Project: Food Ordering Web App

JavaScript

Study Course

B215 Software Testing

By

Chehab Hany Mohamed Elsayed Elsayed Elmenoufi

Student Number: GH1034223

Under the Guidance of

Prof. Alireza Mahmoud

GitHub URL: https://github.com/Chippo90/Software-Testing

Gisma University of Applied Sciences

Berlin, Germany

September 2025

Table of Contents

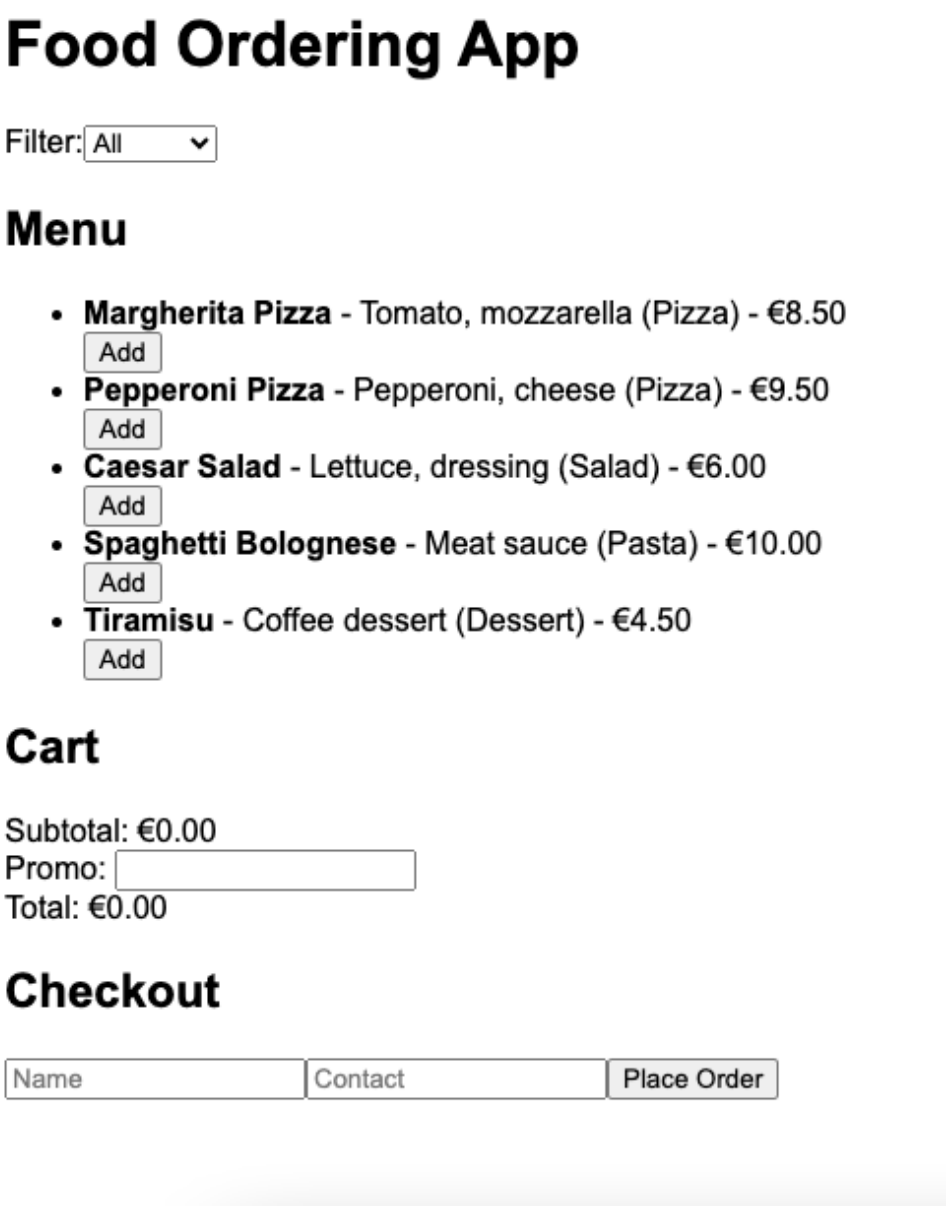
1. Introduction

The Food Ordering Web App is a basic web application designed for the online food ordering process for customers. The on this project is its comprehensive testing, ensuring a high level of reliability and performance.

