



## 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

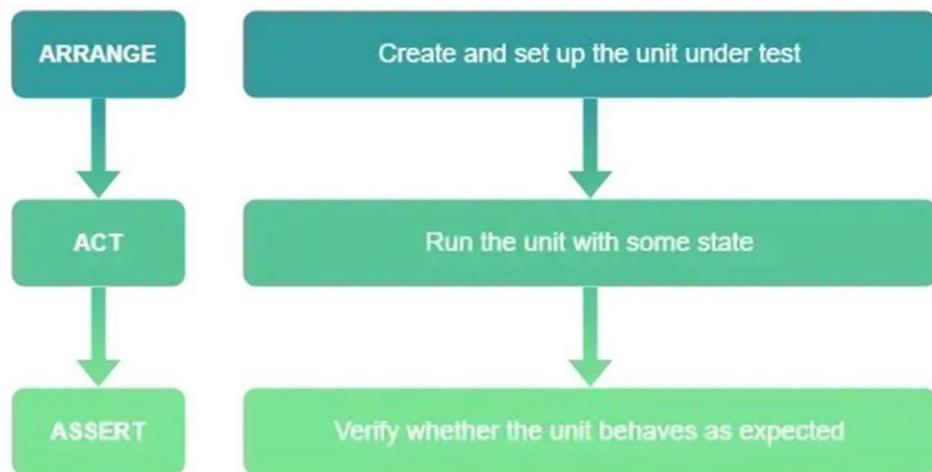
The utilization of the AAA pattern in writing test cases can aid in creating tests that are both easily comprehensible and maintainable, while simultaneously improving the ability to identify and detect errors and bugs.

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 :

```

1 import otpGenerator from "otp-generator";
2 import { generateOTP, sendOTP } from "../../backend/utils/sendOTP.js"; // Adjust the import path based on your directory structure
3 import { transporter } from "../../backend/utils/mailTransporter.js"; // Adjust the import path for mailTransporter
4
5 jest.mock("otp-generator"); // Mock otp-generator module
6 jest.mock("../../backend/mailTransporter", () => ({
7   transporter: {
8     sendMail: jest.fn(),
9   },
10 }));
11
12 describe("OTP Service", () => {
13
14   // Test for OTP generation
15   describe("generateOTP", () => {
16     test("should generate a 6-digit OTP", () => {
17       const otp = "123456"; // Example of what you expect the OTP to be
18       otpGenerator.generate.mockReturnValue(otp); // Mock the return value of otpGenerator.generate
19
20       const result = generateOTP();
21       expect(result).toBe(otp); // Check if OTP matches the expected value
22       expect(otpGenerator.generate).toHaveBeenCalled({
23         digits: true,
24         lowerCaseAlphabets: false,
25         upperCaseAlphabets: false,
26         specialChars: false,
27       });
28     });
29   });
30
31   // Test for sendOTP function
32   describe("sendOTP", () => {
33     test("should send OTP email successfully", async () => {
34       const email = "test@example.com";
35       const otp = "123456";
36
37       // Mock the behavior of sendMail to simulate email sending success
38       transporter.sendMail.mockResolvedValue({ response: "Email sent successfully" });
39
40       const result = await sendOTP(email, otp);
41
42       expect(result).toBe(true); // Ensure the function returns true on success
43       expect(transporter.sendMail).toHaveBeenCalledWith({
44         from: process.env.EMAIL_USER,
45         to: email,
46         subject: "Email Verification OTP",
47         html: expect.stringContaining(otp), // Check if OTP is included in the HTML content
48       });
49       expect(transporter.sendMail).toHaveBeenCalledTimes(1); // Ensure sendMail was called once
50     });
51
52     test("should return false if sendMail fails", async () => {
53       const email = "test@example.com";
54       const otp = "123456";
55
56       // Simulate an error from sendMail
57       transporter.sendMail.mockRejectedValue(new Error("Failed to send email"));
58
59       const result = await sendOTP(email, otp);
60
61       expect(result).toBe(false); // Ensure the function returns false on failure
62       expect(transporter.sendMail).toHaveBeenCalledTimes(1); // Ensure sendMail was called once
63     });
64   });
65 });

```

## OUTPUT :

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "SE PROJECT".
- Editor:** Displays the file `sendOtp.test.js` containing Jest test code for an OTP generator and mail transporter.
- Terminal:** Shows the output of the test run, indicating 1 failed test and 2 passed tests.
- Bottom Status Bar:** Includes system icons like battery level (140%), signal strength, and network status.

```
import { otpGenerator, generateOTP, sendOTP } from "@utils/sendOtp"; // Adjust the import path based on your directory structure
import { transporter } from "@utils/mailTransporter"; // Adjust the import path for mailTransporter

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

expect(transporter.sendMail).toHaveBeenCalledTimes(1); // Ensure sendMail was called once

at toHaveBeenCalledTimes (tests/backend/utils/sendOtp.test.js:62:36)
at call (tests/backend/utils/sendOtp.test.js:2:1)
at Generator.tryCatch (tests/backend/utils/sendOtp.test.js:2:1)
at Generator._invoke [as next] (tests/backend/utils/sendOtp.test.js:2:1)
at asyncGeneratorStep (tests/backend/utils/sendOtp.test.js:2:1)
at asyncGeneratorStep (tests/backend/utils/sendOtp.test.js:2:1)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 2 passed, 3 total
Snapshots:  0 total
Time:        0.735 s
```

## 2. Index :

### i. typeindex -

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "SE PROJECT".
- Editor:** Displays the file `index.test.js` containing Jest test code for LoginSchema and SignupSchema validation.
- Bottom Status Bar:** Includes system icons like battery level (140%), signal strength, and network status.

