SunSational Shades

EECS 4481

Phase Three

Design Document

Group Members:

Faiz Ahmed

Mhd Yazan Armoush

Stefan Petrovic

Andrew Rezaev

Diego Santosuosso

Version:	**1**
Print Date:	**09-04-2023**
Release Date:	**09-04-2023**
Release State:	**Third Deliverable**
Approval State:	
Approved by:	
Prepared by:	
Reviewed by:	
Path Name:	
File Name:	**Project Team B: SunSational Shades**
Document No:	

Document Change Control

Version	Date	Authors	Summary of Changes
1	10-02-2023	Team B	None
2	30-03-2023	Team B	Worked on the server side implementation along with elementary client-side implementation.
3	09-04-2023	Team B	Worked on the front-end implementation using React, and deployed the website using Netlify and AWS

Document Sign-Off

Name (Position)	Signature	Date
Faiz Ahmed	Faiz	09-04-2023
Mhd Yazan Armoush	Yazan	09-04-2023
Stefan Petrovic	Stefan	09-04-2023
Andrew Rezaev	Andrew	09-04-2023
Diego Santosuosso	Diego	09-04-2023

Contents

1 Introduction

1.1 Purpose

This document details the requirements of the system Sunsational Shades, a new eCommerce site for the sale of sunglasses. It provides a comprehensive presentation of the construction of the system. The website is designed to offer customers an efficient and convenient online shopping experience where they can browse and purchase a wide range of sunglasses.

SunSational Shades is a state-of-the-art eCommerce platform that allows customers to navigate through its extensive catalog of sunglasses easily. The site is user-friendly, and customers have the ability to sort items by price, name, category, brand, and colour. Additionally, customers are able to view product details, add items to their shopping cart, and check out using a secure payment gateway. The site also features a review and rating system, which allows customers to share their experiences with others and provide valuable feedback to the company. It also has a virtual try-on feature developed with cutting-edge technology, and a chatbot that answers customer questions. This chatbot plays a crucial role in providing customers with the support they need during their shopping journey.

An additional feature regarding a loyalty program is included in the system, in which users get loyalty points (which they can redeem for cash to be used in the website) as they place their sunglasses orders through the eCommerce system.

The eCommerce system was developed using the Three-Tier architecture in order to organise its implementation into three main logical and computing components: the presentation layer, the application layer, and the data layer. It keeps high cohesion and low coupling between the various modules in the system in order to maintain proper design.

Sunsational Shades is developed with one simple goal in mind: to make it easy and convenient for customers to purchase good-quality sunglasses while also offering them the support they need to make informed decisions. With its state-of-the-art features and user-friendly interface, SunSational Shades is poised to become a leading player in the eCommerce market for sunglasses.

1.2 Overview

This document outlines the design decisions, system structure, system requirements, use case and action diagrams, detailed software architecture (including package and component diagrams), a description of the various modules, and an activities plan. It is an overview of the system's conceptual software architecture, including its design patterns and high-level view.

1.3 Resources

The development of SunSational Shades required a variety of resources, including a robust eCommerce platform, a comprehensive database of product information, and a secure payment gateway. In order to comprehend the overall system structure, it is important to understand the software requirements and the specific project specifications listing all the features and use cases to be included in the online platform.

2 Major Design Decisions

2.1 Architectural Styles:

We have built our e-commerce website in MERN stack using the following 3-tier architecture to structure our application:

1. **Presentation tier:** The presentation tier of our e-commerce website consists of the front-end components that are built using React.js. We have used React.js to create user interface components such as product listings, shopping cart, checkout pages, and user account management. We have also used React.js to create responsive and dynamic web interfaces to enhance user experience.

2. **Application tier:** The application tier of our e-commerce website consists of the back-end components that are built using Node.js and Express.js. We have used Node.js and Express.js to create a RESTful API that handles the business logic and processes user requests from the presentation tier. This API handles tasks such as managing product inventory, processing orders, and handling payments. Additionally, we have used Mongoose, an Object Document Mapper, to interact with MongoDB, our database.

3. **Data storage tier:** The data storage tier of our e-commerce website consists of MongoDB, a NoSQL database. We have used MongoDB to store and manage the website's data, such as user, product, and order information.

By using the 3-tier architecture, we have separated the concerns of our application into three distinct layers, each with its own responsibilities. This way, our architecture promotes modularity, flexibility, and scalability, making our e-commerce website more maintainable and easier to update in the future.

2.2 Design Patterns:

1. **Front Controller Pattern:** We implemented the Front Controller pattern by creating a single endpoint that handled all incoming requests. This endpoint received the request and then dispatched it to the appropriate controller based on the URL and request method. The Front Controller also handled authentication and authorization before dispatching the request to the appropriate controller. This helped us achieve better separation of concerns and improve the overall maintainability of our application. The centralized request-handling process also provided a more consistent user experience and made it easier to add new features or modify existing ones in the future. The entire backend is divided into separate files for each model and for each route, as well as separate modules for authentication requests.

2. **Observer Pattern:** We used the Observer pattern to notify objects when a change occurred in a subject. This pattern allowed us to achieve loose coupling between objects and

simplified the process of updating dependent objects. It was used in various cases in our application. For example, when a user (who is the subject) places an order, the order's model (which is the observer) will detect the change and update its database. In this case, the customer model updates the orders model of a change in state, and the order's model responds accordingly.

3. **Singleton Pattern:** We used the Singleton pattern to ensure that only one instance of a class was created and used throughout the application. It was specifically used in the inventory model. This pattern was useful in managing resources, such as database connections, and prevented unnecessary duplication. Using this pattern in the inventory model is crucial, as there should only be one instance of the inventory. More than one instance would result in errors in the application.

2.3 Modularization:

Two important design criteria we considered when building our e-commerce website using the MERN stack were high cohesion and low coupling.

High cohesion means that the elements within a module or component are closely related and work together to achieve a single, well-defined purpose or responsibility. To achieve high cohesion, we defined specific functions for each module, such as product catalogue management, payment processing, customer management, and order management. We then created separate modules for each function to ensure that the code was well-organized and easy to understand, maintain, and scale. Within each module, we designed the code to be as focused and self-contained as possible, minimizing the impact of changes to one module on the rest of the website.

We avoided tightly coupling modules to achieve low coupling, allowing them to be updated and modified independently. We used clear interfaces between modules to ensure effective communication while reducing their dependence on each other. We also employed abstractions and design patterns to make the code more flexible and easier to maintain. All of our code is divided into different models and routes, each with specific functions.

By implementing these design choices, we were able to build a scalable, flexible, modularized, and easy-to-maintain e-commerce website providing a seamless shopping experience for our customers.

2.4 Development Methodology: Agile

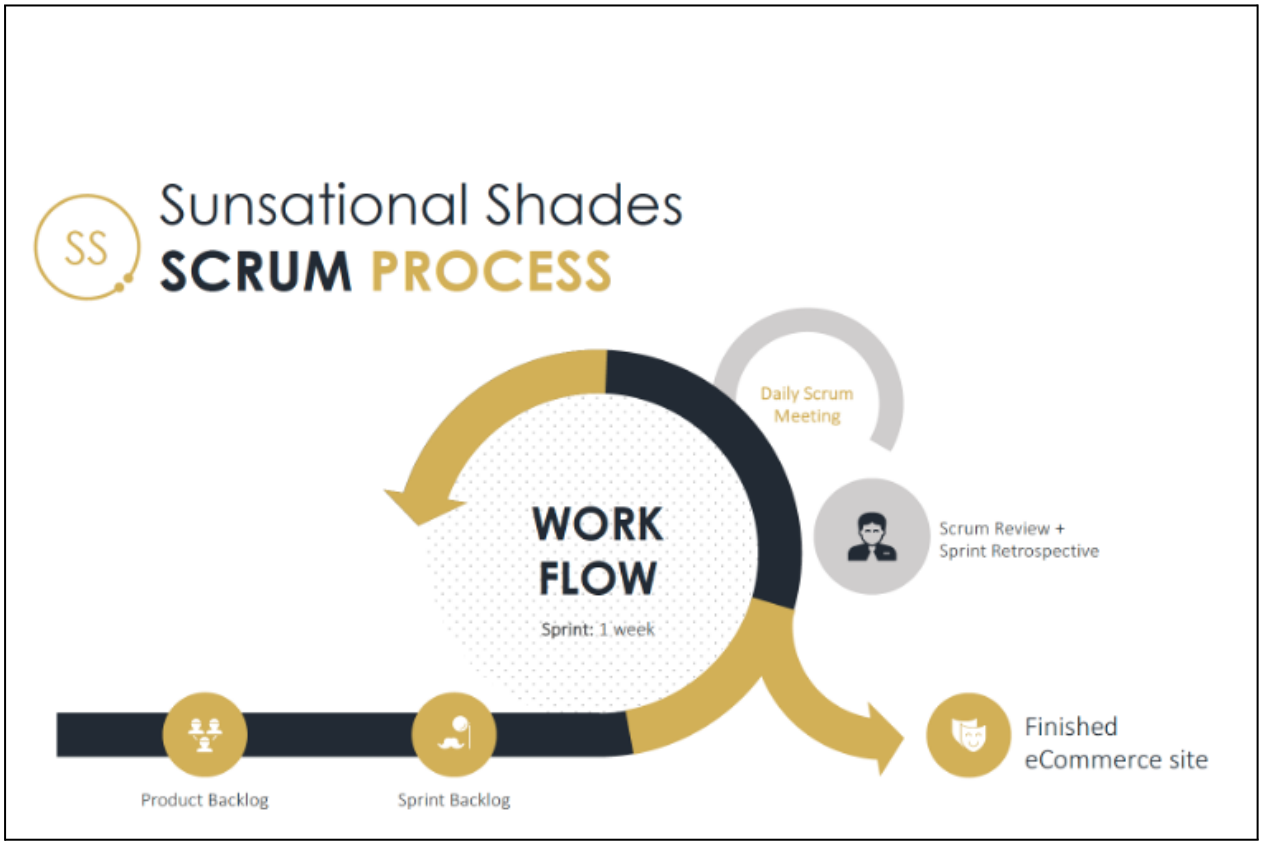


Figure 2.4: Scrum Process

The eCommerce site will be developed in an agile methodology, with sprints of one week in length. Refer to (Activity Plan) section to reference the Gantt Chart with the full list of the product backlog and their specific tasks.

