IT314 - Software Engineering

Software Requirements Specifications

Parking Management System

Group No. 29

Group Members

202201431 Heet Thakkar

202201407 Herik Patel

202201417 Vats Shah

202201409 Ramya Shah

202201418 Dev Soni

202201415 Darshan Jogadiya

202201448 Neel Vasoya

202201436 Stavan RaviSaheb

202201401 Vraj Patel

Under the guidance of

Professor: Dr. Saurabh Tiwari

Mentor: Jaydeep

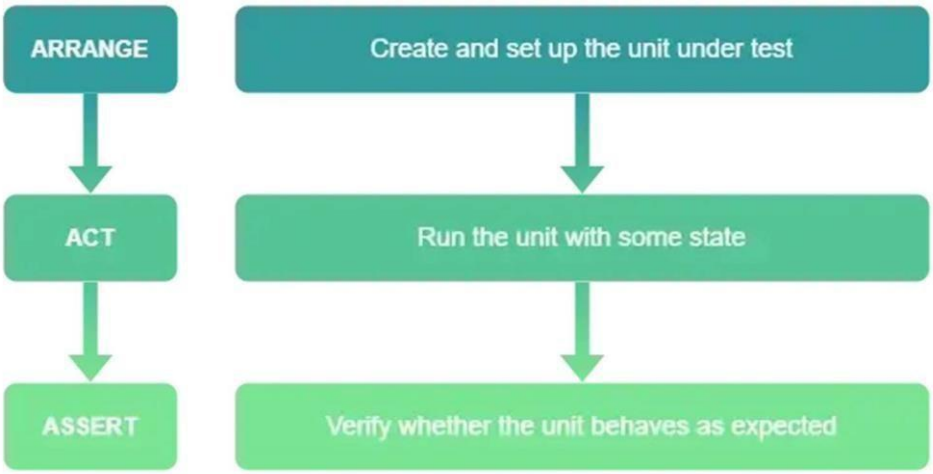
The utilization of the AAA pattern in writing test cases can aid in creating tests that are both easily comprehensible and maintainable, while simultaneously improving the ability to identify and detect errors and bugs.

Arrange: prepare and initialize

Act: invoke method testing

Assert: validate result expected

The AAA (Arrange-Act-Assert) pattern is a widely adopted method for composing test cases that facilitates the creation of tests with clear and logical structure. It involves a set of steps that include: arranging the necessary preconditions, performing the action to be tested, and verifying the expected result.



AAA pattern

1. Arrange: Prepare for the test by creating objects, configuring mock data, or executing other necessary setup tasks that establish the required preconditions.

2. Act: Execute the action that is being tested, which may involve calling a function, interacting with a widget, or performing another relevant action.

3. Assert: In the Assert step, it is necessary to validate that the action produced the intended outcome. This validation process may include verifying the return value of a function, assessing the state of an object, or confirming the truth of another relevant condition.

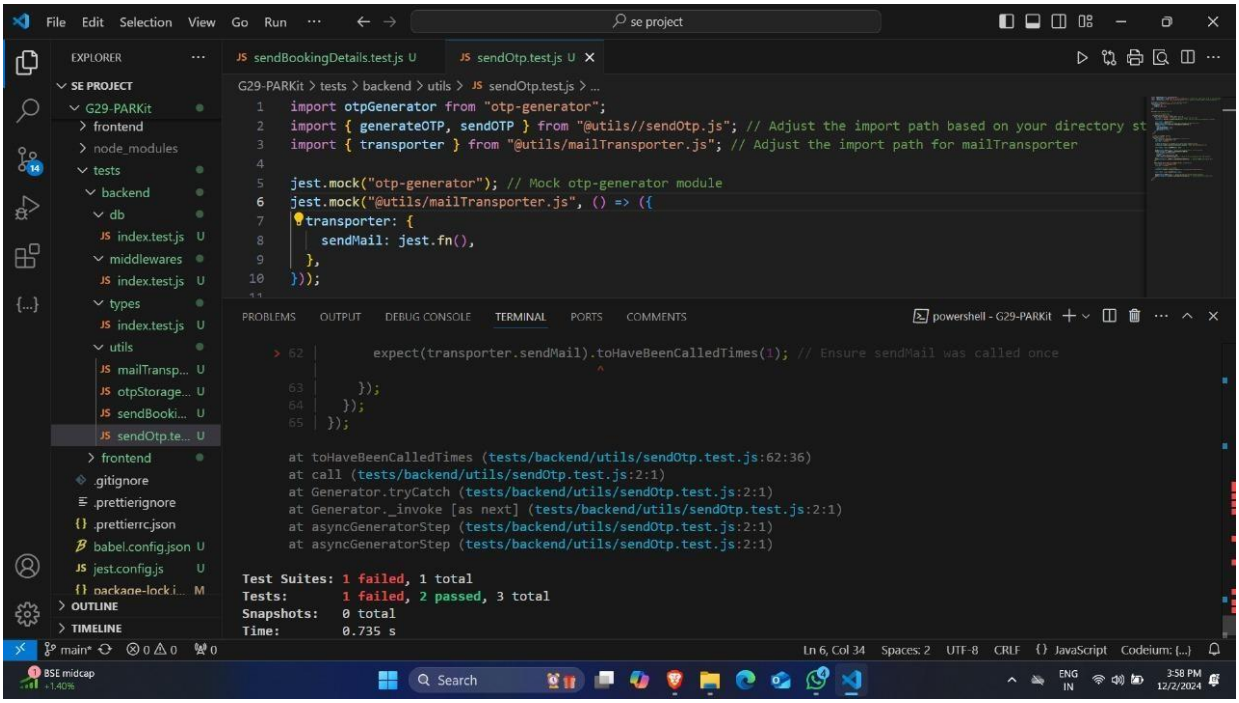
- All parts of unit testing with code :

1. Send OTP :

```javascript
import otpGenerator from "otp-generator";
import { generateOTP, sendOTP } from "../../../backend/utils/sendOtp.js"; // Adjust the import path based on your directory structure
import { transporter } from "../../../backend/utils/mailTransporter.js"; // Adjust the import path for mailTransporter

jest.mock("otp-generator"); // Mock otp-generator module
jest.mock("../../../backend/mailTransporter", () => ({
  transporter: {
    sendMail: jest.fn(),
  },
}));

describe("OTP Service", () => {

  // Test for OTP generation
  describe("generateOTP", () => {
    test("should generate a 6-digit OTP", () => {
      const otp = "123456"; // Example of what you expect the OTP to be
      otpGenerator.generate.mockReturnValue(otp); // Mock the return value of otpGenerator.generate

      const result = generateOTP();
      expect(result).toBe(otp); // Check if OTP matches the expected value
      expect(otpGenerator.generate).toHaveBeenCalledWith(6, {
        digits: true,
        lowerCaseAlphabets: false,
        upperCaseAlphabets: false,
        specialChars: false,
      });
    });
  });
```

```javascript
  // Test for sendOTP function
  describe("sendOTP", () => {
    test("should send OTP email successfully", async () => {
      const email = "test@example.com";
      const otp = "123456";

      // Mock the behavior of sendMail to simulate email sending success
      transporter.sendMail.mockResolvedValue({ response: "Email sent successfully" });

      const result = await sendOTP(email, otp);

      expect(result).toBe(true); // Ensure the function returns true on success
      expect(transporter.sendMail).toHaveBeenCalledWith({
        from: process.env.EMAIL_USER,
        to: email,
        subject: "Email Verification OTP",
        html: expect.stringContaining(otp), // Check if OTP is included in the HTML content
      });
      expect(transporter.sendMail).toHaveBeenCalledTimes(1); // Ensure sendMail was called once
    });

    test("should return false if sendMail fails", async () => {
      const email = "test@example.com";
      const otp = "123456";

      // Simulate an error from sendMail
      transporter.sendMail.mockRejectedValue(new Error("Failed to send email"));

      const result = await sendOTP(email, otp);

      expect(result).toBe(false); // Ensure the function returns false on failure
      expect(transporter.sendMail).toHaveBeenCalledTimes(1); // Ensure sendMail was called once
    });
  });
});
```

