DAT076 / DIT127 Web Applications

Project: Gift Genie
Group 08

Bobby Pang
Sundarakrishnan Ganesh
Kobala Wijesinghe Appuhamilage Thushini Maleesha Ranasinghe
Baseer Qayoumi

March 12, 2024

1: Introduction

Gift Genie is an innovative web application designed to streamline the process of creating and managing wish lists for various occasions, from birthdays and anniversaries to holiday seasons. Our team was motivated by the challenge of simplifying the gift-giving experience, making it more enjoyable and less time-consuming. By leveraging modern web technologies within the MERN stack (MongoDB, Express.js, React.js, Node.js) while also making use of strict type definition standards of typescript, Gift Genie offers a user-friendly platform that allows individuals to specify their gift preferences and share them with friends and family, ensuring that gift giving is both thoughtful and efficient. The project's codebase is available on GitHub (accessed here), providing open access to our development process and final product.

2: Use Cases

1. **Add Product to Wishlist:** Users can easily add products to their wish list, specifying details like product name, URL, price, and quantity. This feature is designed to be intuitive, with a simple form guiding the user through each step.

2. **Fetch Product Details from Amazon:** Leveraging Amazon's Product Advertising API, this innovative feature allows users to paste a product URL from Amazon, and the app automatically populates the product's details, reducing manual input and enhancing user experience.

3. **Retrieve Products from Wishlist:** Users can view a comprehensive list of all products added to their wishlist. This feature is designed with a clean and accessible UI, ensuring users can easily browse and manage their desired items.

4. **Retrieve Specific Product from Wishlist:** Users have the ability to view detailed information about a specific product within a wishlist. This includes product name, description, price, image and a link to the product's online store, facilitating a detailed look at each wishlist item.

5. **Update Product in Wishlist:** This functionality provides users with the flexibility to update product details at any time, ensuring the wishlist remains current and accurate.

6. **Delete Product from Wishlist:** With a simple click, users can remove products they no longer desire, keeping their wishlist tidy and relevant.

7. **Create Wishlist:** Users can create multiple wishlists for different occasions, enabling better organization and personalization of their gift preferences.

8. **View Wishlists for a User:** A dedicated dashboard allows users to view all their created wishlists, offering a bird's eye view of their gift preferences across different occasions.

9. **Update Wishlist Details:** Users can edit the names and descriptions of their wishlists to better reflect the occasion or their current preferences.

10. **Delete Wishlist:** This feature allows for the deletion of entire wishlists, including all associated products, providing users with control over their data and preferences.
11. **Register User:** Before engaging with the wishlist functionalities, users must first register for an account within the Gift Genie application. This process involves entering basic information such as name, email, and password.

3: User Manual

Gift Genie simplifies the process of creating and managing wishlists for any occasion. Here's how to get started and make the most out of your Gift Genie experience directly from your web browser.

3.1 Getting Started

To access Gift Genie, open your preferred web browser and navigate to the application's URL. Upon launching Gift Genie for the first time, you will be greeted by the HomePage.

