Redux presentation :

Learning goals :

1. Complete understanding of what is redux
2. Complete understanding of why Redux
3. In depth understanding all pieces of Redux
    - Store
    - Reducer
    - Action

What is Redux?

Redux is a state management tool for JavaScript applications.
While it is frequently used with React, it is compatible with many other React-like frameworks such as Preact and Inferno as well as Angular and even just plain JavaScript.
The main concept behind Redux is that the

- **Entire state of an application is stored in one central location.**
- Each component of an application can have direct access to the state of the application without **having to send props down to child components or using callback functions to send data back up to a parent.**

Why Redux?

- Predictable state updates make it easier to understand how the data flow works in the application
- The use of "pure" reducer functions makes logic easier to test, and enables useful features like "time-travel debugging".
- Centralizing the state makes it easier to implement things like logging changes to the data, or persisting data between page refreshes

  State transfer between components is pretty messy in React since it is hard to keep track of which component the data is coming from. It becomes really complicated if users are working with a large number of states within an application.
  Redux solves the state transfer problem by storing all of the states in a single place called a store. So, managing and transferring states becomes easier as all the states are stored in the same convenient store. Every component in the application can then directly access the required state from that store.

Pillars of Redux :

Store :

A store is an object that holds the application's state tree. There should only be a single store in a Redux app, as the composition happens at the reducer level.

Action

An action is a plain object that represents an intention to change the state. They must have a property to indicate the type of action to be carried out.
Actions are payloads of information that send data from your application to your store.
Any data, whether from UI events or network callbacks, needs to eventually be dispatched as actions.
Actions must have a type field, indicating the type of action being performed.

Reducers

Reducers are pure functions that specify how the application's state changes in response to actions sent to the store.
Actions only describe what happened, not how the application's state changes.
A reducer is a function that accepts the current state and action, and returns a new state with the action performed.
combineReducers() utility can be used to combine all the reducers in the app into a single index reducer which makes maintainability much easier.

Pros :

1.There is always one source of truth, the store. Thus, there is no confusion about how to sync the current state with actions and other parts of the application.

2.The code is easier to maintain because it has a predictable outcome and strict structure.

3.Redux makes coding more consistent due to more stringent code organization procedures

4.It’s very useful, especially during the initial render, making for better user experience and search engine optimization.

5.Developers can track everything going on in the app in real-time—from actions to state changes

Cons :

1.Since it has no encapsulation, any component can access data, which may potentially cause security issues

2.Some parts of the code are just boilerplate. However, these parts have to be incorporated with no alteration, and this restricts the design

3.As the state is immutable in Redux, the reducer updates the state by returning a new state every time which can cause excessive use of memory

(All steps in a nutshell)

1. Creation of Store (the brain of our application)
2. Creation of RootReducer (where based on the action type and new info it updates the info in the brain)
3. Creation of Action (where the action type is defined)
4. Connecting the react component with the brain => to call an action => change the info in the brain => receive the updated info