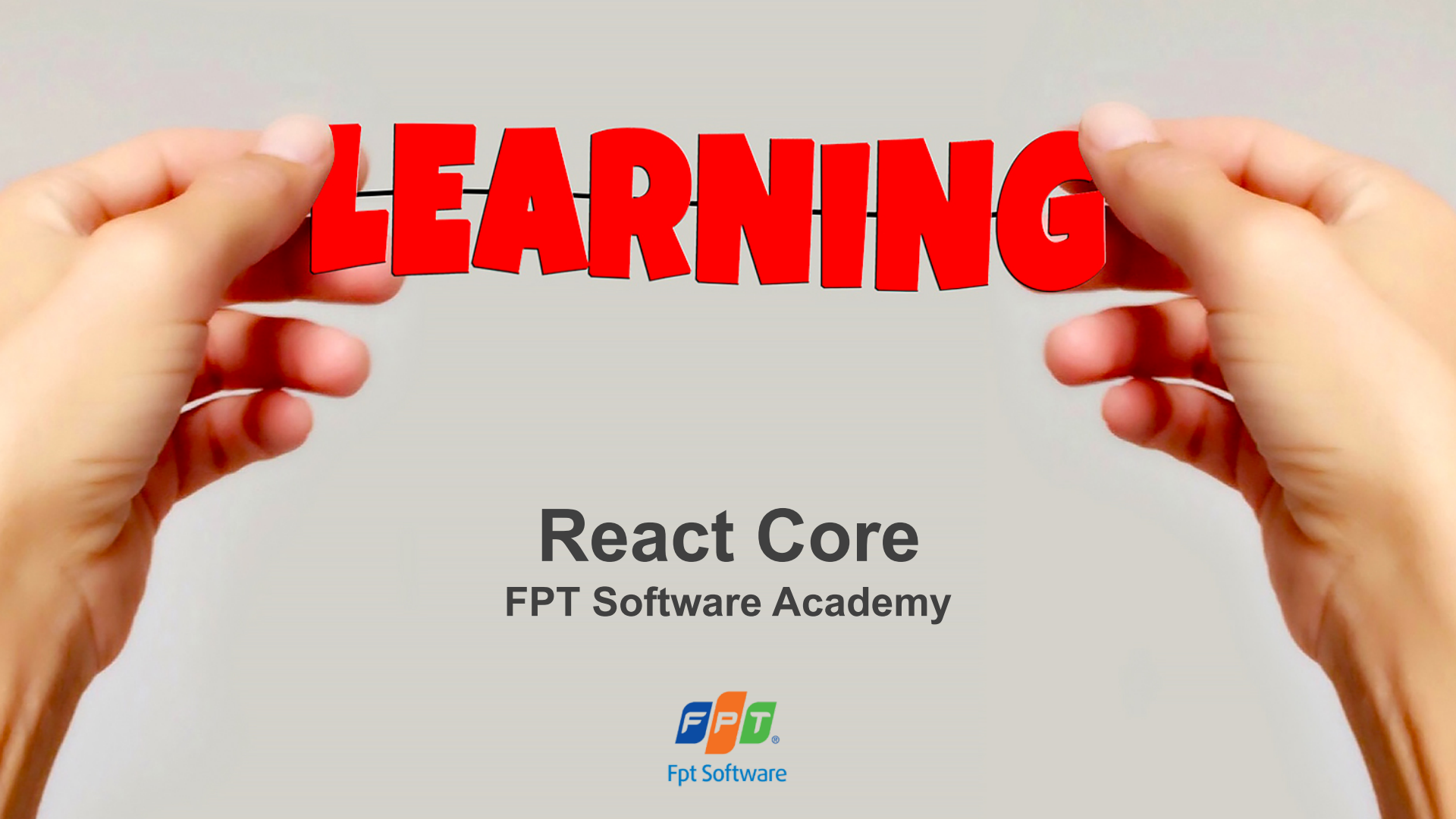


React Core

FPT Software Academy

Fpt Software

Contents

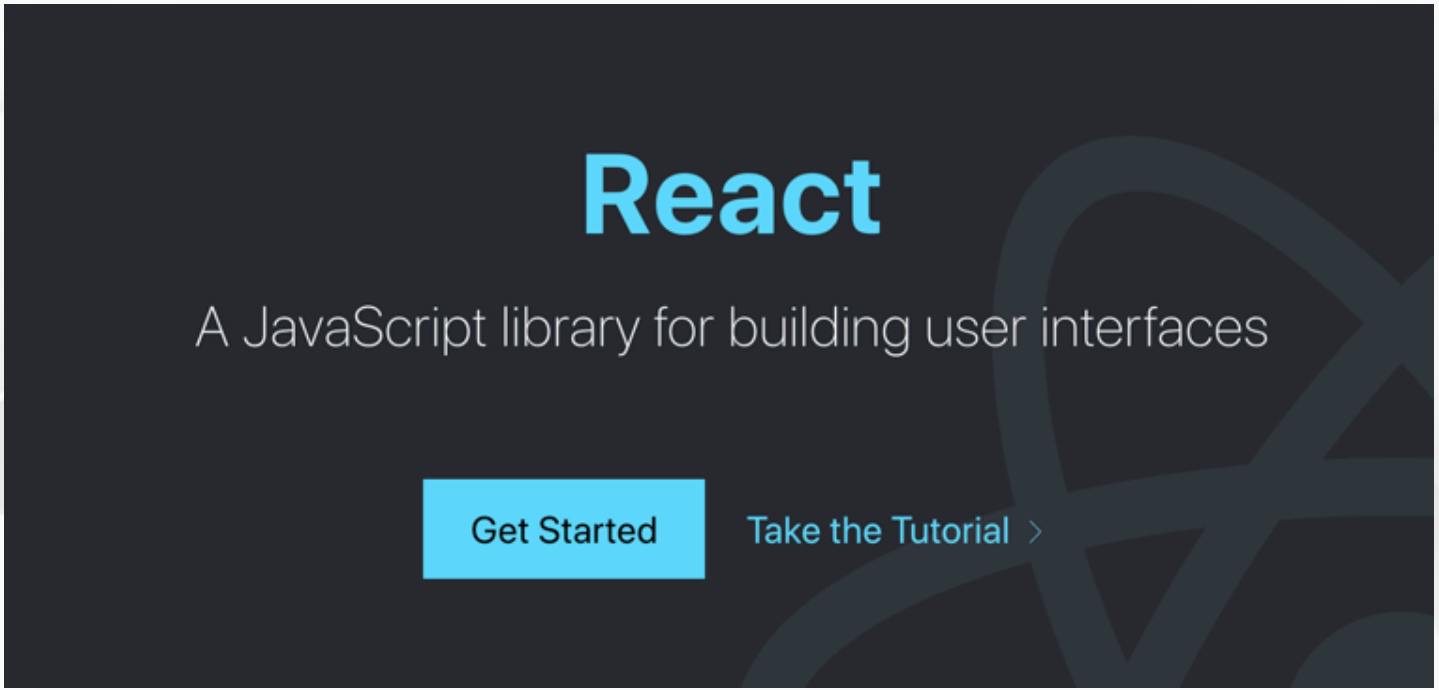
