SWIFT SHOP

Project Report #2

https://github.com/stephenasuncionDEV/swift-shop

By Damandeep Singh, Ryuho Kikuchi, Cyrus Lum, and Stephen Asuncion

Contents

Overview

Swift Shop is an ecommerce website which sells various kind of electronics like laptops, desktops, keyboards etc. This website not only offers the quality products, but also offer the products at the best price. We are providing a simple website that is easy to navigate and offers great accessibility.

We made best use of some software which really helped us throughout the project cycle as given below:

- Figma.com - for designing the website
- Notion.so - for project management and scheduling
- Discord - for team discussion/meeting
- Google Docs, Slides and Sheets - for documenting and planning
- Github - for version source control
- Vercel – for deployment
- Lucid Chart - for creating data flow

Most of the things went as planned. However, we lacked at some points due to some circumstances. We were almost able to meet all deadlines except few situations. This project is also cost-free as we only used free services.

Working together as a team was really enjoyable. It not only helped us to learn new skills, but also taught many great lessons which will definitely help in the future projects

Software Development Life Cycle (SDLC)

We chose Agile methodology and used scrum as our agile framework. As predicted things did not go well as planned always, we needed to have some changes after each sprint. As agile offers flexibility of introducing changes to fix problems at any stage, it was easy for us to add or remove tings as required.

Talking about standouts, due to work schedules of team members, it was not possible to meet on a fixed day at the given time. We have to discuss sometimes trough discord, which allowed team members to response whenever they found time.

Using notion as a project management tool really helped keeping everyone up to date. It helped us visualize where we are and what's the next project. Moving towards the completion of project was really an exciting experience.

Implemented Features (High Level)

Commerce.js

- Customer Login/Logout
- Product Retrieval
- Category Retrieval
- Order Retrieval
- Cart System
- Shipping Information (Countries, Subdivisions, Rates)
- Checkout System
- Coupon Verification

Stripe

- Coupon Retrieval
- Payment Refund
- Customer Retrieval
- Payment Method

Application Features

Commerce.js

- ***Get and display products including their information:*** We used commerce API to get products images, their description, and prices. We get products using different calls to API like it includes call to API to get categories, products of a particular category and etc.

- ***Cart:*** We have implemented a cart system using commerce API which allows user to add products and remove products from it. It also shows the total amount to be paid by the user. Adding or removing products also change the corresponding total.

- ***Shipping Rates:*** Commerce.js provides a way to create/modify shipping rates. We used this feature by retrieving shipping rates from the API.

Application Features cont.

Stripe

- ***Customer Retrieval:*** We are using stripe API to get customer data. Our website's landing page shows the amount of guest customers. API calls to stripe could be found in the route's controller.

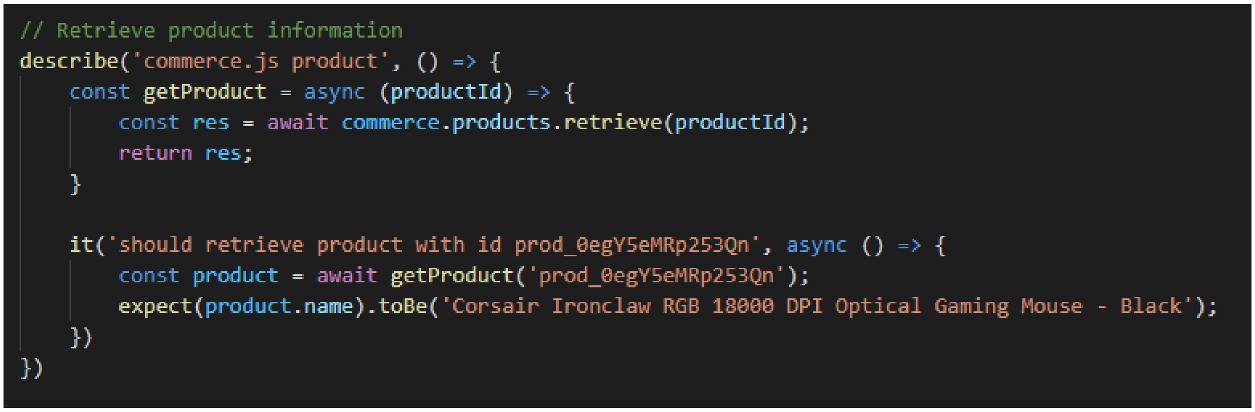
- ***Discount/Coupon Code:*** Customers have the ability to enter a discount/coupon code. We use stripe to get the lists of discount codes. API calls to stripe could be found in the route's controller.

- ***Refund:*** If the customers change mind later, they could request for refund by providing their charge id. They simply have to go to the payments page and enter their order's charge id to request a refund. The charge id will be provided by an administrator. API calls to stripe could be found in the route's controller.

