



Roman Kuzero

E-commerce Application Using the MERN Stack with TypeScript & Redux Toolkit

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

3 December 2023

Abstract

Author: Roman Kuzero
Title: E-commerce Application Using the MERN Stack with TypeScript & Redux Toolkit
Number of Pages: 34 pages
Date: 3 December 2023

Degree: Bachelor of Engineering
Degree Programme: Information Technology
Professional Major: Mobile Solutions
Supervisors: Amir Dirin, Project Manager

The goal of this final year project was to demonstrate the power of the modern MERN stack coupled with TypeScript and Redux Toolkit through the development of the 'qzeromarket' e-commerce application. This platform is specifically designed to offer a seamless user experience, focusing on robust user interaction.

In the back-end, the application is crafted using Node.js and Express, establishing a scalable, maintainable architecture that integrates with MongoDB for secure data management. User security remains a top priority, reinforced by the use of JWT for robust authentication. TypeScript and Redux Toolkit collaborate to ensure smooth navigation and consistency across the platform.

The user interface design is characterized by clarity and visual appeal, adhering to modern design principles that immerse users in an engaging experience. Testing, guided by heuristic evaluation and stringent unit testing protocols, guarantees exceptional performance and unwavering reliability.

In conclusion, 'qzeromarket' successfully met its goals, integrating current tech trends and best practices. The readable and maintainable codebase emphasizes coding for humans and machines alike. Rigorous testing has confirmed the application's reliability and functionality, establishing it as a robust, user-focused platform that effectively achieves its intended outcomes.

Keywords: MERN Stack, E-commerce

Contents

1	Introduction	3
2	Research Questions and Methodology	3
2.1	Research Questions	4
2.2	Methodology	4
3	Foundations of E-commerce	5
3.1	Evolution of E-commerce	5
3.2	E-commerce Business Models	6
3.2.1	Business-to-business (B2B)	6
3.2.2	Business-to-consumer (B2C)	6
3.2.3	Consumer-to-business (C2B)	6
3.2.4	Consumer-to-consumer (C2C)	7
3.2.5	Other Existing Business Models	7
3.3	Understanding the Tools of E-commerce	7
4	Exploring Modern E-commerce Development Stacks	8
4.1	LAMP Stack	8
4.2	MEAN Stack	8
4.3	ASP.NET Core	9
4.4	MERN Stack	9
4.4.1	React	9
4.4.2	TypeScript	10
4.4.3	Redux and Redux Toolkit	11
5	System Design and Visualization	12
5.1	Use Case Diagram	12
5.2	ERD: Concept and Choice	13
5.3	Design Principles and Objectives	14
5.4	User Interface Design	14
5.4.1	Overview of All Pages	15
5.4.2	Store Page	15
6	Implementation	17

	ad
6.1 Technologies Used	17
6.1.1 Back-end	17
6.1.2 Front-end	17
6.2 Architecture	18
6.2.1 API	18
6.2.2 Client	19
6.3 UML Diagrams	21
6.3.1 Sequence Diagram	21
6.3.2 Activity Diagram	22
6.4 Code Implementation	23
6.4.1 Redux Store Configuration	23
6.4.2 Redux Slice – Shopping Cart	24
7 Testing	25
7.1 UI Testing	25
7.1.1 Nielsen's Heuristic	25
7.2 Unit Testing	27
7.2.1 Unit testing in Redux Toolkit	29
7.2.2 Back-end Unit Testing	31
7.2.3 Testing with Postman	32
8 Discussion and Conclusion	33
8.1 Discussion	33
8.2 Conclusion	34
References	34

Appendices

Appendix 1: Detailed UI screenshots of "QZM" e-commerce application

Appendix 2: Heuristic Evaluation workbook pages

List of Abbreviations

- API:** Application Programming Interface, a set of tools and protocols that allows different software applications to communicate with each other
- AWS:** Amazon Web Services, a subsidiary of Amazon providing on-demand cloud computing platforms and APIs
- B2G:** Business to Government, refers to businesses selling products or services directly to government entities
- C2G:** Consumer to Government, refers to individuals interacting or transacting with government entities, often for services or payments
- DOM:** Document Object Model, a programming interface for web documents, representing the structure of a document as a tree of objects
- HTTP:** Hypertext Transfer Protocol: A protocol used for transferring data over the Internet, especially web pages in the form of HTML documents
- ITU:** The International Telecommunication Union, the United Nations specialized agency for information and communication technologies
- JWT:** JSON Web Token, a compact and self-contained way for securely transmitting information between parties as a JSON object
- LAMP:** Stands for the Linux operating system, the Apache web server, the MySQL database server, and the PHP programming language

- MEAN:** The acronym MEAN stands for MongoDB, Express, Angular, and Node.js
- MERN:** Stands for MongoDB, Express, React, and Node.js
- MVC:** Model-View-Controller: A design pattern that separates an application into three interconnected components: Model (data), View (user interface), and Controller (logic and user input)
- NoSQL:** NoSQL databases (also known as "not only SQL") are non-tabular databases that store data in ways that relational tables do not
- OOP:** Object-Oriented Programming, a programming paradigm based on the concept of "objects", which can contain data and code to manipulate the data
- PHP:** PHP Hypertext Preprocessor, a popular open-source scripting language used for web development, embedded into HTML
- SPA:** Single page application
- SQL:** A structured query language is a programming language used to store and process data in a relational database
- TDD:** Test-Driven Development, a software development approach in which tests are written before the actual code, ensuring that the code functions as intended
- UCD:** User-centered design, an iterative design process that focuses on the users and their needs in each phase of design and development
- UI:** User Interface, the space where interactions between humans and machines occur, primarily focused on the look and layout

URL: Uniform Resource Locator, the address used to access web resources on the Internet

1 Introduction

In today's age the way business is done has been transformed by the rise of e-commerce. This change has affected companies of all sizes. The thesis explores this transformative landscape and discusses the challenges and opportunities the transformation brings.

Moving from brick-and-mortar stores to online platforms comes with various technical hurdles. It requires consideration of security measures and the ability to handle growth smoothly. Building an e-commerce platform that puts users first demands planning. The goal of this study is to shed light on these challenges focusing on aspects such as scalability, flexibility, and security.

One crucial decision businesses face when venturing into e-commerce is choosing the right technology stack. The study is centred on the MERN Stack, as well as technologies such as TypeScript and Redux Toolkit. What advantages or limitations does this combination offer? Exploring this question forms a theme in this study.

In summary, this thesis aims to provide an understanding of the technological components that underpin e-commerce while laying the groundwork for further, in-depth discussion.

2 Research Questions and Methodology

The rise of e-commerce has introduced both challenges and opportunities, making it crucial to comprehend the technological underpinnings that power online platforms. This chapter delineates the research questions guiding this thesis, followed by the methodology employed to address them.

2.1 Research Questions

This section presents the research questions that aim to explore various facets of e-commerce application development.

1. Which technologies are commonly used for e-commerce application development? What differences do they have in terms of scalability, flexibility, and security?
2. What challenges do e-commerce platforms face, both in terms of technology and operations? How do these challenges impact the choice of development tools?
3. Why is the selection of a web development stack important for e-commerce applications, and what factors guide this decision?
4. What distinguishes web development for e-commerce applications from other types of projects? How do these differences affect the choice of technologies and tools?
5. How does the MERN stack, combined with TypeScript and Redux Toolkit, address the specific needs of e-commerce development?

2.2 Methodology

The research began with an examination of the existing literature, which shed light on the challenges faced by the e-commerce sector and guided the subsequent design phase. Application interfaces were crafted in Figma, embracing User-centered Design principles for an optimal user experience.

The e-commerce application was developed utilizing the MERN Stack with TypeScript and the Redux Toolkit, chosen for their reliability and flexibility. Throughout the development phase, a Test-Driven Development methodology was followed to ensure functionality and quality.

Nielsen's Heuristic Evaluation played a role in testing the user interface providing a structured approach to refining it [1]. This method allowed for iterative improvements of the interface in line with established usability principles.

Overall, the research combined analysis, user-focused design, meticulous development practices and strategic UI evaluation to create a robust and user-friendly e-commerce application.

3 Foundations of E-commerce

Since the very beginning of recorded history as well as beyond, commerce (the trade of products and services) has been a key driver for human survival. The past decades have brought a new type of commerce to the top: e-commerce [2]. A growing number of companies is embracing the e-commerce trend as the appeal of traditional physical trading of goods and currency declines.

3.1 Evolution of E-commerce

The inception of e-commerce does not have a specific date since the emergence of the concept of e-commerce arose from a chain of various events. E-commerce acquired a wide range of opportunities with the advent of the Internet, especially after the launch of the first website on August 6, 1991 [2, 3]. Since then, many companies have relocated to websites.

Table 1 provides a timeline of key events in the evolution of e-commerce, illustrating the rapid growth and pivotal developments in the field:

Table 1. Key events in e-commerce evolution

Year	Event
1991	The launch of the first website.
1994	Netscape was created, offering consumers a straightforward web browser with Secure Sockets Layer, a secure online transaction technology.
1995	Amazon and Ebay, the two major e-commerce brands, were introduced.

1997	Business-to-business (B2B) e-commerce is launched. The United States Postal Service issues electronic postal stamps [4].
1998	PayPal, a well-known online payment system, makes its debut.
2000	Peak of the dot-com bubble.
2023	According to the ITU, roughly 5.4 billion people, or 67% of the world's population, are utilizing the Internet in 2023 [5].

3.2 E-commerce Business Models

There are several models of e-commerce available today, including business-to-business (B2B), business-to-consumer (B2C), consumer-to-business (C2B), and consumer-to-consumer (C2C) models, all of which are fundamental forms discussed in detail in the section.

3.2.1 Business-to-business (B2B)

B2B electronic commerce encompasses any electronic product or service transactions between businesses. In general, this strategy is used for electronic trade by manufacturers and conventional industrial wholesale organizations. [6, 7.]

3.2.2 Business-to-consumer (B2C)

In a B2C situation, the seller is a business, and the buyer is a consumer. Because this case resembles physical retailing, it is usually referred to as electronic retailing. It is still a two-way function in most cases, but it is normally done purely over the Internet where an electronic store is put up. [6, 7.]

3.2.3 Consumer-to-business (C2B)

Consumer-to-business e-commerce involves trade between consumers and businesses in which consumers pick the price they want to be charged and vendors decide whether to accept the price or not. This concept is built on three

components: a consumer operating to be a seller, a business serving as a purchaser, and an intermediary that handles the relationship between sellers and buyers. [6, 7.]

