



# React pt. 1: Intro, JSX, Props and State

Kianoosh Abbasi

CSC309 Winter 2022

Some content is from Dr. Sadia Sharmin's slides of CSC309 Winter 2021: [www.rainsharmin.com](http://www.rainsharmin.com)

# So far

- HTML, CSS, and Django backend
- JavaScript front-end
  - Basics: objects, functions, events
  - DOM
  - jQuery
  - Ajax
  - Advanced topics: closures, arrow functions, promises

# This session

- Single-page applications
- Intro to React
- JSX
- Props, state, events

# Web applications so far

- A **backend** server **listens** for requests
- Upon entering the **URL**, the **browser** sends a request
- Server returns an **HTML** page in response  
Containing many **links** to other **static** files (js, css, image, etc.)
- **Separate** requests sent for static files
- Browser **renders** the HTML and CSS, and **runs** the scripts

# Web applications so far

- Each **link** or form **submission** yields to a new **webpage**  
The same **scenario** once again
- But does it really have to be a **reload** for each **page**?
- Short answer: **No!**  
Example: **Ajax!**

# Web applications so far

- Ajax could send a request and handle response in background
- It could change the page accordingly!
- Ideally, we only need the hard URL reload only once  
Subsequent requests/renders can be handled by Ajax!

# Single-page applications

- **Seamless** user experience

Everything does not get **reset** every time

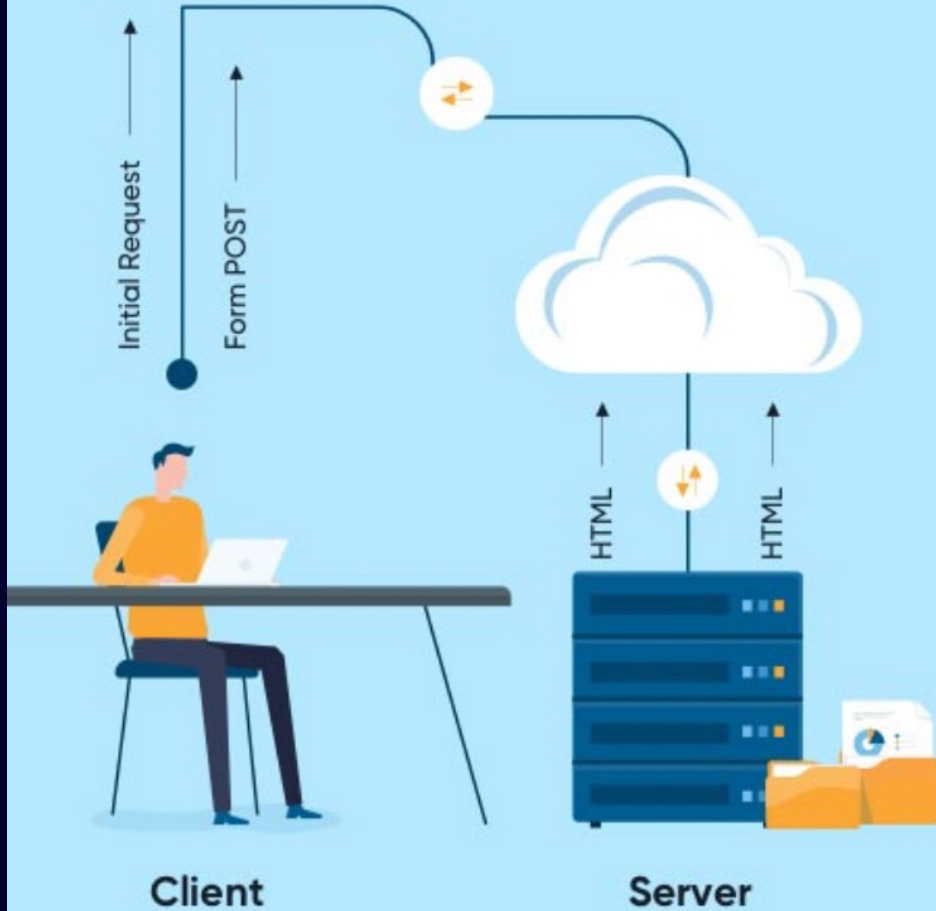
- **Efficiency**

The whole page does not get **updated**

- **Faster** load time

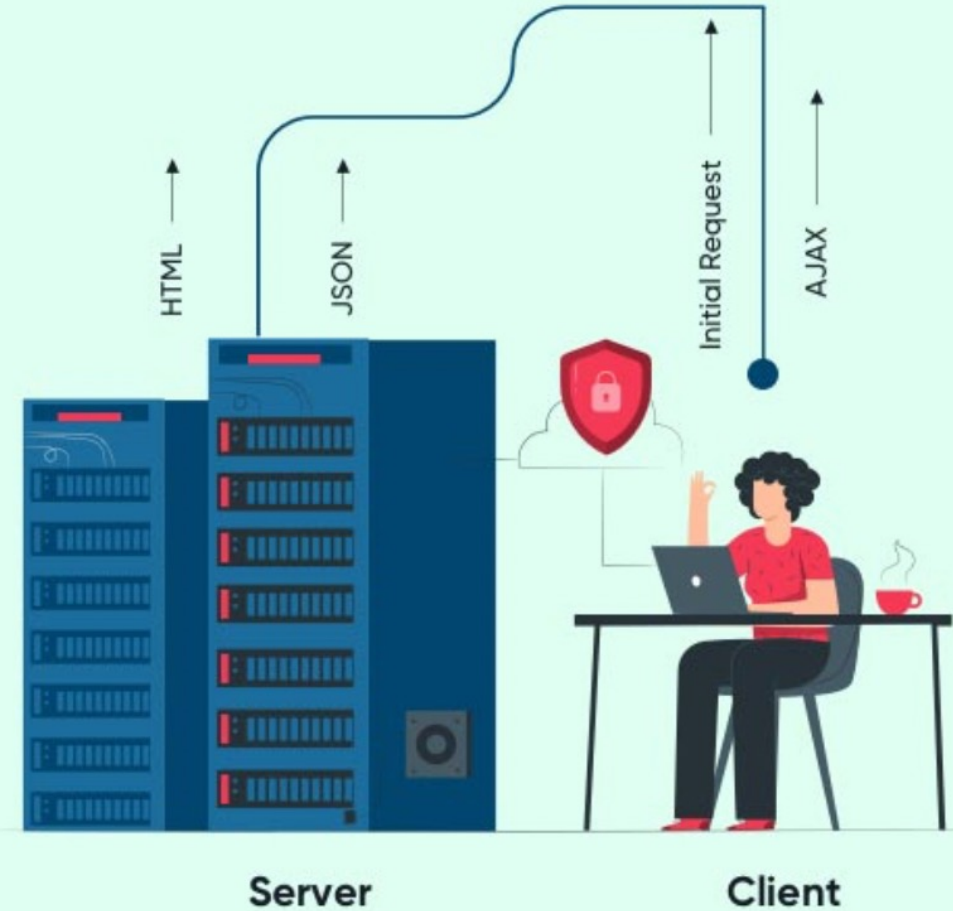
The initial load (when **nothing** is there) takes less time