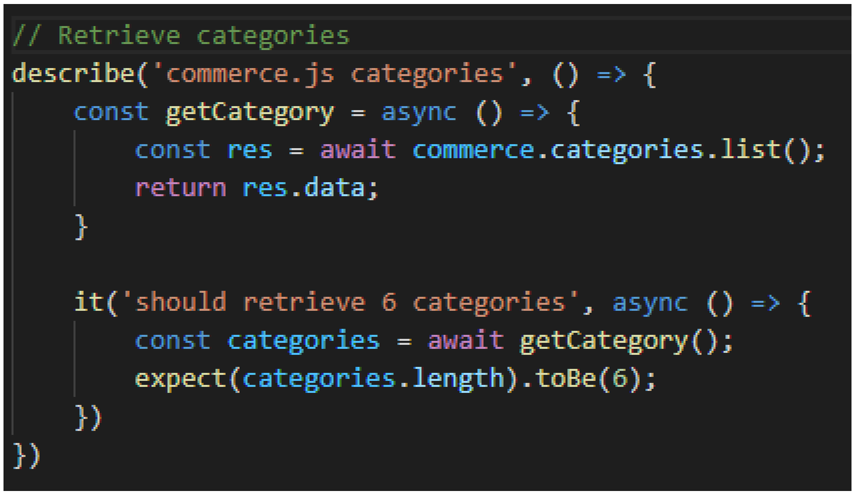
Testing

Unit Test

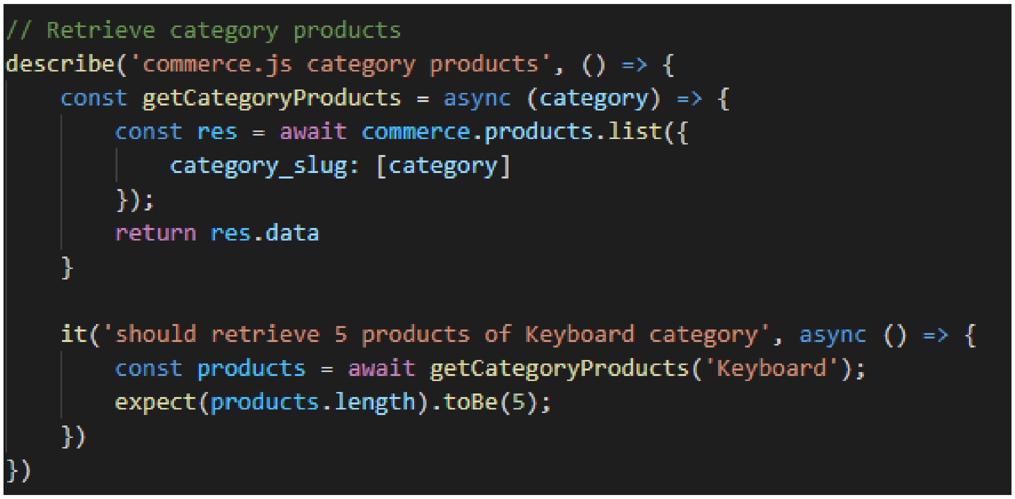
(Note: All functions that are being tested came from a provider)



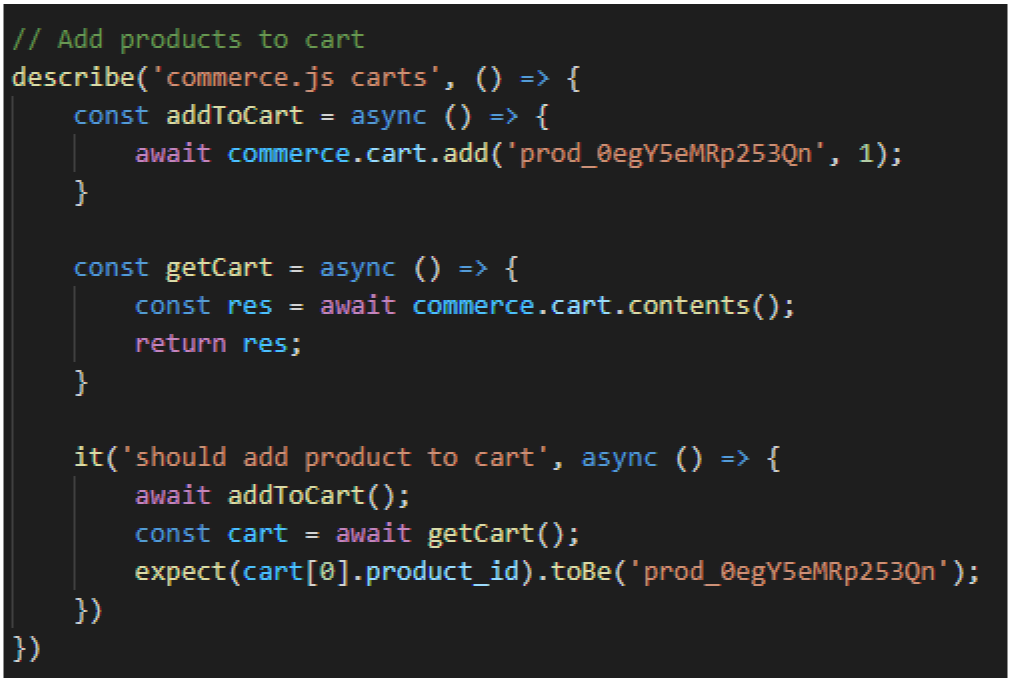
```
// Retrieve product information
describe('commerce.js product', () => {
    const getProduct = async (productId) => {
        const res = await commerce.products.retrieve(productId);
        return res;
    }

    it('should retrieve product with id prod_0egY5eMRp253Qn', async () => {
        const product = await getProduct('prod_0egY5eMRp253Qn');
        expect(product.name).toBe('Corsair Ironclaw RGB 18000 DPI Optical Gaming Mouse - Black');
    })
})
```