3.2.4 Consumer-to-consumer (C2C)

Any electronic exchange of products or services between customers is classified as C2C. This transaction is often handled by a third party that provides an online transaction platform. [6, 7.]

3.2.5 Other Business Models

There are numerous other types of e-commerce available nowadays. Type names differ from one source to the next, and types evolve over time, with new ones appearing. Other e-commerce models include B2G (business-to-government) and C2G (consumer-to-government).

3.3 Understanding the Tools of E-commerce

There are numerous significant dates in the history of e-commerce, as well as numerous business models. However, there are tools and technologies behind every online store. They are the platform's foundation, driving its functionality and growth. A well-designed e-commerce website is more than just visually appealing. If done effectively, it will result in real sales and profits for the e-commerce business by easing the order process for clients and generating brand equity that will help a firm grow. The value of these technological solutions will be discussed further in Section 4.

4 Exploring Modern E-commerce Development Stacks

4.1 LAMP Stack

Any web application is developed with several technologies. The term "stack" refers to the combination of these technologies. This concept was popularized by

the LAMP stack [8]. The acronym "LAMP" stands for the following open-source components:

- Linux: The operating system serves as the base layer, providing the necessary functionality for the other components to function.
- Apache: The Apache web server manages incoming requests and serves web resources via HTTP, enabling anyone in the public domain to access the application using a basic web address.
- MySQL: A relational database management system that is used to store application data. MySQL enables the storage of all information in a format that can be readily queried using the SQL language.
- PHP: A scripting language used in conjunction with Apache to create dynamic websites.

4.2 MEAN Stack

The MEAN stack was among the first open-source stacks to adopt SPA and NoSQL. The MEAN stack's primary language is JavaScript, and the stack consists of the following technologies [8, 9]:

- MongoDB: An open-source, cross-platform document-oriented database management program.
- Express: An open-source server format built entirely in JavaScript and intended for usage with Node.js [10].
- Angular: A development platform built on TypeScript. It provides a component-based architecture for constructing scalable web applications, as well as a library collection encompassing capabilities such as routing, forms management and client-server communication [11].
- Node.js: A JavaScript runtime application and library that allows web applications to be executed outside of the client's browser.

4.3 ASP.NET Core

ASP.NET Core is an open-source, multi-platform framework for developing contemporary, cloud-enabled, Internet-connected applications. ASP.NET Core enables the creation of server-rendered web pages, back-end server applications, and HTTP APIs for mobile applications, among other functionalities. Built on .NET 7, which is the latest version at this time, the framework offers high-performance, cross-platform, and open-source runtime capabilities. [12, 13.]

4.4 MERN Stack

The MERN stack is yet another well-known collection of JavaScript technology. This stack integrates MongoDB, Express, Node.js (described before in the MEAN stack section), and React to facilitate the creation of web and mobile applications [10, 14].

4.4.1 React

Facebook launched React, an open-source JavaScript library for designing user interfaces in 2013 [15]. React, unlike AngularJS, is not a framework. It is a public library. As a result, it does not imply a framework design such as the MVC pattern on its own. React is utilized for rendering views (the 'V' in MVC), while the architecture and implementation of the remaining components of the application are determined by the developer's discretion [8, 16].

To manage dynamic changes, the library employs the concept known as the virtual DOM. When anything changes, React creates a new virtual DOM based on the new truth (state) and compares it to the previous virtual DOM. React then computes the differences between the old and modified Virtual DOMs and applies them to the actual DOM [17, 18]. The above method assures that the real DOM receives minimal modifications, resulting in improved performance. Figure 1 illustrates a visual illustration of this approach.

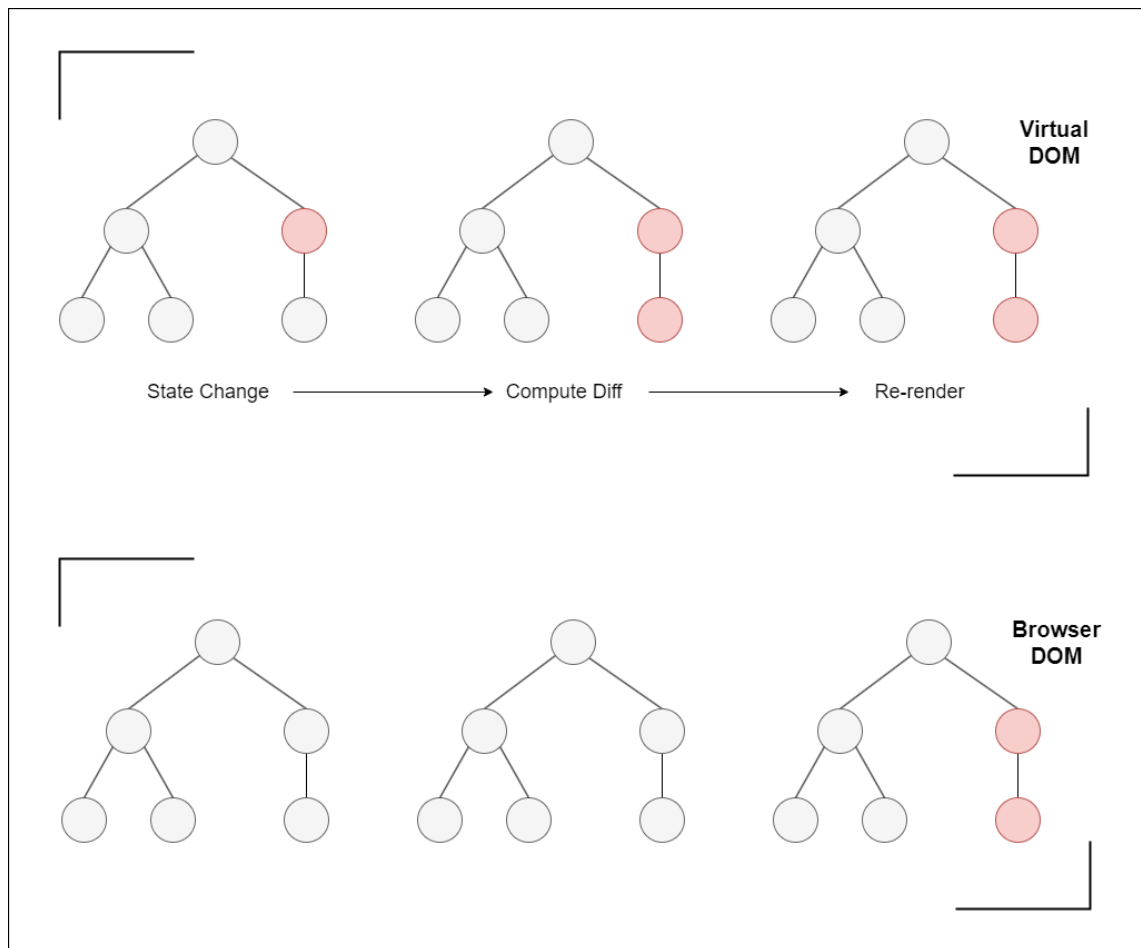


Figure 1. Comparison of virtual DOM update and actual DOM behaviour.

4.4.2 TypeScript

Microsoft TypeScript is an open-source programming language that is created and maintained by the company [19]. According to a Stack Overflow developer survey from 2023, TypeScript, which includes JavaScript, is the fourth most popular programming language. JavaScript is the most popular, followed by Python and SQL [20]. The advantages of TypeScript over JavaScript are listed below [19, 21]:

- **Static Typing:** TypeScript uses static typing for early error detection, whereas JavaScript uses dynamic typing.
- **Compile-time Type Checking:** TypeScript catches type-related errors at compile time, while JavaScript does so at runtime.

- Object-Oriented Features: TypeScript provides a comprehensive implementation of OOP concepts.
- Module Support: TypeScript inherently supports modules for better code organization.
- Ease of Migration: Being a superset of JavaScript, any valid JavaScript code is also valid TypeScript, simplifying the transition for developers.

4.4.3 Redux and Redux Toolkit

Redux is a JavaScript developer tool that keeps the state of apps predictable. It enables developers to build easily testable applications that can run on a variety of platforms such as clients, servers, and even native environments. While Redux is frequently associated with React, it can be used in conjunction with any JavaScript framework or library to simplify state management in online applications.

The key concepts of Redux include:

- Single source of truth: The state of the entire application is stored in one central location, often referred to as the store.
- State is read-only: The only way to change the state is by emitting an action, which is an object that describes what happened.
- Changes are made with pure functions: Actions are processed by pure functions called reducers to produce the next state.

[22.]

The Redux Toolkit is the official, opinionated, all-in-one Redux development toolbox. It was created to support in the simplification of typical Redux use cases and the reduction of boilerplate code. Here is what the Redux Toolkit offers [22]:

- Pre-configured store: The Redux Toolkit provides a function to create a store with middleware and DevTools already set up.
- Simplified reducer creation: With createSlice, developers can generate a reducer and its actions simultaneously.

- **Middleware:** The toolkit comes with the 'redux-thunk' middleware, allowing to build async logic that interacts with the store.

Utilizing Redux Toolkit, developers can more efficiently set up a Redux store, define reducers and actions, and manage side effects, making the whole Redux workflow more straightforward and less error-prone [23].

5 System Design and Visualization

In addition to being aesthetically pleasing, designing a user interface (UI) is about forging a connection between people and computer systems. A well-conceived user interface can significantly enhance the user experience. This chapter will cover the system's design process, detailing the tools utilized, such as Figma, and the methodologies followed. The Use Case Diagram will be discussed, providing clarification on the system's functionality. The primary objective was to create an intuitive and user-focused interface.

5.1 Use Case Diagram

A use case diagram is a type of behaviour diagram in the Unified Modeling Language (UML). The use case diagram illustrates the functional needs of the program. Use case diagrams may be used to understand how the system should function [24].

Figure 2 shows an example of a use case diagram for the e-commerce application.

