



# Presentations of part 2:

Group member: Trần Đức Tâm, Nguyễn Ngô Thành Phát, Trần Minh Trí

# I. Flux

Install

```
> npm i flux
```

## 1) Definition:

Flux is an application architecture that Facebook uses internally for building the client-side web application with React. It is not a library nor a framework.

It is useful when the project has dynamic data and we need to keep the data updated. If we just use static data for making website and just view it, no update, no stored it, no share the status then flux is useless with us.

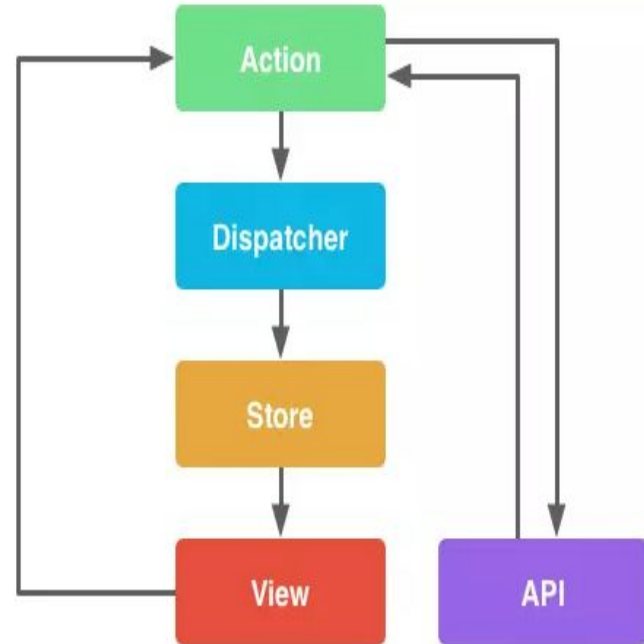
# 1. Flux

## 2) Structure and data flow:

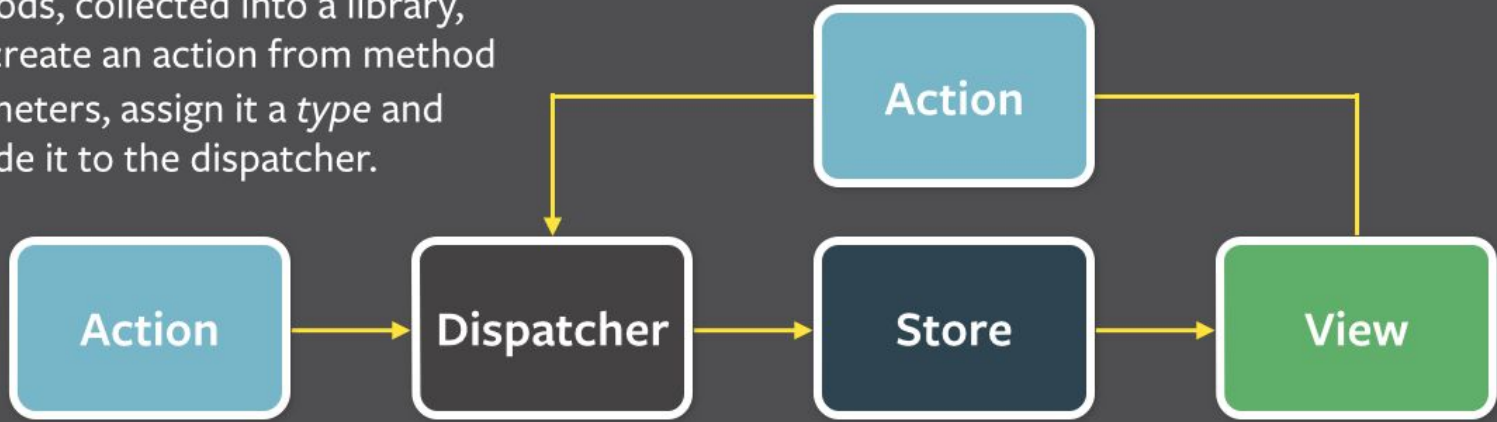
### a) Structure:

It has four part: Action - Dispatcher - Stores - Controller views.

### b) Data flow:



*Action creators* are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.



Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.

After stores update themselves in response to an action, they emit a *change* event.

Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

# I. Flux

## 3) Method:

`_register()`: It is used to register a store's action handler callback.

`_unregister()`: It is used to unregisters a store's callback.

