CHAPTER 1

Introduction

1.1 BACKGROUND OF STUDY

In today's fast-paced world, where every penny counts, consumers are increasingly turning to price comparison websites to make informed purchasing decisions. These platforms act as virtual marketplaces, bridging the gap between buyers and sellers by offering a comprehensive array of product prices. Among these, PriceWar.com stands out for its unique focus on essential home groceries, catering to the everyday needs of consumers. From onions and chilies to fish and chicken, PriceWar.com provides users with a centralized hub for comparing prices, thereby empowering them to make cost-effective choices.

The prevalence of internet usage, with a staggering 59% penetration rate in Malaysia alone, underscores the relevance and potential impact of PriceWar.com. As more individuals embrace online shopping and digital convenience, the demand for platforms like PriceWar.com is expected to soar, offering a timely solution for busy consumers seeking to streamline their grocery shopping experience.

1.2 PROBLEM STATEMENT

While the concept of price comparison websites has gained traction in various sectors, the domain of home groceries remains relatively underserved, particularly in regions like India. This dearth of comprehensive platforms leaves consumers grappling with the challenge of effectively comparing prices for essential items, often resulting in overspending or missed savings opportunities. Moreover, prevalent misconceptions, such as assuming that supermarkets invariably offer the best prices, further compound the issue, leading to suboptimal purchasing decisions.

For retailers, traditional advertising methods present their own set of challenges, including high costs and adverse environmental impacts. In this landscape, the need for a sophisticated, user-

friendly price comparison website like PriceWar.com becomes increasingly apparent, offering a win-win solution for both consumers and sellers alike.

1.3 OBJECTIVES

This project aims to achieve the following objectives:

1 **Provide customers with a comprehensive list of price comparisons:** By aggregating prices from various retailers and supermarkets, PriceWar.com aims to arm consumers with the information they need to make informed purchasing decisions.
2 **Foster price consciousness among consumers:** Through regular exposure to price differentials, PriceWar.com seeks to cultivate a culture of mindful spending among users, empowering them to prioritize value and savings.
3 **Regularly update the price database:** To ensure accuracy and relevance, PriceWar.com is committed to maintaining a dynamic database that reflects real-time price fluctuations and promotions.
4 **Offer users a seamless platform:** By prioritizing user experience and accessibility, PriceWar.com endeavors to provide a user-friendly interface that facilitates effortless price comparison and navigation.

1.4 SCOPE OF STUDY

The scope of the Deal Checker project encompasses all internet users in India, with a particular emphasis on individuals utilizing the internet for business purposes. As internet usage continues to surge in Malaysia, PriceWar.com stands poised to capture a significant market segment, offering unparalleled convenience and value to users. Moreover, the platform's mobile accessibility ensures that users can access price comparisons anytime, anywhere, further enhancing their shopping experience.

Additionally, social media integration presents a ripe opportunity for PriceWar.com to extend its reach and engage with users on popular platforms, thereby maximizing its impact and relevance in the digital landscape.

SIGNIFICANCE OF THE PROJECT

Deal Checker isn't just another price comparison website it's a game-changer for both consumers and sellers. For consumers, it represents a lifeline in an increasingly complex retail landscape, offering clarity and transparency amidst a sea of choices and price points. By empowering users to make informed decisions, PriceWar.com plays a pivotal role in promoting financial literacy and responsible spending habits.

For sellers, Deal Checker offers a powerful marketing platform to showcase their products and promotions to a captive audience. Through targeted email campaigns and strategic partnerships, sellers can leverage Deal Checker to drive sales and boost brand visibility in a cost-effective manner.

1.5 FEASIBILITY STUDY

The feasibility of the Deal Checker project hinges on both technical and operational considerations. From a technical standpoint, the utilization of cutting-edge tools such as NEXT.js, Axios, and Cheerio ensures a robust and scalable platform capable of delivering real-time price comparisons with precision and efficiency. Likewise, the integration of languages like Tailwind enhances the platform's aesthetics and user experience, further solidifying its appeal to consumers.

Operationally, Deal Checker is designed to meet the evolving needs and expectations of its users, offering a seamless and intuitive interface that simplifies the price comparison process. By focusing on technical excellence and operational efficiency, PriceWar.com endeavors to set a new standard for price comparison websites, revolutionizing the way consumers shop for groceries online.

CHAPTER 2

Technologies Used

2.1. Next.js for Dynamic UI:

Description: Next.js, a React framework, is chosen for its ability to create dynamic and server-rendered web applications. Features like server-side rendering (SSR) and efficient client-side routing contribute to a seamless user experience.

2.1.1. Why Next.js:

- React Framework: Next.js is a React framework that simplifies the development of React applications by providing a set of conventions and tools.
- Server-Side Rendering (SSR): Next.js supports SSR out of the box, enabling faster page loads by rendering pages on the server and sending the fully rendered page to the client
- Static Site Generation (SSG): It also supports SSG, allowing you to pre-render pages at build time, resulting in highly optimized static assets that can be served by a CDN.
- Automatic Code Splitting: Next.js automatically splits code into small, optimized chunks, which are loaded only when needed. This helps reduce initial loading times and improves performance.
- Routing System: Next.js has a powerful and intuitive routing system that simplifies the creation of dynamic routes and nested routing structures.
- Zero Configuration: Getting started with Next.js is easy, thanks to its zero-configuration setup. You can start building a React application without the need for complex configurations.
- Customizable Head and Document: Next.js provides a customizable <Head> component for managing document head elements and a _document.js file for customizing the HTML document.
- API Routes: Easily create API routes within your Next.js application, allowing you to handle server-side logic or connect to databases on the server side.
- Built-in CSS Support: Next.js comes with built-in support for styling, including CSS, Sass, and CSS-in-JS solutions. It also supports the latest CSS features like CSS modules.

- Extensibility: Next.js is highly extensible, allowing developers to add custom webpack configurations, plugins, and middleware to tailor the development environment to specific needs.
- Community and Ecosystem: Being a popular and widely adopted framework, Next.js has a thriving community and a rich ecosystem of plugins, libraries, and tools that can enhance the development experience.
- Continuous Development: Next.js is actively developed and maintained by the community and the Vercel team, ensuring that it stays up-to-date with the latest React features and best practices.

2.2. Axios for Efficient Data Retrieval:

Description: Axios, a JavaScript library, excels in making HTTP requests. Its simplicity and flexibility make it ideal for fetching data from external APIs.

2.2.1. Why Axios:

- Promise-Based: Axios is a promise-based HTTP client, allowing you to take advantage of JavaScript's native promise syntax for handling asynchronous operations.
- Browser and Node.js Support: Axios can be used both in the browser and in Node.js environments, providing a consistent API for making HTTP requests across different platforms.
- Simple API: Axios provides a simple and clean API for sending HTTP requests. It supports various HTTP methods like GET, POST, PUT, and DELETE, and you can customize requests with headers, params, and data.
- Request and Response Interceptors: Axios allows you to use interceptors to globally handle request and response logic. This can be useful for tasks such as modifying headers, logging, or handling errors.
- Canceling Requests: Axios provides a built-in mechanism for canceling requests, which can be useful in scenarios where a user navigates away from a page or when a component unmounts.
- Automatic JSON Data Transformation: Axios automatically converts request and response data to and from JSON format, simplifying the process of working with JSON APIs.

