**Requirement Gathering and Analysis Phase**

**Technology Stack (Architecture & Stack)**

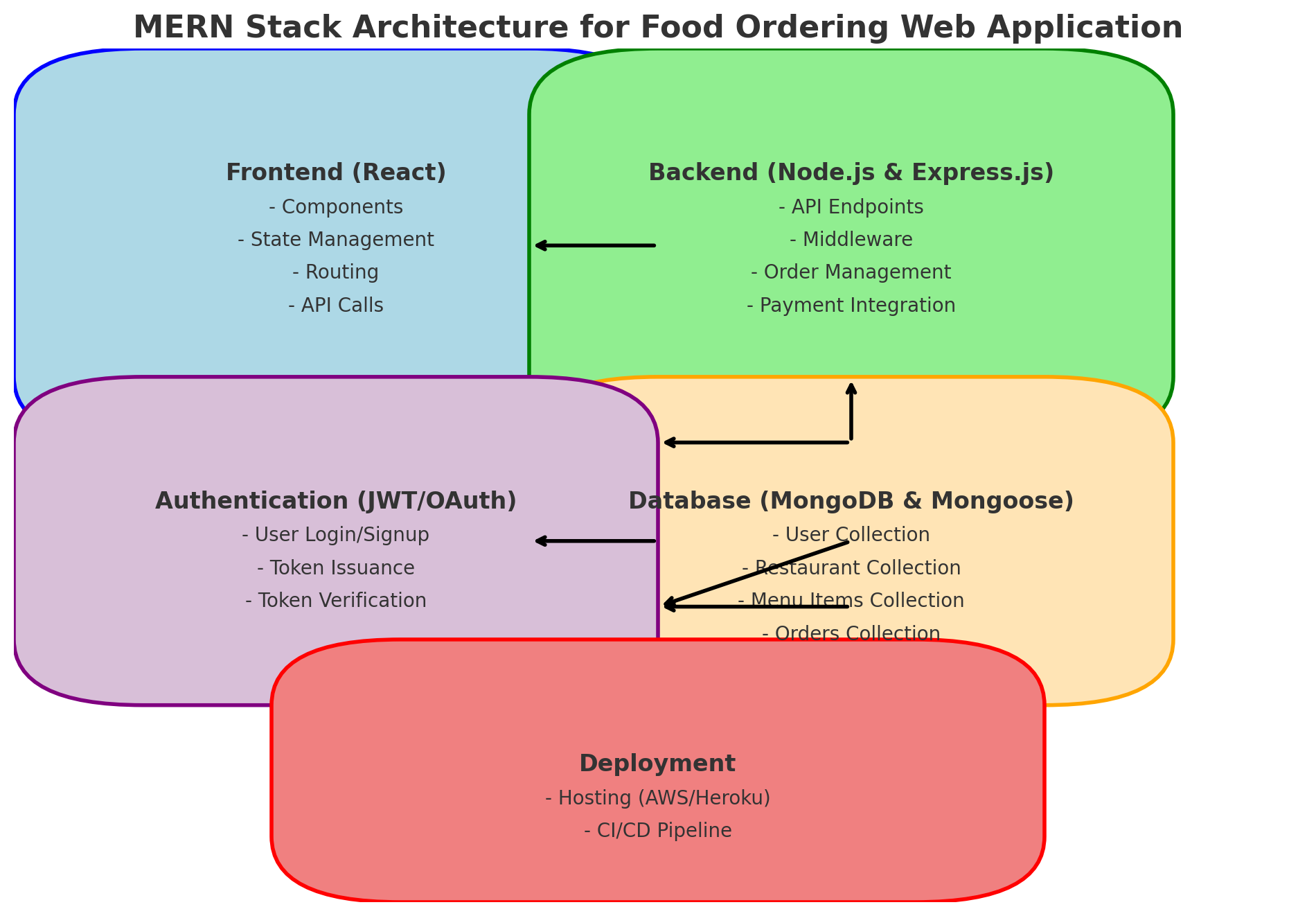
|  |  |
| --- | --- |
| Date | 06-07-2024 |
| Team ID | SWTID1720074953 |
| Project Name | SB Foods-Food Ordering app |
| Maximum Marks |  |

**Technical Architecture:**

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2

**Example: Order processing during pandemics for offline mode**

**Reference:** [**https://developer.ibm.com/patterns/ai-powered-backend-system-for-order-processing-during-pandemics/**](https://developer.ibm.com/patterns/ai-powered-backend-system-for-order-processing-during-pandemics/)