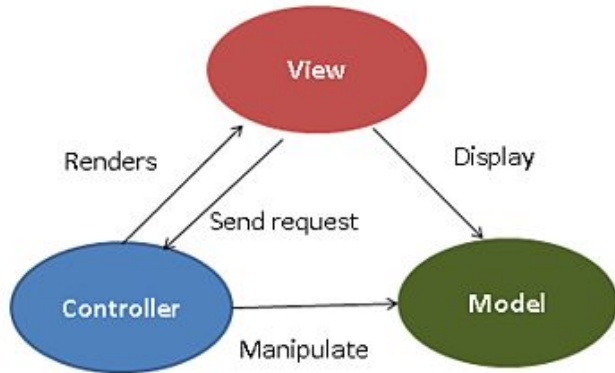
`_waitFor()`: It is used to wait for the specified callback to run first.

`_dispatch()`: It is used to dispatches an action payload.

`_isDispatching()`: It is used to checks if the dispatcher is currently dispatching an action.

# I.Flux

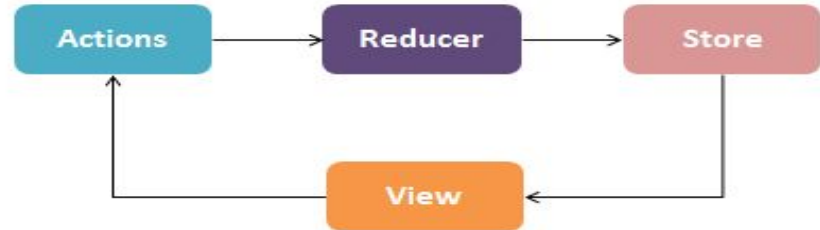
## 4) Comparing MVC vs Flux vs Redux



**MVC Architecture**



**Flux Architecture**



**Redux Architecture**

	<b>MVC</b>	<b>Flux</b>	<b>Redux</b>
<b>Data Flow Direction</b>	Follows the bidirectional flow	Follows the unidirectional flow	Follows the unidirectional flow
<b>Stores</b>	No concept of store	Includes multiple stores	Includes single store
<b>Business Logic resides</b>	Controller handles entire logic	Store handles all logic	Reducer handles all logic
<b>Debugging handled</b>	Debugging is difficult due to bidirectional flow	Ensures simple debugging with the dispatcher	Single store makes debugging lot easier
<b>can be used</b>	Shines well in both client and server-side frameworks	Supports client-side framework	Supports client-side framework

## II. React-router

Install

```
> npm i react-router
```

### 1) Definition:

\_React-router is a routing library standard use in ReactJs.

\_It help dev routing the data-flow in application clearly, so it can display true with the URL on browser.

\_It also help dev make a guard for application in a middle level.

Example when user login successfully, we will not allow user come to page signin - signup anymore, unless they sign out.

\_It help dev control the location of page, like after some event, we want the browser back to home, or when user come to page that relative with id, react-router will help us take the id, path, much easier.



## II. React-router

### 2) Structure:

a) BrowserRouter and HashRouter:

\_BrowserRouter is used regular more than hashrouter, it use history API in HTML5 to follow history routing

\_HashRouter use hash of url to remember everything.

```
import * as React from "react";
import * as ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";

ReactDOM.render(
  <BrowserRouter>
    { /* The rest of your app goes here */ }
  </BrowserRouter>,
  root
);
```

## II. React-router

### 2) Structure:

```
<Routes>
  <Route path="/" element={<Dashboard />}>
    <Route
      path="messages"
      element={<DashboardMessages />}
    />
    <Route path="tasks" element={<DashboardTasks />} />
  </Route>
  <Route path="about" element={<AboutPage />} />
</Routes>
```

b)Routes and Route:

\_Is a mapping between URL and component. It mean when user connect by URL on browser, a suitable component will be render on UI.

\_Some element in route : path - URL link , index - only active when address URL don't exactly have path, element - use to render component.