- Cross-Origin Resource Sharing (CORS) Handling: Axios handles CORS automatically, simplifying the process of making requests to different domains.
- Error Handling: Axios provides a convenient way to handle HTTP errors by rejecting promises when the status code indicates an error. This makes it easy to implement error handling logic in your applications.
- Support for Download and Upload Progress: Axios supports tracking the progress of download and upload operations, which can be useful for displaying progress bars in your applications.
- Security Features: Axios includes built-in protection against common security vulnerabilities, such as Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF).
- Community Support: Axios is widely used and has a large community, making it easy to find resources, tutorials, and solutions to common issues.
- Integration with Promises: As a promise-based library, Axios integrates seamlessly with other promise-based JavaScript features, making it easy to work with asynchronous code in a consistent manner.

2.3. Cheerio for Web Scraping:

Description: Cheerio, a lightweight HTML parsing library, is essential for web scraping, enabling efficient data extraction from HTML.

2.3.1. Why Cheerio:

- jQuery-Like API: Cheerio provides a jQuery-like API for parsing and manipulating HTML and XML documents. Developers familiar with jQuery will find it easy to use and navigate.
- Server-Side DOM Manipulation: Cheerio is designed to work on the server side, making it well-suited for tasks like web scraping and parsing HTML in a Node.js environment.
- Lightweight: Cheerio is a lightweight library that focuses on providing a simple and efficient way to traverse and manipulate HTML documents. It does not have the overhead of a full browser environment.

- Selector Support: Cheerio supports CSS-style selectors, allowing you to easily target and manipulate specific elements in the HTML document. This is similar to how selectors work in jQuery.
- Fast and Efficient: Cheerio is known for its speed and efficiency when it comes to parsing and manipulating HTML. It's a performant choice for tasks that involve dealing with large amounts of HTML data.
- No External Dependencies: Cheerio has minimal external dependencies, which makes it easy to integrate into Node.js projects without worrying about a large number of additional dependencies.
- HTML Parsing: Cheerio excels at parsing HTML strings and turning them into a traversable DOM structure, allowing developers to extract specific information or modify the document as needed.
- Attribute Manipulation: Cheerio provides methods for easily manipulating attributes of HTML elements, making it convenient to modify or extract specific data from the document.
- Modular: Cheerio is modular, allowing you to use it in combination with other Node.js libraries for tasks such as making HTTP requests (e.g., with Axios) before parsing the HTML.
- Well-Documented: Cheerio has comprehensive documentation and a straightforward API, making it easy for developers to get started and find the information they need.
- Community Support: Cheerio has a supportive community, and developers often share tips, tricks, and solutions for common use cases related to web scraping and HTML parsing.
- Customizable Parsing Options: Cheerio allows you to customize parsing options, providing flexibility for different use cases and handling various HTML structures.

2.4. Tailwind for Stylish Design:

Description: Tailwind CSS, a utility-first framework, offers pre-defined classes for consistent and visually appealing design.

2.4.1. Why Tailwind:

- Utility-First Approach: Tailwind CSS follows a utility-first approach, providing a set of low-level utility classes that can be composed to build designs directly in your markup. This approach eliminates the need for writing custom CSS styles for every component.
- Flexibility: Tailwind is highly flexible, allowing developers to create unique designs without being restricted by pre-designed components. It provides a wide range of utility classes for styling elements, spacing, typography, and more.
- Responsive Design: Tailwind makes it easy to create responsive designs with utility classes that allow you to specify styles based on screen sizes. This helps in building websites that work well on various devices.
- Customization: Tailwind is highly customizable, and you can configure it to include only the styles you need for your project. This helps in keeping the final CSS file size minimal.
- Easy to Learn: With its utility-first approach, Tailwind is easy to learn, especially for developers familiar with HTML and CSS. The class names are intuitive, making it simple to understand and use.
- Rapid Prototyping: Tailwind enables rapid prototyping by allowing developers to quickly build and iterate on designs using utility classes. This is especially beneficial during the early stages of development.
- Community Plugins: Tailwind has a thriving community that contributes various plugins, extensions, and tools. These can enhance the functionality and utility of Tailwind for specific use cases.
- Dark Mode Support: Tailwind includes built-in support for implementing dark mode in your designs, making it easier to create websites that adapt to user preferences or system settings.
- PurgeCSS Integration: Tailwind can be integrated with tools like PurgeCSS to automatically remove unused styles, resulting in smaller CSS file sizes for production, despite the extensive utility classes available.
- Theming: Tailwind allows you to define custom themes, making it easy to maintain a consistent design system across your project. You can customize colors, fonts, spacing, and more.
- Active Development: Tailwind is actively developed and maintained, ensuring

2.5. Redis for In-Memory Data Storage and Caching:

Description: Redis is an open-source, in-memory data structure store used as a database, cache, and message broker. Its fast performance, versatility, and support for various data structures make it a popular choice for real-time applications, caching, and queuing systems.

2.5.1. Why Redis:

- In-Memory Data Storage: Redis stores data in memory, resulting in extremely fast read and write operations. This makes it ideal for use cases that require high throughput and low latency, such as caching frequently accessed data.
- Data Structures: Redis supports various data structures such as strings, hashes, lists, sets, sorted sets, bitmaps, and hyperloglogs. This versatility allows developers to model complex data types and perform operations like atomic increments, set intersections, and sorted set operations efficiently.
- Persistence Options: Redis offers different persistence options, including snapshotting and append-only file (AOF) persistence, ensuring data durability and recoverability in case of failures.
- Pub/Sub Messaging: Redis provides pub/sub messaging capabilities, allowing applications to implement real-time communication and event-driven architectures.
- Lua Scripting: Redis supports Lua scripting, enabling developers to execute complex operations atomically on the server side, reducing round-trips between the client and server.
- High Availability: Redis supports master-slave replication and automatic failover through Redis Sentinel or Redis Cluster, ensuring high availability and data redundancy.
- Scalability: Redis can be horizontally scaled by adding more nodes to the cluster, allowing applications to handle increasing workloads seamlessly.
- Advanced Features: Redis offers advanced features such as transactions, pipelining, and client-side caching, enhancing performance and flexibility in application development.
- Geospatial Indexing: Redis includes support for geospatial indexing and queries, enabling location-based searches and proximity calculations.
- Redis Modules: Redis can be extended with custom modules, allowing developers to add new functionalities such as search, machine learning, and graph processing.

- Active Community: Redis has a large and active community that provides support, documentation, and contributions, ensuring its continuous improvement and adoption in various use cases.
- Integration Ecosystem: Redis integrates with popular programming languages, frameworks, and tools, making it easy to use in existing software stacks and architectures.

2.6. JWT-Based Authentication for Secure User Authorization:

Description: JSON Web Tokens (JWT) are a compact, URL-safe means of representing claims to be transferred between two parties. JWTs can be used to authenticate users and securely transmit information between the client and server.

