React Context API

IIHT

Time To Complete: 10 to 12 hr

CONTENTS

1 Project Abstract

Managing state across deeply nested components can become complex when relying solely on prop drilling. React's **Context API** provides a powerful solution for sharing state and functions throughout the component tree without manually passing props at every level. This project introduces the Context API by building a multilingual web interface. Users can select a language from a dropdown, and the application dynamically updates content across multiple components based on the selected language. This example demonstrates how to create a global state using context, consume it with custom hooks, and enable reactive UI changes throughout the application.

2 Problem Statement

You are tasked with building a language-aware React application using the **Context API**. The app should allow users to select a language (English, Spanish, or French) using a dropdown, and reflect that selection in both the `Header` and `Content` components. The current language state should be stored centrally using React Context and updated through a shared function. The application must:

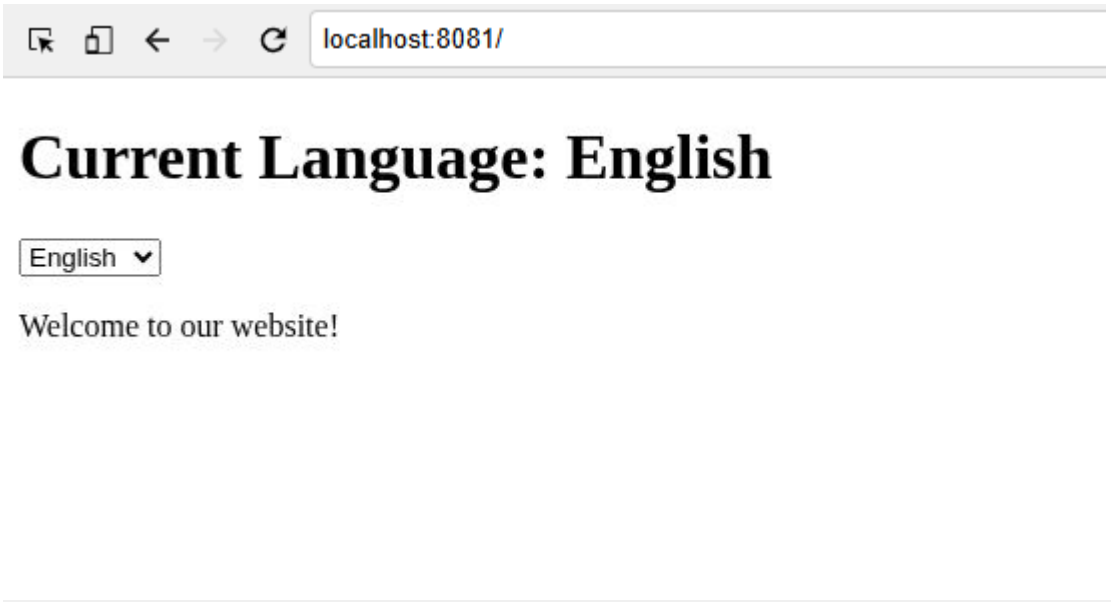
- Use `createContext` and `useContext` to create and access the language state.
- Provide a `LanguageProvider` that wraps the application to manage global language state.
- Enable any component to read and update the language without prop drilling.
- Demonstrate a dynamic content update based on user interaction.

This project helps reinforce the use of React's Context API for shared state and real-time UI reactivity across component hierarchies.

3 Proposed React Context API Application Wireframe

UI needs improvisation and modification as per given use case and to make test cases passed.

3.1 ScreenShots



4 BUSINESS-REQUIREMENT:

As an application developer, develop the React Context API (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Welcome Page	As a user I should be able to visit the welcome page as the default page. Acceptance criteria: 1. Use `LanguageContext` to store and manage the current language selection. 2. Provide access to the language and update function (`changeLanguage`) across all child components. 3. The UI should display: ● Current language in the **Header**. ● A **dropdown** to switch between languages. ● A **message** in the selected language displayed in the `Content` section. **State & Context Overview:** **LanguageContext (Shared State):** 1. Context created using `createContext`. 2. Manages: ● `language`: current selected language (`'en'`, `'es'`, `'fr'`)

		● `changeLanguage`: function to update the language 3. State is managed using `useState` in `LanguageProvider`. **useLanguage (Custom Hook):** 4. Wraps `useContext(LanguageContext)` for easier usage in components. **Component Breakdown:** **App Component** 1. Wraps all content inside `LanguageProvider` to share state globally. 2. Renders: ● Shows `<LoadingSpinner />` while loading. ● Displays error if present. ● Renders: ➔ `<Header />` → shows current language ➔ `<LanguageSelector />` → dropdown to select a language ➔ `<Content />` → text content based on selected language **Header Component** 3. Accesses `language` from context. 4. Displays a heading indicating the current language (e.g., "Current Language: English") **LanguageSelector Component** 5. Accesses `changeLanguage` function from context. 6. Renders a `<select>` dropdown with options for English, Spanish, and French. 7. Updates the global language when the selection changes. **Content Component** 8. Accesses `language` from context. 9. Displays a message in the selected language: ● `en` → Welcome to our website! ● `es` → ¡Bienvenido a nuestro sitio web! ● `fr` → Bienvenue sur notre site web!

		HTML Structure: 1. **App Component:** ● Use a top-level `<div>` with class `"App"` to wrap the layout. ● Render the following components in order: ➔ `<Header />` ➔ `<LanguageSelector />` ➔ `<Content />` 2. **Header Component:** ● Use a `<header>` tag. ● Inside it, use an `<h1>` with the text: ➔ **"Current Language: English"**, ➔ **"Current Language: Spanish"**, or ➔ **"Current Language: French"** — based on the current context value. 3. **LanguageSelector Component:** ● Use a `<div>` containing a `<select>` dropdown. ● Provide the following `<option>` values: ➔ **English** ➔ **Spanish** ➔ **French** 4. **Content Component:** ● Use a `<div>` with a single `<p>` tag inside. ● The paragraph displays a welcome message in the selected language: ➔ English → "Welcome to our website!" ➔ Spanish → "¡Bienvenido a nuestro sitio web!" ➔ French → "Bienvenue sur notre site web!" **Dynamic Behavior:** 1. Changing the value in the language dropdown: ● Updates the global `language` state. ● Triggers re-render of all subscribed components (`Header` and `Content`).

		2. No props are passed between these components — **everything is powered by context**. **** Kindly refer to the screenshots for any clarifications. ****

5 CONSTRAINTS

1. You should be able to press the “TAB” key and “SHIFT + TAB” to navigate from top field to bottom field and vice-versa.

6 MANDATORY ASSESSMENT GUIDELINES

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to**
   Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.
3. **This editor Auto Saves the code.**
4. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
5. **This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**
   Note: The application will not run in the local browser
6. **You can follow series of command to setup React environment once you are in your project-name folder:**
   a. **npm install -> Will install all dependencies -> takes 10 to 15 min.**
   b. **npm run start -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:8080/8081 to open project in browser -> takes 2 to 3 min.**
   c. **npm run jest -> to run all test cases and see the summary.**

d. **npm run test -> to run all test cases and register results on the server. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min.**

7. **Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on "Submit Assessment" after you are done with code.**