React

CS568 – Web Application Development I

Assistant Professor Umur Inan

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

Content

- React overview
- Create the first React app
  - App.js
  - Package.json
- React Element
- React Component
- JSX

create-react-app

- It sets up the development environment so that we can use the latest JavaScript features and optimizes your app for production.
- Node >= 8.10 and npm >= 5.6
- it uses Babel and webpack

create-react-app

- npm i -g create-react app
- create-react-app my-first-app
- cd my-first-app
- npm start
- Browse : http://localhost:3000

What is React?

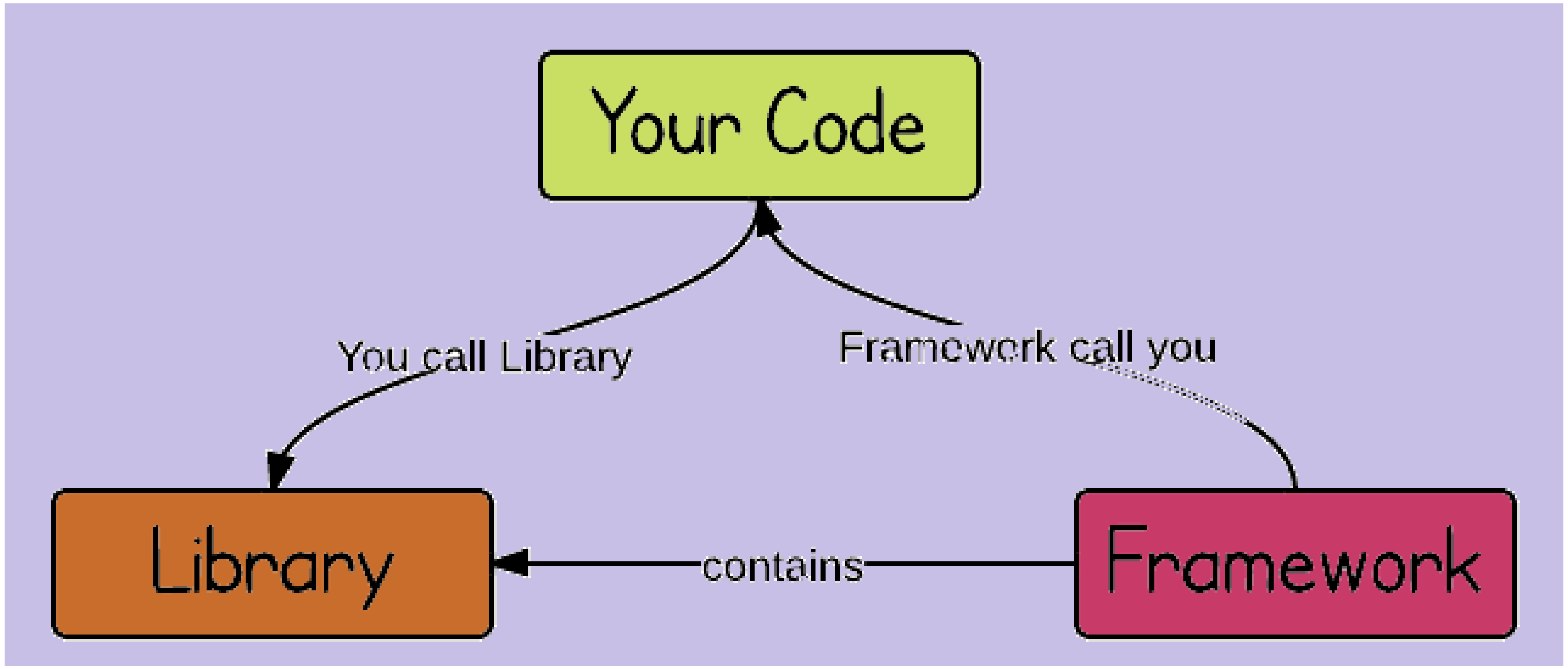
React is a JavaScript **library** for building **user interfaces**.

- One of the most popular libraries, with over 100,000 stars on GitHub.
- React is **not a framework** (unlike Angular).
- React is an **open-source** project created by **Facebook**.
- React is used to build user interfaces (**UI**) on the **front end**.
- React is the **view layer** of an MVC application (Model View Controller)

Library vs Framework

A **library** performs specific, well-defined operations.

A **framework** is a skeleton where the application defines the "meat" of the operation by filling out the skeleton.



What is React?

One of the most important aspects of React is the fact that you can **create components**.

Components are **custom**, **reusable** HTML elements to quickly and efficiently build user interfaces.

React also streamlines how data is stored and handled, using **state** and **props**.

Use create-react-app library to create the first React app.

