Developing Commercial Application using React,Redux and Custom API

T. Raghavendra Gupta[1], S. Rakesh Reddy[2], S. Shiva shanker prasad[3], S. Vivek Vardan[4], P. Abhiram Sharma[5], Mr. T. Raghavendra Gupta[6]
[1]*Associate Professor, Department of CSE, HITAM, Hyderabad, India*
[2,3,4,5] *Student of Computer Science and Engineering, HITAM Hyderabad, India*
[6]*Guide, Department of CSE, HITAM, Hyderabad, India*

***Abstract*— In this project, we will develop a fully functional website using React, Redux, Node.js, and Express.js, with a focus on performance and efficient state management. React component-based architecture will be leveraged to ensure minimal and optimized re-renders, enhancing the website's performance and reducing server load. Redux will be integrated to provide a centralized store accessible by all React components, enabling consistent and efficient DOM manipulation across the application. We will create a custom API using Node.js, running locally to serve data requested by the React components, while Express.js will handle backend routing and logic, ensuring smooth communication between the frontend and backend. The project aims to deliver a high-performance, responsive website with seamless frontend-backend integration and efficient data management.**

***Keywords*— React.js, Redux, Node.js, Express.js, Efficient State Management, Component-Based Architecture, API Development, Backend Routing, Responsive Web Design, Custom API.**

I. INTRODUCTION

The rapid growth of e-commerce has transformed how businesses operate and how consumers engage with products and services. With the increasing demand for online shopping, the development of efficient, scalable, and user-friendly e-commerce platforms has become a critical area of research and innovation. This paper presents the design and development of 'e shop,' a fully functional e-commerce website that integrates modern web technologies to deliver an enhanced shopping experience.

The primary goal of the 'e shop' platform is to provide users with a seamless and intuitive interface, ensuring efficiency in browsing, selecting, and purchasing products. Built using React, the website employs a component-based architecture to ensure scalability and maintainability. React enables the creation of dynamic and responsive user interfaces that adapt to various devices, thereby improving accessibility and user engagement. To manage complex application states effectively, the project incorporates Redux, a robust state management library that ensures data consistency and smooth interactions across the platform. One of the unique aspects of the 'e shop' is its focus on performance and usability. The design emphasizes faster loading times, simplified navigation, and a clutter-free interface to cater to the needs of modern users. The website's modular architecture allows for easy integration of additional features, such as user authentication, payment gateways, and analytics, which can be incorporated in future iterations to enhance functionality further.

This paper also explores the challenges encountered during the development process, including integrating front-end and back-end components, managing application states efficiently, and ensuring responsiveness across devices. It discusses the strategies adopted to overcome these challenges and the role of modern development tools in streamlining the workflow. The research aims to highlight the potential of contemporary web technologies like React and Redux in building high-performance, responsive, and scalable e-commerce platforms. By combining theoretical insights with practical implementation, this paper contributes to the growing body of knowledge on e-commerce development, offering valuable guidance for developers and researchers seeking to create innovative online shopping solutions.

In conclusion, 'e shop' represents a step forward in delivering an efficient and user-centric e-commerce experience. The project demonstrates the power of modern web frameworks and state management tools in addressing the demands of a rapidly evolving

digital marketplace, paving the way for further advancements in the field of e-commerce technology.

II. RELATED WORK

Now in this related work part, we will discuss some work that has been done in this field.

A. Managing State in React Applications with Redux : Timo McFarlane

The research paper titled "Managing State in React Applications with Redux" by Timo McFarlane, submitted as a Bachelor's thesis in November 2019 at Tampere University of Applied Sciences, delves into the intricacies of state management in React applications. The primary objective was to design and implement a new state management architecture using Redux for an existing project. The study also explored other methods for handling state and component lifecycle in React applications, particularly focusing on React Hooks, a feature introduced in February 2019.