2.6.1. Why JWT-Based Authentication:

- Stateless Authentication: JWT-based authentication is stateless, meaning server-side sessions or database lookups are not required to authenticate each request. This reduces server overhead and improves scalability.
- Security: JWTs can be digitally signed using a secret or public/private key pair, ensuring data integrity and preventing tampering. This provides a secure way to transmit authentication tokens over the network.
- Decentralized Authorization: Since JWTs contain all necessary information within the token itself, authorization decisions can be made locally by the client or intermediate services without needing to query a centralized authentication server.
- Cross-Domain Authentication: JWTs can be easily shared across different domains or services, enabling single sign-on (SSO) and seamless authentication between multiple applications.
- Customizable Claims: JWTs can contain custom claims or metadata in addition to standard claims like expiration time and issuer, allowing developers to include user-specific data or permissions within the token.
- Scalability: JWT-based authentication scales well in distributed systems, as each service or microservice can independently verify JWTs without relying on a centralized authentication server.

- Performance: Since JWTs are self-contained and do not require additional database lookups or session management, authentication can be performed quickly, reducing latency and improving application performance.
- Compatibility: JWTs can be used with any programming language or framework that supports cryptographic operations, making them a versatile choice for implementing authentication in a wide range of applications.
- Industry Standard: JWTs are widely adopted and supported by industry standards such as OAuth 2.0 and OpenID Connect, ensuring interoperability and compatibility with existing authentication protocols and frameworks.

CHAPTER 3

API Endpoints

3.1. Dynamic API Endpoint:

Description: The dynamic API endpoint (https://deal-checker.vercel.app/api/item/*) dynamically retrieves detailed product information based on user queries.

3.1.1. Why Dynamic API Endpoint:

Flexibility: Accommodates a wide range of product queries for adaptability. Real-Time Data: Provides up-to-date information for an enhanced user experience.

3.1.2. Feature Explanation:

This documentation provides a comprehensive explanation of the code used to scrape data fro m a website using Axios and Cheerio in a Next.js application.

3.1.3. Importing Required Modules:

To begin with, the code imports the necessary modules: NextResponse from the "next/server" p ackage, axios, and cheerio. These modules are required for making HTTP requests and parsing HTML respectively.

3.1.4. Setting Up URLs:

The code defines two URL variables: main_url and base_url. These variables store the main UR L and the base URL for the search query on Google Shopping.

3.1.5. Scraper Function:

The scraper function is an asynchronous function that takes a URL as an input parameter. This function is responsible for making an HTTP GET request to the specified URL and scraping data from the response.

Within the scraper function, the following steps are performed:

1. Make an HTTP GET request to the specified URL using Axios and retrieve the respon se.
2. Load the response data into a Cheerio instance to parse and manipulate the HTML.
3. Iterate over each element with the class "sh-dgr__content" and extract the desired data using Cheerio selectors.
4. Construct an object with the extracted data and push it to the data array.
5. Return the data array.

3.1.6. GET Request Handler:

The code exports an async function named GET that handles the GET requests to the server. This function takes two parameters: request and {params}. Within the GET function, the following steps are performed:

1. Extract the value of the item parameter from the params object.
2. Construct the search URL by appending the item value to the base_url.
3. Call the scraper function with the constructed URL and await the scraped data.
4. Return the scraped data as a JSON response using NextResponse.json().

Example Usage:

This code can be used to fetch data from a website using Axios and Cheerio in a Next.js appli cation. The scraped data is returned as a JSON response, which can be further processed or di splayed as needed.

3.2. Comparison API Endpoint:

Description: The comparison API endpoint (https://deal-checker.vercel.app/api/comparison/*) offers a comprehensive view of product pricing and delivery details across various online retailers.

3.2.1. Why Comparison API Endpoint:

Informed Decision-Making: Enables users to make decisions based on pricing, delivery, and brand reputation.

User Empowerment: Provides the ability to choose the best option based on individual preferences.

3.2.2. Usage

To use the web scraper, you need to import the required modules and call the scraper function with the URL of the product page on Google Shopping as the argument. The scraper function returns an array of objects containing the scraped data.

Here's an example of how to use the web scraper:

In the example above, the GET function is an API endpoint that receives a request and extracts the comp parameter from the request URL. It then constructs the URL for the product page on Google Shopping and calls the scraper function to retrieve the data. The scraped data is returned as a JSON response using the NextResponse.json method.

3.2.3. Scraper Function

The scraper function is the main function responsible for scraping the data from the provided URL. It uses Axios to make HTTP requests and Cheerio to parse the HTML response.

The function follows the following steps:

1. It initializes the url2 variable and sets the baseUrl to "https://www.google.com".
2. It makes an HTTP GET request to the specified URL using Axios and sets the User-Agent header to mimic a web browser.
3. It loads the HTML response into the Cheerio library.
4. It extracts the URL of the first search result from the loaded HTML using the CSS selector ".EJbZzc".
5. If the extracted URL is "https://www.google.comundefined", it sets url2 to the original URL.
6. It initializes the mainData and content arrays to store the scraped data.
7. It makes another HTTP GET request to the extracted URL using Axios.
8. It loads the HTML response into Cheerio.

9. It extracts the image link, title, review, and review out of the product from the loaded HTML using CSS selectors.
10. It iterates over each offer row in the loaded HTML using the CSS selector ".sh-osd__offer-row".
11. For each offer row, it extracts the company brand, company link, delivery details, item price, and total price using CSS selectors.
12. It creates an object with the extracted data and adds it to the content array.
13. It creates an object with the main data, including the title, review, review out of, image link, and content array, and adds it to the mainData array.
14. It returns the mainData array containing all the scraped data.

3.2.4. Error Handling

The code includes error handling to catch any exceptions that may occur during the scraping process. If an error occurs, it will be logged to the console along with the message "Scrapper Error".

Please note that web scraping may be subject to legal restrictions and terms of service of the websites being scraped. Make sure to comply with all applicable laws and obtain proper authorization before scraping any website.

That's it! You now have a basic understanding of the web scraper code and how to use it to extract data from Google Shopping product pages. Happy scraping!

3.3. Authentication API Endpoint:

Description: The authentication API endpoint (https://deal-checker.vercel.app/api/users/*) provides JWT-based authentication for secure user authorization.

3.3.1. Why Authentication API Endpoint:

- Secure User Authorization: JWT-based authentication ensures secure and stateless user authentication, enhancing the overall security of the application.
- Seamless Integration: Integrates seamlessly with existing systems and frameworks, allowing for easy implementation and deployment.

3.3.2. Authentication Process:

The authentication process involves the following steps:

1. User Authentication: Users provide their credentials (e.g., username and password) to the authentication endpoint.
2. Token Generation: Upon successful authentication, the server generates a JWT containing user information and a digital signature.
3. Token Issuance: The JWT is issued to the client and stored securely, typically in local storage or a cookie.
4. Token Verification: For subsequent requests, clients include the JWT in the authorization header.
5. Access Control: The server verifies the JWT's signature and grants access to protected resources if the token is valid.

3.3.3. Token Expiration:

JWTs typically have an expiration time to mitigate the risk of unauthorized access. After the token expires, users must re-authenticate to obtain a new token.

3.3.4. Refresh Tokens:

To maintain user sessions without requiring frequent re-authentication, refresh tokens can be used. When a JWT expires, clients can use a refresh token to obtain a new JWT without re-entering credentials.

3.3.5. Implementation Details:

The authentication API endpoint utilizes libraries such as jsonwebtoken for JWT generation and verification, ensuring robust security measures are in place.

