Software Testing Project Report

By:
Soumya Chakraborty (MT2022162)
Shubham Mondal (MT2022169)

Application Description:

Recipes Server-side REST Application: This application is an API built using Node.js and Express.js, with MongoDB as the database using Mongoose for interaction. It manages recipes and users with various endpoints to perform CRUD (Create, Read, Update, Delete) operations.

Here's a breakdown of the functionality:

1. **Get All Recipes**: A GET request to the `/` endpoint fetches all recipes stored in the database.
2. **Create a New Recipe**: A POST request to the `/` endpoint allows the creation of a new recipe. It requires authentication (via `verifyToken` middleware) to ensure only authorized users can add recipes.
3. **Get a Recipe by ID**: A GET request to `/:recipeId` endpoint retrieves a specific recipe based on its unique ID.
4. **Save a Recipe**: A PUT request to the `/` endpoint allows users to save a recipe. It expects a `recipeID` and `userID` in the request body to associate a recipe with a specific user.
5. **Get IDs of Saved Recipes**: A GET request to `/savedRecipes/ids/:userId` endpoint retrieves the IDs of recipes that a specific user has saved.
6. **Get Saved Recipes**: A GET request to `/savedRecipes/:userId` endpoint fetches the actual recipes that a user has saved by using their IDs from the `user.savedRecipes` array.

The code involves routers and controllers to manage these various endpoints. It uses models for `Recipes` and `Users`, enabling operations such as creating, updating, and retrieving recipes, as well as managing user information related to saved recipes.

There are error handling mechanisms in place (sending appropriate status codes and error messages) for various scenarios where database operations or requests might fail.

Additionally, there's authentication middleware (`verifyToken`) used to protect certain routes, ensuring that only authenticated users can perform specific actions like creating or saving recipes.

This application serves as a backend API to manage recipes and user-related functionalities, providing endpoints to interact with the data stored in the MongoDB database.

Github Repo : https://github.com/sm0223/recipes

Testing Strategies used:

- Component Interaction Model (CIM): As a graph, CIM
  - Models Individual Components.
  - Combines atomic sections
  - Intra-component
- Application Transition Graph (ATG): As a graph, ATG is
  - Each node is one CIM
  - Edges are Transitions among CIMs
  - Inter-Component

Component Interaction Models:

Component Name	Service EndPoint	Atomic Sections	Code

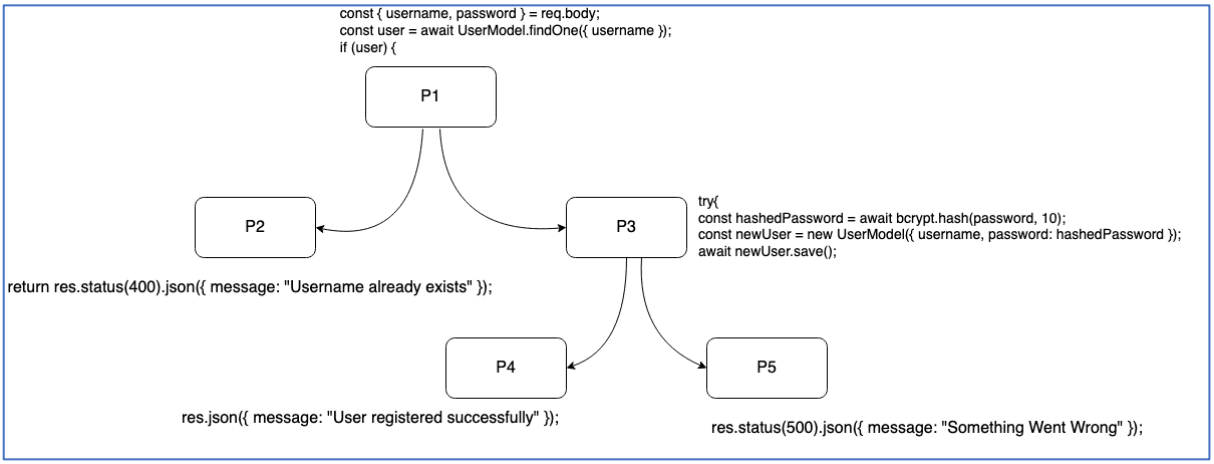
Register User	**POST: /register**	P1	const { username, password } = req.body;
			const user = await UserModel.findOne({ username });
			if (user) {

		P2	return res.status(400).json({ message: "Username already exists" });

		P3	try{
			const hashedPassword = await bcrypt.hash(password, 10);
			const newUser = new UserModel({ username, password: hashedPassword });
			await newUser.save();

		P4	res.json({ message: "User registered successfully" });
		P5	res.status(500).json({ message: "Something Went Wrong" });
		Component Expression:	(P1. (P2 | (P3. (P4|P5)))