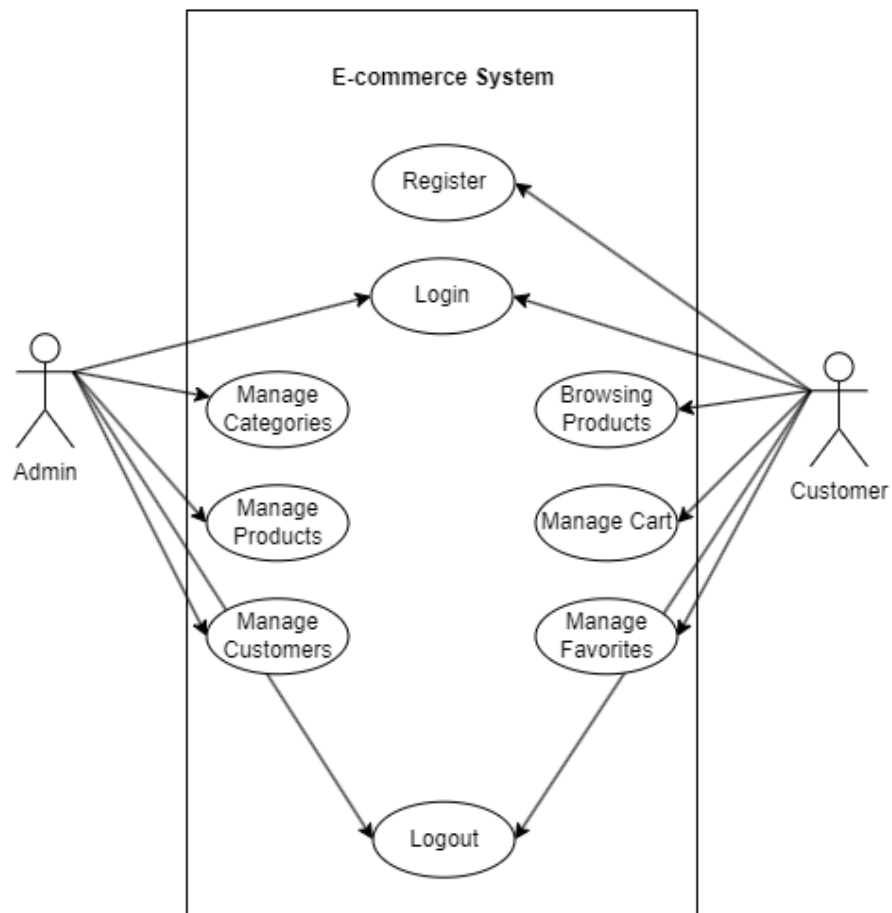


Figure 2. Use case diagram of e-commerce application.

For the created e-commerce system, the use case diagram visually represents interactions between actors (Admin and Customers) and system functionalities, offering clarity on main processes and user interactions.

5.2 ERD: Concept and Choice

As its name suggests, an entity-relationship diagram (ERD) characterizes data through entities and the relationships that bind them. An entity can be understood

as any distinct item—be it a person, place, object, event, or abstract concept—about which data is stored.

It is essential to highlight that ERDs are primarily designed for relational databases. When dealing with NoSQL databases, like the one I implemented in my project, the usual practice often sidesteps the creation of ERDs. This deviation is largely attributed to the adaptable, non-relational nature of NoSQL databases. Consistent with this approach, I chose not to construct an ERD for this project.

5.3 Design Principles and Objectives

While crafting the UI, I grounded my decisions in renowned design principles. My goal was a consistent, functional, and engaging interface.

Embracing the User-centered design (UCD) [25] philosophy, I prioritized understanding user needs and preferences. Every design element was chosen to enhance the user experience, aiming to resonate with users.

Simplicity and intuitiveness were paramount. The interface was designed to integrate aesthetic appeal with functionality while minimizing complexity.

5.4 User Interface Design

The UI makes the system's features accessible. It is the key element of device communication and human-computer interaction. For designing this interface, I chose Figma, a modern, free UI design tool [26]. Two significant screenshots that highlight my methodology and design ethos are presented in the following sections.

5.4.1 Overview of All Pages

Figure 3 offers a visual summary of the application's user interface, highlighting how the various pages are designed to work together seamlessly. For detailed views of each individual page, please refer to Appendix 1.

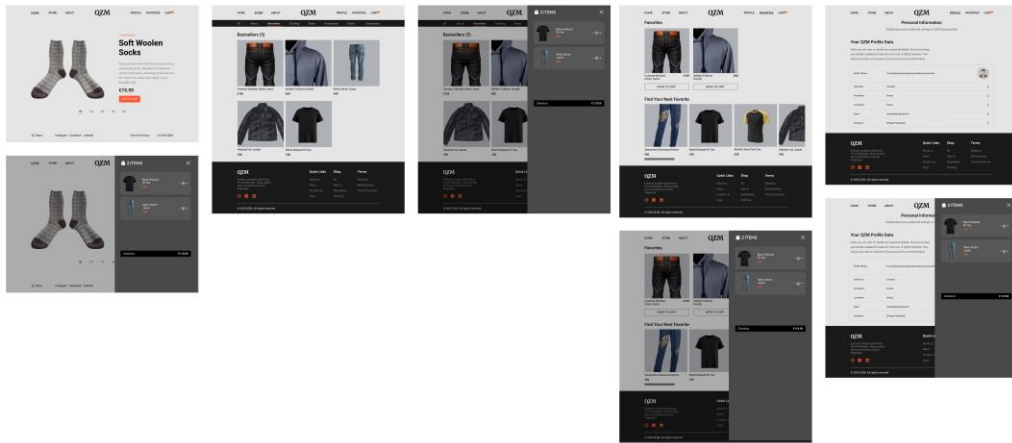


Figure 3. Overview of all pages.

This figure provides a snapshot of the system's UI, demonstrating the consistency in design across the entire application. It visually conveys the structured layout and the logical flow between different sections, illustrating the careful planning that went into the interface development.

5.4.2 Store Page

The Store Page is a key interface within e-commerce platforms, designed to present products and enable smooth user navigation. Figure 4 illustrates the UI design of the Store Page, emphasizing its structured layout and user-friendly design.

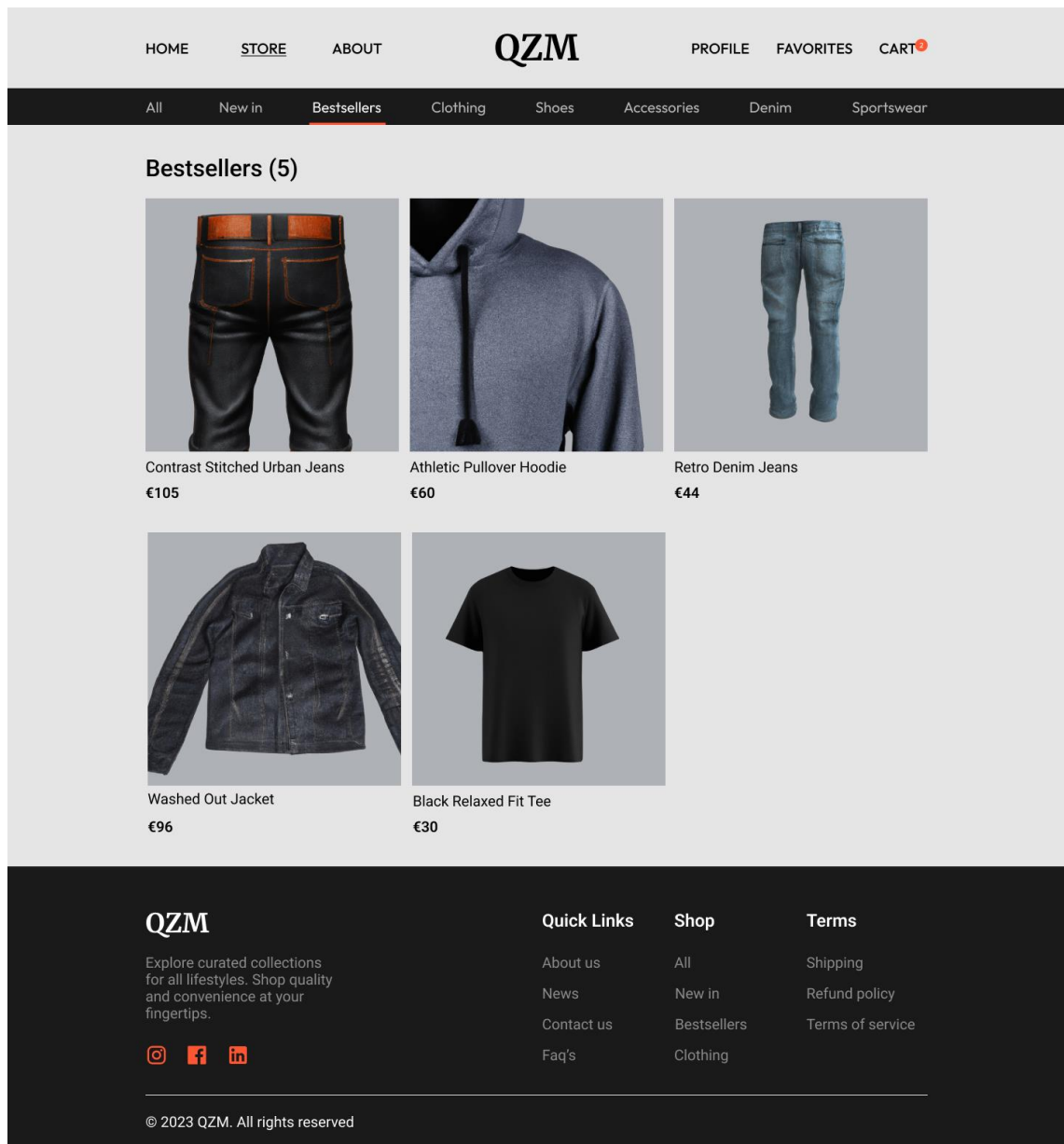


Figure 4. UI design of store page.

The Store page, the core of an e-commerce platform, displays products and facilitates user engagement through intuitive design and organized layouts.

6 Implementation

6.1 Technologies Used

6.1.1 Back-end

The back-end of the e-commerce system serves as the backbone of the application, handling data management, server-side logic, and authentication processes. The following technologies were carefully selected to ensure robust performance, security, and scalability.

- Database: MongoDB backed by the Mongoose library was vital for our flexible and scalable data storage requirements.
- Framework: Node.js with Express formed the framework, streamlining efficient request handling.
- Authentication: JWT ensured user identity verification seamlessly [27].
- Image Storage: AWS S3 provided reliable server-side image storage.
- Payment: Transactions were secure and smooth with the Stripe API [28].

6.1.2 Front-end

The front-end of the application is the interface through which users interact with the e-commerce platform. It was imperative to select technologies that not only deliver an engaging and responsive user experience but also align with the overall architecture of the system. The technologies chosen for the front-end are outlined below.

- Framework: ReactJS's efficient rendering with the virtual DOM was crucial.
- Design: MUI (Material-UI) provided modern, responsive design components [29].
- State Management: The Redux Toolkit modernized our state management.