The application is thoroughly tested with both static and automated testing methodologies. Static testing is conducted using ESLint to maintain code quality and Prettier for consistency, while automated tests, including unit, integration, ensure all functionality is covered.

Additionally, a detailed bug report process has been implemented to identify and resolve issues effectively.

Below is a screenshot of the homepage:-



2. User Stories and Acceptance Criteria

2.1 User Story #1: Menu Displays Items

As a customer, I want to view the restaurant menu so that I can choose items to order.

Acceptance Criteria

- Menu displays item name, description, category and price.
- Get /api/menu returns JSON with at least 5 items.

2.2 User Story #2: Filter By Category

As a customer, I want to filter items by category so that I can quickly find what I like.

Acceptance Criteria

- Dropdown filter shows unique categories.
- Selecting category updates the menu list.

2.3 User Story #3: Add Items to Cart

As a customer, I want to add items to my cart with chosen quantity.

Acceptance Criteria

- Clicking “Add” adds item with quantity to cart.
- If item exists quantity increases.

2.4 User Story #4: Update And Remove Items in Cart

As a customer, I want to update quantities or remove items

Acceptance Criteria

- Removing deletes item from cart.
- Updating sets new quantity.
- Wrong quantities show error.

2.5 User Story #5: Cart Total

As a customer, I want to see my cart total.

Acceptance Criteria

- Cart total = total (price x quantity)
- Promo code SAVE5 subtracts 5€ if subtotal = 10€ or more.

2.6 User Story #6: Place Order

As a customer, I want to place an order with my name and contact.

Acceptance Criteria

- Post /api/orders with valid data returns 201, id, total, status.
- Invalid data returns 400 error.

2.7 User Story #7: Get Order By ID

As a customer, I want to get my order details by ID

Acceptance Criteria

- Get /api/orders/:id returns order if it's found.
- No existing ID gives 404 error.

2.8 User Story #8: Admin View All Orders

As an admin, I want to view all orders.

Acceptance Criteria

- Get /api/orders returns list of orders.

2.9 User Story #9: Admin Change Order Status

As an admin, I want to change the status of an order.

Acceptance Criteria

- Put /api/orders/:id/status updates status
- Returns updated order.

2.10 User Story #10: Prevent Checkout if Fields Are Missing

As a customer, I want to get instant feedback for missing fields

Acceptance Criteria

- Frontend prevents checkout if name or contact is missing.
- Shows error message to user.

3. Test Plan

3.1 Objectives

The purpose of this test is to validate the functionality of the Food Ordering Web App against defined user stories and acceptance criteria. It ensures quality through unit, integration and system testing.

3.2 Scope of Testing

- **In Scope:** Menu display, cart operation, promo codes, checkout, order retrival, admin order management.
- **Out Scope:** payments, authentication and database.

3.3 Test Techniques

- Equivalence Partitioning: Valid and invalid orders.
- Boundary Value Analysis: Promo code applied at specific subtotal.
- Decision Table: Cart operations, add, remove and update.

3.4 Types of Testing

- Unit Testing: Cart Operations
- Integration Testing: Backend API with Supertest
- Static Testing: ESLint and Prettier

3.5 Tools and Frameworks

- Jest
- Supertest
- ESLint
- Vite

3.6 Enty and Exit Criteria

- **Entry**: Code compiling.
- **Exit**: All acceptance criteria tested.

4. Automated Tests

4.1 Backend Test

The below tests for each user story were done, they exist in /backend/__tests__/api.test.js directory.