3.4. Favorite Product API Endpoint:

Description: The favorite product API endpoint (https://deal-checker.vercel.app/api/favorite/*) allows users to mark products as favorites for easy access and tracking.

3.4.1. Why Favourite Product API Endpoint:

- User Convenience: Enables users to save and retrieve favorite products, enhancing user experience and engagement.
- Personalization: Allows for personalized recommendations and targeted marketing based on user preferences.

3.4.2. Usage:

To mark a product as a favorite, users can send a POST request to the favorite product endpoint with the product ID or URL as a parameter. Similarly, users can retrieve their list of favorite products by sending a GET request to the endpoint.

3.4.3. Implementation:

The favorite product endpoint utilizes a database or storage mechanism to store user preferences and favorite products securely. User authentication is required to ensure data privacy and security.

CHAPTER 4

UI Pages

4.1 Main Page:

The main page serves as a visually engaging introduction, capturing the essence of the platform.

Users are greeted with enticing visuals and a clear call-to-action, encouraging them to explore the site.

4.1.1. Feature Explanation

This documentation provides an explanation of the code for the "Deal Checker" feature. The code is written in JavaScript and is used to create the homepage of the Deal Checker applicati on.

4.1.2. Importing Required Modules

The code begins with importing the necessary modules, `Image` and `Link`, from the `next/ima ge` and `next/link` libraries respectively. These modules are used to display images and creat e links in the application.

4.1.3. Home Component

The `Home` component is the main component of the homepage. It renders the content and layo ut of the Deal Checker application.

4.1.4. Homepage Layout

The homepage layout is defined using HTML-like JSX syntax. The layout is divided into different sections to display various elements such as navigation links, a header, a description, and buttons.

4.1.5. Navigation Links

The navigation links section contains a link to the source code of the application on GitHub. Clicking on the link will open the GitHub page in a new tab.

4.1.6. Header

The header section displays the title of the application, "Deal Checker". It also includes a "PRO" label to indicate the pro version of the application.

4.1.7. Description

The description section provides a brief description of the Deal Checker application. It explains that the application is an online shopping assistant that helps users find the best deals.

4.1.8. Buttons

The buttons section includes a "Get Started" button that redirects users to the search page of the application. Clicking on the button will navigate to the specified URL.

4.1.9. Additional Links

The additional links section contains two links: "Docs" and "API Example". Clicking on these links will open the specified URLs in new tabs. The "Docs" link provides in-depth information about the Deal Checker features and API, while the "API Example" link demonstrates how to use the API and mentions that the API returns data in JSON format.

This concludes the documentation for the code. The provided code creates the homepage layout for the Deal Checker application, including navigation links, a header, a description, and buttons. It also includes additional links to the documentation and an API example.

4.2 Search Page:

The search page is designed with user convenience in mind, featuring an intuitive search bar that triggers real-time results.

4.2.1. UI Elements

The component's UI consists of the following elements:

4.2.2. Heading and Subheading

The component displays a heading and a subheading at the top of the page. The heading highl ights the website's mission, while the subheading provides additional information about the website’s focus.

4.2.3. Search Input

The component includes a search input field where users can enter their search queries. As users’ type, the search state variable is updated accordingly. Users can also press the "Enter" key to submit their search.

4.2.4. Search Button

Next to the search input field, there is a search button. Clicking on this button triggers the han delSubmit function, which redirects the user to the search results page based on their search q uery.

4.2.5. Description

Below the search input field and button, there is a description section that provides an overvie w of the website's purpose and benefits. It emphasizes the website's ability to help users find t he best deals by comparing prices and reviews from various websites.

4.2.6. Divider

A horizontal divider is displayed after the description section to visually separate the content.

4.2.7. Additional Description

Following the divider, there is an additional description section that further explains the benef its of using Deal Checker for online shopping. It highlights the ability to filter search results a nd view ratings and reviews from other customers.

4.3 Item List Page:

The item list page strikes a balance between information and simplicity, presenting users with a concise overview of available products.

Each product card includes essential details, inviting users to explore further.

4.3.1 State Variables

The component uses the `useState` hook to define two state variables: `data` and `Loading`.

- `data` is an array that will store the fetched data from the API.
- `Loading` is a boolean variable that indicates whether the data is currently being loaded

4.3.2. useEffect Hook

The component uses the `useEffect` hook to fetch data from the API when the `search` param eter changes.

- The `useEffect` hook is called whenever the `search` parameter changes.
- Inside the `useEffect` hook, an asynchronous function is defined to fetch data from th e API using the `axios` library.
- The fetched data is then stored in the `data` state variable using the `setData` function.
- Finally, the `Loading` state variable is set to `false` to indicate that the data has finished loading.

4.3.3. Conditional Rendering

The component uses conditional rendering to display either a loading skeleton or the fetched data based on the value of the `Loading` state variable.

- If `Loading` is `true`, a loading skeleton component is rendered to indicate that the data is being loaded.
- If `Loading` is `false`, the fetched data is rendered using the `Card` component.

4.3.4. Rendering Individual Items

Inside the conditional rendering block, the fetched data is mapped over to render individual it ems using the `Card` component.

- The `map` function is used to iterate over the `data` array.

- For each item in the `data` array, an instance of the `Card` component is rendered with t he corresponding item data passed as props.

4.4 Full Details Page:

The full details page is the heart of the platform, offering a deep dive into a selected product. Price and feature comparisons are presented in a visually appealing format, aiding users in making informed choices.

4.5 About Page:

The "About" page is a personal touch, providing users with insights into the individuals behind the "Deal Checker" project.
It features information about you, the creator, showcasing your background, motivations, and the journey of developing the platform.

4.5.1 Creating the Profile Page

The profile page is created as a functional component named `page`. It returns JSX code that represents the structure and content of the profile page.

```
const page = () => {
  return (
    // JSX code representing the profile page
  );
};
```

4.5.2 Structure and Styling

The profile page is structured using HTML elements and CSS classes. The `className` attribute is used to apply the specified CSS classes.

```
// Content for the left side of the profile page

// Content for the right side of the profile page
```

The `flex` class is used to create a flex container, allowing the left and right sides of the profile page to be displayed side by side on larger screens (`md:flex-row`). On smaller screens, they

are displayed in a column (`flex-col`). The `items-center` and `justify-center` classes are used to center the content vertically and horizontally.

4.6. Authentication Page:

Description: The authentication page provides user interfaces for logging in, signing up, and recovering forgotten passwords.

4.6.1. Login Page:

- Description: The login page allows existing users to log in to their accounts.
- Features:
    - Username/Email and Password fields for authentication.
    - "Remember Me" option for persistent login sessions.
    - "Forgot Password?" link for password recovery.

4.6.2. Signup Page:

- Description: The signup page enables new users to create an account.
- Features:
    - Fields for entering username, email, password, and other required information.
    - Terms of Service and Privacy Policy acceptance checkboxes.

4.6.3. Forgot Password Page:

- Description: The forgot password page allows users to reset their forgotten passwords.
- Features:
    - Field for entering the registered email address.
    - "Reset Password" button to initiate the password reset process.
    - Instructions or link sent to the user's email for resetting the password.

4.6.4. Reset Password Page:

- Description: The reset password page enables users to set a new password after initiating a password reset.

