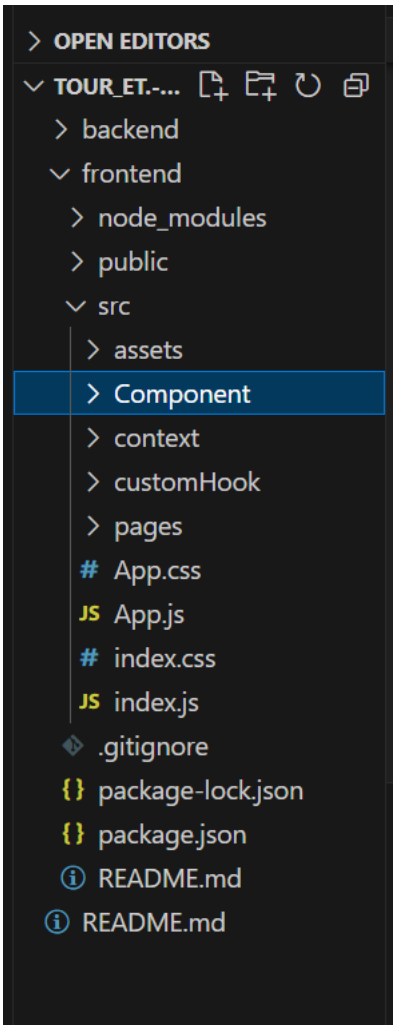
DOCUMENTATION FOR TOURISM WEBSITE

Frontend Part :

For front-end part we've here used is react javascript framework and created client side application using command **npx create-react-app frontend .** This command creates a frontend using react.

The components used here are :



Front-end side components uses :

- Components : Inside components we've subcomponents that are :
  - **Account** consisting of login.jsx and register,jsx
  - **Book** consisting of hotel.jsx and rooms.jsx
  - **Static** consisting of banner.jsx, footer.jsx, header.jsx, not found.jsx

  - Context : Inside context file we have AuthContext.js, cartContext.js, context.js

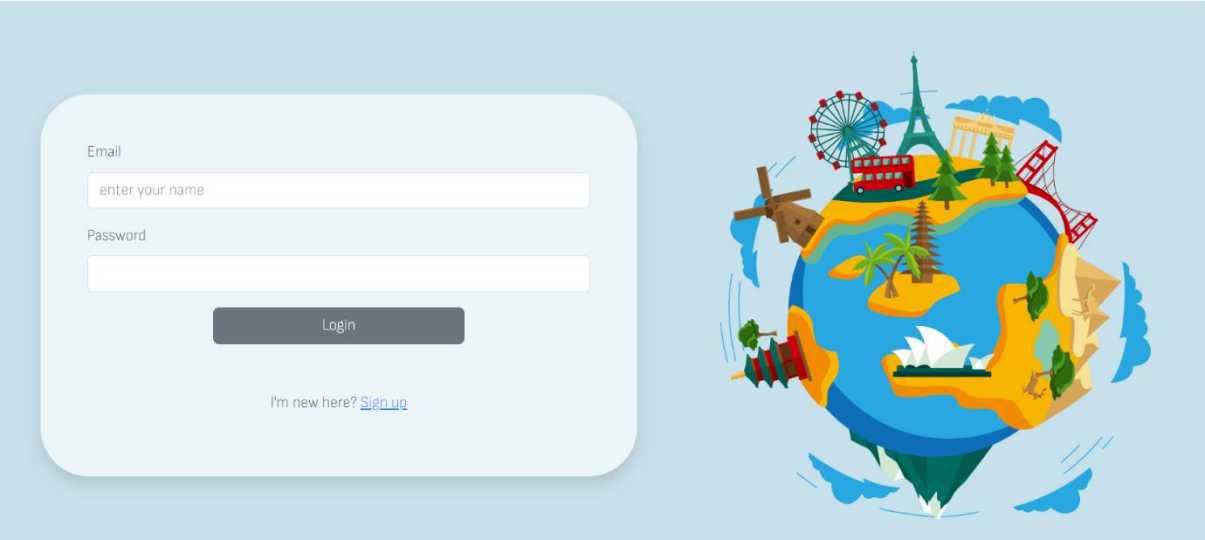
  - CustomHook : Inside this we have useAuthContext.js, useFetch.js
  - Pages : Inside this we have several files as About.jsx, book.jsx, cart.jsx,contact.jsx,home.jsx,home.jsx,packagedetail.jsx and reviewpage.jsx

EXPLAINATION OF COMPONENTS :

This is a React component called CustomButton, which functions as a custom navigation bar with buttons for logging in, becoming a seller, and seeing the cart.

- Library Imports: Imports React hooks (useState and useContext) and components/icons from Material-UI for state management and UI elements.
- Styling with Emotion: Utilizes the styled function from @emotion/styled to create styled components (Wrapper, container, and LoginButton) with custom styling.
- Context and Component Integration: Obtains a data context from '../../context/DataProvider', includes the LoginDialog component, and defines the primary functional component (CustomButton). This suggests a comprehensive integration of state management, UI styling, and potential user authentication feature.

