CMPE 273, Spring 2022

Under the supervision of Professor: Dr. Simon Shim

Group 1

Roshan Chokshi
Ravi Shanker Thadishetti
Ravi Teja Kallepalli
Subhash Reddy
Saikrishna Dosapati
Sandeep Birudukota
Varun Raj Badri

Individual Contributions:

1. Roshan Chokshi - Primarily worked on the backend to build the comments, post-tags of the Stack overflow application. Built the backend flow for the chat service, and dashboard of the application. Integrated the all the services to enable users to post, comment and share search answers on the application.

2. Ravi Shanker Thadishetti - Worked on the initial architecture design and the backend of the application, developed to set-up a framework for tags and comments, admin section. Integrated the react components with the multiple APIs and worked on the user authentication using the JWT. Configured kafka to enable the chat service, posts and tags for user.

3. Ravi Teja Kallepalli - Developed the User Interface of the dashboard of the application and built multiple components for the users to post questions and comments. Worked on the answers vote component to enable the users to up and down vote a particular answer on the application.

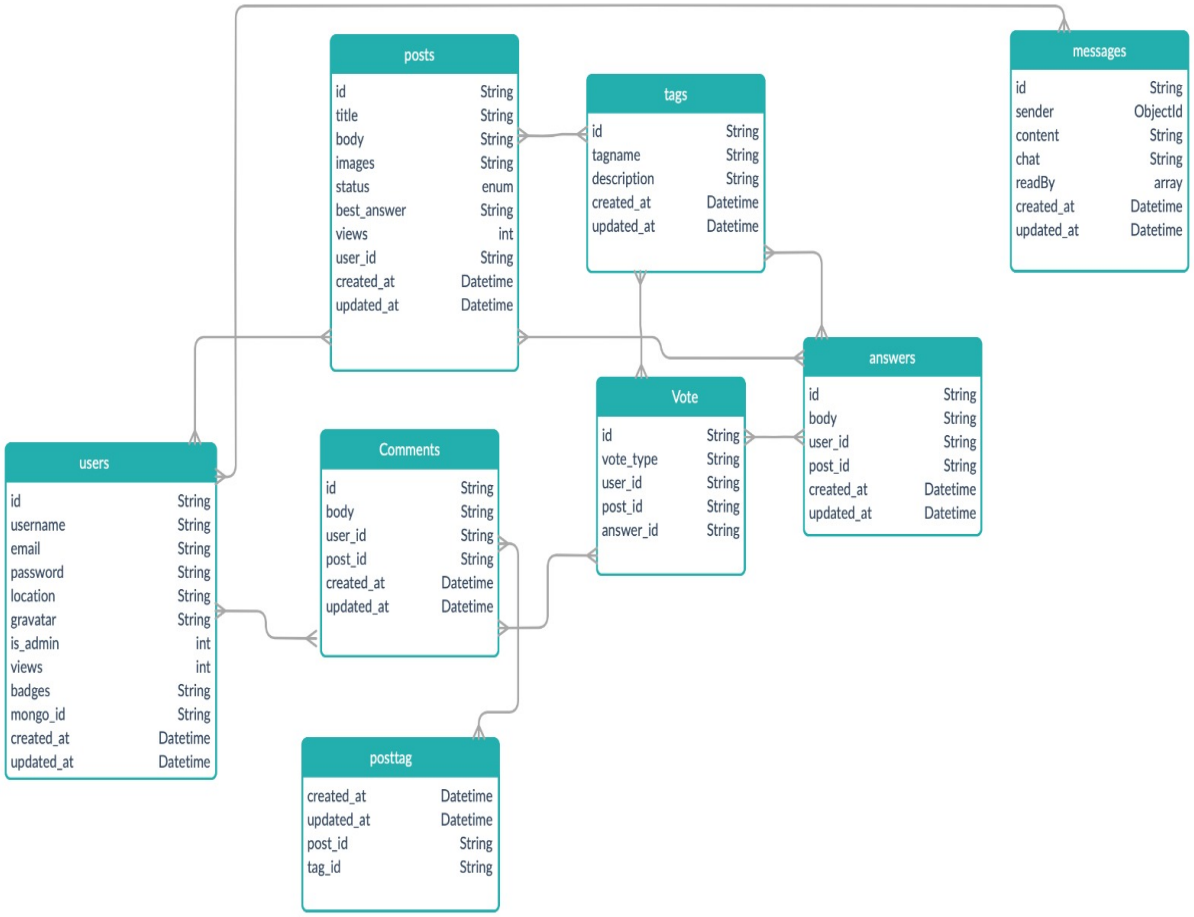
4. Subhash Reddy - Developed User interface for the admin view to show the user all the questions and answers. Integrated the admin UI components with the backend APIs to fetch the responses from the APIs.

5. Saikrishna Dosapati - Worked on developing the User interface and deploying the application on the EC2 instance, implemented the UI components to implement badges and tags, and implemented Integrated the mongo db and MySQL server with the application.

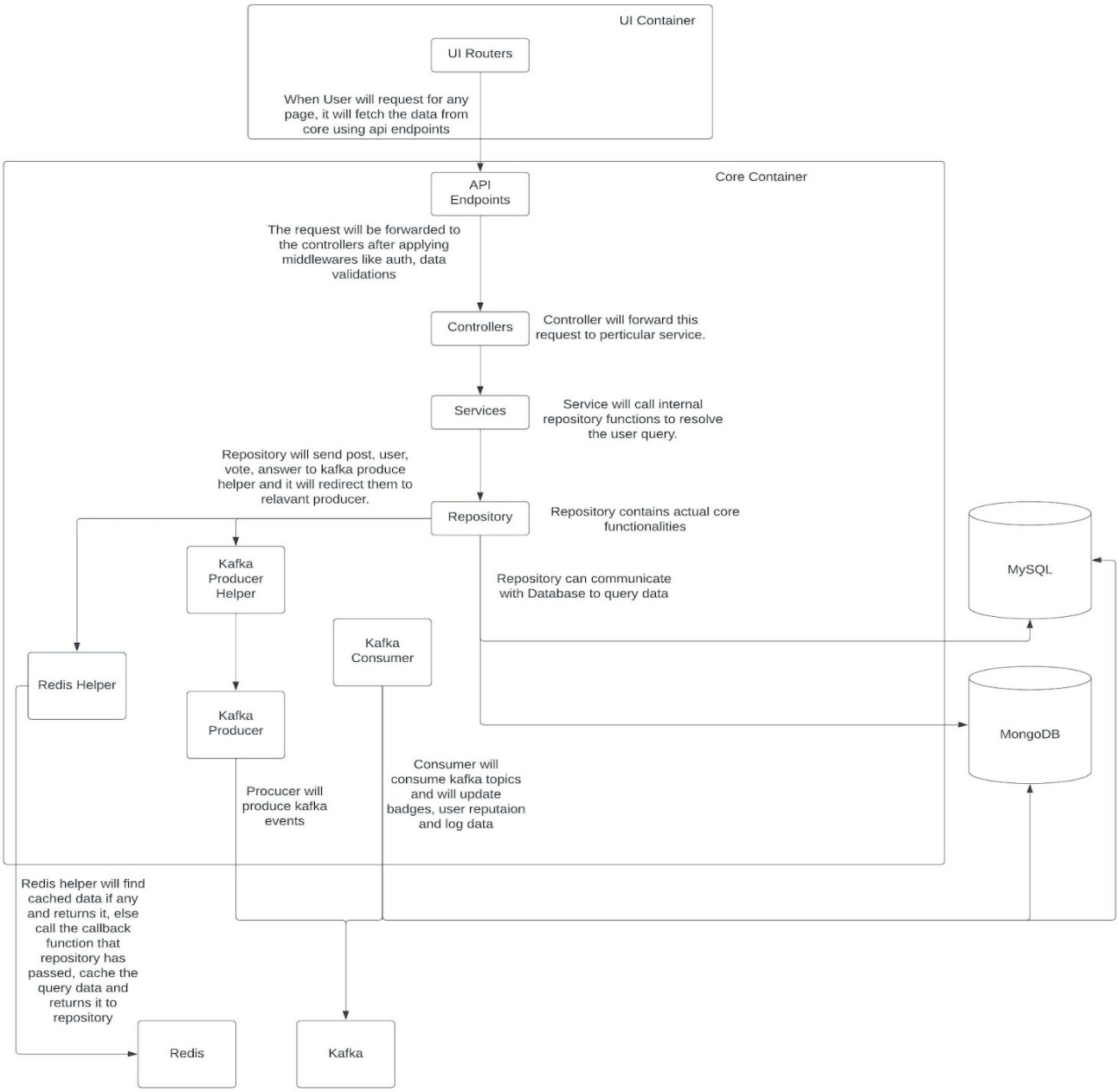
6. Sandeep Birudukota - Worked on developing the UI models and controllers for the admin portal. To enable the admin approval feature when a user post an answer with images. Developed the login and signup components and integrated with the APIs to authenticate the users onto the application.

