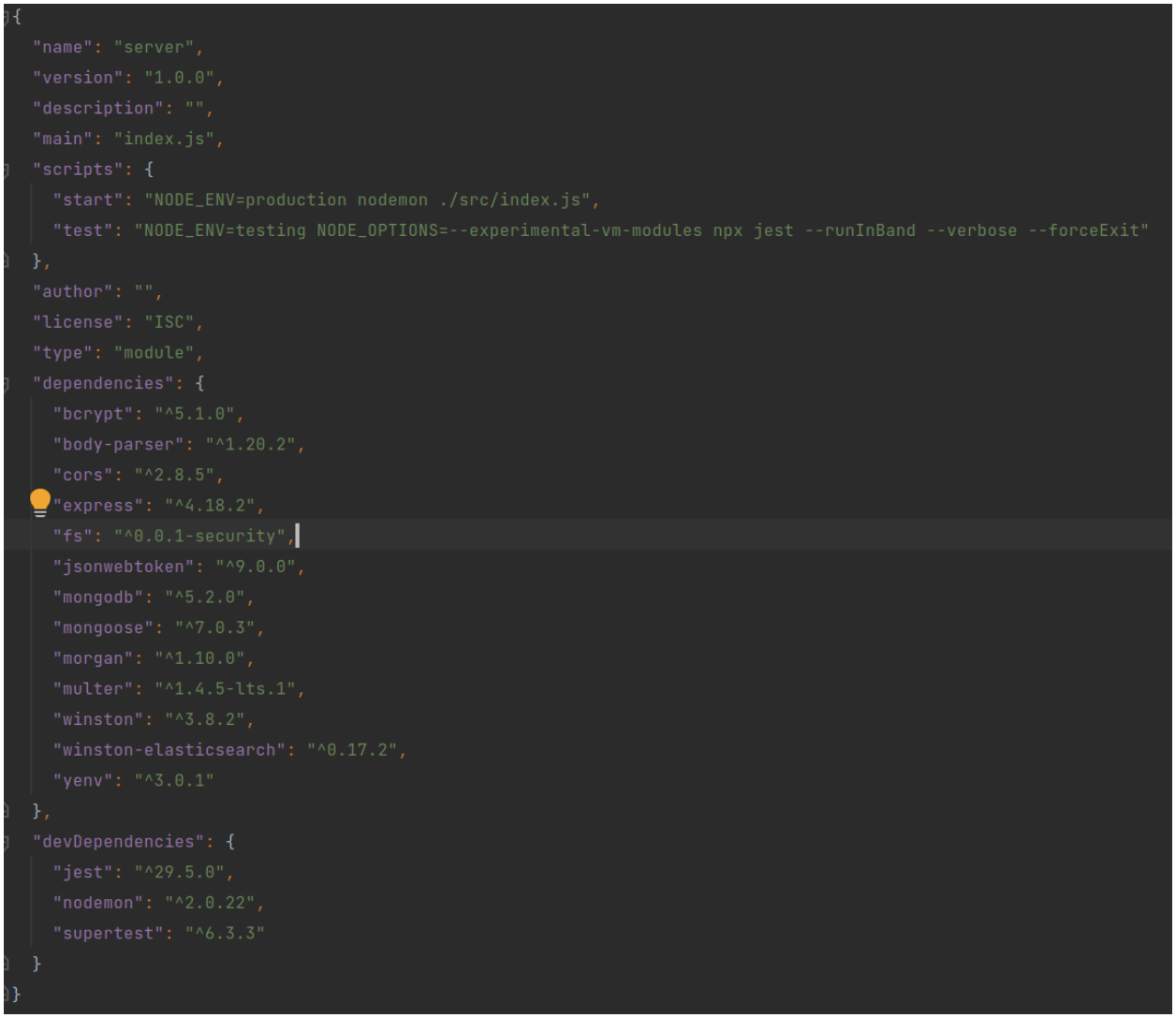
Here is a snippet of *event_api.test.js* which shows how we initialize our db before testing any API.



```
beforeAll(async () => {
  jest.setTimeout(30000)

  await UserModel.deleteMany({})
  await UserModel.insertMany(initializeUsers)
  await EventModel.deleteMany({})
  await EventModel.insertMany(initializeEvents)
})

describe('Create Event api', () => {
  test('Correct event details with no duplicate eventname responds with status code 200', async () => {
    const event = {
      eventname: 'Hole Digger',
      hobbyname: 'Carrom',
      registrationDate: 'April 30th 2023',
      location: 'sfakjb',
      district: 'asod',
      state: 'adhf',
      description: 'asdkjb',
      eventDate: '15th May 2023',
      minParticipation: 10,
      userOwner: '626bae84e8e209ce46777191',
    }
    await api.post('/api/events').send(event).expect(200)
  }, 50000)
  test('Duplicate eventname will responds with status code 400', async () => {
    const event = {
      eventname: 'Hole Digger',
      hobbyname: 'Carrom',
      registrationDate: 'April 30th 2023',
      location: 'sfakjb',
      district: 'asod',
      state: 'adhf',
      description: 'asdkjb',
      eventDate: '15th May 2023',
```

