

**VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY**



**VŨ THANH HẢI – 523H0022
NGUYỄN QUỐC THẮNG – 523H0094
NGÔ CHÍ THUẬN – 523H0102**

MIDTERM ESSAY

WEB PROGRAMING AND APPLICATIONS

HO CHI MINH CITY, 2025

**VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY**



**VŨ THANH HẢI – 523H0022
NGUYỄN QUỐC THẮNG – 523H0094
NGÔ CHÍ THUẬN – 523H0102**

MIDTERM ESSAY

WEB PROGRAMING AND APPLICATIONS

Advised by
Mr. Mai Van Manh

HO CHI MINH CITY, 2025

ACKNOWLEDGEMENT

We would like to sincerely thanks to Ton Duc Thang University for providing an enriching academic environment that has played a pivotal role in shaping our educational journey.

We extend our appreciation to the Department of Software Engineering for their support, guidance and commitment to fostering an environment conducive to learning and growth.

We are deeply grateful to the subject Web Programing and Applications for its imparting knowledge, its guidance, and its force to a deep understanding of the subject matter, which has greatly enriched our academic experience.

Lastly, we are thankful to the assistance provided by Mr. Mai Van Manh, for his insights, and support during the preparation of this report.

Ho Chi Minh city, 10th April 2025.

Vũ Thanh Hải

Nguyễn Quốc Thắng

Ngô Chí Thuận

DECLARATION OF AUTHORSHIP

We hereby declare that this is our research project and is under the scientific guidance of Mr. Mai Van Manh. The research content and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments, and evaluation were collected by the author from different sources and clearly stated in the reference section.

In addition, the project also uses some comments, assessments as well as data of other authors, other organizations with citations and annotated sources.

If something wrong happens, **we'll take full responsibility for the content of my project.** Ton Duc Thang University is not related to the infringing rights, the copyrights that We give during the implementation process (if any).

Ho Chi Minh city, 10th April 2025.

Vũ Thanh Hải

Nguyễn Quốc Thắng

Ngô Chí Thuận

TABLE OF CONTENT

Chapter 1. Technology Overview and Review	3
1. React.js.....	3
1.1 Overview of React.js	3
1.2 Component-Based Architecture	3
1.2.1. Components in React.js:.....	3
a. Class Components:	4
b. Function Components:	5
1.2.2. Core Concepts of Components:	6
a. Props (Properties):	6
b. State:	6
c. Comparison Props vs. State:	7
d. Lifecycle Methods (Class Components), Hooks (Function Components):	8
e. Composition of Components:	9
1.2.3. Benefits of Component-Based Architecture:	9
1.3 Virtual DOM and JSX (JavaScript XML).....	9
1.3.1. Virtual DOM in React.js:	9
a. Traditional DOM vs. Virtual DOM:	10
b. How the Virtual DOM Works:	10
c. Advantages of the Virtual DOM:	11
1.3.2. JSX in React.js:.....	11
a. JSX vs. HTML:	12
b. Advantages of JSX:	12
2. UI Libraries: Material-UI and Ant Design.....	13
2.1 Material-UI (MUI)	13
2.1.1. Material Design Philosophy	14
2.1.2. Component Ecosystem.....	14
2.1.3. Customization and Theming	15
a. Global Themes:	15
b. Component Customization:	15
c. Dark Mode Support:	15
2.1.4. Responsiveness and Layouts	15
2.1.5. Conclusion	16
2.2 Ant Design (AntD)	16
2.2.1. Philosophy and Design Principles	16
2.2.2. Rich Component Library	17
2.2.3. Customization and Theming	17
2.2.4. Internationalization (i18n)	18
2.2.5. Form Management & Validation	18

2.2.6. Responsive Design & Layout System	18
2.2.7. Limitations and Considerations	19
2.2.8. Conclusion	19
3. Comparison: React vs. Other Frameworks (Angular and Vue.js)	19
3.1 React vs. Angular	20
3.1.1. Framework vs. Library	20
3.1.2. Language and Syntax.....	20
3.1.3. Data Binding.....	21
3.1.4. State Management.....	21
3.1.5. Performance Considerations	21
3.1.6. Ecosystem and Tooling.....	22
3.1.7. Community and Industry Adoption	22
3.1.8. Conclusion	23
3.2 React vs. Vue.js	23
3.2.1. Design Philosophy and Core Concepts.....	24
3.2.2. Component Syntax and Developer Experience	24
3.2.3. State Management.....	24
3.2.4. Reactivity System	25
3.2.5. Ecosystem and Tooling.....	25
3.2.6. Community and Industry Adoption	25
3.2.7. Conclusion	26
Chapter 2. Product Design and Demonstration.....	27
REFERENCES.....	39

Chapter 1. Technology Overview and Review

1. React.js

1.1 Overview of React.js

React.js, often referred to simply as React, is a powerful, open-source JavaScript library used to build user interfaces (UIs), particularly for single-page applications (SPAs). Created by Facebook in 2011 and released publicly in 2013, React has since become one of the most popular libraries for front-end development, revolutionizing the way developers approach web and mobile app design. Unlike traditional JavaScript frameworks, React focuses on creating reusable components that can efficiently render changes in the user interface, making it an essential tool for modern web development.

1.2 Component-Based Architecture

1.2.1. Components in React.js:

One of the fundamental concepts in React is its **component-based architecture**. Components are the building blocks of a React application, representing independent, reusable UI elements. Each component manages its own state and logic, which allows for modular and maintainable code.

For instance, a "Button" component may handle its own internal state, such as whether it is clicked or hovered, and can be reused across multiple parts of an application. This approach enhances the scalability and maintainability of codebases, especially in large applications.

A **component** in React is a self-contained unit of a user interface (UI) that can be reused, composed, and managed independently. It can be thought of as a function

or class that returns a part of the UI based on the input data (known as "props") and its internal state. React components are responsible for rendering parts of the user interface and handling the logic for how those parts should behave.

Components in React can be broken down into two main types:

a. Class Components:

Class components are the traditional way to define React components, and they are based on JavaScript ES6 classes. A class component must extend **React.Component** (or **React.PureComponent** for optimized performance) and implement a `render()` method, which returns the JSX (React's JavaScript XML syntax) representing the component's UI.