- Features:
    - Fields for entering the new password and confirming it.
    - Validation checks for password strength.
    - "Submit" button to confirm the password reset.

4.7. Favorite Page:

Description: The favorite page displays a list of products marked as favorites by the user.

4.7.1. Features:

- Product Cards for displaying favorite products.
- Option to remove products from the favorites list.
- Call-to-action buttons for quick actions such as adding to cart or sharing favorites.
- Search bar for filtering or searching within the favorites list.
- Responsive design for optimal viewing across devices.

CHAPTER 5

System Design Overview

5.1. Entity-Relationship (ER) Diagram

Description: The Entity-Relationship (ER) Diagram offers a visual depiction of the system's data model, showcasing entities, attributes, and the relationships between them. Entities represent real-world objects or concepts, while attributes define their properties. Relationships illustrate connections between entities, reflecting associations or dependencies within the system's data structure.

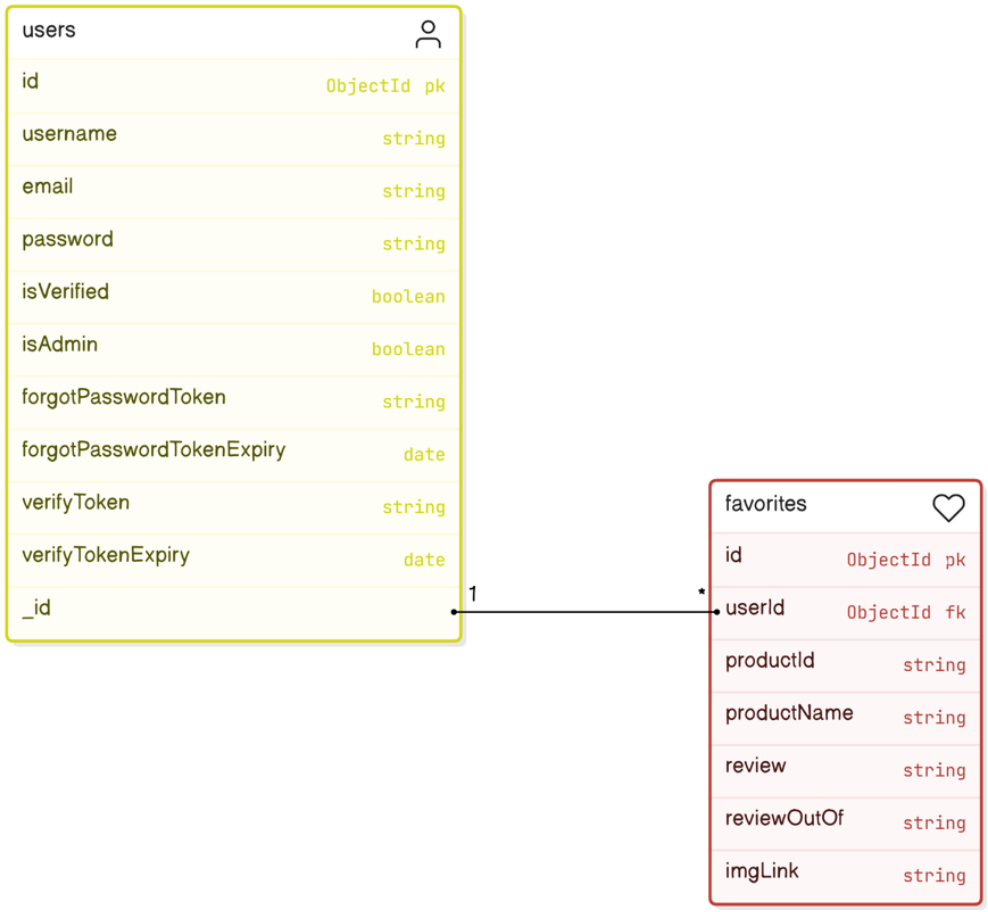

Figure – 5.1: ERD

5.2. High-Level Design (HLD)

Description: The High-Level Design (HLD) provides a conceptual overview of the system's architecture, emphasizing its structural components, interactions, and interfaces. It outlines the overall system structure, including layers, modules, and external dependencies. The HLD elucidates how various components collaborate to fulfill system functionalities without delving into implementation specifics, serving as a blueprint for system development and integration.

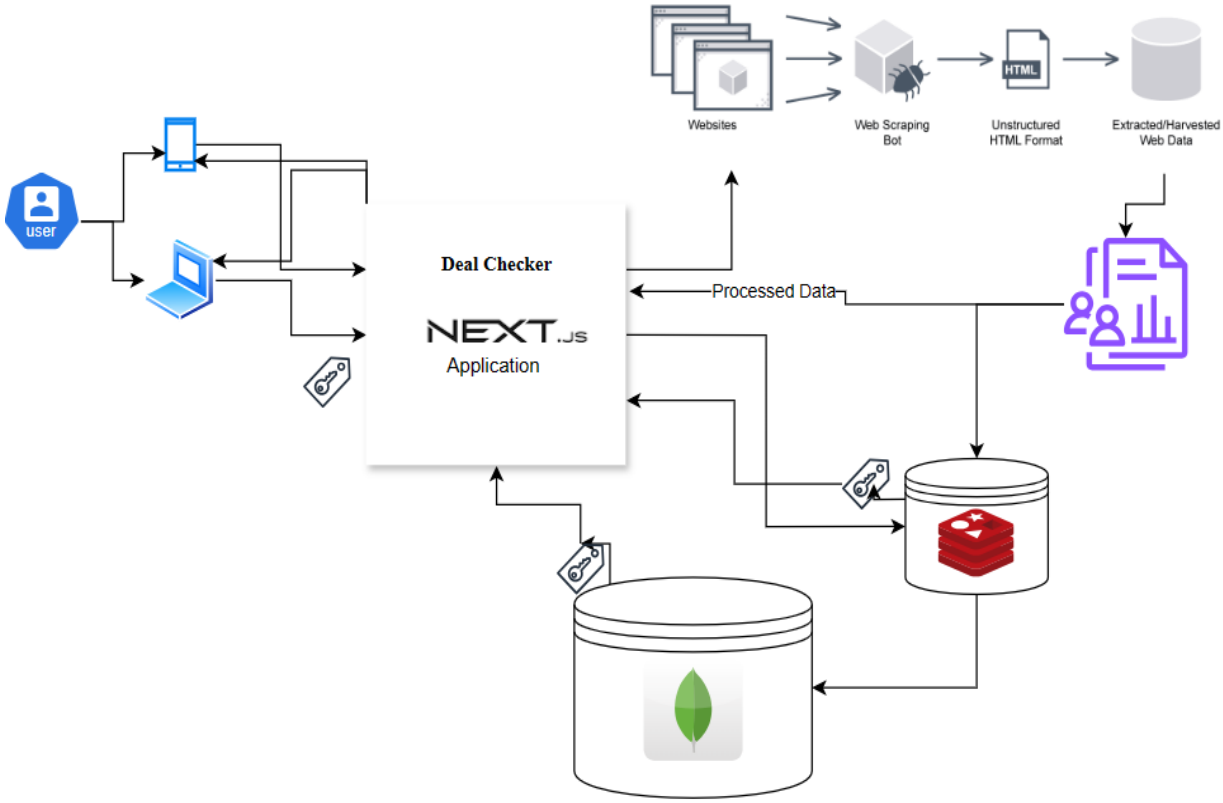

Figure – 5.2: HLD

CHAPTER 6

Streamlined Development

Deploying the Cn using Continuous Integration/Continuous Deployment (CI/CD) pipelines ensures efficient development processes and reliable deployment workflows. With GitHub Actions for CI/CD, the project undergoes automated linting checks, test case executions, and seamless deployment on Vercel (https://deal-checker.vercel.app/). Vercel's automatic deployment and CDN support further contribute to a seamless user experience, with minimal downtime and optimal performance.