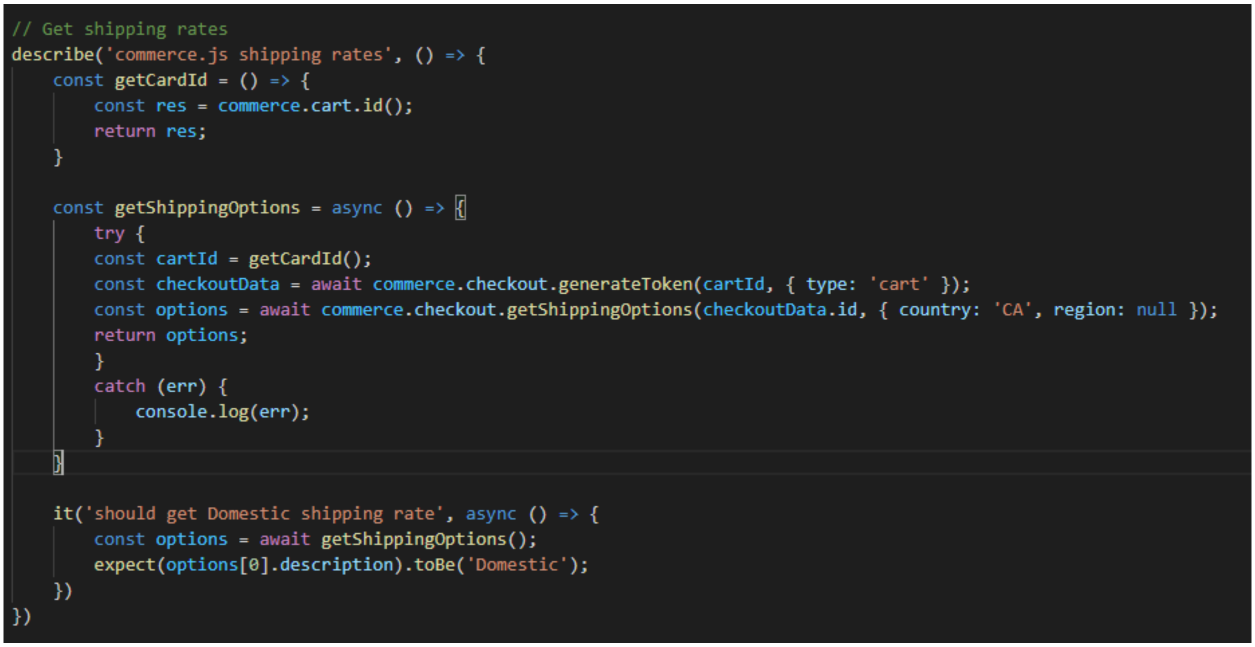
```
// Retrieve categories
describe('commerce.js categories', () => {
    const getCategory = async () => {
        const res = await commerce.categories.list();
        return res.data;
    }

    it('should retrieve 6 categories', async () => {
        const categories = await getCategory();
        expect(categories.length).toBe(6);
    })
})
```



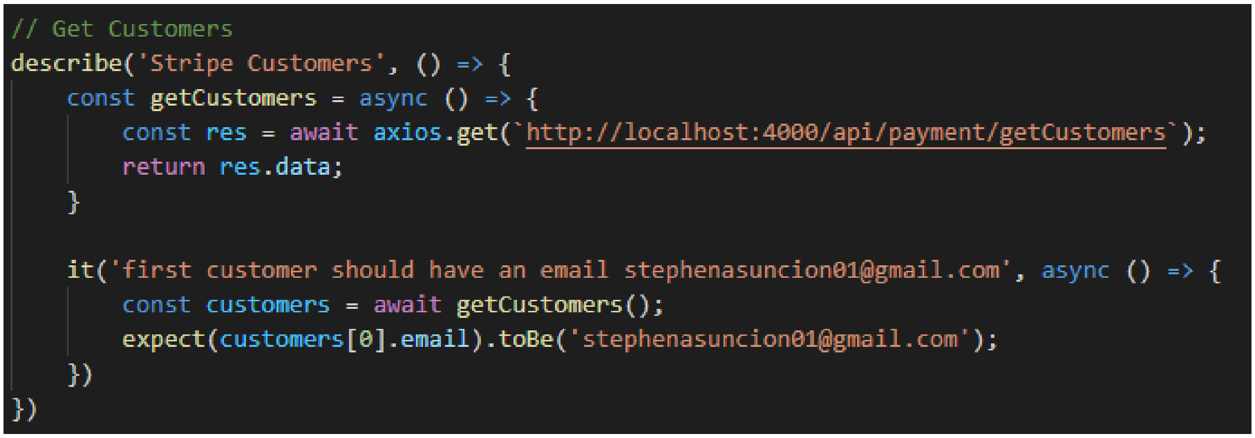
```
// Retrieve category products
describe('commerce.js category products', () => {
    const getCategoryProducts = async (category) => {
        const res = await commerce.products.list({
            category_slug: [category]
        });
        return res.data
    }

    it('should retrieve 5 products of Keyboard category', async () => {
        const products = await getCategoryProducts('Keyboard');
        expect(products.length).toBe(5);
    })
})
```



