Report

for

‘Connecta’

An online social networking system

ANKUSH NIROULA	**(CS23BTKMU11002)**
ANISH MANANDHAR	**(CS23BTKMU11001)**
VIGNAN KOTA	**(CS21BTECH11029)**
DAVID MALOTH	**(CS21BTECH11035)**

Class of Use Cases	**Use Cases**	**Completed**
Use Cases related to user registration and authentication	Create	✔
	Register	✔
	Login	✔
	Reset password	
Use Cases related to post management	View feed	✔
	Like post	✔
	Create post	✔
	Delete post	
User Cases related to search functionality	Perform search	✔
	View recent searches	✔
	Delete recent search	
Use Cases related to connection management	View user profile	✔
	Send connection request	✔
	View connection requests	
	Accept connection request	
	Remove connection	
User Cases related to profile management	View profile	✔
	Edit Profile	
	Delete profile	

Group Division

Name	Task
Anish Manandhar	Routing, UI Design, Functionality Implementation,Jest Test Case
Ankush Niroula	Rest API , Authentication, UI Design

Methodology

Using JavaScript to its full potential from the server to the client, the MERN stack provides a complete solution for full-stack web development. The Connecta depends upon the MERN Stack which includes:

1. MongoDB: This NoSQL database can handle big and complicated data structures because of its scalability and flexibility. Because of its document-oriented design, data manipulation is simple and easily integrated with JavaScript.
2. Express.js: Express.js and Node.js are the foundation of the MERN stack's server-side architecture. Express.js offers a simple and adaptable framework for managing HTTP requests and routes, making it easier to create reliable and effective web applications.
3. React.js: React.js introduced a component-based design that made it easy for us to create dynamic user interfaces, front-end development. By reducing the requirement for direct HTML DOM manipulation, its virtual DOM (Document Object Model) assures optimal performance, leading to faster rendering and an improved user experience.
4. Node.js: Node.js provides high-performance and non-blocking I/O operations and is used as the server-side runtime environment for the MERN stack. Its event-driven architecture makes it perfect for developing real-time applications and APIs since it facilitates the effective handling of several requests simultaneously.