Fpt Software

1. React Overview

1. React Overview

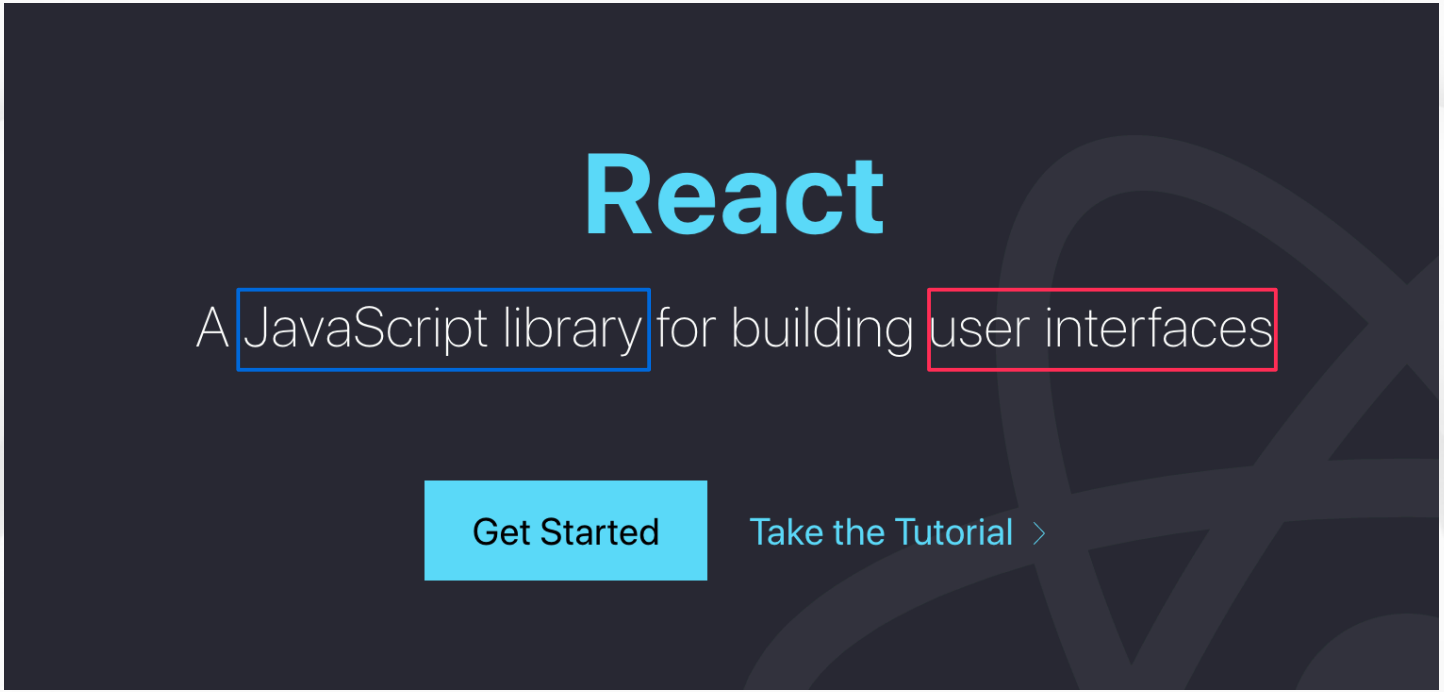
Fpt Software

- What is React ?