OUTPUT :



2. Index :

i. typeindex -

```
import { LoginSchema, SignupSchema, ReservationSchema, FeedbackSchema, ResetPasswordSchema } from "../../../backend/db/index.js";

describe('Validation Schemas', () => {
  // Test cases for LoginSchema
  describe('LoginSchema', () => {
    test('should validate correct email and password', () => {
      const validData = { email: 'test@example.com', password: 'password123' };
      expect(() => LoginSchema.parse(validData)).not.toThrow();
    });

    test('should throw error if email is invalid', () => {
      const invalidData = { email: 'invalid-email', password: 'password123' };
      expect(() => LoginSchema.parse(invalidData)).toThrowError('Invalid email');
    });

    test('should throw error if password is missing', () => {
      const invalidData = { email: 'test@example.com' };
      expect(() => LoginSchema.parse(invalidData)).toThrowError('Required');
    });
  });

  // Test cases for SignupSchema
  describe('SignupSchema', () => {
    test('should validate correct signup data', () => {
      const validData = {
        name: 'John Doe',
        email: 'john.doe@example.com',
        password: 'password123',
        confirmPassword: 'password123',
        agreeToTerms: true,
      };
      expect(() => SignupSchema.parse(validData)).not.toThrow();
    });
```

```
    test('should throw error if name is too short', () => {
      const invalidData = {
        name: 'J',
        email: 'john.doe@example.com',
        password: 'password123',
        confirmPassword: 'password123',
        agreeToTerms: true,
      };
      expect(() => SignupSchema.parse(invalidData)).toThrowError('Name must be at least 2 characters long');
    });

    test('should throw error if password and confirmPassword do not match', () => {
      const invalidData = {
        name: 'John Doe',
        email: 'john.doe@example.com',
        password: 'password123',
        confirmPassword: 'password456',
        agreeToTerms: true,
      };
      expect(() => SignupSchema.parse(invalidData)).toThrowError("Passwords don't match");
    });
  });

  // Test cases for ReservationSchema
  describe('ReservationSchema', () => {
    test('should validate correct reservation data', () => {
      const validData = {
        reservationDate: '2024-12-01',
        reservationTime: '14:00',
        duration: '2 hours',
        vehicleNumberPlate: 'ABC1234',
      };
      expect(() => ReservationSchema.parse(validData)).not.toThrow();
    });

    test('should throw error if vehicle number plate is too short', () => {
      const invalidData = {
        reservationDate: '2024-12-01',
        reservationTime: '14:00',
        duration: '2 hours',
        vehicleNumberPlate: 'A',
      };
      expect(() => ReservationSchema.parse(invalidData)).toThrowError('Number plate is required');
    });

    test('should throw error if vehicle number plate has invalid characters', () => {
      const invalidData = {
        reservationDate: '2024-12-01',
        reservationTime: '14:00',
        duration: '2 hours',
        vehicleNumberPlate: 'ABC@123',
      };
      expect(() => ReservationSchema.parse(invalidData)).toThrowError('Only uppercase letters, numbers and spaces are allowed');
    });
  });

  // Test cases for FeedbackSchema
  describe('FeedbackSchema', () => {
    test('should validate correct feedback data', () => {
      const validData = {
        name: 'John Doe',
        email: 'john.doe@example.com',
        rating: '5',
        message: 'Great service!',
      };
      expect(() => FeedbackSchema.parse(validData)).not.toThrow();
    });

```

```
103     test('should throw error if message is too short', () => {
104       const invalidData = {
105         name: 'John Doe',
106         email: 'john.doe@example.com',
107         rating: '5',
108         message: 'Bad',
109       };
110       expect(() => FeedbackSchema.parse(invalidData)).toThrowError('String must contain at least 10 character(s)');
111     });
112   });
113
114   // Test cases for ResetPasswordSchema
115   describe('ResetPasswordSchema', () => {
116     test('should validate correct reset password data', () => {
117       const validData = {
118         password: 'password123',
119         confirmPassword: 'password123',
120       };
121       expect(() => ResetPasswordSchema.parse(validData)).not.toThrow();
122     });
123
124     test('should throw error if passwords do not match', () => {
125       const invalidData = {
126         password: 'password123',
127         confirmPassword: 'password456',
128       };
129       expect(() => ResetPasswordSchema.parse(invalidData)).toThrowError("Passwords don't match");
130     });
131
132     test('should throw error if password is too short', () => {
133       const invalidData = {
134         password: 'short',
135         confirmPassword: 'short',
136       };
137       expect(() => ResetPasswordSchema.parse(invalidData)).toThrowError('Password must be at least 8 characters long');
138     });
139   });
```

OUTPUT :

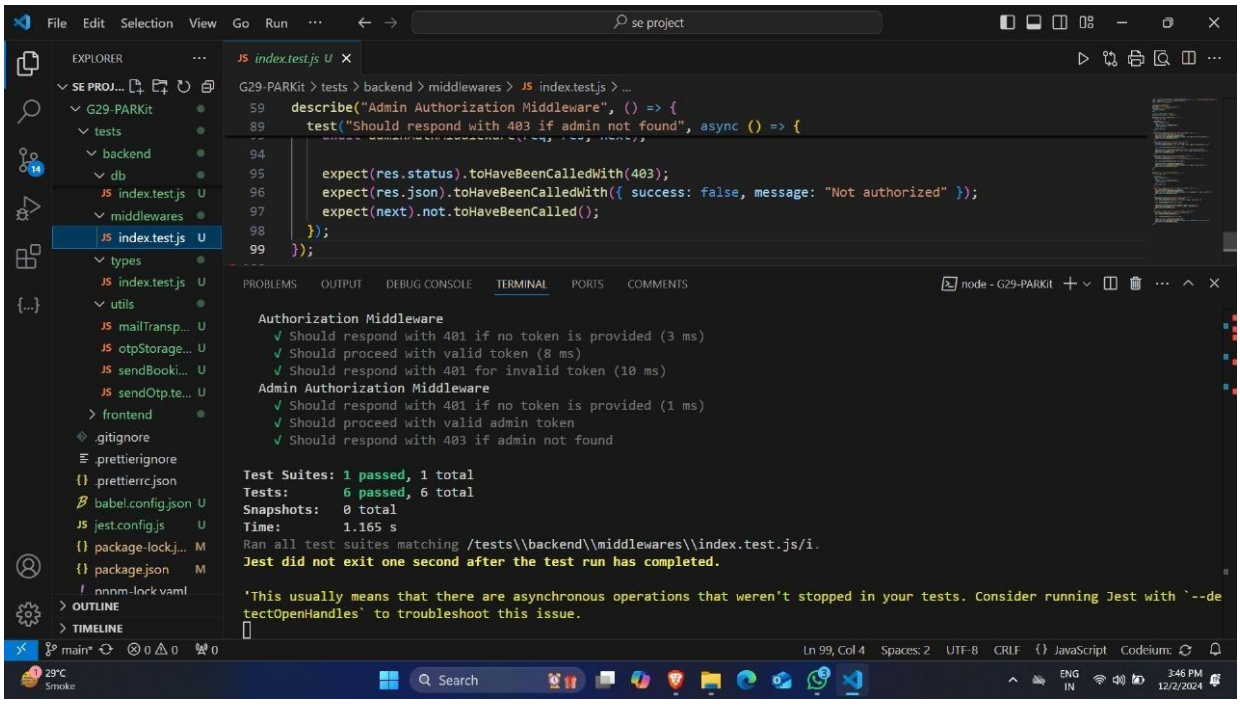
```
PS E:\se project\G29-PARKit> npm run test: file tests/backend/types/index.test.js

> G29-PARKit@1.0.0 test:file
> jest tests/backend/types/index.test.js

 PASS  tests/backend/types/index.test.js

Test Suites: 5 Passed,  5 total
Tests:       14 Passed, 14 total
Snapshots:  0 total
Time:        1.935 s
Ran all test suites matching /tests\\backend\\types\\index.test.js/i.
PS E:\se project\G29-PARKit>
```