6.2 Architecture

The application consists of two main parts: the front-end (Client) and the back-end (API). The Client and the API are discussed in more detail below.

6.2.1 API

The back-end is organized according to a clean, modular architecture pattern, which is as follows:

- Controllers handle the logic for the API endpoints. They interact with the models and services to fetch or manipulate data, then send a response to the client.
- Helpers are utility functions or constants that assist in performing common tasks across different parts of the application.
- Middlewares are functions that have access to the request object, the response object, and the next function in the application's request-response cycle. They can be used for tasks such as logging and authentication.
- Models define the structure of the database collections and the relationship between the data.
- Routes determine how an application responds to client requests to particular endpoints.
- Services store/stores business logic. They interact with the database, execute computations, and so forth.
- Types include/includes custom data types or interfaces used across the application to maintain consistency.
- Utils are general utility functions or constants that don't fit into helpers.

The entry point for the server starts with `server.ts`, which initializes and sets up the server, and `app.ts`, which brings together all the modules.

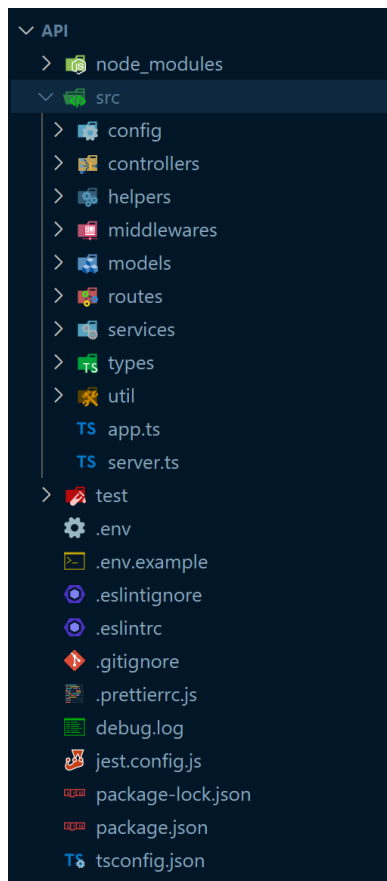


Figure 5. Server-side folder structure.

6.2.2 Client

The front-end architecture, also written in TypeScript, is structured as follows:

- Assets: Contains static files such as images, icons, or fonts.
- Components: Reusable UI pieces that can be combined in various ways across pages.
- Hooks: Custom React hooks to encapsulate logic that can be shared across components.
- Pages: Each file or folder in this section corresponds to a route in the application. They assemble components to form complete views.
- Redux: Manages the application's state. The slices folder contains reducers and actions, while store.ts sets up the global store.

- Styles: This folder contains custom-styled components using Material-UI, tailoring the look and feel of the application for a consistent and modern design.
- Types: TypeScript interfaces and types used across different components and utilities.

The application starts from `index.tsx`, which renders the `App.tsx` component and wraps the whole application in necessary providers, such as Redux's Provider.

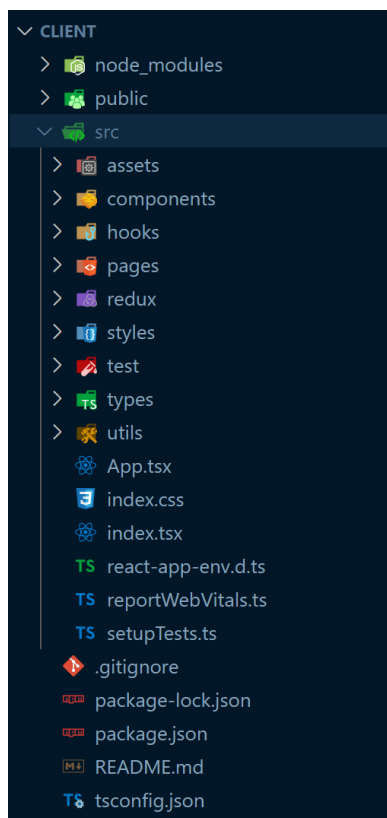


Figure 6. Client-side folder structure.

This folder structure allows for a scalable and maintainable codebase. Each piece of functionality has its place, ensuring that as the project grows, it remains organized and coherent.

6.3 UML Diagrams

To further understand the interactions and user journey within the platform, UML diagrams provide a graphical representation of various processes.

6.3.1 Sequence Diagram

Figure 7 showcases a sequence diagram describing the order of operations in a time-sequenced format, illustrating interactions between different entities in the system.

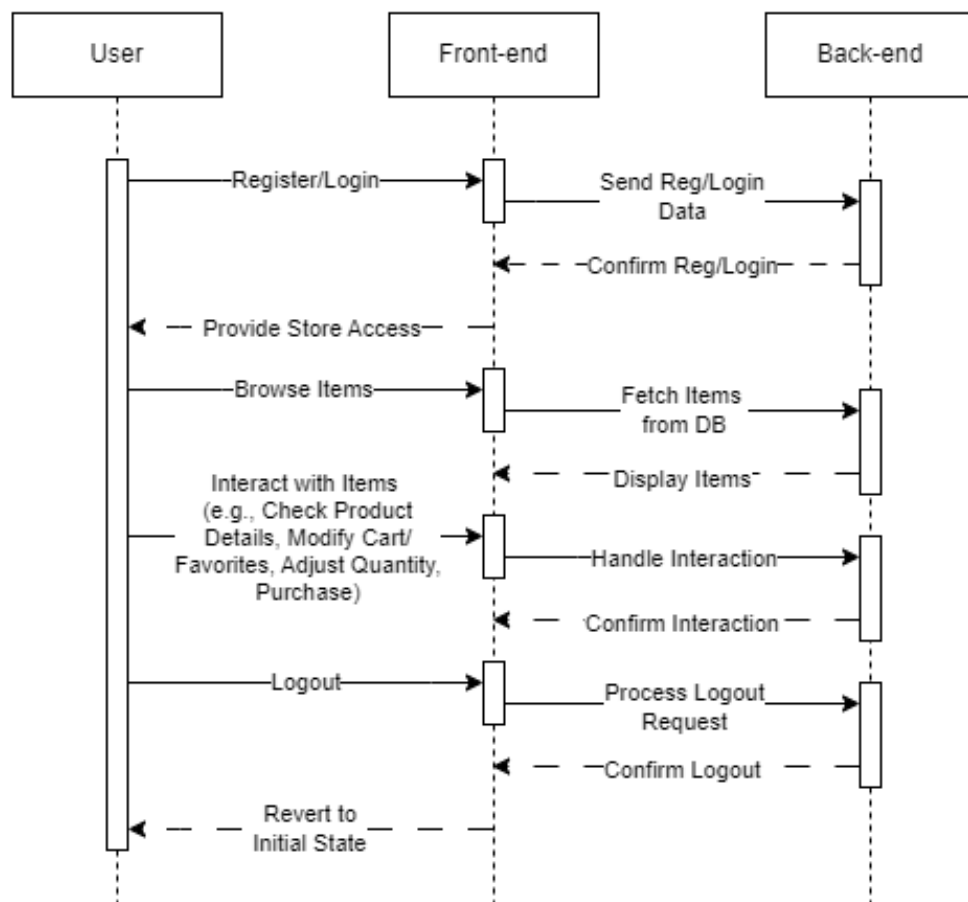


Figure 7. UML sequence diagram.

6.3.2 Activity Diagram

The activity diagram offers a flowchart-style representation of the platform's functionalities and the user's journey through the system.

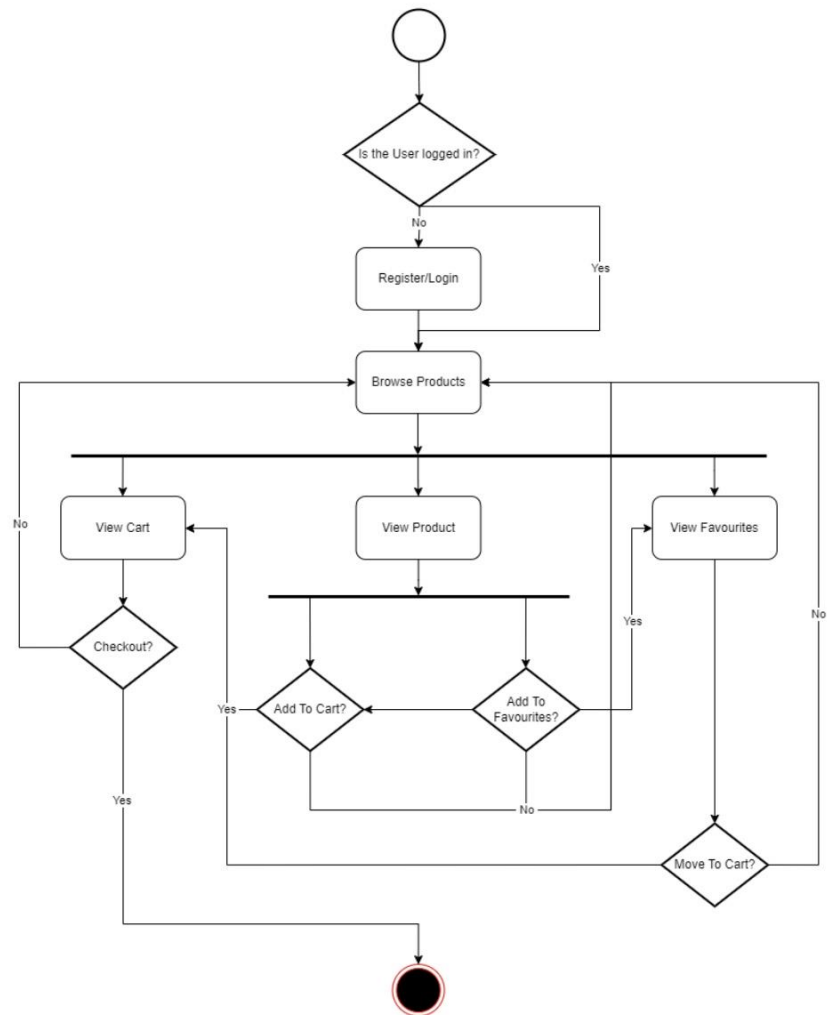


Figure 8. UML activity diagram.

Anyone involved can gain a better understanding of how the platform works and see the steps users take by looking at these diagrams.