CIM Graph:

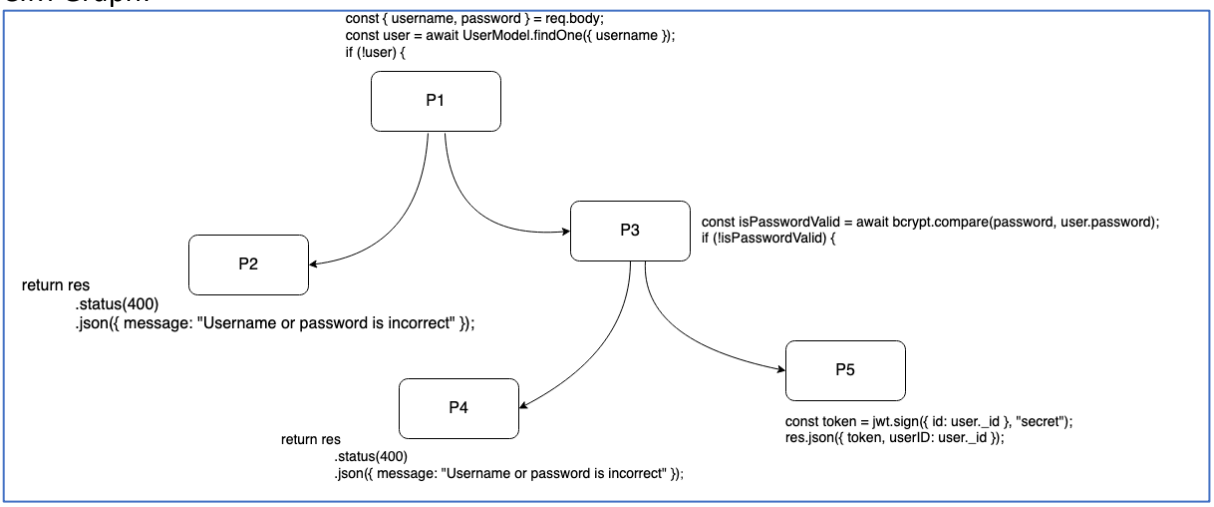


Component Name	Service EndPoint	Atomic Sections	Code

Login	**POST: /login**	P1	const { username, password } = req.body;
			const user = await UserModel.findOne({ username });
			if (!user) {

		P2	return res .status(400) .json({ message: "Username or password is incorrect" });
		P3	const isPasswordValid = await bcrypt.compare(password, user.password); if (!isPasswordValid) {
		P4	return res .status(400) .json({ message: "Username or password is incorrect" });
		P5	const token = jwt.sign({ id: user._id }, "secret"); res.json({ token, userID: user._id });
		Component Expression:	(P1. (P2 | (P3. (P4|P5)))

CIM Graph:



Component Name	Service EndPoint	Atomic Sections	Code

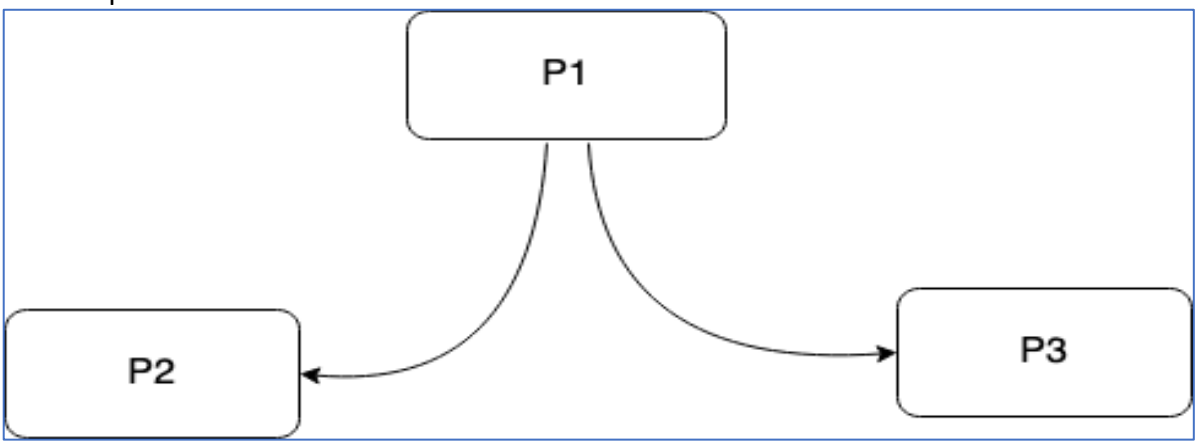
Create a New Recipe	**POST: /**	P1	const recipe = new RecipesModel({
			_id: new mongoose.Types.ObjectId(),
			name: req.body.name,
			image: req.body.image,
			ingredients: req.body.ingredients,
			instructions: req.body.instructions,
			imageUrl: req.body.imageUrl,
			cookingTime: req.body.cookingTime,
			userOwner: req.body.userOwner,
			});
			console.log(recipe);
			try {
			const result = await recipe.save();
			...