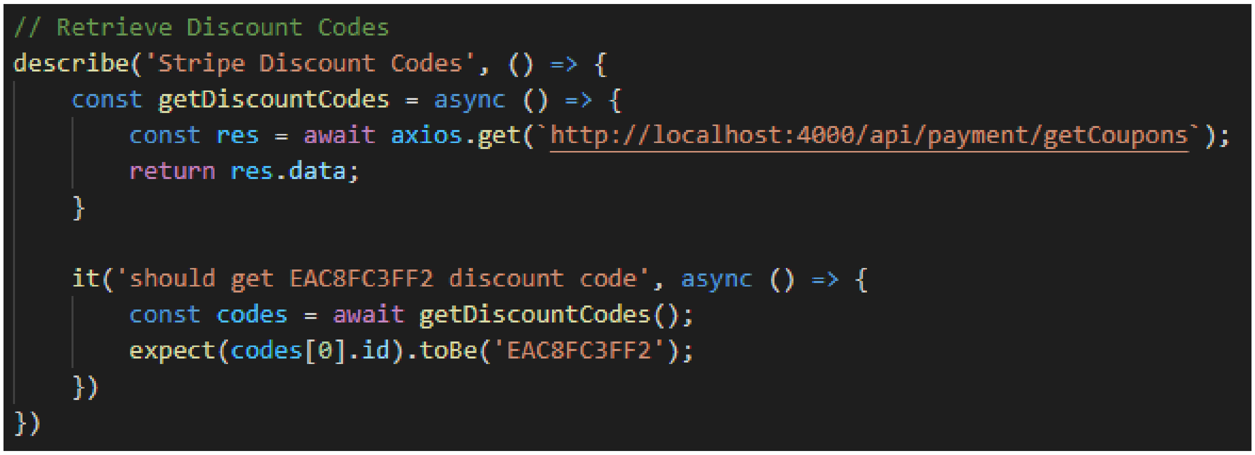
```
// Add products to cart
describe('commerce.js carts', () => {
    const addToCart = async () => {
        await commerce.cart.add('prod_0egY5eMRp253Qn', 1);
    }

    const getCart = async () => {
        const res = await commerce.cart.contents();
        return res;
    }

    it('should add product to cart', async () => {
        await addToCart();
        const cart = await getCart();
        expect(cart[0].product_id).toBe('prod_0egY5eMRp253Qn');
    })
})
```



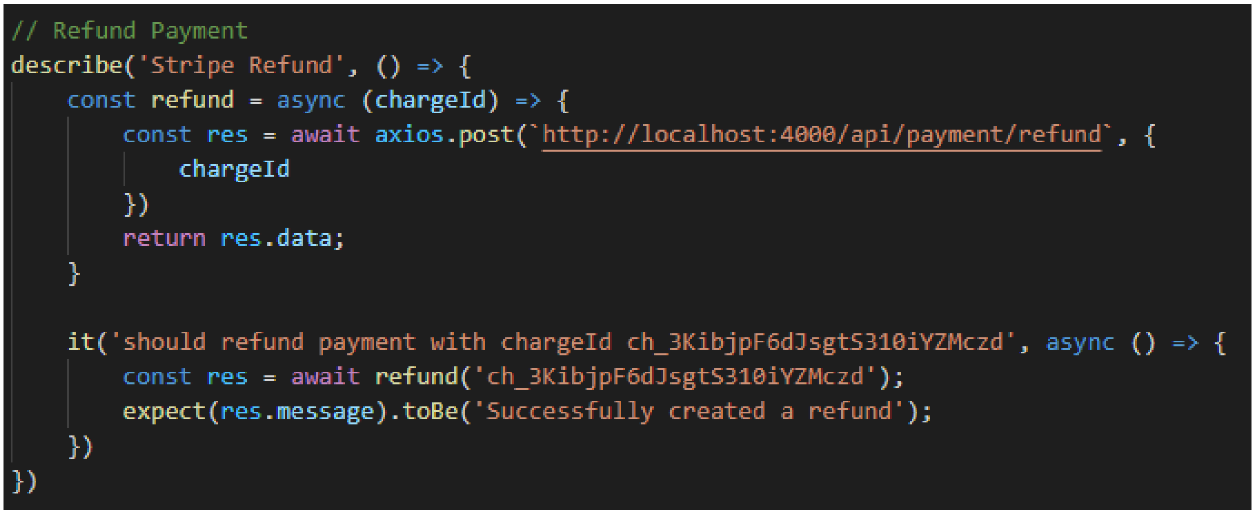
```
// Get shipping rates
describe('commerce.js shipping rates', () => {
    const getCardId = () => {
        const res = commerce.cart.id();
        return res;
    }

    const getShippingOptions = async () => {
        try {
        const cartId = getCardId();
        const checkoutData = await commerce.checkout.generateToken(cartId, { type: 'cart' });
        const options = await commerce.checkout.getShippingOptions(checkoutData.id, { country: 'CA', region: null });
        return options;
        }
        catch (err) {
            console.log(err);
        }
    }

    it('should get Domestic shipping rate', async () => {
        const options = await getShippingOptions();
        expect(options[0].description).toBe('Domestic');
    })
})
```



```
// Get Customers
describe('Stripe Customers', () => {
    const getCustomers = async () => {
        const res = await axios.get(`http://localhost:4000/api/payment/getCustomers`);
        return res.data;
    }

    it('first customer should have an email stephenasuncion01@gmail.com', async () => {
        const customers = await getCustomers();
        expect(customers[0].email).toBe('stephenasuncion01@gmail.com');
    })
})
```