3 Quality Attributes

The architecture of the Sunsational Shades website supports several critical quality attributes for a successful eCommerce system. Using its three-tier model (Presentation Tier, Application Tier, and Data Tier), the website has a well-structured architecture supporting various quality attributes, such as security, performance, scalability, availability, and maintainability.

Security: Security is implemented at multiple levels, including the use of jwtAuth and tokenCheck utilities to authenticate users and sign them with a session token. The MongoDB database provides additional security features such as authentication, authorization, password encryption, and access control that maintain security as the topmost priority of the architecture design process.

Performance: The three-tier architecture supports good performance by separating concerns and allowing for independent optimization and scaling of each tier. React's efficient rendering of user interfaces also contributes to better performance.

Scalability: MongoDB's scalability features support the architecture's scalability, such as horizontal scaling and sharding. For example, React's component-based architecture helps modularize and reuse components in the front-end. In the backend, routers help to enable easy scaling and independent deployment of each individual router. This makes it easy to scale the website to support hundreds of thousands of simultaneous users.

Availability: Good availability is ensured by the well-structured three-tier architecture and the use of MongoDB's replication and failover features. In the front-end, for instance, React's virtual DOM helps achieve better availability by enabling faster rendering and user interface updates, even when the network is slow or unreliable. In the backend, the server ensures that the system is always connected to the MongoDB database in order to read and write to it accordingly.

Maintainability: Finally, the three-tier architecture supports good maintainability by ensuring a clear separation of concerns and making it easier to maintain and update each tier independently. Models and routers also help to organize and modularize the code. Each model has a separate file and route, separating and organizing the code to facilitate testing and maintenance. In summary, the architecture of the Sunsational Shades website delivers a secure, high-performance, scalable, available, and maintainable eCommerce system.

4 Diagrams

4.1 Deployment Diagram

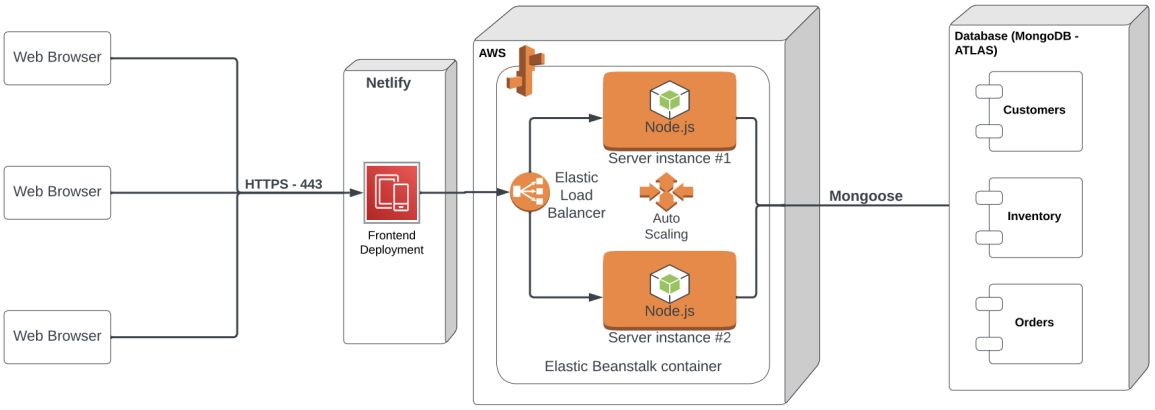


Figure 4.1.1: Deployment Diagram

Figure 4.1.1 demonstrates the deployment mechanism that our website implemented. AWS along with Netlify are the main technologies used to deploy our site. They facilitate the entire deployment process and make the entire web application cloud-native and easily scalable.