ii. Middle ware index –

```
import { authorizationMiddleware, adminAuthMiddleware } from "../../../backend/middlewares/index.js";
import { Admin } from "../../../backend/db/index.js";
import jwt from "jsonwebtoken";

// Mock other dependencies
jest.mock("jsonwebtoken");
jest.mock("../../../backend/db/index.js");
jest.mock("dotenv", () => ({
  config: jest.fn(),
}));

// Set mock environment variables
process.env.JWT_SECRET = "testSecret";

describe("Authorization Middleware", () => {
  let req, res, next;

  beforeEach(() => {
    req = { headers: {} };
    res = {
      status: jest.fn().mockReturnThis(),
      json: jest.fn()
    };
    next = jest.fn();
  });

  test("Should respond with 401 if no token is provided", async () => {
    await authorizationMiddleware(req, res, next);
    expect(res.status).toHaveBeenCalledWith(401);
    expect(res.json).toHaveBeenCalledWith({ success: false, msg: "No token provided" });
    expect(next).not.toHaveBeenCalled();
  });

  test("Should proceed with valid token", async () => {
    req.headers.authorization = "Bearer validToken";
    jwt.verify.mockReturnValue({ _id: "123", role: "user", email: "test@example.com" });

    await authorizationMiddleware(req, res, next);

    expect(jwt.verify).toHaveBeenCalledWith("validToken", process.env.JWT_SECRET);
    expect(req.user).toEqual({ _id: "123", role: "user", email: "test@example.com" });
    expect(next).toHaveBeenCalled();
  });

  test("Should respond with 401 for invalid token", async () => {
    req.headers.authorization = "Bearer invalidToken";
    jwt.verify.mockImplementation(() => { throw new Error("Invalid token"); });

    await authorizationMiddleware(req, res, next);

    expect(res.status).toHaveBeenCalledWith(401);
    expect(res.json).toHaveBeenCalledWith({ success: false, msg: "Invalid token" });
    expect(next).not.toHaveBeenCalled();
  });
});


describe("Admin Authorization Middleware", () => {
  let req, res, next;

  beforeEach(() => {
    req = { headers: {} };
    res = {
      status: jest.fn().mockReturnThis(),
      json: jest.fn()
    };
    next = jest.fn();
  });

```

```
70  test("Should respond with 401 if no token is provided", async () => {
71    await adminAuthMiddleware(req, res, next);
72    expect(res.status).toHaveBeenCalledWith(401);
73    expect(res.json).toHaveBeenCalledWith({ success: false, message: "No token provided" });
74    expect(next).not.toHaveBeenCalled();
75  });
76
77  test("Should proceed with valid admin token", async () => {
78    req.headers.authorization = "Bearer adminToken";
79    Admin.findOne.mockResolvedValue({ _id: "adminId", token: "adminToken" });
80
81    await adminAuthMiddleware(req, res, next);
82
83    expect(Admin.findOne).toHaveBeenCalledWith({ token: "adminToken" });
84    expect(req.admin).toEqual({ _id: "adminId", token: "adminToken" });
85    expect(next).toHaveBeenCalled();
86  });
87
88  test("Should respond with 403 if admin not found", async () => {
89    req.headers.authorization = "Bearer invalidAdminToken";
90    Admin.findOne.mockResolvedValue(null);
91
92    await adminAuthMiddleware(req, res, next);
93
94    expect(res.status).toHaveBeenCalledWith(403);
95    expect(res.json).toHaveBeenCalledWith({ success: false, message: "Not authorized" });
96    expect(next).not.toHaveBeenCalled();
97  });
98 });
```

OUTPUT :



iii. DB index –

```
import mongoose from "mongoose";
import { MongoMemoryServer } from "mongodb-memory-server";
import { User, ParkingSlot, Reservation, Feedback, AdminStats } from "../../../backend/db/index.js"; // Update the path

let mongoServer;

// **Setup** MongoMemoryServer for a test database
beforeAll(async () => {
  mongoServer = await MongoMemoryServer.create();
  const uri = mongoServer.getUri();
  await mongoose.connect(uri, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  });
});

// **Cleanup** the database after each test
afterEach(async () => {
  const collections = mongoose.connection.collections;
  for (const key in collections) {
    await collections[key].deleteMany({});
  }
});

// **Teardown** the MongoMemoryServer and disconnect
afterAll(async () => {
  await mongoose.disconnect();
  await mongoServer.stop();
});