## II. React-router

### 2) Structure:

#### c)Link:

\_Like tag `<a>`, we import link from react-router. By the way it don't make page reload when active, it also have element "to" like "href" in tag `<a>`.

```
return (  
  <View>  
    <Text>Welcome!</Text>  
    <Link to="/profile">Visit your profile</Link>  
  </View>  
);
```

```
import * as React from "react";  
import { View, Text } from "react-native";  
import { Link } from "react-router-native";  
  
function Home() {  
  return (  
    <View>  
      <Text>Welcome!</Text>  
      <Link to="/profile">Visit your profile</Link>  
    </View>  
  );  
}
```

## II. React-router

```
<ul>
  <li>
    <NavLink
      to="messages"
      style={({ isActive }) =>
        isActive ? activeStyle : undefined
      }
    >
      Messages
    </NavLink>
  </li>
  <li>
    <NavLink
      to="tasks"
      className={({ isActive }) =>
        isActive ? activeClassName : undefined
      }
    >
      Tasks
    </NavLink>
  </li>
</ul>
```

### 2) Structure:

d)Navlink:

\_Like link but better because it support some element like activeClassName and activeStyle.

## II. React-router

### 2) Structure:

e)Outlet:

\_Like a children component. It help nested UI to show up when child routes are rendered.

```
<div>
  <h1>Dashboard</h1>

  {/* This element will render either <DashboardMessages> when the URL is
    "/messages", <DashboardTasks> at "/tasks", or null if it is "/"
    */}
  <Outlet />
</div>
```

## II. React-router

### 2) Structure:

e)Other Api:

`_StacticRouter`: should be used when server-rendering website.

`_NativeRouter`: should be used in React Native apps.

`_MemoryRouter`: is useful in testing scenarios and as a reference implementation for the other routers.

`_unstable_HistoryRouter`: is used with your own history instance.

`_Navigate`: let you programmatically navigate, usually in an event handler or in response to some change in state.

# III . Lodash

## 1) Definition:

\_Lodash is a JavaScript library that help dev handle Array, Object, Function, Collection ..v.v. instead of we using normal long javascript.

\_When we use lodash, it can make code shorter and clearly, it's great tool to solve the data, no need to code long javascript for handle data.It make code can be maintain easier and easy to read, saving time code.

\_Website: <https://lodash.com/docs/4.17.15>

# III . Lodash

## 2) Some method to use:

+Array:

`_.chunk(array,size):`

```
_.chunk(['a', 'b', 'c', 'd'], 2);
```

```
// => [['a', 'b'], ['c', 'd']]
```

```
_.chunk(['a', 'b', 'c', 'd'], 3);
```

```
// => [['a', 'b', 'c'], ['d']]
```

`_.sortedUniq(array):`

```
_.sortedUniq([1, 1, 2]);
```

```
// => [1, 2]
```

`_.last(array):`

```
_.last([1, 2, 3]);
```

```
// => 3
```

Install

```
> npm i lodash
```

```
$ npm i --save lodash.chunk
```



# III . Lodash

## 2) Some method to use:

+Object:

```
var users = {  
  'barney': { 'age': 36, 'active': true },  
  'fred': { 'age': 40, 'active': false },  
  'pebbles': { 'age': 1, 'active': true }  
};  
  
// The `_.matches` iteratee shorthand.  
_.findKey(users, { 'age': 1, 'active': true });  
// => 'pebbles'  
  
// The `_.matchesProperty` iteratee shorthand.  
_.findKey(users, ['active', false]);  
// => 'fred'  
  
// The `_.property` iteratee shorthand.  
_.findKey(users, 'active');  
// => 'barney'
```

## IV. Redux

### 1) **Redux - thunk:**

#### a) **Definition:**

\_ Redux thunk is a kind of middleware.

\_ In redux the action will return a plain javascript object, the reducers will take an object and process it.

