# REDUX

## What is Redux?

- Redux is a state management library for JavaScript applications.
- In other words, redux is used to manage the data or state of complex JavaScript applications.

## Manage application state


State Management Tools
Flux
MobX
Redux
Redux is the most popular tool
for state management.

Section 01

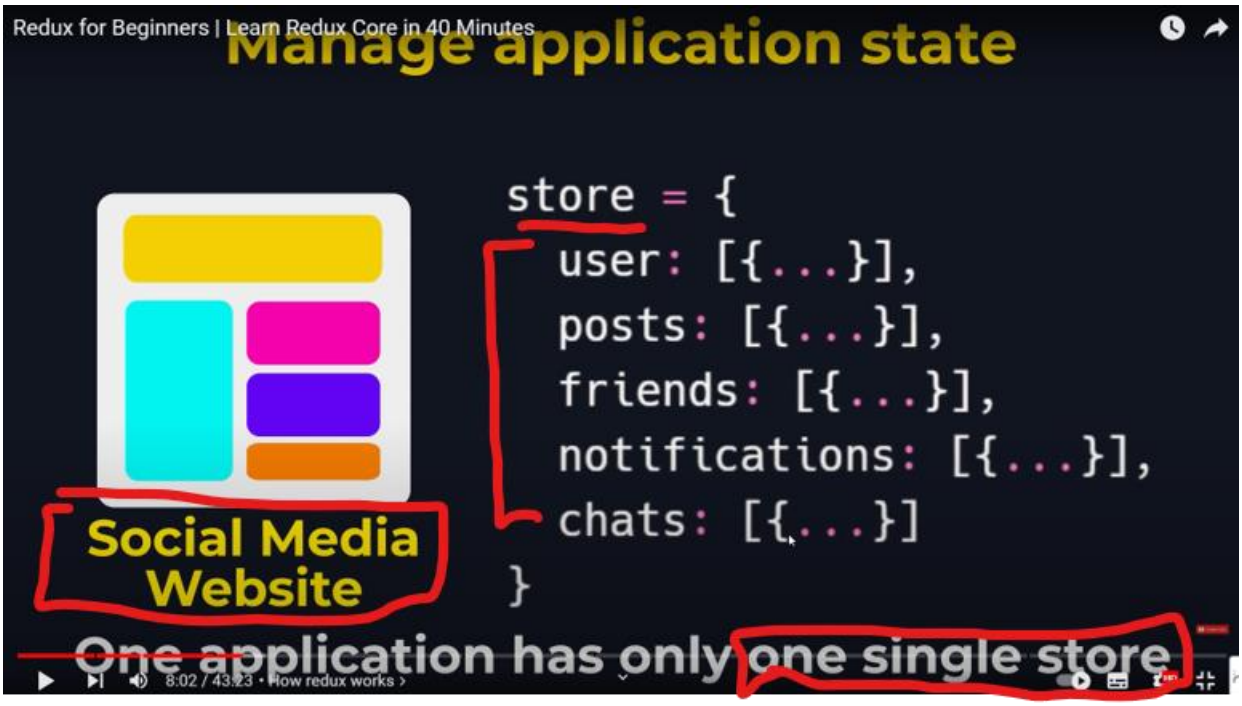
# 03 When you use Redux?

## When we use Redux

- Complex User interfaces in terms of data like facebook, amazon
- Data flow is complex