7. Varun Raj Badri - Designed and Developed the UserInterface for the chat feature on the application, worked to integrate the APIs to test the end to end flow of the chat feature among the users of the application.

Data Base Schema:-



System Architecture:-



Object Management Policy:

We used several storage management systems to store various types of things. We used MongoDB to store things like Messages and badges since it allows us to store complicated structures. MongoDB also improves data retrieval faster by minimizing joins and making it easier to read. To preserve the ACID (Atomicity, Consistency, Isolation, and Durability) state, transactional data such as login credentials were kept in MySQL.

HANDLING HEAVYWEIGHT RESOURCES:

The demand on the server was handled and distributed via load balancing.
Redis caching was utilized for the most frequently used APIs to reduce the burden on the database, boosting speed and the retrieval process.
We used connection pooling to accomplish system performance and scalability. We were able to develop a fault tolerant and scalable system utilizing kafka because when the number of concurrent users rose, each job was done with certainty.

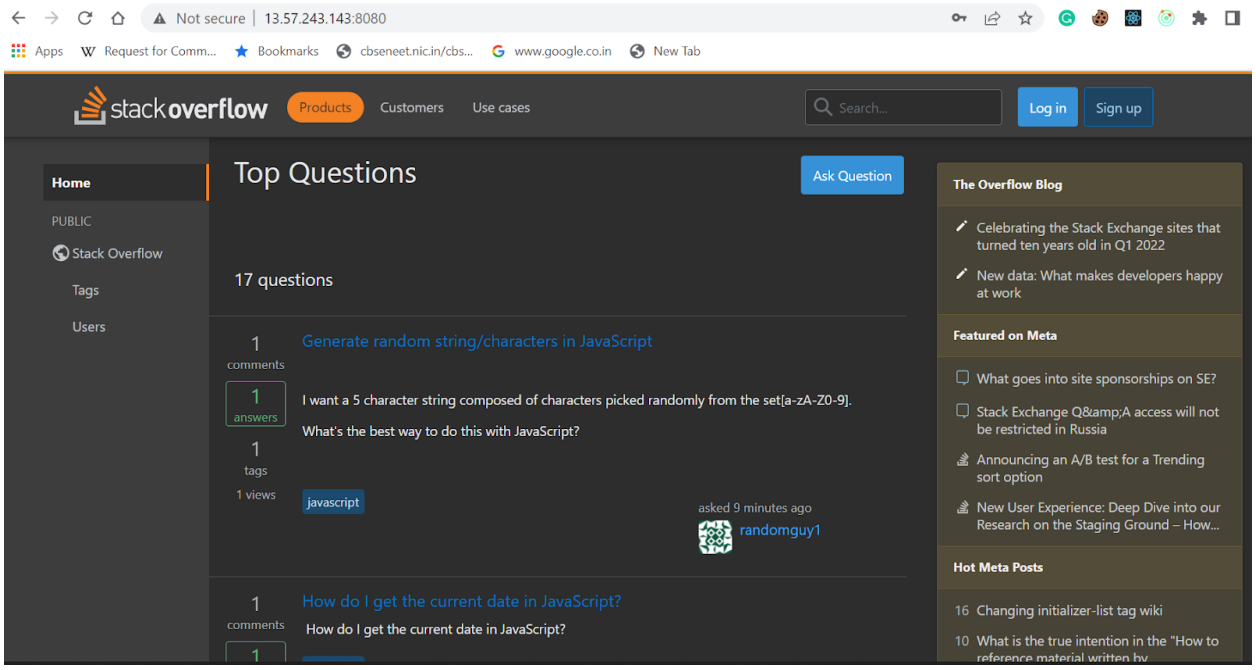
POLICY USED TO DECIDE ENTRY OF DATA INTO DATABASE:

MySQL: Login credentials are saved in MySQL for data and/or referential integrity. It also enables us to store data in an organized fashion and connect data from many databases. Most of the data is stored in this database.
MongoDB: The structure of mongodb is not defined

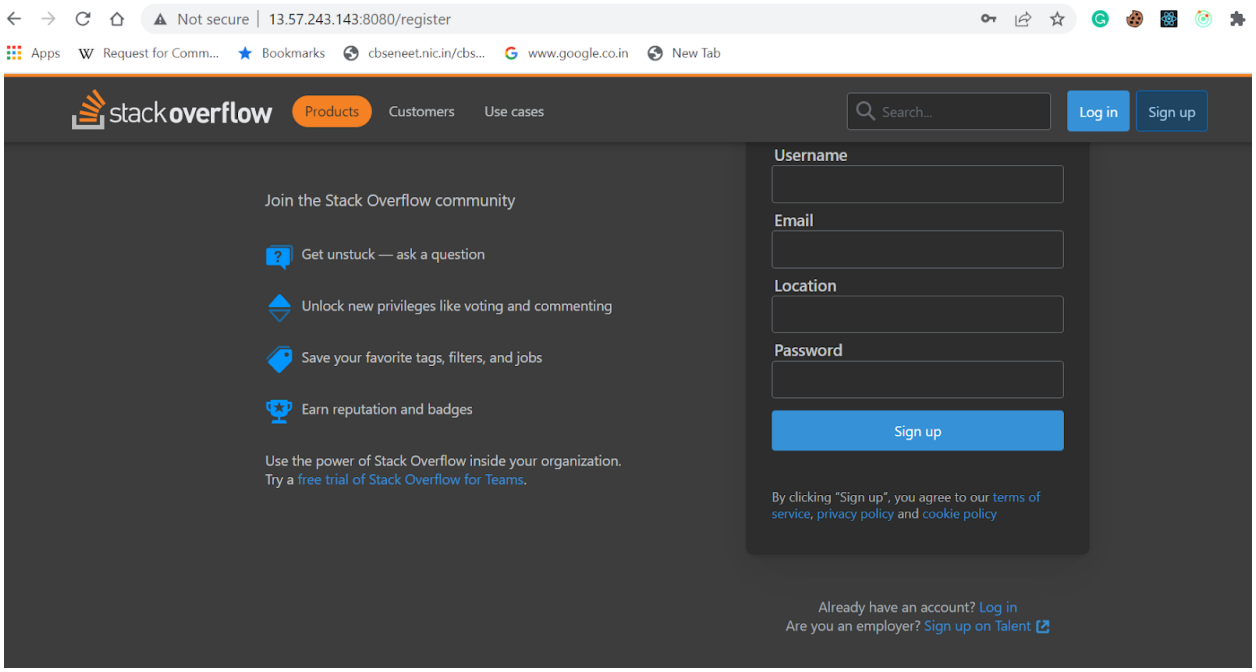
because it is a schema-less database. Messages and badges are stored in Mongodb for speedier retrieval.
The application's speed may be improved by using auto sharding and scaling.
The data was stored in the Redux store by several React components. Any change in the status prompted the server to change.