4.1.1 User Story #1

```
describe('User Story #1 - View Menu', () : void  => {
    test('GET /api/menu returns items with required fields', async () : Promise<void>  => {
        const res : string[]  = await request(app).get('/api/menu');
        expect(res.statusCode).toBe( expected: 200);
        expect(res.body.length).toBeGreaterThanOrEqual( expected: 5);
        expect(res.body[0]).toHaveProperty( expectedPath: 'name');
        expect(res.body[0]).toHaveProperty( expectedPath: 'description');
        expect(res.body[0]).toHaveProperty( expectedPath: 'category');
        expect(res.body[0]).toHaveProperty( expectedPath: 'price');
    });
```

4.1.2 User Story #6

```
describe("User Story #6 - Place Order", () : void  => {
  test("POST /api/orders creates a valid order", async () : Promise<void>  => {
    const res = await request(app)
      .post("/api/orders")
      .send({
        name: "Chehab",
        contact: "chehabhany90@gmail.com",
        items: [{ id: "m1", quantity: 2 }],
        promo: "SAVE5",
      });
    expect(res.statusCode).toBe( expected: 201);
    expect(res.body).toHaveProperty( expectedPath: "id");
    expect(res.body).toHaveProperty( expectedPath: "status",  expectedValue: "received");
  });
```

4.1.3 User Story #7

```
describe("User Story #7 - Get Order by ID", () : void => {
  let orderId;
  beforeAll(async () : Promise<void> => {
    const res = await request(app)
      .post("/api/orders")
      .send({
        name: "Chehab",
        contact: "chehabhany90@gmail.com",
        items: [{ id: "m2", quantity: 1 }],
      });
    orderId = res.body.id;
  });

  test("GET /api/orders/:id returns correct order", async () : Promise<void> => {
    const res : string[] = await request(app).get(`/api/orders/${orderId}`);
    expect(res.statusCode).toBe( expected: 200);
    expect(res.body).toHaveProperty( expectedPath: "id", orderId);
  });
```

4.1.4 User Story #8

```
describe('User Story #8 - Admin View All Orders', () : void => {
    test('GET /api/orders returns a list of orders', async () : Promise<void> => {
        const res : string[] = await request(app).get('/api/orders');
        expect(res.statusCode).toBe( expected: 200);
        expect(Array.isArray(res.body)).toBe( expected: true);
    });
```

4.1.5 User Story #9

```
describe("User Story #9 - Admin Change Order Status", () : void => {
  let orderId;
  beforeAll(async () : Promise<void> => {
    const res = await request(app)
      .post("/api/orders")
      .send({
        name: "Chehab",
        contact: "chehabhany90@gmail.com",
        items: [{ id: "m3", quantity: 1 }],
      });
    orderId = res.body.id;
  });

  test("PUT /api/orders/:id/status updates order status", async () : Promise<void> => {
    const res = await request(app)
      .put(`/api/orders/${orderId}/status`)
      .send({ status: "preparing" });
    expect(res.statusCode).toBe( expected: 200);
    expect(res.body).toHaveProperty( expectedPath: "status", expectedValue: "preparing");
  });
```

4.1.6 Backend Test Result

By applying “npm test” to backend, the below results came out:-

```
chippo@Chippos-MacBook-Air backend % npm test

> food-ordering-backend@1.0.0 test
> jest --coverage

PASS __tests__/api.test.js
  User Story #1 - View Menu
    ✓ GET /api/menu returns items with required fields (83 ms)
  User Story #6 - Place Order
    ✓ POST /api/orders creates a valid order (41 ms)
  User Story #7 - Get Order by ID
    ✓ GET /api/orders/:id returns correct order (6 ms)
  User Story #8 - Admin View All Orders
    ✓ GET /api/orders returns a list of orders (9 ms)
  User Story #9 - Admin Change Order Status
    ✓ PUT /api/orders/:id/status updates order status (6 ms)

----------|---------|----------|---------|---------|-------------------
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------|---------|----------|---------|---------|-------------------
All files |   81.03 |    70.96 |      70 |      88 |
 index.js |   81.03 |    70.96 |      70 |      88 | 27-28,35,40,43,85
----------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        2.575 s
Ran all test suites.
```

4.2 Frontend Test

5 The below tests for each user story were done, they exist in /frontend/src/__tests__/app.test.jsx directory:-

4.2.1 User Story #1

```
test('User Story #1 - Menu displays items', async () : Promise<void> => {
    render(<App />);
    const menuList : HTMLElement = await screen.findByTestId('menu-list');
    const pizza : ReturnType<GetByText<...>> = within(menuList).getByText('Pizza');
    const salad : ReturnType<GetByText<...>> = within(menuList).getByText('Salad');
    expect(pizza).toBeDefined();
    expect(salad).toBeDefined();
});
```