6.1. Introduction to CI/CD

- Understanding the principles of Continuous Integration (CI) and Continuous Deployment (CD).
- Benefits of implementing CI/CD pipelines in software development projects.
- Overview of key components and stages in a typical CI/CD workflow.

6.2. Setting up Automated Workflows with GitHub Actions

- Introduction to GitHub Actions and its role in automating development workflows.
- Creating GitHub Actions workflow files to define CI/CD pipelines.
- Configuring workflow triggers and execution environments for different stages of the pipeline.

6.3. Ensuring Code Quality with Linting and Testing

- Implementing linting checks using tools like ESLint to enforce code style and standards.
- Integrating automated test suites (unit tests, integration tests, etc.) into the CI/CD pipeline.
- Analyzing test coverage metrics to assess the effectiveness of test suites.

6.4. Continuous Integration: Building and Integrating Code Changes

- Automatically triggering CI builds upon code commits or pull requests.
- Performing code quality checks, including static code analysis and dependency vulnerability scanning.

6.5. Continuous Deployment: Automating Deployment Processes

- Configuring automated deployment to staging or production environments.
- Monitoring deployment health and rollback mechanisms for handling deployment failures.

CHAPTER 7

Project Snapshots

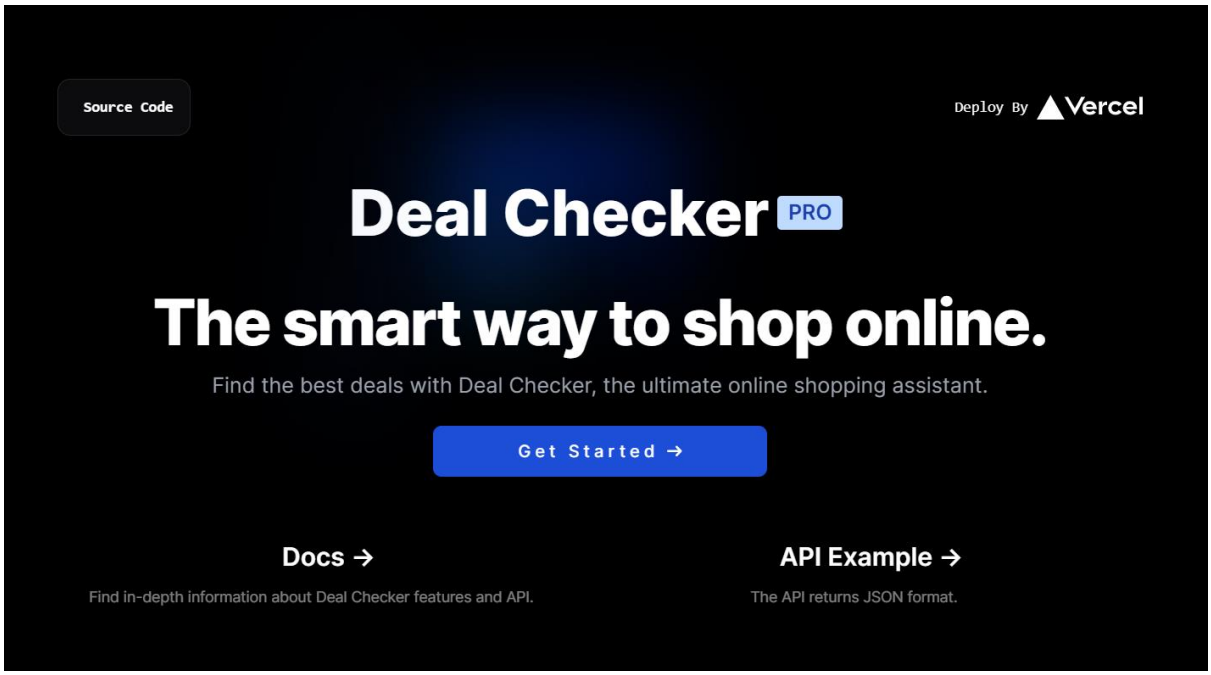

Figure – 7.1: Home Page

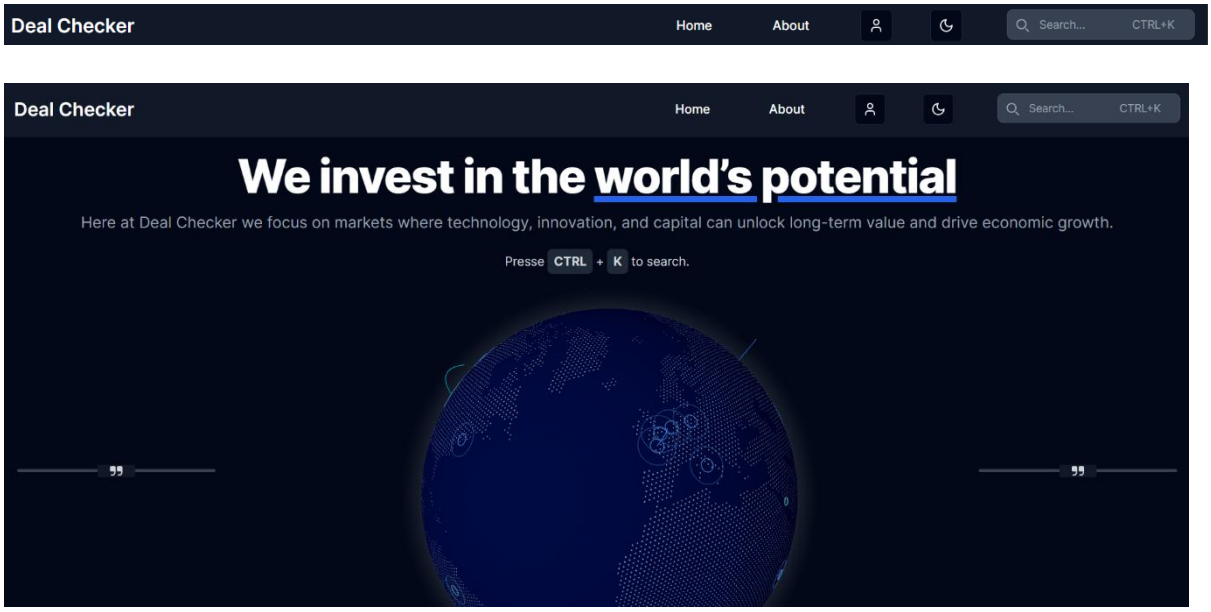

Figure – 7.2: Search Page

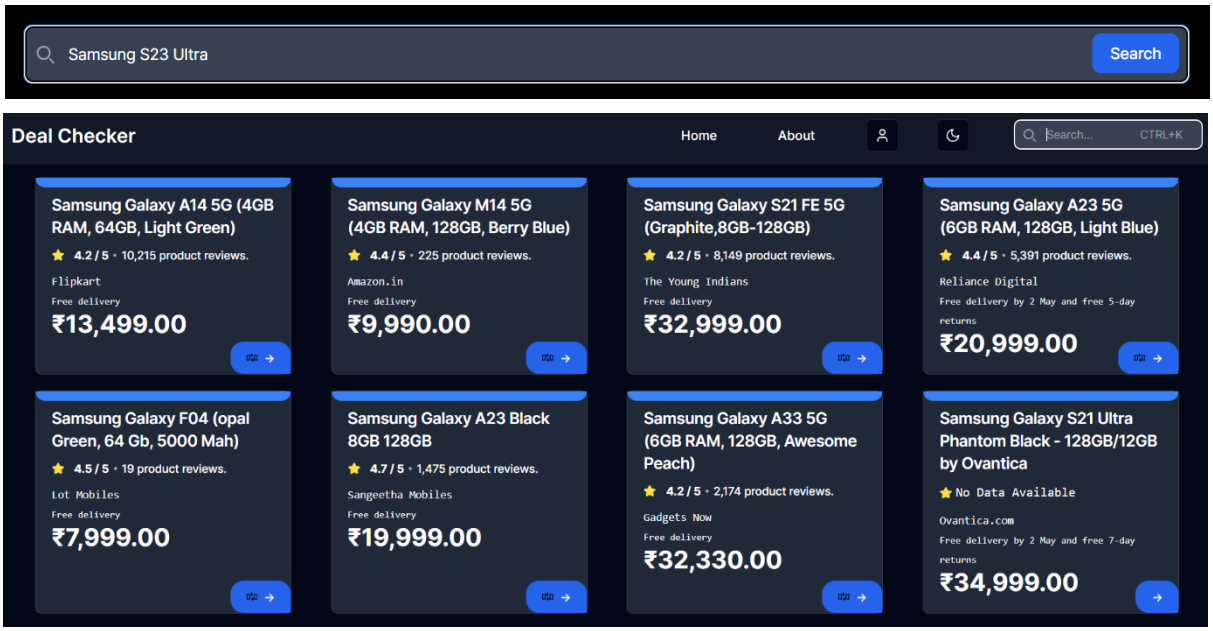

Figure – 7.3: Search Page

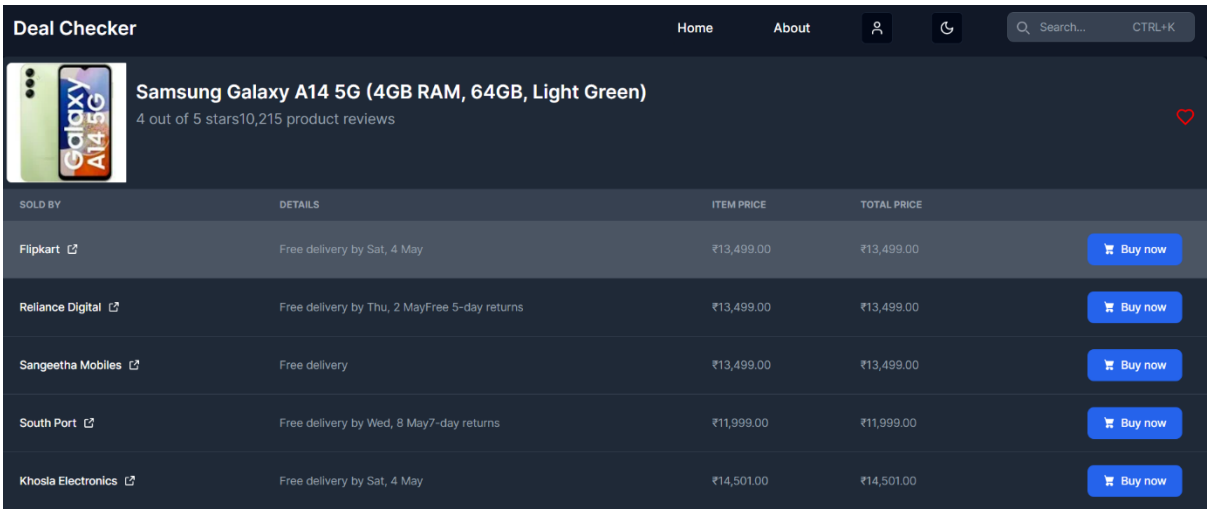

Figure – 7.4: Compare Page

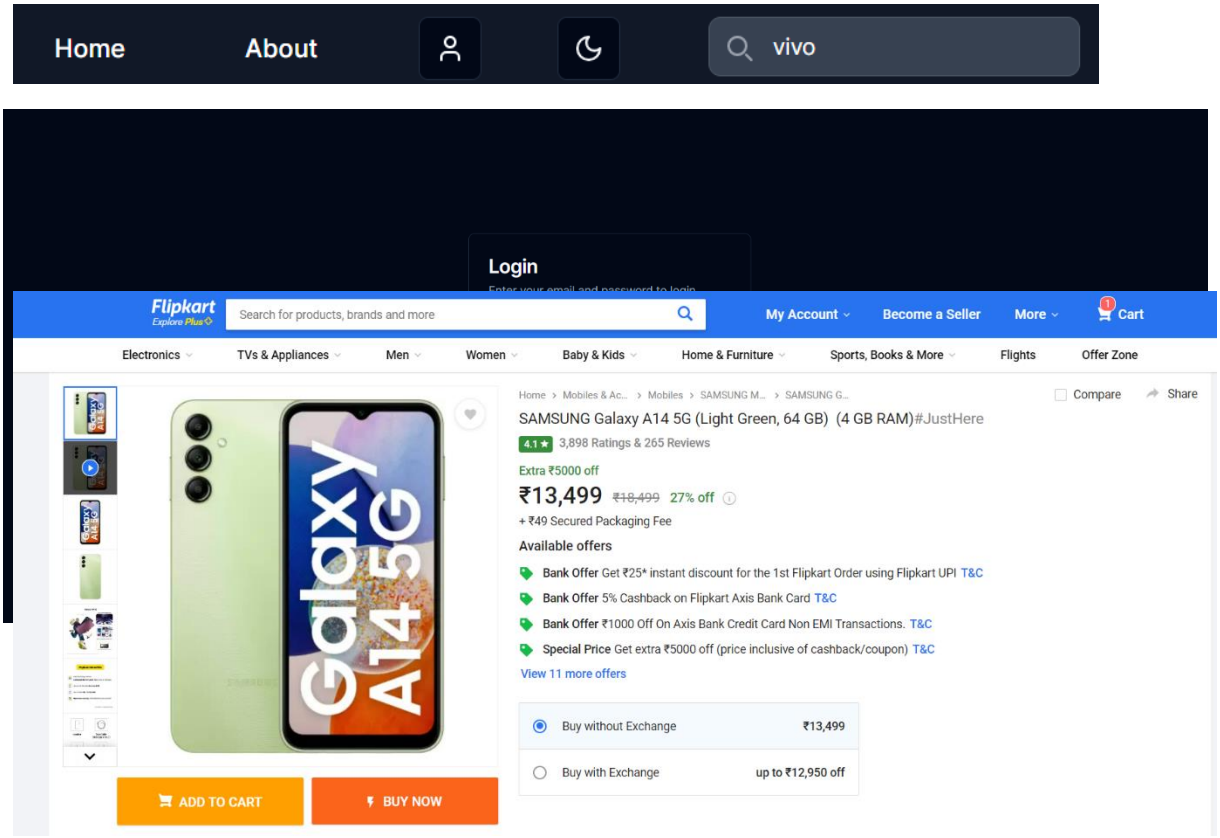

Figure – 7.5: Flipkart Product Page

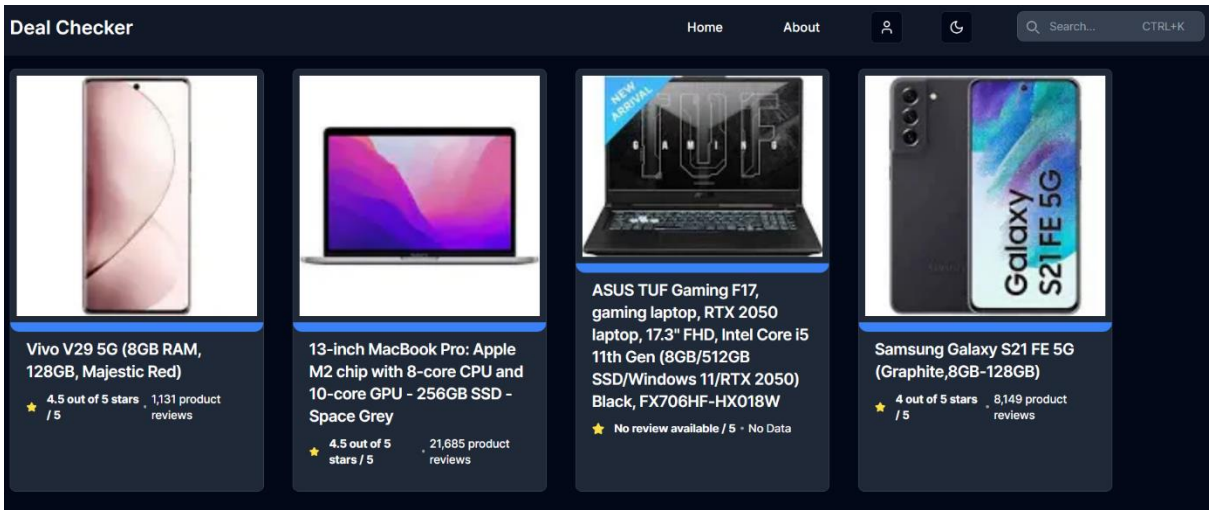

Figure – 7.6: Favourite Page

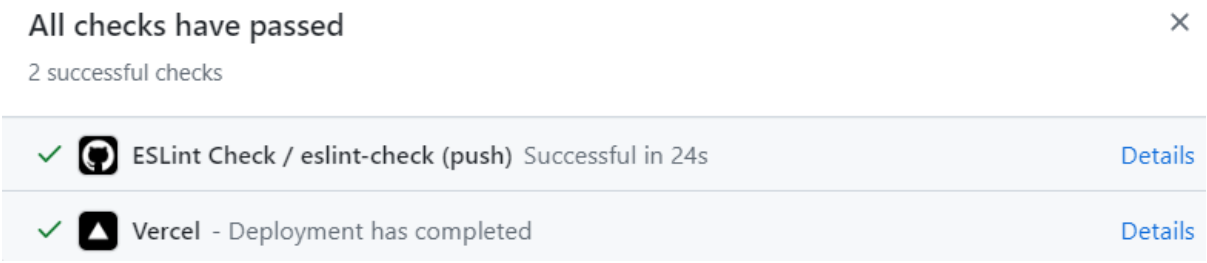

Figure – 7.7: CI/CD pipeline in github Action

Project Links:

GitHub Repository:

The source code and project files for "Deal Checker" can be found on the GitHub repository: Deal Checker GitHub Repository [https://github.com/chandanPradhan09/College-Project-Minor]

Live Deployment:

Explore the live deployment of "Deal Checker" and experience the platform in action: Deal Checker Live Deployment [https://deal-checker.vercel.app/]

CHAPTER 8

Conclusion

In conclusion, the "Deal Checker" project represents a cutting-edge and user-centric approach to online shopping, leveraging a carefully selected set of technologies to address the challenges users commonly encounter in the e-commerce landscape.

8.1. User-Centric Design:

The implementation of Next.js for dynamic UI ensures that users experience a seamless and responsive platform. The utilization of server-side rendering enhances the initial loading speed, while efficient client-side routing contributes to a smooth and enjoyable browsing experience. This user-centric design is further complemented by Tailwind CSS, which not only facilitates a stylish and visually appealing layout but also ensures consistency and ease of maintenance.

8.2. Efficient Data Retrieval and Parsing:

Axios, chosen for its simplicity and flexibility, facilitates efficient data retrieval from external APIs, ensuring that users have access to real-time and up-to-date product information. The integration of Cheerio for web scraping plays a pivotal role in extracting relevant data from HTML, enabling the platform to dynamically update product details and maintain accuracy.

8.3. Empowering Users through Comparison:

The implementation of dynamic and comparison API endpoints empowers users with the ability to make informed decisions. The dynamic API endpoint provides users with detailed and real-time product information, adapting to evolving market offerings.

In essence, "Deal Checker" is not merely a product comparison platform; it's a comprehensive solution designed to enhance the entire online shopping journey. By combining advanced technologies, a user-friendly interface, and a commitment to transparency, the platform stands as a testament to the continuous pursuit of excellence in the realm of digital commerce.