Screenshots of Application:

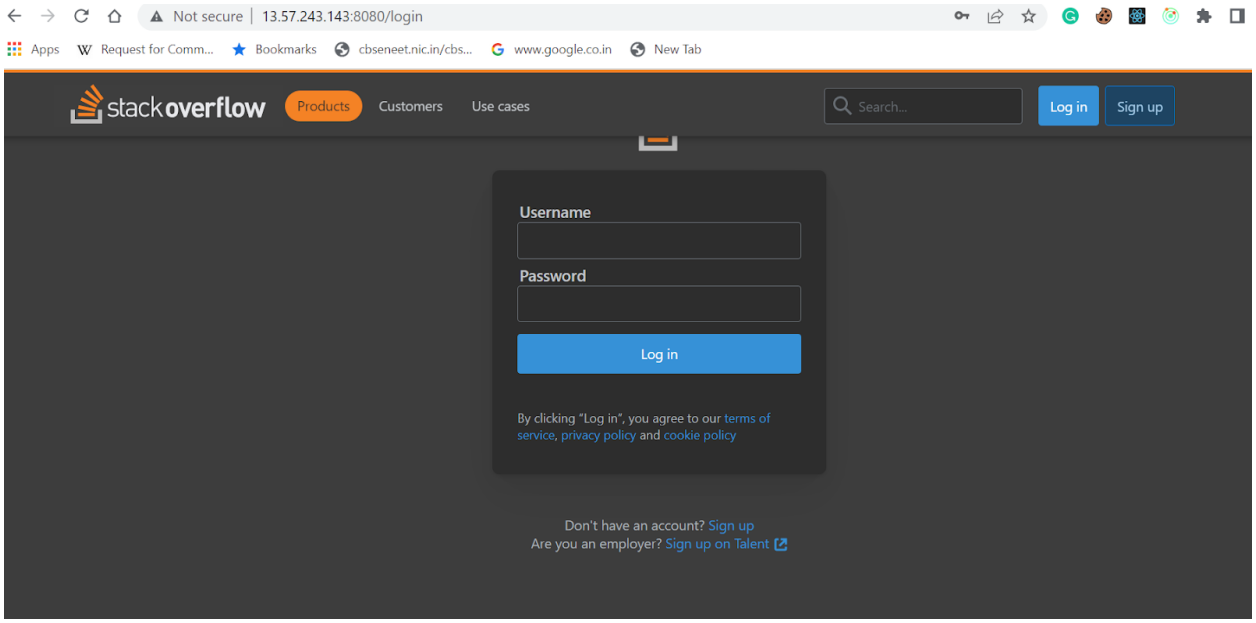
Landing Page



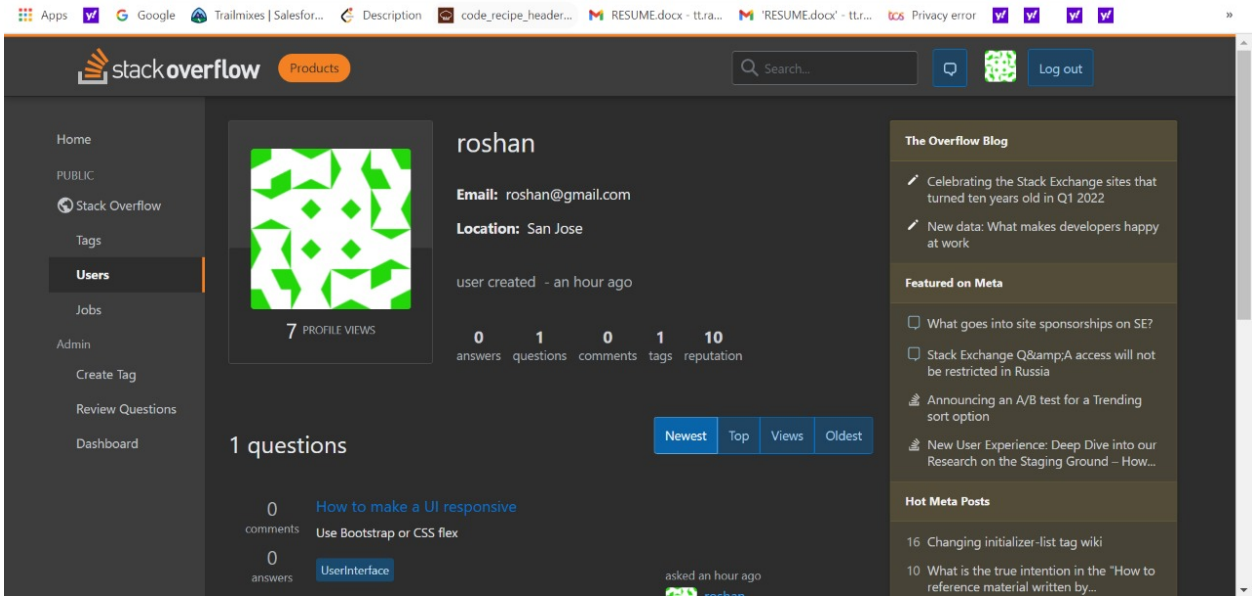
Register Page:



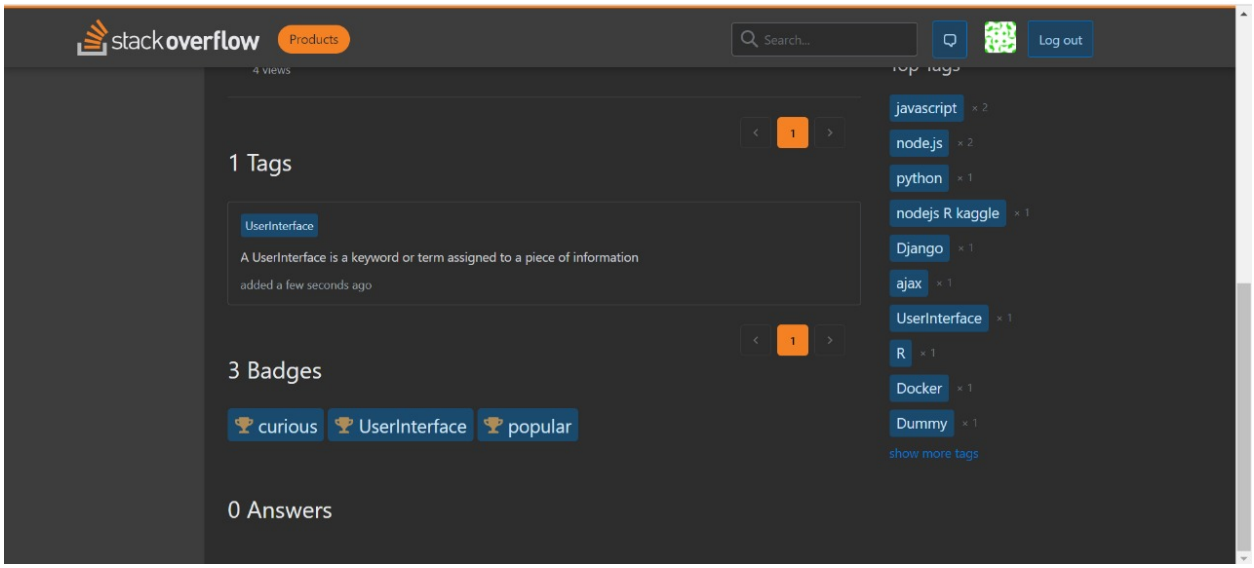
Login Page:



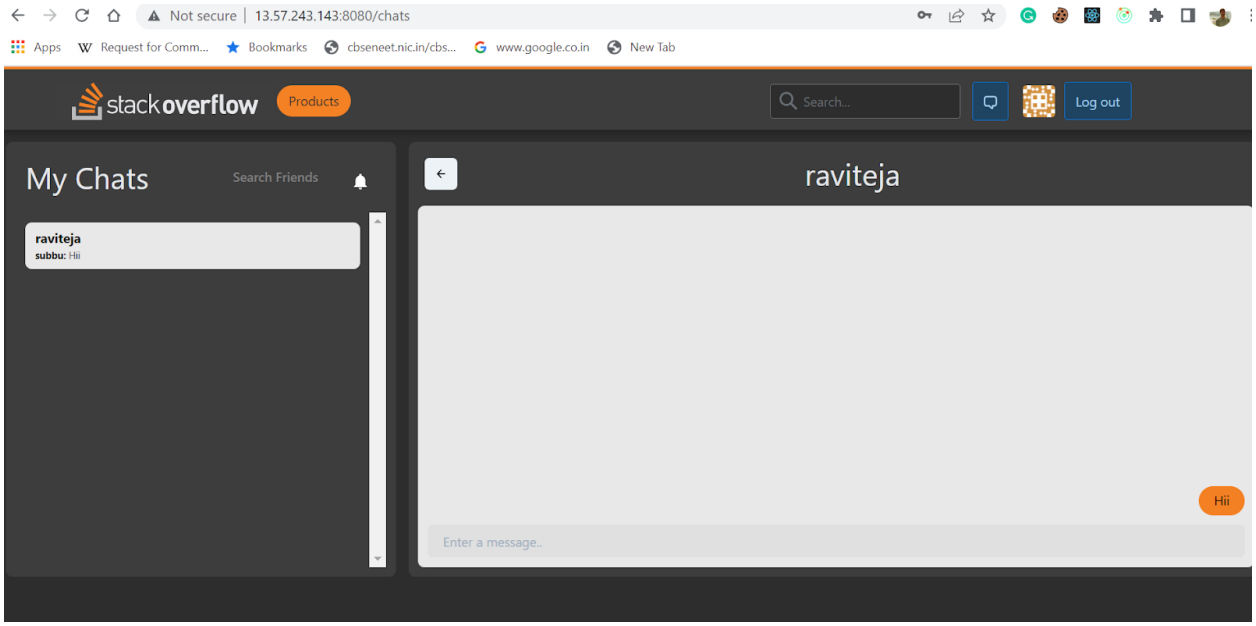
User Profile Page:



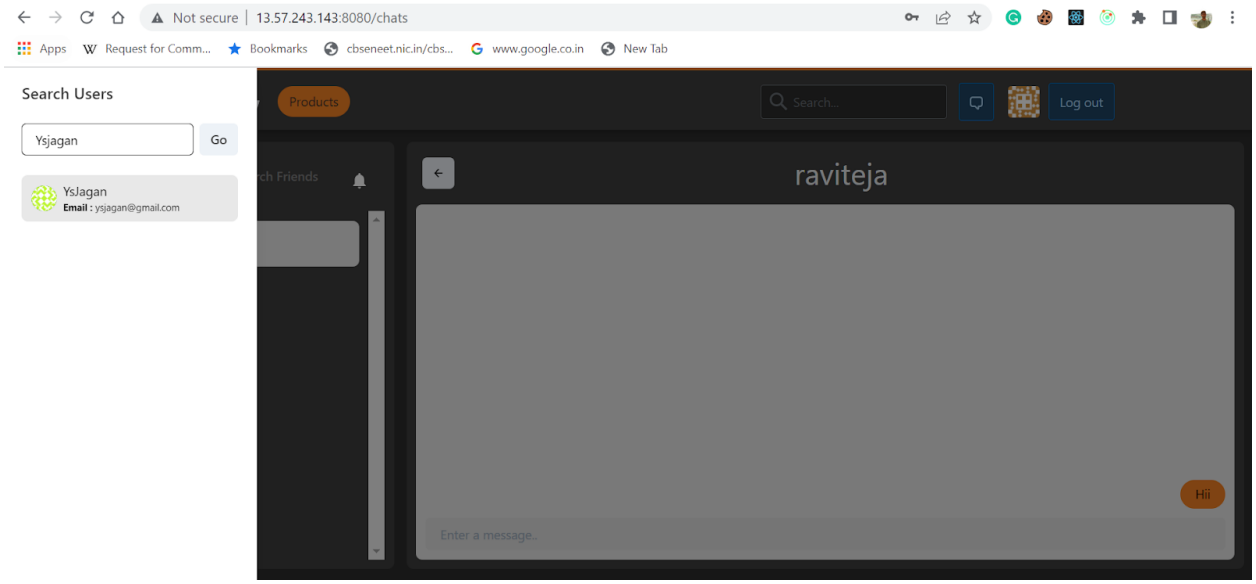
Tags, Badges & Answers:



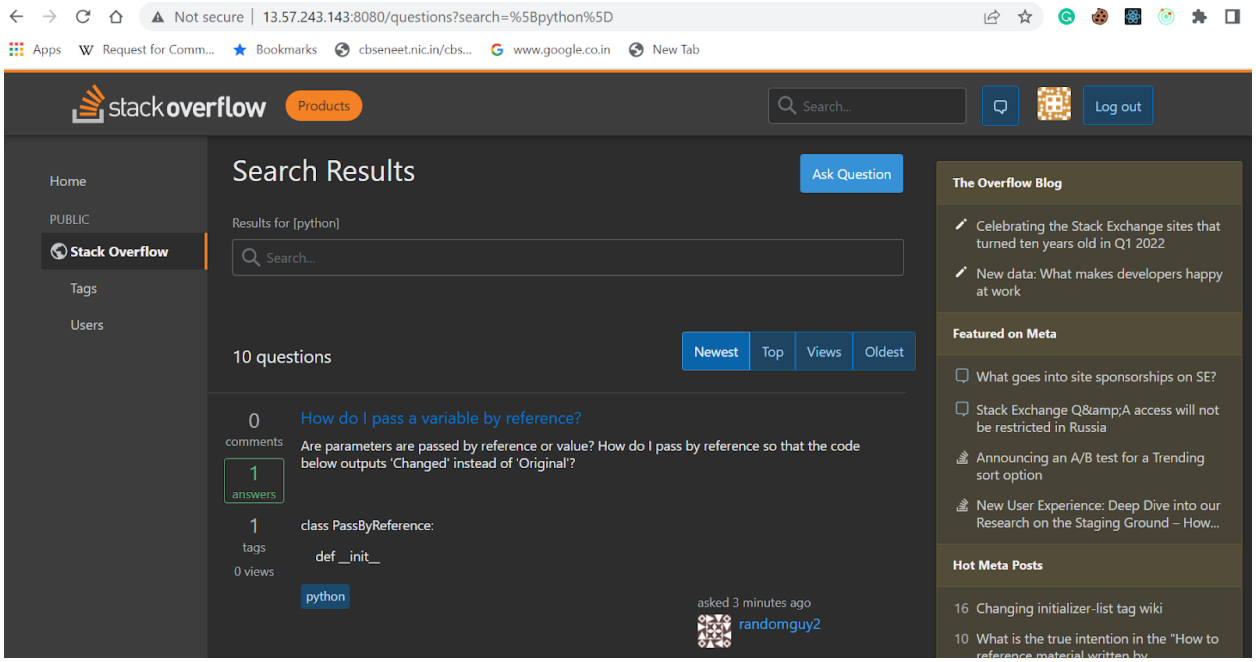
Chat Page:



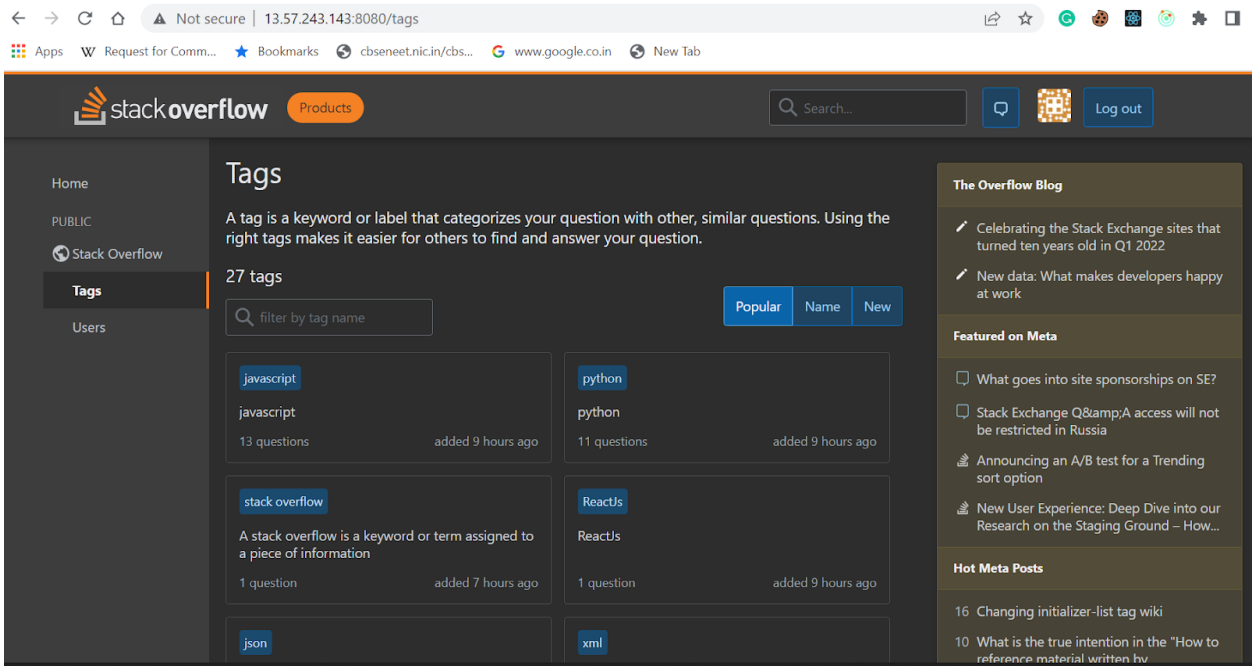
User Searching Page:



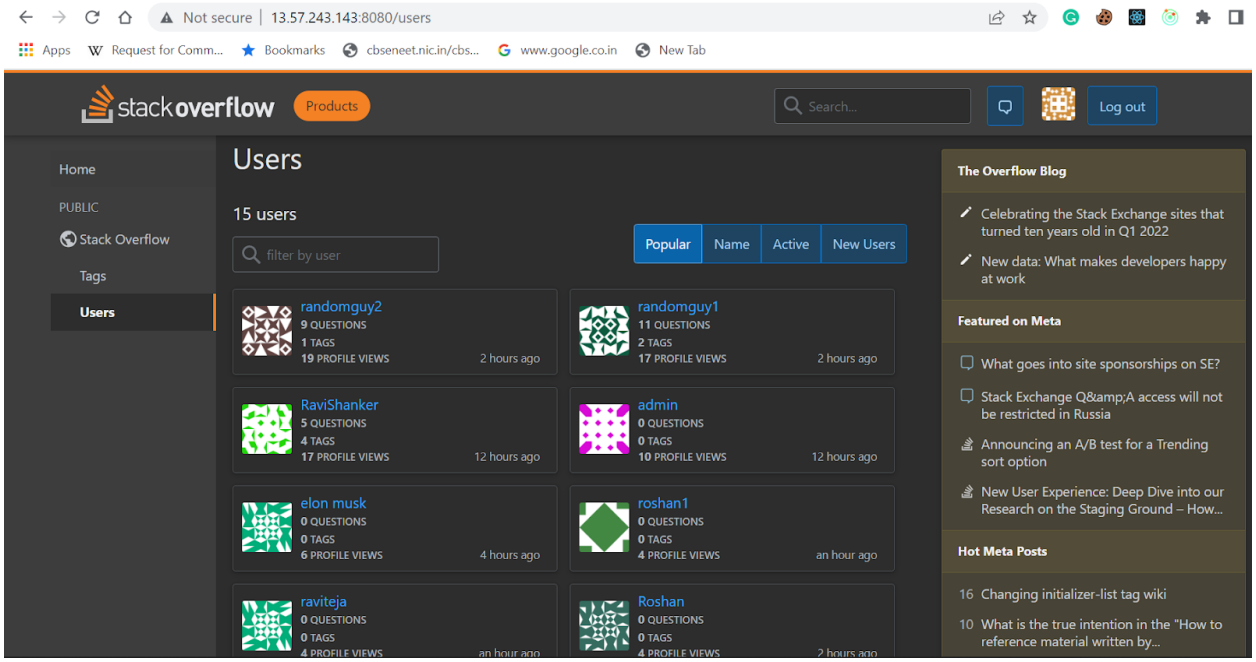
Question Searching:



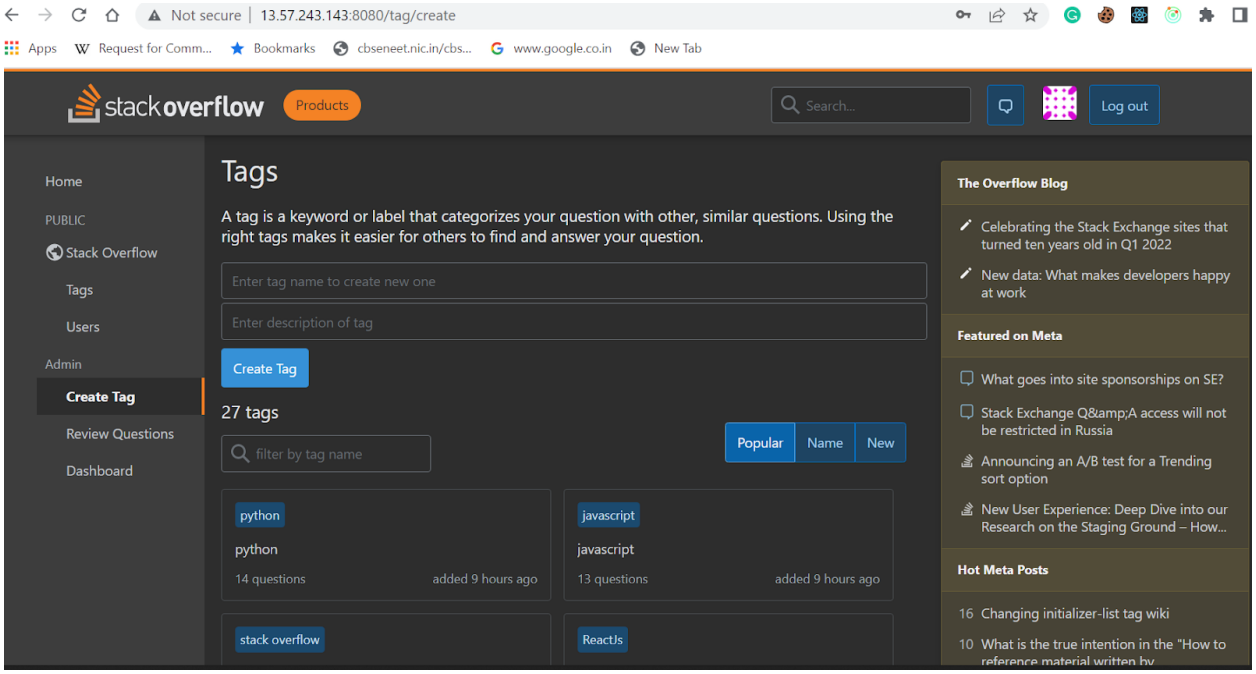
Tags Page:



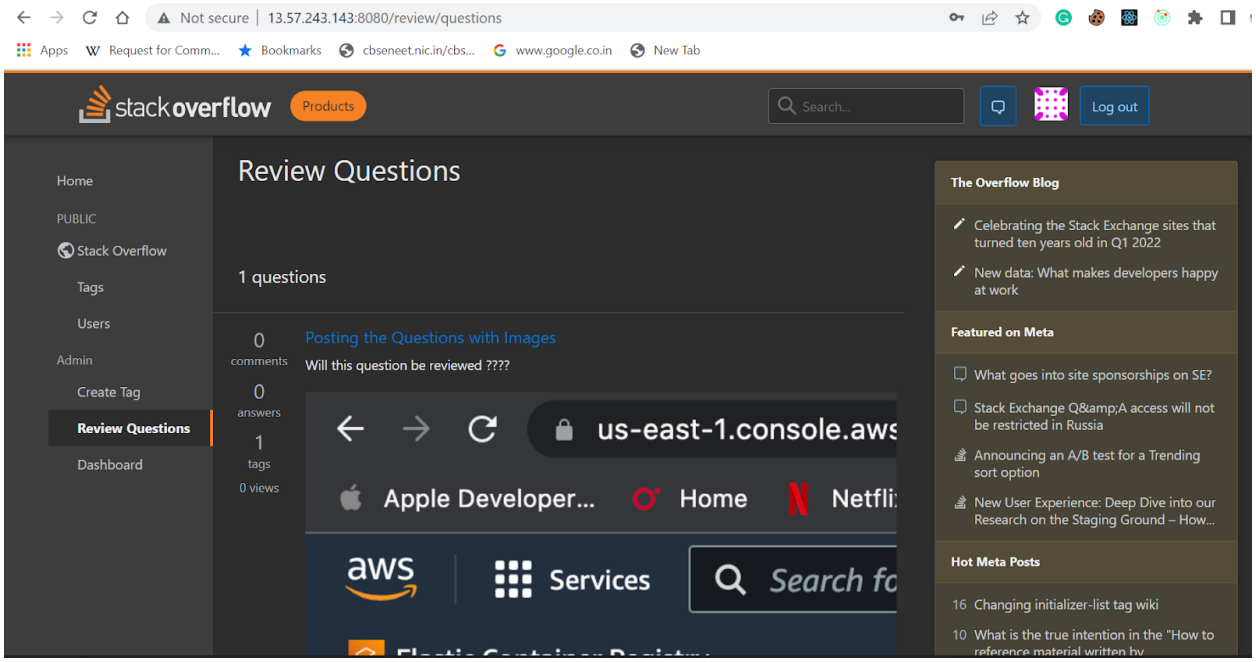
Users Page:



Admin Tag Creation Page:



Admin Question Reviewing Page:



A code listing your server implementation for entity objects

Users Model :

```
const mongoose = require('mongoose');
const { BADGES } = require('../constants');

const userSchema = mongoose.Schema(
 {
   name: {
     type: 'String',
     unique: true,
     required: true,
```

```
    },
    email: {
      type: 'String',
      unique: true,
      required: true,
    },
    pic: {
      type: 'String',
      required: true,
      default:

'https://icon-library.com/images/anonymous-avatar-icon/anonymous
-avatar-icon-25.jpg',
    },
    badges: {
      type: Map,
      of: {
        badgeType: {
          type: 'String',
          default: BADGES.BRONZE,
        },
        count: {
          type: Number,
          default: 0,
        },
      },
    },
  },
  { timestaps: true },
);

const MongoUser = mongoose.model('User', userSchema);
```

```
module.exports = MongoUser;
```

Messages Model :

```
const mongoose = require("mongoose");

const messageSchema = mongoose.Schema(
 {
   sender: { type: mongoose.Schema.Types.ObjectId, ref: "User"
},
   content: { type: String, trim: true },
   chat: { type: mongoose.Schema.Types.ObjectId, ref: "Chat" },
   readBy: [{ type: mongoose.Schema.Types.ObjectId, ref: "User"
}],
 },
 { timestamps: true }
);

const Message = mongoose.model("Message", messageSchema);
module.exports = Message;
```

Chats Models :

```
const mongoose = require("mongoose");

const chatModel = mongoose.Schema(
 {
   chatName: { type: String, trim: true },
   users: [{ type: mongoose.Schema.Types.ObjectId, ref: "User"
}],
   latestMessage: {
```

```
      type: mongoose.Schema.Types.ObjectId,
      ref: "Message",
    },
  },
  { timestamps: true }
);

const Chat = mongoose.model("Chat", chatModel);

module.exports = Chat;
```

Posts Models :

```
const { DataTypes } = require('sequelize');
const db = require('../config/db.config');

const Post = function (post) {
  this.title = post.title;
  this.body = post.body;
  this.userId = post.userId;
  this.tagName = post.tagName;
  this.images = post.images;
  this.status = post.status;
};

const PostsModelSequelize = db.define('posts', {
  id: {
    type: DataTypes.UUID,
    allowNull: false,
    primaryKey: true,
```

```
    defaultValue: DataTypes.UUIDV4,
  },
  title: {
    type: DataTypes.STRING(250),
    allowNull: false,
  },
  body: {
    type: DataTypes.TEXT,
    allowNull: false,
  },
  images: {
    type: DataTypes.TEXT('long'),
    allowNull: true,
  },
  status: {
    type: DataTypes.ENUM,
    values: ['APPROVED', 'PENDING', 'REJECTED'],
    defaultValue: 'PENDING',
  },
  views: {
    type: DataTypes.INTEGER,
    allowNull: false,
    defaultValue: 0,
  },
  best_answer: {
    type: DataTypes.UUID,
    allowNull: true,
    defaultValue: null
  }
}, {
  db,
  tableName: 'posts',
```

```
 underscored: true,
 timestamps: true,
 indexes: [
   {
     name: 'PRIMARY',
     unique: true,
     using: 'BTREE',
     fields: [
       { name: 'id' },
     ],
   },
   {
     name: 'user_id',
     using: 'BTREE',
     fields: [
       { name: 'user_id' },
     ],
   },
 ],
});

module.exports = {
 Post,
 PostsModelSequelize,
};
```

Answers Models :

```
const { DataTypes } = require('sequelize');
const db = require('../config/db.config');
```

```
const Answer = function (answer) {
 this.body = answer.body;
 this.userId = answer.userId;
 this.postId = answer.postId;
};

const AnswersModelSequelize = db.define('answers', {
 id: {
   type: DataTypes.UUID,
   allowNull: false,
   primaryKey: true,
   defaultValue: DataTypes.UUIDV4,
 },
 body: {
   type: DataTypes.TEXT,
   allowNull: false,
 },
}, {
 db,
 tableName: 'answers',
 underscored: true,
 timestamps: true,
 indexes: [
   {
     name: 'PRIMARY',
     unique: true,
     using: 'BTREE',
     fields: [
       { name: 'id' },
     ],
   },
   {
```

```
      name: 'post_id',
      using: 'BTREE',
      fields: [
        { name: 'post_id' },
      ],
    },
    {
      name: 'user_id',
      using: 'BTREE',
      fields: [
        { name: 'user_id' },
      ],
    },
  ],
});

module.exports = { Answer, AnswersModelSequelize };
```

Comments Models :

```
const { DataTypes } = require('sequelize');
const db = require('../config/db.config');

const Comment = function (answer) {
  this.body = answer.body;
  this.userId = answer.userId;
  this.postId = answer.postId;
};

const CommentsModelSequelize = db.define('comments', {
  id: {
    type: DataTypes.UUID,
    allowNull: false,
```

```
    primaryKey: true,
    defaultValue: DataTypes.UUIDV4,
  },
  body: {
    type: DataTypes.TEXT,
    allowNull: false,
  },
}, {
  db,
  tableName: 'comments',
  underscored: true,
  timestamps: true,
  indexes: [
    {
      name: 'PRIMARY',
      unique: true,
      using: 'BTREE',
      fields: [
        { name: 'id' },
      ],
    },
    {
      name: 'post_id',
      using: 'BTREE',
      fields: [
        { name: 'post_id' },
      ],
    },
    {
      name: 'user_id',
      using: 'BTREE',
      fields: [
```

```
        { name: 'user_id' },
      ],
    },
 ],
});

module.exports = { Comment, CommentsModelSequelize };
```