6.4 Code Implementation

6.4.1 Redux Store Configuration

For state management, I utilized Redux, along with the Redux Toolkit to simplify common Redux patterns. Listing 1 illustrates a snapshot of how the Redux store was set up with persistence.

```
import { combineReducers, configureStore } from '@reduxjs/toolkit';
import { persistReducer } from 'redux-persist';
import storage from 'redux-persist/lib/storage';

import { userReducer, productReducer, cartReducer, favoritesReducer } from './slices';

const reducers = combineReducers({
  userReducer,
  productReducer,
  cartReducer,
  favoritesReducer,
});

const persistConfig = {
  key: 'root',
  storage,
};

const persistedReducer = persistReducer(persistConfig, reducers);

const store = configureStore({
  reducer: persistedReducer,
});

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

export default store;
```

Listing 1. Redux store configuration.

This setup ensures the application state is persisted across sessions, enhancing the user experience.

6.4.2 Redux Slice – Shopping Cart

To manage the shopping cart's state, a specific Redux slice was created. This slice contains async thunks to fetch and modify the user's shopping cart and reducers to handle the actions.

Listing 2 gives a brief overview.

```
import { createAsyncThunk, createSlice } from '@reduxjs/toolkit';

const initialState: CartSliceState = {
  usersShoppingCart: { _id: '', cartItems: [], totalPrice: 0 }
}

export const getUsersShoppingCart = createAsyncThunk(
  'getUsersShoppingCart',
  async ({userId, token}: GetUsersShoppingCartProps) => {
    // API call logic...
  }
)

// ... (Other async thunks)

const cartSlice = createSlice({
  name: 'cart slice',
  initialState: initialState,
  reducers: {
    countTotalPrice: (state) => {
      // Logic to count total price...
    }
  },
  extraReducers: (builder) => {
    builder.addCase(getUsersShoppingCart.fulfilled, (state, action) => {
      state.usersShoppingCart = action.payload
    })
    // ... (Other cases)
  }
})

export const cartReducer = cartSlice.reducer;
```

Listing 2. Cart slice implementation for shopping cart management.

By employing the Redux Toolkit, the codebase remains concise and modular, with added benefits from enhanced developer tools and middleware.

7 Testing

Since most e-commerce apps are business vital, testers are still learning how to best test them, and the industry is still broad and growing. Millions of dollars have been invested on websites, and investors anticipate success. Unfortunately, the history of e-commerce is littered with costly failures some of which may have been prevented if the facility had been tested more thoroughly before it was released to the public [30].

The negative effects of a poorly functioning website are startling, and they even affect the actual brick-and-mortar stores associated with the online platform. Based on the findings from past research, errors identified on an e-commerce site led to significant customer dissatisfaction: 28% of customers quit shopping at the site, 23% stop buying from the site, and 6% were so dissatisfied that they stopped buying at the brick-and-mortar store on which the site is based [31]. Customers may have an idea or an impression that if the firm cannot give a decent website, they may not be able to provide a quality product from their stores.

7.1 UI Testing

User Interface (UI) testing is an important part of ensuring an e-commerce platform's optimal functioning and aesthetic appeal. The goal here was not just to create a visually appealing interface, but to create one that resonates with consumers while still offering basic functionality. With this goal in mind, Nielsen's Heuristic Evaluation emerged as a critical testing approach, providing a structured method for criticizing and improving the user experience.

7.1.1 Nielsen's Heuristic

Nielsen's Heuristic, illustrated in Figure 9 involved examining the Heuristic Evaluation Workbook [1]. It covers a range of UI elements based on established usability principles.

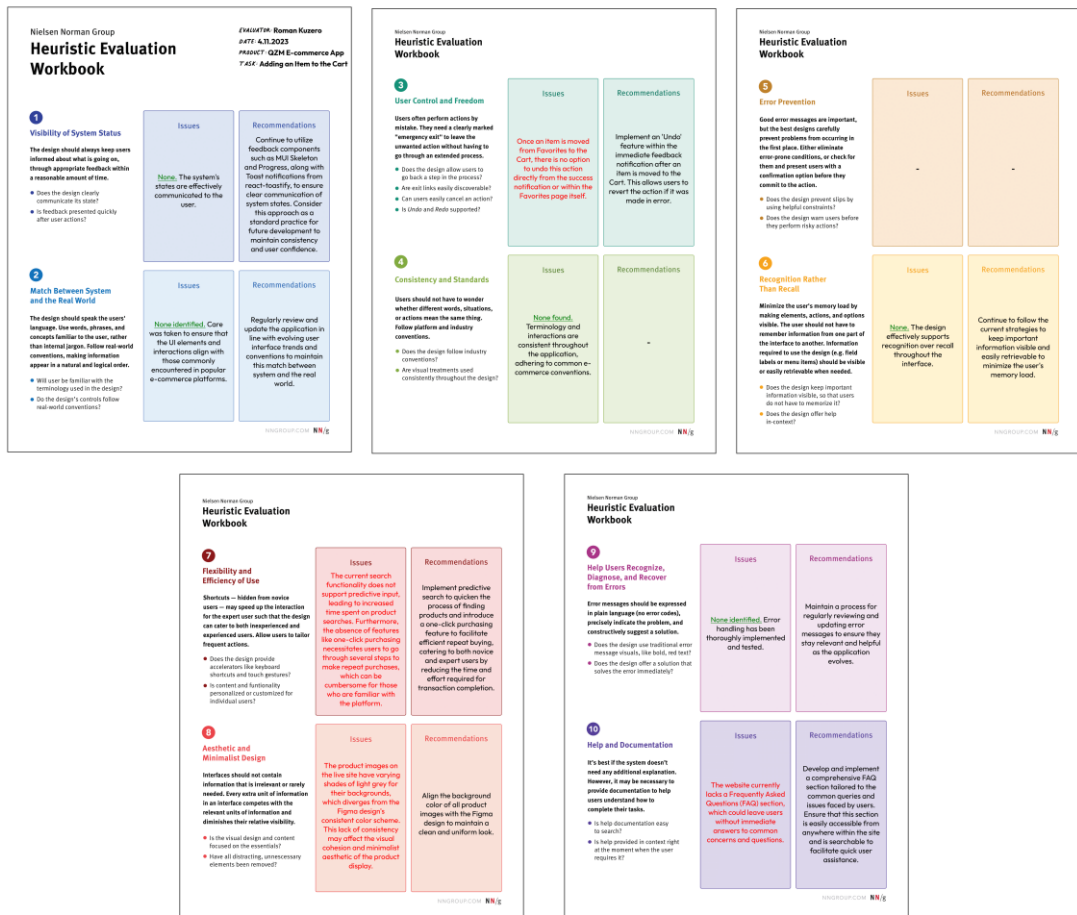


Figure 9. Heuristic evaluation workbook overview.

Figure 10 gives a glimpse into a page of the workbook illustrating the iterative process of identifying issues and implementing improvements. Through this approach every aspect of the interface was carefully analysed. This included error message clarity and the necessity and accessibility of help documentation (?) and ensuring that the interface is not visually appealing but intuitive and reliable for end users.

Heuristic Evaluation Workbook

9

Help Users Recognize, Diagnose, and Recover from Errors

Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.

- Does the design use traditional error message visuals, like bold, red text?
- Does the design offer a solution that solves the error immediately?

Issues

None identified. Error handling has been thoroughly implemented and tested.

Recommendations

Maintain a process for regularly reviewing and updating error messages to ensure they stay relevant and helpful as the application evolves.

10

Help and Documentation

It's best if the system doesn't need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.

- Is help documentation easy to search?
- Is help provided in context right at the moment when the user requires it?

Issues

The website currently lacks a Frequently Asked Questions (FAQ) section, which could leave users without immediate answers to common concerns and questions.

Recommendations

Develop and implement a comprehensive FAQ section tailored to the common queries and issues faced by users. Ensure that this section is easily accessible from anywhere within the site and is searchable to facilitate quick user assistance.

Figure 10. Detailed view of heuristic evaluation workbook pages.

7.2 Unit Testing

Unit testing is an essential component of the software development process, and while various definitions exist, the core attributes of a unit test remain consistent across them. A unit test is an automated procedure which includes the following:

- Validates a specific segment of code, commonly referred to as a 'unit'.
- Executes rapidly.
- Operates in isolation, ensuring external factors do not influence the outcome.

The perception of what constitutes a "rapid" unit test can vary among developers. Yet, this subjectivity often becomes secondary in importance. The true measure lies in the practicality: if the execution time of a test suite aligns with a developer's expectations and does not hinder the development process, then the tests can be considered efficiently rapid [32].

The importance of comprehensive unit tests is obvious in project development. They influence how we distribute resources effectively. Figure 11 illustrates the difference in growth dynamics between projects with good, bad, or no tests. The chart illustrates how the success path of a project can hinge on the quality and existence of unit tests. It highlights the need to prioritize building robust tests from the beginning of a project [33].