As the project evolves, there is a commitment to ongoing improvement and adaptation to user needs. The technologies selected and the thoughtful design choices underscore a dedication to providing users with a reliable, efficient, and enjoyable online shopping experience.

References

[1] Next.js Documentation:

The official documentation of Next.js served as a crucial resource in implementing dynamic UI components, server-side rendering, and client-side routing. Next.js Documentation

[2] Tailwind CSS Documentation:

The Tailwind CSS documentation provided extensive guidance on leveraging utility-first CSS for a consistent and visually appealing design. Tailwind CSS Documentation

[3] Cheerio Documentation:

The documentation for Cheerio was a valuable reference in implementing web scraping for efficient data extraction from HTML. Cheerio Documentation

[4] Axios Documentation:

The Axios documentation was consulted for best practices in making asynchronous HTTP requests, ensuring efficient data retrieval from external APIs. Axios Documentation

[5] Flowbit Documentation:

The Flowbit documentation provided insights into effective data parsing strategies, contributing to the overall efficiency of the web scraping process. Flowbit Documentation

[6] npm (Node Package Manager):

The npm website was used as a central hub for discovering, installing, and managing project dependencies, ensuring a streamlined development process. npm Documentation

[7] Vercel Documentation:

The Vercel documentation offered guidance on deploying and hosting the "Deal Checker" project, ensuring a reliable and scalable platform. Vercel Documentation

These resources played a vital role in the successful development, deployment, and maintenance of the "Deal Checker" project, contributing to its overall functionality, design, and user experience.