		P2	res.status(201).json({
			createdRecipe: {
			name: result.name,
			image: result.image,
			ingredients: result.ingredients,
			instructions: result.instructions,
			_id: result._id,
			},

		P3	catch (err) {
			// console.log(err);
			res.status(500).json(err);
			}

		Component Expression:	P1 . (P2 | P3)

CIM Graph:

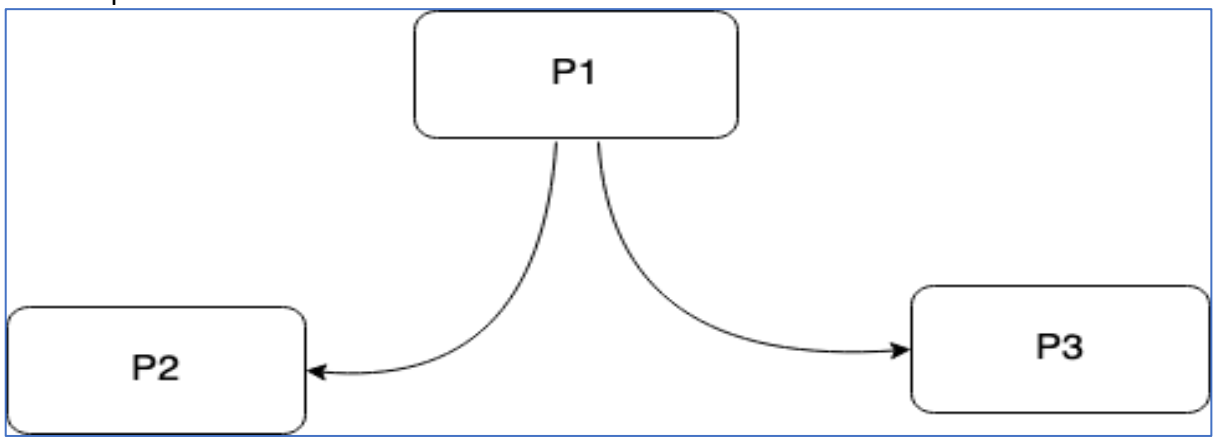


Component Name	Service EndPoint	Atomic Sections	Code
Get a Recipe	**GET: /:recipeId**	P1	try { const result = await RecipesModel.findById(req.params.recipeId);
		P2	res.status(200).json(result);

		P3	catch (err) {
			res.status(500).json(err);
			}

		Component Expression:	P1 . (P2 | P3)

CIM Graph:



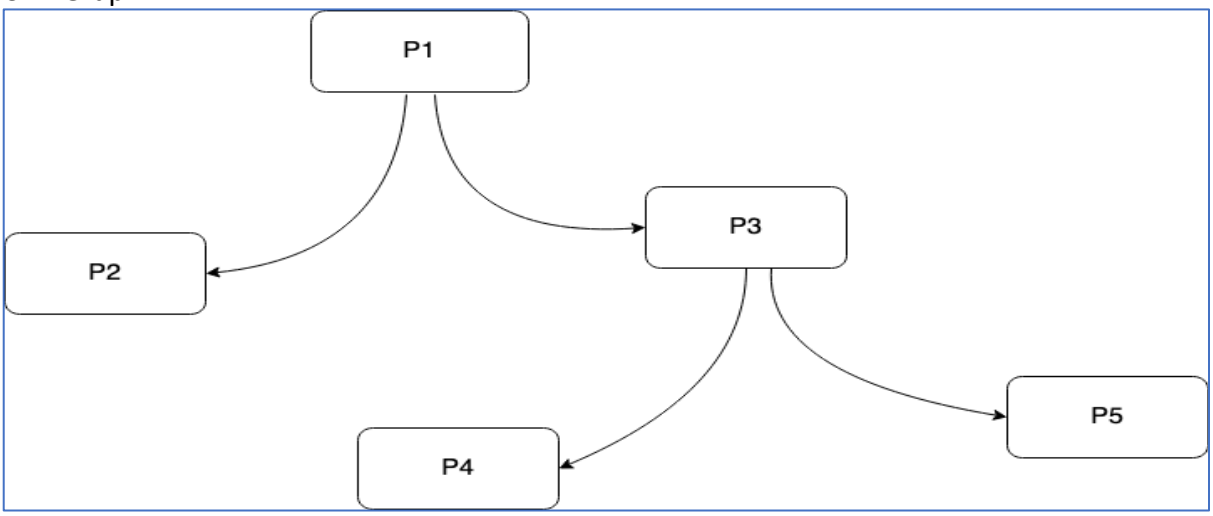
Component Name	Service EndPoint	Atomic Sections	Code