Vote Models :

```
const { DataTypes } = require('sequelize');
const db = require('../config/db.config');
const { VOTES } = require('../constants');

const Vote = function (vote) {
   this.voteType = vote.voteType;
   this.userId = vote.userId;
   this.postId = vote.postId;
   this.answerId = vote.answerId;
};


const VoteModelSequelize = db.define('vote', {
 id: {
   type: DataTypes.UUID,
   allowNull: false,
   primaryKey: true,
   defaultValue: DataTypes.UUIDV4,
 },
```

```
 voteType: {
   type: DataTypes.INTEGER,
   allowNull: false,
 },
 user_id: {
   type: DataTypes.UUID,
   allowNull: false
 }
}, {
 db,
 tableName: 'vote',
 underscored: true,
 timestamps: false,
 indexes: [
   {
     name: 'PRIMARY',
     unique: true,
     using: 'BTREE',
     fields: [
       { name: 'id' },
     ],
   },
   {
     name: 'post_id',
     using: 'BTREE',
     fields: [
       { name: 'post_id' },
       // { name: 'user_id' }



     ],
   },
```

```
    {
      name: 'answer_id',
      using: 'BTREE',
      fields: [
        { name: 'answer_id' },
        // { name: 'user_id' }
      ],
    }
  ],
});

module.exports = { Vote, VoteModelSequelize };
```

Tags Models :

```
const { DataTypes } = require('sequelize');
const db = require('../config/db.config');

// constructor
// eslint-disable-next-line func-names
const Tag = function (tag) {
 this.tagname = tag.tagname;
 this.description = tag.description;
};

const TagsModelSequelize = db.define('tags', {
 id: {
   type: DataTypes.UUID,
   allowNull: false,
   primaryKey: true,
   defaultValue: DataTypes.UUIDV4,
```

```
 },
 tagname: {
   type: DataTypes.STRING(255),
   allowNull: false,
   unique: 'tagname',
 },
 description: {
   type: DataTypes.TEXT,
   allowNull: false,
 },
}, {
 db,
 tableName: 'tags',
 underscored: true,
 timestamps: true,
 indexes: [
   {
     name: 'PRIMARY',
     unique: true,
     using: 'BTREE',
     fields: [
       { name: 'id' },
     ],
   },
   {
     name: 'tagname',
     unique: true,
     using: 'BTREE',
     fields: [
       { name: 'tagname' },
     ],
   },
```

```
 ],
});

module.exports = { Tag, TagsModelSequelize };
```

A code listing your server implementation of the security and session objects

API Authentication with JWT Implementation :

```
const JWT = require('jsonwebtoken');
const config = require('../config');
const { responseHandler } = require('../helpers');

const auth = (req, res, next) => {
 const token = req.header('x-auth-token');

 // Check if no token
 if (!token) {
   return res
     .status(401)
     .json(responseHandler(false, 401, 'Sign-in required',
null));
 }

 // Verify token
 try {
   JWT.verify(token, config.JWT.SECRET, (error, decoded) => {
     if (error) {
```

```
        return res
          .status(400)
          .json(responseHandler(false, 400, 'Try again', null));
      }
      req.user = decoded.user;
      next();
    });
  } catch (err) {
    console.error(`error: ${err}`);
    return res
      .status(500)
      .json(responseHandler(false, 500, 'Server Error', null));
  }
};

module.exports = auth;
```

Routes Secured By JWT :

```
const express = require('express');
const {
 auth,
 checkOwnership
} = require('../middleware');
const { allUsers } = require('../controllers/users');

const router = express.Router();

router.use('/auth', require('./auth'));
router.use('/users', require('./users'));
router.use('/posts', require('./posts'));
router.use('/tags', require('./tags'));
```

```
router.use('/posts/answers', require('./answers'));
router.use('/posts/comments', require('./comments'));
router.use('/chat', require('./chat'));
router.use('/message', require('./message'));
router.use('/dashboard', require('./dashboard'));

router.route('/user')
 .get(auth, allUsers);

module.exports = router;
```

A code listing your main server Code

```
const express = require('express');
const morgan = require('morgan');
const helmet = require('helmet');
const cors = require('cors');
const compression = require('compression');
const http = require('http');
const xss = require('xss-clean');
const cookieParser = require('cookie-parser');
const debug = require('debug')('backend:server');

const index = require('./src/routers/index');
const portUtils = require('./src/config/port');
const connectDB = require('./src/config/db.mongo');
```

```
var sleep = require('system-sleep');

const kafkaAdminConnect = require('./src/kafka/kafkaAdmin');

// Connect mongoDB database
connectDB();

// Create express app
const app = express();


kafkaAdminConnect()

// compressing api response
app.use(compression());
// 10 mb body limit
app.use(express.json({ limit: '10mb' }));

// logger
app.use(morgan('dev'));

// Get port from environment and store in Express.
const PORT = portUtils.normalizePort(process.env.PORT ||
'5000');
app.set('port', PORT);

// cors enable
app.use(cors());

app.use(xss());

// security config
```

```
app.use(helmet());

app.use(cookieParser());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// all the api routers
app.use('/api', index);

// index setup
const server = http.createServer(app);

// Event listener for HTTP server 'listening' event.
const onListening = () => {
 const address = server.address();
 const bind = typeof address === 'string' ? `pipe ${address}` :
`port ${address.port}`;
 debug(`Server running on ${bind},
http://localhost:${address.port}`);
 console.log(`Server running on ${bind},
http://localhost:${address.port}`);
};

// Listen on provided port, on all network interfaces.
server.listen(PORT);
server.on('error', portUtils.onError);
server.on('listening', onListening);


// ------------------------------ Socker IO
------------------------ //
const io = require("socket.io")(server, {
```

```
 pingTimeout: 60000,
 cors: {
   origin: "*:*",
 },
});

io.on("connection", (socket) => {
 console.log("Connected to socket.io");
 socket.on("setup", (userData) => {
   socket.join(userData.mongoId);
   socket.emit("connected");
 });

 socket.on("join chat", (room) => {
   socket.join(room);
   console.log("User Joined Room: " + room);
 });

 socket.on("typing", (room) => socket.in(room).emit("typing"));
 socket.on("stop typing", (room) => socket.in(room).emit("stop
typing"));

 socket.on("new message", (newMessageRecieved) => {
   var chat = newMessageRecieved.chat;

   if (!chat.users) return console.log("chat.users not
defined");

   chat.users.forEach((user) => {
     if (user._id == newMessageRecieved.sender._id) return;
     socket.in(user._id).emit("message recieved",
newMessageRecieved);
```

```
    });
  });

  socket.off("setup", () => {
    console.log("USER DISCONNECTED");
    socket.leave(userData.mongoId);
  });
});
```

A code listing your database access or connection

MySQL Config Code :

```
const { Sequelize } = require('sequelize');
const dotenv = require('dotenv');

const config = require('./index');

dotenv.config();

const sequelize = new Sequelize(config.DB.DATABASE,
config.DB.USER, config.DB.PASSWORD,
 {
   dialect: 'mysql',
   host: config.DB.HOST,
```

```
  define: {
    timestamps: false,
  },
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000,
  },
 });

(async () => await sequelize.sync())();

module.exports = sequelize;
```

MongoDB Config Code :

```
const mongoose = require("mongoose");
const colors = require("colors");

const connectDB = async () => {
 console.log("Started", process.env.MONGO_URI)
 try {

  const conn = await mongoose.connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  });

  console.log(`MongoDB Connected:
```

```
${conn.connection.host}`.cyan.underline);
 } catch (error) {
   console.log(`Error: ${error.message}`.red.bold);
   process.exit();
 }
};

module.exports = connectDB;
```

A code listing your Mocha test and output results screenshots :