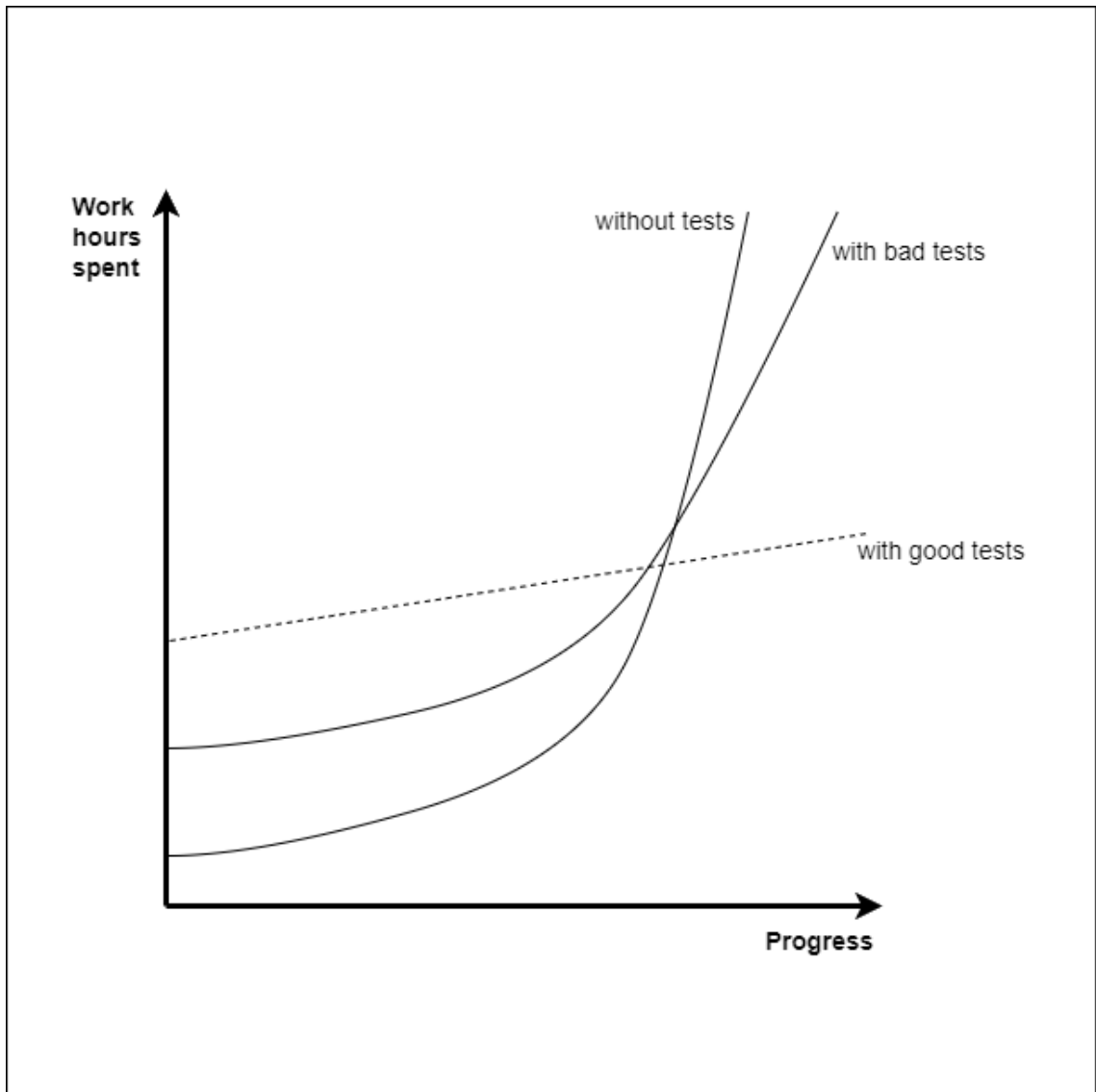


Figure 11. Impact of test quality on project progress and work hours spent [33].

7.2.1 Unit Testing in Redux Toolkit

As previously stated, the Redux Toolkit is effective at managing the state of the front-end application. Unit testing is crucial to guarantee its reliability. The purpose of these tests is to validate the logic of state management.

Listing 3 illustrates this clearly.

```
import createTestStore from "../utils/testStore"
import { getProductById } from "../../../redux/slices/productSlice";

let store = createTestStore()

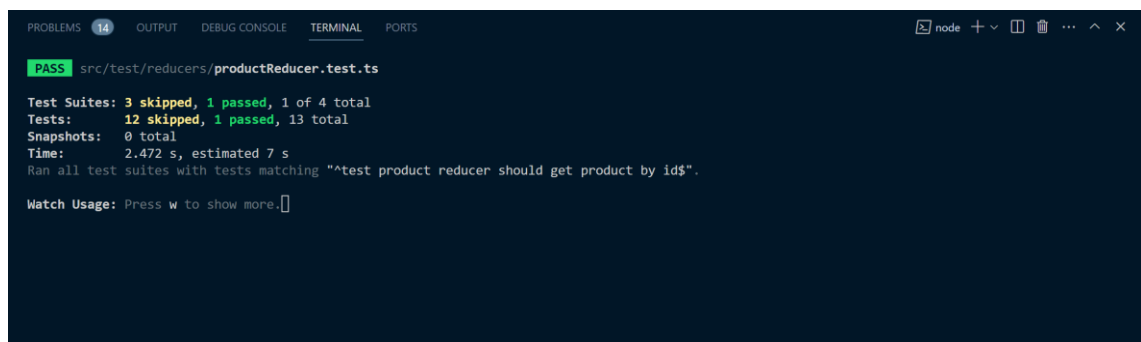
beforeEach(() => {
  store = createTestStore()
}) describe('test product reducer', () => {
  test('should get product by id', async () => {
    await store.dispatch(getProductById('<PRODUCT_ID_PLACEHOLDER>'))
    expect(store.getState().productReducer.currentProduct).toBeDefined()
  })

  // ... other tests
})
```

Listing 3. Product reducer test example.

Each individual test starts by setting up a store using the custom utility function called `createTestStore()`. After that we dispatch the `getProductById` action. Ensuring that the Redux store can appropriately, retrieving a product using its ID is the primary objective here.

Beyond the specific action and validation, it is worth noting that this test leverages a custom utility function, `createTestStore()`, to initiate a fresh store for every individual test. Once the store is set, the `getProductById` action is dispatched. The test then verifies if the product retrieval was successful.



```
PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL PORTS node + - [] ... ^ x
PASS src/test/reducers/productReducer.test.ts
Test Suites: 3 skipped, 1 passed, 1 of 4 total
Tests:      12 skipped, 1 passed, 13 total
Snapshots: 0 total
Time:       2.472 s, estimated 7 s
Ran all test suites with tests matching "^test product reducer should get product by id$".
Watch Usage: Press w to show more.
```

Figure 12. Product reducer test result.

The test results confirm that the reducer is capable of retrieving the product information from the database.

```
export const getProductByID = createAsyncThunk(
  'getProductByID',
  async (id: string | undefined) => {
    try {
      const response = await axios.get(
        `https://qzero-market-backend.herokuapp.com/api/products/${id}`
      );
      return response.data ? response.data : undefined;
    } catch (err) {
      console.log(err);
    }
  }
);
```

Listing 4. Product reducer example

This reducer takes advantage of Redux Toolkits createAsyncThunk for handling API calls. Its primary function is to fetch a product based on its ID while also keeping track of any encountered errors. By comparing this with the test, it is possible to say that the application's state management is robust.

7.2.2 Back-end unit testing

Similarly to how it was crucial to perform unit testing for front-end features, the back-end services of the application were also subjected to a thorough testing process. The back-end acts as the foundation of the application handling tasks such as data processing, storage and essential logic. To guarantee its reliability and strength I made sure to conduct systematic unit testing for this part.

It is important to note that although the methodologies used for unit tests are similar to those of the front-end, they play a role in maintaining the integrity of the back-end components. The core principle remains unchanged; it is important to test elements separately to verify their functioning.

8 Discussion and Conclusion

8.1 Discussion

The study of technology use in e-commerce development has shown how important the software stacks that prioritize scalability, flexibility, and security are. Because of its structured and adaptable nature, the MERN stack, coupled with TypeScript and the Redux Toolkit, has been recognized as an effective combination for e-commerce applications.

When it comes to e-commerce, there are challenges to consider, such as managing traffic and optimizing user experience. These difficulties have a significant impact on the selection of development tools. The chosen tools must not only meet requirements but also be adaptable to future market and technological trends.

Choosing an optimal web development stack is crucial because it directly impacts an application's ability to effectively scale data and provide a platform for transactions. This decision-making process is driven by project needs well as considerations for potential scalability in the future.

The emphasis on transactions and the need for security measures distinguishes web development for e-commerce. As a result, technologies with a track record of dependability are preferred.

Developers can achieve all success requirements by using the MERN stack alongside TypeScript and Redux Toolkit in e-commerce development projects. The Redux Toolkit simplifies state management, which is critical for optimizing user interfaces. TypeScript's powerful type system improves dependability and maintainability.

The success of the developed e-commerce application also hinges on a systematic approach to testing. Test-Driven Development, employing Jest for unit testing, serves as a foundation to ensure correct functionality. To ascertain the

stability and reliability of the back-end and API, comprehensive Postman tests evaluate the application's server-side capabilities across various scenarios.

Heuristic Evaluation, guided by the Heuristic Evaluation Workbook, plays a crucial role in meticulously examining the user interface to align it with established usability standards. The incorporation of a diverse range of testing methods, including unit front-end and back-end tests, and UI tests, strengthens the application's dependability and underscores its preparedness for real-world demands in the e-commerce industry.

8.2 Conclusion

This study presents a tailored e-commerce application crafted to meet specific market needs while also providing a blueprint for future development. The technological choices made—anchored in the MERN stack, TypeScript, and Redux Toolkit—have established a foundation for an application that not only meets current e-commerce demands but is also poised for future enhancements.

While the current state of the project reflects a significant achievement, certain areas await further development. Finalizing UI adjustments identified and incorporating comprehensive payment solutions like Stripe are among the next steps. These improvements are critical to the evolution of the application, ensuring that it remains responsive to user needs and technological advancements in the fast-paced e-commerce landscape.

References

- 1 J. Nielsen, "10 Usability Heuristics for User Interface Design," Nielsen Norman Group. [Online]. Available: <https://www.nngroup.com/articles/ten-usability-heuristics/>. Accessed: October 21, 2023.
- 2 S. K. Mourya and S. Gupta, "Introduction," in *E-Commerce*, 1st ed. Oxford, United Kingdom: ASI, 2014, pp. 22-39. [Online]. Available: <https://ebookcentral.proquest.com/lib/metropolia-ebooks/reader.action?docID=5248356&ppg=22>. Accessed: September 20, 2023.
- 3 C. Ferrera and E. Kessedjian, "Evolution of E-commerce and Global Marketing," in *International Journal of Technology for Business (IJTB)*, Mar. 2019. [Online]. Available: <https://zenodo.org/record/2591544>. Accessed: September 20, 2023.
- 4 K. T. Smith, "Worldwide Growth of E-commerce," in *E-Business*, Mar. 2009. [Online]. Available: https://www.researchgate.net/publication/285773976_Worldwide_Growth_of_E-Commerce. Accessed: September 21, 2023.
- 5 *Individuals using the Internet*, International Telecommunication Union (ITU), 2023. [Online]. Available: <https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx>. Accessed: September 21, 2023.
- 6 V. Jain, B. Malviya, and S. Arya, "An Overview of Electronic Commerce (e-Commerce)," in *Journal of Contemporary Issues in Business and Government (CIBGP)*, 2021. [Online]. Available: <https://doi.org/10.47750/cibg.2021.27.03.090>. Accessed: September 25, 2023.
- 7 S. K. Mourya and S. Gupta, "E-commerce: Architecture to Models," in *E-Commerce*, 1st ed. Oxford, United Kingdom: ASI, 2014, pp. 40-55. [Online]. Available: <https://ebookcentral.proquest.com/lib/metropolia-ebooks/reader.action?docID=5248356&ppg=22>. Accessed: September 25, 2023.
- 8 V. Subramanian, "Introduction," in *Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node*, 2nd ed. New York City, NY, USA: Apress, May 2019. [Online]. Available: https://doi.org/10.1007/978-1-4842-4391-6_1. Accessed: October 4, 2023.
- 9 MEAN.JS official website. [Online]. Available: <https://meanjs.org/>. Accessed: October 6, 2023.
- 10 P. D. Dutonde, S. S. Mamidwar, S. Korvate, S. Bafna, and D. D. Shirbhate, "Website Development Technologies: A Review," in *International Journal for Research in Applied Science & Engineering*

