In Redux, when we want to **send data to the store**, we use something called an **action**. An **action** is an object that describes what happened and carries the data (called payload) to be stored or processed.

Once the action is sent using **dispatch**, it reaches the **reducer**, which updates the state based on the action type and payload.

The updated data is then stored in the **Redux store**.

To **get or access the data from the store**, we use a method like useSelector (in React), which allows components to read the current state from the store and display or use it in the UI.

So, in short:

* **Action**: Used to send data to the Redux store.
* **Store**: Central place where data is kept.
* **We get data from the store** using selectors in our components.

To install npm I react-redux @reduxjs/toolkit

**Real-World Analogy: A Library System**

Imagine you are managing a **library system** where books are added to the library collection, and you need to keep track of the list of books available in the library. Each book is like a **customer** in your system. Now, let’s go through the Redux flow using this analogy.

**Step 1: Setting Up the Store (Library Database)**

In the Redux world, the store is where all your application’s data is kept. This is like the **library database** where all the information about books (customers) is stored. You create this database (store) once and connect it to your application.

* **Redux Store** is your **library database**.
* **Reducers** in Redux are like the **library staff** who handle requests like adding new books, removing old ones, etc.

When you create the Redux store and add the **reducer** for managing customers, you are essentially setting up a **system to store customer data** in your application.

**Step 2: Dispatching an Action (Adding a New Book to the Library)**

When you want to add a customer to the list (or a book to the library), you need to **dispatch an action**. This is like telling the **library staff** that you want to add a new book to the collection.

* **Action**: This is like a **request form** to the library staff. It contains the necessary information (like the book title, author, etc.), which in your case is the customer’s name.
* **Dispatch**: You are essentially **sending this request to the library staff**, asking them to add the book (customer) to the collection.

**Step 3: Reducer Handling the Action (Library Staff Adding the Book)**

The **reducer** (the library staff) listens for the action. When the **dispatch** sends the action to the store, the reducer checks the action type and decides how to update the store (the library database).

In your case, when the action is dispatched to add a customer, the **reducer adds the customer’s name** to the existing list of customers stored in the state.

* **Reducer**: It’s like the library staff who receives the action and updates the database with the new information (adding a new book).
* **State**: This is the **library’s book collection** (or the customer list in your case). The reducer modifies the state based on the action payload (the customer’s name).

**Step 4: The State Update and Re-rendering the UI (Library’s Book Shelf Changes)**

Once the **reducer updates the state** (the list of books/customers), this new information is now available to any part of the application that needs it.

The component (Customer.jsx in your case) listens for changes to the Redux state using the useSelector hook, just like a visitor (you) might want to see the updated **book collection** on the library shelf.

* **useSelector**: It’s like going to the library shelf and **checking the updated book collection**.
* **Re-render**: When the state changes (e.g., a new book is added), the **component re-renders** to show the updated list, just like the library display changes after a new book is added.

**Step 5: Viewing the Updated Customer List (Seeing the Books on the Shelf)**

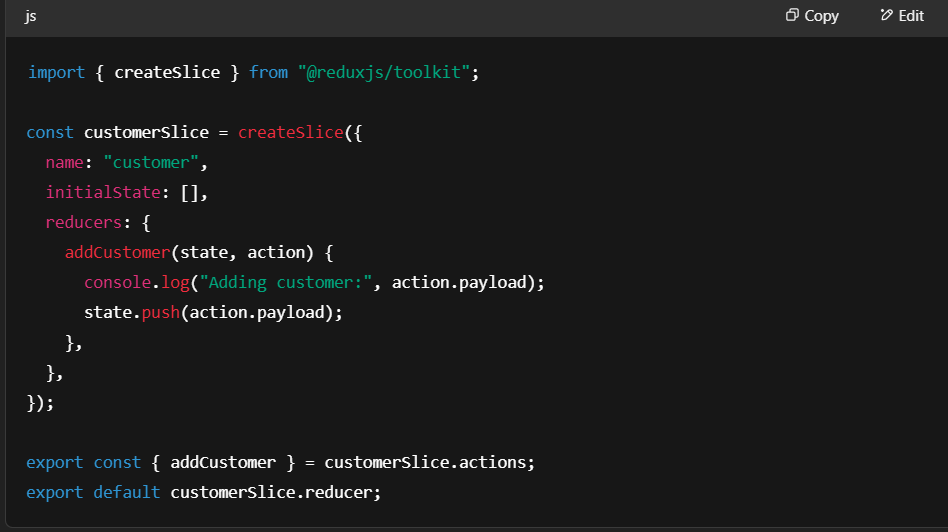
After the action is processed by the reducer and the state is updated, the UI automatically re-renders. This is like walking into the library and seeing the updated book collection, which now includes the new book.

* **React Component**: Your React component acts as the **display window of the library** that shows the list of books (customers). Every time the Redux state is updated (i.e., when a customer is added), the component re-renders to show the new state.

**Putting It All Together in the Flow (Library System)**

Here’s the flow in simple steps:

1. **You add a new customer**: You fill out the customer’s name and click "Add". This is like submitting a **request to the library staff to add a new book**.
2. **Action is dispatched**: The action (add customer) is dispatched, which contains the necessary information (the customer’s name).
3. **Reducer handles the action**: The reducer (library staff) updates the state (library database) by adding the customer (book).
4. **State is updated**: The state is updated in the Redux store, just like the library database is updated.
5. **UI re-renders**: The component (library display) updates the UI to show the new customer in the list (new book on the shelf).



 initialState: [] → You start with an empty array of customers.

 addCustomer(state, action) → This function gets triggered when you want to add a customer.

* action.payload contains the customer name.
* It logs to console and pushes the customer name into the Redux state (mutating the array).

 createSlice() is a Redux Toolkit helper that **automatically creates action creators** (dispatch function )and reducers.(library staff that is add function)

A screenshot of a computer program

AI-generated content may be incorrect.

 useDispatch() → Hook to send an action to Redux.

 useSelector() → Hook to get current customers from Redux store.

**When Button is Clicked:**

1. Checks if the input isn't empty.
2. Sends the name to Redux using dispatch(addCustomer(name)).
3. The slice reducer updates the state with the new customer.
4. Component re-renders with updated customers.

For debug install react redux tool extension in chrome developer tools