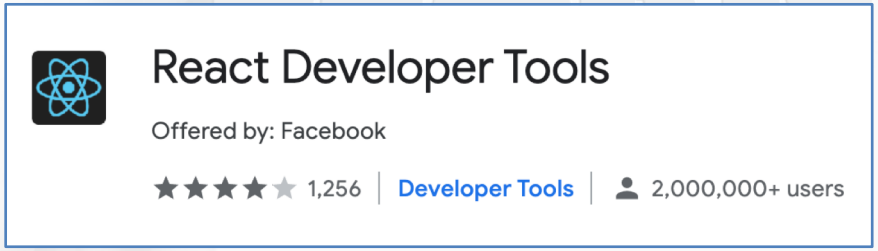
1. React Overview

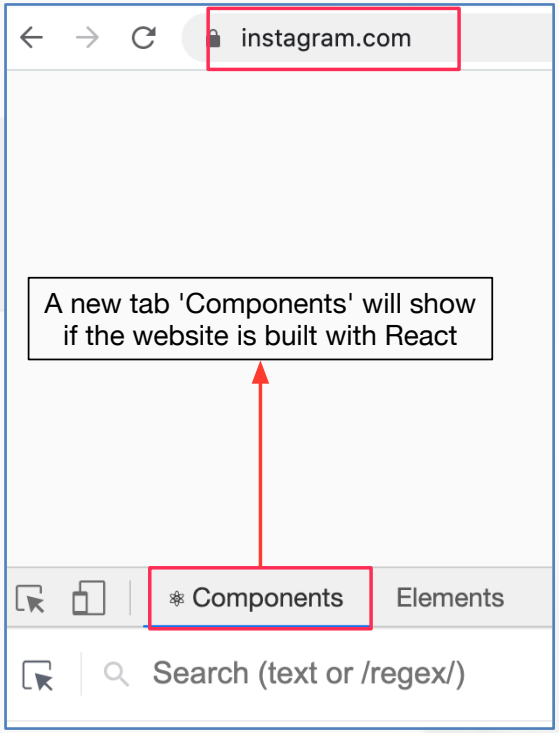
- What is React ?



1. React Overview

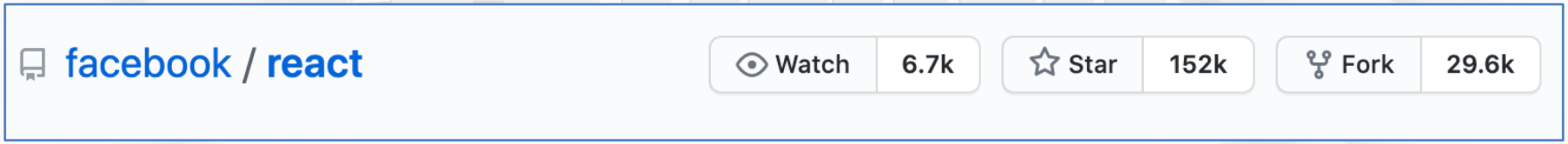
- To check a site is built with React ?





1. React Overview

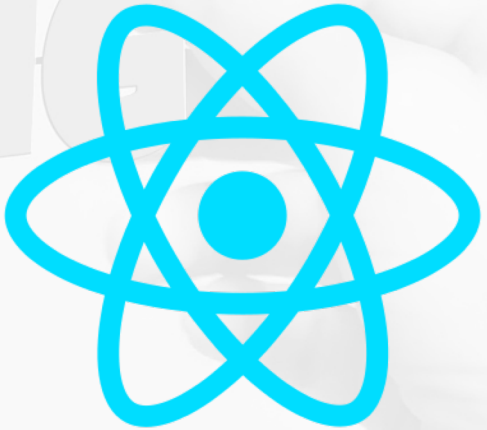
- Why React ?
  - Created by Facebook and maintained by community