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

```

35  test('should throw error if name is too short', () => {
36    const invalidData = {
37      name: 'J',
38      email: 'john.doe@example.com',
39      password: 'password123',
40      confirmPassword: 'password123',
41      agreeToTerms: true,
42    };
43    expect(() => SignupSchema.parse(invalidData)).toThrowError('Name must be at least 2 characters long');
44  });
45
46  test('should throw error if password and confirmPassword do not match', () => {
47    const invalidData = {
48      name: 'John Doe',
49      email: 'john.doe@example.com',
50      password: 'password123',
51      confirmPassword: 'password456',
52      agreeToTerms: true,
53    };
54    expect(() => SignupSchema.parse(invalidData)).toThrowError("Passwords don't match");
55  });
56});
57
58 // Test cases for ReservationSchema
59 describe('ReservationSchema', () => {
60   test('should validate correct reservation data', () => {
61     const validData = {
62       reservationDate: '2024-12-01',
63       reservationTime: '14:00',
64       duration: '2 hours',
65       vehicleNumberPlate: 'ABC1234',
66     };
67     expect(() => ReservationSchema.parse(validData)).not.toThrow();
68   });
69
70   test('should throw error if vehicle number plate is too short', () => {
71     const invalidData = {
72       reservationDate: '2024-12-01',
73       reservationTime: '14:00',
74       duration: '2 hours',
75       vehicleNumberPlate: 'A',
76     };
77     expect(() => ReservationSchema.parse(invalidData)).toThrowError('Number plate is required');
78   });
79
80   test('should throw error if vehicle number plate has invalid characters', () => {
81     const invalidData = {
82       reservationDate: '2024-12-01',
83       reservationTime: '14:00',
84       duration: '2 hours',
85       vehicleNumberPlate: 'ABC@123',
86     };
87     expect(() => ReservationSchema.parse(invalidData)).toThrowError('Only uppercase letters, numbers and spaces are allowed');
88   });
89 });
90
91 // Test cases for FeedbackSchema
92 describe('FeedbackSchema', () => {
93   test('should validate correct feedback data', () => {
94     const validData = {
95       name: 'John Doe',
96       email: 'john.doe@example.com',
97       rating: '5',
98       message: 'Great service!',
99     };
100    expect(() => FeedbackSchema.parse(validData)).not.toThrow();
101  });

```

```

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 –

```
1 import { authorizationMiddleware, adminAuthMiddleware } from "../../../../backend/middlewares/index.js";
2 import { Admin } from "../../../../backend/db/index.js";
3 import jwt from "jsonwebtoken";
4
5 // Mock other dependencies
6 jest.mock("jsonwebtoken");
7 jest.mock "../../../../backend/db/index.js";
8 jest.mock("dotenv", () => ({
9   config: jest.fn(),
10 }));
11
12 // Set mock environment variables
13 process.env.JWT_SECRET = "testSecret";
14
15 describe("Authorization Middleware", () => {
16   let req, res, next;
17
18   beforeEach(() => {
19     req = { headers: {} };
20     res = {
21       status: jest.fn().mockReturnThis(),
22       json: jest.fn()
23     };
24     next = jest.fn();
25   });
26
27   test("Should respond with 401 if no token is provided", async () => {
28     await authorizationMiddleware(req, res, next);
29     expect(res.status).toHaveBeenCalledWith(401);
30     expect(res.json).toHaveBeenCalledWith({ success: false, msg: "No token provided" });
31     expect(next).not.toHaveBeenCalled();
32   });
33
34   test("Should proceed with valid token", async () => {
35     req.headers.authorization = "Bearer validToken";
36     jwt.verify.mockReturnValue({ _id: "123", role: "user", email: "test@example.com" });
37
38   await authorizationMiddleware(req, res, next);
39
40   expect(jwt.verify).toHaveBeenCalled("validToken", process.env.JWT_SECRET);
41   expect(req.user).toEqual({ _id: "123", role: "user", email: "test@example.com" });
42   expect(next).toHaveBeenCalled();
43 });
44
45   test("should respond with 401 for invalid token", async () => {
46     req.headers.authorization = "Bearer invalidToken";
47     jwt.verify.mockImplementation(() => { throw new Error("Invalid token"); });
48
49     await authorizationMiddleware(req, res, next);
50
51     expect(res.status).toHaveBeenCalledWith(401);
52     expect(res.json).toHaveBeenCalledWith({ success: false, msg: "Invalid token" });
53     expect(next).not.toHaveBeenCalled();
54   });
55 });
56
57
58 describe("Admin Authorization Middleware", () => {
59   let req, res, next;
60
61   beforeEach(() => {
62     req = { headers: {} };
63     res = {
64       status: jest.fn().mockReturnThis(),
65       json: jest.fn()
66     };
67     next = jest.fn();
68   });
69 })
```

```

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).toHaveBeenCalled();
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 :

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure with files like `index.test.js`, `types`, `utils`, and `middlewares`.
- Terminal:** Displays the output of the Jest test run:
 

```

        Authorization Middleware
        ✓ Should respond with 401 if no token is provided (3 ms)
        ✓ Should proceed with valid token (8 ms)
        ✓ Should respond with 401 for invalid token (10 ms)
      Admin Authorization Middleware
        ✓ Should respond with 401 if no token is provided (1 ms)
        ✓ Should proceed with valid admin token
        ✓ Should respond with 403 if admin not found

      Test Suites: 1 passed, 1 total
      Tests:       6 passed, 6 total
      Snapshots:  0 total
      Time:        1.165 s
      Ran all test suites matching /tests\\backend\\middlewares\\index.test.js/i.
      Jest did not exit one second after the test run has completed.

      'This usually means that there are asynchronous operations that weren't stopped in your tests. Consider running Jest with '--detachedOpenHandles' to troubleshoot this issue.
      
```
- Status Bar:** Shows system information including temperature (29°C), battery level (80%), and system time (3:46 PM, 12/2/2024).