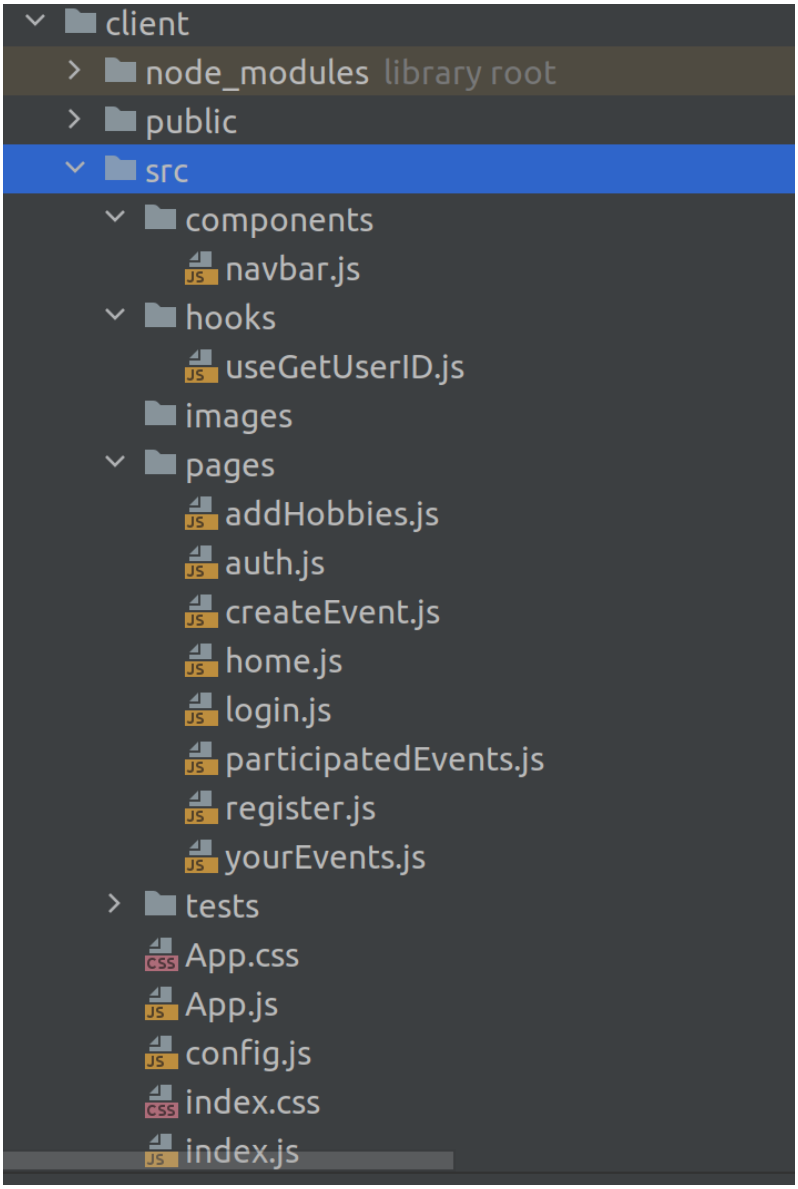
package.json

This file is the heart of our Node project. It records important metadata about our project which is required before publishing to npm and also defines functional attributes of our project that npm uses to install dependencies, run scripts, and identify the entry point to our package.



```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "NODE_ENV=production nodemon ./src/index.js",
    "test": "NODE_ENV=testing NODE_OPTIONS=--experimental-vm-modules npx jest --runInBand --verbose --forceExit"
  },
  "author": "",
  "license": "ISC",
  "type": "module",
  "dependencies": {
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "fs": "^0.0.1-security",
    "jsonwebtoken": "^9.0.0",
    "mongodb": "^5.2.0",
    "mongoose": "^7.0.3",
    "morgan": "^1.10.0",
    "multer": "^1.4.5-lts.1",
    "winston": "^3.8.2",
    "winston-elasticsearch": "^0.17.2",
    "yenv": "^3.0.1"
  },
  "devDependencies": {
    "jest": "^29.5.0",
    "nodemon": "^2.0.22",
    "supertest": "^6.3.3"
  }
}
```

5.3.2 Frontend



Directory Structure

```jsx
return (
    <div className="App">
      <Router>
        {
            cookies.access_token &&
            <NavBar />
        }

        <Routes>
          <Route path="/" element={<Auth />}></Route>
          <Route path="/register" element={<Register />}></Route>
          <Route path="/home" element={<Home />}></Route>
          <Route path="/addHobbies" element={<AddHobbies />}></Route>
          <Route path="/yourEvents" element={<YourEvents />}></Route>
          <Route
              path="/participatedEvents"
              element={<ParticipatedEvents />}
          ></Route>