```

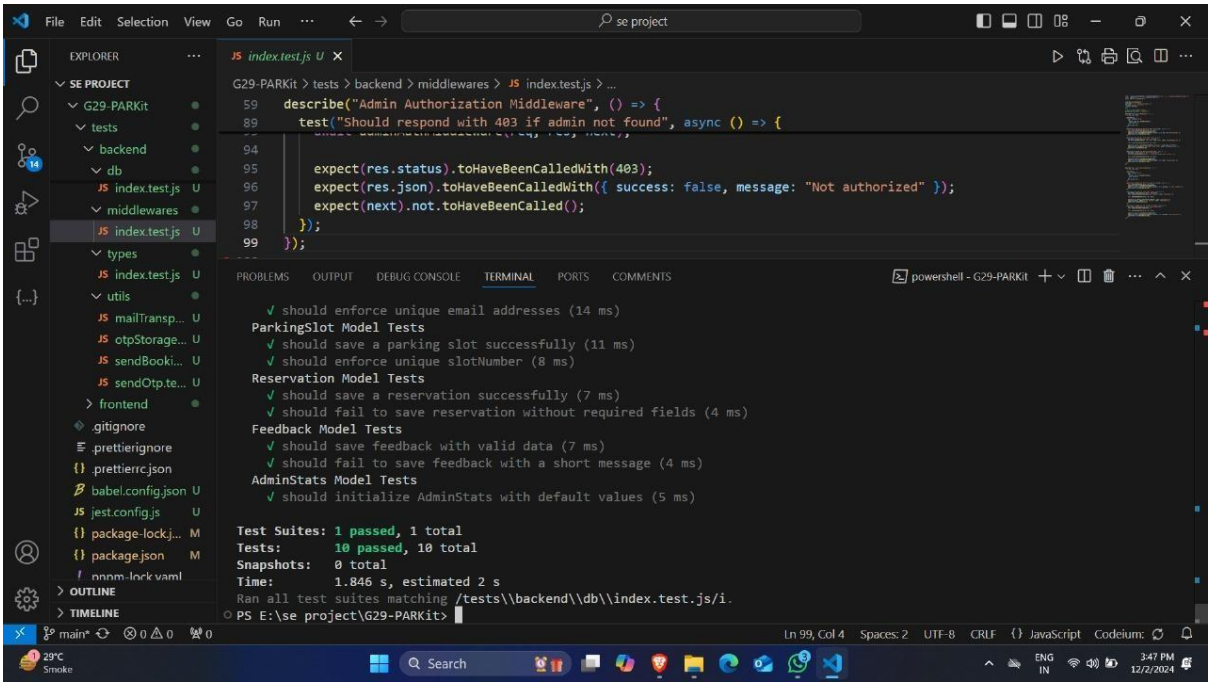
```
describe("User Model Tests", () => {
  it("should save a user with valid data", async () => {
    const user = new User({
      name: "John Doe",
      email: "john.doe@example.com",
      password: "password123",
      phone: "9876543210",
    });

    const savedUser = await user.save();

    // **Assertions**
    expect(savedUser._id).toBeDefined(); // Check if the user is saved
    expect(savedUser.name).toBe("John Doe");
    expect(savedUser.email).toBe("john.doe@example.com");
  });

  it("should throw an error for missing required fields", async () => {
    const user = new User({
      email: "john.doe@example.com", // Missing name, password, phone
    });

    // **Expect error to be thrown**
    await expect(user.save()).rejects.toThrow();
  });

  it("should enforce unique email addresses", async () => {
    const user1 = new User({
      name: "John Doe",
      email: "john.doe@example.com",
      password: "password123",
      phone: "9876543210",
    });
```

```
    const user2 = new User({
      name: "Jane Doe",
      email: "john.doe@example.com", // Same email as user1
      password: "password456",
      phone: "1234567890",
    });

    await user1.save();
    await expect(user2.save()).rejects.toThrow(); // **Should throw unique constraint error**
  });
});

describe("ParkingSlot Model Tests", () => {
  it("should save a parking slot successfully", async () => {
    const slot = new ParkingSlot({
      slotNumber: "A1",
      level: "1",
    });

    const savedSlot = await slot.save();

    // **Assertions**
    expect(savedSlot._id).toBeDefined();
    expect(savedSlot.slotNumber).toBe("A1");
    expect(savedSlot.isOccupied).toBe(false); // Default value
  });

  it("should enforce unique slotNumber", async () => {
    const slot1 = new ParkingSlot({ slotNumber: "A1", level: "1" });
    const slot2 = new ParkingSlot({ slotNumber: "A1", level: "2" }); // Duplicate slotNumber

    await slot1.save();
    await expect(slot2.save()).rejects.toThrow(); // **Should throw unique constraint error**
  });
});

```

```
describe("Reservation Model Tests", () => {
  it("should save a reservation successfully", async () => {
    // Create User and ParkingSlot
    const user = await new User({
      name: "John Doe",
      email: "john.doe@example.com",
      password: "password123",
      phone: "9876543210",
    }).save();

    const slot = await new ParkingSlot({
      slotNumber: "A2",
      level: "2",
    }).save();

    // Create Reservation
    const reservation = new Reservation({
      user: user._id,
      parkingSlot: slot._id,
      vehicleNumberPlate: "ABC-1234",
      reservationTime: new Date(),
      endTime: new Date(Date.now() + 3600 * 1000), // 1 hour later
      duration: 1,
      status: "confirmed",
    });

    const savedReservation = await reservation.save();

    // **Assertions**
    expect(savedReservation._id).toBeDefined();
    expect(savedReservation.user.toString()).toBe(user._id.toString());
    expect(savedReservation.parkingSlot.toString()).toBe(slot._id.toString());
  });

```

```
  it("should fail to save reservation without required fields", async () => {
    const reservation = new Reservation({
      vehicleNumberPlate: "XYZ-5678",
    });

    await expect(reservation.save()).rejects.toThrow();
  });
});

describe("Feedback Model Tests", () => {
  it("should save feedback with valid data", async () => {
    const feedback = new Feedback({
      name: "Jane Doe",
      email: "jane.doe@example.com",
      rating: "5",
      message: "Great service and easy to use!",
    });

    const savedFeedback = await feedback.save();

    // **Assertions**
    expect(savedFeedback._id).toBeDefined();
    expect(savedFeedback.rating).toBe("5");
  });

  it("should fail to save feedback with a short message", async () => {
    const feedback = new Feedback({
      name: "Jane Doe",
      email: "jane.doe@example.com",
      rating: "5",
      message: "Bad", // Too short
    });

    await expect(feedback.save()).rejects.toThrow();
  });
});

describe("AdminStats Model Tests", () => {
  it("should initialize AdminStats with default values", async () => {
    const adminStats = new AdminStats();
    const savedStats = await adminStats.save();

    // **Assertions**
    expect(savedStats._id).toBeDefined();
    expect(savedStats.totalIncome).toBe(0);
    expect(savedStats.totalBookings).toBe(0);
  });
});
```

OUTPUT :


File Edit Selection View Go Run
se project
EXPLORER
SE PROJECT
G29-PARKit
tests
backend
db
index.test.js
middlewares
index.test.js
types
index.test.js
utils
mailTransp...
otpStorage...
sendBooki...
sendOtp.te...
frontend
.gitignore
.prettierignore
.prettierrc.json
babel.config.json
jest.config.js
package-lock.j...
package.json
pnpm-lock.yaml
OUTLINE
TIMELINE
index.test.js U
G29-PARKit > tests > backend > middlewares > index.test.js > ...
59 describe("Admin Authorization Middleware", () => {
89 test("Should respond with 403 if admin not found", async () => {
94
95 expect(res.status).toHaveBeenCalledWith(403);
96 expect(res.json).toHaveBeenCalledWith({ success: false, message: "Not authorized" });
97 expect(next).not.toHaveBeenCalled();
98 });
99 });
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
powershell - G29-PARKit
√ should enforce unique email addresses (14 ms)
ParkingSlot Model Tests
√ should save a parking slot successfully (11 ms)
√ should enforce unique slotNumber (8 ms)
Reservation Model Tests
√ should save a reservation successfully (7 ms)
√ should fail to save reservation without required fields (4 ms)
Feedback Model Tests
√ should save feedback with valid data (7 ms)
√ should fail to save feedback with a short message (4 ms)
AdminStats Model Tests
√ should initialize AdminStats with default values (5 ms)
Test Suites: 1 passed, 1 total
Tests: 10 passed, 10 total
Snapshots: 0 total
Time: 1.846 s, estimated 2 s
Ran all test suites matching /tests\\backend\\db\\index.test.js/i.
PS E:\se project\G29-PARKit>
main*
Ln 99, Col 4 Spaces: 2 UTF-8 CRLF JavaScript Codeium:
29°C Smoke
Search
ENG IN
3:47 PM 12/2/2024