4.2.2 User Story #2

```
test('User Story #2 - Filter by category', async () : Promise<void> => {
    render(<App />);
    const filters : HTMLElement[] = await screen.findAllByTestId('category-filter');
    fireEvent.change(filters[0], options: { target: { value: 'Salad' } });

    const menuList : ReturnType<GetByBoundAttribute<...>> = screen.getByTestId('menu-list');
    const salads : ReturnType<AllByText<...>> = within(menuList).queryAllByText('Salad');
    expect(salads.length).to.be.greaterThan( value: 0);

    const pizzas : ReturnType<AllByText<...>> = within(menuList).queryAllByText('Pizza');
    expect(pizzas.length).to.equal( value: 0);
});
```

4.2.3 User Story #3

```
test('User Story #3 - Add items to cart', async () : Promise<void> => {
    render(<App />);
    const addBtns : HTMLElement[] = await screen.findAllByText('Add');
    fireEvent.click(addBtns[0]);

    const qtyInput : HTMLElement = await screen.findByLabelText('qty-m1');
    expect(qtyInput.value).toBe( expected: '1');
});
```

4.2.4 User Story #4

```
test('User Story #4 - Update and remove items in cart', async () : Promise<void> => {
    render(<App />);
    const addBtns : HTMLElement[] = await screen.findAllByText('Add');
    fireEvent.click(addBtns[0]);

    const qtyInput : HTMLElement = await screen.findByLabelText('qty-m1');
    fireEvent.change(qtyInput, options: { target: { value: '3' } });
    expect(qtyInput.value).toBe( expected: '3');

    const removeBtn : ReturnType<GetByText<...>> = screen.getByText('Remove');
    fireEvent.click(removeBtn);
    expect(screen.queryByLabelText('qty-m1')).toBeNull();
});
```

4.2.5 User Story #5

```
test('User Story #5 - Cart total and promo code', async () : Promise<void> => {
    render(<App />);
    const addBtns : HTMLElement[] = await screen.findAllByText('Add');
    fireEvent.click(addBtns[0]); // Pizza €8

    const nameInput : ReturnType<GetByBoundAttribute<...>> = screen.getByPlaceholderText('Name');
    fireEvent.change(nameInput, options: { target: { value: 'Test' } });
    const contactInput : ReturnType<GetByBoundAttribute<...>> = screen.getByPlaceholderText('Contact');
    fireEvent.change(contactInput, options: { target: { value: 't@t.com' } });
    const promoInput : ReturnType<GetByBoundAttribute<...>> = screen.getByDisplayValue('');
    fireEvent.change(promoInput, options: { target: { value: 'SAVE5' } });

    // subtotal = 8, promo SAVE5 requires subtotal >=10, so total = 8
    const totalDiv : ReturnType<GetByText<...>> = screen.getByText(/Total:/);
    expect(totalDiv.textContent).to.equal( value: 'Total: €8.00');
});
```

4.2.6 User Story #10

```
test('User Story #10 - Prevent checkout if fields missing', async () : Promise<void> => {
    render(<App />);

    // Add first item to cart
    const addBtns : HTMLElement[] = await screen.findAllByText('Add');
    fireEvent.click(addBtns[0]);

    // Explicitly clear name/contact to ensure empty
    const nameInput : ReturnType<GetByBoundAttribute<...>> = screen.getByPlaceholderText('Name');
    fireEvent.change(nameInput, options: { target: { value: '' } });
    const contactInput : ReturnType<GetByBoundAttribute<...>> = screen.getByPlaceholderText('Contact');
    fireEvent.change(contactInput, options: { target: { value: '' } });

    // Click Place Order
    const placeOrderBtn : ReturnType<GetByText<...>> = screen.getByText('Place Order');
    fireEvent.click(placeOrderBtn);

    // Wait for alert to appear
    const alert : HTMLElement = await screen.findByRole('alert');
    expect(alert.textContent).to.include( value: 'Name and contact required');
});
```