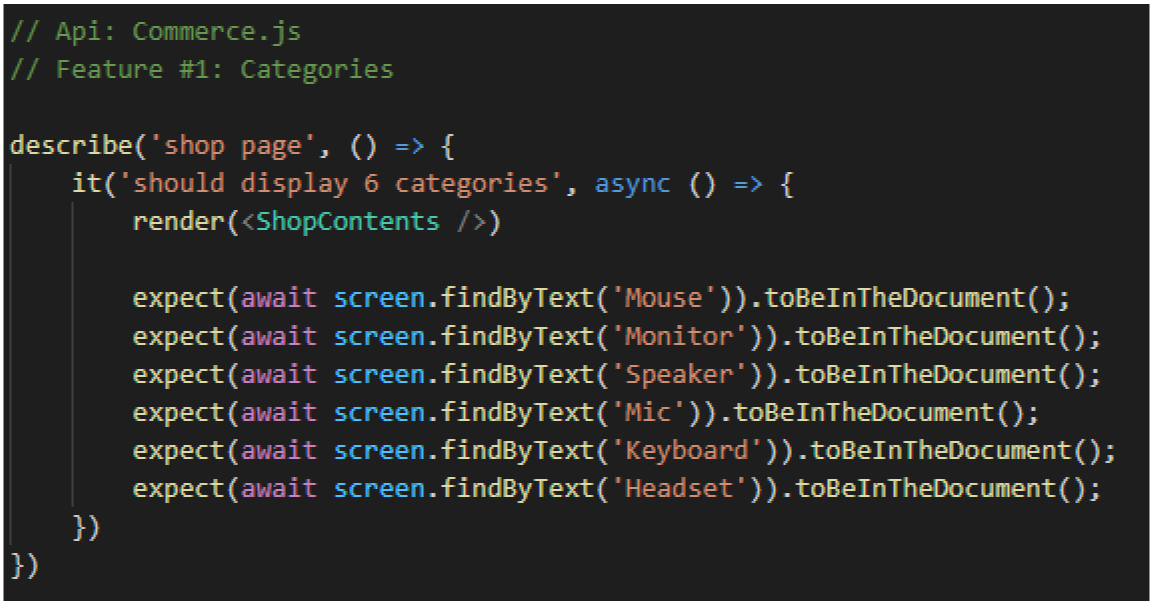


```
// Retrieve Discount Codes
describe('Stripe Discount Codes', () => {
    const getDiscountCodes = async () => {
        const res = await axios.get(`http://localhost:4000/api/payment/getCoupons`);
        return res.data;
    }

    it('should get EAC8FC3FF2 discount code', async () => {
        const codes = await getDiscountCodes();
        expect(codes[0].id).toBe('EAC8FC3FF2');
    })
})
```

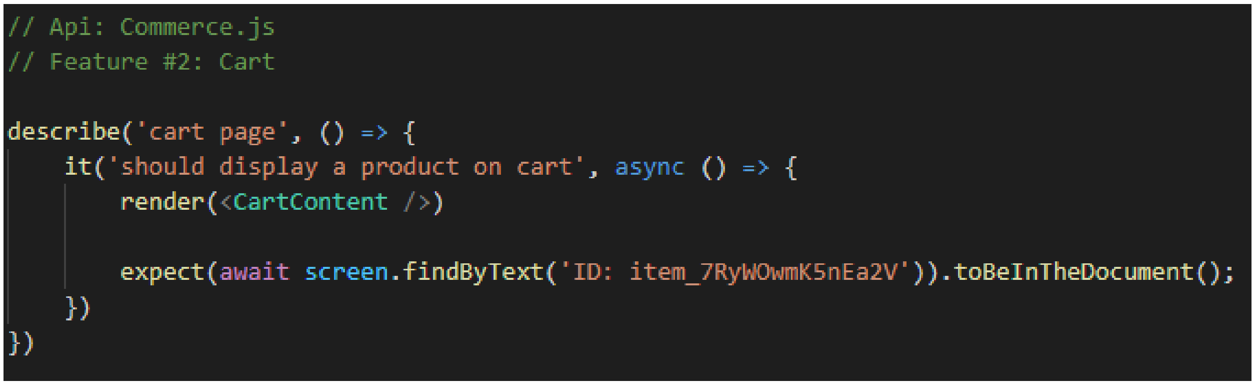


```
// Refund Payment
describe('Stripe Refund', () => {
    const refund = async (chargeId) => {
        const res = await axios.post(`http://localhost:4000/api/payment/refund`, {
            chargeId
        })
        return res.data;
    }

    it('should refund payment with chargeId ch_3KibjpF6dJsgtS310iYZMczd', async () => {
        const res = await refund('ch_3KibjpF6dJsgtS310iYZMczd');
        expect(res.message).toBe('Successfully created a refund');
    })
})
```