Important Files

- **App.js**: This is the file for App Component. App Component is the main component in React which acts as a container for all other components.
- **Package.json**: This File has the list of node dependencies which are needed.

React Elements

An element is like a single frame in a movie. It represents the UI at a certain point in time.

```
function App() {
  return React.createElement('div', null,
    React.createElement('p', {className:’App’}, 'Hello
World. This is my first React App.'));
}
```

React.createElement()

It needs at least 3 arguments (component, props, ...children)

- The element we want to render to DOM
- Properties or an object for configuration
- Children

Configuration – Use camelCase naming standard:

- id
- className
- style

React Elements

- React elements are **immutable**. Once you create an element, you can’t change its children or attributes.
- The way to update the UI is to create a new element and pass it to *ReactDOM.render(element, root DOM)*.
- React Only Updates What’s Necessary - React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.
- Unlike browser DOM elements, React elements are plain objects, and are cheap to create. React DOM takes care of updating the DOM to match the React elements.

JSX

JSX just provides syntactic sugar for the React.createElement function. It is NOT a HTML. It is javascript!

```
function App() {
 return (
   <div className="App">
       <p>
         Hello World. This is my first React App.
       </p>
   </div>
 );
}
```

JSX

- User-Defined Components Must Be Capitalized.
- When an element type starts with a lowercase letter, it refers to a built-in component like <div> or <span> and results in a string 'div' or 'span' passed to React.createElement
- Must return one parent item. Not more than one.
- JSX Prevents Injection Attacks - Everything is converted to a string before being rendered. This helps prevent XSS (cross-site-scripting) attacks.

Embedding Expressions in JSX

Use curly bracket to refer a variable or call a function.

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Returning Multiple Elements

Wrap components and other HTML elements in a div

```
function App() {
 return (
   <div className="App">
       <p>
         Hello World. This is my first React App.
       </p>
       <p>
         It is fun !!!
       </p>
   </div>
 );
}
```

Returning Multiple Elements

return an array of JSX elements

```
function App() {
 return (
   [
      <p>
         Hello World. This is my first React App.
       </p>,
       <p>
         It is fun !!!
       </p>
   ]
 );
}
```

Returning Multiple Elements

use Fragment

```
function App() {
 return (
   <Fragment>
      <p>
         Hello World. This is my first React App.
       </p>
      <p>
         It is fun !!!
       </p>
   </Fragment>
 );
}
```

Fragment motivation

Fragments let you group a list of children without adding extra nodes to the DOM.

```
class Table extends React.Component {
    render() {
      return (
        <table>
          <tr>
            <Columns />
          </tr>
        </table>
      );
    }
  }
```

Be Careful !

```
class Columns extends React.Component {
    render() {
      return (
        <div>
          <td>Hello</td>
          <td>World</td>
        </div>
      );
    }
  }
```

```
<!-- result -->
<table>
  <tr>
    <div>
      <td>Hello</td>
      <td>World</td>
    </div>
  </tr>
</table>
```

Solution with Fragment

```
render() {
    return (
      <React.Fragment>
        <td>Hello</td>
        <td>World</td>
      </React.Fragment>
    );
  }
}
```

```
<!-- result -->
<table>
  <tr>
    <td>Hello</td>
    <td>World</td>
  </tr>
</table>
```

React Components

- Building blocks of react app
- React separates concerns with loosely coupled units called “components” that contain both the markup (HTML) and logic (JS).
- Components let you split the UI into independent, reusable pieces.
- Components are “made of” elements.
- There are 2 types of components:
  - Functional – Stateless, dumb, presentational. Preferred.
  - Class – Stateful, smart, containers. Should override render() method.

Functional Components

- 90% cleaner code than class components.
- Class components are verbose.
- Class components get compiled. The compiled code could be messy.
- More **consistent** and easier to test.
- Class components are more complex.

Functional Components

- Purely presentational
- Represented by a function
- Returns React element
- Aka stateless, dumb, presentational

Class-Based Component

- Inherits from React.Component
- Should override render() method
- Aka containers, smart, stateful

Functional and Class Components

```
function Welcome() {
  return <h1>Hello world!</h1>;
}
```

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello world!</h1>;
  }
}
```

Rendering Components

```
// Element
const element = <div />;
```

```
// Component
const element = <Welcome />
;
```

```
function Welcome(props) {
  return <h1>Hello world!</h1>;
}

const element = <Welcome />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Extracting Components

Don’t be afraid to split components into smaller components!

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      ...
```

Creating an Avatar component

```
function Avatar(props) {
    return (
      <img className="Avatar"
        src={props.user.avatarUrl}
        alt={props.user.name}
      />
    );
  }
```

Including the Avatar component

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <Avatar user={props.author} />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
        …
```