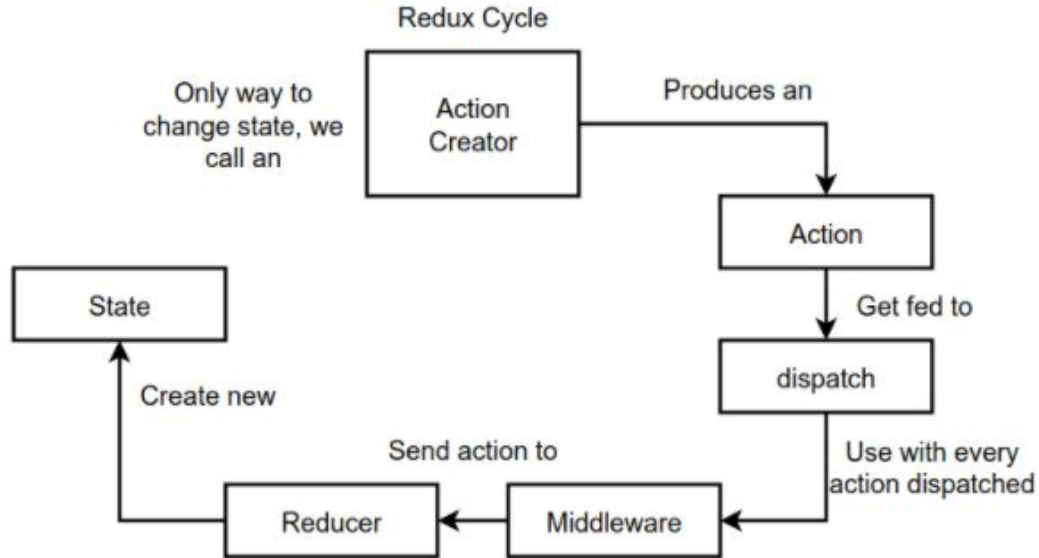
\_ In case the action needs to return a function, these actions are called async action.

\_ Redux Thunk is used to handle complex asynchronous logic that needs to access the Store or simply retrieve data such as an Ajax request.

## IV. Redux

### 1) Redux - thunk:

#### b) Data flow:



## IV. Redux

### 1) Redux - thunk:

c) For example:

Install

```
> npm i redux-thunk
```

JS configReducers.js X

src > redux > JS configReducers.js

```
1  import { applyMiddleware, combineReducers, createStore } from "redux";
2  import reduxThunk from 'redux-thunk';
3  import TodoListReducer from "../reducers/TodoListReducer";
4
5  const configReducers = combineReducers({
6    TodoListReducer
7  });
8
9  let store = createStore(configReducers, applyMiddleware(reduxThunk));
10
11  export default store;
```

## IV. Redux

### 1) Redux - thunk:

c) For example:

```
JS TodoList.js U X
src > components > TodoList > JS TodoList.js > TodoList > [?] showTaskListCompleted > todoLists.map() callback
1  import React, {useState, useEffect} from 'react' 7.2K (gzipped: 3K)
2  import { useSelector, useDispatch } from 'react-redux' 5.9K (gzipped: 2.3K)
3  import { addTaskApi, editTaskAction, getTaskListApi } from '../redux/actions/TodoListActions';
4
5  export default function TodoList(props) {
6    let [name, setName] = useState( {taskName: ''});
7    let dispatch = useDispatch();
8    let todoLists = useSelector(state => state.TodoListReducer.todoLists);
9    useEffect(() => {
10     dispatch(getTaskListApi())
11   }, [])
```

## IV. Redux

### 1) Redux - thunk:

c) For example:

```
JS TodoListActions.js U X
src > redux > actions > JS TodoListActions.js > addTaskApi
1 import Axios from 'axios'; 16.8K (gzipped: 5.9K)
2 import { GET_TASK_LIST } from "../types/ToDoListTypes";
3
4 export const getTaskListApi = () => {
5   return async dispatch => {
6     try {
7       let {data, status} = await Axios({
8         url: 'http://svcy.myclass.vn/api/ToDoList/GetAllTask',
9         method: 'GET'
10      })
11      if(status === 200) {
12        dispatch({
13          type: GET_TASK_LIST,
14          data
15        })
16      }
17    } catch (err) {
18      console.log("🔥 ~ file: TodoListActions.js ~ line 9 ~ getTaskListApi ~ err", err)
19    }
20  }
21 }
```

## IV. Redux

### 1) Redux - thunk:

c) For example:

```
JS TodoListReducer.js U X
src > redux > reducers > JS TodoListReducer.js > [🔍] default
1  import { EDIT_TASK, GET_TASK_LIST } from "../types/ToDoListTypes";
2
3  let initialState = {
4    todoLists: [],
5  }
6
7  const TodoListReducer = (state = initialState, action) => {
8    switch (action.type) {
9      case GET_TASK_LIST: {
10        state.todoLists = action.data;
11        return {...state}
12      }
13    }
14  }
```

## IV. Redux

### 1) **Redux - saga:**

#### **a)Definition:**

Redux-saga is a Redux middleware library that simplifies the management of page effects in Redux applications.

Lets you write asynchronous code that looks like synchronous.  
Make the most of ES6's generator feature.