4.3 Use Case Diagram

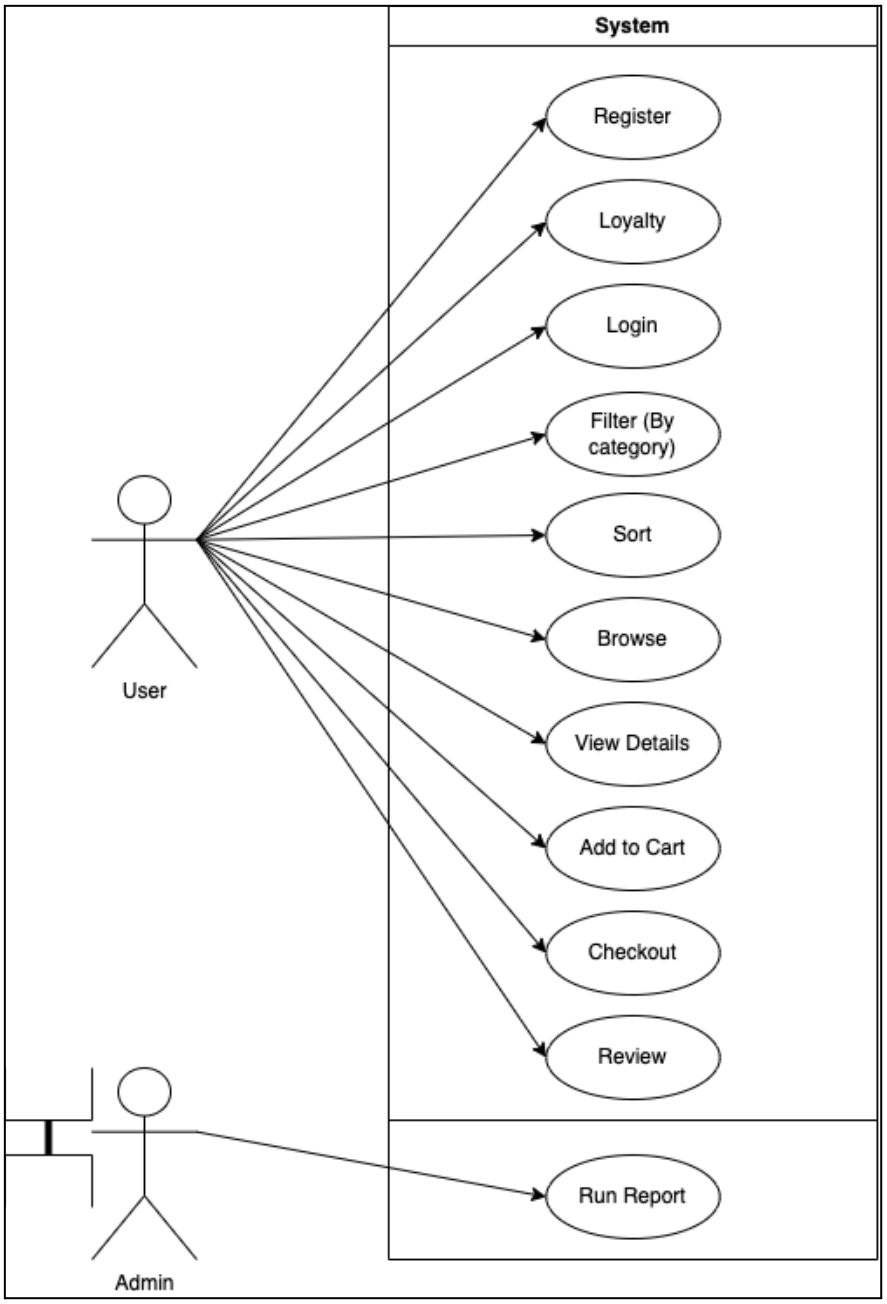


Fig 4.2.1 Use case diagram depicting different uses of the website

4.3 Sequence Diagrams

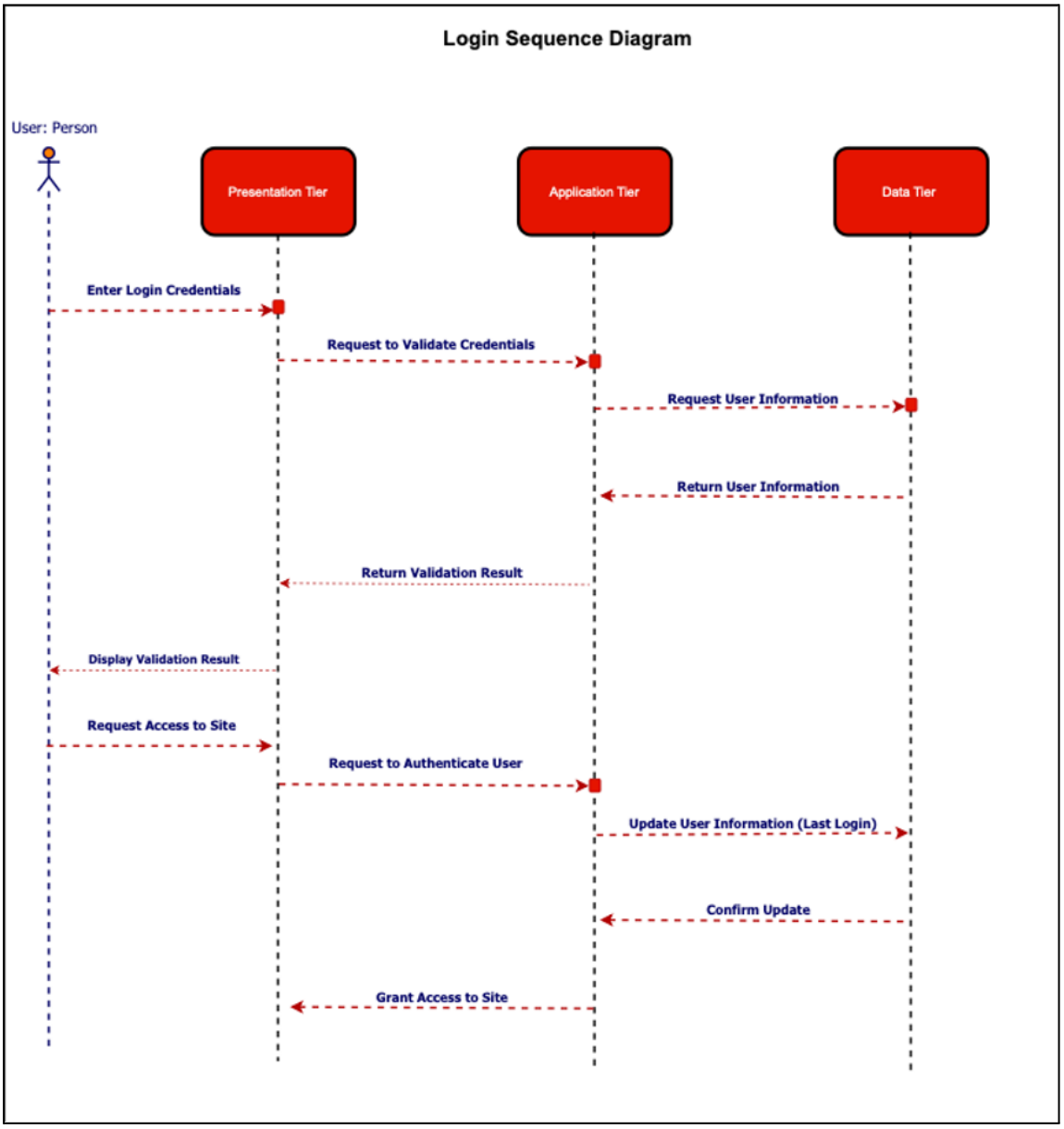


Fig 4.3.1 A sequence diagram showing how a user gets access to the site by logging in

5 Architecture

5.1 Updated Component Diagram

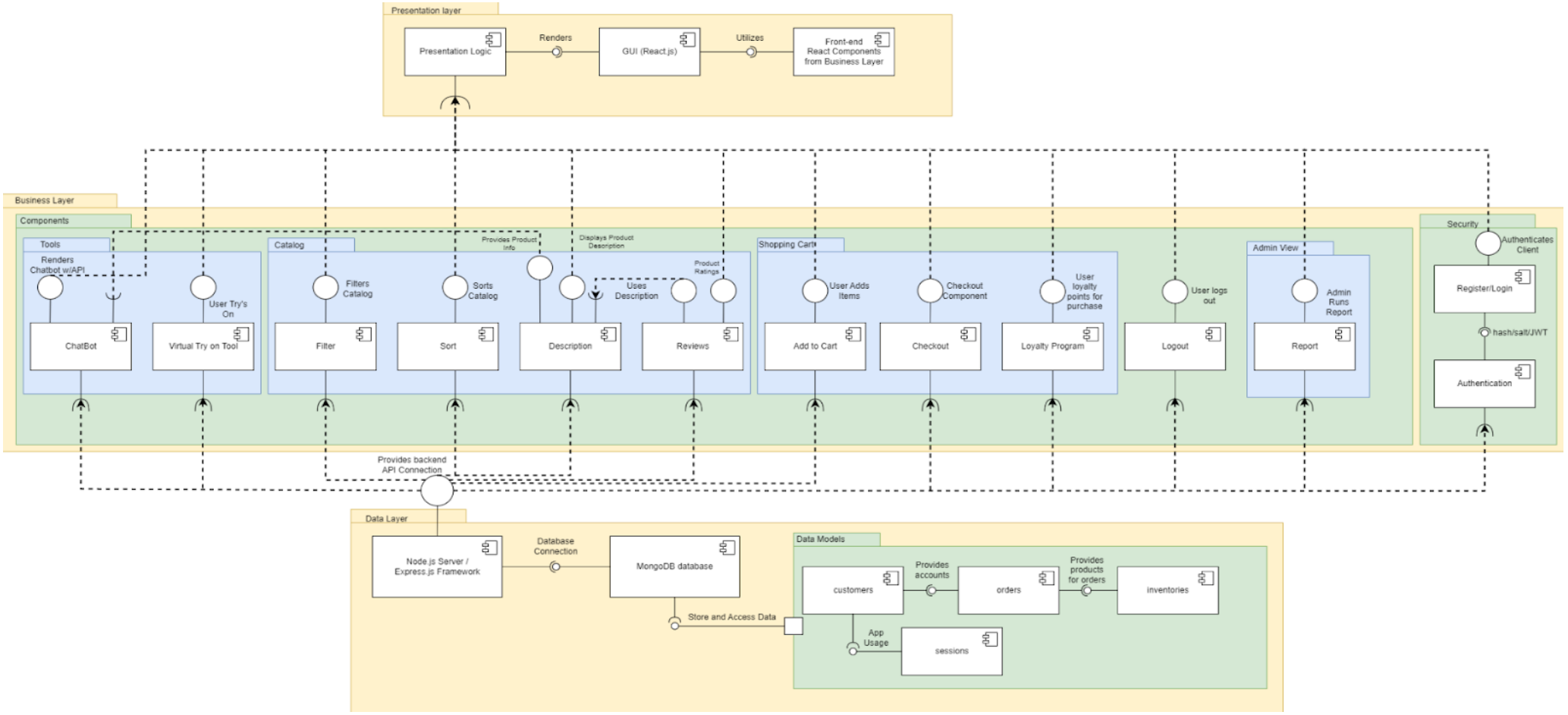


Figure 5.1: Updated Component Diagram

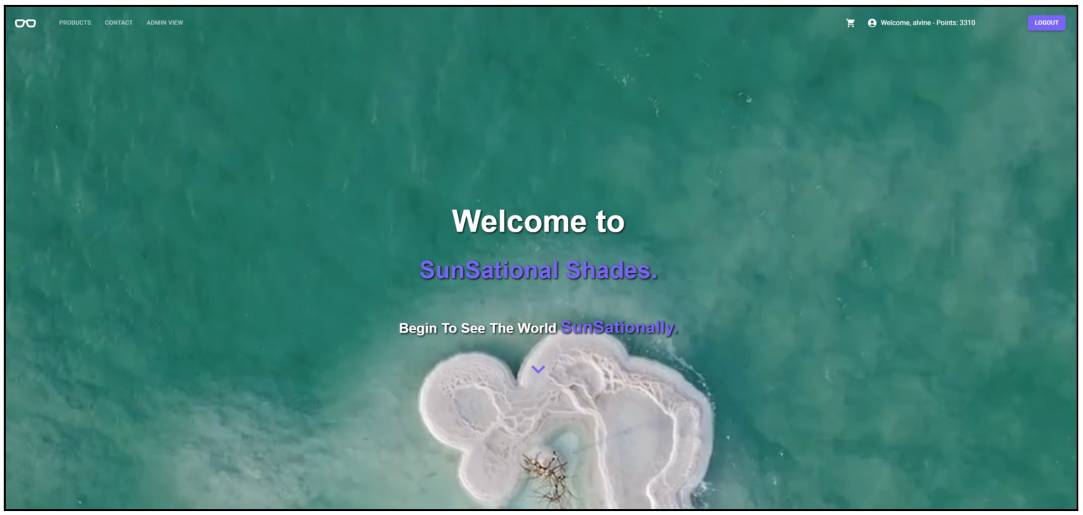
Figure 5.1 shows the updated component diagram that demonstrates the concrete architecture of the Sunsational Shades eCommerce site, which provides a more realistic implementation of the system compared to the initial component diagram presented in the Deliverable 1, which described its conceptual architecture.

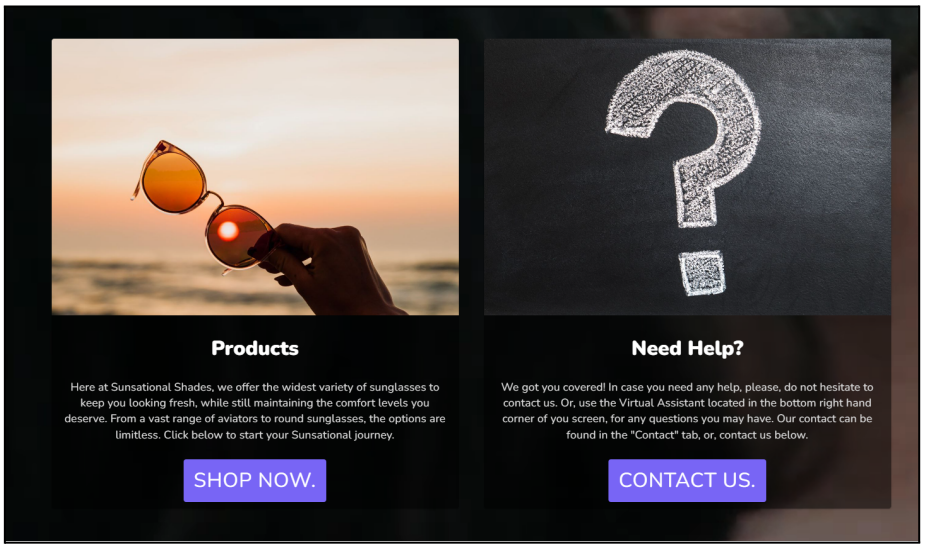
While the high-level view of the system is maintained throughout both component diagrams, the updated version shows a more detailed design of the system's architecture. Every feature has its own module stored and refactored into its own file in order to ensure readability, maintainability, and scalability.