3. Sign UP :

```
import { render, screen, fireEvent, waitFor } from "@testing-library/react";
import { SignUp } from "../../../../frontend/src/components/SignUp.jsx";
import axios from "axios";
import { BrowserRouter as Router } from "react-router-dom";

// Mocking the axios post request
jest.mock("axios");

describe("SignUp Component", () => {

  // Test case 1: Check if the form renders correctly with the initial fields
  test("renders the SignUp form and submits correctly", async () => {
    render(
      <Router>
        <SignUp />
      </Router>
    );

    // Check if the form fields are rendered correctly
    expect(screen.getByLabelText(/name/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/email/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/password/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/confirm password/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/phone number/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/I agree to the Terms & Conditions/i)).toBeInTheDocument();

    // Fill out the form fields
    fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
    fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
    fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });
    fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));

    // Submit the form
    fireEvent.click(screen.getByRole("button", { name: /sign up/i }));

```

```
    // Mock API response for successful submission
    axios.post.mockResolvedValueOnce({ data: { success: true, data: { token: "test_token" } } });

    // Assert that the loading state is displayed
    expect(screen.getByRole("button", { name: /sending otp.../i })).toBeInTheDocument();

    // Wait for the next step (email verification form)
    await waitFor(() => {
      expect(screen.getByText(/verify email/i)).toBeInTheDocument();
    });
  });

  // Test case 2: Check if the form displays error when passwords don't match
  test("displays error when password and confirm password do not match", async () => {
    render(
      <Router>
        <SignUp />
      </Router>
    );

    // Fill out the form with mismatched passwords
    fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
    fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
    fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "differentpassword" } });
    fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });
    fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));

    // Submit the form
    fireEvent.click(screen.getByRole("button", { name: /sign up/i }));

    // Wait for and assert the error message about password mismatch
    expect(await screen.findByText(/passwords don't match/i)).toBeInTheDocument();
  });

```

```
  // Test case 3: Check if the phone number format validation works
  test("displays error when phone number is invalid", async () => {
    render(
      <Router>
        <SignUp />
      </Router>
    );

    // Fill out the form with an invalid phone number
    fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
    fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
    fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "12345" } });
    fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));

    // Submit the form
    fireEvent.click(screen.getByRole("button", { name: /sign up/i }));

    // Check for the phone number validation error message
    expect(await screen.findByText(/phone number must be 10 digits/i)).toBeInTheDocument();
  });

  // Test case 4: Check if the agree to terms checkbox is required
  test("displays error when the user does not agree to terms", async () => {
    render(
      <Router>
        <SignUp />
      </Router>
    );

    // Fill out the form without checking the terms and conditions checkbox
    fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
    fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
    fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });

    // Submit the form
    fireEvent.click(screen.getByRole("button", { name: /sign up/i }));

    // Check for the error message about agreeing to terms
    expect(await screen.findByText(/you must accept the terms and conditions/i)).toBeInTheDocument();
  });

  // Test case 5: Verify OTP submission and token storage
  test("submits OTP and stores token", async () => {
    render(
      <Router>
        <SignUp />
      </Router>
    );

    // Step 1: Simulate submitting the initial signup form
    fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
    fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
    fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });
    fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));

    // Mock the axios.post response for successful sign-up
    axios.post.mockResolvedValueOnce({ data: { success: true, data: { token: "test_token" } } });

    fireEvent.click(screen.getByRole("button", { name: /sign up/i }));
```

```
    // Step 2: Verify OTP
    fireEvent.change(screen.getByLabelText(/Enter OTP/i), { target: { value: "123456" } });

    axios.post.mockResolvedValueOnce({
      data: { success: true, data: { token: "test_token" } },
    });
                any
    fireEvent.click(screen.getByRole("button", { name: /verify otp/i }));

    // Check if the token is stored in localStorage
    expect(localStorage.getItem("token")).toBe("Bearer test_token");
  });

  // Test case 6: Check if the loading state is visible during API requests
  test("shows loading state during API requests", async () => {
    render(
      <Router>
        <SignUp />
      </Router>
    );

    // Fill out the form and submit it
    fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
    fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
    fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
    fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });
    fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));

    // Mocking the axios call to simulate a loading state
    axios.post.mockResolvedValueOnce({ data: { success: true, data: { token: "test_token" } } });

    // Submit the form
    fireEvent.click(screen.getByRole("button", { name: /sign up/i }));

    // Ensure the loading spinner is shown during the request
    expect(screen.getByRole("button", { name: /sending otp.../i })).toBeInTheDocument();

    // Wait for the form to move to the OTP verification step
    await waitFor(() => {
      expect(screen.getByText(/verify email/i)).toBeInTheDocument();
    });
  });
});
```

OUTPUT :

```
PS E:\se project\G29-PARKit> npm run test: file tests/frontend/src/components/SignUp.test.js

> G29-PARKit@1.0.0 test:file
> jest tests/frontend/src/components/SignUp.test.js

 PASS tests/frontend/src/components/SignUp.test.js

Test Suites: 1 Passed, 1 total
Tests:       6 Passed, 6 total
Snapshots:   0 total
Time:        1.758 s
Ran all test suites matching /tests\\frontend\\src\\components\\SignUp.test.js/i.
PS E:\se project\G29-PARKit>
```

4. Booking :

```jsx
import React from "react";
import { render, screen, fireEvent, waitFor } from "@testing-library/react";
import { ToastProvider } from "@/hooks/use-toast";
import { MemoryRouter } from "react-router-dom";
import Component from "../../../../frontend/src/components/Booking.jsx";
import axios from "axios";

jest.mock("axios");

describe("Booking Component", () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  // Utility function to render the component with providers
  const renderWithProviders = () => {
    return render(
      <ToastProvider>
        <MemoryRouter>
          <Component />
        </MemoryRouter>
      </ToastProvider>
    );
  };

  /**
   * Test Case 1: Render form fields
   * Ensure all form fields and the booking button are present in the component.
   */
  test("renders form fields", () => {
    renderWithProviders();