### iii. DB index –

```
1 import mongoose from "mongoose";
2 import { MongoMemoryServer } from "mongodb-memory-server";
3 import { User, Parkingslot, Reservation, Feedback, AdminStats } from "../../../../backend/db/index.js"; // Update the path
4
5 let mongoServer;
6
7 // **Setup** MongoMemoryServer for a test database
8 beforeAll(async () => {
9   mongoServer = await MongoMemoryServer.create();
10  const uri = mongoServer.getUri();
11  await mongoose.connect(uri, {
12    useNewUrlParser: true,
13    useUnifiedTopology: true,
14  });
15 });
16
17 // **Cleanup** the database after each test
18 afterEach(async () => {
19   const collections = mongoose.connection.collections;
20   for (const key in collections) {
21     await collections[key].deleteMany({});
22   }
23 });
24
25 // **Teardown** the MongoMemoryServer and disconnect
26 afterAll(async () => {
27   await mongoose.disconnect();
28   await mongoServer.stop();
29 });
30
```

```
31 describe("User Model Tests", () => {
32   it("should save a user with valid data", async () => {
33     const user = new User({
34       name: "John Doe",
35       email: "john.doe@example.com",
36       password: "password123",
37       phone: "9876543210",
38     });
39
40     const savedUser = await user.save();
41
42     // **Assertions**
43     expect(savedUser._id).toBeDefined(); // Check if the user is saved
44     expect(savedUser.name).toBe("John Doe");
45     expect(savedUser.email).toBe("john.doe@example.com");
46   });
47
48   it("should throw an error for missing required fields", async () => {
49     const user = new User({
50       email: "john.doe@example.com", // Missing name, password, phone
51     });
52
53     // **Expect error to be thrown**
54     await expect(user.save()).rejects.toThrow();
55   });
56
57   it("should enforce unique email addresses", async () => {
58     const user1 = new User({
59       name: "John Doe",
60       email: "john.doe@example.com",
61       password: "password123",
62       phone: "9876543210",
63     });
64
65     // **Expect error to be thrown**
66     await expect(user1.save()).rejects.toThrow();
67   });
68
69   it("should update user information", async () => {
70     const user = new User({
71       name: "John Doe",
72       email: "john.doe@example.com",
73       password: "password123",
74       phone: "9876543210",
75     });
76
77     // **Update user information**
78     user.name = "Jane Doe";
79     user.email = "jane.doe@example.com";
80     user.password = "newpassword123";
81     user.phone = "1234567890";
82
83     const updatedUser = await user.save();
84
85     // **Assertions**
86     expect(updatedUser.name).toBe("Jane Doe");
87     expect(updatedUser.email).toBe("jane.doe@example.com");
88     expect(updatedUser.password).not.toBe("password123");
89     expect(updatedUser.phone).toBe("1234567890");
90   });
91
92   it("should delete a user", async () => {
93     const user = new User({
94       name: "John Doe",
95       email: "john.doe@example.com",
96       password: "password123",
97       phone: "9876543210",
98     });
99
100    const deletedUser = await user.deleteOne();
101
102    // **Assertions**
103    expect(deletedUser.deletedCount).toBe(1);
104  });
105});
```

```

64  const user2 = new User({
65    name: "Jane Doe",
66    email: "john.doe@example.com", // Same email as user1
67    password: "password456",
68    phone: "1234567890",
69  });
70
71  await user1.save();
72  await expect(user2.save()).rejects.toThrow(); // **Should throw unique constraint error**
73 });
74 });
75
76 describe("ParkingSlot Model Tests", () => {
77  it("should save a parking slot successfully", async () => {
78    const slot = new ParkingSlot({
79      slotNumber: "A1",
80      level: "1",
81    });
82
83    const savedSlot = await slot.save();
84
85    // **Assertions**
86    expect(savedSlot._id).toBeDefined();
87    expect(savedSlot.slotNumber).toBe("A1");
88    expect(savedSlot.isOccupied).toBe(false); // Default value
89  });
90
91  it("should enforce unique slotNumber", async () => {
92    const slot1 = new ParkingSlot({ slotNumber: "A1", level: "1" });
93    const slot2 = new ParkingSlot({ slotNumber: "A1", level: "2" }); // Duplicate slotNumber
94
95    await slot1.save();
96    await expect(slot2.save()).rejects.toThrow(); // **Should throw unique constraint error**
97  });
98 });
99

```

```

100 describe("Reservation Model Tests", () => {
101  it("should save a reservation successfully", async () => {
102    // Create User and Parkingslot
103    const user = await new User({
104      name: "John Doe",
105      email: "john.doe@example.com",
106      password: "password123",
107      phone: "9876543210",
108    }).save();
109
110    const slot = await new ParkingSlot({
111      slotNumber: "A2",
112      level: "2",
113    }).save();
114
115    // Create Reservation
116    const reservation = new Reservation({
117      user: user._id,
118      parkingslot: slot._id,
119      vehicleNumberPlate: "ABC-1234",
120      reservationTime: new Date(),
121      endTime: new Date(Date.now() + 3600 * 1000), // 1 hour later
122      duration: 1,
123      status: "confirmed",
124    });
125
126    const savedReservation = await reservation.save();
127
128    // **Assertions**
129    expect(savedReservation._id).toBeDefined();
130    expect(savedReservation.user.toString()).toBe(user._id.toString());
131    expect(savedReservation.parkingslot.toString()).toBe(slot._id.toString());
132  });
133

```

```

134     it("should fail to save reservation without required fields", async () => {
135       const reservation = new Reservation({
136         vehicleNumberPlate: "XYZ-5678",
137       });
138
139       await expect(reservation.save()).rejects.toThrow();
140     });
141   });
142
143 describe("Feedback Model Tests", () => {
144   it("should save feedback with valid data", async () => {
145     const feedback = new Feedback({
146       name: "Jane Doe",
147       email: "jane.doe@example.com",
148       rating: "5",
149       message: "Great service and easy to use!",
150     });
151
152     const savedFeedback = await feedback.save();
153
154     // **Assertions**
155     expect(savedFeedback._id).toBeDefined();
156     expect(savedFeedback.rating).toBe("5");
157   });
158
159   it("should fail to save feedback with a short message", async () => {
160     const feedback = new Feedback({
161       name: "Jane Doe",
162       email: "jane.doe@example.com",
163       rating: "5",
164       message: "Bad", // Too short
165     });
166
167     await expect(feedback.save()).rejects.toThrow();
168   });
169 });
170