- **LOGIN**



- User Login Component: The code defines a React component named Login, which serves as the login page for the application.
- Form Handling: It uses the useState hook to manage form data (fullInfo), captures user input through form elements, and updates the state accordingly.
- Authentication and Redirect: Upon form submission, it sends a POST request to an authentication API endpoint. If successful, it stores user data in local storage, dispatches a login action using a custom authentication context hook (useAuthContext), and redirects the user based on their role (admin or regular user).
- Styling and UI: The component includes a styled form with email and password inputs, error handling for invalid submissions, and a link to the registration page. Additionally, there's an image displayed on larger screens for visual appeal.

- **UTILS**

    - This JavaScript module includes functions names isDate, isObj, stringifyValue, and buildForm. isDate determines whether the provided value val is a JavaScript Date object.
    - Returns true if val is a Date object, and false otherwise.
    - isObj determines whether the specified value val is an object.
    - Returns true if val is an object; otherwise, false.

- stringify takes a value (val) and changes it into a string. If val is an object (except Date objects), the conversion is performed using JSON.stringify.
- returns the string representation of val.
- buildForm It generates a new HTML form dynamically.
- Sets the form's method to 'post' and the action to the value specified in the action argument.
- The params object generates hidden input fields for each key-value combination.
- It turns the values to strings using stringifyValue.
- Returns the built form.
- It accepts an object's information, including attributes, actions, and parameters.
- The buildForm function generates a form with the supplied action and parameters.
- Adds the form to the document body.
- Submit the form.
- Removes the form from the document body once it has been submitted.

APP.JS

- Imports: The code imports various components, pages, and libraries for a React application, including routing components from react-router-dom and custom components like Login, Register, Header, etc.
- Router Setup: It utilizes BrowserRouter to enable client-side routing and contains a <Switch> component to render specific components based on the matched route.
- Route Definitions: Different routes are defined using <Route>, specifying paths and corresponding components to render. The exact prop ensures that only exact matches trigger component rendering.
- Nested Structure: The main route ("/") has a nested <Switch> to handle routes specific to the content section between the <Header> and <Footer>, providing a structured layout for the React application.

INDEX.JS

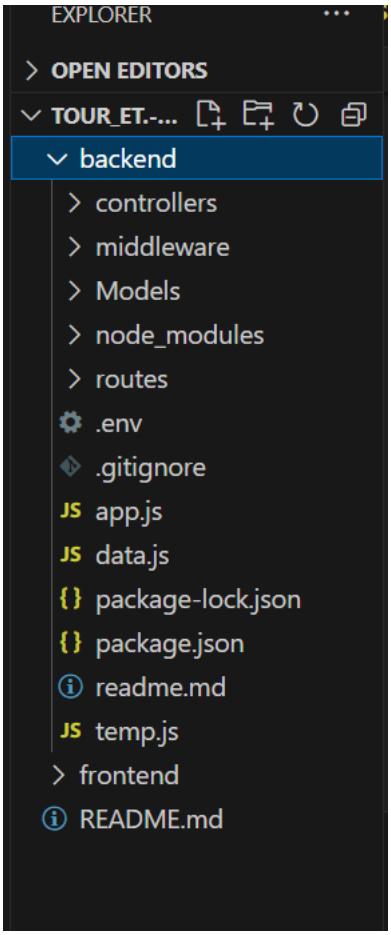
- Library Imports: Imports React and ReactDOM from react and react-dom/client, respectively.

- Global Styling: Imports a global stylesheet (index.css) for consistent styling throughout the application.
- Root Element Setup: Creates a root element using ReactDOM.createRoot to specify where the React app will be mounted in the DOM.
- Context Providers: Wraps the main <App /> component with context providers (AuthContextProvider and CartContextProvider) for managing shared state.
- Strict Mode Rendering: Renders the application within React.StrictMode to catch and highlight potential issues during development

Back-end Part :

For back-end part we've here created a backend folder inside the main tour-et folder where we've created **app.js, data.js, temp.js** file. Also installed nodemon using **npm install nodemon.**

The components used here are :



Back-end side components uses :

- Controllers : Inside components there are 7 files named bookingcontroller.js , commentcontroller.js , hostelcontroller.js, packagecontroller.js, roomcontroller.js and usercontroller.js and wishlistcontroller.js
- Database : Inside app.js file we have database
- model : Inside this folder we've bookingmodel.js,commentmodel.js,hotelmodel.js,packagemodel.js,roommodel.js,usermodel.js and wishlistmodel.js

- routes : Inside this we have bookingrouter.js, commentRouter.js, hotelRouter.js, packageRouter.js, roomRouter.js, userRouter.js and wishlistRouter.js

- middleware : Inside this we have auth.js

- **App.js**

- This code sets up a Node.js server using Express, creating endpoints for various routes such as packages, hotels, users, comments, bookings, rooms, and wishlists.

- It establishes a connection to a MongoDB database using Mongoose, with specified connection options and error handling.

- The server is started to listen on the specified port from the environment variables, and console logs indicate successful database connection and server start.

- **DATABASE**
  - Import mongoose from'mongoose';

```
mongoose.set("strictQuery", false);
async function connectToDb() {
  try {
```

```
      await mongoose.connect(process.env.MONGODBURL, {
        useNewUrlParser: true,
        useUnifiedTopology: true,
      });
      console.log(`Connected to database successfully`);
    } catch (error) {
      console.log(`this is an error message`);
      console.log(error.message);
    }
  }
```

OUR TOURISM WEBSITE