## Traditional Page Lifecycle



Source: Microsoft

## SPA Lifecycle



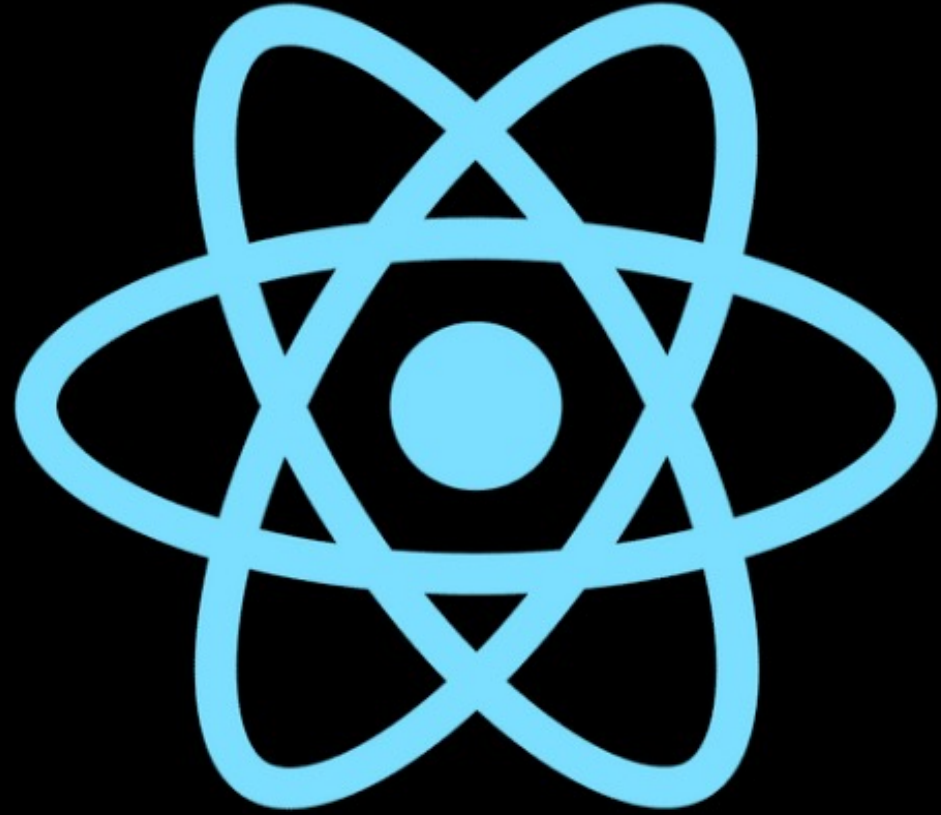
net solutions

Source: <https://www.netsolutions.com/insights/single-page-application/>



# Creating a single-page application

- **Nobody** does that with **pure** Ajax
- Many **frameworks** are out there to **help** you
- Another **advantage**: backend/frontend **separation**  
Lecture 6 recap: Front-end merits an **independent** project  
More on that next week
- Examples: **React**, Angular, Vue



**React JS**

# React

- Released by Facebook in 2013
- A JS library for building interactive user interfaces
- React takes charge of re-rendering when something changes
  - You no longer need to manipulate elements manually

# React

- Creates a **virtual DOM** in memory
- When **something** changes, it **re-renders** its own DOM  
More about the “**something**” later
- **Compares** the new and old DOMs and finds out what has been **updated**
- Updates the **specific** elements of the **browser's DOM**

# What's the point

- Updating and re-rendering the **actual DOM** is **expensive**
- **Not feasible** to re-render the **entire** page on every change
- This way, React **only** changes what **really** needs to change

# JSX

- **React** uses a special **variation** of JavaScript that allows for **merging** HTML and JS together
- Example:  

```
const element = <h1>Hello, world!</h1>;
```
- Browsers **do not** understand this syntax  
Should be **translated** before execution

# Translation

Visit <https://babeljs.io/>

## JSX

```
const element = <span className="red">Hello, world!</span>

const name = "Hello world";
const id = "div-1"

const element2 = (
  <p>
    <div id={id}>
      Hi, there is a {name} here!
    </div>
  </p>
)
```

## JS

```
"use strict";

const element = /*#__PURE__*/React.createElement("span", {
  className: "red"
}, "Hello, world!");
const name = "Hello world";
const id = "div-1";
const element2 = /*#__PURE__*/React.createElement("p", null,
  /*#__PURE__*/React.createElement("div", {
    id: id
  }, "Hi, there is a ", name, " here!"));
```

Note: these are React elements, not real JS elements

# Make it real

- **Import** React and Babel (JSX) scripts to your HTML

```
<script src="https://unpkg.com/react@17/umd/react.production.min.js"></script>  
<script src="https://unpkg.com/react-dom@17/umd/react-dom.production.min.js"></script>  
<script src="https://unpkg.com/@babel/standalone@7.16.8/babel.min.js"></script>
```

- **Render** your element in an **actual** JS element

```
<script type="text/babel">  
  
  const element = <h1>Hello World!</h1>;  
  ReactDOM.render(element, document.body)  
  
</script>
```



# Components

- **Key** concept in React
- Allows you to make your **elements** reusable
- It's a **function** or **class** that returns a JSX **element**
- Can be **re-used** like a known **tag**

# Function components

- Example:

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

- How to re-use it

```
ReactDOM.render(<SayHello />,  
  document.getElementById("root")  
)
```