```

```

171 describe("AdminStats Model Tests", () => {
172   it("should initialize AdminStats with default values", async () => {
173     const adminStats = new AdminStats();
174     const savedStats = await adminStats.save();
175
176     // **Assertions**
177     expect(savedStats._id).toBeDefined();
178     expect(savedStats.totalIncome).toBe(0);
179     expect(savedStats.totalBookings).toBe(0);
180   });
181 });

```

## OUTPUT :

The screenshot shows a Microsoft Visual Studio Code interface. The left sidebar (Explorer) displays a project structure for 'G29-PARKit' with several test files like 'index.test.js' under 'tests' and 'backend'. The main editor area shows a portion of 'index.test.js' with Jest test cases. Below the editor is a terminal window titled 'powershell - G29-PARKit' showing the output of a Jest test run. The terminal output includes statistics: 1 test suite, 10 tests passed, 0 snapshots, and a total time of 1.846s. It also indicates that all test suites matching '/tests\\backend\\db\\index.test.js' were run. The status bar at the bottom shows the file path 'E:\se project\G29-PARKit>', the current line 'In 99, Col 4', and the date/time '12/2/2024 3:47 PM'.

```
JS index.test.js ✘
G29-PARKit > tests > backend > middlewares > JS index.test.js > ...
  describe("Admin Authorization Middleware", () => {
    test("Should respond with 403 if admin not found", async () => {
      expect(res.status).toHaveBeenCalledWith(403);
      expect(res.json).toHaveBeenCalled({ success: false, message: "Not authorized" });
      expect(next).not.toHaveBeenCalled();
    });
  });
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>
```

### 3. Sign UP :

```
1 ✓ import { render, screen, fireEvent, waitFor } from "@testing-library/react";
2 import { Signup } from "../../../../frontend/src/components/signup.jsx";
3 import axios from "axios";
4 import { BrowserRouter as Router } from "react-router-dom";
5
6 // Mocking the axios post request
7 jest.mock("axios");
8
9 ✓ describe("signUp Component", () => {
10
11   // Test case 1: check if the form renders correctly with the initial fields
12   test("renders the SignUp form and submits correctly", async () => {
13     render(
14       <Router>
15         <Signup />
16       </Router>
17     );
18
19     // Check if the form fields are rendered correctly
20     expect(screen.getByLabelText(/name/i)).toBeInTheDocument();
21     expect(screen.getByLabelText(/email/i)).toBeInTheDocument();
22     expect(screen.getByLabelText(/password/i)).toBeInTheDocument();
23     expect(screen.getByLabelText(/confirm password/i)).toBeInTheDocument();
24     expect(screen.getByLabelText(/phone number/i)).toBeInTheDocument();
25     expect(screen.getByLabelText(/I agree to the Terms & Conditions/i)).toBeInTheDocument();
26
27     // Fill out the form fields
28     fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
29     fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
30     fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
31     fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
32     fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });
33     fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));
34
35     // Submit the form
36     fireEvent.click(screen.getByRole("button", { name: /sign up/i }));
37   });

```

```
38   // Mock API response for successful submission
39   axios.post.mockResolvedValueOnce({ data: { success: true, data: { token: "test_token" } } });
40
41   // Assert that the loading state is displayed
42   expect(screen.getByRole("button", { name: /sending otp.../i })).toBeInTheDocument();
43
44   // Wait for the next step (email verification form)
45   await waitFor(() => {
46     | expect(screen.getByText(/verify email/i)).toBeInTheDocument();
47   });
48 });
49
50 // Test case 2: Check if the form displays error when passwords don't match
51 test("displays error when password and confirm password do not match", async () => {
52   render(
53     <Router>
54       <Signup />
55     </Router>
56   );
57
58   // Fill out the form with mismatched passwords
59   fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
60   fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
61   fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
62   fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "differentpassword" } });
63   fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });
64   fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));
65
66   // Submit the form
67   fireEvent.click(screen.getByRole("button", { name: /sign up/i }));
68
69   // Wait for and assert the error message about password mismatch
70   expect(await screen.findByText(/passwords don't match/i)).toBeInTheDocument();
71 });
72
```

```

73 // Test case 3: Check if the phone number format validation works
74 test("displays error when phone number is invalid", async () => {
75   render(
76     <Router>
77       <SignUp />
78     </Router>
79   );
80
81   // Fill out the form with an invalid phone number
82   fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
83   fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
84   fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
85   fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
86   fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "12345" } });
87   fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));
88
89   // Submit the form
90   fireEvent.click(screen.getByRole("button", { name: /sign up/i }));
91
92   // Check for the phone number validation error message
93   expect(await screen.findByText(/phone number must be 10 digits/i)).toBeInTheDocument();
94 });
95
96 // Test case 4: Check if the agree to terms checkbox is required
97 test("displays error when the user does not agree to terms", async () => {
98   render(
99     <Router>
100      <SignUp />
101    </Router>
102  );
103
104   // Fill out the form without checking the terms and conditions checkbox
105   fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
106   fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
107   fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
108   fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
109   fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });
110
111   // Submit the form
112   fireEvent.click(screen.getByRole("button", { name: /sign up/i }));
113
114   // Check for the error message about agreeing to terms
115   expect(await screen.findByText(/you must accept the terms and conditions/i)).toBeInTheDocument();
116 });
117
118 // Test case 5: Verify OTP submission and token storage
119 test("submits OTP and stores token", async () => {
120   render(
121     <Router>
122       <SignUp />
123     </Router>
124   );
125
126   // Step 1: Simulate submitting the initial signup form
127   fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
128   fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "johndoe@example.com" } });
129   fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
130   fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
131   fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });
132   fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));
133
134   // Mock the axios.post response for successful sign-up
135   axios.post.mockResolvedValueOnce({ data: { success: true, data: { token: "test_token" } } });
136
137   fireEvent.click(screen.getByRole("button", { name: /sign up/i }));
138