****



**Table-1 : Components & Technologies:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Component** | **Description** | **Technology** |
|  | Food Delivery Application | The food delivery application allows users to browse through a variety of food items, add them to their cart, place orders, and track them. Users can register and log in to manage their profiles, view order history, and update delivery information. The application supports both Cash on Delivery (COD) and online payment via Stripe. It includes features for adding items to the cart, adjusting quantities, and processing orders securely. |  **Frontend:**   * **React:** Used for building the user interface components, managing state, and handling frontend logic. * **React Router:** For routing and navigation within the application. * **React Context API:** Used to manage global state such as user authentication, cart items, and order details across components. * **Axios:** For making HTTP requests to the backend APIs.    **Backend:**   * **Node.js:** Backend runtime environment for executing JavaScript code. * **Express.js:** Framework for building RESTful APIs and handling HTTP requests. * **MongoDB:** NoSQL database used for storing user data, food items, orders, and cart information. * **Mongoose:** MongoDB object modeling tool used to model application data and simplify interactions with MongoDB. * **jsonwebtoken (JWT):** Used for user authentication and authorization. * **bcryptjs:** For hashing passwords before storing them in the database. |
|  | FoodDisplay | The **FoodDisplay** component is a React functional component designed to display a list of food items based on a selected category. It uses context to fetch the list of food items from a global state and filters these items based on the provided category. If the category is set to "All", it displays all items. Otherwise, it displays only the items that match the selected category. |  **React**: A JavaScript library for building user interfaces, used to create and manage the component structure.   **Context API**: A React feature used to manage global state and share it across the component tree without passing props manually at every level.   **CSS**: Styling is applied through the FoodDisplay.css file to ensure the component is visually appealing.   **JavaScript (ES6)**: Utilized for functional programming, destructuring, and array methods like map |
|  | FoodItem | The **FoodItem** component is a React functional component designed to display individual food items with details such as the image, name, price, and description. It integrates with the context to manage cart operations, allowing users to add or remove items from their cart. The component also displays the current count of each item in the cart. |  **React**: A JavaScript library for building user interfaces, used to create and manage the component structure.   **Context API**: A React feature used to manage global state and share it across the component tree without passing props manually at every level.   **CSS**: Styling is applied through the FoodItem.css file to ensure the component is visually appealing.   **JavaScript (ES6)**: Utilized for functional programming, use of hooks (useState and useContext), and array methods. |
|  | ExploreMenu | The **ExploreMenu** component is a React functional component designed to display a list of menu categories that users can explore. It allows users to select a category, which is then highlighted, and toggles between the selected category and showing all categories. This component uses context to fetch the list of menu categories and utilizes local state to manage the selected category. | * **React**: A JavaScript library for building user interfaces, used to create and manage the component structure. * **Context API**: A React feature used to manage global state and share it across the component tree without passing props manually at every level. * **CSS**: Styling is applied through the ExploreMenu.css file to ensure the component is visually appealing. * **JavaScript (ES6)**: Utilized for functional programming, destructuring, and array methods like map. |
|  | Footer | The **Footer** component is a React functional component designed to display the footer section of a website. It includes the company logo and description, social media icons, company links, and contact information. This component provides a structured and visually appealing way to present important information and links typically found at the bottom of a webpage. |  **React**: A JavaScript library for building user interfaces, used to create and manage the component structure.   **CSS**: Styling is applied through the Footer.css file to ensure the component is visually appealing.   **JavaScript (ES6)**: Utilized for functional programming and managing component structure |
|  | Header | The **Header** component is a React functional component designed to display the header section of a website. It includes a heading, a descriptive paragraph, and a button that prompts users to view the menu. This component is intended to be visually appealing and provide users with a quick introduction to the website's purpose. |  **React**: A JavaScript library for building user interfaces, used to create and manage the component structure.   **CSS**: Styling is applied through the Header.css file to ensure the component is visually appealing.   **JavaScript (ES6)**: Utilized for functional programming and managing component structure. |
|  | LoginPopup | The **LoginPopup** component is a React functional component that provides a popup form for user authentication, allowing users to either log in or sign up. It manages user input, handles form submissions, and integrates with backend APIs for user authentication. The component displays different input fields and buttons based on whether the user is logging in or signing up. It also includes client-side validation and error handling using toast notifications. |  **React**: For creating the component and managing state and context.   **CSS**: Styling is applied through the LoginPopup.css file to ensure the component is visually appealing.   **JavaScript (ES6)**: Used for functional programming, managing state with hooks (useState, useContext), and handling events.   **Axios**: For making HTTP requests to the backend API.   **React Toastify**: For displaying toast notifications for error messages.   **Node.js and Express**: The backend server to handle user authentication requests.   **MongoDB**: For storing user data such as user credentials. |
|  | Navbar | The **Navbar** component is a navigation bar for the "Sip and Play" food delivery app, providing links to various sections of the website and user-specific actions such as login/logout and accessing the cart. It includes navigation links to different sections of the site, a search icon, and a user profile section that conditionally renders login or profile options based on the user's authentication status. | IBM Block Storage or Other Storage Service or Local Filesystem |
|  | StoreContextProvider | The **StoreContextProvider** component is a context provider for managing global state related to the "Sip and Play" food delivery app. It includes state management for the food list, cart items, user authentication token, and several utility functions to handle adding/removing items from the cart, fetching data, and calculating total cart amounts. |  **React**: For creating the component and managing state and context.   **React Context API**: To create and provide a global state context.   **Axios**: For making HTTP requests to the backend server.   **JavaScript (ES6)**: Used for functional programming, managing state with hooks (useState, useEffect), and handling events.   **Node.js and Express**: The backend server to handle API requests (not shown in the code but implied).   **MongoDB**: For storing and retrieving food items and cart data (not shown in the code but implied). |
|  | App | The App component serves as the root component for the "Sip and Play" food delivery application. It sets up the routing for different pages, handles the login popup state, and includes the navigation bar and footer. |  **React**: For creating the component and managing state.   **React Router DOM**: For handling client-side routing.   **React Toastify**: For displaying toast notifications.   **JavaScript (ES6)**: Used for functional programming and managing state with hooks (useState). |
|  | Cart | The **Cart** component displays the items added to the cart, provides a summary of the cart totals including the delivery fee, and allows the user to proceed to checkout or enter a promo code. It uses context to manage state and interacts with the store context to fetch cart details and perform actions like removing items from the cart. |  **React**: For creating the component and managing state.   **React Router DOM**: For navigating to different routes.   **Context API**: For accessing the store context (StoreContext) to manage the cart state.   **JavaScript (ES6)**: Used for functional programming and managing state with hooks (useContext). |
|  | Home | The Home component serves as the main landing page for the "Sip and Play" food delivery app. It integrates several components to provide a comprehensive user experience. |  **React**: For building the component-based UI.   **useState**: For managing local state within the component. |
|  | MyOrders | The **MyOrders** component fetches and displays a list of orders associated with the logged-in user. Each order is presented with details such as the ordered items, total amount, number of items, order status, and an option to track the order. |  **React**: For building the component-based UI.   **useState**: For managing local state within the component.   **useEffect**: For fetching data asynchronously after the component mounts.   **axios**: For making HTTP requests to the backend API.   **Context API**: Utilized through useContext to access global state and functions from StoreContext. |
|  | PlaceOrder | The **PlaceOrder** component allows users to provide delivery information, choose a payment method (Cash on Delivery or Stripe), and finalize their order. It interacts with the backend API to handle order placement and payment processing. |  **React**: For building the component-based UI.   **useState**: For managing local state within the component.   **useEffect**: For handling side effects like redirecting users based on conditions.   **axios**: For making HTTP requests to the backend API.   **react-router-dom**: For navigation within the application.   **react-toastify**: For displaying toast notifications for user feedback.   **Context API**: Utilized through useContext to access global state and functions from StoreContext.   **assets**: Imports images/icons required for UI elements. |
|  | Verify | The Verify component is used to verify the payment status after a user completes a payment and is redirected back to the application. It extracts parameters (success and orderId) from the URL query string using useSearchParams and sends a POST request to the backend (/api/order/verify). Depending on the verification result (response.data.success), it navigates the user to either the MyOrders page or the home page (/). |  **React**: For building the component-based UI.   **useEffect**: For executing side effects like fetching data.   **axios**: For making HTTP requests to the backend API.   **react-router-dom**: For navigation within the application.   **useSearchParams**: Hook for accessing and manipulating URL query parameters.   **Context API**: Utilized through useContext to access global state (StoreContext) |