          <Route path="/createEvent" element={<CreateEvent />}></Route>
        </Routes>
      </Router>
    </div>
```

JSX returned by App component

```jsx
export const useGetUserID = () => {
    return window.localStorage.getItem( key: 'userID')
}
```

Fetch logged in user's ID from local storage

```jsx
export default function Loader(){
    return(
        <div className='loader-container'>
            <svg className="spinner" viewBox="0 0 50 50">
                <circle className="path" cx="25" cy="25" r="20" fill="none" strokeWidth="5"></circle>
            </svg>
        </div>
    )
}
```

Loader component

```jsx
return (
    <Navbar collapseOnSelect expand="lg" bg="dark" variant={"dark"}>
        <Navbar.Toggle aria-controls="navbarScroll"/>
        <Navbar.Collapse id="navbarScroll">
            <Nav>
                <Nav.Link eventKey="1" as={Link} to="/home">Home</Nav.Link>
                <Nav.Link eventKey="2" as={Link} to="/addHobbies">Add hobbies</Nav.Link>
                <Nav.Link eventKey="3" as={Link} to="/createEvent">Create Event</Nav.Link>
                <Nav.Link eventKey="4" as={Link} to="/yourEvents">Your Events</Nav.Link>
                <Nav.Link eventKey="5" as={Link} to="/participatedEvents">Participated Events</Nav.Link>
            </Nav>
        </Navbar.Collapse>
        <Button
            onClick={logout}
            variant="outline-danger"
        >
            Logout
        </Button>
    </Navbar>
```

NavBar component

```jsx
const BACKEND_URL = `${process.env.REACT_APP_BACKEND_URL}`
const FRONTEND_URL = `${process.env.REACT_APP_FRONTEND_URL}`
```

config.js

```
REACT_APP_BACKEND_URL=http://socialconnect.com
REACT_APP_FRONTEND_URL=http://socialconnect.com
```