#### **b)What is generator function:**

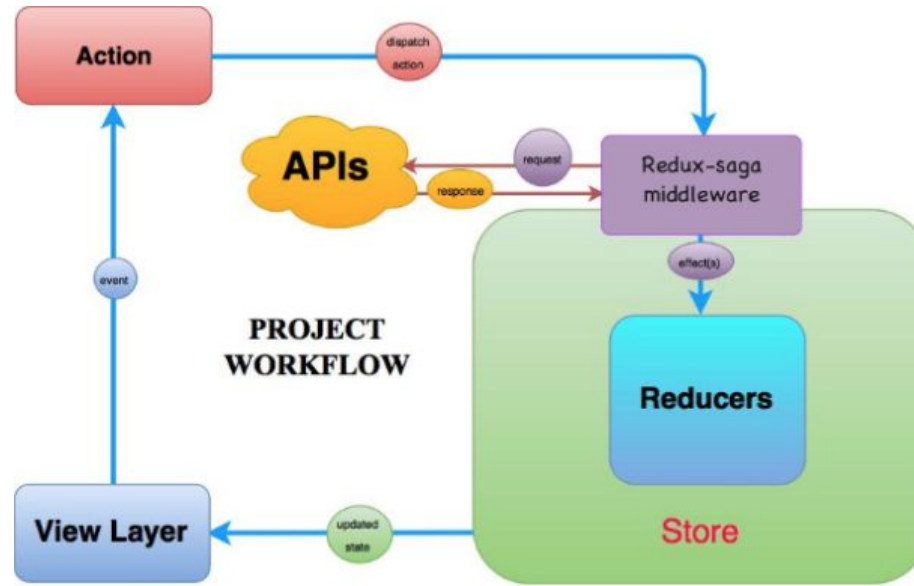
Normal functions return the result 1 time, but for generator functions they can be executed, interrupt the result return and continue the execution thanks to "yield".



## IV. Redux

### 1) Redux - saga:

#### c) Dataflow:



## IV. Redux

### 1) **Redux - saga:**

#### **d) Why use redux-saga:**

\_Contrary to redux thunk, you don't end up in callback hell, you can test your asynchronous flows easily and your actions stay pure.

## IV. Redux

### 1) **Redux - saga:**

#### **e) Libraries for responding to actions:**

- Fork: Uses a non-blocking mechanism to call a function, often used to call functions that track action sagas.
- Take: This function is triggered when an action saga is dispatched.
- TakeEvery: similar to Take, which tracks a specific action change and then calls a specific saga (same take, but can be called many times - blocking), a combination of Take and Fork
- TakeLastest: Executes a series of actions and returns the result of the last action.
- Delay: used to delay the execution of the saga (blocking)
- Put: Used to dispatch an action (non-blocking)
- All: Call multiple sagas for simultaneous execution.

## IV. Redux

For example:

Install

```
> npm i redux-saga
```

JS configReducers.js U X

src > redux > JS configReducers.js > [e] default

```
1 import { applyMiddleware, combineReducers, createStore } from "redux"; 4K (gzipped: 1.6K)
2 import reduxThunk from 'redux-thunk'; 557 (gzipped: 327)
3 import TodoListReducer from "./reducers/TodoListReducer";
4 import createMiddlewareSaga from "redux-saga"; 12K (gzipped: 4.7K)
5 import {rootSaga} from './saga/rootSaga'
6
7 const configReducers = combineReducers({
8   | TodoListReducer
9   |});
10
11 const middlewareSaga = createMiddlewareSaga()
12
13 let store = createStore(configReducers, applyMiddleware(reduxThunk, middlewareSaga));
14 middlewareSaga.run(rootSaga);
15
16 export default store;
```

## IV. Redux

For example:

JS TodoList.js U X

src > components > TodoList > JS TodoList.js > TodoList > [🔍] handleSubmit > 🔑 type

```
53   const getTaskListApi = () => {  
54     dispatch({  
55       type: 'getTaskListAction',  
56     })  
57   }  
58   useEffect(() => {  
59     getTaskListApi()  
60   }, [])
```