3.1.1 HomePage

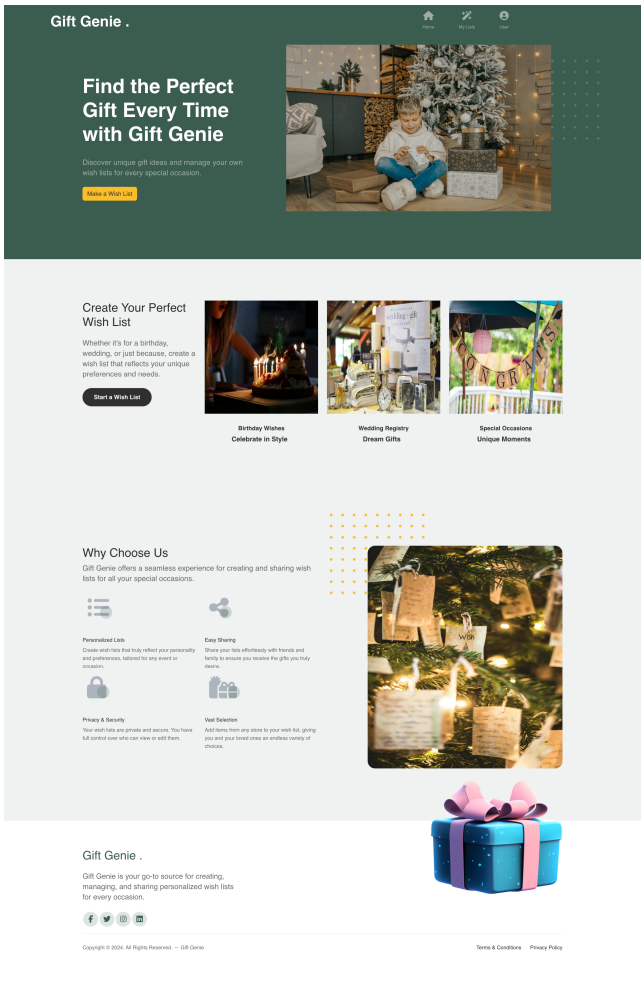

Figure 1: HomePage - The starting point of your Gift Genie experience.

From here, you can either log in or, if you are already logged in, proceed directly to your WishlistPage.

3.1.2 Registration and Login

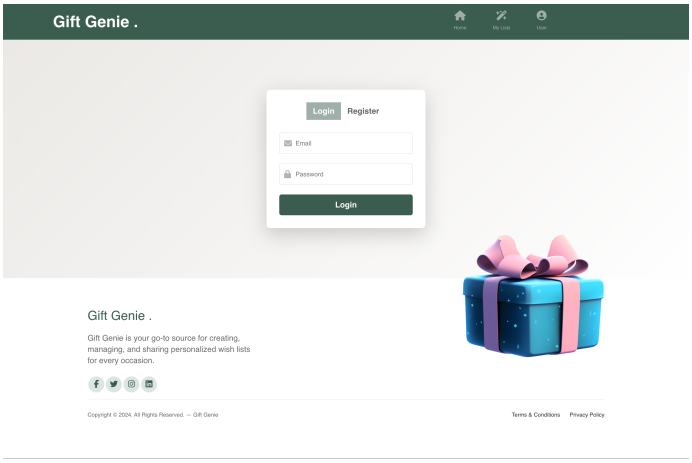

Figure 2: Login Page - Enter your email and password to log in.

The login page requires your email and password. If you're new to Gift Genie, follow the prompts to register.

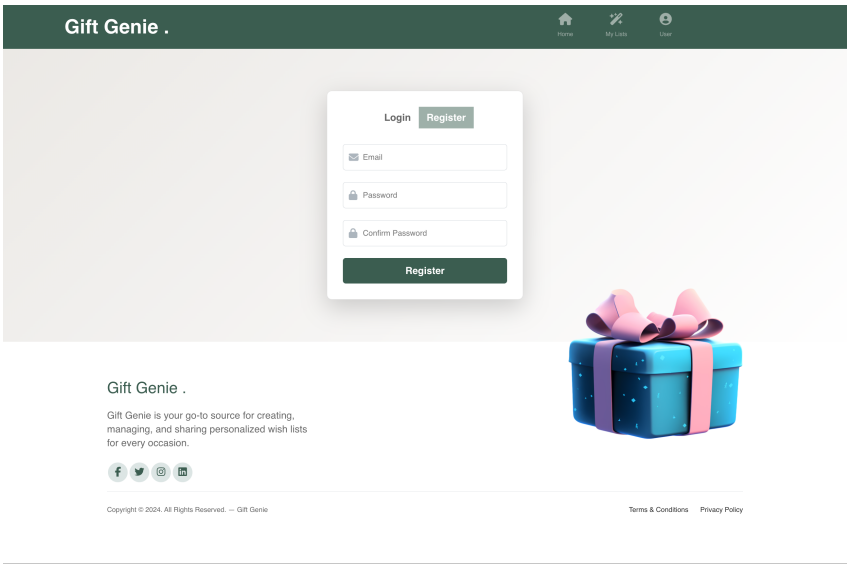

Figure 3: Registration Page - Enter your email and password to register.

3.2 Managing Wishlists

Once logged in, you can manage your wishlists from the WishlistPage. Here, all previously created wishlists are stored. To create a new wishlist, click the "New Wishlist" button and provide a name and description. Hover over wishlists for options to delete or update the wishlist.