```

```

139 // Step 2: Verify OTP
140 fireEvent.change(screen.getByLabelText(/Enter OTP/i), { target: { value: "123456" } });
141
142 axios.post.mockResolvedValueOnce({
143   | data: { success: true, data: { token: "test_token" } },
144 });
145   | any
146 fireEvent.click(screen.getByRole("button", { name: /verify otp/i }));
147
148 // Check if the token is stored in localStorage
149 expect(localStorage.getItem("token")).toBe("Bearer test_token");
150 );
151
152 // Test case 6: Check if the loading state is visible during API requests
153 test("shows loading state during API requests", async () => {
154   render(
155     <Router>
156       <SignUp />
157     </Router>
158   );
159
160   // Fill out the form and submit it
161   fireEvent.change(screen.getByLabelText(/name/i), { target: { value: "John Doe" } });
162   fireEvent.change(screen.getByLabelText(/email/i), { target: { value: "john.doe@example.com" } });
163   fireEvent.change(screen.getByLabelText(/password/i), { target: { value: "password123" } });
164   fireEvent.change(screen.getByLabelText(/confirm password/i), { target: { value: "password123" } });
165   fireEvent.change(screen.getByLabelText(/phone number/i), { target: { value: "1234567890" } });
166   fireEvent.click(screen.getByLabelText(/I agree to the Terms & Conditions/i));
167
168   // Mocking the axios call to simulate a loading state
169   axios.post.mockResolvedValueOnce({ data: { success: true, data: { token: "test_token" } } });
170
171   // Submit the form
172   fireEvent.click(screen.getByRole("button", { name: /sign up/i }));
173
174   // Ensure the loading spinner is shown during the request
175   expect(screen.getByRole("button", { name: /sending otp.../i })).toBeInTheDocument();
176
177   // Wait for the form to move to the OTP verification step
178   <div> await waitFor(() => {
179     | expect(screen.getByText(/verify email/i)).toBeInTheDocument();
180   });
181 </div>
182 );
183

```

## 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 :

```
1 import React from "react";
2 import { render, screen, fireEvent, waitFor } from "@testing-library/react";
3 import { ToastProvider } from "@/hooks/use-toast";
4 import { MemoryRouter } from "react-router-dom";
5 import Component from "../../../Frontend/src/components/Booking.jsx";
6 import axios from "axios";
7
8 jest.mock("axios");
9
10 describe("Booking Component", () => {
11   beforeEach(() => {
12     jest.clearAllMocks();
13   });
14
15   // Utility function to render the component with providers
16   const renderWithProviders = () => {
17     return render(
18       <ToastProvider>
19         <MemoryRouter>
20           <Component />
21         </MemoryRouter>
22       </ToastProvider>
23     );
24   };
25
26 /**
27 * Test Case 1: Render form fields
28 * Ensure all form fields and the booking button are present in the component.
29 */
30 test("renders form fields", () => {
31   renderWithProviders();
32 })
```

```

33  expect(screen.getByLabelText(/date/i)).toBeInTheDocument();
34  expect(screen.getByLabelText(/select time/i)).toBeInTheDocument();
35  expect(screen.getByLabelText(/duration/i)).toBeInTheDocument();
36  expect(screen.getByLabelText(/vehicle number plate/i)).toBeInTheDocument();
37  expect(screen.getByRole("button", { name: /book parking slot/i })).toBeInTheDocument();
38 });
39
40 /**
41 * Test Case 2: Display availability dynamically
42 * Simulate selecting a date, time, and duration, and check if the availability status updates.
43 */
44 test("shows availability status dynamically", async () => {
45   // Mock API response to indicate availability
46   axios.post.mockResolvedValueOnce({ data: { available: true } });
47
48   renderWithProviders();
49
50   const dateInput = screen.getByText(/pick a date/i);
51   const timeSelect = screen.getByRole("combobox", { name: /select time/i });
52   const durationSelect = screen.getByRole("combobox", { name: /duration/i });
53
54   // Simulate user input
55   fireEvent.click(dateInput);
56   fireEvent.click(screen.getByText(/1\1\2025/i)); // Select a future date
57   fireEvent.change(timeSelect, { target: { value: "10:00" } });
58   fireEvent.change(durationSelect, { target: { value: "2" } });
59
60   // Wait for the availability status to update
61   await waitFor(() =>
62     | expect(screen.getByText(/slot available/i)).toBeInTheDocument();
63   );
64 });
65
66 /**
67 * Test Case 3: Handle API failure when checking availability
68 * Simulate an API failure during availability check and verify error handling.
69 */
70 test("shows error when API fails to check availability", async () => {
71   // Mock API error
72   axios.post.mockRejectedValueOnce(new Error("Network error"));
73
74   renderWithProviders();
75
76   const dateInput = screen.getByText(/pick a date/i);
77   const timeSelect = screen.getByRole("combobox", { name: /select time/i });
78   const durationSelect = screen.getByRole("combobox", { name: /duration/i });
79
80   // Simulate user input
81   fireEvent.click(dateInput);
82   fireEvent.click(screen.getByText(/1\1\2025/i)); // Select a future date
83   fireEvent.change(timeSelect, { target: { value: "10:00" } });
84   fireEvent.change(durationSelect, { target: { value: "2" } });
85
86   // Wait for the error message
87   await waitFor(() =>
88     | expect(screen.getByText(/failed to check availability/i)).toBeInTheDocument();
89   );
90 });
91
92 /**
93 * Test Case 4: Submit form successfully
94 * Simulate a valid form submission and check if the success message is displayed.
95 */
96 test("submits form successfully", async () => {
97   // Mock successful booking response
98   const mockResponse = {
99     data: {
100       data: { slotNumber: 42 },
101     },
102   };