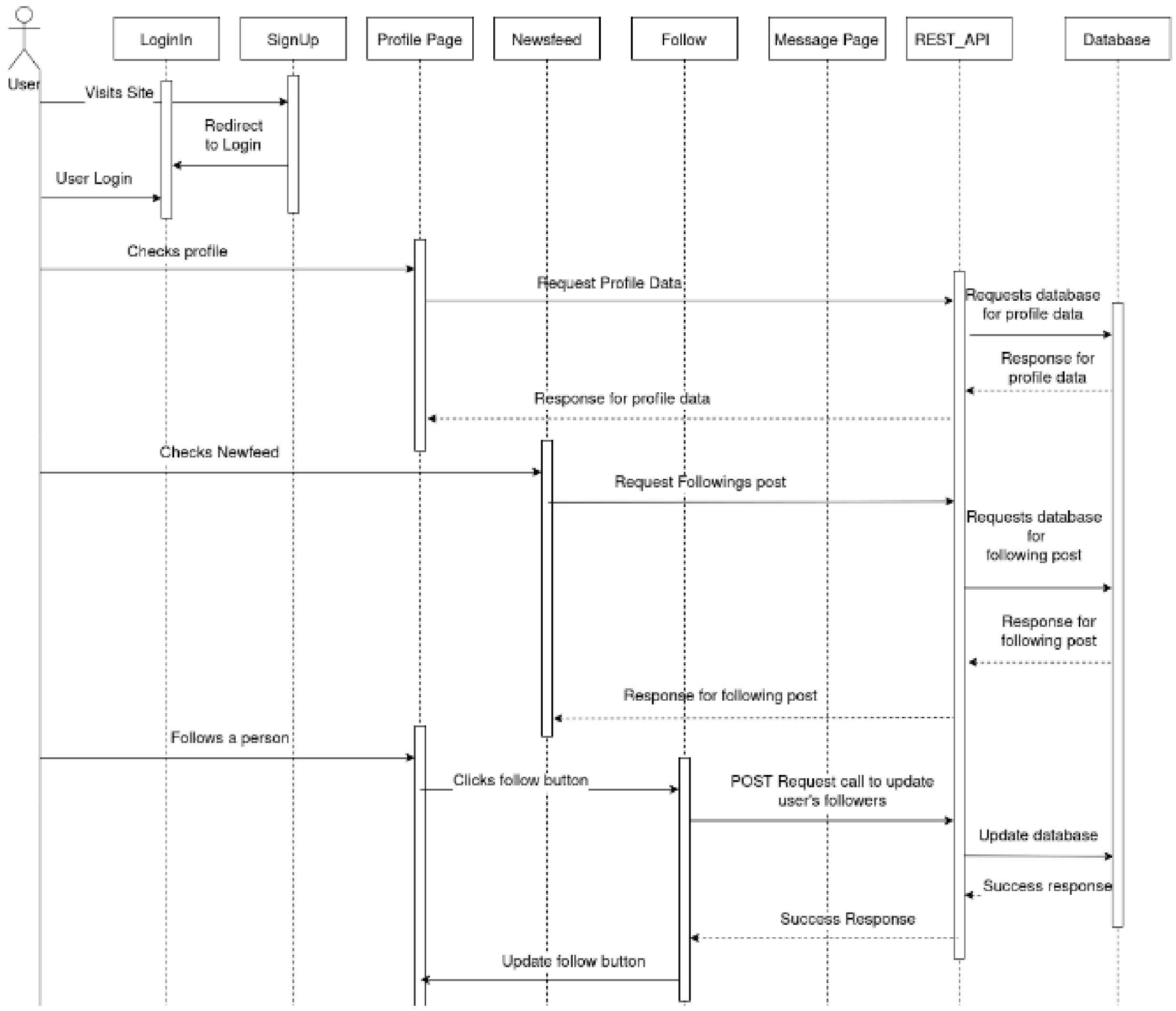


Fig : Sequence Diagram Of Connecta

A sequence diagram is an interaction diagram that shows an object's interaction promptly. We have eight different objects in our system i.e. login, signup, profile page, newsfeed, follow, message page, REST API, and Database.

The first time a user visits our site, they are redirected to the signup page and then the login page, where they can create and log in to their accounts. All users must have an account. Users can then check for their profile data which shows the user's profile briefly. Then, the newsfeed page shows the posts of the ones they have followed. The posts can be liked or disliked. Most importantly, REST API is used for the easy access of database through different requests i.e. GET, POST, UPDATE, and DELETE.

Use Cases

- Login
- Registration of The User
- Hashing Of The Password
- Feed Management
- Post Management
- React To The Posts
- Profile Management

Schema Of The Database

1. User profile

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

const UserSchema = new mongoose.Schema(
  {
    username: {
      type: String,
      require: true,
      min: 3,
      max: 20,
      unique: true,
    },
    email: {
      type: String,
      required: true,
      max: 50,
      unique: true,
    },
    password: {
      type: String,
      required: true,
      min: 6,
    },
    profilePicture: {
      type: String,
      default: "",
    },
    coverPicture: {
      type: String,
      default: "",
    },
    followers: {
      type: Array,
      default: [],
    },
    followings: {
      type: Array,
      default: [],
    },

    desc: {
      type: String,
      max: 50,
    },
   location: {
      type: String,
      max: 50,
    }    relationship: {
      type: Number,
      enum: [1, 2, 3],
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("User", UserSchema);
```

2. Post Schema

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

const PostSchema = new mongoose.Schema(
  {
    userId: {
      type: String,
      required: true,
    },
    desc: {
      type: String,
      max: 500,
    },
    image: {
      type: String,
    },
  reacts: {
      type: Array,
      default: [],
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Post", PostSchema);
```

Modules

Modules	LOC
Registration	102
Login	15
Posts	102
Profile Management	85
Dummy Data	139
Following User	19
Unfollow User	20
Getting All The Users	10
ApiCalls	11
AuthContext	39
Feed	37
Home	18
Client Login	66
Messenger	159

Total Lines Of Code = 1200

Test Cases

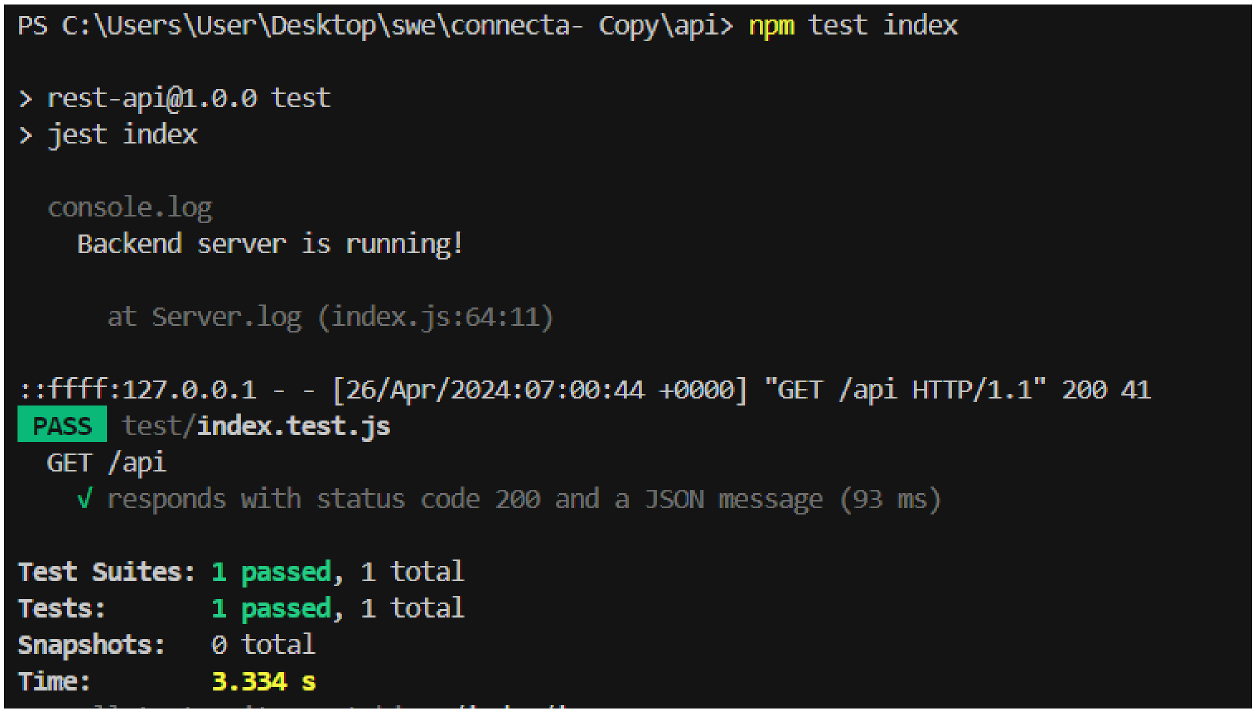
Jest is a popular JavaScript testing framework maintained by Facebook. It's widely used for testing JavaScript code, particularly in projects built with tools like React, Vue.js, Angular, and Node.js. Jest provides a simple and powerful way to write and execute test cases, ensuring your code's reliability and correctness. Jest has been used to develop the unit test cases for the Connecta.
Jest test case typically involves the following steps:

1. Setup: Defining the initial state or conditions necessary for the test. This includes setting up mock data, configuring dependencies, or preparing the environment.
2. Execution: This includes executing the specific function or code snippet users want to see. This involves calling the function or triggering the code behaviour under test.
3. Assertion: Verify the expected outcome or behaviour of the code being tested. Jest provides a rich set of assertion functions to make assertions about the results of the code execution.
4. Teardown: Optionally, clean up any resources or reset the environment after the test. This ensures that subsequent tests start with a clean slate.

Unit Test Case for the RestApi

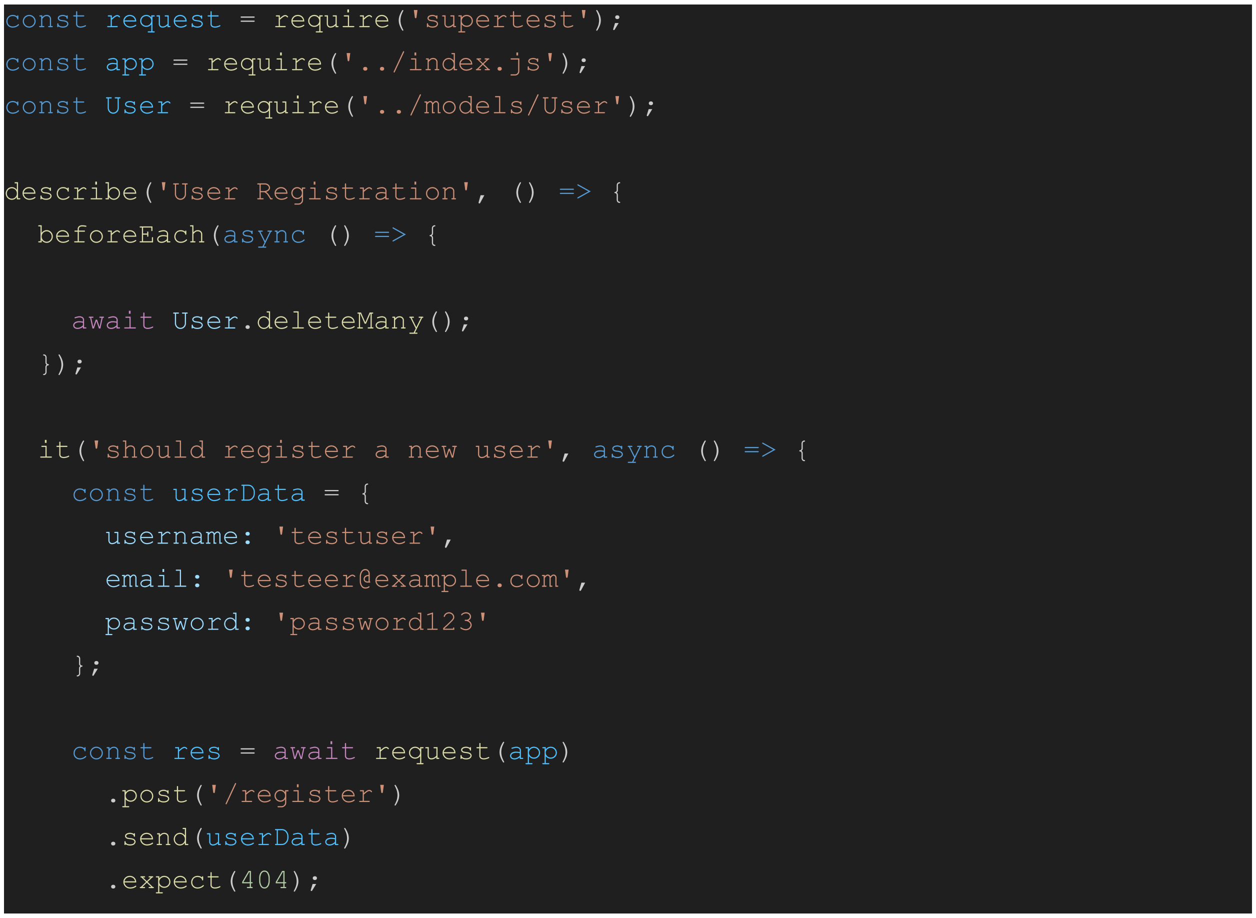
```
const app = require('../index.js');

describe('GET /api', () => {
  it('responds with status code 200 and a JSON message', async () => {
    const response = await request(app).get('/api');

    expect(response.status).toBe(200);
    expect(response.body).toEqual({
      msg: "Status code: 200, working good!"
    });
  });
});
```