## IV. Redux

For example:

```
JS TodoListSaga.js U X
src > redux > saga > JS TodoListSaga.js > addTaskAction
1 import Axios from "axios" 16.8K (gzipped: 5.9K)
2 import { call, takeLatest, put } from 'redux-saga/effects' 2.9K (gzipped: 1.3K)
3 import { GET_TASK_LIST } from '../types/TodoListTypes';
4
5 function * getTaskListAction() {
6   try {
7     let {data, status} = yield call (() => {
8       return Axios({
9         url: 'http://svcy.myclass.vn/api/ToDoList/GetAllTask',
10        method: 'GET'
11      })
12    })
13    if(status === 200) {
14      yield put({
15        type: GET_TASK_LIST,
16        taskLists: data
17      })
18    }
19  } catch(err) {
20    console.log("🚀 ~ file: getTaskListAction.js ~ line 5 ~ *getTaskListAction ~ err", err)
21  }
22 }
```

## IV. Redux

For example:

JS *TodoListReducer.js* U X

src > redux > reducers > JS *TodoListReducer.js* > ...

```
1  import { ADD_TASK, EDIT_TASK, GET_TASK_LIST } from "../types/ToDoListTypes";
2
3  let initialState = {
4    todoLists: [],
5  }
6
7  const TodoListReducer = (state = initialState, action) => {
8    switch (action.type) {
9      case GET_TASK_LIST: {
10        state.todoLists = action.taskLists;
11        return {...state}
12      }
13    }
14  }
```

## IV. Redux

For example:

JS rootSaga.js U X

src > redux > saga > JS rootSaga.js > rootSaga

```
1  import {all} from 'redux-saga/effects' 625 (gzipped: 370)
2  import * as TodoListSaga from './TodoListSaga';
3
4  export function * rootSaga() {
5    yield all([
6      TodoListSaga.getTaskListApi(),
7      TodoListSaga.addTaskApi()
8    ])
9  }
```



## V. ReactJS Fundamentals

- 1) **React is not a Framework**
- 2) **JSX IN REACT**
- 3) **Component**
- 4) **State**
- 5) **Props**
- 6) **Virtual DOM**
- 7) **HOOKS**

# V. ReactJS Fundamentals

## 1. **React is not a Framework:**

+React is not a framework, the framework is a complete solution. Everything we need to make your application is built-in. A framework usually wants us to code everything a certain way. If we try to code with other way, the frameworks will fight with us.

+But in react, we don't have everything to build our application. We have to install another library to solve our problem. Libraries are more flexible to work with.

# V. ReactJS Fundamentals

## 2. JSX IN REACT:

+We can make HTML elements by using Javascript. But it's very hassle to do it. By the way, ReactJS make it easy for us. We can make HTML elements by using JSX.

+JSX syntax look like HTML syntax, Before displaying in the browser, JSX is compiled by a compiler to React.JSX is power full more then Javascript template string.

+Some syntax of JSX:

```
<MyButton color="blue" shadowSize={2}>
```

Click Me

```
</MyButton>
```

\*SelfClosingElement:

```
<div className="sidebar" />
```

# V. ReactJS Fundamentals

## 2. **JSX IN REACT:**

+JSX is not HTML, so don't be mistake, example like this:

In HTML we have :

```
<div class="container">...</div>
```

In JSX we have:

```
<div className="container">...</div>
```

# V. ReactJS Fundamentals

## 2. JSX IN REACT:

+Why we should use JSX instead of JS, this is 2 example:

\*In JSX

```
<div className="red">Children Text</div>;  
<MyCounter count={3 + 5} />;  
var gameScores = {  
  player1: 2,  
  player2: 5};  
<DashboardUnit data-index="2">  
<h1>Scores</h1>  
<Scoreboard className="results" scores={gameScores} />  
</DashboardUnit>;
```

\*In JS:

```
React.createElement("div", {className: "red"}, "Children Text");  
React.createElement(MyCounter, {count: 3 + 5});  
React.createElement(DashboardUnit, {"data-index": "2"},  
  React.createElement("h1", null, "Scores"),  
  React.createElement(Scoreboard, {className: "results", scores: gameScores})  
);
```

# V. ReactJS Fundamentals

## 2. JSX IN REACT:

+Embed expression:

— We can embed any JS expression in JSX by put it in {}

— Ex:

```
function formatName(user) {  
  return user.firstName + ' ' + user.lastName;  
}
```

```
const user = {  
  firstName: 'Harper',  
  lastName: 'Perez'  
};
```

```
const element = (  
  <h1>  
    Hello, {formatName(user)}!  
  </h1>  
);
```

```
function getGreeting(user) {  
  if (user) {  
    return <h1>Hello, {formatName(user)}!</h1>;  
  }  
  return <h1>Hello, Stranger.</h1>;  
}
```