```

```

103    axios.post.mockResolvedValueOnce(mockResponse);
104
105    renderWithProviders();
106
107    const dateInput = screen.getByText(/pick a date/i);
108    const timeSelect = screen.getByRole("combobox", { name: /select time/i });
109    const durationSelect = screen.getByRole("combobox", { name: /duration/i });
110    const numberPlateInput = screen.getByLabelText(/vehicle number plate/i);
111    const submitButton = screen.getByRole("button", { name: /book parking slot/i });
112
113    // Simulate user input
114    fireEvent.click(dateInput);
115    fireEvent.click(screen.getByText(/1\|1\|2025/i)); // Select a future date
116    fireEvent.change(timeSelect, { target: { value: "10:00" } });
117    fireEvent.change(durationSelect, { target: { value: "2" } });
118    fireEvent.change(numberPlateInput, { target: { value: "ABC123" } });
119
120    // Submit the form
121    fireEvent.click(submitButton);
122
123    // Wait for the success message
124    await waitFor(() => {
125      | expect(screen.getByText(/slot 42 has been assigned to you/i)).toBeInTheDocument();
126    });
127  });
128
129 /**
130 * Test Case 5: Handle unauthorized access during form submission
131 * Simulate an unauthorized error and ensure the user is redirected to the login page.
132 */
133 test("redirects to login on unauthorized error", async () => {
134   // Mock unauthorized error
135   axios.post.mockRejectedValueOnce({ response: { status: 401 } });
136
137   renderWithProviders();
138
139   const dateInput = screen.getByText(/pick a date/i);
140   const timeSelect = screen.getByRole("combobox", { name: /select time/i });
141   const durationSelect = screen.getByRole("combobox", { name: /duration/i });
142   const numberPlateInput = screen.getByLabelText(/vehicle number plate/i);
143   const submitButton = screen.getByRole("button", { name: /book parking slot/i });
144
145   // Simulate user input
146   fireEvent.click(dateInput);
147   fireEvent.click(screen.getByText(/1\|1\|2025/i)); // Select a future date
148   fireEvent.change(timeSelect, { target: { value: "10:00" } });
149   fireEvent.change(durationSelect, { target: { value: "2" } });
150   fireEvent.change(numberPlateInput, { target: { value: "ABC123" } });
151
152   // Submit the form
153   fireEvent.click(submitButton);
154
155   // Wait for the error message and redirection
156   await waitFor(() => {
157     | expect(screen.getByText(/please login to book a parking slot/i)).toBeInTheDocument();
158   });
159 });
160 });

```

## 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.

PS E:\se project\G29-PARKit>

```

## 5. Mail Transporter :

```
1 ✓ import nodemailer from "nodemailer";
2   import { transporter } from "../../../../backend/utils/mailTransporter.js"; // Adjust the import based on the actual path
3
4   // Mock nodemailer
5   jest.mock("nodemailer");
6
7   ✓ describe("Email Transporter", () => {
8     let sendMailMock;
9     ...
10    ✓ beforeAll(() => {
11      // Create a mock for the sendMail method
12      sendMailMock = jest.fn().mockResolvedValue({ success: true });
13      nodemailer.createTransport.mockReturnValue({ sendMail: sendMailMock });
14    });
15
16    ✓ test("should create a transporter with correct config", () => {
17      // Check if the transporter is created with the correct configuration
18      expect(nodemailer.createTransport).toHaveBeenCalledWith({
19        service: "gmail",
20        auth: {
21          user: process.env.EMAIL_USER,
22          pass: process.env.EMAIL_PASS,
23        },
24      });
25    });
26
27    ✓ test("should send an email with correct data", async () => {
28      // Sample email data
29      const mailOptions = {
30        from: "test@example.com",
31        to: "recipient@example.com",
32        subject: "Test Email",
33        text: "This is a test email",
34      };
35
36      // Call the sendMail function
37      await transporter.sendMail(mailOptions);
38
39      // Check if sendMail was called with the correct email data
40      expect(sendMailMock).toHaveBeenCalledWith(mailOptions);
41      expect(sendMailMock).toHaveBeenCalledTimes(1);
42    });
43
44    ✓ test("should handle sendMail errors", async () => {
45      // Mock sendMail to throw an error
46      sendMailMock.mockRejectedValue(new Error("Email sending failed"));
47
48      // Sample email data
49      const mailOptions = {
50        from: "test@example.com",
51        to: "recipient@example.com",
52        subject: "Test Email",
53        text: "This is a test email",
54      };
55
56      try {
57        // Call the sendMail function and expect it to throw
58        await transporter.sendMail(mailOptions);
59      } catch (error) {
60        // Check that the error was handled correctly
61        expect(error.message).toBe("Email sending failed");
62      }
63    });
64  });


```

## OUTPUT :

The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the command `npm run test:file tests/backend/utils/mailTransporter.test.js` and its execution results. The results show 3 passed tests and 0 snapshots taken. The terminal also shows the test suite summary: 1 passed, 1 total test, and 3 total tests.

```
PS E:\se project\G29-PARKit> npm run test:file tests/backend/utils/mailTransporter.test.js
> G29-PARKit@1.0.0 test:file
> jest tests/backend/utils/mailTransporter.test.js

PASS  tests/backend/utils/mailTransporter.test.js
  Email Transporter
    ✓ should create a transporter with correct config (2 ms)
    ✓ should send an email with correct data (1 ms)
    ✓ should handle sendMail errors (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:  0 total
Time:        0.523 s, estimated 1 s
Ran all test suites matching /tests\\backend\\utils\\mailTransporter.test.js/i.
PS E:\se project\G29-PARKit>
```

## 6. Foegot Password :