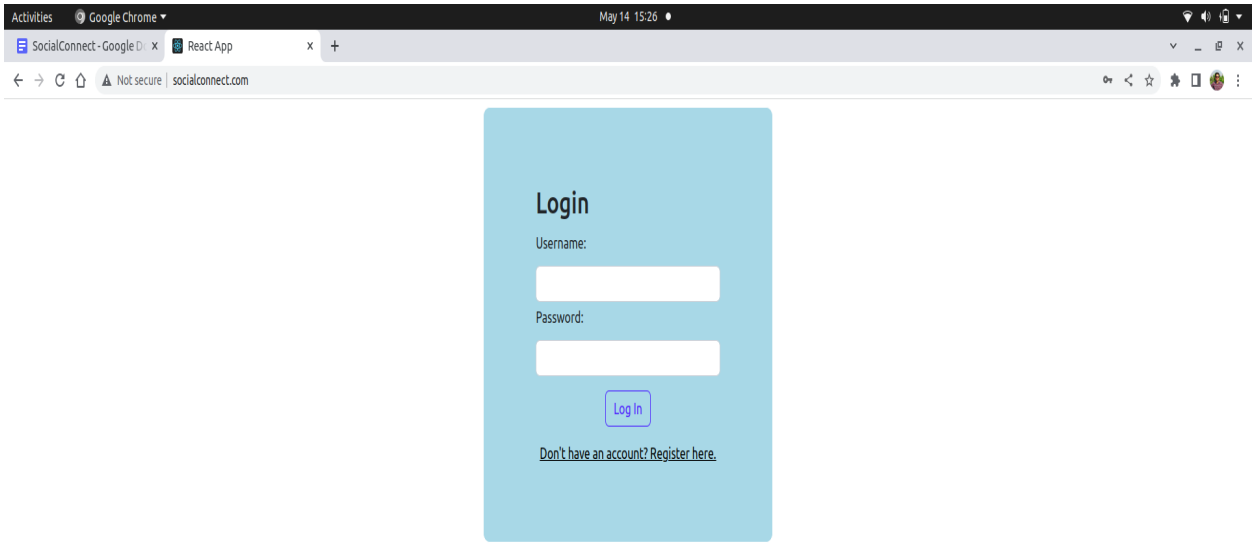
Environment variables

```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.3.5",
    "bootstrap": "^5.2.3",
    "jquery": "^3.6.4",
    "react": "^18.2.0",
    "react-bootstrap": "^2.7.4",
    "react-cookie": "^4.1.1",
    "react-date-picker": "^10.0.3",
    "react-dom": "^18.2.0",
    "react-multiple-select-dropdown-lite": "^2.0.6",
    "react-router-dom": "^6.10.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
```