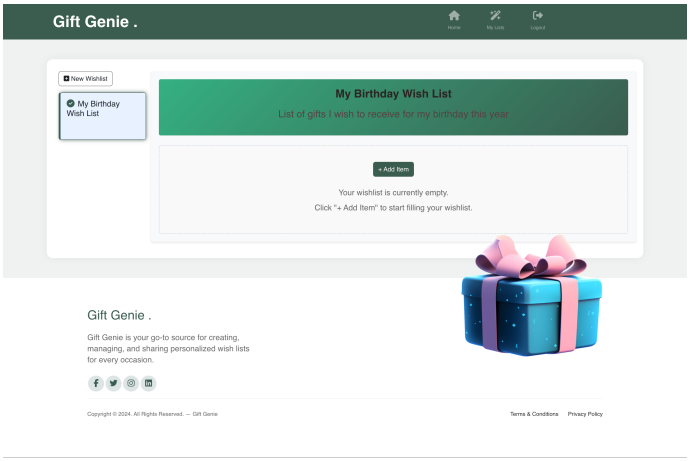

Figure 4: WishlistPage - View, create, delete, or update your wishlists.

3.2.1 Adding Items to a Wishlist

To add items to a wishlist, navigate to the ProductPage by clicking the "Add item" button on your wishlist. Paste an Amazon URL to automatically fetch product details. You can also manually edit these details. Save the product to add it to your wishlist.

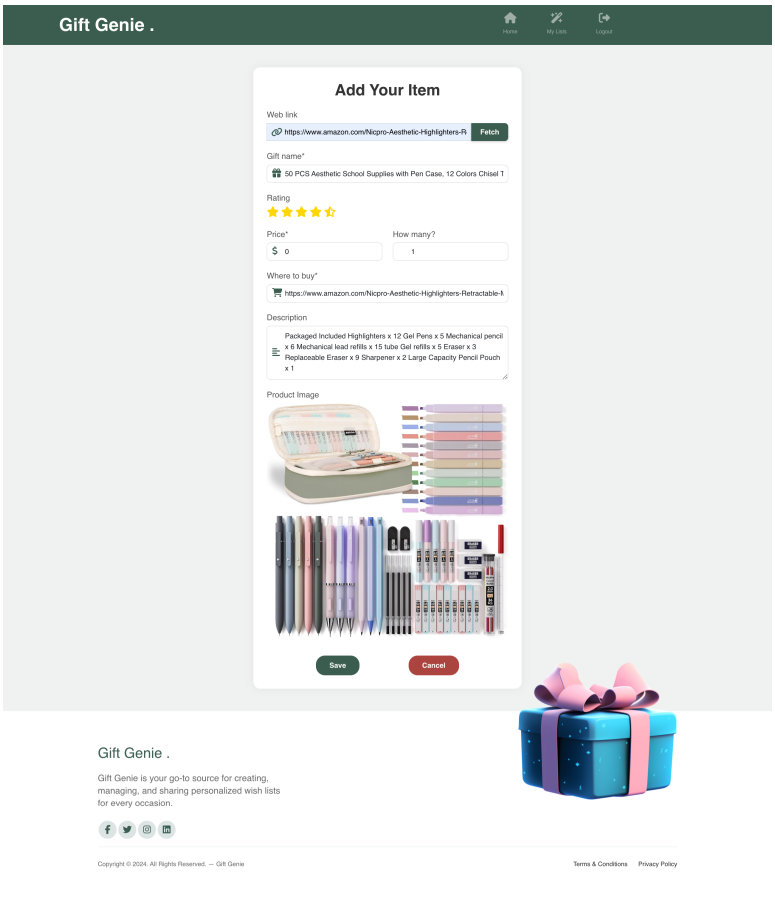

Figure 5: ProductPage - Add items to your wishlist by pasting an Amazon URL.

3.2.2 Viewing and Editing Wishlist Items

Once products are added, you can view them in your wishlist. Hover over wishlist items for options to delete or update. Clicking on an item allows you to modify its details like item name, price, quantity, etc.