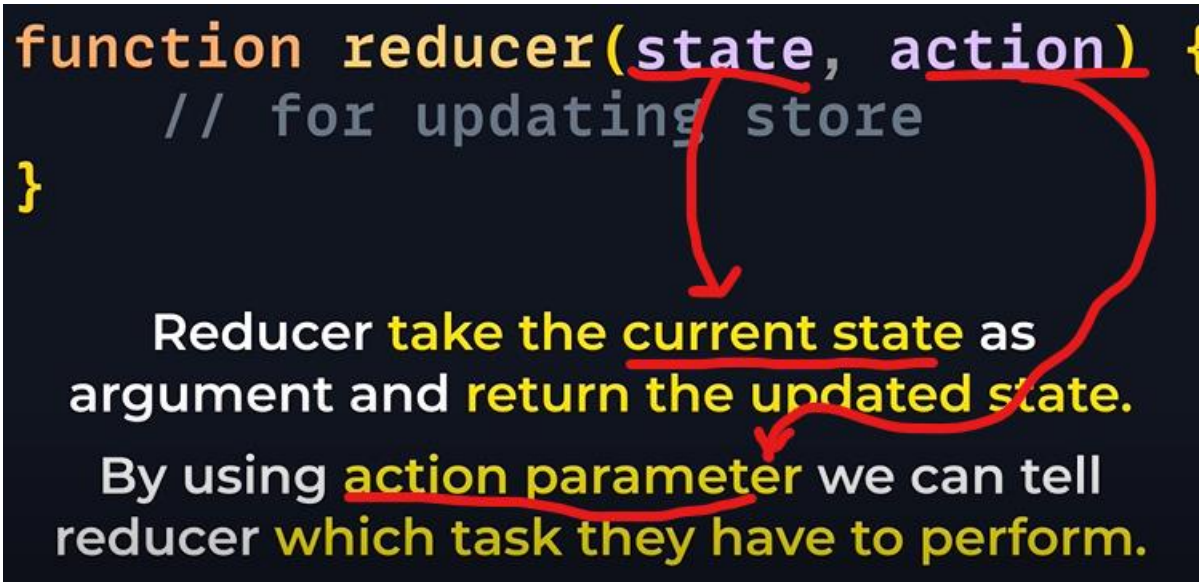
## When we don't use Redux

- Small or medium size of applications
- Simple UI & static data


**Reducer function is used to update the store**

**Reducer function take 2 parameters**



**Actions** - What to do

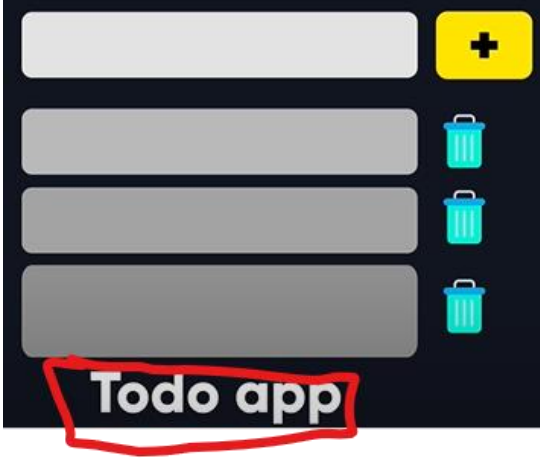
**Reducers** - How to do

**Store** - Keep data in single place

Section 03

# 03 Introduction of application

## Introduction of our application




- Add the tasks
- Remove the tasks
- Mark as completed

## Steps for implementing redux

- Designing the store
- List our actions (What to do)
- Create reducer function (How to do)
- Create redux store

Section 03

# 04 Designing store structure


**For example we 2 data array in single store we have add 2 reducers**

Section 03

# 05 Listing all actions

Step – 2

## Listing all actions

### Listing all actions

- ADD_TASK
- REMOVE_TASK
- TASK_COMPLETED

Action = What to do

```
const addTaskAction =
{
  type: "ADD_TASK",
  task: "This is new task!"
}
const removeTaskAction =
{
  type: "REMOVE_TASK",
  id: 1
}
```

# Section 03
# 06 Creating Reducer

## Create reducer.js in src

## Create Add reducer function

```
let id=0;
function reducer(state=[],action)
{
    if(action.type=="ADD_Task")
        {
            return[
                ...state,
                {
                    id:++id,
                    task:action.payload.task,
                    completed:false
                }
            ]
        }

}
```