Key Features of Class Components:

- **State Management:** Class components have an internal state, which is stored in the state object. You use the **this.setState()** method to update the state, which triggers a re-render of the component.
 - **Lifecycle Methods:** Class components have lifecycle methods that allow you to hook into different phases of the component's life cycle (e.g., when the component is mounted, updated, or unmounted).
 - **Binding Event Handlers:** In class components, you need to explicitly bind event handler methods to the component instance. This is because **this** keyword inside a method does not automatically refer to the component instance.
- ⇒ **Class Components** offer a traditional approach to React development, providing a clear and structured way to manage state, lifecycle methods,

and event handlers. They are still commonly used in existing codebases and legacy applications.

b. Function Components:

Function components were originally simpler and stateless, but with the introduction of **React Hooks** in version 16.8, they gained the ability to manage state, handle side effects, and access lifecycle-like behavior, making them just as powerful as class components—if not more so.

Key Features of Function Components:

- **Stateless by Default (with Hooks):** Initially, function components were stateless and used purely for presenting UI based on props. However, with the introduction of React Hooks like useState, useEffect, and others, function components can now manage state and handle side effects just like class components.
 - **Handling Side Effects with useEffect:** Function components can use the useEffect hook to perform side effects like fetching data, subscribing to events, or interacting with the DOM. useEffect replaces lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.
 - **No Need to Bind Event Handlers:** Unlike class components, function components don't require explicit binding of methods because functions automatically capture this value they are executed in.
- ⇒ **Function Components**, with the advent of React Hooks, have become the modern approach to writing components. They are more concise, flexible,

and easier to understand, making them the preferred choice for most new React projects.

1.2.2. Core Concepts of Components:

a. **Props (Properties):**

Props (short for properties) are the mechanism React uses to pass data from one component to another, usually from a parent component to a child component. Props are read-only and cannot be modified by the child component. They are immutable, meaning their value cannot be changed by the component receiving them.

Props are often used to pass data to child components and to define the behavior of child components via functions. Think of props as the "input" to a component. They can be anything from simple data (like strings or numbers) to functions (like event handlers).

Props are **used for communication** between components, enabling data to flow from parent to child in a predictable manner. They help create reusable components by allowing a single component to display different content or behavior based on the props it receives.

b. **State:**

State refers to data that is owned and managed by the component itself. It is mutable, meaning it can be updated during the lifecycle of the component. When state changes, React triggers a re-render of the component to reflect the new state, making the UI update automatically in response to changes.

State is used to handle dynamic data that changes over time, such as user inputs, API responses, or any value that should trigger a re-render when updated.

In function components, state is typically managed using the `useState` hook, while in class components, state is managed using `this.state` and `this.setState`.

c. Comparison Props vs. State:

Feature	Props	State
Definition	Props are inputs to a component, passed from a parent component.	State is data that is owned and managed by the component itself.
Mutability	Immutable (cannot be modified by the component that receives them).	Mutable (can be changed within the component).
Data Flow	Props flow downward from parent to child components.	State is local to the component and does not pass between components unless explicitly done so.
Purpose	Used to pass data and event handlers from parent to child components.	Used to manage data that can change over time within a component.
Ownership	Owned by the parent component and passed to the child component.	Owned by the component itself.
Change Handling	Props cannot be changed directly by the child component; they are read-only.	State can be updated within the component using the <u><code>setState</code></u> method (class components) or <u><code>useState</code></u> hook (function components).

Usage	Primarily for passing data and functions to child components.	Primarily for handling dynamic data that affects how the component renders or behaves.
-------	---	--

d. **Lifecycle Methods (Class Components), Hooks (Function Components):**

In class components, **lifecycle methods** are used to handle different stages of a component's life, such as mounting, updating, and unmounting. Some common lifecycle methods are:

- **componentDidMount()**: Called after the component is rendered for the first time.
- **componentDidUpdate()**: Called after a component's state or props have changed.
- **componentWillUnmount()**: Called before the component is removed from the DOM.
- **shouldComponentUpdate()**: Used to determine if a component should re-render in response to state or prop changes.

In function components, React introduced **hooks** to handle similar functionality. The **useEffect** hook is commonly used to manage side effects (like data fetching or DOM manipulation) that would traditionally be handled by lifecycle methods in class components.

e. Composition of Components:

React promotes **composition** of components, which means you can combine multiple smaller components to build more complex UIs. Components can be nested, meaning a component can render other components inside it.

1.2.3. Benefits of Component-Based Architecture:

- **Reusability:** Components can be reused across different parts of the application or even in different applications. This promotes code reuse, reduces redundancy, and simplifies maintenance.
- **Encapsulation:** Components encapsulate their own logic, which makes it easier to reason about and maintain code. Each component can manage its own state and behavior independently.
- **Separation of Concerns:** By splitting the UI into smaller components, each responsible for a specific part of the user interface, developers can better manage concerns related to presentation, data management, and logic.
- **Testability:** Components can be tested independently. Since each component is isolated, unit testing becomes simpler and more reliable.
- **Performance Optimization:** React optimizes the rendering process by only re-rendering the components whose state or props have changed. This allows for better performance, especially in large-scale applications.

1.3 Virtual DOM and JSX (JavaScript XML)

1.3.1. Virtual DOM in React.js:

The **Virtual DOM** is one of the most significant innovations in React that contributes to its high performance and efficient rendering. To understand the

Virtual DOM in depth, it's important to first grasp how the traditional **DOM** (Document Object Model) works in web development.

a. Traditional DOM vs. Virtual DOM:

In a traditional DOM, when a change is made to the application's state, the entire DOM is re-rendered. This involves updating the HTML structure in the browser, which can be a costly operation in terms of performance, especially for complex applications with large UIs. Re-rendering the entire DOM can lead to noticeable lag and sluggish behavior, particularly in real-time or interactive applications.

The **Virtual DOM** is a solution to this performance problem. Instead of updating the actual DOM directly every time a change occurs, React creates an in-memory representation of the DOM, known as the Virtual DOM. This virtual version of the UI is essentially a lightweight copy of the actual DOM that React uses to track changes and determine which parts of the UI need to be updated.

b. How the Virtual DOM Works:

- **Initial Render:** When a React application is first rendered, React creates a Virtual DOM that mirrors the structure of the actual DOM.
- **State Change:** When a component's state or props change, React updates the Virtual DOM first instead of the real DOM. This is much faster because manipulating the Virtual DOM is a less resource-intensive operation.
- **Reconciliation:** After the state change, React compares the updated Virtual DOM with the previous version (this is called **diffing**). It performs an algorithm to identify exactly which parts of the Virtual DOM have changed.