Save a Recipe	**PUT: /**	P1	try{
			const recipe = await RecipesModel.findById(req.body.recipeID);
			const user = await UserModel.findById(req.body.userID);
			}

		P2	catch (err) {
			res.status(500).json(err);
			}

		P3	user.savedRecipes.push(recipe); await user.save();
		P4	res.status(201).json({ savedRecipes: user.savedRecipes });
		P5	res.status(500).json(err);
		Component Expression:	(P1. (P2 | (P3. (P4|P5)))

CIM Graph:



Component Name	Service EndPoint	Atomic Sections	Code

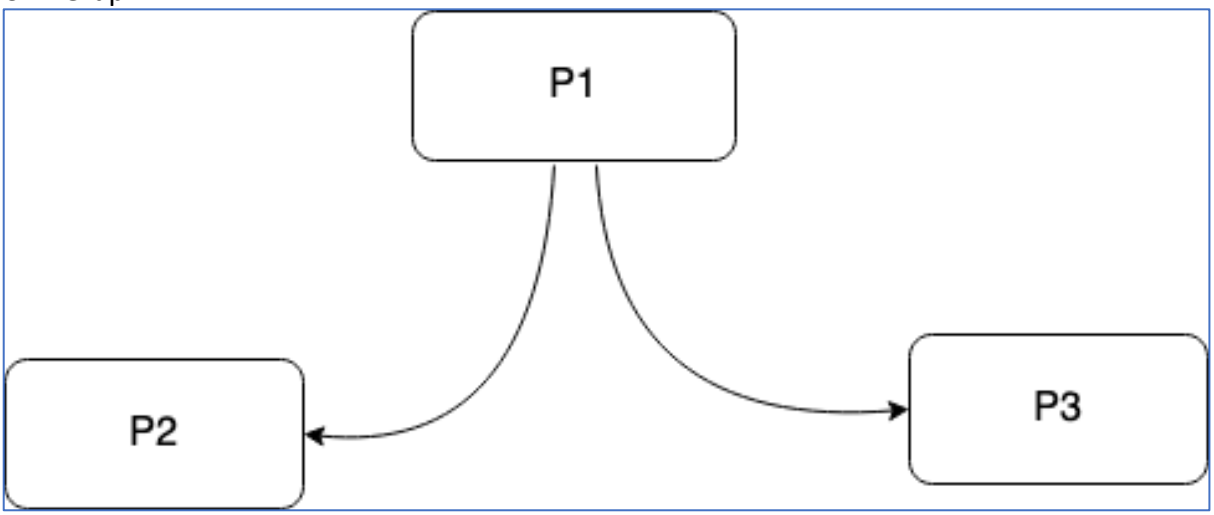
Get Saved Recipes	**GET: /savedRecipes/:userId**	P1	try {
			const user = await UserModel.findById(req.params.userId);
			const savedRecipes = await RecipesModel.find({
			_id: { $in: user.savedRecipes },
			});
			. . .

		P2	res.status(201).json({ savedRecipes });