  - Easy to learn and use
  - Fast, scalable and simple

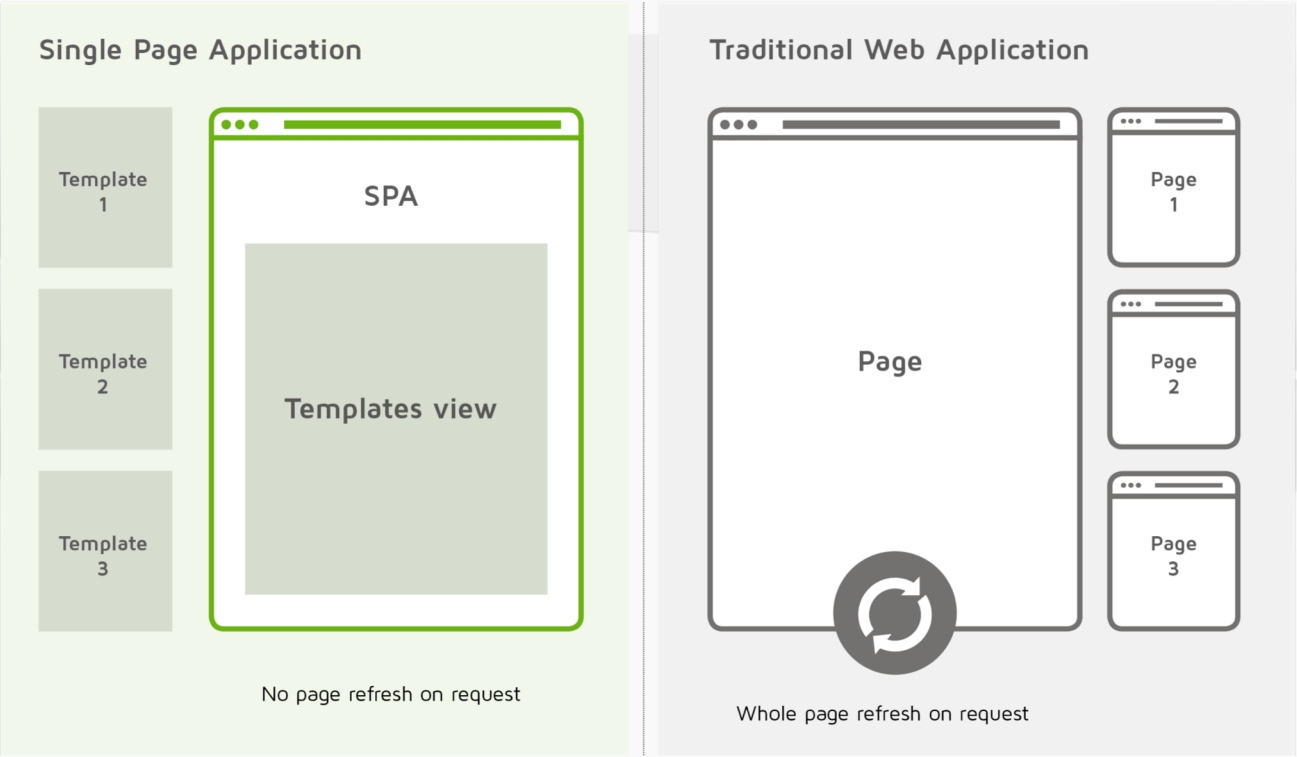
1. React Overview

- React alternatives?



1. React Overview

- SPA and MPA?



1. React Overview

- How to add React to your website ?
  1. Using CDN

```
<!-- Load React. -->
<!-- Note: when deploying, replace "development.js" with "production.min.js". -->
<script
  src="https://unpkg.com/react@16/umd/react.development.js"
  crossorigin
></script>
<script
  src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"
  crossorigin
></script>
```

1. React Overview

- How to add React to your website ?
  1. Using CDN
  2. Add custom js file

```
<script src="main.js" defer></script>
```

1. React Overview

- How to add React to your website ?
  1. Using CDN
  2. Add custom js file
  3. Add an HTML tag container for React to render

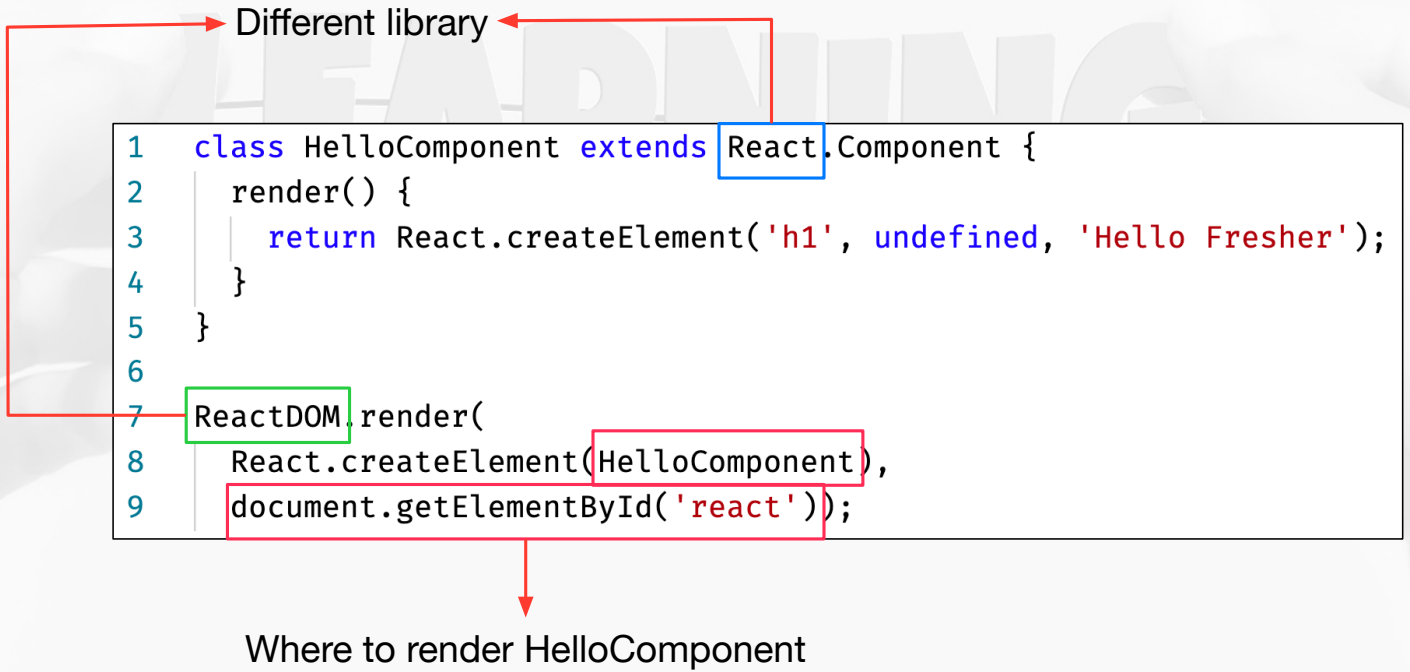
```
<body>
  <div id="react"></div>
</body>
```

1. React Overview

- How to add React to your website ?
  1. Using CDN
  2. Add custom js file
  3. Add an HTML tag container for React to render
  4. Add React code

1. React Overview

- How to add React to your website ?