The results of the study were promising, with a new state management architecture successfully designed and implemented. The paper provides valuable insights into various state management methods and their effectiveness in React applications. Commissioned by Visma Consulting Oy, a software company specializing in digital services and custom software solutions, this research contributes significantly to the understanding and practical application of state management in modern web development.[1]

B. Comparison of Redux and React Hooks Methods in Terms of Performance : Daria Pronina and Iryna Kyrychenko

The research paper titled "Comparison of Redux and React Hooks Methods in Terms of Performance" by Daria Pronina and Iryna Kyrychenko from Kharkiv National University of Radio electronics, Ukraine, focuses on comparing two state management methods in React applications: Redux and React Hooks. The primary objective of the study is to analyse the performance of these two methods to provide insights for choosing the appropriate state management approach in React applications. The study involved creating a simple front-end application with functional requirements such as rendering, editing, and deleting posts, and managing user data. This application was implemented using both Redux and React Hooks to compare their performance.

The findings indicate that Redux, while popular for its scalability and predictability, introduces significant boilerplate code and performs worse in terms of memory usage and state update time compared to React Hooks. React Hooks, on the other hand, offer a more optimized performance with less code and better memory management. The study concludes that React Hooks are more suitable for high-performance applications, while Redux may not be preferred for applications with stringent performance requirements. This paper provides valuable recommendations for developers on choosing the right state management approach based on performance consideration.[2]

C. Front-End Development in React: Songtao Chen

The research paper provides an in-depth exploration of React.js, a popular JavaScript library for building user interfaces. It begins by discussing the evolution of web development from traditional server-side rendering to modern client-side rendering in Single Page Applications (SPAs). React's client-side rendering approach enables dynamic and interactive user experiences by leveraging the Virtual DOM to reduce the need for full-page reloads. This architecture ensures faster response times and seamless navigation, mimicking the behaviour of native applications. React's core features are highlighted, including its declarative syntax, which allows developers to focus on what they want to render rather than how to manipulate the DOM. The framework's component-based architecture promotes reusability, modularity, and easier maintenance. React's high performance is attributed to its Virtual DOM, which efficiently updates only the necessary parts of the UI instead of the entire DOM. These features collectively make React a preferred choice for building scalable and dynamic applications.

The paper also discusses the broader React ecosystem. For state management, tools like Flux and Redux are introduced, emphasizing their ability to handle complex data flows predictably in large-scale applications. The use of CSS Modules is explored as a solution to manage styles at scale, enabling encapsulated and reusable CSS. Additionally, the paper delves into the importance of static typing with tools like Flow and TypeScript, which enhance code

maintainability and readability. Other aspects, such as Webpack for module bundling and tools like React Developer Tools for debugging, are presented as critical components of the React development workflow.

The paper briefly compares React to other frameworks, such as Angular.js, highlighting React's simplicity, flexibility, and strong community support. However, challenges like longer initial page load times due to preloading scripts and styles, as well as SEO limitations caused by JavaScript-dependent content, are acknowledged. Despite these challenges, the paper underscores React's ability to simplify complex web development tasks and improve user experiment.[3]

D React and Redux : Matthias Kevin Caspers

Many popular JavaScript frameworks for developing single page applications (SPAs) and rich internet applications (RIA) use mutable state, templating, a two-way data-binding system and impose the imperative programming paradigm on the developer. This paper will show some problems that the aforementioned design choices have and present React and Redux as an alternative for developing SPAs and RIAs. React is a user interface (UI) library, written in JavaScript and developed by Facebook. It's often described as being the V in MVC. Redux is an easy-to-use predictable state container, created by Dan Abramov in JavaScript. React and Redux are both independent from one another. They do however work very well together in practice. Since React is a library for building UIs and Redux is a state container, they do not come with many of the features that fully fledged SPA/RIA frameworks like Angular JS or Ember JS provide. There exist however dozens of libraries which can provide these features to them. This paper will introduce the reader to React and Redux, as well as the problems which they solve. The first part of this paper will focus solely on React. Redux will be covered in the second part. The third part will cover how React and Redux can be used together. It will be shown, that React and Redux present a powerful way for developing SPAs/RIAs, which in many regards is superior to the way SPAs and RIAs were written prior to the emergence of React.[4][5]