Result : Passed



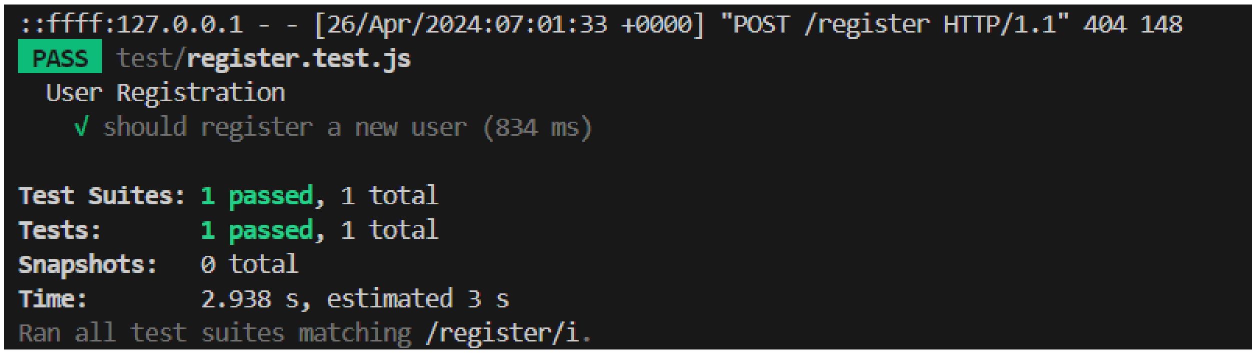
```
PS C:\Users\User\Desktop\swe\connecta- Copy\api> npm test index

> rest-api@1.0.0 test
> jest index

  console.log
    Backend server is running!

      at Server.log (index.js:64:11)

::ffff:127.0.0.1 - - [26/Apr/2024:07:00:44 +0000] "GET /api HTTP/1.1" 200 41
PASS  test/index.test.js
  GET /api
    √ responds with status code 200 and a JSON message (93 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        3.334 s
```

Test Case For the Registration



```
const request = require('supertest');
const app = require('../index.js');
const User = require('../models/User');

describe('User Registration', () => {
  beforeEach(async () => {

    await User.deleteMany();
  });

  it('should register a new user', async () => {
    const userData = {
      username: 'testuser',
      email: 'testeer@example.com',
      password: 'password123'
    };

    const res = await request(app)
      .post('/register')
      .send(userData)
      .expect(404);
  })})
```