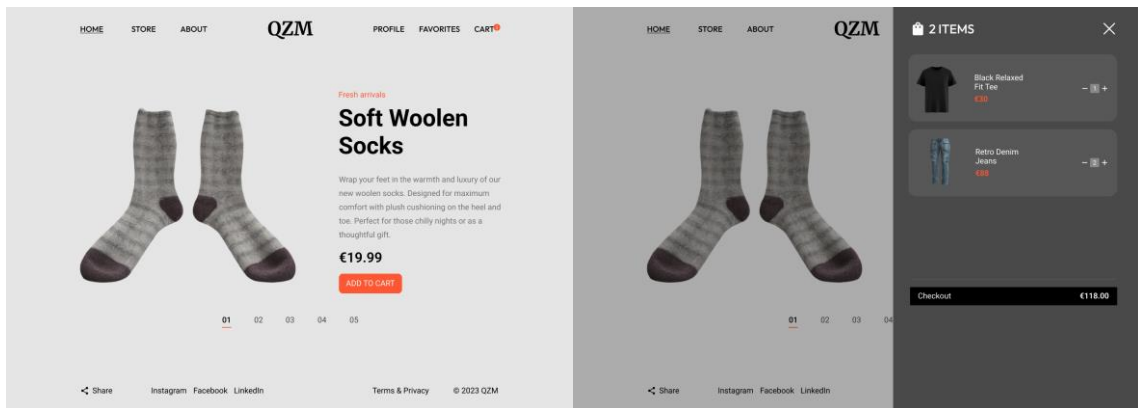
- Technology (IJRASET)*, Jan. 2022. [Online]. Available: <https://doi.org/10.22214/ijraset.2022.39839>. Accessed: October 9, 2023.
- 11 "What Is Angular?" Angular official website. [Online]. Available: <https://angular.io/guide/what-is-angular>. Accessed: October 9, 2023.
 - 12 D. Roth, R. Anderson, and S. Luttin, "Overview of ASP.NET Core," in *ASP.NET Core documentation*, Microsoft, Oct. 3, 2023. [Online]. Available: <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0>. Accessed: October 10, 2023.
 - 13 A. Lock, "Getting Started with ASP.NET Core," in *ASP.NET Core in Action, 3rd ed.* New York, NY, USA: Simon and Schuster, 2023. [Online]. Available: <https://books.google.fi/books?id=JuXPEAAAQBAJ&lpg=PR28&ots=BVkJ9uLd38&dq=ASP.Net%20Core&lr&pg=PR28#v=onepage&q=ASP.Net%20Core&f=false>. Accessed: September 20, 2023.
 - 14 M. Mehra, M. Kumar, A. Maurya, C. Sharma, and Shanu, "MERN Stack Web Development," in *Annals of RSCB*, vol. 25, no. 6, pp. 11756–11761, June 2021.
 - 15 A. Fedosejev, "Installing Powerful Tools for Your Project," in *React.js Essentials*. Birmingham, UK: Packt Publishing Ltd, 2015. [Online]. Available: <https://books.google.fi/books?id=Rhl1CgAAQBAJ&lpg=PP1&ots=JkAwmByQPD&dq=react%20js&lr&pg=PP1#v=onepage&q=react%20js&f=false>. Accessed: October 10, 2023.
 - 16 S. Aggarwal, J. Verma, "Comparative Analysis of MEAN Stack and MERN stack," *International Journal of Recent Research Aspects (IJRRA)*, Mar. 2018. [Online]. Available: <http://www.ijrra.net/Vol5issue1/IJRRA-05-01-26.pdf>. Accessed: October 11, 2023.
 - 17 D. Muyltermans, "How Does the Virtual DOM Compare to Other DOM Update Mechanisms in JavaScript Frameworks?" M.Sc. thesis, Interactive Digital Media, Univ. of Dublin, Dublin, Ireland, 2019. [Online]. Available: <http://www.daisyms.com/THESIS.pdf>. Accessed date: October 11, 2023.
 - 18 "Virtual DOM and Internals." React documentation. <https://legacy.reactjs.org/docs/faq-internals.html#gatsby-focus-wrapper>. Accessed: October 11, 2023.
 - 19 E. Elrom, "Learn React Basic Concepts," in *React and Libraries*. New York City, NY, USA: Apress, Mar. 2021. [Online]. Available: https://doi.org/10.1007/978-1-4842-6696-0_1. Accessed: October 11, 2023.
 - 20 "Stack Overflow Developer Survey 2023." Stack Overflow. Available: <https://survey.stackoverflow.co/2023/>. Accessed: October 11, 2023.

- 21 B. Cherny, "TypeScript: A 10_000 Foot View," in *Programming TypeScript*. Sebastopol, CA, USA: O'Reilly Media, May 2019. [Online]. Available: <https://learning.oreilly.com/library/view/programming-typescript/9781492037644/ch01.html>. Accessed: October 11, 2023.
- 22 D. B. Duldulao and R. J. L. Cabagnet, "Managing State Using Redux with Redux Toolkit," in *Practical Enterprise React: Become an Effective React Developer in Your Team*. New York City, NY, USA: Apress, Aug. 2021. [Online]. Available: https://doi.org/10.1007/978-1-4842-6975-6_9. Accessed: October 11, 2023.
- 23 "Redux Toolkit - Quick Start," Redux Toolkit documentation. [Online]. Available: <https://redux-toolkit.js.org/introduction/getting-started>. Accessed: October 11, 2023.
- 24 R. Fauzan, D. Siahaan, S. Rochimah, and E. Triandini, "A Different Approach on Automated Use Case Diagram Semantic Assessment," in *International Journal of Intelligent Engineering and Systems (IJIES)*, Dec. 2020. [Online]. Available: <https://doi.org/10.22266/ijies2021.0228.46>. Accessed: October 15, 2023.
- 25 C. Abras, D. Maloney-Krichmar, and J. Preece, "User-Centered Design," in *Encyclopedia of Human-Computer Interaction*. Thousand Oaks, CA, USA: Sage Publications, 2004. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/6190316/10.1.1.94.381-libre.pdf?1390844883=&response-content-disposition=inline%3B+filename%3DUser_centered_design.pdf&Expires=1698418838&Signature=TaLuGJey3KPJXR9e4JAwXT8K0Zy962lojlmNu1LzYb5C~orum9WFDEHtYJ48kHCaTwSiJeUODB7rhkQzM3OHISBOL-BRWHBj8nFwvq0aRITLVlg1ohZHN~v~YT7SumLexG2mdMprHTa693~t03nSKvcXzBNPw5z-0qJFHliKy-7CscWKOIRRJahQp0ArV3JtwoVd1jmPlsWDIHQkxaQ9Pytv6c4g-YaEwav1NU2iKemkloalSZfmxA088-ZRvQ54Er0CM~2SUG93eZUB77ogARxdXnQeEbkGOe8ykO. Accessed: October 17, 2023.
- 26 Figma official design website. [Online]. Available: <https://www.figma.com/design/>. Accessed: October 17, 2023.
- 27 JWT official website. [Online]. Available: <https://jwt.io/>. Accessed: October 20, 2023.
- 28 Stripe official website. [Online]. Available: <https://stripe.com/en-fi>. Accessed: October 20, 2023.
- 29 Material-UI official documentation. [Online]. Available: <https://mui.com/material-ui/getting-started/>. Accessed: October 20, 2023.
- 30 S. Kundu, "Web Testing: Tool, Challenges and Methods," in *International Journal of Computer Science Issues (IJCSI)*, Mar. 2012. [Online]. Available: <https://www.IJCSI.org>. Accessed: October 22, 2023.

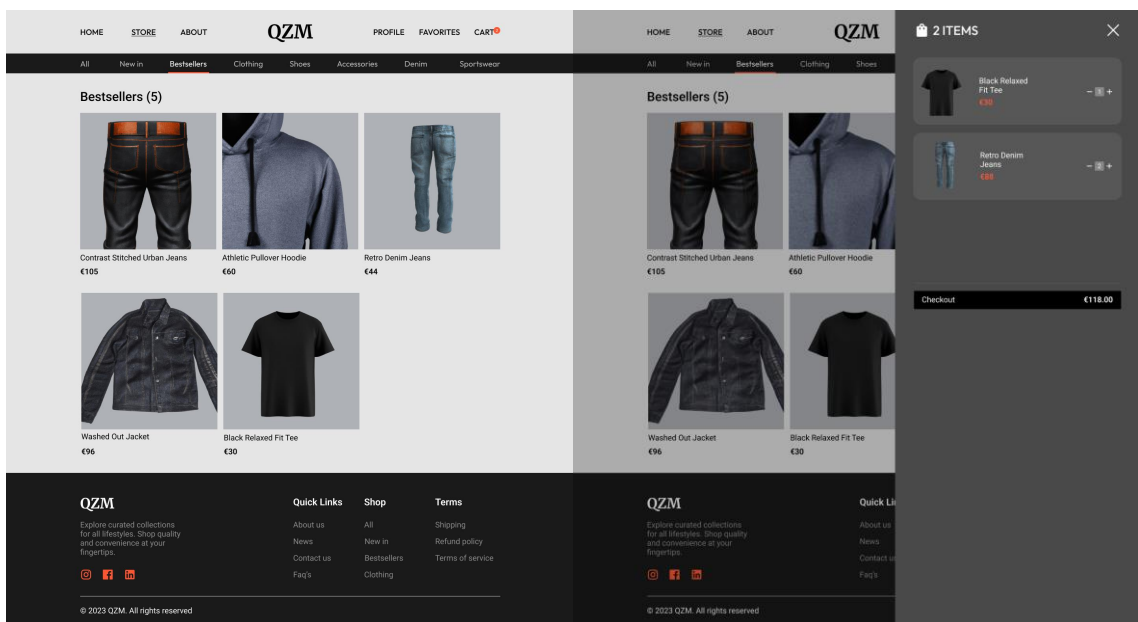
- 31 P. Gerrard, "Risk-Based E-Business Testing Part 1: Risks and Test Strategy," Aug. 15, 2002. [Online]. Available: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c97aa1f81c9f7c79de657e59991fbf76ddb0fa08>. Accessed: October 22, 2023.
- 32 V. Khorikov, "What is a unit test?" in *Unit Testing Principles, Practices, and Patterns*. New York City, NY, USA: Simon and Schuster, Jan. 6, 2020. [Online]. Available: <https://books.google.fi/books?id=rDszEAAAQBAJ&lpg=PT15&ots=MI6tNlpKI2&dq=Unit%20testing&lr&pg=PT48#v=onepage&q=Unit%20testing&f=false>. Accessed: October 23, 2023.
- 33 V. Khorikov, "The goal of unit testing?" in *Unit Testing Principles, Practices, and Patterns*. New York City, NY, USA: Simon and Schuster, Jan. 6, 2020. [Online]. Available: <https://books.google.fi/books?id=rDszEAAAQBAJ&lpg=PT15&ots=MI6tNlpKI2&dq=Unit%20testing&lr&pg=PT48#v=onepage&q=Unit%20testing&f=false>. Accessed: October 23, 2023.