6 Description of Web Design

The key to an excellent web design for any project contains minimalism and easy to understand user interfaces, that anyone (especially non-technical persons) should be able to use. Especially in an E-commerce system where a very wide variety of users can exist, it is important to keep it simple. With this in mind, we went for this approach in our own project: simple and clean.

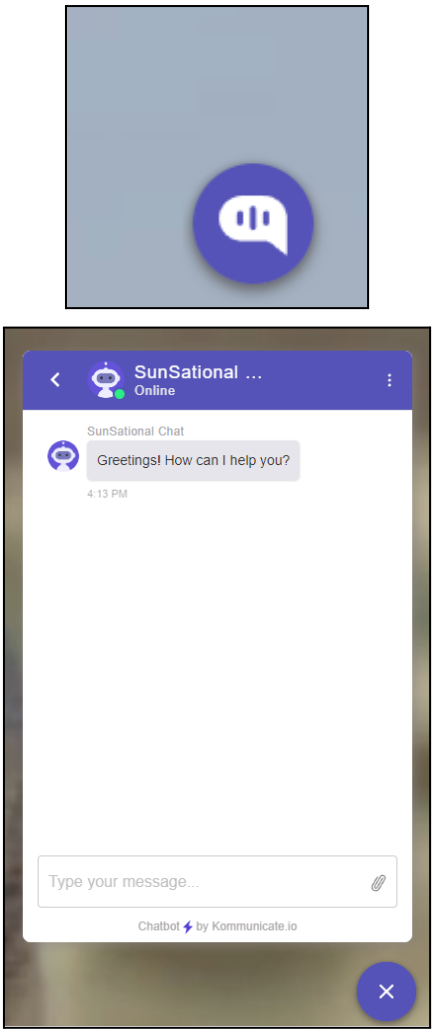
To start, when first accessing the webpage, the user is greeted with a stellar landing page that provides some pretty good visual effects as well as being fully responsive, with clean animations.





The purpose of this page here is to help guide the user through the webpage more conveniently, immediately directing them to the products section.The visualization of this landing page gives the user a more sophisticated and professional feel of the website, and due to its simple navigation, it is easy to find things. In case the user needs help, they can get help right away by contacting our company through the contact button. Additionally, there is a

chatbot button located at the bottom right hand corner of *every* page that can assist the user with anything they need when using the site, again, showcasing accessibility:



The entirety of the front-end is rendered using React.js, as discussed in previous deliverables. WIth the help of some powerful react libraries, it was possible to create very easy to understand user interfaces. The library that was primarily used for forms, buttons, and other rendered components, was Material-UI (MUI). MUI contains very many DOM components that are easy to style and increases production speeds greatly.

The Navigation Bar

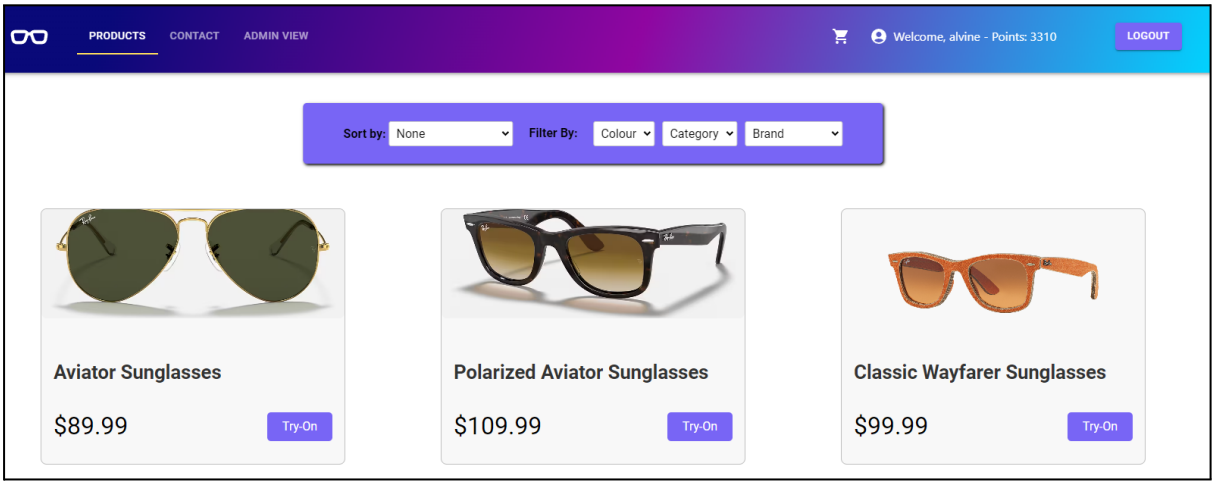
The most important component of our website is our Navbar. Everything the user could possibly need will always be available in the navbar, with interactive and responsive tabs to let the user know where they are on our site at all times. For example, the entirety of the sites functionality for an anonymous user vs. an admin:

WIth this, it is an easy to understand navigation system within our webpage, with the sunglasses logo being clickable and easy to take the user back to the landing page. The cart and profile page is also visible to the user at all times, with icons making it self explanatory on where to go if they need to update their information on their profile, or, go to their cart when they are done shopping. In addition to all of this, the user can always figure out how many loyalty points they have with us, as an authenticated user will always be able to access this information.

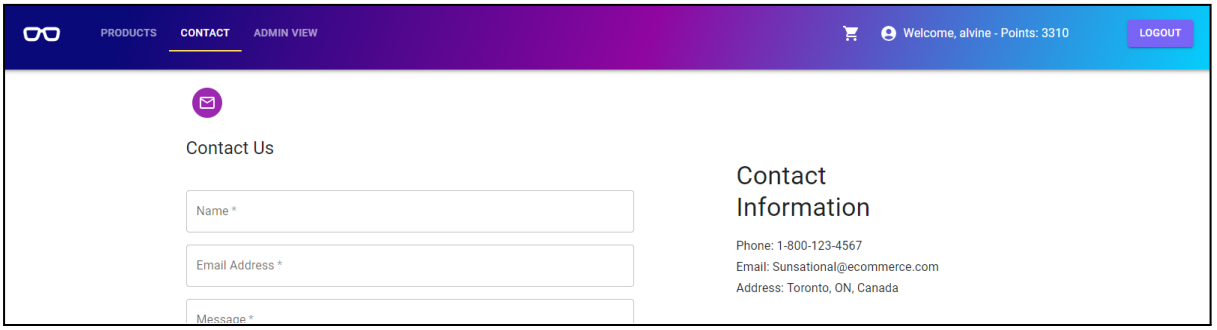
The main takeaway here is that the navbar is ALWAYS persistent throughout the webpage: this is done thanks to React routing. Essentially, the navbar will never disappear from the webpage (apart from the virtual try-on tool), to make sure that the user can always navigate the site without having to click on back arrows and so on.

The usefulness of this means that we can simply render different components underneath the navbar, without ever having to refresh the page to navigate to other pages. This increases the performance and smoothness of the UI. For example, when going from /products to /contact, the navbar is persistent, and only the component (products and contact, respectively) are rendered underneath:

When clicking on the products tab (filtering is shown in the demo page, it's always at the top of the user's screen):



Then, by clicking on the contact tab, the yellow underline shows the user where they currently are on our site. Note: This underline will disappear accordingly if lets say, the logout button is clicked. The site is fully responsive in this fashion.

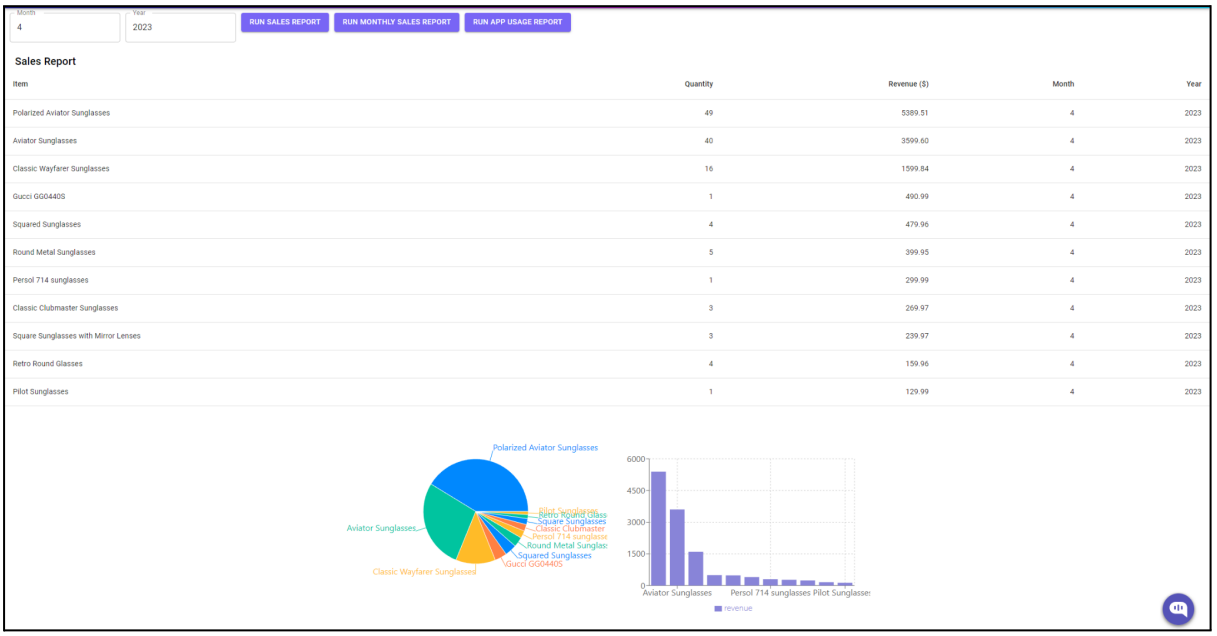


Other UI features

As discussed, the navbar is the site's most important feature as it contains everything the user needs to navigate the site, and is always persistent and accessible no matter where the user is.