1.) Retrieving Users

```
var chai = require('chai');
var chaiHttp = require('chai-http');
const { expect } = chai
chai.use(chaiHttp);
describe('Mocha Testing', function() {
     var host = "http://localhost:5000";
     var path = "/api/users";
     it('Accessing Users ', function(done) {
      chai
          .request(host)
```

```
            .get(path)
            .send({myparam: 'test'})
            .end(function(error, response, body) {
             const result = response.statusCode
             expect(result).to.equal(200)
             done()
```

Results :

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL
(base) USCS-Mac303:core admin$ node_modules/.bin/mocha

  Test group
    ✔ Retrieving Users   (158ms)

  1 passing (163ms)

(base) USCS-Mac303:core admin$
```

2.) Retrieving Current Posts/Questions

```
var chaiHttp = require('chai-http');
const { expect } = chai
chai.use(chaiHttp);
describe('Mocha Testing', function() {
     var host = "http://localhost:5000";
```

```
var path = "/api/posts";
it('Accessing Users Posts ', function(done) {
 chai
     .request(host)
     .get(path)
     .send({myparam: 'test'})
     .end(function(error, response, body) {
      const result = response.statusCode
      expect(result).to.equal(200)
      done()
```

Results:

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL

(base) USCS-Mac303:core admin$ node_modules/.bin/mocha


  Test group
    ✔ Retrieving All posts  (154ms)


  1 passing (159ms)

(base) USCS-Mac303:core admin$
```

3.) Retrieving Tags :

```
var chai = require('chai');
var chaiHttp = require('chai-http');
const { expect } = chai
chai.use(chaiHttp);
describe('Mocha Testing', function() {
```

```
var host = "http://localhost:5000";
var path = "/api/tags";
it('Accessing Tags of Users  ', function(done) {
 chai
     .request(host)
     .get(path)
     .send({myparam: 'test'})
     .end(function(error, response, body) {
      const result = response.statusCode
      expect(result).to.equal(200)
      done()
```

Results :

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL
(base) USCS-Mac303:core admin$ node_modules/.bin/mocha

  Test group
    ✔ Retrieving Tags  (145ms)

  1 passing (154ms)

(base) USCS-Mac303:core admin$
```

4.) Accessing Users With credentials :

```
const chai = require('chai');
   const chaiHttp = require('chai-http');
   const should = chai.should();

   var server = "http://localhost:5000";

   chai.use(chaiHttp);
```

```
describe('Todo API', function() {

    it('Login User With JWT Token , function(done) {
        chai.request(server)
                    .post('/api/auth?x-auth')
                    // send user login details
                    .send({
                        'username': 'ravit2839',
                        'password': 'ravit2839'
                    })
                     .end((err, res) => {
        res.body.should.have.property('token');
        var token = res.body.token;
         })
    })
})
```

Results :

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL

(base) USCS-Mac303:core admin$ node_modules/.bin/mocha

  Test group
    ✔ Accessing Users With Login Credentials  (579ms)

  1 passing (585ms)

(base) USCS-Mac303:core admin$
```

5.)Accessing the question with some question :id

```
var chai = require('chai');
var chaiHttp = require('chai-http');
const { expect } = chai
chai.use(chaiHttp);
```

```
const auth="JWT
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6InJhdml0ZWphIiwiZ
W1haWwiOiJoYXBwaWVlZTIwMTZAZ21haWwuY29tIiwiaWF0IjoxNjQ3ODI1NTkwfQ.z5bu7HQb
Y-_gkdWIQFWXTGiXU4ZWc9C9UKDuQYjQ3H4";

describe('Test group', function() {
  var host = "http://localhost:5000/";
  var path = "api/posts/ed3ec119-ccbf-4cf7-a3c4-5cfdd2382b59";

  it('', function(done) {
  chai
  .request(host)
  .get(path)

  .set('content-type', 'application/x-www-form-urlencoded')

  .send({myparam: 'test'})
  .end(function(error, response, body) {
  const result = response.statusCode
  expect(result).to.equal(200)
  done()

  });
  });
  })
```

Results:

```
PS C:\Users\Ravi Shanker\Downloads\Archive (13)\core> npm test

> stackoverflowclone@1.0.0 test
> mocha --exit --recursive --timeout 10000000


  Test group
    ✓ (1003ms)


  1 passing (1s)
```

6.)Fetch all posts with specific tags

```
var chai = require('chai');
var chaiHttp = require('chai-http');
```

```
const { expect } = chai
chai.use(chaiHttp);
const auth="JWT
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6InJhdml0ZWphIiwiZ
W1haWwiOiJoYXBwaWVlZTIwMTZAZ21haWwuY29tIiwiaWF0IjoxNjQ3ODI1NTkwfQ.z5bu7HQb
Y-_gkdWIQFWXTGiXU4ZWc9C9UKDuQYjQ3H4";

describe('Test group', function() {
  var host = "http://localhost:5000/";
   var path = "api/tags/json";

  it('', function(done) {
  chai
  .request(host)
  .get(path)

  .set('content-type', 'application/x-www-form-urlencoded')

  .send({myparam: 'test'})
  .end(function(error, response, body) {
  const result = response.statusCode
  expect(result).to.equal(200)
  done()

  });
  });
  })
```

Results:

```
PS C:\Users\Ravi Shanker\Downloads\Archive (13)\core> npm test

> stackoverflowclone@1.0.0 test
> mocha --exit --recursive --timeout 10000000



  Test group
    ✓ (709ms)


  1 passing (736ms)
```

7)Fetch all tags the user(individual) has used.

```
var chai = require('chai');
var chaiHttp = require('chai-http');
const { expect } = chai
chai.use(chaiHttp);
const auth="JWT
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6InJhdml0ZWphIiwiZ
W1haWwiOiJoYXBwaWVlZTIwMTZAZ21haWwuY29tIiwiaWF0IjoxNjQ3ODI1NTkwfQ.z5bu7HQb
Y-_gkdWIQFWXTGiXU4ZWc9C9UKDuQYjQ3H4";

describe('Test group', function() {
  var host = "http://localhost:5000/";
    var path = "api/users/7c89f7eb-25a2-4e32-9dc6-a0bb383adbe2/tags";


  it('', function(done) {
  chai
  .request(host)
  .get(path)

  .set('content-type', 'application/x-www-form-urlencoded')

  .send({myparam: 'test'})
  .end(function(error, response, body) {
  const result = response.statusCode
  expect(result).to.equal(200)
  done()

  });
  });
  })
```

Result:

```
PS C:\Users\Ravi Shanker\Downloads\Archive (13)\core> npm test

> stackoverflowclone@1.0.0 test
> mocha --exit --recursive --timeout 10000000



  Test group
    ✓ (666ms)


  1 passing (678ms)
```

8)Fetch a single user details

```
var chai = require('chai');
var chaiHttp = require('chai-http');
const { expect } = chai
chai.use(chaiHttp);
const auth="JWT
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibmFtZSI6InJhdml0ZWphIiwiZ
W1haWwiOiJoYXBwaWVlZTIwMTZAZ21haWwuY29tIiwiaWF0IjoxNjQ3ODI1NTkwfQ.z5bu7HQb
Y-_gkdWIQFWXTGiXU4ZWc9C9UKDuQYjQ3H4";

describe('Test group', function() {
  var host = "http://localhost:5000/";
     var path = "api/users/7c89f7eb-25a2-4e32-9dc6-a0bb383adbe2"



  it('', function(done) {
  chai
  .request(host)
  .get(path)

  .set('content-type', 'application/x-www-form-urlencoded')

  .send({myparam: 'test'})
  .end(function(error, response, body) {
  const result = response.statusCode
  expect(result).to.equal(200)
  done()
```

```
  });
  });
  })
```

Result:

```
> stackoverflowclone@1.0.0 test
> mocha --exit --recursive --timeout 10000000


  Test group
    ✓ (725ms)


  1 passing (734ms)

PS C:\Users\Ravi Shanker\Downloads\Archive (13)\core>
```