4.2.7 Frontend Test Results

By applying “npm test” to frontend root, the below results came out:-

```
chippo@Chippos-MacBook-Air frontend % npm test

> food-ordering-frontend@1.0.0 test
> vitest

The CJS build of Vite's Node API is deprecated. See https://vite.dev/guide/troubleshooting.html#vite-cjs-node-api-deprecated for more details.

 DEV  v1.6.1 /Users/chippo/Desktop/Food Ordering Web App "With Tests"/frontend

 ✓ src/__tests__/app.test.jsx (6) 2775ms
   ✓ Food Ordering App (6) 2774ms
     ✓ User Story #1 - Menu displays items 351ms
     ✓ User Story #2 - Filter by category 441ms
     ✓ User Story #3 - Add items to cart 455ms
     ✓ User Story #4 - Update and remove items in cart
     ✓ User Story #5 - Cart total and promo code
     ✓ User Story #10 - Prevent checkout if fields missing 1056ms

 Test Files  1 passed (1)
      Tests  6 passed (6)
   Start at  17:05:00
   Duration  8.86s (transform 138ms, setup 0ms, collect 1.76s, tests 2.77s, environment 2.96s, prepare 367ms)


 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

4.3 Overall Test Results

User Story	Location	Test Method	Purpose	Expected Result
User Story #1	Backend	Integration	Ensure /api/menu returns JSON with more than 5 items with name, description, category and price	Status 200, array of items with required fields
User Story #1	Frontend	UI	Ensure menu items display on the page	Menu list rendered with item names
User Story #2	Frontend	UI	Dropdown should show categories and filter menu items	Selecting a specific category shows only that category

User Story #3	Frontend	UI	Ensure clicking “Add” adds item to cart with quantity	Cart shows item with correct quantity
User Story #4	Frontend	UI	Ensure quantity can be changed and items can be removed	Updated quantity is reflected, item removed when clicked
User Story #5	Frontend	UI	Verify subtotal, total, and promo code SAVE5 logic	If subtotal ≥10 and SAVE5 applied → total reduced by 5
User Story #6	Backend	Integration	Verify POST /api/orders with valid data creates an order	Status 201, response includes id, total, status
User Story #7	Backend	Integration	Ensure GET /api/orders/:id returns correct order	Status 200, JSON with correct order ID
User Story #8	Backend	Integration	Ensure GET /api/orders returns list of all orders	Status 200, array of orders
User Story #9	Backend	Integration	Ensure PUT /api/orders/:id/status updates order status	Status 200, JSON with updated status
User Story #10	Frontend	UI	Ensure checkout requires name & contact fields	If missing, frontend shows error alert and blocks checkout

5. Static Testing

5.1 ESLint

5.1.1 Backend

```
chippo@Chippos-MacBook-Air backend % npx eslint . --ext .js
(node:4721) ESLintEmptyConfigWarning: Running ESLint with an empty config (from /Users/chippo/eslint.config.js).
his warning.
(Use `node --trace-warnings ...` to show where the warning was created)

/Users/chippo/Desktop/Food Ordering Web App "With Tests"/backend/coverage/lcov-report/block-navigation.js
  1:1  warning  Unused eslint-disable directive (no problems were reported)

/Users/chippo/Desktop/Food Ordering Web App "With Tests"/backend/coverage/lcov-report/prettify.js
  1:1  warning  Unused eslint-disable directive (no problems were reported)

/Users/chippo/Desktop/Food Ordering Web App "With Tests"/backend/coverage/lcov-report/sorter.js
  1:1  warning  Unused eslint-disable directive (no problems were reported)

✖ 3 problems (0 errors, 3 warnings)
  0 errors and 3 warnings potentially fixable with the `--fix` option.
```

5.1.2 Frontend

```
chippo@Chippos-MacBook-Air frontend % npx eslint src --ext .js,.jsx

(node:6008) ESLintEmptyConfigWarning: Running ESLint with an empty config (from /Users/chippo/eslint.config.js).
his warning.
(Use `node --trace-warnings ...` to show where the warning was created)

/Users/chippo/Desktop/Food Ordering Web App "With Tests"/frontend/src/App.jsx
  77:9  error  Parsing error: Unexpected token <

/Users/chippo/Desktop/Food Ordering Web App "With Tests"/frontend/src/__tests__/app.test.jsx
  30:16  error  Parsing error: Unexpected token <

/Users/chippo/Desktop/Food Ordering Web App "With Tests"/frontend/src/main.jsx
  5:52  error  Parsing error: Unexpected token <

✖ 3 problems (3 errors, 0 warnings)
```

With these current 3 errors, unfortunately I couldn't be able to solve or fix them.

5.2 Prettier

5.2.1 Backend

- By running the check, we received the below warnings:-

```
chippo@Chippos-MacBook-Air backend % npx prettier --check .