2. Component

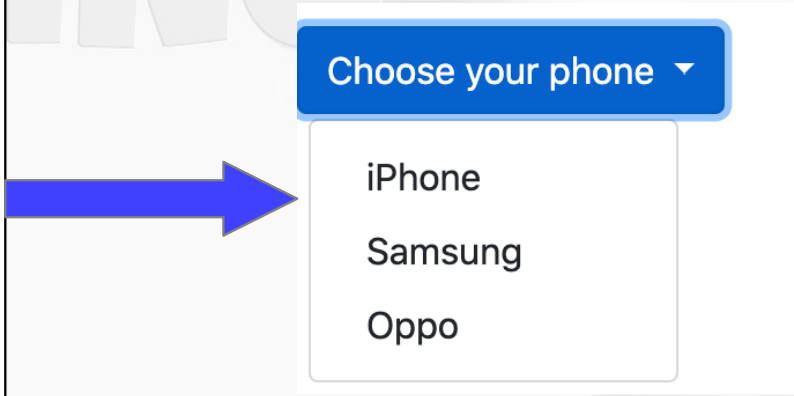
2. Component

- What is a Component ?
  - What if we want a Dropdown feature (say to let user choose his favo rite phone)

2. Component

- Dropdown of favorite Phone

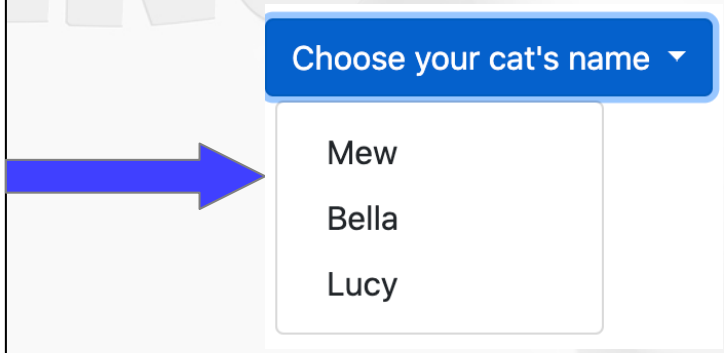
```
<div class="dropdown">
  <button
    class="btn btn-primary dropdown-toggle"
    type="button"
    id="dropdownMenuButton"
    data-toggle="dropdown"
    aria-haspopup="true"
    aria-expanded="false"
  >
    Choose your phone
  </button>
  <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
    <a class="dropdown-item" href="#">iPhone</a>
    <a class="dropdown-item" href="#">Samsung</a>
    <a class="dropdown-item" href="#">Oppo</a>
  </div>
</div>
```



2. Component

- Then Dropdown of favorite Cat name

```
<div class="dropdown">
  <button
    class="btn btn-primary dropdown-toggle"
    type="button"
    id="dropdownMenuButton"
    data-toggle="dropdown"
    aria-haspopup="true"
    aria-expanded="false"
  >
    Choose your cat's name
  </button>
  <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
    <a class="dropdown-item" href="#">Mew</a>
    <a class="dropdown-item" href="#">Bella</a>
    <a class="dropdown-item" href="#">Lucy</a>
  </div>
</div>
```



2. Component

- Do you spot the problem here ?

Dropdown Phone

```
<div class="dropdown">
  <button
    class="btn btn-primary dropdown-toggle"
    type="button"
    id="dropdownMenuButton"
    data-toggle="dropdown"
    aria-haspopup="true"
    aria-expanded="false"
  >
    Choose your phone
  </button>
  <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
    <a class="dropdown-item" href="#">iPhone</a>
    <a class="dropdown-item" href="#">Samsung</a>
    <a class="dropdown-item" href="#">Oppo</a>
  </div>
</div>
```

Dropdown Cat

```
<div class="dropdown">
  <button
    class="btn btn-primary dropdown-toggle"
    type="button"
    id="dropdownMenuButton"
    data-toggle="dropdown"
    aria-haspopup="true"
    aria-expanded="false"
  >
    Choose your cat's name
  </button>
  <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
    <a class="dropdown-item" href="#">Mew</a>
    <a class="dropdown-item" href="#">Bella</a>
    <a class="dropdown-item" href="#">Lucy</a>
  </div>
</div>
```

2. Component

- Do you spot the problem here ?

