Technical report

Steps taken to build and integrate components.

Planning and Design:

For this project, I started by defining the components I would need, based on the features I wanted to implement. I identified reusable components such as the header, footer, contact form, and buttons. I chose **React** for building the UI components and **Next.js** for creating the pages and handling server-side rendering (SSR). Tailwind CSS was chosen for styling due to its flexibility and ease of creating responsive designs.

Component Development:

I created each component as a separate functional component in React, starting with the most fundamental elements like the header and footer. I used **React hooks**, particularly use State and use Effect, for managing state and handling side effects. For instance, in the contact form, I implemented a use State hook to manage form data and form submission handling.

Integration:

To integrate the components, I used **Next.js**'s page system to render components within specific routes (e.g., contact form on /contact). The header, footer, and other common components were added across multiple pages to ensure consistent navigation. State was passed from parent to child components using props, and where necessary, I utilized **React Context API** to share state globally (e.g., for theme switching or user authentication state).

Testing:

I used **React Testing Library** to test individual components. For instance, I tested if the form inputs captured the correct values and if the submit button triggered the appropriate actions. I also tested the layout responsiveness using **Jest** for unit tests and manual checks for responsive design across devices.

Challenges Faced and Solution Implemented

Challenge 1: Difficulty with Layout Shifting on Refresh

The layout was shifting on page refresh, which was due to CSS rendering delays. I solved this by using **CSS Grid** and **Flexbox** for more predictable layouts and ensuring that the layout structure was properly loaded before rendering. Additionally, I used position: relative and min-height for critical sections to maintain consistent spacing.

Challenge 2: State Management Across Components

Initially, I encountered difficulties in managing the application state across various components, especially when dealing with form data and user preferences. To resolve this, I implemented the **React Context API** for global state management, which helped avoid prop drilling and made state accessible across different components without redundancy.

Challenge 3: Handling Form Validation and Async Operations

Asynchronous operations like form submissions were initially problematic, especially with error handling and loading states. I handled these by using **async/await** syntax within event handlers. I also implemented a loading state to display a spinner while the data was being processed, ensuring a smooth user experience.

Challenge 4: Responsive Design Issues

Ensuring the page looked good across different devices was a challenge. I solved this by using **Tailwind CSS**’s responsive utilities to manage breakpoints for different screen sizes. I also manually tested the responsiveness on various devices like the iPhone SE and Samsung Galaxy S8 to ensure it rendered as expected.

Best Practices Followed During Development

Code Organization:

The project was structured with a clear separation of concerns, with each component placed in its own folder inside the `components` directory. Pages were organized in the `pages` folder for easy navigation and routing. I kept the file and folder names descriptive to make it easier for other developers (or future me) to navigate the project.

Reusability:

I focused on creating reusable components. For example, the button component was reused across multiple pages, and the form fields were created as a generic input component that could be customized with props (e.g., type, placeholder, label).

State Management:

I used **React Context API** for global state management, particularly for maintaining the user's theme preferences and form data. This approach helped keep the state logic separate from the UI components, making the code more maintainable and scalable.

Error Handling:

For error handling, I used **try-catch** blocks within asynchronous functions to catch and manage errors, especially when submitting forms or making API calls. I displayed user-friendly error messages in the UI if something went wrong, and I added form validation to ensure correct data was entered.

Version Control:

Throughout the development process, I used **Git** for version control. Each feature or bug fix was tracked with a clear commit message, and I followed a **feature branching** workflow to keep the main branch clean. I also created pull requests to merge new features into the main branch after thorough code reviews.

Summary:

This report outlines the steps taken to build and integrate components in my React and Next.js project, highlighting the challenges faced and solutions implemented. I started by designing and developing reusable components, integrating them through Next.js pages, and testing for functionality and responsiveness. Key challenges included managing state across components, handling asynchronous operations, and ensuring responsive design. Solutions such as using the React Context API for global state management, async/await for handling async tasks, and Tailwind CSS for responsiveness were applied. Best practices like code organization, error handling, version control, and accessibility were followed throughout the development process.

Student: ***Yusra Fatima***
Roll no : ***00411279***
Slot: ***Saturday Afternoon***