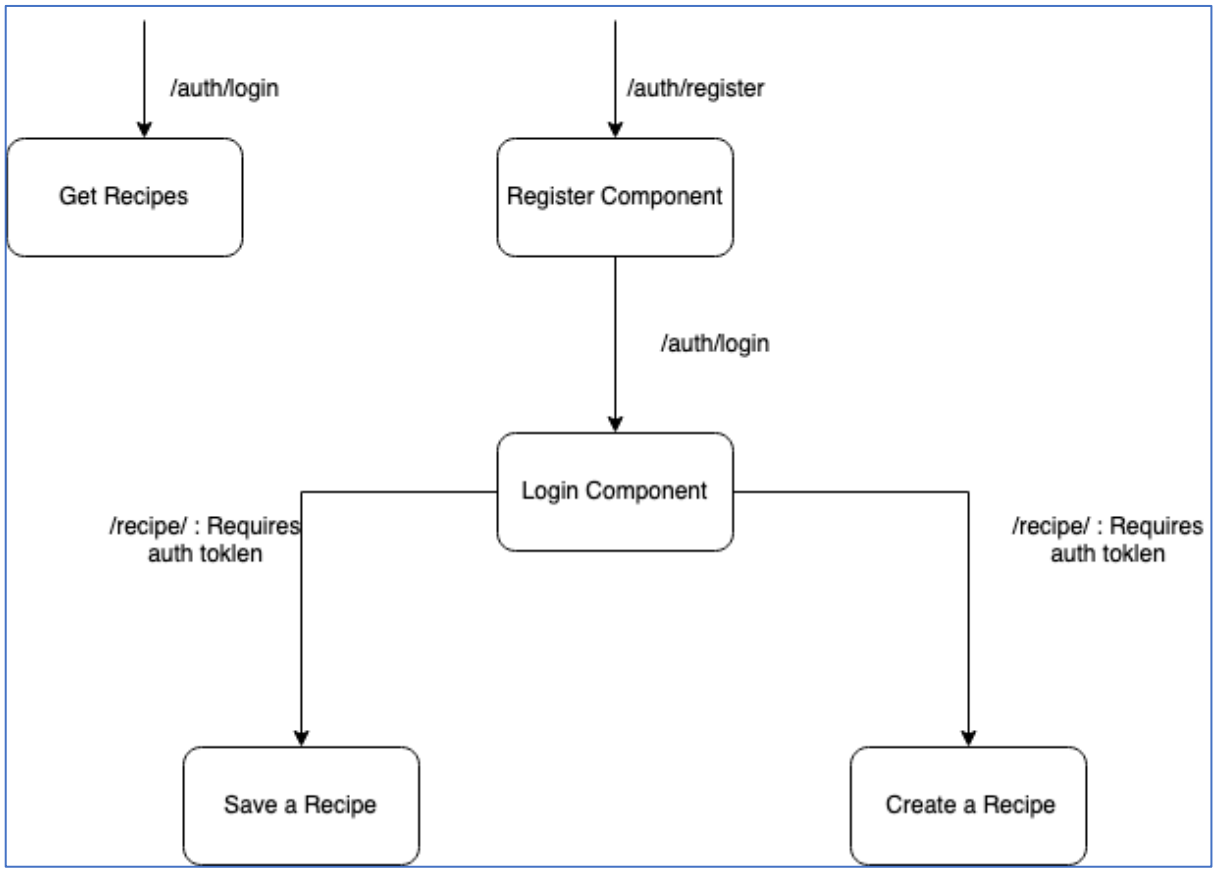
		P3	catch (err) {
			console.log(err);
			res.status(500).json(err);
			}

		Component Expression:	P1 . (P2 | P3)

CIM Graph:



Application Transition graph:



Test/test1.js:

Libraries used:

- *Mocha* : Test Framework used for Unit Testing
- *Chai* : Assertion Library used for assert statements
- *Sinon* : for Stubbing, Spying and Mocking.

```
import { expect } from 'chai';
import sinon from 'sinon';
import bcrypt from 'bcrypt';
import {UserModel} from '../src/models/Users.js';
import chai from 'chai';
import chaiHttp from 'chai-http';
import jwt from 'jsonwebtoken';
import app from '../src/index.js';
import mongoose from 'mongoose';
import { describe, it, before, after } from 'mocha';
import { RecipesModel } from '../src/models/Recipes.js';
import axios from 'axios';
chai.use(chaiHttp);
describe('TESTING FOR REGISTER COMPONENT', () => {
```

```
  let findOneStub;
  let hashStub;
  let saveStub;

  beforeEach(() => {
    findOneStub = sinon.stub(UserModel, 'findOne');
    hashStub = sinon.stub(bcrypt, 'hash');
    saveStub = sinon.stub(UserModel.prototype, 'save');
  });

  afterEach(() => {
    findOneStub.restore();
    hashStub.restore();
    saveStub.restore();
  });

  it('should return "Username already exists" if username is taken', async () => {
    findOneStub.resolves({ username: 'shux' });

    const response = await chai.request(app)
      .post('/auth/register')
      .send({ username: 'shux', password: '1234' });

    expect(response).to.have.status(400);
    expect(response.body.message).to.equal('Username already exists');
  });

  it('should return "User registered successfully" when registering a new user',
async () => {
    findOneStub.resolves(null);
    hashStub.resolves('hashedPassword');
    saveStub.resolves({ username: 'newUser', password: 'hashedPassword' });

    const response = await chai.request(app)
      .post('/auth/register')
      .send({ username: 'newUser', password: 'somePassword' });

    expect(response).to.have.status(200);
    expect(response.body.message).to.equal('User registered successfully');
  });

  it('should return "Something Went Wrong" if an error occurs during registration',
async () => {
    findOneStub.rejects(new Error('Database error'));

    const response = await chai.request(app)
      .post('/auth/register')
      .send({ username: 'newUser', password: 'somePassword' });

    expect(response).to.have.status(500);
    expect(response.body.message).to.equal('Something Went Wrong');
```

```
  });
});



describe('TESTING FOR LOGIN COMPONENT', () => {
    let findOneStub;
    let compareStub;
    let signStub;

    beforeEach(() => {
      findOneStub = sinon.stub(UserModel, 'findOne');
      compareStub = sinon.stub(bcrypt, 'compare');
      signStub = sinon.stub(jwt, 'sign');
    });

    afterEach(() => {
      findOneStub.restore();
      compareStub.restore();
      signStub.restore();
    });

    it('should return "Username or password is incorrect" if username is not
found', async () => {
      findOneStub.resolves(null);

      const response = await chai.request(app)
        .post('/auth/login')
        .send({ username: 'shun', password: '1234' });

      expect(response).to.have.status(400);
      expect(response.body.message).to.equal('Username or password is incorrect');
    });

    it('should return "Username or password is incorrect" if password is invalid',
async () => {
      findOneStub.resolves({ username: 'shux', password: '1234' });
      compareStub.resolves(false);

      const response = await chai.request(app)
        .post('/auth/login')
        .send({ username: 'shux', password: '1345' });

      expect(response).to.have.status(400);
      expect(response.body.message).to.equal('Username or password is incorrect');
    });

    it('should return a token and userID on successful login', async () => {
      const mockUser = { _id: 'user_id', username: 'shux', password: '1234' };
      findOneStub.resolves(mockUser);
      compareStub.resolves(true);
```

```
      signStub.returns('mockToken');

      const response = await chai.request(app)
        .post('/auth/login')
        .send({ username: 'shux', password: '1234' });

      expect(response).to.have.status(200);
      expect(response.body).to.have.property('token', 'mockToken');
      expect(response.body).to.have.property('userID', 'user_id');
    });
  });

const testDBUrl =
"mongodb+srv://shux:ayantika@cluster0.bkaw5xv.mongodb.net/recipetest?retryWrites=tr
ue&w=majority"

describe('TESTING FOR RECIPES COMPONENT', () => {