III. IMPLEMENTATION OF CORE PLATFORM COMPONENTS

1. React.js

React.js is the primary frontend framework used to build the platform's user interface. Its component-based architecture enables the development of reusable and modular UI components, such as navigation bars, product cards, and shopping carts. Each component is designed independently, making the platform highly maintainable and scalable. React's virtual DOM ensures efficient updates, rendering only the components that change instead of refreshing the entire page. This enhances the user experience by delivering faster and more responsive interactions. Features like React hooks (useState, useEffect) allow for efficient state handling and lifecycle management within individual components.

2. Redux

Redux is used for efficient state management, particularly for handling global application states like user sessions, product data, and cart items. By maintaining a centralized store, Redux ensures data consistency across components and eliminates redundancy. For example, when a product is added to the cart, the Redux action updates the global state, automatically re-rendering the shopping cart and navigation bar without manual intervention. Middleware like Redux Thunk enables asynchronous data fetching, such as retrieving product lists from an API, further streamlining state updates.

3. Node.js

Node.js serves as the backend runtime environment, enabling efficient handling of server-side operations. It powers the custom API development and processes user requests, such as fetching product data or managing cart operations. Node.js's non-blocking I/O model ensures high performance and scalability, allowing the platform to handle multiple requests simultaneously. Its event-driven architecture makes it ideal for building responsive and real-time applications like 'e shop.'

4. Express.js

Express.js is the backend framework used for API development and backend routing. It facilitates the creation of RESTful APIs that handle core functionalities, including retrieving product details, adding items to the cart, and processing orders. Express routes are designed to communicate seamlessly with the frontend, ensuring a smooth flow

of data between the user interface and the server. For example, a GET request to /products retrieves all product details from the database, which are then displayed dynamically on the React frontend.

5. Efficient State Management

Efficient state management is critical to ensuring a smooth user experience. Redux handles global states like product listings, cart items, and user sessions, while React manages local component-level states. This combination ensures optimal performance by keeping frequently changing states localized and static data in the global store. The platform uses structured reducers and actions in Redux, minimizing boilerplate code and improving maintainability.

6. Component-Based Architecture

The entire platform is built using a component-based architecture, allowing the reuse of components across different sections. For example, the Product Card component is used both on the product listing page and the recommendations section of the product details page. This modular approach improves scalability, as developers can add new features or update existing ones without affecting other parts of the application. Components are styled independently, ensuring consistent behavior across various devices.

7. API Development

The platform's custom API is designed to handle requests for data such as product details, user authentication, and cart management. Built using Node.js and Express.js, the API ensures seamless communication between the frontend and backend. It adheres to RESTful principles, enabling CRUD (Create, Read, Update, Delete) operations. For instance, when a user adds a product to the cart, a POST request updates the cart data in the backend, which is then fetched and displayed on the frontend using Redux.

8. Responsive Web Design

The platform incorporates responsive web design principles to ensure usability across devices of varying screen sizes. CSS frameworks like Bootstrap and custom media queries are used to make the design adaptable. For instance, the navigation bar collapses into a hamburger menu on smaller screens, and the product grid layout adjusts dynamically to optimize content visibility on mobile devices. This ensures a consistent user experience regardless of the device used.

9. Custom API

The custom API acts as the backbone of the platform, enabling real-time communication between the frontend and backend. It is built to handle key functionalities such as product retrieval, user authentication, and cart updates. The API is designed for future scalability, allowing for additional endpoints like order history, user profiles, and analytics dashboards. Proper error handling and validation are implemented to ensure secure and reliable data transmission.