**Table-2: Application Characteristics:**

| **S.No** | **Characteristics** | **Description** | **Technology** |
| --- | --- | --- | --- |
|  | Open-Source Frameworks | Utilization of open-source frameworks | React, Node.js, Express.js, MongoDB, Mongoose |
|  | Security Implementations | Implementation of security measures | * JWT (JSON Web Tokens), bcryptjs for password hashing * HTTPS for secure communication |
|  | Scalable Architecture | Architecture designed for scalability | Microservices architecture  Horizontal scaling with Node.js clusters |
|  | Availability | Ensuring high availability | Load balancing using NGINX, clustering with Node.js  Distributed MongoDB for data redundancy and availability |
|  | Performance | Considerations for performance optimization | Use of caching (Redis)  CDN (Content Delivery Network) for static assets  Optimization of API responses |

**References:**

[**https://c4model.com/**](https://c4model.com/)

[**https://developer.ibm.com/patterns/online-order-processing-system-during-pandemic/**](https://developer.ibm.com/patterns/online-order-processing-system-during-pandemic/)

[**https://www.ibm.com/cloud/architecture**](https://www.ibm.com/cloud/architecture)

[**https://aws.amazon.com/architecture**](https://aws.amazon.com/architecture)

[**https://medium.com/the-internal-startup/how-to-draw-useful-technical-architecture-diagrams-2d20c9fda90d**](https://medium.com/the-internal-startup/how-to-draw-useful-technical-architecture-diagrams-2d20c9fda90d)