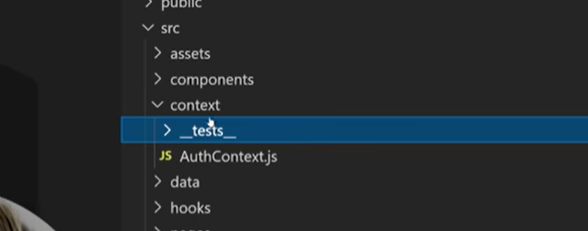
A screenshot of a video chat

Description automatically generated with medium confidence

Assets – contains img or global css



Context - context provides a way to share data between components without explicitly passing it through each level of the component tree. It allows you to create a global state or share specific data between components that are not directly connected by parent-child relationships.

In the simplest terms, context in React is a way to share data between components without having to pass it through multiple levels of the component tree using props.

Imagine you have a large React application with many nested components. Usually, if you want to pass some data from a top-level component to a deeply nested component, you would have to pass that data as props through each intermediate component in the hierarchy. This is called "prop drilling," and it can become cumbersome and make the code harder to maintain.

Context solves this problem by providing a mechanism to create a shared data store that can be accessed by any component in the application, regardless of its position in the component tree. It allows you to define data at a higher level and make it available to any component that needs it, eliminating the need for passing props through multiple layers.

With context, you create a context object that holds the shared data, and then you wrap the components that need access to that data with a context provider component. The provider component makes the data available to its child components. Any component within the provider's descendant components can access the shared data using the context.

By using context, you can simplify the process of sharing data and avoid excessive prop drilling, making your code cleaner and more maintainable.

It's important to note that context should be used judiciously. It's best suited for cases where certain data needs to be accessed by many components throughout the application. If the data is only relevant to a small portion of the component tree, using props might still be the appropriate approach.

In summary, context is a way to create a shared data store in React that allows components to access data without the need for prop drilling. It simplifies the process of sharing data and makes the code more maintainable, especially in larger applications with complex component hierarchies.

Utils – in ReactJS, the term "utils" is short for "utilities" and generally refers to helper functions or modules that provide commonly used functionality or perform specific tasks. Utils are not specific to React itself but are often used in React projects to handle various operations.

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

Components – this folder contains the components that can be reused by other pages

A screenshot of a computer

Description automatically generated with medium confidence

Form Components:

Form components are responsible for handling user input and managing form state. They typically include input fields, checkboxes, radio buttons, and submit buttons. Form components encapsulate the logic for capturing user input, validating the input, and submitting the form data.

A screenshot of a computer

Description automatically generated with medium confidence

UI Components:

UI components, also known as presentational or dumb components, are responsible for rendering the user interface and displaying data. They receive data and event handlers as props and provide a visual representation of the application's state.

A screenshot of a computer program

Description automatically generated with low confidence

In simpler terms, form components are responsible for the "thinking" part of a form, handling user input, and managing the form's behavior. UI components, on the other hand, are responsible for the "look and feel" part of a form, displaying the form visually and providing a pleasant user interface.

Pages – this folder contains the name page and the unique components to it