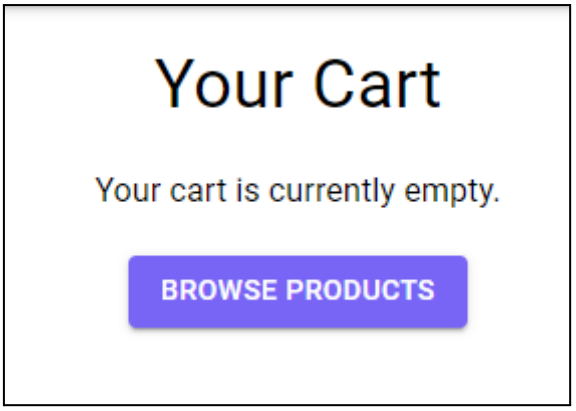
The E-commerce site is fully implemented with every requirement functional, and the demo section in the README file on our Github repo will showcase all of the webpages and their interfaces. For the purpose of this section, it is redundant to show every page on our site as that will be included in the demo. Rather, the section will be following some important features.

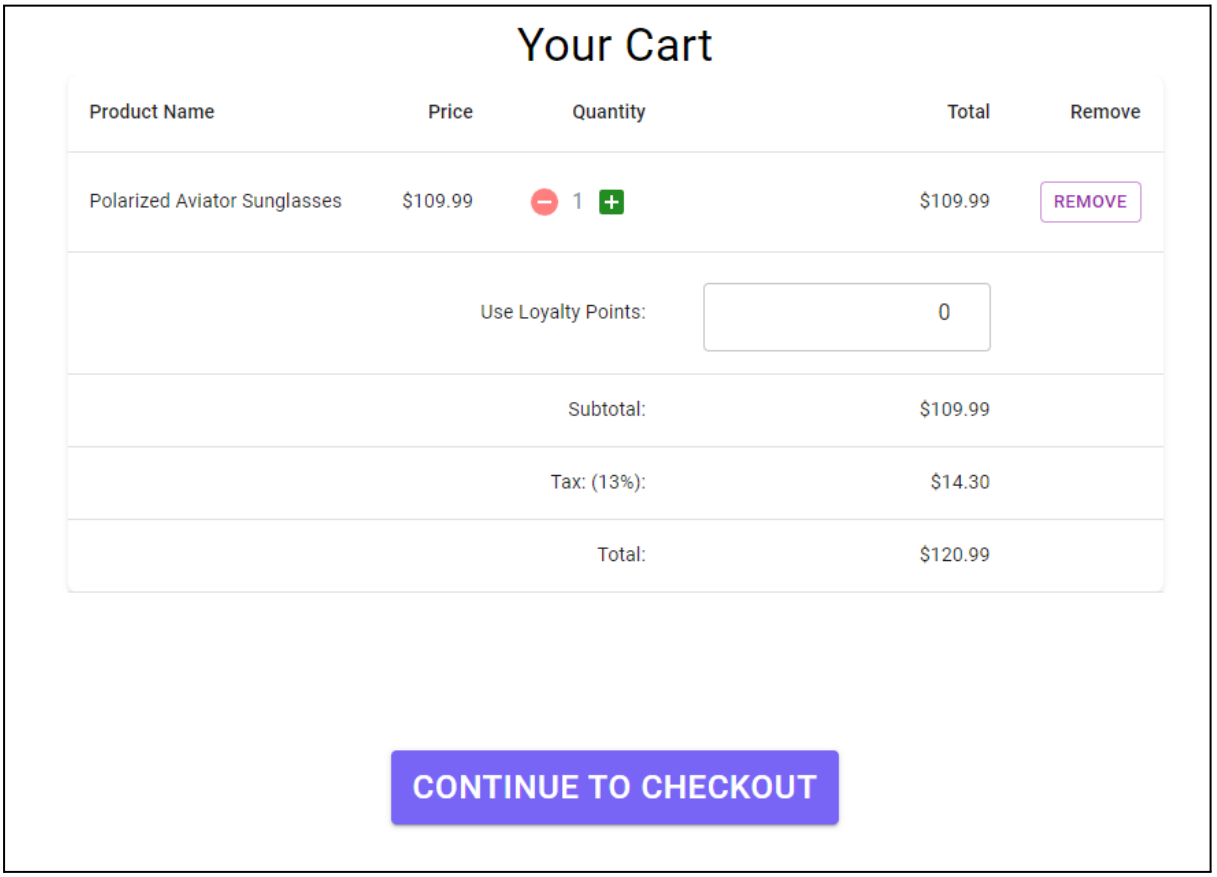
As mentioned previously, the UI in our website is very straightforward and simple to follow. The MUI buttons on every page make it easy to see and use the site. For example, in the admin view tab, we have several reports that we can run:



For an administrator, the buttons make it easy to generate complex reports such as the sales report for April, 2023, seen above. Using *recharts*, a library that generates graphs with JSON data from our backend, we display useful information visually to the admin as well.

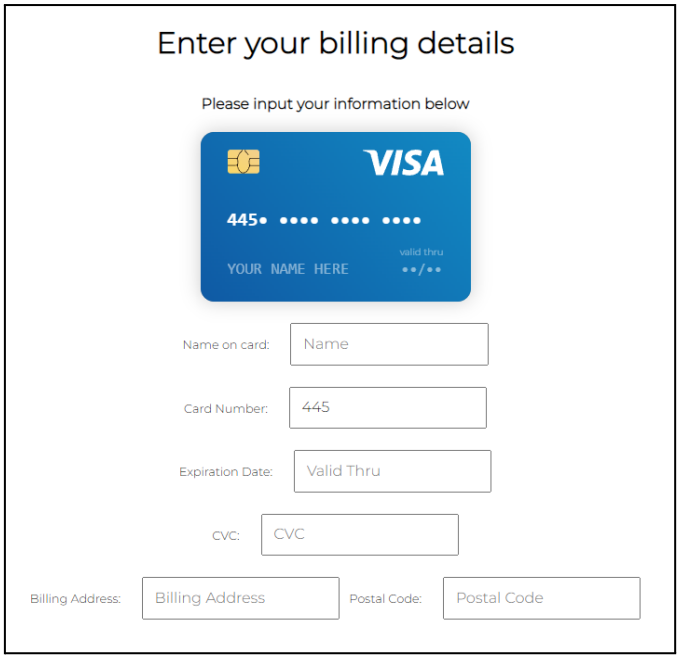
The next distinct feature of the site is the Cart screen, where a user decides what they want in the cart before proceeding to payment.



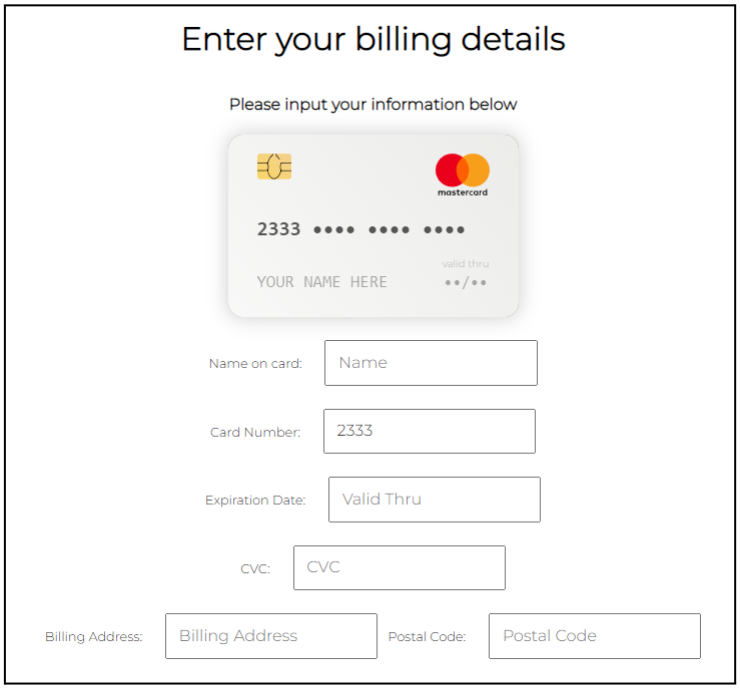


Here, the cart is an easy to understand interface that includes the use of loyalty points, and two buttons for changing quantities. The users can also remove items entirely from the cart. The cart is always updated on our database, so that the user will have the ability to have the same cart under the same user, even on different devices. The user will not be able to proceed to checkout unless they have things in their cart, as seen by the first screenshot above.