- **Update Real DOM:** After determining the minimal number of changes, React then updates only the affected parts of the real DOM. This avoids unnecessary re-rendering and optimizes performance.
- The reconciliation process ensures that React does not update the real DOM unnecessarily, reducing the cost of DOM manipulations. This fine-grained control over the UI ensures that only the most critical updates are made.
- **Efficient Updates:** Because React only updates the parts of the DOM that have actually changed (rather than re-rendering the entire page), the application can respond quickly to user input. This is particularly beneficial in dynamic applications with high interactivity, such as social media platforms or e-commerce sites.

c. **Advantages of the Virtual DOM:**

- **Performance Optimization:** By minimizing direct interactions with the real DOM, React reduces the number of costly operations, leading to faster rendering times, especially in complex applications.
- **Predictable UI Updates:** The Virtual DOM ensures that UI updates happen consistently and predictably, making the application easier to debug.
- **Improved User Experience:** The efficient rendering process results in smoother transitions and quicker updates to the user interface, leading to a more responsive and fluid user experience.

1.3.2. JSX in React.js:

JSX (JavaScript XML) is a syntax extension for JavaScript that allows developers to write HTML-like code within their JavaScript files. While JSX is

not required to use React, it has become the de facto standard because it simplifies the process of building React components.

It looks similar to HTML, but it's actually a JavaScript expression that gets compiled into regular JavaScript at runtime. JSX allows developers to describe the structure of the UI in a more declarative way, which makes the code more readable and maintainable.

a. **JSX vs. HTML:**

While JSX closely resembles HTML, there are a few key differences:

- **Attributes:** In JSX, attributes like `class` and `for` (common in HTML) are replaced with **className** and **htmlFor**, respectively, because `class` and `for` are reserved keywords in JavaScript.
- **Self-closing Tags:** JSX requires self-closing tags to end with a slash (****, **<input />**), unlike HTML, where this is not necessary.
- **JavaScript Expressions:** You can embed JavaScript expressions inside curly braces **{ }** in JSX. This is a powerful feature, enabling dynamic rendering of content based on variables, conditions, or functions.

b. **Advantages of JSX:**

- **Readability:** JSX improves the readability of the code because it visually represents the structure of the UI. Unlike other frameworks that separate the HTML and JavaScript, JSX combines them, making it easier to understand the flow of the application.

- **Declarative Syntax:** With JSX, developers describe the UI as a function of state. This declarative approach simplifies reasoning about how the UI should appear at any point in time.
- **Error Checking:** Since JSX is converted into JavaScript by a transpiler like Babel, any syntax errors are caught during the build process. This helps prevent issues in production and provides helpful error messages during development.
- **Component Structure:** JSX allows developers to build components with both structure and behavior within the same file, making the development process more cohesive. For example, you can define a button component with its structure (HTML-like JSX) and behavior (JavaScript logic) in one place.

2. UI Libraries: Material-UI and Ant Design

While React provides the foundation for building user interfaces, developers often turn to **UI libraries** to speed up development, ensure design consistency, and enhance the visual appeal of their applications. Two of the most popular UI libraries in the React ecosystem are **Material-UI** (now known as **MUI**) and **Ant Design**. Both libraries offer a rich set of pre-designed, customizable components that follow modern design principles, making them ideal choices for creating professional and responsive applications.

2.1 Material-UI (MUI)

Material-UI is a comprehensive React component library that implements **Google's Material Design** — a design language that emphasizes grid-based layouts, responsive animations, and depth effects such as shadows and lighting.

MUI provides a robust toolkit of components including buttons, forms, modals, menus, cards, and more, all of which are fully responsive and accessible.

2.1.1. Material Design Philosophy

At the heart of Material-UI is **Material Design**, a visual language developed by Google that seeks to create a unified experience across all platforms and device sizes. The design system emphasizes:

- **Hierarchy and Intentionality:** Using shadows, elevations, and transitions to convey importance and user interaction.
- **Consistency:** A standardized grid layout and spacing system for uniformity across all components.
- **Intuitive Interaction:** Clear feedback for user actions, such as hover effects, button ripples, and smooth transitions.

Material-UI faithfully implements these principles, allowing developers to easily create UIs that are both beautiful and highly usable.

2.1.2. Component Ecosystem

MUI comes with a rich ecosystem of UI components, ranging from basic inputs to complex widgets:

- **Basic Components:** Buttons, Typography, Icons, Tooltips, and Avatars.
- **Navigation:** AppBars, Drawers, Tabs, Breadcrumbs, and BottomNavigation.
- **Data Display:** Cards, Chips, Lists, Tables, Accordions, and Badges.

- **Forms and Inputs:** TextFields, Selects, Checkboxes, Radio Groups, Sliders, and Switches.
- **Feedback:** Snackbars, Dialogs, Backdrops, and Progress indicators.

These components are **modular and composable**, allowing developers to build complex UIs by combining smaller pieces in a clean and reusable way.

2.1.3. Customization and Theming

One of MUI's most powerful features is its **theming system**, which makes it easy to align an application's UI with a brand or design specification.

a. Global Themes:

Using the **ThemeProvider**, developers can define global styles such as colors, typography, spacing, breakpoints, and more.

b. Component Customization:

Developers can override default styles at the component level using props or the **sx** prop, enabling rapid prototyping and fine-grained control over styling.

c. Dark Mode Support:

MUI supports dark mode out of the box by switching themes, which is especially useful for apps that cater to user preference or system themes.

2.1.4. Responsiveness and Layouts

Material-UI offers a powerful **grid system** based on CSS Flexbox, as well as utility classes to handle layout and spacing responsively. This makes it ideal for building mobile-friendly UIs.

2.1.5. Conclusion

Material-UI provides developers with a powerful set of tools to build visually stunning and functionally rich web applications. Its combination of design consistency, easy theming, accessibility, and responsive layout system makes it a go-to choice for teams looking to scale their front-end development efficiently. Whether you're building a startup MVP or an enterprise-grade platform, MUI empowers you to focus on user experience without reinventing the design wheel.

2.2 Ant Design (AntD)

Ant Design (AntD) is a comprehensive and enterprise-class UI design system created by **Alibaba**, designed to provide a unified user experience across complex business applications. It is one of the most robust UI frameworks available for React, offering a wide range of components that are both visually appealing and functionally rich.

2.2.1. Philosophy and Design Principles

At the heart of Ant Design is a strong focus on **user-centric, data-driven interfaces**. Its design philosophy is based on four core principles:

- **Natural** – interfaces should be easy to understand and intuitive.
- **Certain** – users should always be clear about the outcome of an action.
- **Meaningful** – elements should communicate with clarity and purpose.

- **Growing** – the system should scale with the product and adapt to change.

These principles are especially important in enterprise environments, where applications often deal with large volumes of data and require highly interactive and reliable UI elements.