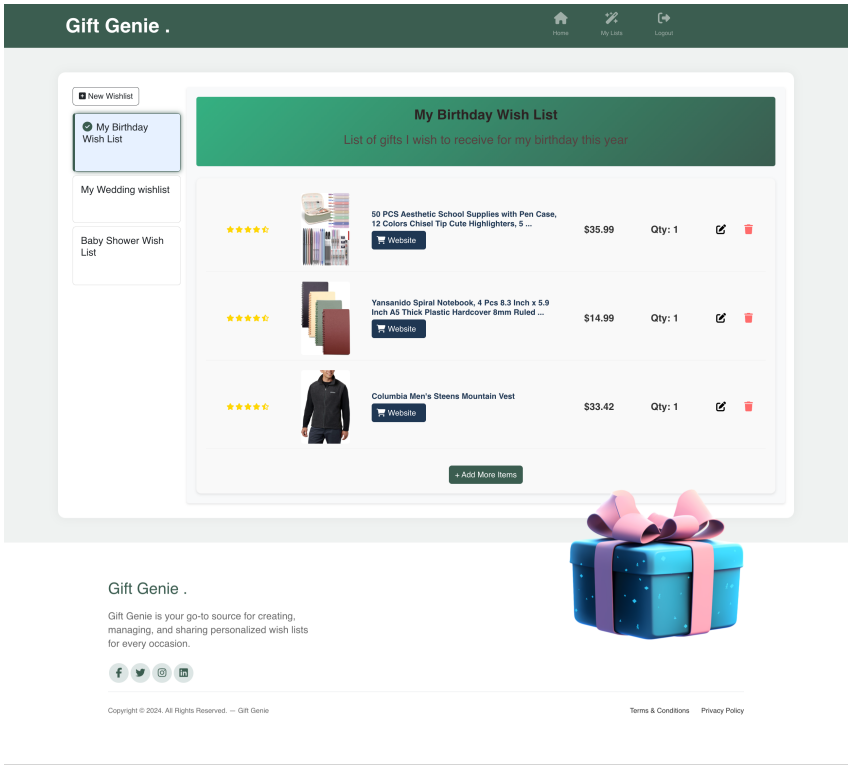

Figure 6: Wishlist with Products Inside - Manage the items in your wishlist.

3.3 Conclusion

Gift Genie is designed to make managing your wishlists and gift ideas as simple and enjoyable as possible. For further assistance, please refer to our FAQ or contact support.

4: Design

The architecture of Gift Genie is meticulously designed to align with the educational goals of the course, following the lecturer's recommendation of utilizing the MERN stack, integrating MongoDB, Express.js, React.js, and Node.js to create a full-stack web application. This choice was motivated by the flexibility, efficiency, and the community support available for these technologies. Additionally, TypeScript was utilized across the stack to enforce type safety and improve code reliability and maintainability.

4.1 Architecture Overview

Gift Genie's architecture is divided into three main layers: the frontend, backend and database, facilitating a clear separation of concerns.

4.1.1 Frontend

The frontend is built using React.js, leveraging its component-based architecture to build a responsive and interactive user interface. React's state management is used to handle user inputs and system states, thus providing a seamless experience. In order to enhance the user interface and improve user experience, the power of bootstrap is leveraged for responsive design elements. Furthermore, custom CSS animations are added to enrich the visual interaction between the user and the system. On top of all this, we also followed strict security guidelines (such as not exposing secrets, input validation etc,) to ensure a protected user data experience.

4.1.2 Backend

In the backend, we made use of Node.js as our primary programming language while we were able to run the server using Express.js, thus forming the backbone components of the backend. Express is responsible for handling HTTP requests and responses, while we also made use of several utility functions to communicate with the database and to perform business logic operations. The architecture of the backend application is designed to be RESTful, making it easy to expand and maintain, thus featuring the potential scalability of our application. As an additional security measure, JWT-based authentication is implemented to protect user data and ensure secure communication between the client and server.

4.1.3 Database

MongoDB is utilized as the database solution and our application Gift Genie utilizes two distinct databases to ensure a clear separation between production and testing environments. This approach guarantees that testing activities do not impact production data, thereby maintaining the integrity and reliability of the application in real-world usage.

For normal operation and user interactions, Gift Genie connects to a MongoDB database, referred to as prodDB. This database stores all user-generated content, including wishlists and product details, ensuring that data persists across sessions and is securely managed.

During testing Gift Genie switches to a separate MongoDB instance, testDB, designed exclusively for testing purposes. This isolation allows our development team to perform unit and integration tests without the risk of corrupting or overwriting production data.

By maintaining separate databases for production and testing, along with a structured testing approach, Gift Genie achieves a high level of code quality, application stability, and user trust.