Detailed UI screenshots of "QZM" e-commerce application

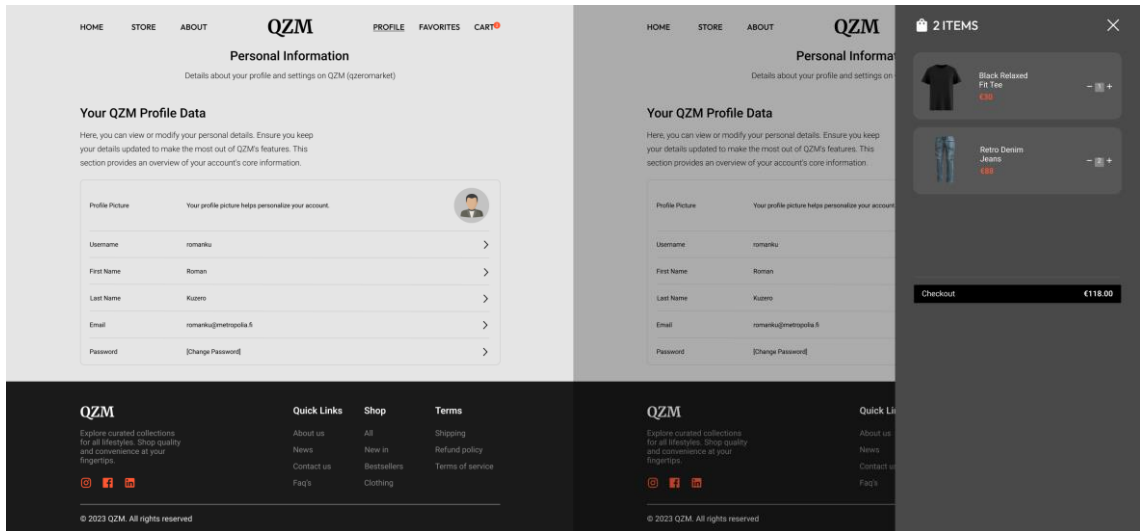
Home page UI design



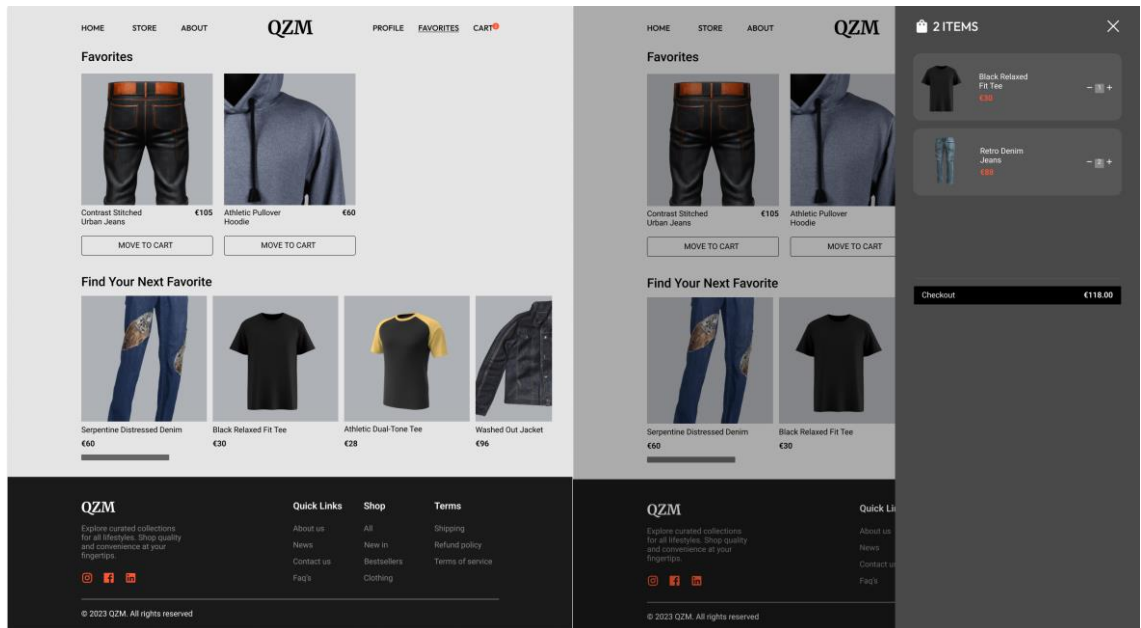
Store page UI design



Profile page UI design



Favourites page UI design



Heuristic Evaluation workbook pages

Nielsen Norman Group

Heuristic Evaluation Workbook

EVALUATOR: Roman Kuzero
DATE: 4.11.2023
PRODUCT: QZM E-commerce App
TASK: Adding an Item to the Cart

1
Visibility of System Status

The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time.

- Does the design clearly communicate its state?
- Is feedback presented quickly after user actions?

Issues	Recommendations
None. The system's states are effectively communicated to the user.	Continue to utilize feedback components such as MUJ Skeleton and Progress, along with Toast notifications from react-toastify, to ensure clear communication of system states. Consider this approach as a standard practice for future development to maintain consistency and user confidence.

2
Match Between System and the Real World

The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order.

- Will user be familiar with the terminology used in the design?
- Do the design's controls follow real-world conventions?

Issues	Recommendations
None identified. Care was taken to ensure that the UI elements and interactions align with those commonly encountered in popular e-commerce platforms.	Regularly review and update the application in line with evolving user interface trends and conventions to maintain this match between system and the real world.

NNGROUP.COM NN/g

Nielsen Norman Group

Heuristic Evaluation Workbook

3
User Control and Freedom

Users often perform actions by mistake. They need a clearly marked "emergency exit" to leave the unwanted action without having to go through an extended process.

- Does the design allow users to go back a step in the process?
- Are exit links easily discoverable?
- Can users easily cancel an action?
- Is Undo and Redo supported?

Issues	Recommendations
Once an item is moved from Favorites to the Cart, there is no option to undo this action directly from the success notification or within the Favorites page itself.	Implement an 'Undo' feature within the immediate feedback notification after an item is moved to the Cart. This allows users to revert the action if it was made in error.

4
Consistency and Standards

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions.

- Does the design follow industry conventions?
- Are visual treatments used consistently throughout the design?

Issues	Recommendations
None found. Terminology and interactions are consistent throughout the application, adhering to common e-commerce conventions.	-

NNGROUP.COM NN/g

Nielsen Norman Group

Heuristic Evaluation Workbook

5
Error Prevention

Good error messages are important, but the best designs carefully prevent problems from occurring in the first place. Either eliminate error-prone conditions, or check for them and present users with a confirmation option before they commit to the action.

- Does the design prevent slips by using helpful constraints?
- Does the design warn users before they perform risky actions?

Issues	Recommendations
-	-

6
Recognition Rather Than Recall

Minimize the user's memory load by making elements, actions, and options visible. The user should not have to remember information from one part of the interface to another. Information required to use the design (e.g. field labels or menu items) should be visible or easily retrievable when needed.

- Does the design keep important information visible, so that users do not have to memorize it?
- Does the design offer help in-context?

Issues	Recommendations
None. The design effectively supports recognition over recall throughout the interface.	Continue to follow the current strategies to keep important information visible and easily retrievable to minimize the user's memory load.

NNGROUP.COM NN/g

Nielsen Norman Group

Heuristic Evaluation Workbook

7
Flexibility and Efficiency of Use

Shortcuts – hidden from novice users – may speed up the interaction for the expert user such that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

- Does the design provide accelerators like keyboard shortcuts and touch gestures?
- Is content and functionality personalized or customized for individual users?

Issues	Recommendations
The current search functionality does not support predictive input, leading to increased time spent on product searches. Furthermore, the absence of features like one-click purchasing necessitates users to go through several steps to make repeat purchases, which can be cumbersome for those who are familiar with the platform.	Implement predictive search to quicken the process of finding products and introduce a one-click purchasing feature to facilitate efficient repeat buying, catering to both novice and expert users by reducing the time and effort required for transaction completion.

8
Aesthetic and Minimalist Design

Interfaces should not contain information that is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.

- Is the visual design and content focused on the essentials?
- Have all distracting, unnecessary elements been removed?

Issues	Recommendations
The product images on the live site have varying shades of light grey for their backgrounds, which diverges from the Figma design's consistent color scheme. This lack of consistency may affect the visual cohesion and minimalist aesthetic of the product display.	Align the background color of all product images with the Figma design to maintain a clean and uniform look.

NNGROUP.COM NN/g

Nielsen Norman Group

Heuristic Evaluation Workbook

9

Help Users Recognize, Diagnose, and Recover from Errors

Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.

- Does the design use traditional error message visuals, like bold, red text?
- Does the design offer a solution that solves the error immediately?

Issues

None identified. Error handling has been thoroughly implemented and tested.

Recommendations

Maintain a process for regularly reviewing and updating error messages to ensure they stay relevant and helpful as the application evolves.

10

Help and Documentation

It's best if the system doesn't need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.

- Is help documentation easy to search?
- Is help provided in context right at the moment when the user requires it?

Issues

The website currently lacks a Frequently Asked Questions (FAQ) section, which could leave users without immediate answers to common concerns and questions.

Recommendations

Develop and implement a comprehensive FAQ section tailored to the common queries and issues faced by users. Ensure that this section is easily accessible from anywhere within the site and is searchable to facilitate quick user assistance.