```

```
    expect(screen.getByLabelText(/date/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/select time/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/duration/i)).toBeInTheDocument();
    expect(screen.getByLabelText(/vehicle number plate/i)).toBeInTheDocument();
    expect(screen.getByRole("button", { name: /book parking slot/i })).toBeInTheDocument();
  });

  /**
   * Test Case 2: Display availability dynamically
   * Simulate selecting a date, time, and duration, and check if the availability status updates.
   */
  test("shows availability status dynamically", async () => {
    // Mock API response to indicate availability
    axios.post.mockResolvedValueOnce({ data: { available: true } });

    renderWithProviders();

    const dateInput = screen.getByText(/pick a date/i);
    const timeSelect = screen.getByRole("combobox", { name: /select time/i });
    const durationSelect = screen.getByRole("combobox", { name: /duration/i });

    // Simulate user input
    fireEvent.click(dateInput);
    fireEvent.click(screen.getByText(/1\/1\/2025/i)); // Select a future date
    fireEvent.change(timeSelect, { target: { value: "10:00" } });
    fireEvent.change(durationSelect, { target: { value: "2" } });

    // Wait for the availability status to update
    await waitFor(() => {
      expect(screen.getByText(/slot available/i)).toBeInTheDocument();
    });
  });

  /**
   * Test Case 3: Handle API failure when checking availability
   * Simulate an API failure during availability check and verify error handling.
   */
  test("shows error when API fails to check availability", async () => {
    // Mock API error
    axios.post.mockRejectedValueOnce(new Error("Network error"));

    renderWithProviders();

    const dateInput = screen.getByText(/pick a date/i);
    const timeSelect = screen.getByRole("combobox", { name: /select time/i });
    const durationSelect = screen.getByRole("combobox", { name: /duration/i });

    // Simulate user input
    fireEvent.click(dateInput);
    fireEvent.click(screen.getByText(/1\/1\/2025/i)); // Select a future date
    fireEvent.change(timeSelect, { target: { value: "10:00" } });
    fireEvent.change(durationSelect, { target: { value: "2" } });

    // Wait for the error message
    await waitFor(() => {
      expect(screen.getByText(/failed to check availability/i)).toBeInTheDocument();
    });
  });

  /**
   * Test Case 4: Submit form successfully
   * Simulate a valid form submission and check if the success message is displayed.
   */
  test("submits form successfully", async () => {
    // Mock successful booking response
    const mockResponse = {
      data: {
        data: { slotNumber: 42 },
      },
    };
```

```
    axios.post.mockResolvedValueOnce(mockResponse);

    renderWithProviders();

    const dateInput = screen.getByText(/pick a date/i);
    const timeSelect = screen.getByRole("combobox", { name: /select time/i });
    const durationSelect = screen.getByRole("combobox", { name: /duration/i });
    const numberPlateInput = screen.getByLabelText(/vehicle number plate/i);
    const submitButton = screen.getByRole("button", { name: /book parking slot/i });

    // Simulate user input
    fireEvent.click(dateInput);
    fireEvent.click(screen.getByText(/1\/1\/2025/i)); // Select a future date
    fireEvent.change(timeSelect, { target: { value: "10:00" } });
    fireEvent.change(durationSelect, { target: { value: "2" } });
    fireEvent.change(numberPlateInput, { target: { value: "ABC123" } });

    // Submit the form
    fireEvent.click(submitButton);

    // Wait for the success message
    await waitFor(() => {
      expect(screen.getByText(/slot 42 has been assigned to you/i)).toBeInTheDocument();
    });
  });

  /**
   * Test Case 5: Handle unauthorized access during form submission
   * Simulate an unauthorized error and ensure the user is redirected to the login page.
   */
  test("redirects to login on unauthorized error", async () => {
    // Mock unauthorized error
    axios.post.mockRejectedValueOnce({ response: { status: 401 } });

    renderWithProviders();

    const dateInput = screen.getByText(/pick a date/i);
    const timeSelect = screen.getByRole("combobox", { name: /select time/i });
    const durationSelect = screen.getByRole("combobox", { name: /duration/i });
    const numberPlateInput = screen.getByLabelText(/vehicle number plate/i);
    const submitButton = screen.getByRole("button", { name: /book parking slot/i });

    // Simulate user input
    fireEvent.click(dateInput);
    fireEvent.click(screen.getByText(/1\/1\/2025/i)); // Select a future date
    fireEvent.change(timeSelect, { target: { value: "10:00" } });
    fireEvent.change(durationSelect, { target: { value: "2" } });
    fireEvent.change(numberPlateInput, { target: { value: "ABC123" } });

    // Submit the form
    fireEvent.click(submitButton);

    // Wait for the error message and redirection
    await waitFor(() => {
      expect(screen.getByText(/please login to book a parking slot/i)).toBeInTheDocument();
    });
  });
});
```

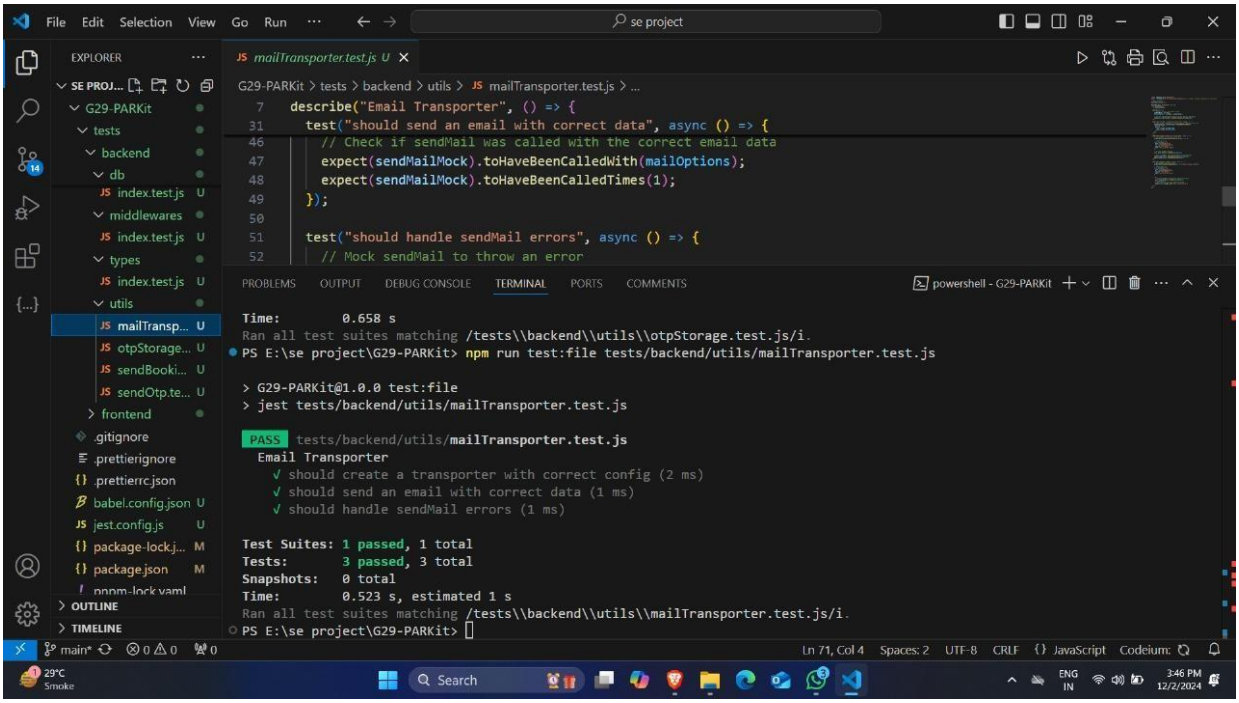
OUTPUT :

```
PS E:\se project\G29-PARKit> npm run test: file tests/frontend/src/components/Booking.test.js

> G29-PARKit@1.0.0 test:file
> jest tests/frontend/src/components/Booking.test.js

 PASS  tests/frontend/src/components/Booking.test.js

Test Suites: 1 Passed,  1 total
Tests:       5 Passed, 5 total
Snapshots:  0 total
Time:        1.355 s
Ran all test suites matching /tests\\frontend\\src\\components\\Booking.test.js/i.
PS E:\se project\G29-PARKit>
```