4.2 API Design

The backend API follows REST principles, accepting HTTP requests and responding in JSON format. The API endpoints are designed to support the primary functionalities of the application, including user authentication, wishlist management, and product handling. The following are the API endpoints:

- **POST /auth/login**: Authenticates a user by verifying email and password credentials, returning a JWT for subsequent requests requiring authentication.
- **POST /auth/register**: Registers a new user with email and password, creating a user profile in the database.

  less Copy code
- **POST /wishlists/wishlist**: Creates a new wishlist, associating it with the authenticated user's ID.
- **GET /wishlists/wishlists**: Retrieves a list of all wishlists created by the authenticated user.
- **GET /wishlists/wishlist/$id**: Fetches details of a specific wishlist by its ID, available to the wishlist's owner.
- **PATCH /wishlists/wishlist/$id**: Allows updating of a wishlist's details (e.g., title, description) by its ID.
- **DELETE /wishlists/wishlist/$id**: Deletes a specific wishlist by its ID, along with all associated products.
- **POST /wishlists/$wishlistId/products**: Adds a new product to a specified wishlist, including details fetched from an Amazon URL if provided.
- **GET /wishlists/$wishlistId/products**: Lists all products within a specified wishlist.
- **GET /wishlists/$wishlistId/products/$productId**: Retrieves detailed information about a specific product within a wishlist.
- **PATCH /wishlists/$wishlistId/products/$productId**: Updates information for a product in a wishlist, identified by the product's ID.
- **DELETE /wishlists/$wishlistId/products/$productId**: Removes a specified product from a wishlist.
- **DELETE /wishlists/$wishlistId/products**: Clears all products from a specified wishlist.
- **GET /products/fetchAmazonProdDetails?url=[productUrl]**: Fetches and returns product details from the Amazon Product Advertising API through Axesso, given a product's URL.

Each endpoint expects request payloads and headers as per REST standards, typically requiring Content-Type: application/json for POST and PATCH requests and Authorization: Bearer [token] for routes necessitating authentication. The responses are JSON-formatted, providing either the requested data, confirmation of actions taken, or error messages with appropriate HTTP status codes to indicate success, client errors (e.g., 404 Not Found, 400 Bad Request), or server errors (500 Internal Server Error).

4.3 Technologies Used

The project leverages a number of libraries and frameworks to enhance functionality and development workflow:

- **Node.js** and **Express.js** for the server environment and web framework.
- **React.js** for building the user interface.
- **MongoDB** and **Mongoose** for database management.
- **Bootstrap** for styling and responsive design.
- **Axios** for making HTTP requests from the frontend.
- **Jest**: Used for unit and integration testing of both the frontend and backend components.
- **JSON Web Tokens (JWT)** for secure user authentication.
- **bcrypt**: Ensures the security of user passwords through hashing.
- **dotenv** for managing environment variables.

5: Responsibilities

Detailed account of each team member's contributions.

1. **Week 1:** In Lab 0, our primary focus was on setting up the development environment and initiating a Node.js project. The most important thing in the initial week was suggestions on ideas of the application. Thushini compiled a document containing various suggestions, which the group later voted on.
2. **Week 2:** The first lab consisted of creating a design mockup for the idea using HTML, CSS and bootstrap. Bobby worked on the login and registration pages. Baseer worked on the item description page. Thushini worked on the homepage including navigation bar and footer and later on merged the html sites together.
3. **Week 3:** For the second lab which was mainly backend, Krishna joined the group and we decided to split up the lab into four different tasks. Bobby was responsible for the wishlist backend which involved the creation, deletion, update and fetching of wishlists. Thushini managed the products backend, incorporating tasks like adding new products, retrieving all products, fetching product details from Amazon automatically when a URL is given, updating existing products, and handling deletions. Krishna was responsible for the login backend which involved the registering an user, hashing the password and login. Baseer was tasked with creating tests for the backend which involved testing the fetching and deletion of products from wishlists. The test also included fetching product details from Axesso. The group individually merged their own branches in to a development branch on GitHub.

4. **Week 4-5:** For the third lab which was dedicated to the creation of a frontend in react, the group decided to split the tasks similarly to the former lab. Since this lab was meant to be in React, the group had to modify the HTML and adjust them using React. Thushini was responsible for the add products page which involved utilising the Amazon API to successfully fetch product details to the page so that it could later be added to a wishlist. Thushini also worked on improving the homepage, navigator and footer for the pages. Krishna worked on the login and registration page making sure the user can authenticate themselves. Bobby and Baseer collectively worked on the wishlistpage, making sure that the user could create, delete, update and fetch wishlists. Bobby and Baseer also collaborated on various aspects of frontend testing, such as both successful and error scenarios. Specific tests included the creation of wishlists and the addition of products to a wishlist. Thushini later on merged the adding product feature to the wishlistpage and helped with the styling of the page.