Dropdown Phone

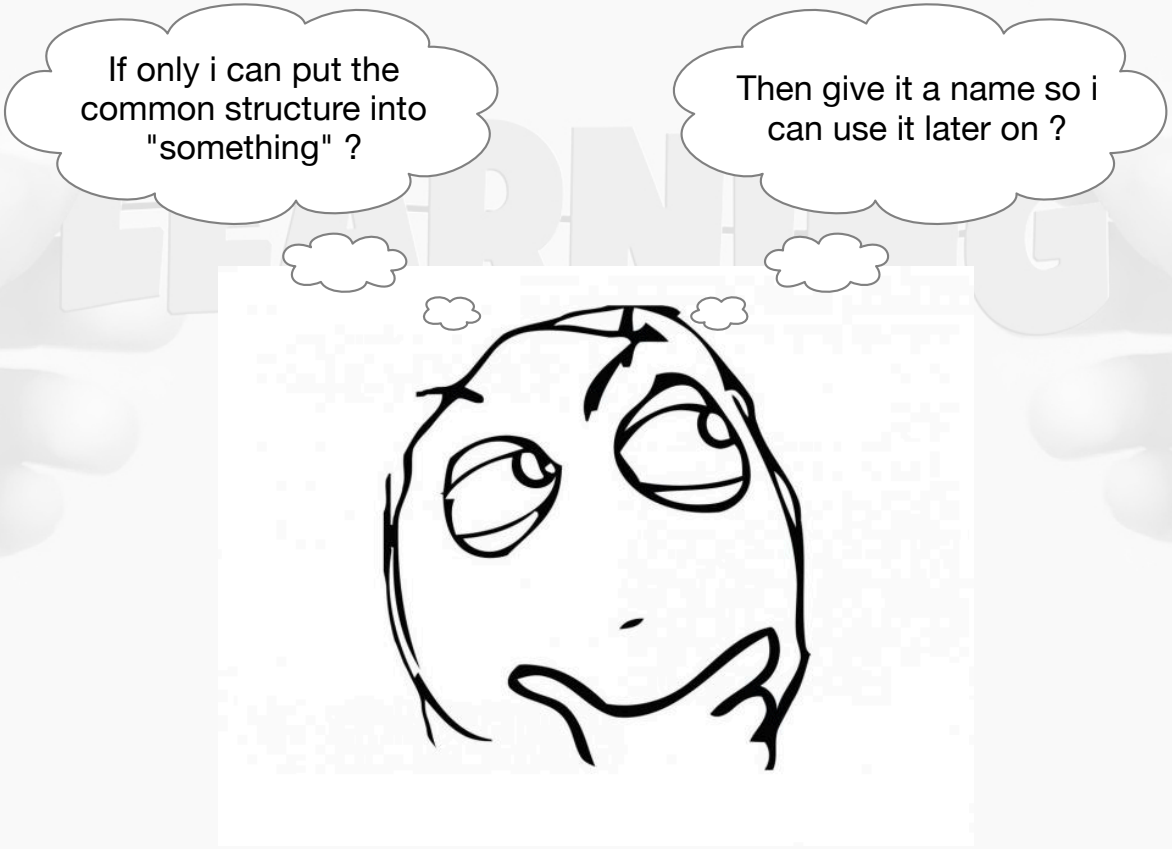
```
<div class="dropdown">
  <button
    class="btn btn-primary dropdown-toggle"
    type="button"
    id="dropdownMenuButton"
    data-toggle="dropdown"
    aria-haspopup="true"
    aria-expanded="false"
  >
    Choose your phone
  </button>
  <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
    <a class="dropdown-item" href="#">iPhone</a>
    <a class="dropdown-item" href="#">Samsung</a>
    <a class="dropdown-item" href="#">Oppo</a>
  </div>
</div>
```

Dropdown Cat

```
<div class="dropdown">
  <button
    class="btn btn-primary dropdown-toggle"
    type="button"
    id="dropdownMenuButton"
    data-toggle="dropdown"
    aria-haspopup="true"
    aria-expanded="false"
  >
    Choose your cat's name
  </button>
  <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
    <a class="dropdown-item" href="#">Mew</a>
    <a class="dropdown-item" href="#">Bella</a>
    <a class="dropdown-item" href="#">Lucy</a>
  </div>
</div>
```