2.2.2. Rich Component Library

Ant Design offers over 60 well-documented components out of the box, covering everything from basic UI elements to more advanced data-driven components. Some of the standout components include:

- **Table:** With built-in sorting, filtering, pagination, expandable rows, and editable cells — perfect for managing large datasets.
- **Form:** A powerful, flexible system with support for validation rules, dependencies, and dynamic fields.
- **DatePicker & RangePicker:** Highly customizable date and time selectors.
- **Modal, Drawer, and Popconfirm:** For creating overlays and dialogues that improve user interaction.
- **Tree & TreeSelect:** Hierarchical selectors useful in managing nested structures like file systems or category filters.

These components not only save development time but are also optimized for performance and accessibility.

2.2.3. Customization and Theming

AntD offers a **customizable theming system** using **Less** variables. Developers can easily override default styles, such as colors, spacing, typography, and

breakpoints, to match their brand or application requirements. In recent versions, AntD has also introduced **design tokens** to better support dynamic theming and dark mode.

This makes Ant Design flexible enough to use across different products within the same organization while maintaining brand consistency.

2.2.4. Internationalization (i18n)

Unlike many libraries, Ant Design has **built-in support for internationalization**, which is a critical feature for global businesses. Components can automatically adapt to different locales, date formats, and text directions (LTR/RTL). It supports dozens of languages and is easy to configure via the **ConfigProvider** component.

2.2.5. Form Management & Validation

AntD's **Form** component is one of its most powerful features. It supports:

- Schema-based validation
- Nested field structures
- Dynamic form items
- Dependency-based updates (e.g., showing/hiding fields based on other inputs)

This is particularly useful for building admin panels, registration flows, and settings dashboards where form interactions are complex and critical.

2.2.6. Responsive Design & Layout System

Ant Design includes a **flexible grid system** based on a 24-column layout. It supports responsive design by providing breakpoint-based column sizing, allowing developers to create mobile-first layouts effortlessly.

This makes it easy to create layouts that adapt smoothly to all screen sizes.

2.2.7. Limitations and Considerations

Despite its many strengths, Ant Design can feel heavy for smaller projects or when minimalism is preferred. Some developers may find the default styles opinionated and harder to override than with other libraries like Tailwind or Chakra UI. Additionally, due to its reliance on Less (though CSS-in-JS support is improving), customization may feel less modern compared to Emotion or Styled-Components.

2.2.8. Conclusion

Ant Design stands out as a mature, production-ready UI framework that empowers developers to build sophisticated, consistent, and scalable interfaces with minimal effort. Its extensive component set, enterprise-oriented features, and thoughtful design principles make it particularly well-suited for **complex dashboards, data-heavy systems, and internal tools**.

3. Comparison: React vs. Other Frameworks (Angular and Vue.js)

In the ever-evolving landscape of front-end development, **React**, **Angular**, and **Vue.js** have emerged as the leading technologies for building modern web applications. While all three aim to simplify the process of creating interactive and dynamic user interfaces, they differ significantly in terms of design philosophy, learning curve, performance, and ecosystem.

3.1 React vs. Angular

Angular is a full-fledged front-end framework developed and maintained by Google. Unlike React, which is a library focused primarily on building UI components, Angular provides an all-in-one solution that includes routing, state management, form validation, and dependency injection out of the box.

3.1.1. Framework vs. Library

The most fundamental distinction is that **Angular is a full-fledged framework**, while **React is a UI library**.

- **Angular** offers a complete development environment with built-in tools such as HTTP clients, form handling, routing, and dependency injection. It is an opinionated framework, meaning it enforces specific design patterns and structures.
- **React**, by contrast, focuses exclusively on building user interfaces. To manage routing, state, and side effects, developers must integrate additional libraries such as **React Router**, **Redux**, or **Zustand**.

3.1.2. Language and Syntax

- **Angular** is built using **TypeScript**, a statically typed superset of JavaScript. This provides powerful type-checking and better tooling, particularly beneficial for large-scale, enterprise-level applications. However, it also increases the initial complexity for beginners unfamiliar with strict typing or object-oriented patterns.

- **React** supports both JavaScript and TypeScript, allowing teams to adopt static typing gradually. It uses **JSX** (JavaScript XML), which allows developers to write HTML-like code within JavaScript. JSX is intuitive once understood, but can be unfamiliar to new developers at first.

3.1.3. Data Binding

- **Angular** supports **two-way data binding**, meaning changes in the UI can automatically update the underlying data model and vice versa. This simplifies development for form-heavy applications but can lead to complex data flow in larger apps.
- **React** uses **one-way data binding**, where data flows from parent to child via props. While this requires more boilerplate for updating the UI, it results in **more predictable** and **easier-to-debug** applications, especially as they scale.

3.1.4. State Management

- **Angular** includes services and RxJS (Reactive Extensions for JavaScript) to handle state and asynchronous data streams. This model is powerful but has a steep learning curve due to the functional reactive programming style required.
- **React** offers multiple state management solutions. The built-in **useState** and **useReducer** hooks are enough for many use cases, while complex apps often adopt libraries like **Redux**, **MobX**, or **Recoil** for global state. The introduction of **React Context** and newer tools like **Zustand** has made state management more flexible and lightweight.

3.1.5. Performance Considerations

Both Angular and React deliver high performance, but the strategies they use differ:

- **React** uses a **virtual DOM**, where changes are first made to a lightweight copy of the DOM, and then a diffing algorithm applies the minimal number of changes to the real DOM. This results in efficient UI updates, especially in dynamic interfaces.
- **Angular** uses **real DOM** with a change detection mechanism powered by **Zone.js**. While efficient for many applications, it can introduce performance overhead if not optimized with techniques like **OnPush** change detection strategy and manual DOM updates.

3.1.6. Ecosystem and Tooling

- **Angular** provides a powerful CLI (Command Line Interface) for scaffolding projects, generating components, and managing builds. It enforces a structured folder architecture, which is ideal for large teams and consistent project organization.
- **React**, being more modular, relies on third-party tools for similar functionality. While the **Create React App** CLI provides a solid foundation, developers often customize their build setup using **Vite**, **Webpack**, or **Next.js**.

3.1.7. Community and Industry Adoption

- **React** has a **larger global developer community** and is widely adopted by major companies such as **Meta (Facebook)**, **Netflix**, **Instagram**, and **Airbnb**. Its vast ecosystem of libraries, tools, and tutorials makes it beginner-friendly and highly extensible.

- **Angular** is often favored in **enterprise environments**, with companies like **Google**, **Microsoft**, and **Deutsche Bank** using it for large, complex applications. Its strong typing and structured architecture appeal to teams that value consistency and formal development practices.

3.1.8. Conclusion

Choosing between **React** and **Angular** ultimately depends on the **project requirements**, **team expertise**, and **long-term scalability goals**. React offers unmatched flexibility, a shallow learning curve, and a vast ecosystem, making it suitable for a wide range of projects. Angular, on the other hand, provides a more structured and comprehensive development environment, ideal for large-scale enterprise applications that benefit from consistency and strong typing.