5. Mail Transporter :

```
import nodemailer from "nodemailer";
import { transporter } from "../../../../backend/utils/mailTransporter.js"; // Adjust the import based on the actual path

// Mock nodemailer
jest.mock("nodemailer");

describe("Email Transporter", () => {
  let sendMailMock;
      ....

  beforeAll(() => {
    // Create a mock for the sendMail method
    sendMailMock = jest.fn().mockResolvedValue({ success: true });
    nodemailer.createTransport.mockReturnValue({ sendMail: sendMailMock });
  });

  test("should create a transporter with correct config", () => {
    // Check if the transporter is created with the correct configuration
    expect(nodemailer.createTransport).toHaveBeenCalledWith({
      service: "gmail",
      auth: {
        user: process.env.EMAIL_USER,
        pass: process.env.EMAIL_PASS,
      },
    });
  });

  test("should send an email with correct data", async () => {
    // Sample email data
    const mailOptions = {
      from: "test@example.com",
      to: "recipient@example.com",
      subject: "Test Email",
      text: "This is a test email",
    };

    // Call the sendMail function
    await transporter.sendMail(mailOptions);

    // Check if sendMail was called with the correct email data
    expect(sendMailMock).toHaveBeenCalledWith(mailOptions);
    expect(sendMailMock).toHaveBeenCalledTimes(1);
  });

  test("should handle sendMail errors", async () => {
    // Mock sendMail to throw an error
    sendMailMock.mockRejectedValue(new Error("Email sending failed"));

    // Sample email data
    const mailOptions = {
      from: "test@example.com",
      to: "recipient@example.com",
      subject: "Test Email",
      text: "This is a test email",
    };

    try {
      // Call the sendMail function and expect it to throw
      await transporter.sendMail(mailOptions);
    } catch (error) {
      // Check that the error was handled correctly
      expect(error.message).toBe("Email sending failed");
    }
  });
});
```

OUTPUT :



6. Foegot Password :

```jsx
import { render, screen, fireEvent, waitFor } from "@testing-library/react";
import { ForgotPassword } from "../../../../frontend/src/components/ForgotPassword.jsx"; // adjust the import as necessary
import axios from "axios";
import { useToast } from "@/hooks/use-toast";

// Mocking axios and toast
jest.mock("axios");
jest.mock("@/hooks/use-toast", () => ({
  useToast: jest.fn(),
}));

describe("ForgotPassword Component", () => {
  let toast;

  beforeEach(() => {
    toast = {
      toast: jest.fn(),
    };
    useToast.mockReturnValue(toast);
  });

  test("renders email form and submits email", async () => {
    render(<ForgotPassword />);

    const emailInput = screen.getByPlaceholderText("Enter your email");
    const submitButton = screen.getByRole("button", { name: /send reset code/i });

    // Simulate entering email
    fireEvent.change(emailInput, { target: { value: "test@example.com" } });

    // Simulate form submission
    fireEvent.click(submitButton);

    // Mock axios to resolve on successful submission
    axios.post.mockResolvedValueOnce({
      data: { success: true },
    });
```

```
38
39      await waitFor(() => {
40        expect(toast.toast).toHaveBeenCalledWith({
41          title: "Success",
42          description: "Verification code sent to your email",
43        });
44      });
45    });
46
47    test("displays error on invalid email", async () => {
48      render(<ForgotPassword />);
49
50      const emailInput = screen.getByPlaceholderText("Enter your email");
51      const submitButton = screen.getByRole("button", { name: /send reset code/i });
52
53      // Simulate entering an invalid email
54      fireEvent.change(emailInput, { target: { value: "invalid-email" } });
55
56      // Simulate form submission
57      fireEvent.click(submitButton);
58
59      await waitFor(() => {
60        expect(screen.getByText(/please enter a valid email address/i)).toBeInTheDocument();
61      });
62    });
63
64    test("renders OTP form and submits OTP", async () => {
65      render(<ForgotPassword />);
66
67      // Switch to step 2 (OTP form)
68      fireEvent.click(screen.getByRole("button", { name: /send reset code/i }));
69
70      const otpInput = screen.getByPlaceholderText("Enter verification code");
71      const submitButton = screen.getByRole("button", { name: /verify code/i });
72
73      // Simulate entering OTP
74      fireEvent.change(otpInput, { target: { value: "123456" } });
75
76      // Simulate form submission
77      fireEvent.click(submitButton);
78
79      // Mock axios to resolve on successful OTP verification
80      axios.post.mockResolvedValueOnce({
81        data: { success: true },
82      });
83
84      await waitFor(() => {
85        expect(toast.toast).toHaveBeenCalledWith({
86          title: "Success",
87          description: "Verification code confirmed",
88        });
89      });
90    });
91
92    test("displays error on invalid OTP", async () => {
93      render(<ForgotPassword />);
94
95      // Switch to step 2 (OTP form)
96      fireEvent.click(screen.getByRole("button", { name: /send reset code/i }));
97
98      const otpInput = screen.getByPlaceholderText("Enter verification code");
99      const submitButton = screen.getByRole("button", { name: /verify code/i });
100
101     // Simulate entering invalid OTP
102     fireEvent.change(otpInput, { target: { value: "wrongotp" } });
103
104     // Simulate form submission
105     fireEvent.click(submitButton);
106
```

```
    await waitFor(() => {
      expect(toast.toast).toHaveBeenCalledWith({
        title: "Error",
        description: "Invalid verification code",
        variant: "destructive",
      });
    });
  });

  test("renders new password form and submits new password", async () => {
    render(<ForgotPassword />);

    // Switch to step 3 (New Password form)
    fireEvent.click(screen.getByRole("button", { name: /send reset code/i }));
    fireEvent.click(screen.getByRole("button", { name: /verify code/i }));

    const passwordInput = screen.getByPlaceholderText("New password");
    const confirmPasswordInput = screen.getByPlaceholderText("Confirm new password");
    const submitButton = screen.getByRole("button", { name: /reset password/i });

    // Simulate entering new password
    fireEvent.change(passwordInput, { target: { value: "NewPass123" } });
    fireEvent.change(confirmPasswordInput, { target: { value: "NewPass123" } });

    // Simulate form submission
    fireEvent.click(submitButton);

    // Mock axios to resolve on successful password reset
    axios.post.mockResolvedValueOnce({
      data: { success: true },
    });

    await waitFor(() => {
      expect(toast.toast).toHaveBeenCalledWith({
        title: "Success",
        description: "Password reset successful",
      });
    });
  });

  test("displays error on password mismatch", async () => {
    render(<ForgotPassword />);

    // Switch to step 3 (New Password form)
    fireEvent.click(screen.getByRole("button", { name: /send reset code/i }));
    fireEvent.click(screen.getByRole("button", { name: /verify code/i }));

    const passwordInput = screen.getByPlaceholderText("New password");
    const confirmPasswordInput = screen.getByPlaceholderText("Confirm new password");

    // Simulate entering mismatched passwords
    fireEvent.change(passwordInput, { target: { value: "NewPass123" } });
    fireEvent.change(confirmPasswordInput, { target: { value: "WrongPass123" } });

    // Simulate form submission
    fireEvent.click(screen.getByRole("button", { name: /reset password/i }));

    await waitFor(() => {
      expect(screen.getByText(/passwords don't match/i)).toBeInTheDocument();
    });
  });
});
```