Same structure and CSS but different data

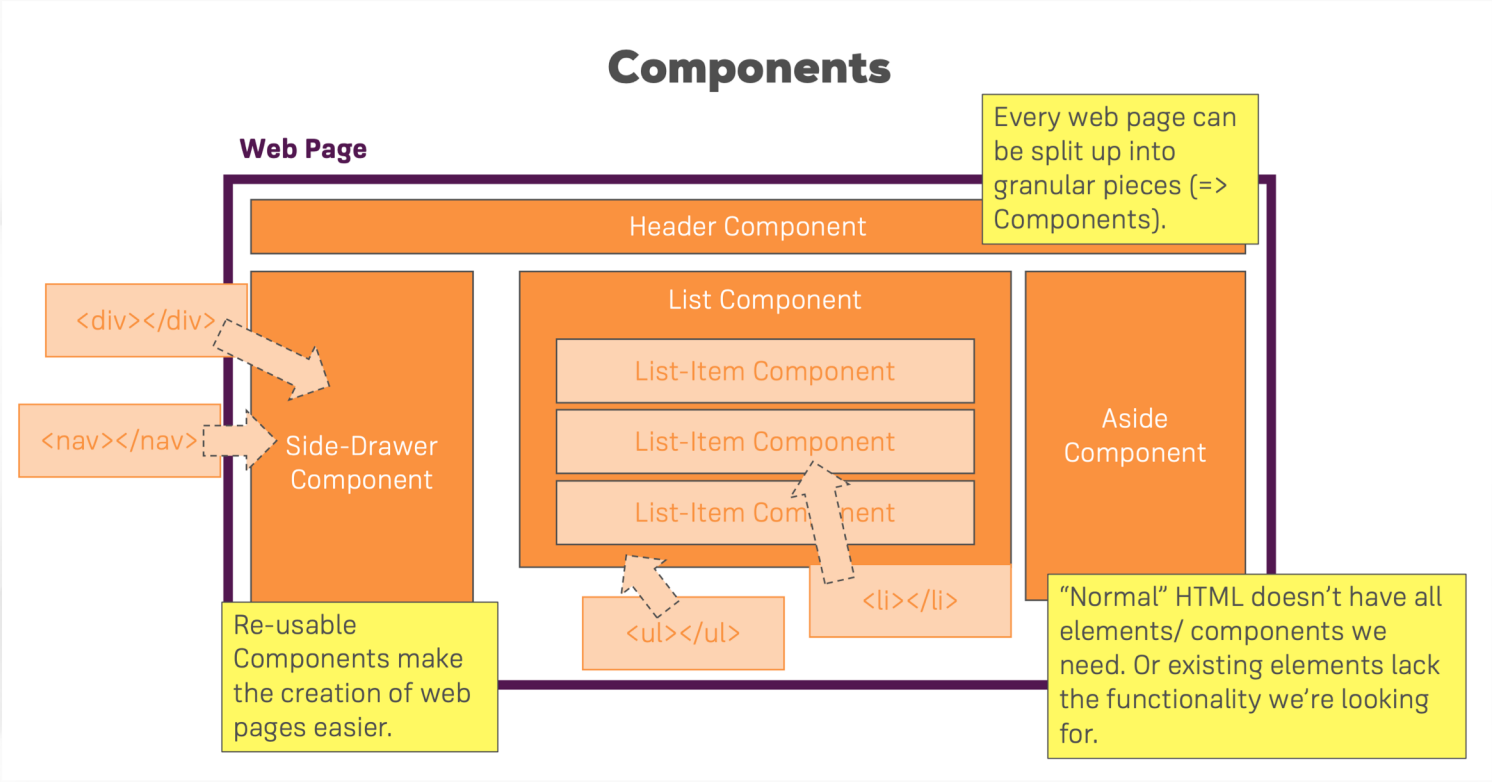
2. Component


If only i can put the common structure into "something" ?
Then give it a name so i can use it later on ?

2. Component

- That’s exactly what’s a Component is
- Components are independent and reusable bits of code (HTML, CSS and JS)

2. Component


Components
Web Page
Every web page can be split up into granular pieces (=> Components).
Header Component

<nav></nav>
Side-Drawer Component
List Component
List-Item Component
List-Item Component
List-Item Com nent
Aside Component

Re-usable Components make the creation of web pages easier.
"Normal" HTML doesn't have all elements/ components we need. Or existing elements lack the functionality we're looking for.

2. Component

Fpt Software

In React, there 2 types of Components:

Class Component

```
class HelloComponent extends React.Component {
  render() {
    return React.createElement('h1', undefined, 'Hello Fresher');
  }
}
```

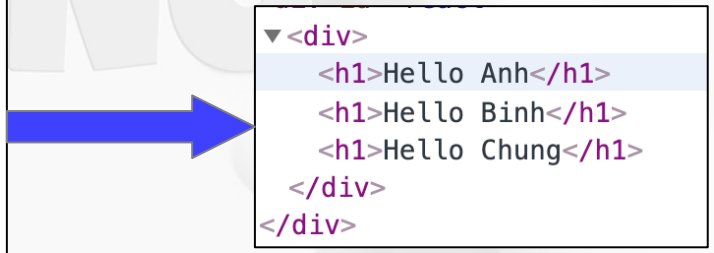
Function Component

```
const HelloComponent = () => {
  return React.createElement('h1', undefined, 'Hello Fresher');
};
```

3. Core Concepts - JSX

- Check out the source code below:

```
class HelloComponent extends React.Component {
  render() {
    return React.createElement('div', undefined, [
      React.createElement('h1', undefined, 'Hello Anh'),
      React.createElement('h1', undefined, 'Hello Binh'),
      React.createElement('h1', undefined, 'Hello Chung'),
    ]);
  }
}
```

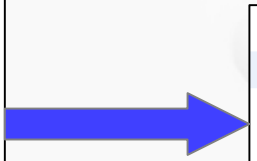


3. Core Concepts - JSX

- Check out the source code below:

Lot of works just to a simple feature

```
class HelloComponent extends React.Component {
  render() {
    return React.createElement('div', undefined, [
      React.createElement('h1', undefined, 'Hello Anh'),
      React.createElement('h1', undefined, 'Hello Binh'),
      React.createElement('h1', undefined, 'Hello Chung'),
    ]);
  }
}
```



```
<div>
  <h1>Hello Anh</h1>
  <h1>Hello Binh</h1>
  <h1>Hello Chung</h1>
</div>
```

3. Core Concepts - JSX

- Declarative, they say:

> ## Declarative
>
> React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.
>
> Declarative views make your code more predictable and easier to debug.

3. Core Concepts - JSX

- You need JSX: allow you to write HTML code in JS

JS Code

```
class HelloComponent extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello Anh</h1>
        <h1>Hello Binh</h1>
        <h1>Hello Chung</h1>
      </div>
    );
  }
}
```

3. Core Concepts - JSX

- For JSX to work you must add Babel JSX support
  1. Add Babel CDN
  2. Add type="text/babel" to your script tag
- Or use full feature support with **Create-React-App**

3. Core Concepts - JSX

- If you want to add variable to HTML use { } syntax

```
class App extends Component {
  render() {
    const name = 'Anh';

    return <div className="App">Hello {name}</div>;
  }
}
```

3. Core Concepts – Props

- You have wrapped the common code into Component
- Now you want to parameterize the data

```
class App extends Component {
  render() {
    const catNames = ['Mew', 'Bella', 'Lucy'];
                     pass
    return <div className="App">
      <Dropdown />
    </div>;
  }
}
```

3. Core Concepts – Props

- That's what Props are

```
class App extends Component {
  render() {
    const catNames = ['Mew', 'Bella', 'Lucy'];

    return <div className="App">
      <Dropdown lists={catNames} />
    </div>;
  }
}
```

send value via props

```
interface DropdownProps {
  lists: string[];
}

export class Dropdown extends Component<DropdownProps> {
  render() {
    const { lists } = this.props;
```