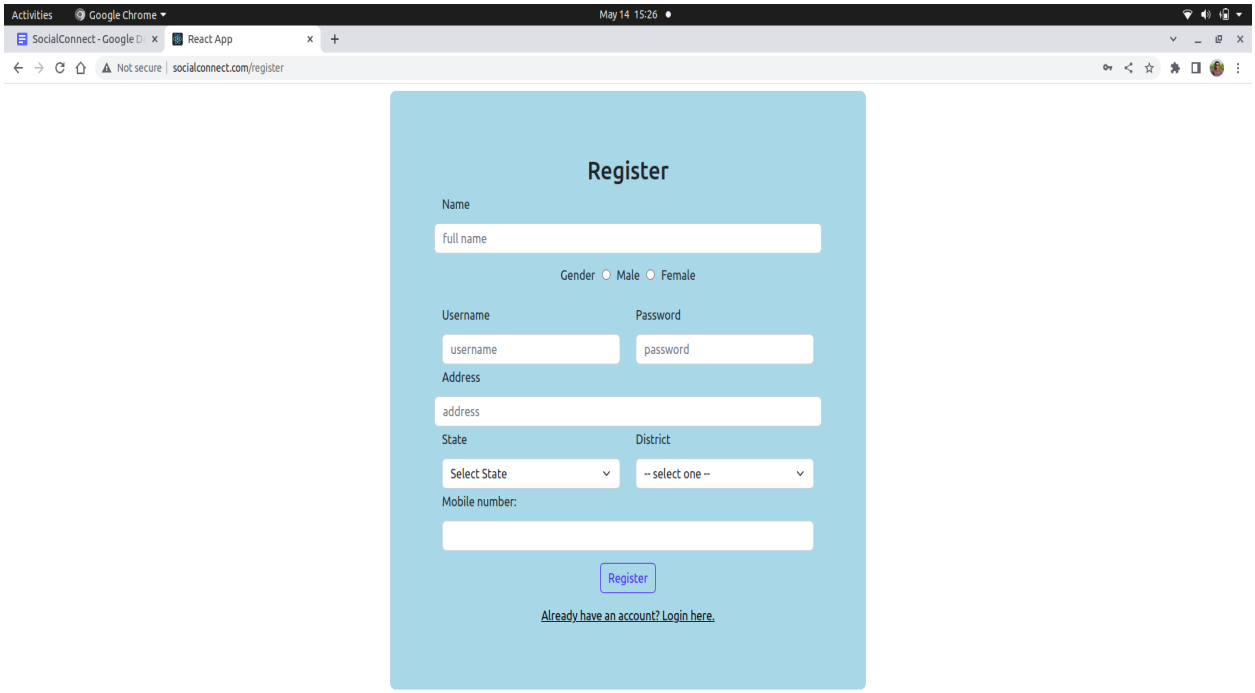
Package.json

6. Result and discussion

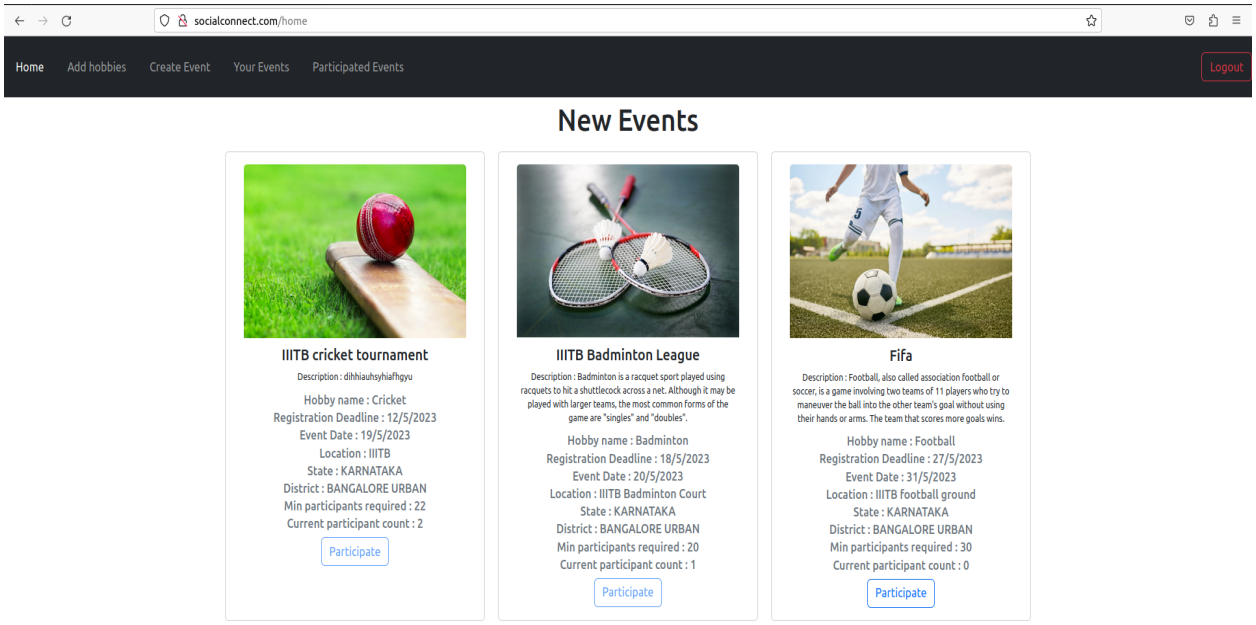