5. **Week 6 and onwards:** The fourth lab consisted of adding a database using MongoDB to preserve data inbetween sessions or when the server is stopped and restarted. After Thushini Setup the Database connection, similar to what we have done during past weeks Krishna worked on connecting the database with Authentication, Thushini worked on Products and Baseer and Bobby collaborated on Wishlists. The remaining time was spent on creating more use cases, testing , refactoring, and polishing of comments and finally the project report.

6: Testing Strategy

Testing is a critical component of the Gift Genie development process, ensuring the application is robust, reliable, and user-friendly. Our testing strategy encompasses a comprehensive suite of tests, covering all 11 use cases identified in the project. This approach includes unit tests for individual components and integration tests that verify the interaction between various parts of the system, including backend services, the database, and the frontend interface.

6.1 Unit Testing

For the backend, we utilized Jest, a powerful testing framework, to create tests for our Express.js controllers and MongoDB models. This ensures that our business logic is correct and that our database interactions behave as expected.

Example: ProductService Back-end Test
A unit test for the *ProductService* demonstrates how we ensure products are correctly retrieved from wishlists. This test verifies the retrieval process and error handling, using mocked responses to simulate database interactions.

```
1 it('returns all products for a given wishlist ID', async () => {
2     const wishlistId = 1;
3     // Example product instances
4     const mockProducts: Product[] = [
5         new Product(1, wishlistId, 'Test Product 1', 'Description 1', 'url
    ', 'image', 10, 1, '1 star', new Date()),
```

```
        new Product(2, wishlistId, 'Test Product 2', 'Description 2', 'url', 'image', 20, 2, '2 stars', new Date()),
    ];
    (productModel.find as jest.Mock).mockReturnValue({
        exec: jest.fn().mockResolvedValue(mockProducts),
    });

    const products = await productService.getProductsFromWishlist(wishlistId);
    // Assert that correct products are returned
    expect(products).toEqual(mockProducts);
    expect(productModel.find).toHaveBeenCalledWith({ wishlist_id: wishlistId });
});
```

Example: AuthPage Front-end Test
The AuthPage front-end test example illustrates how we verify the behavior of our authentication page, particularly focusing on user registration with non-matching passwords. Through this test, we simulate a user filling out the registration form with an email and two passwords that do not match. When the form is submitted, we check to ensure that the appropriate error message is displayed, indicating the mismatch.

```
it('shows an error when trying to register with non-matching passwords', async () => {
    render(<MemoryRouter><Auth /></MemoryRouter>);

    // Switch to the registration form
    fireEvent.click(screen.getByTestId('register-switch'));

    // Fill the form with mismatching passwords
    fireEvent.change(screen.getByPlaceholderText(/Email/i), { target: { value: 'user@example.com' } });
    fireEvent.change(screen.getAllByPlaceholderText(/Password/i)[0], { target: { value: 'password123' } });
    fireEvent.change(screen.getByPlaceholderText(/Confirm Password/i), { target: { value: 'password124' } });

    // Attempt to submit the form
    fireEvent.click(screen.getByTestId('auth-submit'));

    // Check for the password mismatch error message
    await waitFor(() => expect(toast.error).toHaveBeenCalledWith("Passwords don't match."));
});
```

6.2 Integration Testing

Integration tests assess the combined operation of multiple system components. For the backend, this involved testing API endpoints to ensure they correctly process requests and return the expected responses. For the frontend, we used React Testing Library to simulate user interactions with the web interface and verify the correct behavior of the application.