In the modern development landscape, both technologies are capable of delivering high-performance, responsive applications. The decision is less about which tool is "better" and more about which tool aligns best with the **project's scope**, **team structure**, and **desired development workflow**.

3.2 React vs. Vue.js

Vue.js, created by Evan You, is another popular JavaScript framework that combines features from both React and Angular. It is designed to be progressive, meaning it can be adopted incrementally—used for anything from small widgets to full-scale applications.

While both **React** and **Vue.js** are used to build dynamic user interfaces and share similarities such as component-based architecture and virtual DOM rendering, their underlying philosophies, ecosystems, and developer experiences differ

significantly. Understanding these nuances is key when deciding which library best suits a particular project or development team.

3.2.1. Design Philosophy and Core Concepts

- **React** is **library-first** and intentionally minimal. It focuses solely on building UI components, leaving concerns like routing and global state management to external libraries. This makes React extremely **flexible**, but also puts more responsibility on the developer to structure the project.
- **Vue.js** is more of an **integrated framework**, offering official tools for routing (**Vue Router**), state management (**Pinia** or **Vuex**), and even full-stack development (**Nuxt.js**). This opinionated structure helps teams move faster with fewer decisions to make.

3.2.2. Component Syntax and Developer Experience

- **React** uses **JSX**, a syntax extension that mixes HTML with JavaScript. JSX is powerful, but can be confusing for newcomers, as logic and markup are tightly integrated.
- **Vue** uses **template-based syntax**, separating structure (HTML), logic (JavaScript), and styling (CSS) in a single **.vue** file, which many find more readable and beginner-friendly. This makes Vue especially appealing for developers coming from traditional HTML/CSS/JS backgrounds.

3.2.3. State Management

- **React** uses **hooks** like **useState** and **useEffect** to manage component state and side effects. For larger applications, libraries like **Redux**, **Zustand**, or

Recoil are often used. However, the ecosystem offers many state management options, which can be both empowering and overwhelming.

- **Vue** previously relied on **Vuex**, but has since shifted toward **Pinia**, a more modern and intuitive state management solution. Since Pinia is an official library, it integrates smoothly with the Vue ecosystem, offering consistency and simplicity.

3.2.4. Reactivity System

- React's reactivity is based on **state diffing and reconciliation** via the virtual DOM. Updates are triggered explicitly using **setState** or hook setters.
- Vue's reactivity system is **more automatic** and **fine-grained**. Vue tracks dependencies at the property level using reactive proxies, which means it can optimize updates even more precisely with less developer intervention.

3.2.5. Ecosystem and Tooling

- **React** has a vast ecosystem, including tools like **Next.js** for server-side rendering and static generation, **React Native** for mobile development, and a wide array of third-party libraries. This ecosystem is ideal for custom solutions and large, flexible applications.
- **Vue** offers a more curated and integrated experience. The **Vue CLI** and **Vite** make setup and scaffolding easy, while **Nuxt.js** provides a full-stack SSR solution out of the box. Vue also tends to have more official plugins, which reduces the need to vet third-party tools.

3.2.6. Community and Industry Adoption

- **React** has broader adoption worldwide, especially in enterprise environments and large-scale applications. Major companies like **Facebook**, **Netflix**, **Airbnb**, and **Shopify** use React in production.
- **Vue** has strong popularity in Asia and is favored in smaller to medium projects or where fast prototyping is required. Companies like **Alibaba**, **Xiaomi**, and **GitLab** use Vue.

3.2.7. Conclusion

In summary, **React** is ideal for developers and teams who value **flexibility**, **performance**, and a vast ecosystem, especially when building large-scale applications with custom architecture. On the other hand, **Vue.js** excels in projects where **ease of use**, **fast development**, and **intuitive design patterns** are prioritized. While both tools are capable of building modern, high-performance interfaces, the best choice depends on the project's complexity, team experience, and long-term goals.

Chapter 2. Product Design and Demonstration

1. General Introduction to the Website Project

Decor Dream is an e-commerce website project that specializes in providing workspace decor products and modern decorative items. This website was developed by a group of students as part of the Web Programming and Applications course, with the dual objective of applying technical knowledge in practice and exploring a domain that is closely tied to today's digital lifestyle.

The idea behind Decor Dream originated from a member's personal passion for technology, workspace aesthetics, and the art of organization. From this inspiration, the entire team aimed to convey a meaningful message about the importance of investing in personal working environments, as well as the role of hobbies and entertainment in maintaining balance in modern life.

The website not only serves as a platform to showcase various workspace design styles but also acts as a marketplace, an information-sharing channel, and a user-friendly experience with an eye-catching and intuitive interface.

The target audience of Decor Dream includes:

- Technology enthusiasts.
- Students seeking creative study spaces.
- Office workers looking to enhance their work environment.
- Gamers or remote workers who desire a workspace that is both stylish and personalized.

With the goal of combining aesthetics and functionality, the Decor Dream project demonstrates the practical application of React.js and modern web technologies in building a compact, effective, and realistic e-commerce system.

2. Key Features and Functional Highlights

The Decor Dream website is designed to deliver an intuitive, convenient, and optimized shopping experience for users. The following are the core features and standout functionalities implemented during the development of the system:

Diverse product categories by theme: A total of 22 clearly categorized themes (e.g., gaming setups, LED lighting, monitor stands, decorative plants, etc.) allow users to easily browse products based on their interests or intended use.

Smart search by product name: The search bar enables users to enter keywords related to the product name and quickly retrieve relevant results. This filtering mechanism is implemented on the client-side, ensuring fast response times and a smooth user experience.

Product pagination: Products are displayed in sets of 9 items per page to optimize screen space and maintain a visually pleasant browsing experience. Pagination not only prevents content overload but also enhances professionalism in how information is presented.

Product detail page: Upon selecting a product, users are directed to a detailed view that features an image carousel and comprehensive information such as the product name, price, star rating, and additional descriptions.

Popup-style shopping cart: The shopping cart interface is implemented as a convenient popup window, allowing users to:

- View selected items.
- Increase or decrease quantities.
- Remove items from the cart.
- See real-time updates to the total amount.