## Create Add reducer function

## It will checks id
## if id gets equal it remove
## if id not equal it keep it as it was

```
let id = 0;
function reducer(state = [], action) {
  if (action.type === "ADD_Task") {
    return [
      ...state,
      {
        id: ++id,
        task: action.payload.task,
        completed: false,
      },
    ];
  } else if (action.type === "REMOVE_TASK") {
    return state.filter((task) => task.id !== action.payload.id);
  }
  return state
}
```

## Above we have use  if – else which not effective use switch case for effectiness -→

```
let id = 0;
function reducer(state = [], action) {
  switch (action.type) {
    case "ADD_TASK":
      return [
        ...state,
        {
          id: ++id,
          task: action.payload.task,
          completed: false,
        },
      ];
    case "REMOVE_TASK":
      return state.filter((task) => task.id !== action.payload.id);
    default:
      return state;
  }
}
```

# Section 03
# 07 Creating redux store

## Creatin redux store→most hardest→ justkidding

## To work on redux we install → `npm install redux`

```
store.js > default
import { legacy_createStore as createStore } from "redux";
import reducer from "./reducer";

// createReducer() --> takes parameter of ROOTREDUCER
//RootReducer --> combination of all reducers
const store = createStore(reducer);

export default store;
```

## -→ now open index.js

## Import store to see what the store contains

```
index.js > ...
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
import store from "./store";

console.log(store);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

```
▼ Object i
  ▶ @@observable: ƒ observable()
  ▶ dispatch: ƒ dispatch(action)
  ▶ getState: ƒ getState()
  ▶ replaceReducer: ƒ replaceReducer(nextReducer)
  ▶ subscribe: ƒ subscribe(listener)
  ▶ [[Prototype]]: Object
```

**all are important we will discuss later**

# Section 03
# 08 Dispatching the actions

In Redux, `dispatch` is a method used to send actions to the Redux store. Actions are plain JavaScript objects that describe an event or change that should happen in the application's state.

## Step-by-Step Example

1. **Define an Action**: First, you define what your action looks like.

```javascript
const addAction = {
  type: 'ADD_TASK',
  payload: {
    task: 'Write a report'
  }
};
```

2. **Dispatch the Action**: Then, you send (dispatch) this action to the Redux store.

```javascript
store.dispatch(addAction);
```

3. **Reducer Updates the State**: The reducer function receives the current state and the dispatched action, and returns the new state.

```javascript
function taskReducer(state = [], action) {
  switch(action.type) {
    case 'ADD_TASK':
      return [...state, action.payload.task];
    default:
      return state;
  }
}
```

# In index.js

```javascript
import store from "./store";


console.log(store.getState());
```

```
▼ Array(0) i
    length: 0
  ▶ [[Prototype]]: Array(0)
>
```

# Output → Empty array

## →By using dispatch add todo

```
import store from "./store";

store.dispatch({ type: "ADD_TASK", payload: { task: "Task 1" } });
console.log(store.getState());
```

```
▼ [{…}]
  ▶ 0: {id: 1, task: 'Task 1', completed: false}
    length: 1
  ▶ [[Prototype]]: Array(0)
>
```

## Output → data is added to array

## → By using dispatch delete todo

```
import store from "./store";

store.dispatch({ type: "ADD_TASK", payload: { task: "Task 1" } });
store.dispatch({ type: "REMOVE_TASK", payload: { id: 1 } });

console.log(store.getState());
```

```
▼ []
    length: 0
  ▶ [[Prototype]]: Array(0)
>
```

## Output → the array is empty

**Above is not good method to follow so, follow below**

**Create file action.js in src**

```
action.js > ...
export const addTask = (task) => {
  return { type: "ADD_TASK", payload: { task: task } }
};
```

**Import in index.js**