```
1 import { render, screen, fireEvent, waitFor } from "@testing-library/react";
2 import { ForgotPassword } from "../../../../../frontend/src/components/ForgotPassword.jsx"; // adjust the import as necessary
3 import axios from "axios";
4 import { useToast } from "@/hooks/use-toast";
5
6 // Mocking axios and toast
7 jest.mock("axios");
8 jest.mock("@/hooks/use-toast", () => ({
9   useToast: jest.fn(),
10 }));
11
12 describe("ForgotPassword Component", () => {
13   let toast;
14
15   beforeEach(() => {
16     toast = {
17       toast: jest.fn(),
18     };
19     useToast.mockReturnValue(toast);
20   });
21
22   test("renders email form and submits email", async () => {
23     render(<ForgotPassword />);
24
25     const emailInput = screen.getByPlaceholderText("Enter your email");
26     const submitButton = screen.getByRole("button", { name: /send reset code/i });
27
28     // Simulate entering email
29     fireEvent.change(emailInput, { target: { value: "test@example.com" } });
30
31     // Simulate form submission
32     fireEvent.click(submitButton);
33
34     // Mock axios to resolve on successful submission
35     axios.post.mockResolvedValueOnce({
36       data: { success: true },
37     });
38   });
39
40   test("shows success toast on successful submission", () => {
41     render(<ForgotPassword />);
42
43     const emailInput = screen.getByPlaceholderText("Enter your email");
44     const submitButton = screen.getByRole("button", { name: /send reset code/i });
45
46     // Simulate entering email
47     fireEvent.change(emailInput, { target: { value: "test@example.com" } });
48
49     // Simulate form submission
50     fireEvent.click(submitButton);
51
52     // Mock axios to resolve on successful submission
53     axios.post.mockResolvedValueOnce({
54       data: { success: true },
55     });
56
57     // Check if toast was created with the correct message
58     expect(toast.toast).toHaveBeenCalledWith("Email sent! Check your inbox.");
59   });
60
61   test("shows error toast on failed submission", () => {
62     render(<ForgotPassword />);
63
64     const emailInput = screen.getByPlaceholderText("Enter your email");
65     const submitButton = screen.getByRole("button", { name: /send reset code/i });
66
67     // Simulate entering email
68     fireEvent.change(emailInput, { target: { value: "" } });
69
70     // Simulate form submission
71     fireEvent.click(submitButton);
72
73     // Mock axios to reject the request
74     axios.post.mockRejectedValueOnce({
75       error: { message: "Email not found" },
76     });
77
78     // Check if toast was created with the correct error message
79     expect(toast.toast).toHaveBeenCalledWith("Error: Email not found");
80   });
81
82   test("clears toast after successful submission", () => {
83     render(<ForgotPassword />);
84
85     const emailInput = screen.getByPlaceholderText("Enter your email");
86     const submitButton = screen.getByRole("button", { name: /send reset code/i });
87
88     // Simulate entering email
89     fireEvent.change(emailInput, { target: { value: "test@example.com" } });
90
91     // Simulate form submission
92     fireEvent.click(submitButton);
93
94     // Mock axios to resolve on successful submission
95     axios.post.mockResolvedValueOnce({
96       data: { success: true },
97     });
98
99     // Wait for the toast to appear
100    await waitFor(() => toast.toast);
101
102    // Clear the toast
103    toast.clear();
104
105    // Check if toast is cleared
106    expect(toast.toast).not.toHaveBeenCalled();
107  });
108});
```

```

38      await waitFor(() => {
39        expect(toast.toast).toHaveBeenCalledWith({
40          title: "Success",
41          description: "Verification code sent to your email",
42        });
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

```

```

107    await waitFor(() => {
108      expect(toast.toast).toHaveBeenCalledWith({
109        title: "Error",
110        description: "Invalid verification code",
111        variant: "destructive",
112      });
113    });
114  });
115
116 test("renders new password form and submits new password", async () => {
117   render(<ForgotPassword />);
118
119   // Switch to step 3 (New Password form)
120   fireEvent.click(screen.getByRole("button", { name: /send reset code/i }));
121   fireEvent.click(screen.getByRole("button", { name: /verify code/i }));
122
123   const passwordInput = screen.getByPlaceholderText("New password");
124   const confirmPasswordInput = screen.getByPlaceholderText("Confirm new password");
125   const submitButton = screen.getByRole("button", { name: /reset password/i });
126
127   // Simulate entering new password
128   fireEvent.change(passwordInput, { target: { value: "NewPass123" } });
129   fireEvent.change(confirmPasswordInput, { target: { value: "NewPass123" } });
130
131   // Simulate form submission
132   fireEvent.click(submitButton);
133
134   // Mock axios to resolve on successful password reset
135   axios.post.mockResolvedValueOnce({
136     data: { success: true },
137   });
138
139   await waitFor(() => {
140     expect(toast.toast).toHaveBeenCalledWith({
141       title: "Success",
142       description: "Password reset successful",
143     });
144   });
145 });
146
147 test("displays error on password mismatch", async () => {
148   render(<ForgotPassword />);
149
150   // Switch to step 3 (New Password form)
151   fireEvent.click(screen.getByRole("button", { name: /send reset code/i }));
152   fireEvent.click(screen.getByRole("button", { name: /verify code/i }));
153
154   const passwordInput = screen.getByPlaceholderText("New password");
155   const confirmPasswordInput = screen.getByPlaceholderText("Confirm new password");
156
157   // Simulate entering mismatched passwords
158   fireEvent.change(passwordInput, { target: { value: "NewPass123" } });
159   fireEvent.change(confirmPasswordInput, { target: { value: "WrongPass123" } });
160
161   // Simulate form submission
162   fireEvent.click(screen.getByRole("button", { name: /reset password/i }));
163
164   await waitFor(() => {
165     expect(screen.getByText(/passwords don't match/i)).toBeInTheDocument();
166   });
167 });
168 });

```

## OUTPUT :

```

o 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.

