# Unit 7: Store, Data flow

# Store

Holds application state;

Allows access to state via getState();

Allows state to be updated via dispatch(action);

Registers listeners via subscribe(listener);

Handles unregistering of listeners via the function returned by subscribe(listener).

It's important to note that you'll only have a single store in a Redux application. When you want to split your data handling logic, you'll use reducer composition instead of many stores.

```
import { createStore } from 'redux'
import todoApp from './reducers'
let store = createStore(todoApp)
```

# Store

**Dispatching actions**

**(eg in the code sample)**

## Data Flow

Redux architecture revolves around a strict unidirectional data flow.

The data lifecycle in any Redux app follows these 4 steps:

1. You call store.dispatch(action).

An action is a plain object describing *what happened*. For example:

```
{ type: 'LIKE_ARTICLE', articleId: 42 }
```

```
{ type: 'FETCH_USER_SUCCESS', response: { id: 3, name: 'Mary' } }
```

```
{ type: 'ADD_TODO', text: 'Read the Redux docs.' }
```

## Data Flow

2. The Redux store calls the reducer function you gave it.
The store will pass two arguments to the reducer: the current state tree and the action. For example, in the todo app, the root reducer might receive something like this:

# Data Flow

```
let previousState = {
  visibleTodoFilter: 'SHOW_ALL',
  todos: [
    {
      text: 'Read the docs.',
      complete: false
    }
  ]
}

let action = {
  type: 'ADD_TODO',
  text: 'Understand the flow.'
}

// Your reducer returns the next application state
let nextState = todoApp(previousState, action)
```

## Data Flow

3. The stage changes lead to changes in React components