# Context API

* Allows Components anywhere in the COMPONENT TREE to access a State that is shared by a **CONTEXT**

## What is Context API?

* A system to pass data through the app without **manually passing props**
* It allows us to **BROADCAST GLOBAL STATE**

## Context API parts

1. The **PROVIDER**
   1. Gives all child components access to value
   2. Common to be placed at the very top
   3. We pass the value we want to be broadcasted into the provider
2. The **CONSUMERS**
   1. All components that read the **PROVIDED value**

When the **context VALUE** changes, **ALL CONSUMERS WILL BE RE-RENDERED**

# Creating and Providing Context

1. Create the **Provider**
   1. We have to create a new Context
      1. Const PostContext = createContext()
      2. The variable starts with **UPPERCASE** because it’s actually a COMPONENT
      3. A screen shot of a computer

         Description automatically generated
2. We wrap the children’s that want to have access to the context A screen shot of a computer program

   Description automatically generated
   1. Provide the value, as an **OBJECT**
   2. A screen shot of a computer program

      Description automatically generated
3. Consume the Context data – useContext(context name)
   1. We use the **useContext hook** to get the values from the provider
   2. A computer screen shot of text

      Description automatically generated
   3. A computer screen shot of a code

      Description automatically generated
   4. A black background with white text

      Description automatically generated

# Custom Provider and custom HOOK

* Advanced pattern
* We move everything that is related to the Context and move it into a separate file
* We move all the state and the state updating logic and move it into the new Component
* A screen shot of a computer screen

  Description automatically generated
* We return the same thing ( PostContext.Provider)
  + With the same OBJECT
  + And also the {children} since we have placed all other components inside it in the main APP
  + A screen shot of a computer program

    Description automatically generated

# The Custom Hook

* We are creating it into the same new File
* We are getting the context
* We are returning the context
* Export the custom HOOK
* A screen shot of a computer program

  Description automatically generated
* Use the **CUSTOM** **HOOK** instead of the useContext hook
  + A computer screen shot of a computer screen

    Description automatically generated
* In case somebody uses the custom HOOK **outside of the PostProvider component**, we throw an error
* A screen shot of a computer code

  Description automatically generated

# Advanced STATE MANAGEMENT

## Types of State

1. State Accessibility
   1. Local State
      1. Needed by 1 or a few components
      2. Accessible inside the component or child components
   2. Global State
      1. Needed by many components
      2. Accessible to every component in the app
2. State Domain
   1. Remote State
      1. Application data that is loaded from remote Server (API)
      2. Usually ASYNC
      3. Needs Re-fetching + updating
   2. UI State
      1. Everything else
      2. Filters, open panels, theme
      3. Usually Synchronous
      4. Stored in the application
      5. Easy to Handle

## State Placement

* Local Component
  + useState, useReducer, useRef
  + Local State
* Parental Component
  + useState, useReducer, useRef
  + Lifting up State
* Context
  + Context API + useState or useReducer
  + Context API is used to **PROVIDE ACCESS TO THE STATE**
  + useState, useReducer is used to **update the state**
  + **BEST FOR UI STATE**
* 3rd Party Library
  + Redux, React Query, SWR, Zustand
  + **BEST FOR REMOTE STATE**
  + Global State ( remote or UI)
* URL
  + React Router
  + Global State, passing between pages
* Browser
  + Local Storage, session Storage

## State Management TOOLS

* **Local UI state**
  + useState
  + useReducer
  + useRef
* **Local Remote**
  + Fetch + useEffect + useState / useReducer
  + Good in SMALL APPLICATIONS
* **Global Remote**
  + Context API + useState/ useReducer
  + Reduxe, Zustand, Recoil
  + Highly Specialized TOOLS for Remote State
    - React Query
    - SWR
    - RTK Query
* **Global UI**
  + Context API + useState/ useReducer
  + Redux, Zustand, Recoil
  + React Router

# NEXT STEPS ARE DONE IN THE WORLDWISE app from 11.React Router

# Apply the Context API into WorldWise

* A common technique is to create a new folder for Context components
* We create a new Context 🡪 CitiesContext, which will also have the custom hook
  + Create the new context
    - Const CitiesContext = createContext()
  + Create the Provider
    - Function CitiesProvider({children}){}
  + Move the State logic from APP to the new Component
  + Create the Hook
  + Refactor the App to use the Context created

# User Authentication

* We are using a FAKE Authentication (the user and password are hardcoded)
* We will create a new CONTEXT and provide access to the whole application

## Protect an Authentication (Protecting a Route) – against unauthorized access

* We Need to WRAP the whole application in the Redirect
* We check if the isAuth is true. If NOT, it will REDIRECT TO THE HOMEPAGE
* A screen shot of a computer

  Description automatically generated
* We wrap the whole APP with this Component
* A screen shot of a computer code

  Description automatically generated
* Now, if a user is REFRESHING THE PAGE, it will take him straight to HOMEPAGE
* If the users tries to access with the URL, it will not work if it’s not authenticated