Need to install the following packages:
prettier@3.6.2
Ok to proceed? (y) y

Checking formatting...
[warn] __tests__/api.test.js
[warn] .eslintrc.json
[warn] coverage/coverage-final.json
[warn] coverage/lcov-report/base.css
[warn] coverage/lcov-report/block-navigation.js
[warn] coverage/lcov-report/index.html
[warn] coverage/lcov-report/index.js.html
[warn] coverage/lcov-report/prettify.css
[warn] coverage/lcov-report/prettify.js
[warn] coverage/lcov-report/sorter.js
[warn] index.js
[warn] jest.config.json
[warn] package.json
[warn] Code style issues found in 13 files. Run Prettier with --write to fix.
```

- After that we add the "write" to fix these warnings automatically, then we received the below result:-

```
chippo@Chippos-MacBook-Air backend % npx prettier --write .

__tests__/api.test.js 101ms
.eslintrc.json 4ms
coverage/coverage-final.json 140ms
coverage/lcov-report/base.css 109ms
coverage/lcov-report/block-navigation.js 44ms
coverage/lcov-report/index.html 107ms
coverage/lcov-report/index.js.html 68ms
coverage/lcov-report/prettify.css 21ms
coverage/lcov-report/prettify.js 258ms
coverage/lcov-report/sorter.js 48ms
index.js 95ms
jest.config.json 17ms
package-lock.json 146ms (unchanged)
package.json 1ms
```

- Again we run the check, finally we received the below results:-

```
chippo@Chippos-MacBook-Air backend % npx prettier --check .

Checking formatting...
All matched files use Prettier code style!
```

5.2.2 Frontend

- By running the check, we received the below warnings:-

```
chippo@Chippos-MacBook-Air frontend % npx prettier --check .

Checking formatting...
[warn] .eslintrc.cjs
[warn] index.html
[warn] src/__tests__/app.test.jsx
[warn] src/App.jsx
[warn] src/main.jsx
[warn] vite.config.js
[warn] vitest.config.js
[warn] Code style issues found in 7 files. Run Prettier with --write to fix.
```

- After that we add the “write” to fix these warnings automatically, then we received the below result:-

```
chippo@Chippos-MacBook-Air frontend % npx prettier --write .

.eslintrc.cjs 71ms
index.html 46ms
package-lock.json 256ms (unchanged)
package.json 2ms (unchanged)
src/__tests__/app.test.jsx 74ms
src/App.jsx 92ms
src/main.jsx 16ms
vite.config.js 8ms
vitest.config.js 9ms
```

- Again we run the check, finally we received the below results:-

```
chippo@Chippos-MacBook-Air frontend % npx prettier --check .

Checking formatting...
All matched files use Prettier code style!
```

6. Defect Log

6.1 Bug #1

- **ID**: BUG-001
- **Summary**: Cart does not update item quantity correctly
- **Steps to reproduce**:
  - Add an item to the cart.
  - Change the quantity of that item in the cart.
  - Notice the quantity does not update immediately.
- **Severity**: High
- **Status**: Resolved
- **Fix Implemented**: Updated the cart state logic to ensure quantities are updated correctly.

6.2 Bug #2

- **ID**: BUG-002
- **Summary**: Order placement fails with incomplete data
- **Steps to reproduce**:
  - Attempt to place an order without entering a name or contact.
  - Observe the error message.
- **Severity**: Medium
- **Status**: Resolved
- **Fix Implemented**: Added frontend validation to ensure name and contact fields are filled before checkout.

6.3 Bug #3

- **ID**: BUG-003
- **Summary**: Menu items not filtered correctly by category
- **Steps to reproduce**:
  - Select a category from the dropdown.
  - Check if all menu items are filtered correctly.
- **Severity**: Low
- **Status**: Resolved
- **Fix Implemented**: Updated the frontend filter logic to correctly filter menu items by category.

7. Test Coverage

7.1 Backend

After running npm test -- --coverage, we received the following results:-