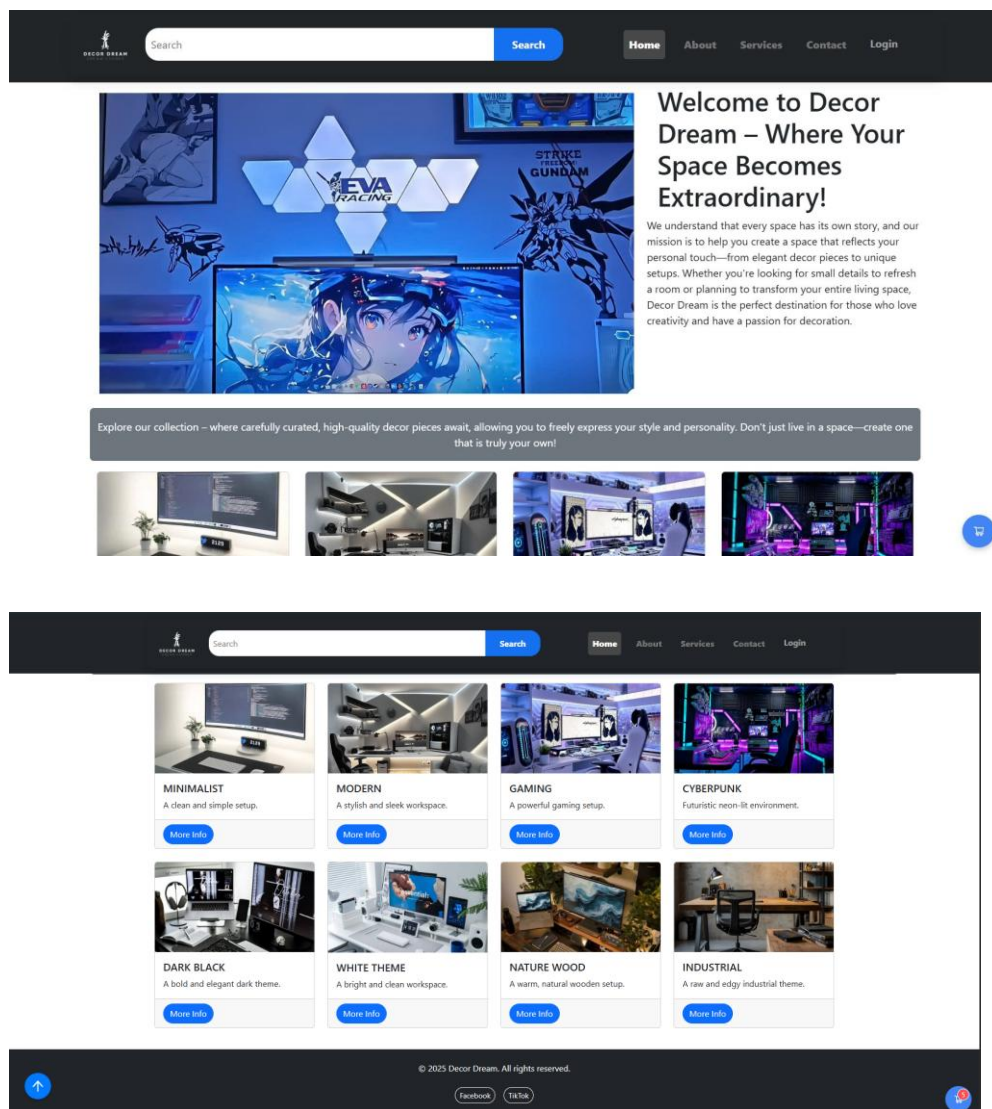
Duplicate prevention in the cart: The system ensures that duplicate products cannot be added to the cart. If an item already exists, users can only adjust its quantity, thereby avoiding uncontrolled repetition.

Login/logout state management: The website supports toggling between logged-in and logged-out states, with corresponding adjustments to the interface and available functionalities (e.g., displaying "Login" or "Logout" buttons, showing or hiding the cart).

Cart data saved in JSON format: Cart information is stored in JSON files, simulating server-side order storage. This feature supports testing and mimics backend behavior in a client-side environment.

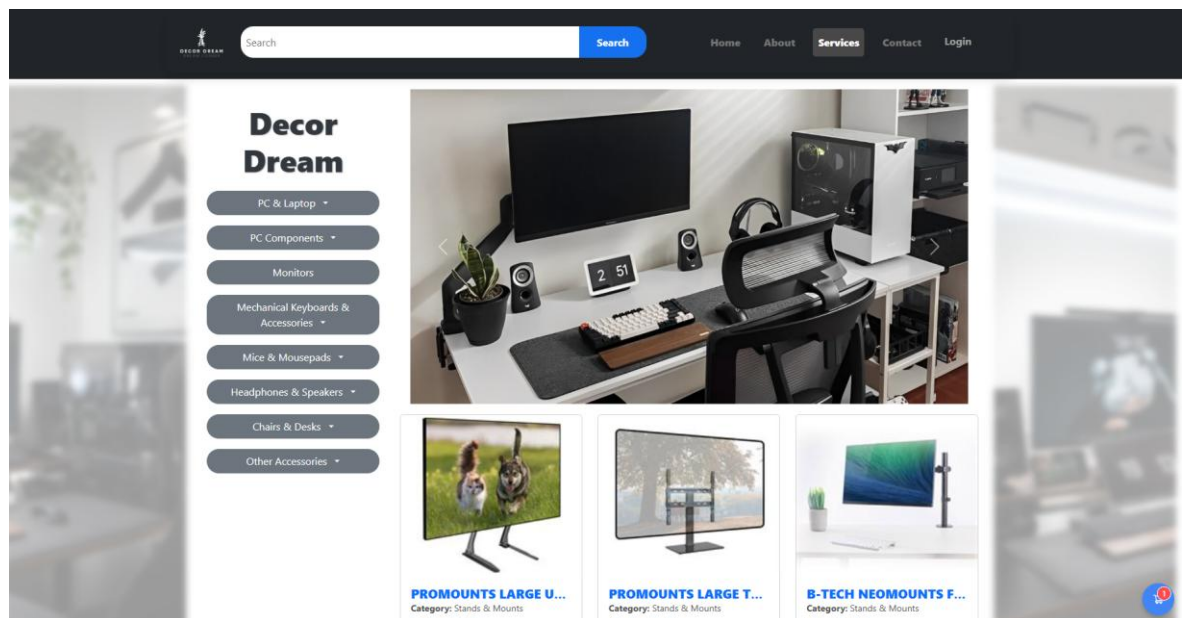
3. UI and Illustrative Images

Homepage: The landing page displays the main options available to users when they visit the website. It features an introductory video, followed by various computer desk setup style options. When users click the **"More Info"** button, a modal (from a UI library) will pop up, showing detailed information for each style.




About page: The About page introduces key members of the team with professional profile cards, each displaying a circular profile image, full name, phone number, and personal email address. This layout fosters trust and gives a personal touch to the brand. Beneath the profiles, there is a highlighted section featuring a short brand message.