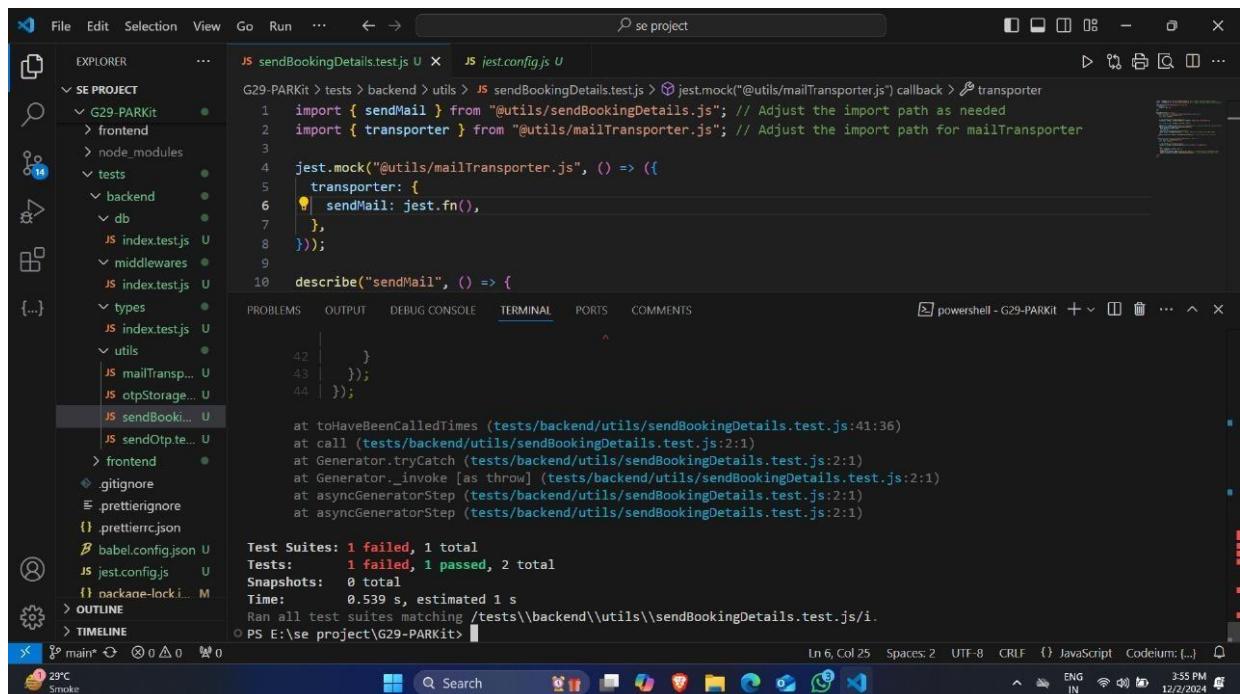
o PS E:\se project\G29-PARKit>

```

## 7. SendBookingDetails :

```
1 import { sendMail } from "../../backend/utils/sendBookingDetails.js"; // Adjust the import path as needed
2 import { transporter } from "../../backend/utils/mailTransporter.js"; // Adjust the import path for mailTransporter
3
4 jest.mock("../../backend/mailTransporter", () => ({
5   transporter: {
6     sendMail: jest.fn(),
7   },
8 });
9
10 describe("sendMail", () => {
11   test("should send a confirmation email successfully", async () => {
12     const receiver = "test@example.com";
13     const otp = "123456";
14
15     // Mock sendMail to resolve successfully
16     transporter.sendMail.mockResolvedValue({ response: "Email sent successfully" });
17
18     const result = await sendMail({ receiver, otp });
19
20     expect(result.response).toBe("Email sent successfully"); // Ensure the email was sent successfully
21     expect(transporter.sendMail).toHaveBeenCalledWith({
22       from: process.env.EMAIL_USER,
23       to: receiver,
24       subject: "Parking Reservation Confirmation",
25       html: expect.stringContaining(otp), // Ensure the OTP is included in the HTML content
26     });
27     expect(transporter.sendMail).toHaveBeenCalledTimes(1); // Ensure sendMail was called once
28   });
29
30   test("should throw an error if email sending fails", async () => {
31     const receiver = "test@example.com";
32     const otp = "123456";
33
34     // Mock sendMail to simulate an error
35     transporter.sendMail.mockRejectedValue(new Error("Failed to send email"));
36
37     try {
38       await sendMail({ receiver, otp });
39     } catch (error) {
40       expect(error.message).toBe("Failed to send email"); // Ensure the error is handled properly
41     }
42   }
43 });
44});
```

## OUTPUT :



The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under "SE PROJECT". The "tests" folder contains "backend", "db", "middlewares", and "utils" subfolders. Inside "utils", there are files: "index.test.js", "mailTransp...", "otpStorage...", "sendBooki...", "sendOptTe...", and ".gitignore".
- Terminal Tab:** Displays the command "PS E:\se project\G29-PARKit>".
- Status Bar:** Shows "In 6, Col 25" and "12/2/2024 2:55 PM".
- Output:** The terminal output shows the test results:

```
Test Suites: 1 failed, 1 total
Tests:    1 failed, 1 passed, 2 total
Snapshots: 0 total
Time:    0.539 s, estimated 1 s
Ran all test suites matching /tests\\backend\\utils\\sendBookingDetails.test.js.i.
```

## 8. OTPStorage :

```
1 import { storeOTP, verifyOTP } from "../../backend/utils/otpstorage.js"; // Adjust the import path
2
3 jest.useFakeTimers(); // Use fake timers to mock Date.now() behavior
4
5 describe("OTP Service", () => {
6   const email = "test@example.com";
7   const otp = "123456";
8
9   beforeEach(() => {
10     jest.clearAllMocks(); // Clear mocks before each test
11   });
12
13   test("should store OTP and validate correctly", () => {
14     // Store the OTP
15     storeOTP(email, otp);
16
17     // Simulate immediate OTP verification
18     const isValid = verifyOTP(email, otp);
19
20     expect(isValid).toBe(true); // The OTP should be valid
21   });
22
23   test("should return false if OTP is incorrect", () => {
24     // Store the OTP
25     storeOTP(email, otp);
26
27     // Try to verify with an incorrect OTP
28     const isValid = verifyOTP(email, "wrongOTP");
29
30     expect(isValid).toBe(false); // The OTP should be invalid
31   });
32 }
```

```

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 :

The screenshot shows a code editor interface with a terminal window open at the bottom. The terminal output is as follows:

```

ReferenceError: otpStore is not defined
      // The OTP should be deleted after successful verification
      expect(isValid).toBe(true);
      expect(otpStore.has(email)).toBe(false); // OTP should no longer exist in the store
    );
  });

Test Suites: 1 Failed, 1 total
Tests:       2 Failed, 3 passed, 5 total
Snapshots:  0 total
Time:        0.658 s
Ran all test suites matching /tests\\backend\\utils\\otpstorage.test.js/1.

PS E:\se project\G29-PARKIT>

```

The code editor's sidebar shows the file structure of the project, including files like otpstorage.test.js, otpstorage.js, and various configuration and ignore files.