1. Login Page



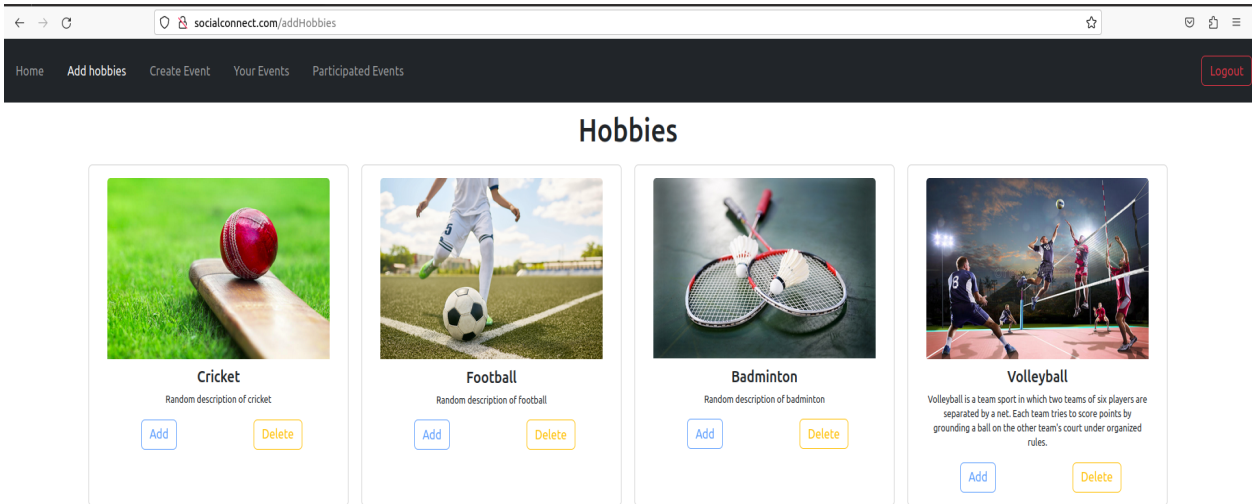
2. Register Page:



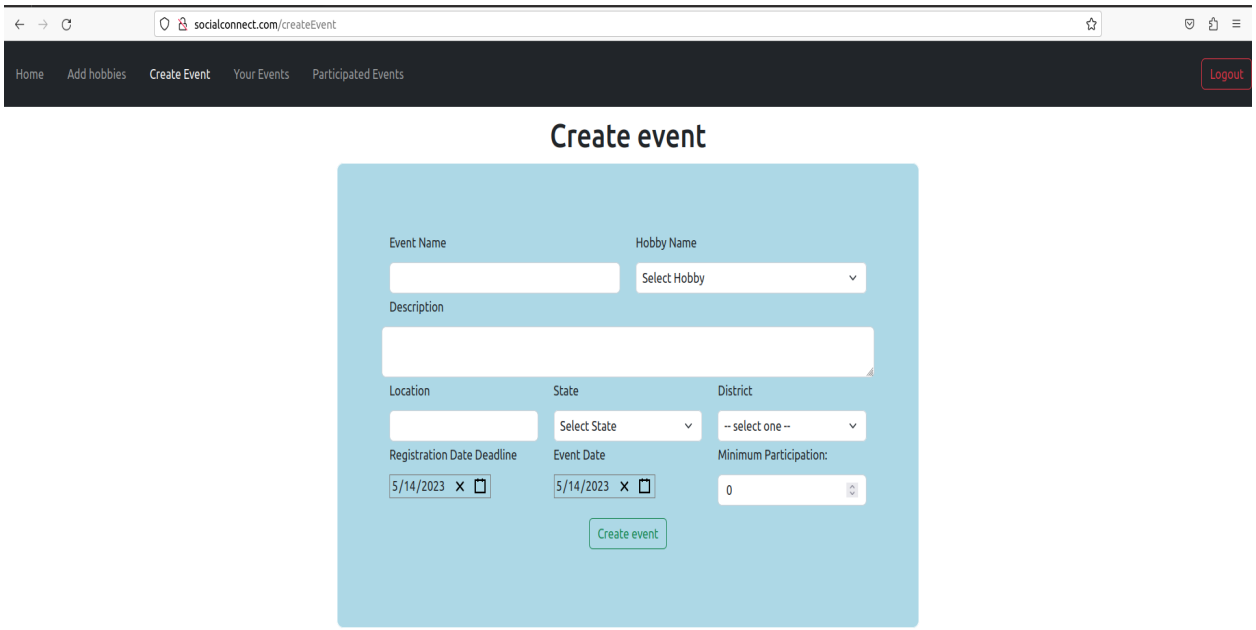
3. Home Page:



4. Hobbies page:

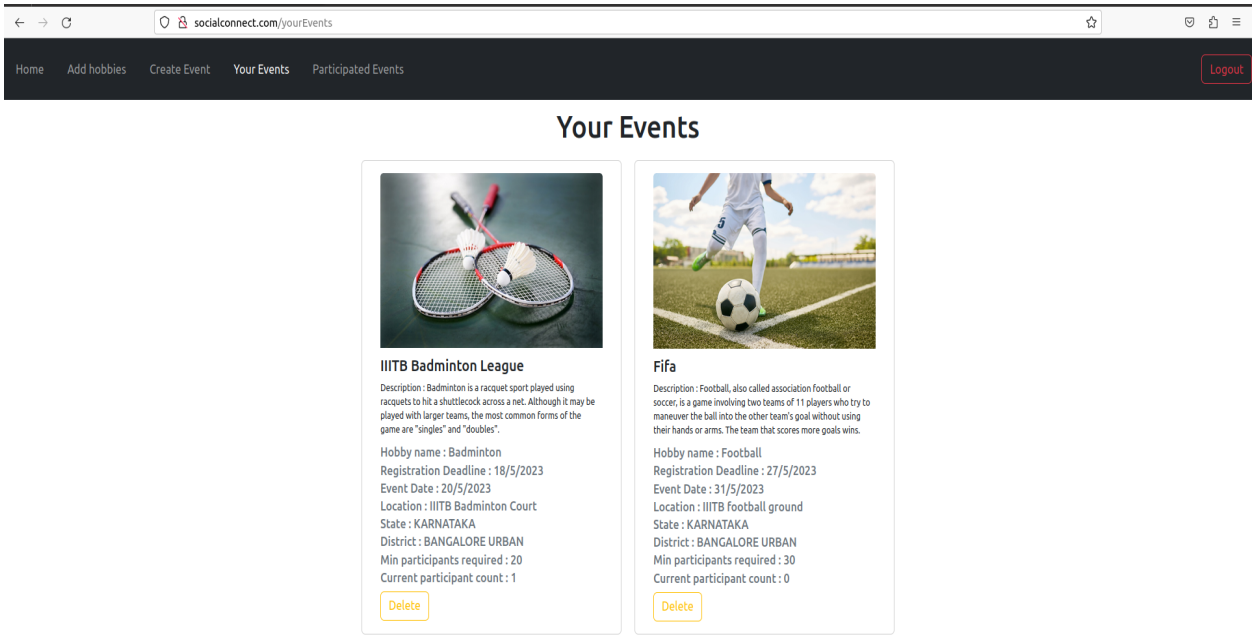


5. Create Event Page :

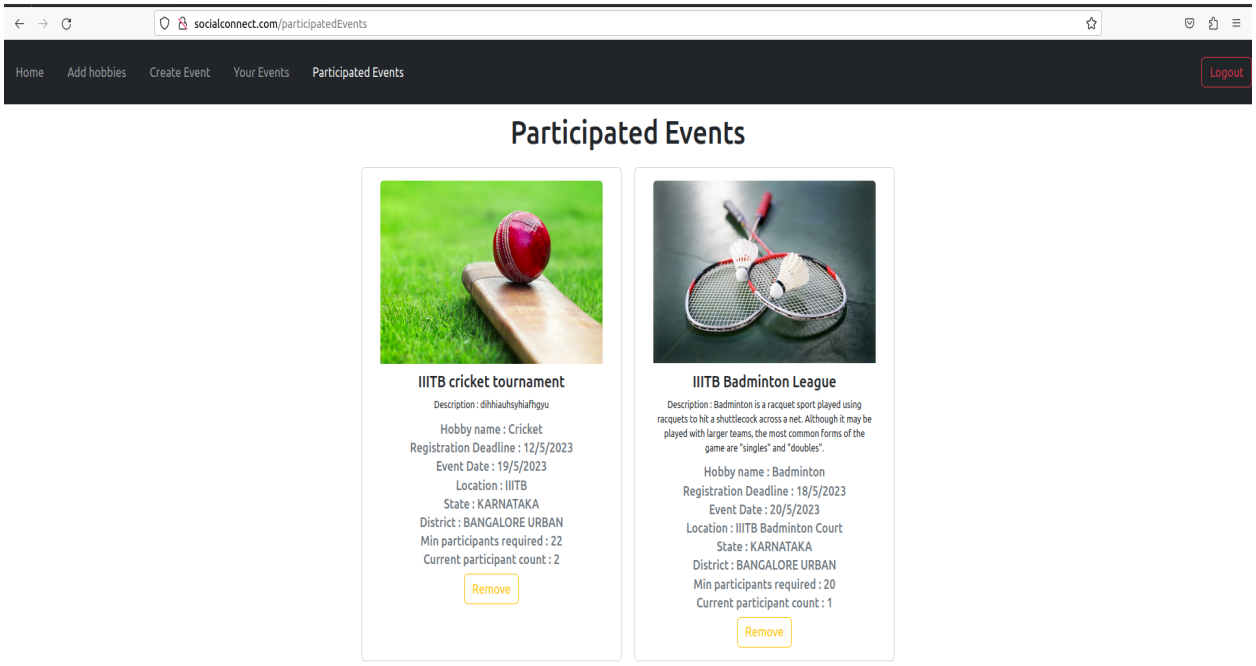


---

6. Your Events Page:



7. Participated Events Page:



7. Scope for Future Work

a. Users can see exact map locations on google maps integrated on the event page.
b. Users can make friends and chat with other people.
c. Users can share photos and videos of events.
d. Users get automated email if a new event is created or removed.

8. Conclusion:

We have successfully created the web-app SocialConnect that will be helpful for users to share their watchlists with anyone they would want to.

We have integrated the entire DevOps CI/CD pipeline with the help of Jenkins.

Used Kubernetes containers for increasing portability and adopting a microservice architecture development.

Used Docker Compose to orchestrate container deployment and manage them.

Used supertest for backend API testing.

Used Ansible for deployment and Ansible Vault with Jenkins for secrets management with templating via yenv.

9. References

1. https://fullstackopen.com/
2. https://github.com/john-smilga/node-express-course
3. https://reactjs.org/docs/getting-started.html
4. https://expressjs.com/
5. https://mongoosejs.com/docs/api.html
6. https://www.npmjs.com/
7. https://docs.ansible.com/
8. https://blog.ktz.me/secret-management-with-docker-compose-and-ansible/
9. https://www.youtube.com/watch?v=X48VuDVv0do&ab_channel=TechWorldwithNana