Result: Passed

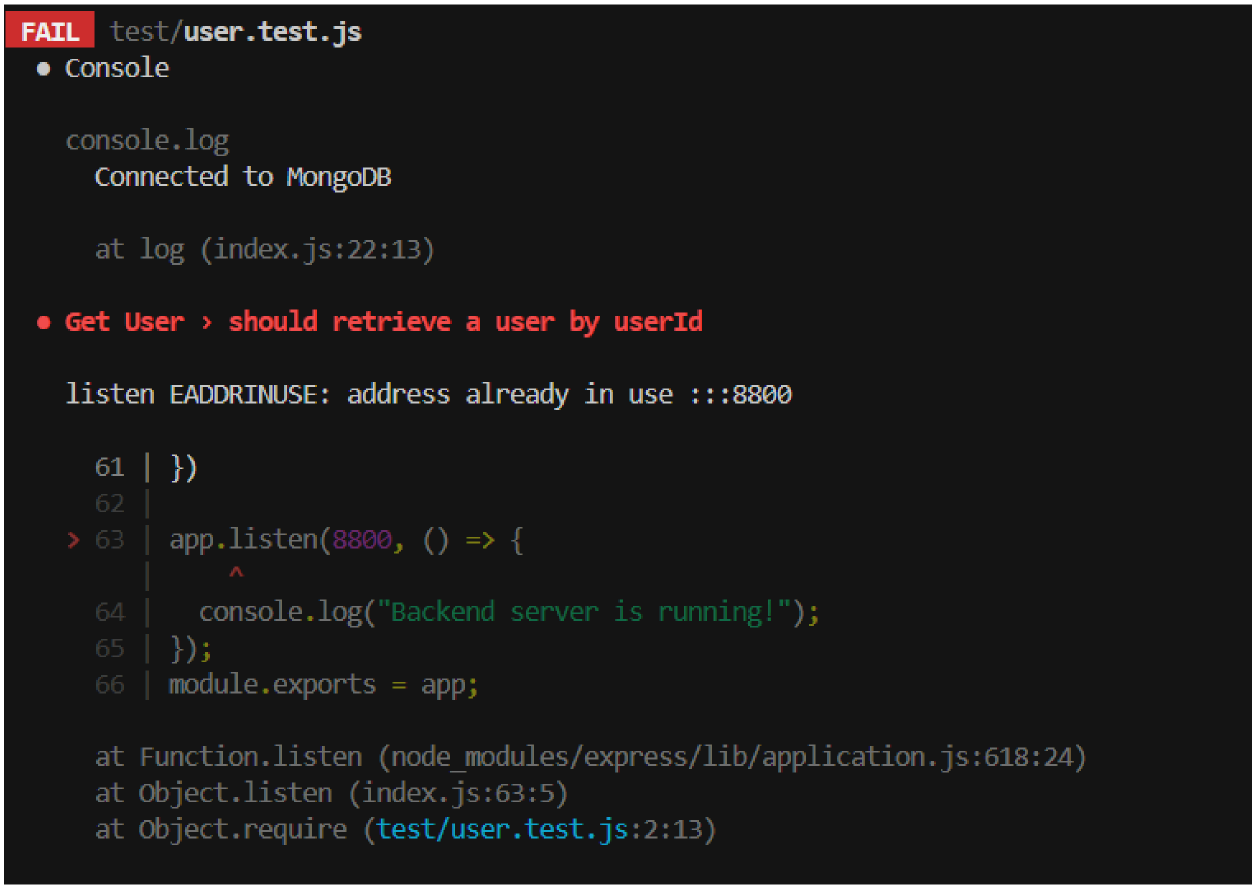


```
::ffff:127.0.0.1 - - [26/Apr/2024:07:01:33 +0000] "POST /register HTTP/1.1" 404 148
 PASS  test/register.test.js
  User Registration
    √ should register a new user (834 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        2.938 s, estimated 3 s
Ran all test suites matching /register/i.
```

Unit Test Case for getting the user

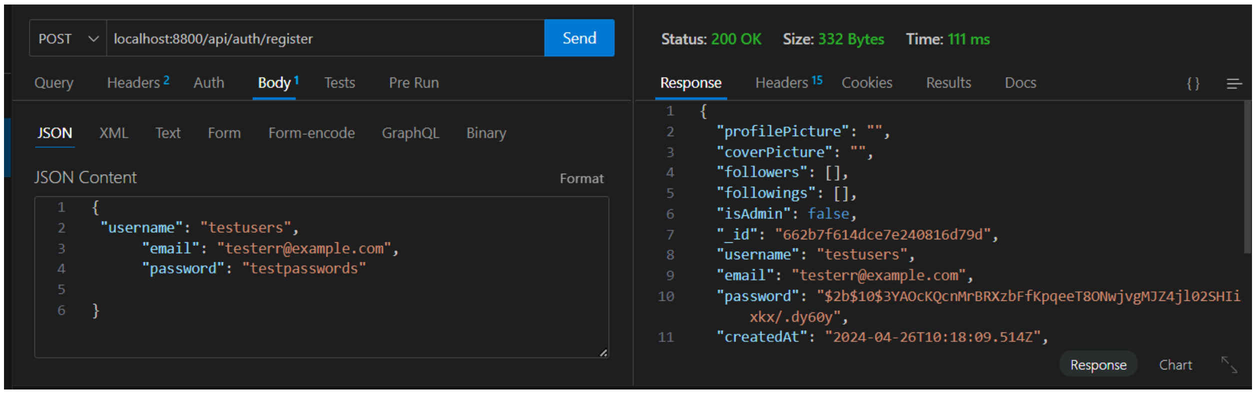
```
const request = require('supertest');
const app = require('../index');
const User = require('../models/User');

describe('Get User', () => {

  it('should retrieve a user by userId', async () => {
    // Create a test user
    const newUser = await User.create({
      username: 'testuser',
      email: 'test@example.com',
      password: 'password123'
    });

    const res = await request(app)
      .get('/users')
      .query({ userId: newUser._id })
      .expect(404);


  });


});
```

Result: Failed



```
FAIL  test/user.test.js
 ● Console

    console.log
      Connected to MongoDB

      at log (index.js:22:13)

 ● Get User › should retrieve a user by userId

    listen EADDRINUSE: address already in use :::8800

      61 | })
      62 |
    > 63 | app.listen(8800, () => {
         |     ^
      64 |   console.log("Backend server is running!");
      65 | });
      66 | module.exports = app;

      at Function.listen (node_modules/express/lib/application.js:618:24)
      at Object.listen (index.js:63:5)
      at Object.require (test/user.test.js:2:13)
```

