Mini Project Report

Ctrl Alt Elite
Group number 8

1 Communication

1.1 Objective

Given the task to build an online clothing store, we held an initial meeting with all the group members where we discussed the projects scope, goals, and objectives. It was decided that our objective would be to build an online thrift store. An online thrift store where users can not only purchase clothes from other user, but also submit a request to have their own clothing item be uploaded and sold on the store. We initially decided to use HTML, CSS and JavaScript for this project as everyone in the group was familiar with these languages but later implemented React and Next.JS.

1.2 Requirements and functional aspects

1.2.1 Process User request. Allow users to submit a request to have their clothing item sold through the online thrift store.
1.2.2 Product listing. Provide a detailed description of the product on sale, including the brand, size, images, and a condition rating.
1.2.3 Manage the product catalog. Being able to add and remove products on the catalog. As well as store and retrieve product information such as name, description and images associated with the product.
1.2.4 Shopping cart and check out functionality. Online shopping cart that would store the products users added to their cart. As well as a seamless checkout process.
1.2.5 Help option. A section that explains how the online thrift store works and how to sell their items

1.3 Assumptions

1.3.1 All the technology required to create, manage, and host the online store will be available to use for free.
1.3.2 Every member of the team had the skill set required to produce an application like this.
1.3.3 Assumed it would be a lot more interesting to build than a regular online clothing store.
1.3.4 Assumed that there would be a lot of people interested in selling their clothes online instead of throwing them away.

1.4 Constraints

1.4.1 Budget constraints. With a budget of R0, we soon discovered that many of the API and technology needed required were not available to use for free.
1.4.2 Lacked the necessary skills. We had very little experience with building web apps so a lot of up-skilling had to be done before getting started and some functionalities of the website had to be revised.
1.4.3 Time constraints. With many other commitments, it was difficult to focus on the project alone and soon realized that time was not on our side. In order to manage the time, we spilt the team members into different groups, with each group tackling one aspect of the project.

1.5 Non-functional aspects

Since the store is an online store and would be accessible to any device, it should be responsive to whichever device it is accessed on. Without compromising the useability.

2 Planning

In the initial stages it was decided that we would create a web application. After further discussion we decided to use the waterfall process model. The reason for this is because none of the group members had experience building apps. Therefore, it would be difficult to deliver multiple software increments (as described in the agile process model) because we would need time to upskill. Using the waterfall approach helped us to create a clear structure of what needed to be done.

Initially groups were formed with each group focusing on a different aspect of the creation of the app, namely front-end, back-end and the database. It was decided to use a database to create a more dynamic web-app that would also increase the reusability of the app. After the formation of the groups, everyone was given time to upskill so that a high-quality product could be produced. At the end of the allocated time, each group was to submit a timeline on how long it would take to implement their tasks. This time was also given for the front-end, back-end and database groups to create a use-case diagram, a wireframe design, and an ER diagram respectively. The front-end team split the work by pages, meaning each member would focus on creating a page for the web-app. The back-end team split the work by functions,

meaning each member would focus on the different functions of the web-app.
This particular process model worked well for the group although during the final stages of the project there was a small rush to finish everything so some functions that we initially planned were cut.

`3 Modelling

Table 1: Use case for Sell item

Use Case	Sell Item
Description	This use case depicts the process of a user (seller) listing items for sale on a platform for online thrift stores. As part of this procedure, sellers can upload item descriptions and photographs, set prices, and arrange shipment.
Actors	Seller (User) Online thrift platform
Variations	Sellers may choose to include additional item description details. Sellers may set fixed prices. Online store may offer option for seller to sell items
Non-functional	Usability: provide intuitive and user-friendly interface for sellers to list items. Performance: The system should be able to efficiently handle a high number of listings and user interactions. User data and payment information should be handled securely. Reliability: During peak usage periods, the system should be available and reliable. Scalability: The system should be able to scale to handle an increase in the number of users and listings.
Issues	User-generated content management entails managing and moderating user-generated listings and interactions. Payment processing: Ensuring secure and dependable transaction payment processing. Legal and regulatory compliance: Ensuring that e-commerce and privacy regulations are followed. Returns and disputes: Creating procedures for processing returns and settling conflicts between sellers and the platform.

Table 2: Use case for Buy item

Use Case	Buy Item
Description	This use case depicts a user (buyer) viewing and purchasing items for sale on an online thrift store platform. Buyers can use the platform to browse item details, make purchases, and finalize transactions.
Actors	Buyer(User) Online thrift platform
Variations	Buyers may search and filter items based on various categories. Buyers may choose different payment methods. Buyers may opt for different shipping options and carriers. Online store may facilitate dispute in case of issues with purchased items
Non-functional	Usability: The system should provide a simple and easy-to-use interface for purchasers to browse and buy things. Performance: The system should respond

	quickly to searches, item views, and transactions. User data and payment information should be handled securely. Reliability: During peak usage periods, the system should be available and reliable. Scalability: The system should be able to scale to handle an increase in the number of users and listings.
Issues	Payment Processing: Ensuring secure and reliable payment processing for events. Seller-buyer communication: Implementation of a message or communication system on the platform. Compliance with laws and regulations: Ensuring compliance with e-commerce and data protection regulations. Returns and Disputes Handling: Establish processes for handling returns and resolving disputes between buyers and sellers.