IV. RESULTS AND DISCUSSIONS

The development of the 'e shop' e-commerce platform demonstrates the successful integration of modern web technologies to create a high-performance, responsive, and user-friendly application. The platform's component-based architecture, powered by React.js, enables modularity and reusability across the application. Key features such as product listings, dynamic routing, and cart management showcase the efficiency of the design. The use of Redux for state management ensures consistency in data flow across components, significantly reducing redundancy and improving performance. Testing across multiple devices and browsers confirms that the platform maintains a seamless user experience, regardless of the screen size or operating system, fulfilling the goal of responsive web design.

The efficient handling of global state through Redux has resolved common challenges such as data inconsistency and complex state updates. The centralized store allowed for real-time updates to features like the shopping cart and user interface components without the need for redundant API calls or manual state synchronization. For instance, when a user adds or removes an item from the cart, the Redux store updates the cart state globally, ensuring that changes are immediately reflected across all relevant components. This not only enhances the platform's efficiency but also minimizes the load on the backend server, demonstrating a thoughtful approach to balancing frontend and backend responsibilities.

The backend implementation using Node.js and Express.js further validates the scalability and robustness of the platform. Custom APIs were developed to handle critical operations, including fetching product data, managing cart interactions, and processing orders. These APIs were designed to adhere to RESTful principles, ensuring clarity and ease of integration with the frontend. Testing of the APIs revealed minimal latency in data retrieval and a strong ability to handle concurrent user requests, indicating that the backend architecture is well-suited for real-world use cases. Future integration of a database for storing user and product data can further enhance the platform's functionality and enable additional features such as user authentication and order history.

Overall, the 'e shop' project highlights the effectiveness of combining React.js, Redux, Node.js, and Express.js for building a scalable e-commerce platform. The responsive design and optimized performance make it a strong foundation for future enhancements, such as payment gateway integration, advanced filtering options, and personalized user experiences. The results demonstrate that the platform successfully addresses the initial challenges of performance, state management, and scalability, providing a competitive and modern solution in the growing e-commerce landscape.

V. FUTURE WORK

The future work for the 'e shop' platform focuses on expanding its capabilities to create a more robust, secure, and personalized e-commerce experience. Key improvements include integrating payment gateways (such as PayPal or Stripe) for seamless transactions, ensuring secure and smooth checkout processes. Additionally, the implementation of user authentication will allow for personalized profiles, order tracking, and enhanced user interaction with the platform. This feature will also enable role-based access control for admins to manage products and customer data.

To enhance the shopping experience, advanced search features and product recommendations powered by machine learning algorithms will be introduced, allowing for more personalized product suggestions and efficient navigation. Analytics and reporting tools will be integrated to provide insights into user behaviour, sales trends, and platform performance, helping business owners make informed decisions.

The platform will also be scaled by integrating cloud services for better data management and improved performance as user traffic increases. These enhancements will ensure that the 'e shop' remains competitive, secure, and user-centric, providing a complete solution for both customers and business owners in the e-commerce space.

REFERENCES

[1] Microsoft Word - Mcfarlane_Timo.docx Managing State in React Appli cations with Redux by Timo McFarlane
[2] Microsoft Word - Comparison_of_Redux_and_React_Hooks_app roaches_in_terms_of_performance.docx. by Daria Pronina 1 and Iryna Kyrychenko
[3] https://www.researchgate.net/publication/374154236 by Songtao Chen1 , Upendar Rao Thaduri 2 , Venkata Koteswara Rao Ballamudi 3*
[4] Roy Thomas Fielding. REST APIs must be hypertextdriven. URL: http://roy.gbiv.com/untangled/2008/restapis-must-be-hypertext-driven
[5] https://uol.de/f/2/dept/informatik/ag/svs/download/reader/reader-seminar-ws2016.pdf : Matthias Kevin Caspers Carl von Ossietzky University of Oldenburg, Germany Department of Computer Science matthias.kevin.caspers@uni-oldenburg.de.