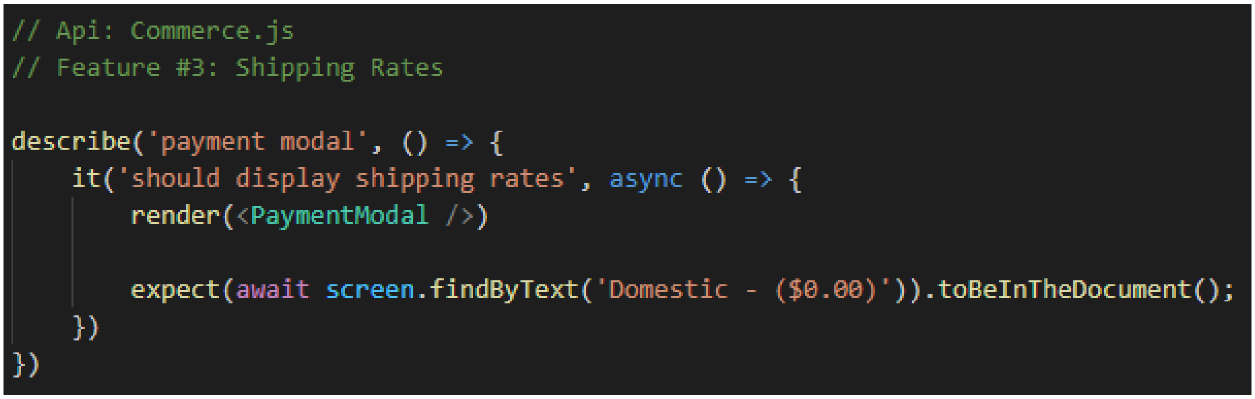
Integration test



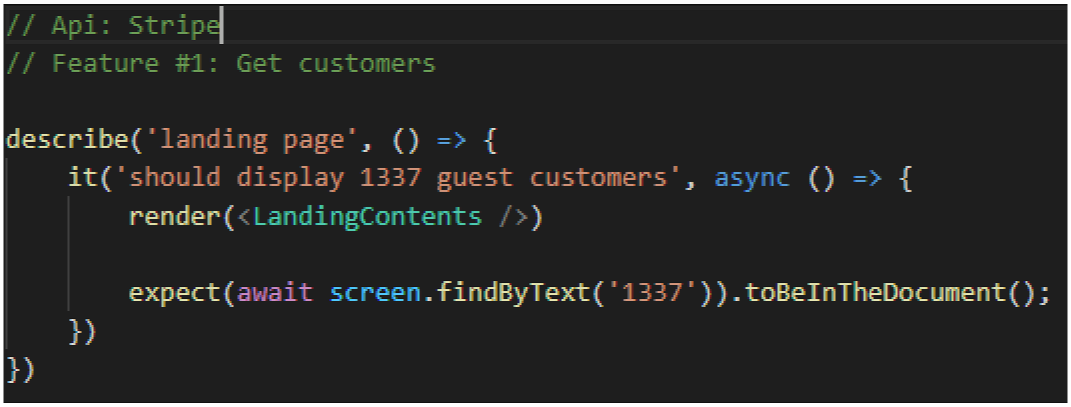
```
// Api: Commerce.js
// Feature #1: Categories

describe('shop page', () => {
    it('should display 6 categories', async () => {
        render(<ShopContents />)

        expect(await screen.findByText('Mouse')).toBeInTheDocument();
        expect(await screen.findByText('Monitor')).toBeInTheDocument();
        expect(await screen.findByText('Speaker')).toBeInTheDocument();
        expect(await screen.findByText('Mic')).toBeInTheDocument();
        expect(await screen.findByText('Keyboard')).toBeInTheDocument();
        expect(await screen.findByText('Headset')).toBeInTheDocument();
    })
})
```



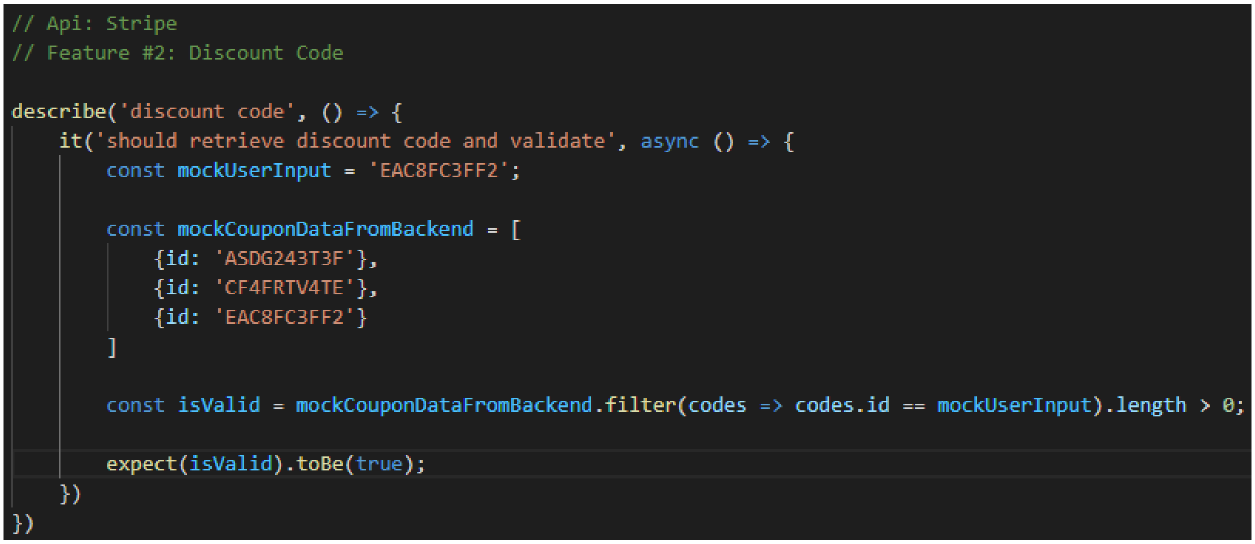
```
// Api: Commerce.js
// Feature #2: Cart

describe('cart page', () => {
    it('should display a product on cart', async () => {
        render(<CartContent />)

        expect(await screen.findByText('ID: item_7RyWOwmK5nEa2V')).toBeInTheDocument();
    })
})
```