Performance Test Case
API Test Case
API Test For the Registration



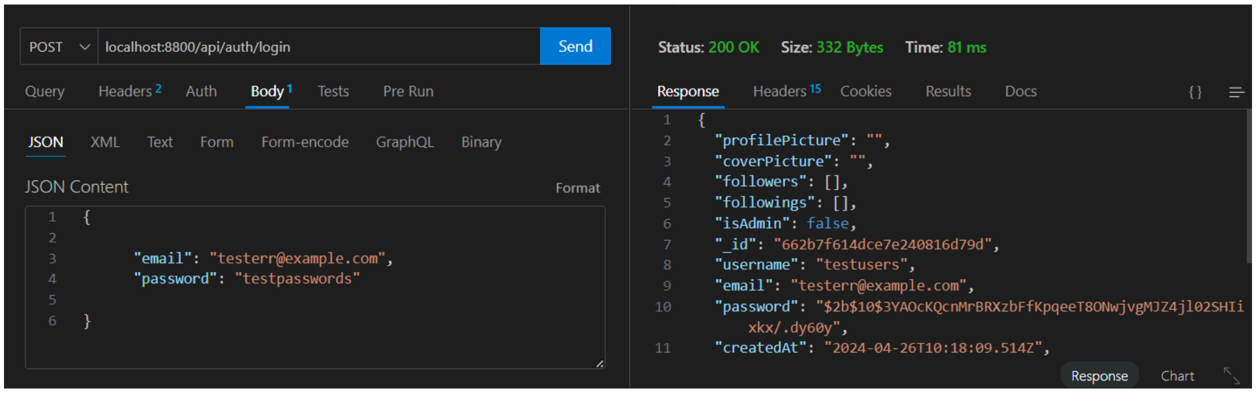
localhost:8800/api/auth/register

```
{
 "username": "testusers",
      "email": "testerr@example.com",
      "password": "testpasswords"

}
```

Status: 200 OK   Size: 332 Bytes   Time: 111 ms

```
{
  "profilePicture": "",
  "coverPicture": "",
  "followers": [],
  "followings": [],
  "isAdmin": false,
  "_id": "662b7f614dce7e240816d79d",
  "username": "testusers",
  "email": "testerr@example.com",
  "password": "$2b$10$3YAOcKQcnMrBRXzbFfKpqeeT8ONwjvgMJZ4jl02SHIi
      xkx/.dy60y",
  "createdAt": "2024-04-26T10:18:09.514Z",
```

API Test Case For Login


POST localhost:8800/api/auth/login Send

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
{

        "email": "testerr@example.com",
        "password": "testpasswords"