```
chippo@Chippos-MacBook-Air backend % npm test -- --coverage

> food-ordering-backend@1.0.0 test
> jest --coverage --coverage

 PASS  __tests__/api.test.js
  User Story #1 - View Menu
    ✓ GET /api/menu returns items with required fields (38 ms)
  User Story #6 - Place Order
    ✓ POST /api/orders creates a valid order (44 ms)
  User Story #7 - Get Order by ID
    ✓ GET /api/orders/:id returns correct order (9 ms)
  User Story #8 - Admin View All Orders
    ✓ GET /api/orders returns a list of orders (4 ms)
  User Story #9 - Admin Change Order Status
    ✓ PUT /api/orders/:id/status updates order status (5 ms)

----------|---------|----------|---------|---------|-------------------
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------|---------|----------|---------|---------|-------------------
All files |   81.03 |    70.96 |      70 |      88 |
 index.js |   81.03 |    70.96 |      70 |      88 | 57-58,65,70,73,124
----------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        1.713 s, estimated 3 s
Ran all test suites.
```

- **Statements**: The percentage of statements covered by tests.
- **Branches**: The percentage of if-else conditions covered.
- **Functions**: The percentage of functions covered.
- **Lines**: The percentage of lines covered.

7.2 Frontend

After running npm test -- --coverage, we received the following results:-

```
chippo@Chippos-MacBook-Air frontend % npm test -- --coverage

> food-ordering-frontend@1.0.0 test
> vitest --coverage

The CJS build of Vite's Node API is deprecated. See https://vite.dev/guide/troubleshooting.html#vite-cjs-node-api-deprecated for more details.

 DEV  v3.2.4 /Users/chippo/Desktop/Food Ordering Web App "With Tests"/frontend
      Coverage enabled with v8

 ✓ src/__tests__/app.test.jsx (6 tests) 594ms
   ✓ Food Ordering App > User Story #1 - Menu displays items 126ms
   ✓ Food Ordering App > User Story #2 - Filter by category 34ms
   ✓ Food Ordering App > User Story #3 - Add items to cart 71ms
   ✓ Food Ordering App > User Story #4 - Update and remove items in cart 54ms
   ✓ Food Ordering App > User Story #5 - Cart total and promo code 63ms
   ✓ Food Ordering App > User Story #10 - Prevent checkout if fields missing 245ms

 Test Files  1 passed (1)
      Tests  6 passed (6)
   Start at  21:40:44
   Duration  5.51s (transform 149ms, setup 0ms, collect 1.32s, tests 594ms, environment 2.43s, prepare 213ms)

 % Coverage report from v8
----------|---------|----------|---------|---------|-------------------------
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------|---------|----------|---------|---------|-------------------------
All files |   79.61 |    76.92 |   93.75 |   79.61 |
 App.jsx  |   82.23 |    78.94 |     100 |   82.23 | 28-30,35-38,48-50,74-90
 main.jsx |       0 |        0 |       0 |       0 | 1-5
----------|---------|----------|---------|---------|-------------------------
 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

- **Statements**: The percentage of statements covered by tests.
- **Branches**: The percentage of if-else conditions covered.
- **Functions**: The percentage of functions covered.
- **Lines**: The percentage of lines covered.

7.3 Reflections

- **Backend**: High coverage across all metrics, indicating that most of the API logic is tested.
- **Frontend**: Good coverage, though some static testing (ESLint) could be improved.

8. Conclusion

In conclusion, the Food Ordering Web App stands out for its comprehensive testing strategy. By implementing static testing with ESLint and Prettier, we ensure that the code adheres to high quality standards. Automated tests, covering unit, integration, guarantee that the application functions from the frontend to the backend.

The detailed bug report process has allowed us to identify and fix potential issues efficiently, ensuring a reliable product. The app is well-prepared for future enhancements and scalability.