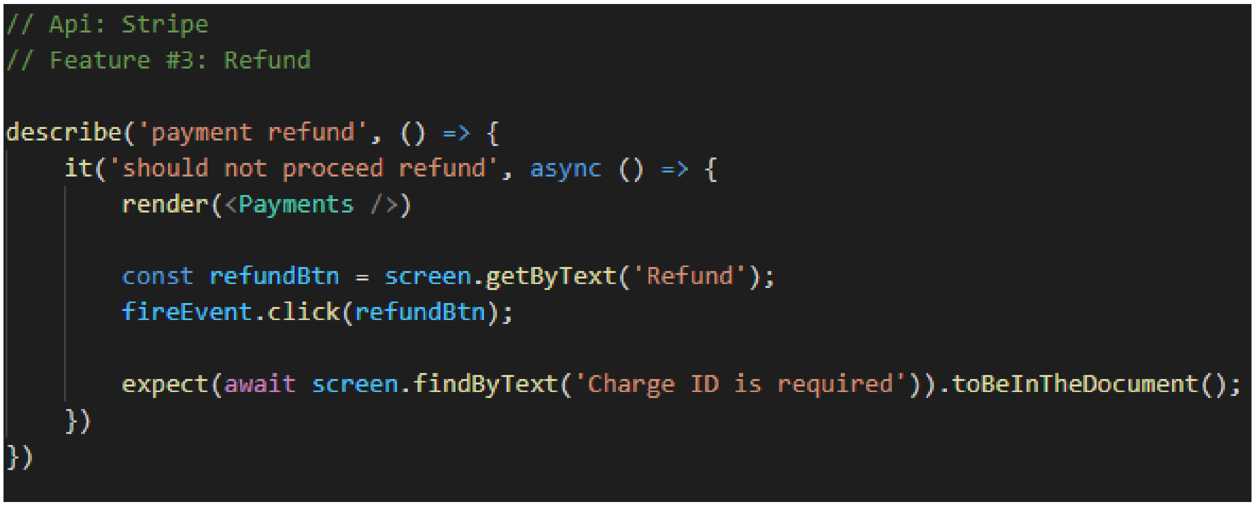
```
// Api: Commerce.js
// Feature #3: Shipping Rates

describe('payment modal', () => {
    it('should display shipping rates', async () => {
        render(<PaymentModal />)

        expect(await screen.findByText('Domestic - ($0.00)')).toBeInTheDocument();
    })
})
```



```
// Api: Stripe
// Feature #1: Get customers

describe('landing page', () => {
    it('should display 1337 guest customers', async () => {
        render(<LandingContents />)

        expect(await screen.findByText('1337')).toBeInTheDocument();
    })
})
```



```
// Api: Stripe
// Feature #2: Discount Code

describe('discount code', () => {
    it('should retrieve discount code and validate', async () => {
        const mockUserInput = 'EAC8FC3FF2';

        const mockCouponDataFromBackend = [
            {id: 'ASDG243T3F'},
            {id: 'CF4FRTV4TE'},
            {id: 'EAC8FC3FF2'}
        ]

        const isValid = mockCouponDataFromBackend.filter(codes => codes.id == mockUserInput).length > 0;

        expect(isValid).toBe(true);
    })
})
```



```
// Api: Stripe
// Feature #3: Refund

describe('payment refund', () => {
    it('should not proceed refund', async () => {
        render(<Payments />)

        const refundBtn = screen.getByText('Refund');
        fireEvent.click(refundBtn);

        expect(await screen.findByText('Charge ID is required')).toBeInTheDocument();
    })
})
```