}
```

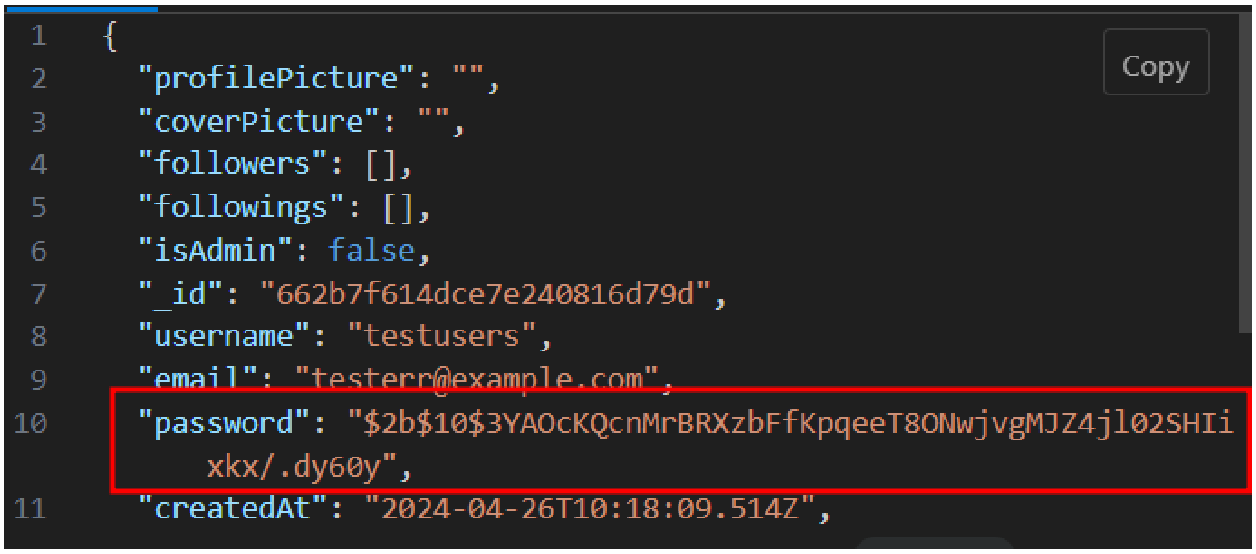
Status: 200 OK Size: 332 Bytes Time: 81 ms

Response Headers 15 Cookies Results Docs

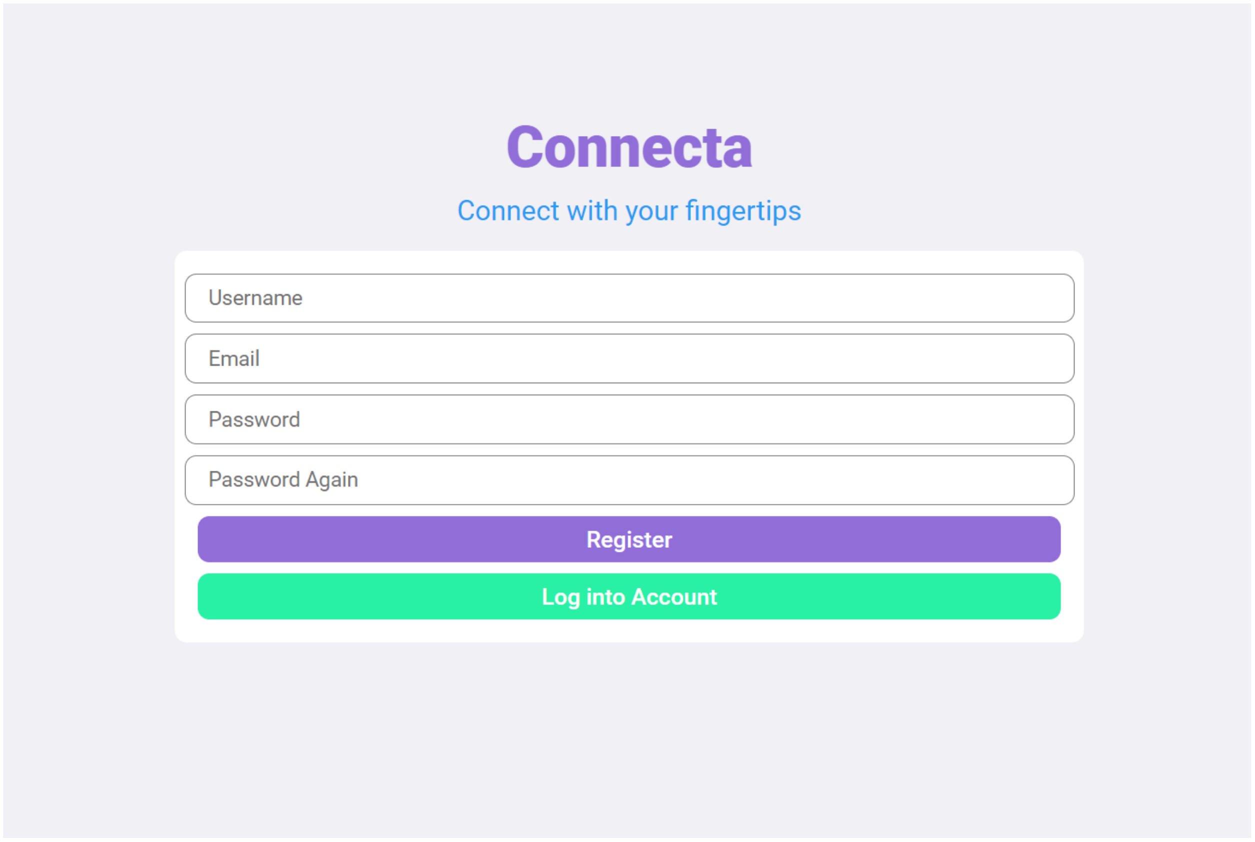
```
{
  "profilePicture": "",
  "coverPicture": "",
  "followers": [],
  "followings": [],
  "isAdmin": false,
  "_id": "662b7f614dce7e240816d79d",
  "username": "testusers",
  "email": "testerr@example.com",
  "password": "$2b$10$3YAOcKQcnMrBRXzbFfKpqeeT8ONwjvgMJZ4jl02SHIi
      xkx/.dy60y",
  "createdAt": "2024-04-26T10:18:09.514Z",
```

Response Chart

Test For Hashing Of The Password