Example: WishlistRouter Back-end Test
In this test for WishlistRouter we verify that a specific wishlist can be successfully re-

trieved by its ID, ensuring the API endpoint responds correctly to user requests with the appropriate authorization.

```
// Tests retrieving a specific wishlist by id
it('retrieves a specific wishlist by ID', async () => {
    const response = await request.get(`/wishlists/wishlist/${wishlistId}`)
        .set('Authorization', `${token}`);

    expect(response.statusCode).toBe(200);
    expect(response.body.wishlist_id).toEqual(wishlistId);
});
```

Example: WishlistPage Front-end Test

An integration test for the *WishlistPage* component shows how we simulate user actions to create a new wishlist. This test checks the interaction flow, from requiring the user to be authenticated before creating a wishlistpage until clicking the "New Wishlist" button to start filling out the form and submitting a new wishlist.

```
describe('WishlistPage Integration Tests', () => {
  beforeEach(() => {

    // Mock successful authentication
    (Cookies.get as jest.Mock).mockImplementation((key: string) => {
      if (key === 'authToken') return 'fake-token';
      return null;
    });
  });

  it('allows a user to create a new wishlist', async () => {
    const newWishlist = {
      user_id: 1,
      title: 'My New Wishlist',
      description: 'Description of my new wishlist.',
    };

    // Mock the POST request for creating a new wishlist
    axiosMock.onPost('/wishlists/wishlist').reply(201, newWishlist);

    render(
      <MemoryRouter>
        <WishlistPage />
      </MemoryRouter>
    );
    userEvent.click(screen.getByText(/New Wishlist/i));

    const wishlistTitleLabel = await screen.findByText("Wishlist Title");
    userEvent.type(wishlistTitleLabel, newWishlist.title);

    const descriptionLabel = await screen.findByText("Description");
    userEvent.type(descriptionLabel, newWishlist.description);

    userEvent.click(screen.getByText(/Create Wishlist/i));

 // Use findByText to wait for the new wishlist title to appear on the page
 expect(await screen.findByText(/My New Wishlist/i)).toBeInTheDocument();
});
});
```

6.3 Automated Testing for Use Cases

For each of the 11 use cases, we wrote automated tests to cover the happy path and at least one error scenario. This ensures that our application behaves correctly under both expected and unexpected conditions. By covering most error scenarios, we provide a high level of confidence in the stability and reliability of Gift Genie.

7: Future Work

As Gift Genie continues to evolve, our team is committed to enhancing its functionality and user experience. While the current version of the application successfully addresses the core requirements of creating and managing personalized wish lists, we recognize the potential for further innovation and utility. To this end, we have identified several key features for future implementation:

1. **Share Wishlist:** We plan to introduce a feature that allows users to generate a shareable link for their wishlist. This link can be sent to friends and family, enabling them to view the wishlist without the need for creating an account. This feature aims to facilitate easier sharing and collaboration, making it simpler for users to communicate their gift preferences.

2. **Sort the Wishlist or Product by Name:** Enhancing user experience through improved navigation and organization is a priority. Implementing sorting capabilities will allow users to easily organize their wishlists and products alphabetically by name, thereby streamlining the search and selection process within the application.

3. **Duplicate Wishlist:** Recognizing the utility of reusing and adapting existing wishlists for different occasions or recipients, we aim to add functionality that enables users to duplicate an existing wishlist. This feature will save time and effort for users who wish to create multiple similar wishlists with minimal modifications.

4. **Export Wishlist to PDF/CSV:** To increase the versatility of wishlist sharing and management, we wish to develop an option for users to export their wishlists into PDF or CSV formats. This will be particularly useful for users who prefer to share their lists outside the platform or require a hard copy for personal use.

8: Conclusion

This section concludes the project with a short summary of the application and a reflection on the learnings during the course of the project.

8.1 Summary

We as a team set out to build a react.js project which creates and manages wishlists, with products from Amazon. We managed to build exactly what we set out to build. Using Gift Genie, an end user would be able to register themselves onto the application and login using their credentials. Making use of cutting edge authentication and complying to coding standards, the users can be reassured that their data is in their control. Once logged in, end users are free to create and manage wishlists with different products to

their satisfaction. If the user feels that a wishlist is deprecated, they can also choose to purge it.

8.2 Reflections

From the get go, we had incredible dynamics as a team. We were able to co-ordinate within ourselves and managed to get the most out of every week's milestone. We used to hold calls once weekly to discuss weekly accomplishments and also to split the tasks required for the upcoming week. We were also very active over Discord with help incase anyone required assistance with their tasks. Regarding the technology, almost no one struggled as the learning curve was not steep. Thus we were able to progress through the course of the project with ease. However, upon self reflection we realised that a little more organization of tasks using technologies like Jira would have improved the team dynamics and record management by a lot.