  before(async () => {
    // Connect to the test database
    try {
      // console.log('Connecting to the database...');
      await mongoose.connect(testDBUrl, { useNewUrlParser: true,
useUnifiedTopology: true });
      console.log('Database connection successful.');
    } catch (error) {
      console.error('Database connection failed:', error);
      throw error; // Rethrow the error to fail the test setup
    }
  });

  after(async () => {
    // Disconnect from the test database after all tests are completed
    await mongoose.connection.close();
  });

  describe('GET /recipes', () => {
    it('should get all recipes', async () => {
      const response = await chai.request(app).get('/recipes');
      expect(response).to.have.status(200);
      expect(response.body).to.be.an('array');
    }).timeout(5000);
  });

  describe('POST /recipes', () => {
    it('should throw 401 unauthorized while creating a new recipe without logging
in', async () => {
      const newRecipe = {
        name: 'Test Recipe',
        ingredients: [
          "Test Ingredient1",
          "Test Ingredient2",
```

```
        ],
        instructions:"Test Instruction",
        imageUrl: "https://kitchenofdebjani.com/wp-content/uploads/2022/09/Dak-
Bungalow-Chicken-Curry-recipe-debjanir-rannaghar.jpg",
        cookingTime: 200
      };

      const response = await chai.request(app).post('/recipes').send(newRecipe);
      expect(response).to.have.status(401);
      // Add more assertions based on your response structure
    }).timeout(5000);

    it('should create a new recipe', async () => {

      const response = await chai.request(app)
        .post('/auth/login')
        .send({ username: 'shux', password: '1234' });
        const req = {
          body: {
            name: 'Test Recipe',
            ingredients: [
              "Test Ingredient1",
              "Test Ingredient2",
            ],
            instructions:"Test Instruction",
            imageUrl: "https://kitchenofdebjani.com/wp-content/uploads/2022/09/Dak-
Bungalow-Chicken-Curry-recipe-debjanir-rannaghar.jpg",
            cookingTime: 200,
            userOwner: "65643a52e2f97f6a29bf6083"
          }
        };
      const res = await chai.request(app)
        .post('/recipes')
        .set('Authorization', response._body.token) // Attach a mock token
        .send(req.body);

      // Assertions
      expect(res).to.have.status(201);
      expect(res.body.createdRecipe).to.have.property('name', 'Test Recipe');
      expect(res.body.createdRecipe).to.have.property('_id');
    }).timeout(5000);
  });

  describe('GET /recipes/:recipeId', () => {
    it('should get a recipe by ID', async () => {
      const existingRecipe = await RecipesModel.findOne(); // Assuming there is at
least one recipe in the database

      const response = await
chai.request(app).get(`/recipes/${existingRecipe._id}`);
      expect(response).to.have.status(200);
```

```
      expect(response.body).to.have.property('_id', existingRecipe._id.toString());
      // Add more assertions based on your response structure
    }).timeout(5000);
  });

  // Add similar tests for other routes (e.g., PUT, GET saved recipes, etc.)
});
```

```
soumy@Soumyas-MacBook-Air server % npm test

> server@1.0.0 test
> mocha

(node:79623) [MONGOOSE] DeprecationWarning: Mongoose: the `strictQuery` option will be
(Use `node --trace-deprecation ...` to show where the warning was created)
Server started


  TESTING FOR REGISTER COMPONENT
    ✔ should return "Username already exists" if username is taken (44ms)
    ✔ should return "User registered successfully" when registering a new user
    1) should return "Something Went Wrong" if an error occurs during registration

  TESTING FOR LOGIN COMPONENT
    ✔ should return "Username or password is incorrect" if username is not found
    ✔ should return "Username or password is incorrect" if password is invalid
    ✔ should return a token and userID on successful login

  TESTING FOR RECIPES COMPONENT
Database connection successful.
    GET /recipes
      ✔ should get all recipes (1805ms)
    POST /recipes
      ✔ should throw 401 unauthorized while creating a new recipe without logging in
      ✔ should create a new recipe (1322ms)
    GET /recipes/:recipeId
      ✔ should get a recipe by ID (524ms)


  9 passing (7s)
  1 failing
```