# V. ReactJS Fundamentals

## 2. JSX IN REACT:

+Embed expression:

\_We also can use {} to embed expression in element:

\_Ex:

```
const element = <img src={user.avatarUrl}></img>;
```

\_JSX can have children like this:

```
const element = (  
  <div>  
    <h1>Hello!</h1>  
    <h2>Good to see you here.</h2>  
  </div>  
)
```

# V. ReactJS Fundamentals

## 3. **Component:**

\_ In react, we render UI by using components that are:

- Reusable
- Composable
- Stateful

\_ There are 2 kinds of components:

- Functional component: stateless, use Hook, short and simple
- Class component: stateful, much more complicate, have lifecycle, have initialization method, render(), state management.



## V. ReactJS Fundamentals

### 4. State:

\_ State is an object that holds some information that may change over the lifetime of the component.

\_ State only can use in class components. When state change, the component will rerender.

\_ State can use by hook useState in functional Component.

### 5. Props:

\_ React allows us to pass information to a Component using something called props .

\_ Props are pass from component father to children and props are read-only.

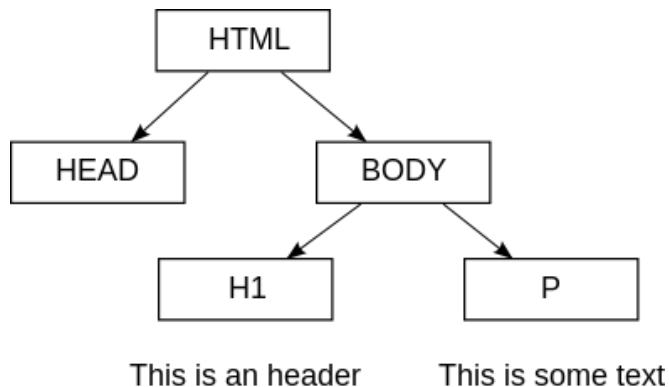
\_ Props can be used in both class-based and functional components.

# V. ReactJS Fundamentals

## 6. Virtual DOM:

\_Virtual DOM is a lightweight copy of actual DOM. When some parts of DOM are changes, react compares the changed DOM with Virtual DOM that is saved in the memory.

\_To compare, it uses the deffer algorithm. The algorithm can identify where changes happened and only update that part of the DOM. Changing or updating the whole DOM in a bit slower. Virtual DOM swiftly changes the DOM.



# V. ReactJS Fundamentals

## 7. Hook:

Hooks are a special kind of function that is used in react components.

All hooks function begins with the word “use”

React hook functions can only be used in functional components. You can't use them in class components.

Some kind of hooks:

- `useState`: to control state of component.

```
const [isLoading, setLoading] = useState(false);

onClick() {
  setLoading(true)
}
```

# V. ReactJS Fundamentals

## 7. Hook:

- `useEffect`: this one like lifecycle `ComponentDidMount`, `componentDidUpdate`, `componentWillUnmount`.

```
const [fullName, setFullName] = useState({name: 'name', familyName: 'family'});
```

```
+componentDidMount:
```

```
  useEffect( () => {  
    console.log(fullName.name);  
  },[]);
```

```
+componentDidUpdate:
```

```
  useEffect( () => {  
    setFullName({name:'acb', familyName:'xyz'});  
  },[fullName.name]);
```

```
+componentWillUnmount:
```

```
  useEffect( () => {  
    Return () => {function()}  
  },[])
```

# V. ReactJS Fundamentals

## 7. Hook:

- useMemo: like lifecycle shouldComponentUpdate. By the way add to second param then when it change, useMemo will be activated. useMemo returns the results of that function execution and will store it in memory to prevent the function from running again if the same parameters were used.

```
const [count, setCount] = useState(0);
const [todos, setTodos] = useState([]);
const calculation = useMemo(() => expensiveCalculation(count), [count]);

const increment = () => {
  setCount((c) => c + 1);
};

const expensiveCalculation = (num) => {
  console.log("Calculating...");
  for (let i = 0; i < 1000000000; i++) {
    num += 1;
  }
  return num;
};
```