Services page: The content layout includes two banners positioned on the left and right sides of the page. At the center is the main display area, featuring a menu with multiple product categories. When a user clicks on a category in the menu (on the left), nine product cards related to that category will be displayed per page. Users can browse more products by navigating through pagination. Additionally, a carousel is placed above the product cards to enhance the visual appeal of the page.



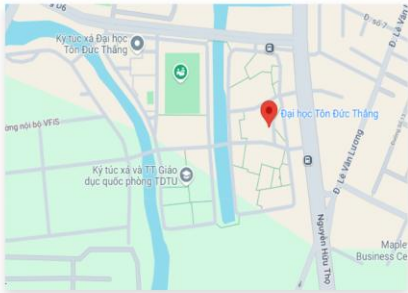
Contact page: Users can send feedback about the website by visiting this page. After filling out the form, a confirmation message will appear, and a thank-you email will be sent to the email address provided in the form.



[Home](#)
[About](#)
[Services](#)
[Contact](#)
[Login](#)

Drop us a message

Contact Us



Address: 19 Nguyen Huu Tho, Tan Phong Ward, District 7, HCM City, Vietnam

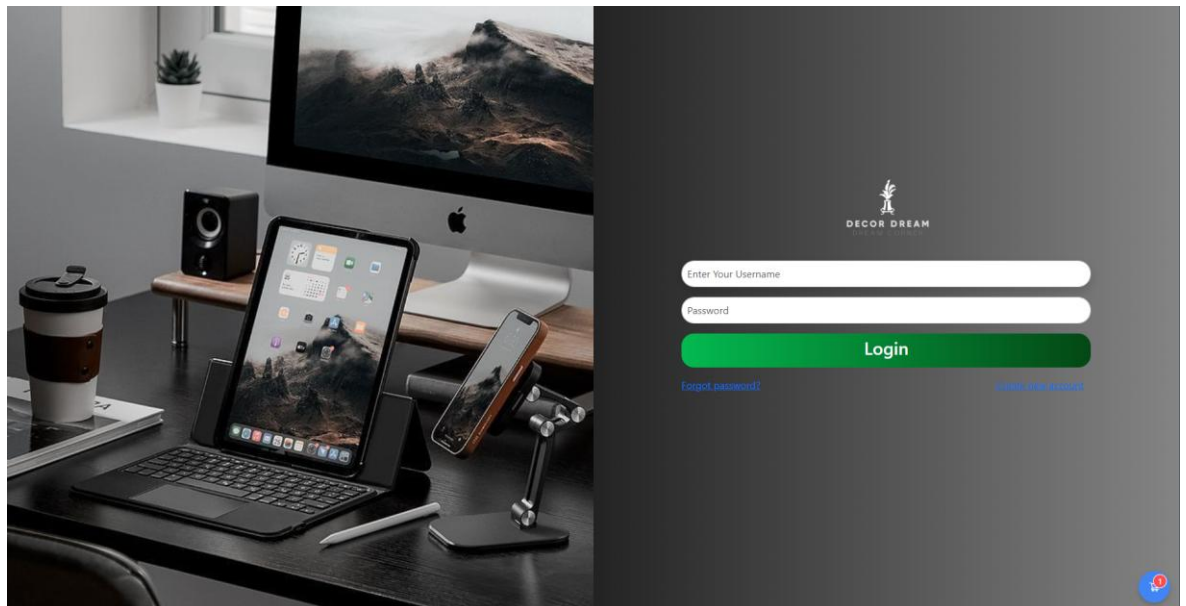
Phone: +1235 2355 98

Email: ngochithuan.dev@gmail.com

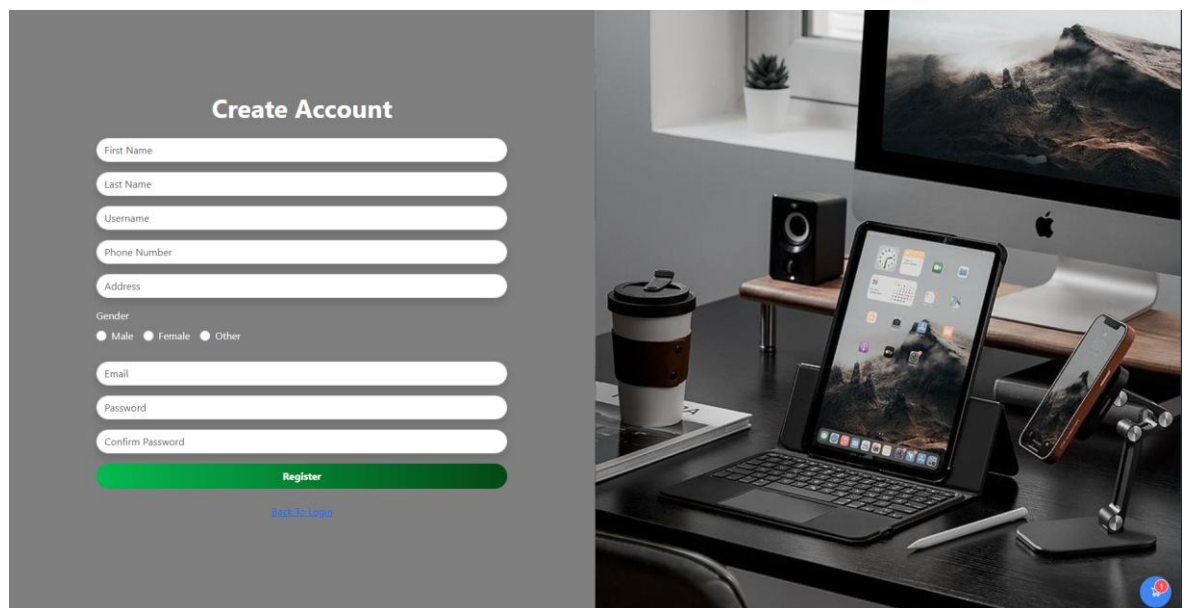
Website: decordream.com

Login page: The overall design features a gray gradient color scheme, with the login button styled in a contrasting green. When hovered over, the button changes color to enhance interactivity. At the top of the page, a logo is displayed to strengthen brand recognition.