Moving, we decided to implement an interactive credit card checkout system that mimics the behaviour in the description (rejects every 3rd request). The credit card screen is responsive, for example, when starting a card with a visa number (card numbers start with 4), the card on screen will change to visa:



Whereas if they have a mastercard (for example), it will change to it if their card number starts with 2:

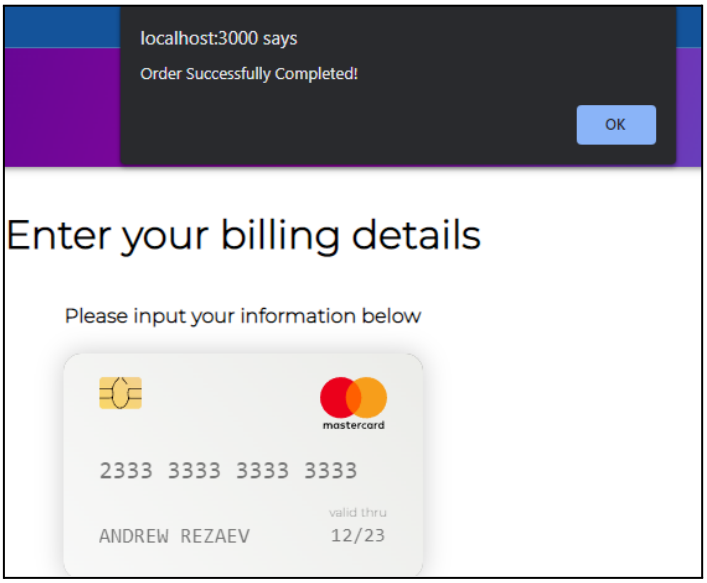


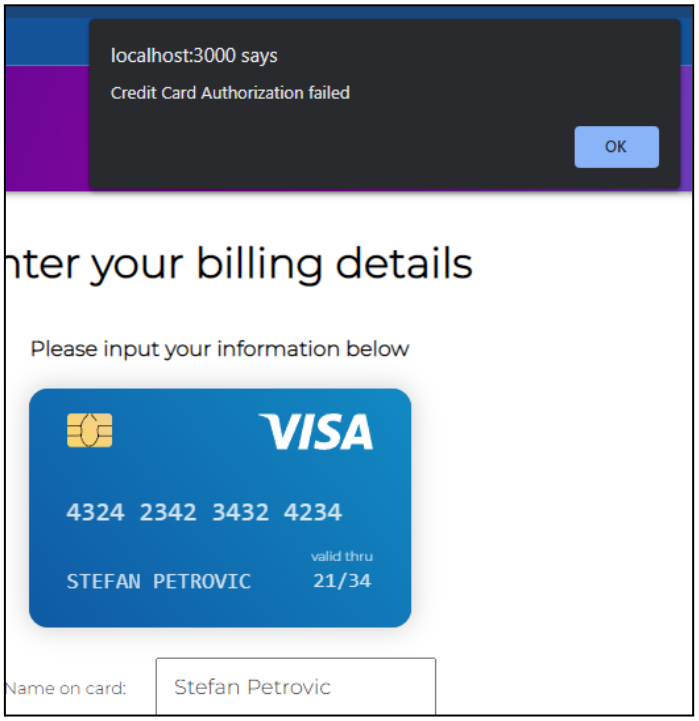
Again, providing such responsiveness even in a screen like this provides the user with more positive and easier to understand interfaces when using our website.

Lastly, the tab icon and tab title for Sunsational Shades has also been implemented as follows, so that the user can keep track in their browser:

Error Handling

In terms of error handling, we send alerts to the front-end to make it easy to understand that something went wrong. Like in the case of the credit cards above, if their payment was successful or rejected, the user would know immediately:





This can also be seen across various other features, as errors will always be displayed in the fashion of an alert.

Conclusion

In conclusion, the team went for a minimalistic approach to the website, displaying lots of visuals and easy to follow buttons which are also easy to find on the site at any time (you

can’t miss them!). For any users of any kind, we believe that the UI for SunSational Shades is very well implemented, easy to navigate, and accessible.

7 Activities Plan

7.1 Project Backlog and Sprint Backlog

- Project and Sprint backlog can be accessed at the following link: Gantt Chart

7.2 Group Meeting Logs

Present Group Members	Meeting Date	Issues Discussed / Resolved
All members present	29/03/2023	- Discussed roles for third deliverable - Discussed styles for our front-end implementation
All members present	30/03/2023	- Began working on front-end. Every member worked on the front-end aspect that they had developed in the backend
All members present	31/03/2023	- Discussed progress and evaluated issues and concerns - Evaluated website bugs
All members present	04/04/2023	- Tested features: bot, virtual tryon, add to cart
All members present	05/04/2023	- Added profile view to the site
All members present	06/04/2023	- Final touchpoints and final styling touches to the website - Began working on report
All members present	07/04/2023	- Discussed deployment and began deployment process - Worked on the design document
All members present	08/04/2023	- Deployed the website using Netlify and AWS - Finished working on report
All members present	09/04/2023	- Final touchpoint before the third deliverable

Figure 6.2: Group Meetings Log

8 Server-Side Implementation

8.1 Back-End Technologies and REST principle

The server-side implementation of an e-commerce website was crucial for ensuring that the application was secure, scalable, and performant. In this context, it was important for us to follow the REST architectural style to ensure that the back-end of the application was designed in a way that was modular, stateless, and easy to maintain.

We chose Node.js and Express as popular choices for building server-side applications in JavaScript, as they offered a wide range of features and capabilities for implementing a RESTful API. Using Express, we defined routes and endpoints that corresponded to specific resources, such as products, orders, and customers. This made it easy to organize the API and ensure that each resource was accessible via a unique URL that mapped to specific CRUD operations, such as GET, POST, PUT, and DELETE.

When implementing a RESTful API, we designed the API in a way that was stateless, meaning that each request contained all the information necessary to complete it. This helped to ensure that the API was scalable, as it avoided the need for maintaining session state on the server side. Furthermore, RESTful APIs used HTTP response codes to indicate the outcome of each request, such as 200 OK for successful requests and 404 Not Found for requests that could not be completed.

To implement the back-end of the e-commerce website, we chose MongoDB as a popular choice for NoSQL databases, as it offered a flexible data model that could handle complex data structures. We also used Mongoose as an ORM(Object Relational Mapper) that provided data modeling and validation capabilities, which helped to ensure that data was stored correctly and efficiently in the database.

In addition to database integration, we implemented authentication and authorization, which were essential for securing the e-commerce website. We used jwtAuth, a popular library for implementing JSON Web Tokens (JWTs) for user authentication and authorization. Using jwtAuth, we ensured that only authorized users were able to access protected resources on the server side, such as placing orders, viewing customer information, and modifying product details.

In conclusion, we implemented the server-side of the e-commerce website using Node.js, Express, MongoDB, and Mongoose, and followed the REST architectural style. This helped to create a secure, scalable, and performant application. By using best practices for API design and choosing appropriate technologies for database integration and authentication, we created a robust back-end that supported the needs of the e-commerce website.

8.2 Chatbot and Virtual-try-On Technologies

8.2.1 Chatbot

We used Dialogflow and Kommunicate, two powerful tools that can be used to build a chatbot for an e-commerce website. Dialogflow is Google's natural language processing (NLP) platform that can be used to understand and interpret user requests, while Kommunicate is a chatbot platform that is used to build and deploy chatbots across multiple channels.

We used Dialogflow to define the intents and entities that our chatbot could understand. For our e-commerce website, these included intents such as "search for a product," "place an order," or "check order status," as well as entities such as product names, order IDs, or shipping addresses.

We used Kommunicate to connect our Dialogflow agent to our e-commerce website. Kommunicate provided an integration with Dialogflow that allowed us to easily deploy our chatbot on our e-commerce website by providing us with a unique API key that we used to integrate the chatbot with our website.

By using Dialogflow and Kommunicate to build a chatbot for our e-commerce website, we were able to provide our customers with an efficient and personalized way to interact with our business. This helped to improve customer satisfaction, increase sales, and reduce support costs.

8.2.1 Virtual Try-On

For the "virtual try-on items" feature, we used @lastcode802/GlassArView, a GitHub repository that provides a framework for creating AR try-on experiences. This Github Repository makes use of JeelizVTOWidget node module that provides the neural networks to track the user's face and render the 3d model of the glasses accordingly. We also created multiple 3D models of glasses using Jeeliz Glasses Studio 3d to convert our 3d models into JSON, which is the acceptable format of the above-mentioned technology to render them in the try-on feature.

To implement the virtual try-on feature, we first loaded the 3D models of the glasses into the GlassArView framework. We also added a camera feed that captures the user's face in real-time. The GlassArView then uses ARKit or ARCore (depending on the device) to detect the user's face and aligns the 3D glasses models accordingly.
The user can then move their head to see how the glasses would look on them in real-time. The GlassArView also allows the user to change the position of the glasses to fit their face.

Once the user finds the perfect fit, they can take a screenshot or save the image to share with others.

Overall, these technologies work together to provide an interactive and engaging experience for our users, making it easier for them to interact with our brand and try on products virtually.

9 Client-Side Implementation

React Usage and Benefits:
For the side client implementation, React was a key technology used in building our sunglasses e-commerce site. We chose to use React due to its many benefits for building modern, responsive user interfaces. With React, we were able to create reusable components that allowed us to efficiently manage the state of our application, resulting in a faster and more responsive user experience. We were able to use pre-built components like React Credit Card to create a secure and user-friendly checkout process, while also integrating features like Jeeliz Virtual Tryon to allow users to virtually try on sunglasses before making a purchase. In addition, using React allowed us to easily integrate with other technologies like Axios, which we used to communicate with APIs and fetch product data from a database.So, using React was a wise choice for our sunglasses e-commerce site as it allowed us to build a scalable, performant, and engaging user interface that has resulted in higher customer satisfaction and increased sales.

Why did we choose to use React?
We chose to use React as our front-end development library when building our sunglasses e-commerce site. By breaking down our user interface into smaller, reusable components, we were able to save time and reduce the amount of code needed, while also making it easier to maintain and update the application over time. We leveraged React's virtual DOM to efficiently handle the application state, resulting in a faster and more responsive user experience. In addition, we took advantage of React's wide range of available tools and libraries, such as Material UI, to create a consistent and visually appealing user interface. Using Axios, we were able to easily integrate with other technologies, making it possible to communicate with APIs and fetch product data from a database. Overall, by choosing to use React, we were able to build a modern and engaging e-commerce site that has resulted in higher customer satisfaction and increased sales.