```
{
  "profilePicture": "",
  "coverPicture": "",
  "followers": [],
  "followings": [],
  "isAdmin": false,
  "_id": "662b7f614dce7e240816d79d",
  "username": "testusers",
  "email": "testerr@example.com",
  "password": "$2b$10$3YAOcKQcnMrBRXzbFfKpqeeT8ONwjvgMJZ4jl02SHIi
      xkx/.dy60y",
  "createdAt": "2024-04-26T10:18:09.514Z",
```

Appendix



Connecta

Connect with your fingertips

Username

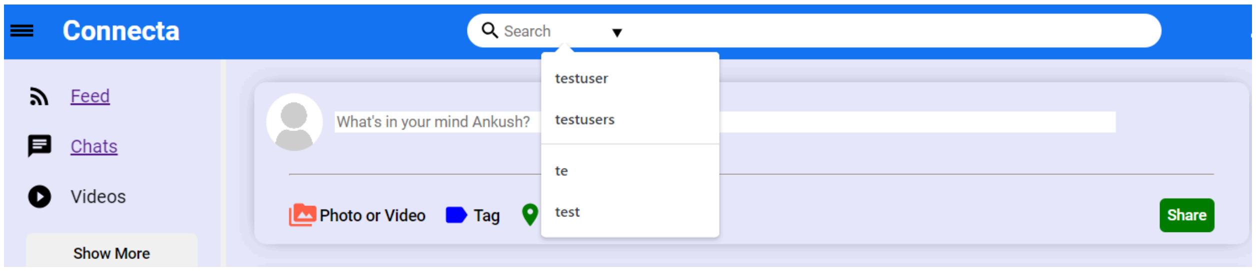
Email

Password

Password Again

Register

Log into Account



Connecta

Search

testuser

testusers

te

test

Feed

Chats

Videos

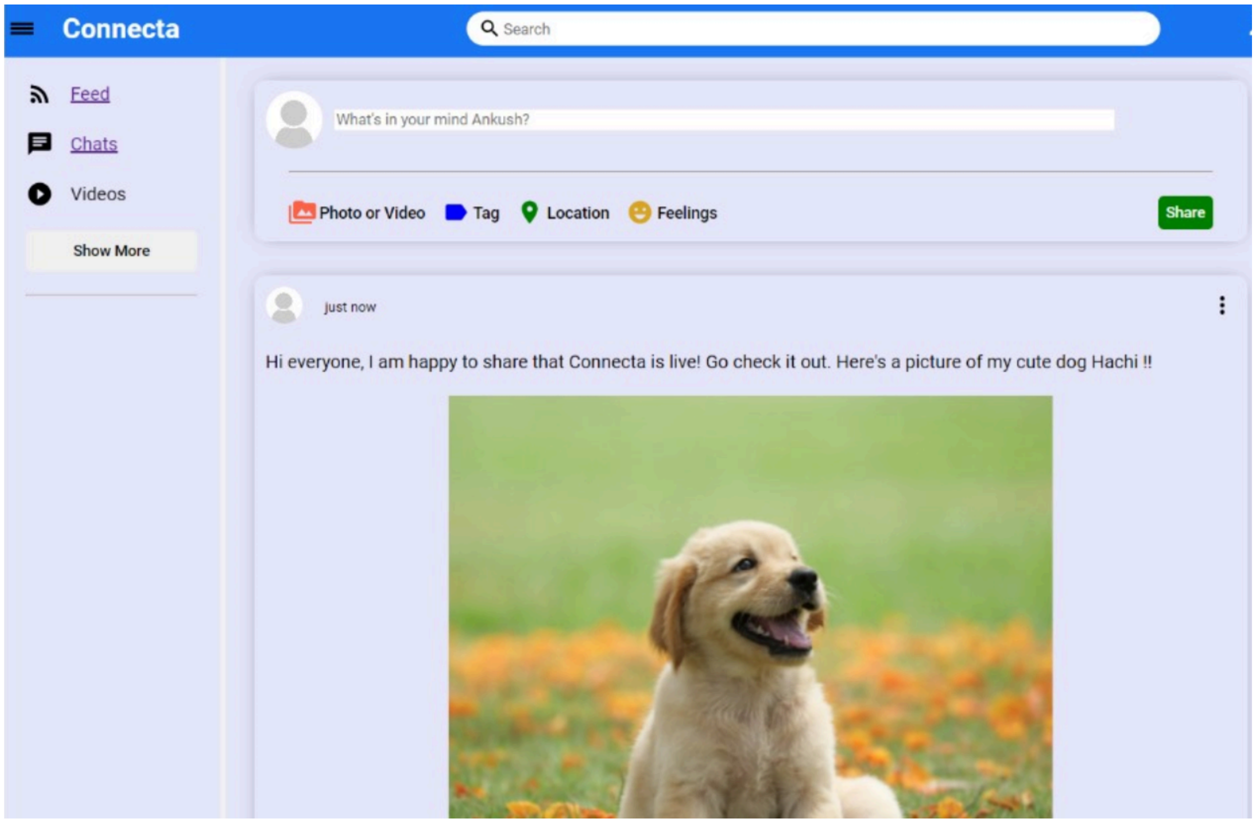
Show More

What's in your mind Ankush?

Photo or Video

Tag

Share



Hi everyone, I am happy to share that Connecta is live! Go check it out. Here's a picture of my cute dog Hachi !!

Conclusion

The execution of the test cases provided valuable insights into the system's performance characteristics and scalability under varying loads. Through meticulous planning and execution, the testing team was able to simulate realistic user scenarios and identify performance bottlenecks, allowing for targeted optimization efforts. The comprehensive test reports generated from the execution of these test cases offer actionable recommendations for enhancing the system's efficiency, responsiveness, and overall user experience. There are various use cases that are yet to be implemented.

To sum up, Connecta, encompasses background concepts, guiding principles, and key methods of each technology. The advantages of such technologies and how to utilize them to build a frontend and backend application are integrated with a NoSQL database. The feasibility of implementing the concepts above in an actual setting is demonstrated by delineating the procedures involved in developing the social media application.