# V. ReactJS Fundamentals

## 7. Hook:

- `useCallback`: like `useMemo` but the main difference is that `useMemo` returns a memoized value and `useCallback` returns a memoized function.

```
const [num, setNum] = useState(0);
const [light, setLight] = useState(true);
const plusFive = () => {
  console.log("I was called!");
  return num + 5;
};
return (
  <div className={light ? "light" :
"dark"}>
    <div>
```

Current number: 0, Plus f  
5

Update Number

Toggle the light

```
const App = () => {
  const [num, setNum] = useState(0);
  const [light, setLight] = useState(true);
  const plusFive = useCallback(() => {
    console.log("I was called!");
    return num + 5;
  }, [num]);
  return (
    <div className={light ? "light" :
"dark"}>
      <div>
```

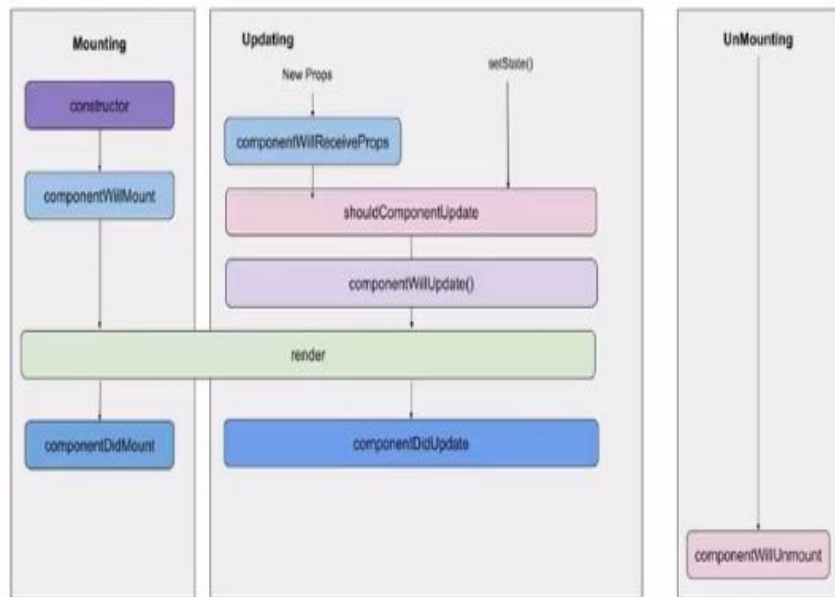
Current number: 0, Plus five  
5

Update Number

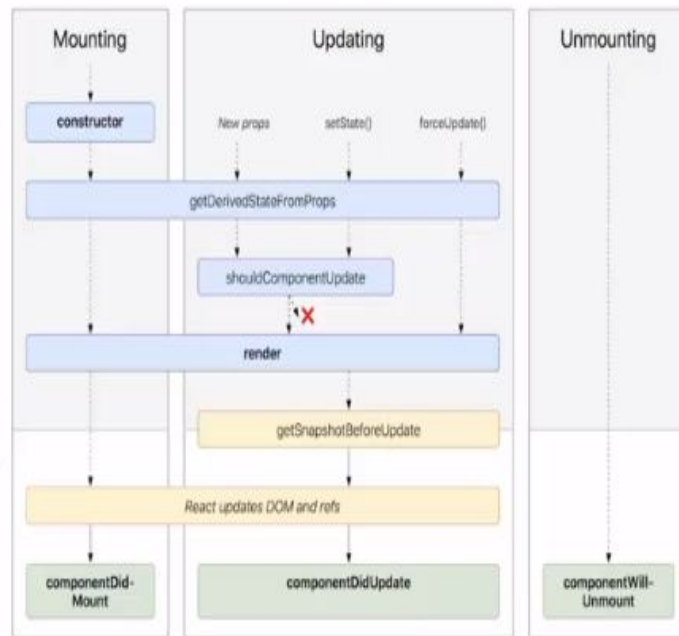
Toggle the light

# V. ReactJS LifeCycle Components


React 16.0 Life Cycle



React phiên bản 16.0



React phiên bản từ 16.4+ trở đi

A person with long, dark, wavy hair is shown from the chest up, aiming a handgun with both hands. They are wearing a dark, long-sleeved shirt. The background is dark and out of focus. The text is overlaid on the left side of the image.

Hãy viết code như thể người maintain là  
một đứa sát nhân điên cuồng biết địa  
chỉ nhà của bạn

**Khuyết danh**





**THANKS FOR WATCHING**