CIT 313 Final Project

Development

- Page Routing - I set up page routing in my React app using **React Router**. I wrapped the app in a <BrowserRouter> and defined routes using <Routes> and <Route>. I created a Layout component with a shared navigation bar that dynamically generates links using a navItems array. The <Outlet> component in Layout renders child routes like Home, About, Cars, and Contact, each linked to its own path. This setup ensures smooth navigation and a consistent layout across the app.

- Prop Use - I used props to pass data into my components for dynamic functionality. For example, I passed a navItems array to the NavBar component in the Layout to create navigation links dynamically.
  I also created separate JSON data files (CarData.js) to store information like car details. These files are imported into pages like CarCard.js and passed as props to child components. This approach keeps my components reusable, simplifies data management, and ensures my application remains organized and scalable.

- Hooks – I implemented React hooks, specifically useState, to manage the state of my components effectively. In the Accordion component, I used useState to toggle the visibility of the content by tracking whether the accordion is open or closed. This state determines the dynamic styling and visibility of the content section. In the CarCard component, I used useState to manage the selectedCar state, which stores the details of the currently selected car. This state is updated when a user clicks the "View Price" button, triggering a modal to display the car's price and image. Clicking outside the modal or the close button resets the state, hiding the modal. These hooks ensure seamless interaction and dynamic updates across the app.

- Hamburger Menu - I implemented a hamburger menu using the useState hook to toggle its visibility. The isOpen state tracks whether the menu is open (true) or hidden (false). Clicking the hamburger button toggles this state, and conditional CSS classes (block for visible, hidden for hidden) dynamically control the menu's display. This makes the menu responsive, showing a full layout on larger screens and a collapsible hamburger menu on smaller screens.

- Components - In my exotic car rental app, I used modular components to create a seamless and dynamic user experience. The NavBar provides navigation with responsive links, while the Header sets the tone with a visually engaging introduction. For browsing cars, CarsList organizes a collection of CarCard components, each displaying details like make, year, and horsepower using data from CarsData. The ShowRoom highlights featured cars, and Carousel, powered by CarouselData, adds a scrolling gallery for promotions or top picks. To enhance user interaction, the Accordion component, populated by AccordionData, displays collapsible sections like FAQs, and ContactForm enables inquiries or bookings. The Footer, built dynamically with FooterData, offers quick links and contact information, while TextArea handles longer user inputs. Each component contributes to the app's functionality, making it organized, responsive, and tailored to the needs of an exotic car rental business.

- Styling - I styled my components using a combination of Tailwind CSS and custom CSS to create a clean, responsive design. Tailwind's utility-first classes allowed me to quickly apply styles like shadow-lg, rounded-xl, and flex for layout and spacing, while hover effects like hover:bg-gray-100 added interactivity. I complemented this with custom CSS for specific elements, such as .car-card-container for grid layouts, .logoImg for branding consistency, and .modal-content for modals. Media queries ensured responsiveness, adjusting elements like header heights and image scales for smaller screens. This blend of Tailwind and custom styles resulted in a cohesive, user-friendly interface for my exotic car rental app.

- Difficulties - While building my exotic car rental app, I faced challenges with Tailwind, page routing, and responsiveness. Some Tailwind classes, like specific hover effects or spacing utilities, didn't apply as expected. To resolve this, I debugged the issue by checking Tailwind's documentation and ensuring proper class usage. For persistent issues, I used regular HTML and custom CSS to manually style components, ensuring the design matched my vision.

  Page routing also caused issues, such as routes not rendering correctly or components not displaying as intended. I resolved these by carefully reviewing my BrowserRouter and Route configurations, ensuring paths and nested routes were properly defined. Testing each route after adjustments helped me identify and fix errors quickly.

  For responsiveness, elements like car cards and navigation didn't scale well on smaller screens. I used Tailwind's responsive utilities but had to supplement them with custom media queries for more precise control. Testing on various devices and screen sizes allowed me to fine-tune the design, ensuring a seamless user experience across platforms. These challenges helped me better understand debugging and tailoring tools to meet specific project needs.