Access lists

3. Core Concepts – Props

- Special children props allow you to passed pass children elements directly

```
class App extends Component {
  render() {
    const catNames = ['Mew', 'Bella', 'Lucy'];

    const phones = ['iPhone', 'SamSung', 'Oppo'];

    return <div className="App">
      <Dropdown lists={catNames}>
        <span>Catname</span>
      </Dropdown>

      <Dropdown lists={phones}>
        <span>Phones</span>
      </Dropdown>
    </div>;
  }
}
```

```
render() {
  const { lists, children } = this.props;

  return (
    <div className="dropdown">
      <button
        className="btn btn-primary dropdown-toggle"
        type="button"
        id="dropdownMenuButton"
        data-toggle="dropdown"
        aria-haspopup="true"
        aria-expanded="false"
      >
        {children}
      </button>
```

3. Core Concepts – Props

- You can only pass 1 thing down with children
- In case, you have multiples value use props
- props **should** not change

3. Core Concepts – State

- When a component needs to keep track of information between renderings the component *itself* can create, update, and **use** state.
- **state** is created in the component's constructor
- **state** is changeable via setState
- **setState** is asynchronous

3. Core Concepts – State

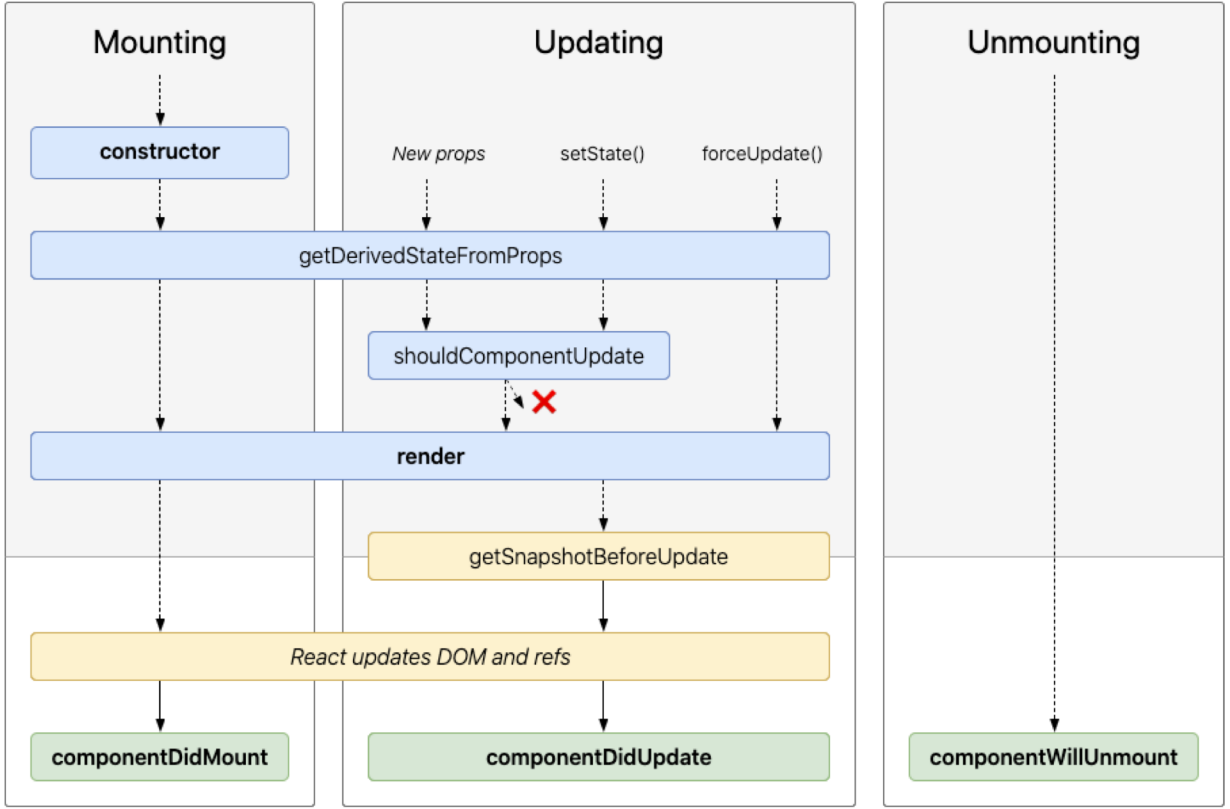
- **Warning:**

1. Do not update state directly:

```
this.state.count = this.state.count + 1
```

2. Do not use: this.setState({ count: this.state.count + 1 });

3. Core Concepts – Lifecycle



3. Core Concepts – Lifecycle

- The following lifecycle methods are of interest:

1. **componentDidMount**: runs right after a component is mounted, setting state here triggers re-rendering
2. **shouldComponentUpdate**: determines whether changes in props and state warrants re-render, class components return true by default
3. **componentWillUnmount**: runs right before a component is destroyed, can do any necessary cleanup such as canceled network requests
4. **componentDidCatch**: catches exceptions from child components, unhandled exceptions unmount the component

3. Core Concepts – Handling Events

- Handling events with React elements is very similar to handling events on DOM elements. There are some syntax differences:
  1. React events are named using camelCase, rather than lowercase.
  2. With JSX you pass a function as the event handler, rather than a st ring.

3. Core Concepts – Conditional Rendering

- Conditional rendering in React works the same way conditions work in JavaScript
- Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them.

3. Core Concepts – Render Lists

- Remember how to transform list in JavaScript ?
- In React, we do the same when work with Lists
- Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity

4. Summary

- In React, you need to think about Component (it's the building block for your app)
- In the Component code, always think about the data model first then its representation to DOM via **render()** method

Happy Coding!

Fpt Software