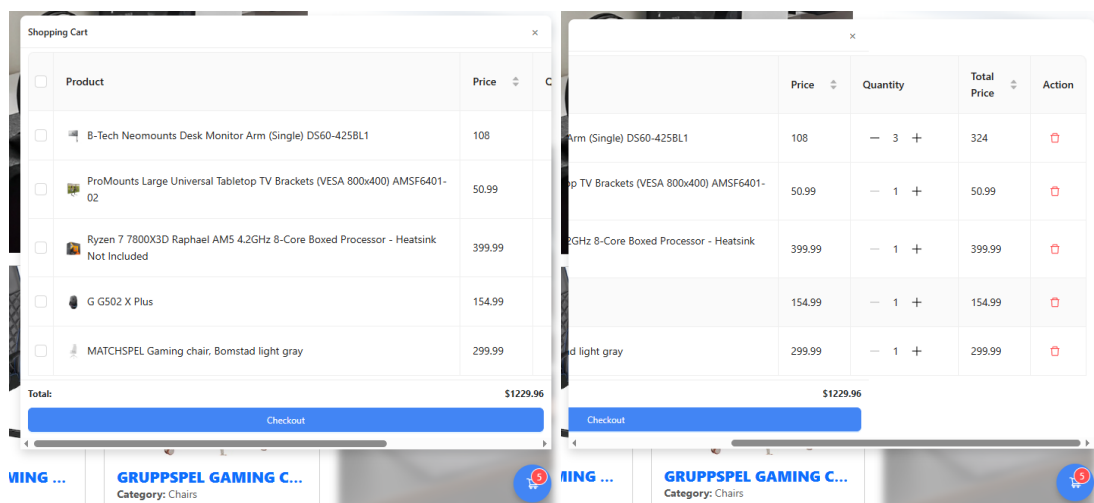
Users can log in to access personalized features specific to their accounts. Additionally, two other options are available: "Forgot Password" for password recovery and "Create New Account" for users who haven't registered yet.



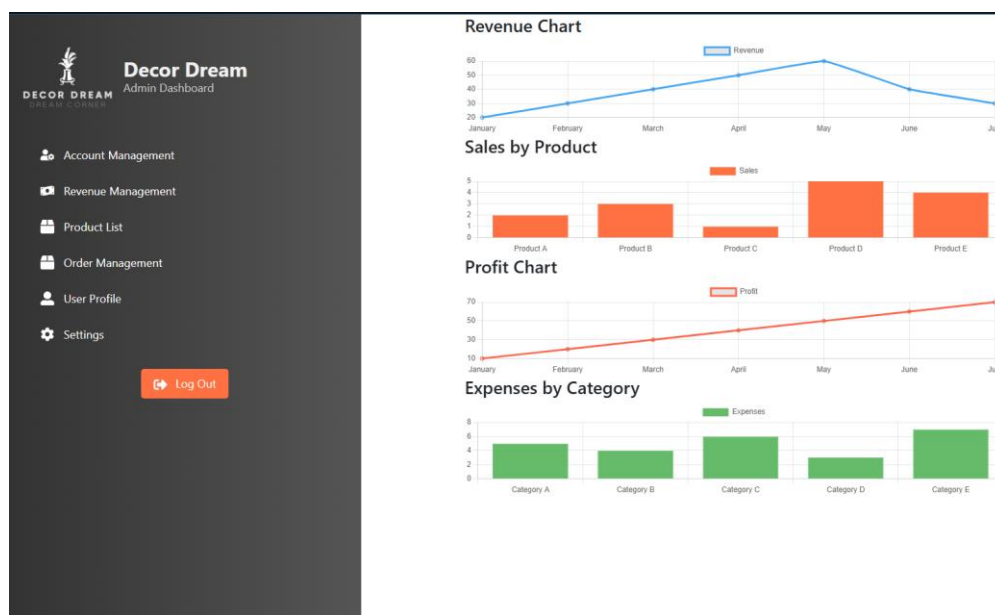
Register page: The page features a split-screen layout. On the left, users can fill out a clean and minimalistic registration form set against a gray gradient background. The form includes fields for first name, last name, username, phone number, address, gender selection, email, password, and password confirmation. The "Register" button stands out in a vibrant green gradient and features a hover effect for better interactivity. Below the form is a "Back to Login" link for easy navigation. On the right side of the page, a high-quality lifestyle image of a modern workspace with Apple devices adds visual appeal and reinforces a tech-savvy, professional tone.



Shopping cart popup: In the shopping cart interface, users can view a complete list of products they have added during the browsing process. The system is designed to prevent duplicate entries, ensuring that the same product cannot be added more than once. Instead, users are encouraged to adjust the quantity of each item directly within the cart. The interface also supports removing items from the shopping list with a single click, and it displays the total price in real time, updating automatically as users make changes. Moreover, the shopping cart has been optimized for responsiveness, adapting seamlessly to various screen sizes and devices to maintain usability on both desktop and mobile platforms. This provides a smooth and accessible experience for all users, regardless of the device they use.



Admin dashboard demonstration: The page features a two-panel layout. On the left is a dark gradient sidebar containing the brand logo, application name (Decor Dream Admin Dashboard), and a navigation menu with key management options: Account Management, Revenue Management, Product List, Order Management, User Profile, and Settings. A bright orange “Log Out” button is placed at the bottom for quick access. The right side presents various data visualizations to provide an overview of the system’s performance: Revenue Chart, Sales by Product, Profit Chart, Expenses by Category.



4. Evaluation and Challenges Encountered

The project demonstrated a high level of overall effectiveness, characterized by a user-friendly interface, smooth interaction flow, and well-structured logic. Nonetheless, several challenges arose during the development process. A major difficulty was the implementation of a robust shopping cart system, which required handling duplicate item detection and dynamic quantity updates — a task that proved to be both technically and logically demanding.

Another significant challenge stemmed from the transition to React, particularly for team members previously accustomed to traditional PHP development. Core concepts such as component-based architecture, state management, and the use of props necessitated a steep learning curve. Furthermore, managing data transmission between components and implementing page routing through React Router added to the project's complexity.

To address the need for product data, a custom Python script was developed to crawl and extract relevant information from existing commercial websites. Overcoming these challenges required a multifaceted approach, including consulting official documentation, leveraging community resources such as StackOverflow, and engaging in peer support and discussion.

5. Future Development Direction

It is important to emphasize that this version of the project primarily focuses on the front-end user interface. In future development phases, the application will be integrated with a real backend using PHP and connected to a relational database. This will replace the current use of a temporary JSON server and allow for dynamic data handling, user authentication, and persistent storage — ultimately enabling a fully functional, production-ready system.

REFERENCES

1. Alex Banks, Eve Porcello, [2020], Learning React: Functional Web Development with React and Redux, O'Reilly Media, United States of America.
2. MUI Team, [2023], Material UI Documentation, MUI Official Docs, <https://mui.com>.
3. Ant Design Team, [2023], Ant Design React UI Framework Documentation, Ant Design Official Docs, <https://ant.design>.
4. Stephen Fluin, [2019], Angular in Action, Manning Publications, United States of America.
5. Evan You, Vue.js Team, [2023], Vue.js Guide and API Documentation, Vue.js Official Docs, <https://vuejs.org>.