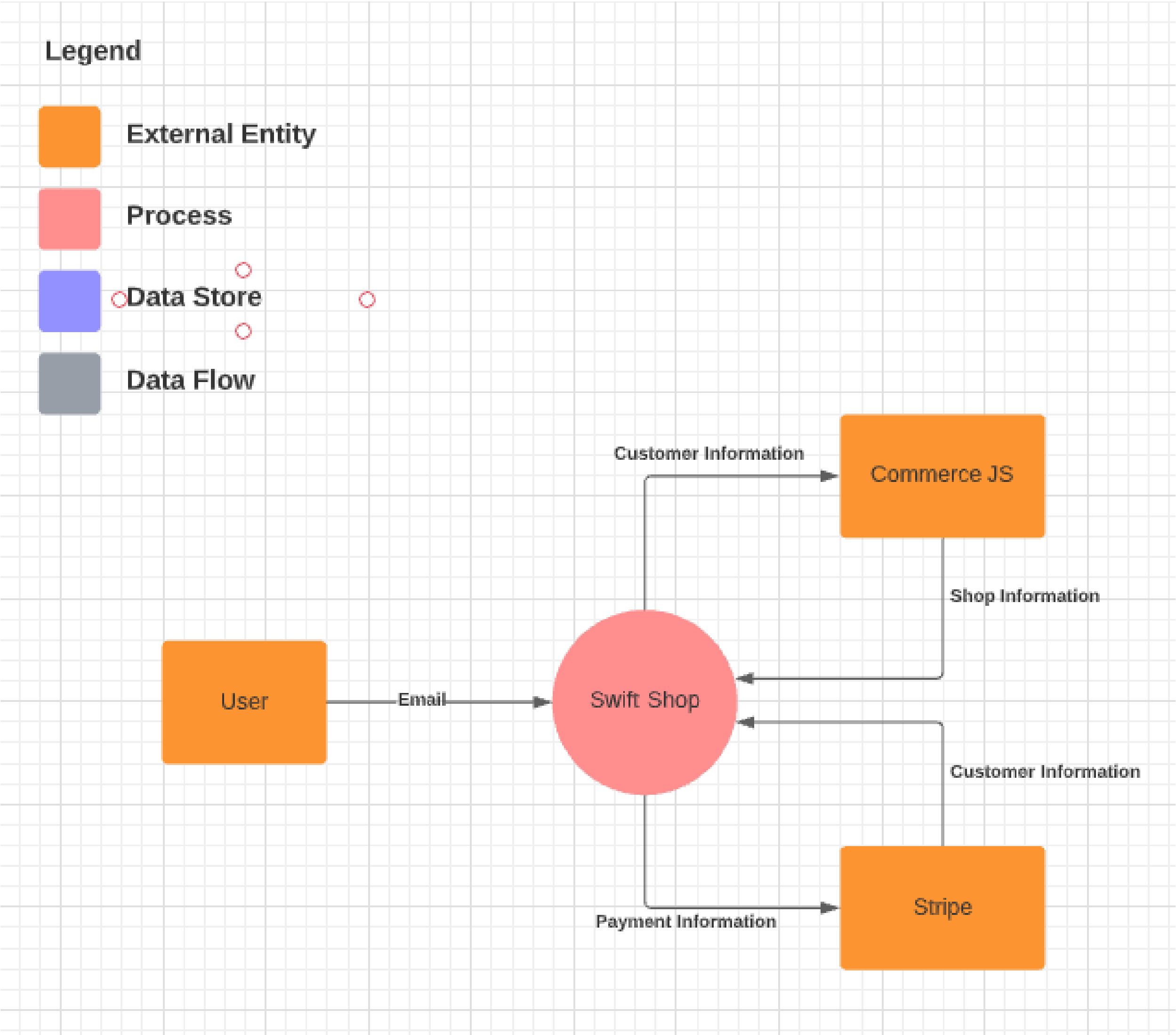
CI/CD infrastructure

- **Build:** We used GitHub for continuous integration. Every time something was pushed to repository, it triggered an automated build and test. After passing all tests, the code was integrated to the repository.

- **Test:** We used jest as our testing framework. Mocks of providers and hooks has been made which is in the ‘__mocks__’ folder and all tests are in ‘__tests__’ folder. A script was created “npm run test” to run the tests.

- **Deployment:** We used Vercel for continuous deployment. Once a branch is merged with the main branch on our repository, it automatically deploys to Vercel.

Data Flow Diagram (High Level)



Takeaway

Working on a big project like this was really exciting and it really made us realize where we planned wrong and what could be improved.

Here are somethings that we learned throughout the project cycle:

1. First, it is important to choose tech stack which is familiar to every member in the team. Completing project within a month is challenging itself, so learning and implementing at the same time is not a good idea.
2. Second, we realized how the things could not go as planned as some unexpected events happen like being ill, going to work, schoolwork etc.
3. Third, it was challenging to maintain a continuous communication among each other as some members were not active.
4. We also realized that we need to add or replace features as we moved forward because of changes in requirements or technical issues.
5. Some of us were not familiar with the use of Chakra UI, folder organization in project and we got to learn these skills.

Division of Work

Unfortunately, we could not follow the WBS we created earlier as some deadlines were not able to meet. So, first we decided to create the Layouts for all pages as a team and then implementing backend. Due to lack of skills and being inactive, Work was unevenly distributed.

Here's the list of tasks performed by each team Member:

1. **Stephen:** Established foundation of project (installing dependencies, configuring routes, public keys etc.), Improved landing layout, improved cart layout, implemented getting and displaying products, cart, checkout, refund etc. i.e., all backend, configured testing and deployment, contributed to creating wireframes, data flow diagrams, report1, WBS.

2. **Daman**: Created Landing, product, and category page, presentaion1, report 2, presenatation2. Also, tried implementing checkout, contributed to creating wireframes, data flow diagrams, report1, WBS.

3. **Cyrus**: Created shop page, contributed in report1, data flow diagram, WBS.

4. **Ryuho:** Contributed in report1, data flow diagram, WBS.