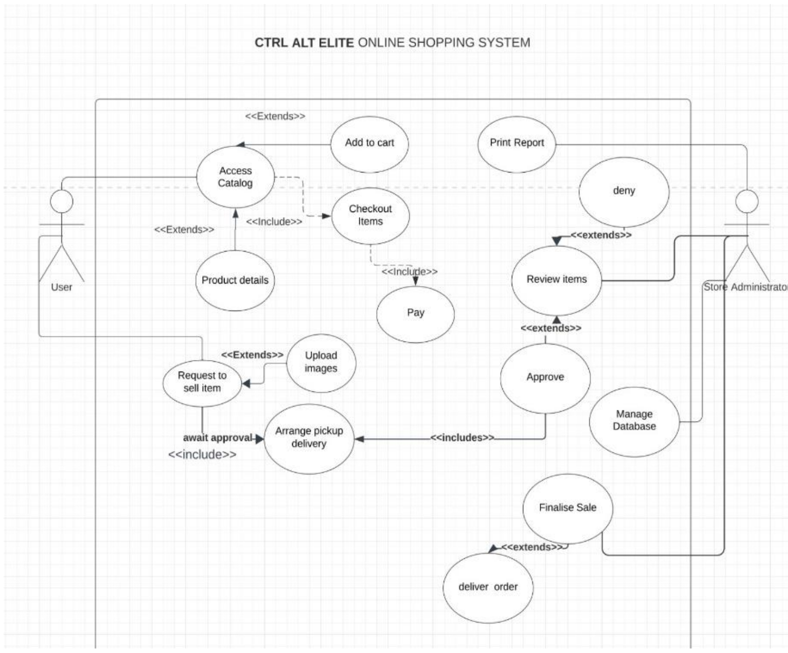

Figure 1: Use case diagram

4. Construction

4.1 Code Description

To create a more modular and dynamic website the backend group agreed to implement the React framework with the intent to further develop the product the deadline. The decision to implement React was made very late into development when majority of the front-end work and some back-end features where already build using Vanilla JS and HTML. To meet the deadline features that did not integrate well within the framework were removed or modified. Our react web app currently consist of five web pages and five React components. To manage items for sale the Sanity API was used to store datasets containing the images and their relative attributes. These features were stored in our project schema (Sanity refers to tables within a dataset as schemas).

4.2 Test Cases

Test Case	Test Step	Expected Outcome	Feedback
Verify images of product display correctly	1. Scroll catalog. 2. Click on product,	1. Product images should be shown correctly. 2. Product description page should open.	Success
Verify product details	1. Click on an item in catalog or the home page.	1. The item image and details loaded should match the item clicked on.	Success
Verify cart calculations.	1. Add a single item to cart and check that total price is correct, 2. Add multiple items to the cart and check that total price is the sum of all product prices	1. Cart displays the price of the single item. 2. Cart displays the total of the price of all items in the cart.	Success

Verify email feature	1. Submit request to sell item through the sell page. Wait for browser notification. 2. Check if email received	1. Receive notification. 2. Both user and the person who approves items receives an email containing item details and attached image.	1. No notification. 2. No email received. Analysis: This function was built from HTML and vanilla JS and met the test cases before implementing React. Due to a lack of experience with the React framework this function could not be properly implemented.
Check Checkout use case	1. Add items to cart and click checkout button in cart. 2. Fill in check out page. 3. Click continue to checkout. 4. Since no payment methods will be added until deployment a thank you for shopping message will appear.	1. Check out page loads. 2. Captures customer data. 3. Thank you for the shopping message appears.	Success

5.Deployment

Oracle Cloud offer free VMs, with no time constraints. These VMs can be used to host projects or websites. if we'd like to deliver our website for users to use and interact with we'd upload all of our files from github to oracle and store all of our data in the cloud. Additionally, we'd have to purchase a domain for our website. We will begin the process by creating an oracle account. We can create a virtual machine instance and select our image and shape. We will add a ssh key and click create. Once our virtual machine is up and running we'd like to copy the public IP address and set it up with our domain name. Using Oracle we'd select which ports we'd like to allow. Finally, We will use putty to complete the server setup before uploading our website files and we should be able to run multiple commands on our server in the terminal that we've opened using putty such as restart, pause etc… after all that our website will now be accessible through an internet browser.