```
import store from "./store";
import { addTask } from "./action";

store.dispatch(addTask("Task 1"));
console.log(store.getState());
```

```
▼ [{…}]
  ▶ 0: {id: 1, task: 'Task 1', completed: false}
    length: 1
  ▶ [[Prototype]]: Array(0)
```

**Output → data added to array**

**Here above to import addTask we used {} bcoz**

**During export of addTask in action.js we just written export but not export default**

# Now remove task -→

# In action.js

```
export const addTask = (task) => {
  return { type: "ADD_TASK", payload: { task: task } };
};

export const removeTask = (id) => {
  return { type: "REMOVE_TASK", payload: { id: id } };
};
```

# Import in index.js

```
import { addTask } from "./action";
import { removeTask } from "./action";

store.dispatch(addTask("Task 1"));
store.dispatch(removeTask(1));
console.log(store.getState());
const root = ReactDOM.createRoot(document.getE
```

```
▼ [] i
    length: 0
  ▶ [[Prototype]]: Array(0)
>
```

# Output → empty array

# Section 03
# 09 Making ActionTypes

## Keeping all the varbiles here if we wanted to change anything change here it will update every where

## Creater actionType.js in src →

```
src > JS actionTypes.js > ...
export const ADD_TASK = "ADD_TASK";
export const REMOVE_TASK = "ADD_TASK";
```

## Now import it in  reducer.js→

```
import * as actionTypes from "./actionTypes";
let id = 0;
export default function reducer(state = [], action) {
  switch (action.type) {
    case actionTypes.ADD_TASK:
      return [
        ...state,
        {
          id: ++id,
          task: action.payload.task,
          completed: false,
        },
      ];
    case actionTypes.REMOVE_TASK:
      return state.filter((task) => task.id !== action.payload.id);
    default:
      return state;
  }
}
```

# Section 03
# 10 Subscribe & Unsubscribe

**Subscribe→ it will run when where there is update in redux store state**

```
store.subscribe(() => {
  console.log("updated", store.getState());
});
```

**Unsubscribe → subscribe method bring unsubscribe in it**

**** we did not get notify if store schange**

```
const unsubscribe=store.subscribe(() => {
  console.log("updated", store.getState());
});

store.dispatch(addTask("Task 1"));
store.dispatch(removeTask(1));
console.log(store.getState());

unsubscribe()
```

# ADD ONE MORE FUNCTIONALITY OF COMPETED OR NOT

## 1)in actionType.js

```
JS actionTypes.js > ...
export const ADD_TASK = "ADD_TASK";
export const REMOVE_TASK = "ADD_TASK";
export const TASK_COMPLETED="TASK_COMPLETED"
```

## 2) in reducer.js

```
switch (action.type) {
  case actionTypes.ADD_TASK:
    return [
      ...state,
      {
        id: ++id,
        task: action.payload.task,
        completed: false,
      },
    ];
  case actionTypes.REMOVE_TASK:
    return state.filter((task) => task.id !== action.payload.id);
  case actionTypes.TASK_COMPLETED:
    return state.map((task) =>
      task.id === action.payload.id ? { ...task, completed: true } : task
    );
  default:
    return state;
}
```

# 3)in action.js

```
JS action.js > ...
import * as actionTypes from "./actionTypes"
export const addTask = (task) => {
  return { type: actionTypes.ADD_TASK, payload: { task: task } };
};

export const removeTask = (id) => {
  return { type: actionTypes.REMOVE_TASK, payload: { id: id } };
};
const completedTask=(id)=>{
    return {type:actionTypes.TASK_COMPLETED,payload:{id:id}}
}
```

Section 03

## 13 Folder Structure for Redux

```
src
  store
    - store.js
      tasks
        - action.js
        - reducer.js
        - actionTypes.js
      employees
        - action.js
        - reducer.js
        - actionTypes.js
```

**Or → other way we can follow any one**

### Folder Structure (Duck Module)

```
src
  store
    - store.js
    - tasks.js
    - employees.js
```

**In duck module → keeping all code in single file**