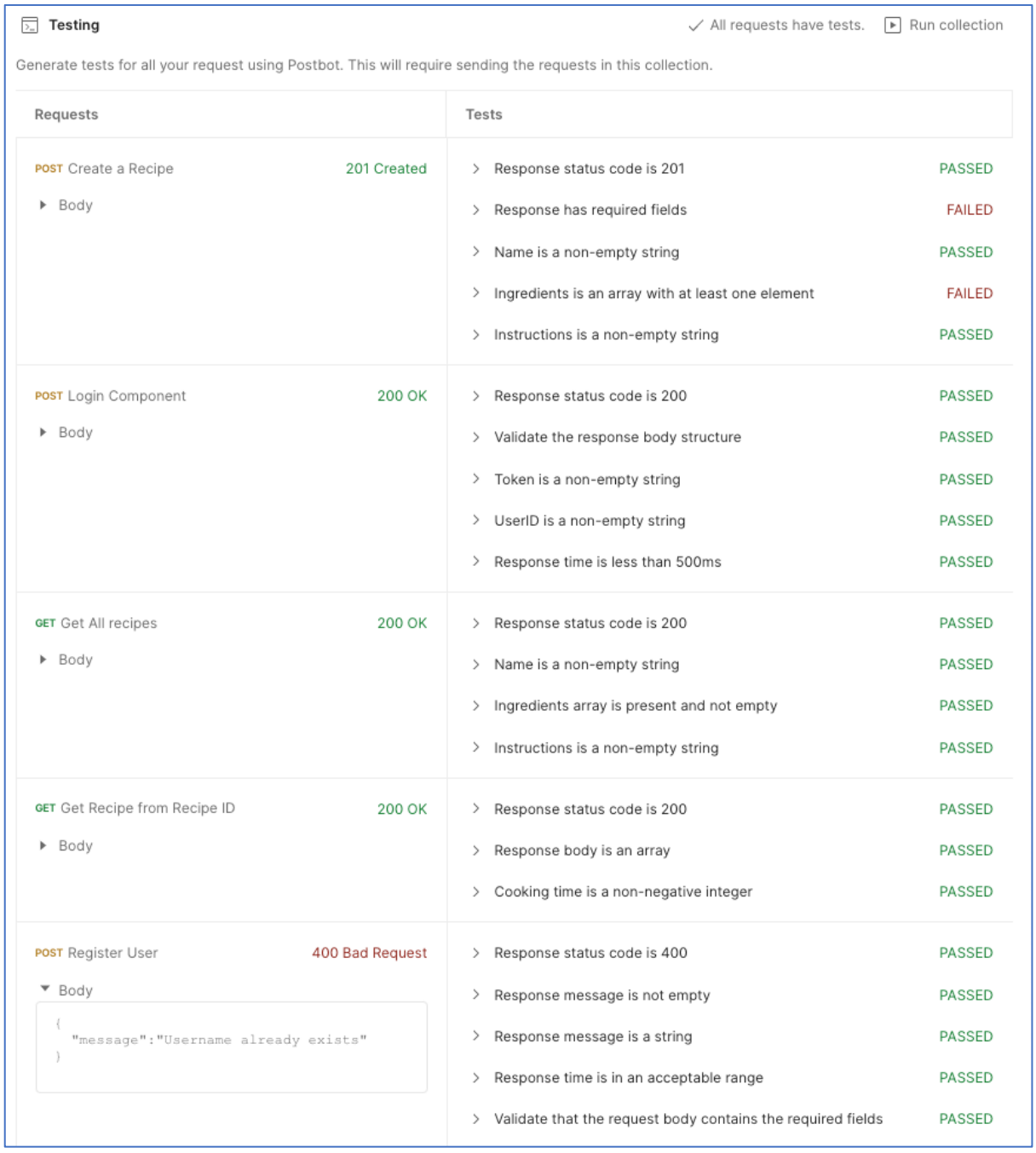
API Testing Using Postman:

```
Postman Tests

POST
Create a Recipe
201 Created

Body

Response  code is 201



pm.test('Response  code is 201', function () {
    pm.response.to.have.(201);
})
Response has required fields



pm.test('Response has required fields', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.name).to.exist.and.to.be.a('string');
    pm.expect(responseData.ingredients).to.exist.and.to.be.an('array');
    pm.expect(responseData.instructions).to.exist.and.to.be.a('string');
    pm.expect(responseData._id).to.exist.and.to.be.a('string');
})
Name is a non-empty string



pm.test('Name is a non-empty string', function () {
    const responseData = pm.response.json();

pm.expect(responseData.createdRecipe.name).to.exist.and.to.be.a('string').and.to.ha
ve.lengthOf.at.least(1, 'Name should not be empty');
})
Ingredients is an array with at least one element



pm.test('Ingredients is an array with at least one element', function () {
    const responseData = pm.response.json();

pm.expect(responseData.ingredients).to.be.an('array').and.to.have.lengthOf.at.least
(1);
})
Instructions is a non-empty string
```

```
pm.test('Instructions is a non-empty string', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');

pm.expect(responseData.createdRecipe.instructions).to.be.a('string').and.to.have.le
ngthOf.at.least(1, 'Value should not be empty');
})
POST
Login Component
200 OK

Body

Response  code is 200



pm.test('Response  code is 200', function () {
    pm.response.to.have.(200);
})
Validate the response body structure



pm.test('Validate the response body structure', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.token).to.exist.and.to.be.a('string');
    pm.expect(responseData.userID).to.exist.and.to.be.a('string');
})
Token is a non-empty string



pm.test('Token is a non-empty string', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');

pm.expect(responseData.token).to.be.a('string').and.to.have.lengthOf.at.least(1,
'Token should not be empty');
})
UserID is a non-empty string



pm.test('UserID is a non-empty string', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('object');
```

```
pm.expect(responseData.userID).to.be.a('string').and.to.have.lengthOf.at.least(1,
'UserID should not be empty');
})
Response time is less than 500ms



pm.test('Response time is less than 500ms', function () {
    pm.expect(pm.response.responseTime).to.be.below(500);
})
GET
Get All recipes
200 OK

Body

Response  code is 200



pm.test('Response  code is 200', function () {
    pm.response.to.have.(200);
})
Name is a non-empty string



pm.test('Name is a non-empty string', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('array');
    responseData.forEach(function (recipe) {
        pm.expect(recipe.name).to.be.a('string').and.to.have.lengthOf.at.least(1,
'Name should not be empty');
    });
})
Ingredients array is present and not empty



pm.test('Ingredients array is present and not empty', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('array');
    responseData.forEach(function (recipe) {
        pm.expect(recipe.ingredients).to.exist;
        pm.expect(recipe.ingredients).to.be.an('array').that.is.not.empty;
    });
})
Instructions is a non-empty string
```

```
pm.test('Instructions is a non-empty string', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('array');
    responseData.forEach(function (recipe) {

pm.expect(recipe.instructions).to.be.a('string').and.to.have.lengthOf.at.least(1,
'Instructions should not be empty');
    });
})
GET
Get Recipe from Recipe ID
200 OK

Body

Response  code is 200



pm.test('Response  code is 200', function () {
    pm.response.to.have.(200);
})
Response body is an array



pm.test('Response body is an array', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('array');
})
Cooking time is a non-negative integer



pm.test('Cooking time is a non-negative integer', function () {
    const responseData = pm.response.json();
    pm.expect(responseData).to.be.an('array').that.is.not.empty;
    responseData.forEach(function (recipe) {
        pm.expect(recipe.cookingTime).to.be.a('number').and.to.be.at.least(0);
    });
})
POST
Register User
400 Bad Request

Body

{
  "message":"Username already exists"
}
Response  code is 400
```



Contributions:

Name	Contribution
Shubham Mondal	☐ Recipes Module Dev and Test ☐ API Testing
Soumya Chakraborty	☐ Login and Registration module Dev and Test ☐ API Testing