What are the packages we have used:
At our sunglasses e-commerce site, we utilised a number of third-party tools and libraries to enhance the user experience and improve the functionality of our application

1. **Material UI:** As a pre-built component library, Material UI provided us with a range of ready-made components that we could easily implement in our site, such as buttons, forms, and navigation elements. Using Material UI helped us to maintain a consistent look and feel throughout the site, while also keeping up with modern design trends.
2. **Recharts:** This tool was used specifically for the admin view of our site, allowing us to display data such as sales and performance metrics in an easy-to-understand format. Recharts provided us with a range of chart types to choose from, as well as customizable colours and labels, making it a highly useful tool for data visualisation.

3. **Jeeliz Virtual Tryon API:** This tool provided our site with a unique feature that allowed customers to upload a photo of themselves and virtually try on different pairs of sunglasses. This not only provided an engaging experience for our users, but also increased the likelihood of a purchase by helping them to make more informed decisions about which sunglasses to buy.
4. **Axios middleware**: By using Axios middleware, we were able to easily communicate with APIs and fetch product data from a database, making our site more robust and scalable. Axios provided us with a range of features such as automatic transformation of response data and cancellation of requests, making it a highly useful tool for integrating with other technologies.
5. **React Credit Card:** This tool allowed us to create a secure and user-friendly checkout process, even though we did not handle payment processing ourselves. By rejecting every third request, we were able to further enhance the security of our application. Overall, React Credit Card helped us to create a smooth and hassle-free checkout process for our users, which is essential for any e-commerce site.

10 Demo Section

Feature to test/check during the demo (35% of Deliverables 3)	Mark	Comment made by the TA/prof
1) The system is cloud-native (4%)		
2) It is possible to register (2%)		
3) It is possible to sign in (2%)		
4) It is possible to sign out (2%)		
5) The system is secure (e.g., the system is using https in its url) (3%)		
6) It is possible to filter items (2%)		
7) It is possible to sort items (2%)		
8) It is possible to add items in the shopping cart (2%)		
9) It is possible to remove items from the shopping cart (2%)		
10) It is possible to check out items (2%)		
11) It is possible to write reviews on items and rate items using five stars (2%)		
12) It is possible to use the distinguished feature		
13) It is possible to use the chatbot (4%)		
14) It is possible to virtually try on an item (4%)		

Table 1 Grading criteria used for the demo

11 Strengths and Weaknesses

SunSational Shades is a pretty robust system but, like any system, it has its strengths and weaknesses. The strengths of our system are definitely on the visual and user experience side. The UI is intuitive and simple to use, and is visually very pleasing to the eye. Furthermore, security is one of our strengths as well. We protect against XSS and SQL Injection attacks by using the latest industry standards. The entire project was written in MERN (MongoDB, Express.js, React.js, Node.js) which is the industry standard for E-Commerce System development. As such, it is written in Javascript only, meaning it is more simple to read and understand. Although our visual side is the main selling point, the backend isn't much behind either. We use the latest industry authentication and authorization methods with JWT Tokens and Secure Cookies. In addition, our backend is robustly tested, so that an adversary or malicious user is not able to make any unauthorised changes to the database or malicious modifications to the server-side.

The weaknesses of our system are mostly due to our deployment on AWS. The added latency of communication to the cloud and back adds a bit of a performance disadvantage. Among our other weaknesses are the amount of products we have (due to this being an imaginary application), that we do not implement "proper" checkout where your credit card is actually charged, and that our Chatbot and Virtual Try On features are not as robust and advanced as we initially intended them to be.

As far as the technology is concerned - yes, we have covered the MERN architecture, but we wish that instead of doing our labs in JEE and Apache, we did them with a more modern and industrial approach by using MERN and other industry standard frameworks. We also wished we learned how to implement cookies with a more hands-on approach. In addition, we feel like the team project should be released even earlier, arguably in the first week of the semester.

However, despite the system's weaknesses, and after having worked on this project throughout the entire semester, our group is genuinely happy with the end result. We enjoyed every second of our development process and we learned a lot about eCommerce. The experience of working collaboratively on a real-life project has greatly improved our teamwork skills. We faced many challenges along the way, but we managed to overcome them with the help of each other. This project has taught us that collaboration is key in achieving success, not just in web development, but in any field.

Moving forward, we hope to continue improving SunSational Shades and address the weaknesses we have identified. We are also excited to apply the skills and knowledge we have gained from this project to future endeavours. Overall, SunSational Shades has been a challenging and rewarding experience, and we are all very proud of what we have accomplished as a team.

12 Test-Driven Development

Test ID	T001
Category	Error Handling
Requirements Coverage	EH001-Errors-Are-The-Same
Initial Condition	The system must have been initialised, and utilised improperly in order to invoke an error message (e.g., invalid form input)
Procedure	For this test case to be invoked, the client must initiate an error first. This can be done in many different ways, for example, 1. The user clicks checkout 2. When filling in their card details, the client puts in the wrong number 3. The error message displayed should have consistent styling across the entire site, not just in this particular instance.
Expected Outcome	The expected outcome of this case is to ensure that all of the error messages on the site have the same CSS styling, e.g., all error messages have consistent colour, font size, etc.
Notes	Since the error message should have consistent front-end styling throughout the entire site, there are multiple places in which the error may occur, and thus need to be tested with multiple procedures.

Test ID	T002
Category	Payment Details
Requirements Coverage	IV-Valid-Credentials
Initial Condition	The customer must add something to their cart and attempt to buy it.
Procedure	1. Client adds desired product(s) to their cart 2. Client proceeds to the checkout screen 3. Client inputs their payment information in the input form provided (e.g., credit card number, billing address, etc.)
Expected Outcome	The expected outcome of this test case is that the user correctly inputs their payment information without any errors thrown by the server. For example, the user must provide a non-expired payment method. If successfully validated, the user will be brought to a “Thank you for shopping with us” UI, or something similar.

Test ID	T003
Category	User Interface
Requirements Coverage	UI-Blocked-Fields
Initial Condition	The user navigates to a portion of the site with “out of stock” or otherwise “unavailable” items.
Procedure	1. The user finds a product that is either out of stock or unavailable to attempt to purchase it 2. The corresponding disabled fields (such as the add to cart button) should be correctly greyed out and unclickable by the user.
Expected Outcome	The expected outcome is that the user should not be able to add something that is unavailable to their cart, as clients cannot buy out of stock / unavailable items from the store.
Notes	As this is an e-commerce site, this is going to be the only instance of disabled fields on the website that needs to be tested for proper functionality. Users cannot add out-of-stock sunglasses to their cart.

Test ID	T004
Category	User Interface
Requirements Coverage	UI-Filter-By
Initial Condition	The user must navigate to the grid of items available, and attempt to filter them.
Procedure	1. The user navigates to a part of the site in which they can filter items, for example, all sunglasses 2. The client attempts to filter the results on a client need basis, for example, “filter by category”
Expected Outcome	The expected outcome is that the user should be able to correctly filter all results on a given page with all server-provided parameters.

Test ID	T005
Category	User Experience
Requirements Coverage	UX-ChatBot
Initial Condition	The user navigates to the support tools of the site and initialises a new chatbot instance for some help.
Procedure	1. User navigates to the Help page 2. User clicks on ChatBot for some help on products 3. User asks their question to get a generated response from ChatBot
Expected Outcome	The expected outcome of this case is that the user correctly gets a response from the ChatBot without errors.

Test ID	T006
Category	Database
Requirements Coverage	DB-Primary-Keys
Initial Condition	Database initialised so that every table has primary keys, and they are not empty values.
Procedure	1. Administrators check to see if each table has a primary key. 2. Administrators ensure that the corresponding primary keys are not null.
Expected Outcome	The expected outcome of the test case is that each table has a primary key column of non-null value.

Test ID	T007
Category	Database
Requirements Coverage	DB-Correct-Values
Initial Condition	The database must be initialised with correct value ranges for every field
Procedure	1. Administrators ensure the correct value when initialising the database tables. 2. For example, if storing information about a postal code field, this should not be longer than varchar(6), as postal codes follow the same format.
Expected Outcome	The expected outcome of the test case is to ensure that each field takes and stores the correct field and type to optimise database management and maintenance.

Test ID	T008

Category	User Experience
Requirements Coverage	UX-Page-Load-Time
Initial Condition	A client connects to the server or attempts to load the site/webpage from a stable connection.
Procedure	1. The user attempts visits the site or a particular webpage from the site
Expected Outcome	The expected outcome of the test case is to check that the user can access the site in a reasonable amount of time, and the webpage doesn't hang when visited, especially for the first time.
Notes	Many modern browsers now cache some parts of sites so that they have faster load times on subsequent visits. The most heavy objects that affect load time are usually things like images (especially if high resolution) and other data intensive objects, So, it is important to ensure that the load time is stable, reliable, and quick, as this will greatly enhance user experience (especially on first visits) and retain traffic throughout the site with consistent performance.

Test ID	T009
Category	Security
Requirements Coverage	SC-Captcha-Verify
Initial Condition	The user will be prompted to complete a CAPTCHA upon logging in to the system.
Procedure	1. The user proceeds to login. 2. After typing in login credentials, the user will be prompted to complete a CAPTCHA to verify the user.
Expected Outcome	The expected outcome of the test case is to verify whether the user is human or not. Even with the technologies available nowadays, there is no technology that can accurately solve these puzzles. This, in turn, will provide a layer of security to the system by only allowing *real* people into the website, as well as mitigate spam and denial-of-service(DoS) attacks.

Test ID	T010
Category	Cookies
Requirements Coverage	CK-001
Initial Condition	The user has to be logged in and have a cookie assigned
Procedure	1. The user uses the website and the cookie is being sent back and forth
Expected Outcome	The cookie doesn’t send any plain-text data that would make the user vulnerable to session hijacking attacks. The cookie does not send any passwords either.