OUTPUT :

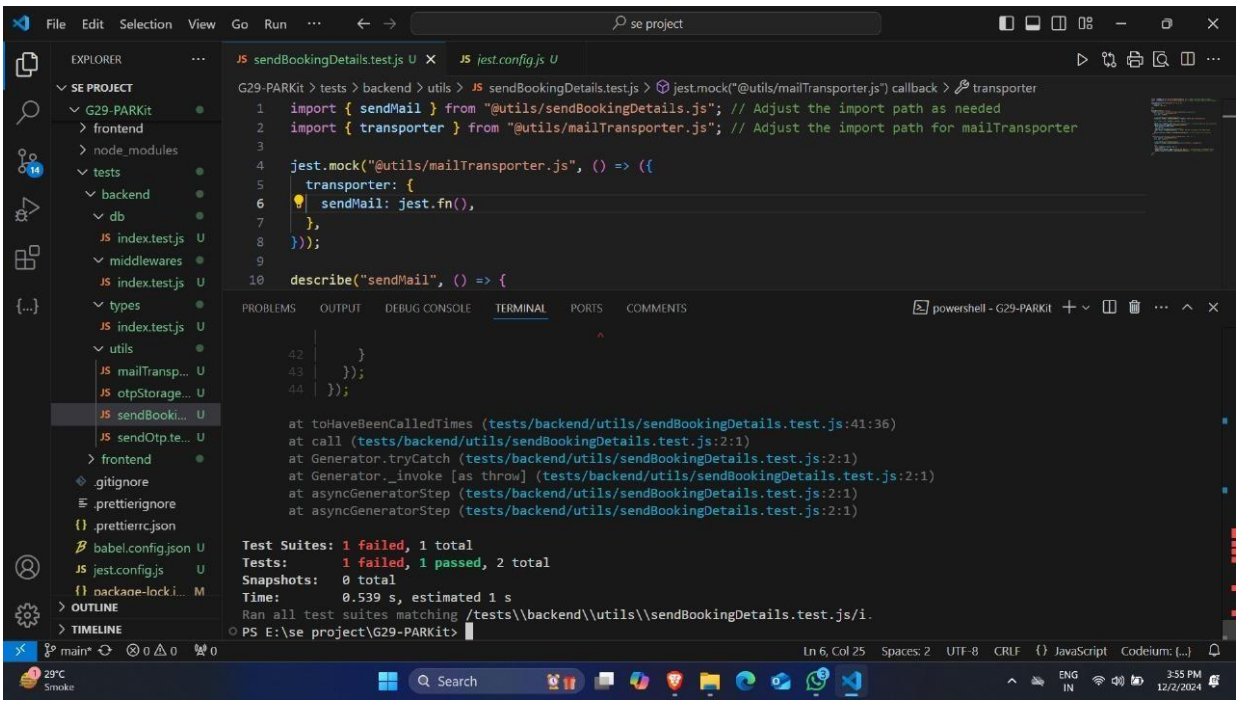
```
PS E:\se project\G29-PARKit> npm run test: file tests/frontend/src/components/ForgotPaswword.test.js

> G29-PARKit@1.0.0 test:file
> jest tests/frontend/src/components/ForgotPaswword.test.js

 PASS tests/frontend/src/components/ForgotPaswword.test.js

Test Suites: 1 Passed, 1 total
Tests:       6 Passed, 6 total
Snapshots:   0 total
Time:        1.532 s
Ran all test suites matching /tests\\frontend\\src\\components\\ForgotPaswword.test.js/i.
PS E:\se project\G29-PARKit>
```

7. SendBookingDetails :

```
import { sendMail } from "../../../backend/utils/sendBookingDetails.js"; // Adjust the import path as needed
import { transporter } from "../../../backend/utils/mailTransporter.js"; // Adjust the import path for mailTransporter

jest.mock("../../../backend/mailTransporter", () => ({
  transporter: {
    sendMail: jest.fn(),
  },
}));

describe("sendMail", () => {
  test("should send a confirmation email successfully", async () => {
    const receiver = "test@example.com";
    const otp = "123456";

    // Mock sendMail to resolve successfully
    transporter.sendMail.mockResolvedValue({ response: "Email sent successfully" });

    const result = await sendMail({ receiver, otp });

    expect(result.response).toBe("Email sent successfully"); // Ensure the email was sent successfully
    expect(transporter.sendMail).toHaveBeenCalledWith({
      from: process.env.EMAIL_USER,
      to: receiver,
      subject: "Parking Reservation Confirmation",
      html: expect.stringContaining(otp), // Ensure the OTP is included in the HTML content
    });
    expect(transporter.sendMail).toHaveBeenCalledTimes(1); // Ensure sendMail was called once
  });
```

```
  test("should throw an error if email sending fails", async () => {
    const receiver = "test@example.com";
    const otp = "123456";

    // Mock sendMail to simulate an error
    transporter.sendMail.mockRejectedValue(new Error("Failed to send email"));

    try {
      await sendMail({ receiver, otp });
    } catch (error) {
      expect(error.message).toBe("Failed to send email"); // Ensure the error is handled properly
      expect(transporter.sendMail).toHaveBeenCalledTimes(1); // Ensure sendMail was called once
    }
  });
});
```

OUTPUT :



8. OTPStorage :

```
import { storeOTP, verifyOTP } from "../../../backend/utils/otpStorage.js"; // Adjust the import path

jest.useFakeTimers(); // Use fake timers to mock Date.now() behavior

describe("OTP Service", () => {
  const email = "test@example.com";
  const otp = "123456";

  beforeEach(() => {
    jest.clearAllMocks(); // Clear mocks before each test
  });

  test("should store OTP and validate correctly", () => {
    // Store the OTP
    storeOTP(email, otp);

    // Simulate immediate OTP verification
    const isValid = verifyOTP(email, otp);

    expect(isValid).toBe(true); // The OTP should be valid
  });

  test("should return false if OTP is incorrect", () => {
    // Store the OTP
    storeOTP(email, otp);

    // Try to verify with an incorrect OTP
    const isValid = verifyOTP(email, "wrongOTP");

    expect(isValid).toBe(false); // The OTP should be invalid
  });

```

```
33  test("should return false if OTP is not found", () => {
34    // Try to verify without storing an OTP
35    const isValid = verifyOTP(email, otp);
36
37    expect(isValid).toBe(false); // The OTP should be invalid because it hasn't been stored
38  });
39
40  test("should return false if OTP has expired", () => {
41    // Store the OTP
42    storeOTP(email, otp);
43
44    // Simulate the passage of 11 minutes (more than the 10-minute expiry time)
45    jest.setSystemTime(Date.now() + 11 * 60 * 1000);
46
47    // Try to verify the OTP after expiration
48    const isValid = verifyOTP(email, otp);
49
50    expect(isValid).toBe(false); // The OTP should be expired
51  });
52
53  test("should delete OTP after successful verification", () => {
54    // Store the OTP
55    storeOTP(email, otp);
56
57    // Verify the OTP
58    const isValid = verifyOTP(email, otp);
59
60    // The OTP should be deleted after successful verification
61    expect(isValid).toBe(true);
62    expect(otpStore.has(email)).toBe(false); // OTP should no longer exist in the store
63  });
64 });
```

OUTPUT :