Test ID	T011
Category	Database
Requirements Coverage	DB-SQLInjection-1
Initial Condition	The user is logged out and on the login page, the system is up and running
Procedure	1. The user provides a SQL injection command in the username/password field. 2. The username input parameter could be **‘; insert into users (...) ;--**
Expected Outcome	The database rejects such an input and the input page prompts the user to try signing in again.
Notes	Tests db modifications

Test ID	T012
Category	Database
Requirements Coverage	DB-SQLInjection-2
Initial Condition	The user is logged out and on the login page, the system is up and running
Procedure	3. The user provides a SQL injection command in the username/password field. 4. The username input parameter could be **‘ OR 1=1;--**
Expected Outcome	The database rejects such an input and the input page prompts the user to try signing in again.
Notes	Tests db column selection

Test ID	T013
Category	Main Products Page (grid)
Requirements Coverage	MP-001
Initial Condition	A user is logged in and on the main page (grid)
Procedure	1. The user scrolls up and down the page 2. The user clicks on items/adds them to cart/sorts/filters 3. The user stops any action
Expected Outcome	All items are correctly displayed on the screen, there are no duplicated or missing items

Test ID	T014
Category	Check “Show Details” for Inventory items
Requirements Coverage	SD-001
Initial Condition	The user has to be logged in and on the main items page
Procedure	1. The user clicks on show details for a particular item 2. The website redirects the user to a specific page for that item
Expected Outcome	When the user is redirected, the item information is shown correctly

Test ID	T015
Category	User Interface
Requirements Coverage	UI2-Sort-By
Initial Condition	The user must navigate to the grid of items available, and attempt to sort them by any category.
Procedure	1. The user logs into the website and is on the main product page 2. The user sorts by any category
Expected Outcome	The products are correctly sorted (by price, by name)

Test ID	T016
Category	Customer Login
Requirements Coverage	UC1 - User Login Main Page
Initial Condition	System initiated and running, user on the login page
Procedure	*1. The user provides a username* *2. The user provides a password* *3. The user logs-in into the system and is presented with the main UI window*)
Expected Outcome	User is redirected from the main page to the home page, navigation bar changes to reflect that the user is logged in.
Notes	When logging in, the user should provide a username that is at least 8 characters long and a password that contains at least 8 characters, with at least 1 uppercase, 1 lowercase, 1 number, and one of the following characters (-, !, -, =, @, #, $, %, ^, &, *)

Test ID	T017
Category	Cookies/JWT
Requirements Coverage	UC2 - User Login Main Page
Initial Condition	System initiated and running, user on the login page
Procedure	*1. The user provides a username* *2. The user provides a password* *3. The user logs-in into the system and is presented with the main UI window*
Expected Outcome	The user receives a JWT Token inside their cookie which is used for authenticating with each request
Notes	When logging in, the user should provide a username that is at least 8 characters long and a password that contains at least 8 characters, with at least 1 uppercase, 1 lowercase, 1 number, and one of the following characters (-, !, -, =, @, #, $, %, ^, &, *)

Test ID	T018
Category	Customer Registration
Requirements Coverage	UR1 - User Registration Main Page
Initial Condition	System initiated and running, user on the register page
Procedure	*1. The user provides a name* *2. The user provides a username* *2. The user provides a password* *4. The user clicks submit*
Expected Outcome	User is redirected from the main page to the home page, navigation bar changes to reflect that the user is logged in.
Notes	When logging in, the user should provide a username that is at least 8 characters long and a password that contains at least 8 characters, with at least 1 uppercase, 1 lowercase, 1 number, and one of the following characters (-, !, -, =, @, #, $, %, ^, &, *)

Test ID	T019
Category	Administrator view - run sales report
Requirements Coverage	AV-001 - Administrator page
Initial Condition	System initiated and running, user logged in (admin)
Procedure	*1. The user goes to the administrator page* *2. The user selects the month for the sales report* *3. The user clicks submit*
Expected Outcome	User is displayed the output with the report of sales for each item in the inventory for the given time period
Notes	The user needs to have administrator privileges - ie. isAdmin value in the Customer database is set to true

Test ID	T020
Category	Administrator View - Run usage report
Requirements Coverage	AV-002 - Administrator page
Initial Condition	System initiated and running, user logged in (admin)
Procedure	*1. The user goes to the administrator page* *2. The user selects the website usage report* *3. The user clicks submit*
Expected Outcome	User is displayed the output with the report of usage of the website for the given time period
Notes	The user needs to have administrator privileges - ie. isAdmin value in the Customer database is set to true

Test ID	T021
Category	Navigation Bar
Requirements Coverage	NB-001 - Any front end page
Initial Condition	System initiated and running, user anywhere on the website
Procedure	*1. The user clicks on one of the tabs in the navigation bar (eg. Login)* *2. The user clicks on another tab*
Expected Outcome	User is redirected to the appropriate page with each tab click, the navigation bar changes colour for each clicked button, front-end state and cookies remain the same

Test ID	T022
Category	Chat Bot
Requirements Coverage	CB-001 - Chatbot Usage
Initial Condition	System initiated and running, user anywhere on the website
Procedure	*1. The user clicks the chatbot icon in the lower-right corner of the website* *2. The communicates with the Chatbot by sending and receiving messages*
Expected Outcome	User is able to communicate with the chatbot and send and receive appropriate messages
Notes	The chatbot is going to perform better with more usage.

Test ID	T023
Category	Recommended items
Requirements Coverage	RI-001 - Home Page
Initial Condition	System initiated and running, user on the home page
Procedure	*1. The user opens the home page (whether logged in or not)*
Expected Outcome	User is displayed with 5 recommended items to purchase (the 5 items with the highest ratings)

Test ID	T024
Category	Customer Logout
Requirements Coverage	UC3 - Anywhere on the Website
Initial Condition	System initiated and running, user logged in
Procedure	*1. The user clicks the logout button*
Expected Outcome	User is redirected to the home page, navigation bar changes to reflect that the user is logged out. The cookie with the JWT token is deleted from the browser

Test ID	T025
Category	User Item Review
Requirements Coverage	UR-001 - Product details page
Initial Condition	System initiated and running, user logged in, on the home or products page
Procedure	*1. The user selects a product and and views it in details* *2. The user leaves a Review Rating*
Expected Outcome	The item gets updated with the latest review. The item rating is an average of all user reviews
Notes	The user has to be logged to leave a review

Test ID	T026
Category	User Item Comment
Requirements Coverage	UR-002 - Product details page
Initial Condition	System initiated and running, user logged in, on the home or products page
Procedure	*1. The user selects a product and and views it in details* *2. The user leaves a Comment on a product*
Expected Outcome	The item gets updated with the latest comment. Users can see all comments.
Notes	The user has to be logged to leave a comment

Test ID	T027
Category	Checkout
Requirements Coverage	CO-001 - Checkout Page
Initial Condition	System initiated and running, user logged in and on the checkout page
Procedure	*1. The user provides their credit card information* *2. The user provides their shipping information* *3. The user submits the order*
Expected Outcome	User is redirected to a order confirmation page - is given an order number and
Notes	User must input a valid Credit Card number

Test ID	T028
Category	Checkout
Requirements Coverage	CO-002 - Checkout Page
Initial Condition	System initiated and running, user logged in and on the checkout page
Procedure	*1. The user provides their credit card information (illegal credit card number)* *2. The user provides their shipping information* *3. The user submits the order*
Expected Outcome	User remains on the same page, order does not go through, user is notified via an alert about the wrong credit card information and is prompted to try again

Test ID	T029
Category	Password Change
Requirements Coverage	PC-001 - Password Change Page
Initial Condition	System initiated and running, user on the password change page
Procedure	*1. The user provides a username* *2. The user provides password 1* *3. The user provides password 2*
Expected Outcome	User password is changed - user redirected to the login page.
Notes	User account has to exist.

Test ID	T030
Category	Customer Login
Requirements Coverage	UC1 - User Login Main Page
Initial Condition	System initiated and running, user on the login page
Procedure	*1. The user provides a username* *2. The user provides a password* *3. The user logs-in into the system and is presented with the main UI window*)
Expected Outcome	User is redirected from the main page to the home page, navigation bar changes to reflect that the user is logged in.
Notes	When logging in, the user should provide a username that is at least 8 characters long and a password that contains at least 8 characters, with at least 1 uppercase, 1 lowercase, 1 number, and one of the following characters (-, !, -, =, @, #, $, %, ^, &, *)

Test ID	T031
Category	Customer Login
Requirements Coverage	UC4 - User Login Main Page
Initial Condition	System initiated and running, user on the login page
Procedure	*1. The user provides a username* *2. The user provides a password (of a correct format, but the incorrect password for an account)*
Expected Outcome	User is not logged in, they get an alert saying that the password is incorrect. Cookie with a JWT Token is not in the browser.
Notes	When logging in, the user should provide a username that is at least 8 characters long and a password that contains at least 8 characters, with at least 1 uppercase, 1 lowercase, 1 number, and one of the following characters (-, !, -, =, @, #, $, %, ^, &, *)

Test ID	T032
Category	Customer Login
Requirements Coverage	UC5 - User Login Main Page
Initial Condition	System initiated and running, user on the login page
Procedure	*1. The user provides a username which is not registered* *2. The user provides a password*
Expected Outcome	User is not logged in, they get an alert saying that the user does not exist. Cookie with a JWT Token is not in the browser.
Notes	When logging in, the user should provide a username that is at least 8 characters long and a password that contains at least 8 characters, with at least 1 uppercase, 1 lowercase, 1 number, and one of the following characters (-, !, -, =, @, #, $, %, ^, &, *)