- You can put any **JS statement** inside the **{}** in JSX
- **Singular** tags must always **end** with **/>**
- Components' **names** should always be **capitalized**  
Lowercase names are **reserved** for **built-in** elements: p, h1, div, etc.
- A JSX element must be **wrapped** in one **enclosing** tag  
If more than one, **wrap** them in a React **fragment**

```
<>
  <p>
    <div id={id}>
      Hi, there is a {name} here!
    </div>
  </p>
  
</>
```

# Props

- React mimics JS **attributes** via **props**  
**Read-only** data coming from the **parent** element
- A **dictionary** containing attributes

```
function Text(props) {  
  return <h4>{props.value}</h4>  
}
```
- To pass props:

```
<Text value="John" />
```

- **Styles** and **classes** are handled a bit **differently** in JSX

- **Example:**

```
function Text(props) {  
  return(  
    <h4 className="text" style={{fontSize: props.size}}>  
      {props.value}  
    </h4>  
  )  
}
```

- **To pass props:**

```
<Text value="Cars" size={30} />
```

- Can you think of a way to **simplify** the above **component**?  
Hint: Use **destructuring**

# A more sophisticated example

- Elements created in a **loop** must have a **unique key** prop
- **Identifies** which item has **changed**, is added, or is removed
- Otherwise, React will have to **re-render** the **whole** list if something **changes**

```
function List({ title, values }) {  
  return (  
    <>  
      <Text value={title} size={40} />  
      <ul>  
        {values.map((item, index) => (  
          <li key={index}>  
            {item}  
          </li>  
        ))}  
      </ul>  
    </>  
  )  
}
```

# Paired tag

- You can use your **component** as a **paired tag**
- What put **inside** tags will be **passed** as the **children** prop

```
function Wrapper({ children }) {  
  return <div className="col">  
    { children }  
  </div>;  
}  
  
const wrapped = (  
  <Wrapper>  
    <List values={[1, 2, 3, "my cat"]} />  
  </Wrapper>  
)
```

# Re-rendering and updates



# Class components

- Another way to define a **component**
- **Extends `React.Component`**; should **implement** the **render** method
- **Props** passed to **constructor**

- **Example:**

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}
```

# State

- Exhibits the **real power** of React!
- **Components** have a built-in **state**  
A **object** initialized in the **constructor**
- Once the state **changes**, component **re-renders**

# State

- Initialize the state object in the constructor

```
class Counter extends React.Component {  
  constructor(props){  
    super(props)  
    this.state = { counter: 0, }  
  }  
}
```

- State values can be accessed via this.state

```
render(){  
  return <h3>{this.state.counter}</h3>  
}
```

# Updating the state

- React states should **never** be **mutated**  
**Breaks** the underlying **assumptions** of React
- To **update** the state, call the **setState** method  
Other approaches will **not** trigger **re-render**
- Never **assign** state other than in the **constructor**

# Updating the state

- **Wrong way #1:**  
`this.state.counter += 1`
- **Wrong way #2:**  
`this.state = {  
 counter: this.state.counter + 1  
}`
- **Correct way:**  
`this.setState({  
 counter: this.state.counter + 1  
})`

# Events

- React has the **same** set of **events** as plain JS
- React events are written in **camelCase**  
**onClick** vs **onclick**
- The **action** must be a **function**, not any statement  
**onClick={() => alert()}** vs **onclick="alert()"**

# Events

- You can define the **event handler** as a **method** inside the class

- Example:

```
increment(){  
  this.setState({counter: this.state.counter + 1})  
}
```

- Usage

```
<button onClick={this.increment}> Click me </button>
```

# This won't work!

- Remember last week's discussion about **this**
- Each JS function has its **own this**, which is the **caller** object
- The object that **calls** the event handler is **not** your **component object**



# Solution

```
constructor() {  
  this.onClick = this.onClick.bind(this);  
}
```

Congrats, 3 this in 1 LOC, and it's not even app logic. Oh, it's official docs.

— André Staltz (@andrestaltz) August 23, 2016

# Another solution

- Recap: **arrow functions** do not introduce their **own this**
- Instead, they **capture this** from the **outer scope**
- Fortunately, the **class body** has the **proper this**
- Therefore, arrow functions **work!**

# Example: a two-way Celsius to Fahrenheit converter

Using a custom component for input boxes

# Notes

- To **store** and **use** input's value:
  - Add** it to state
  - Read** it from state as well
- Read the new value from **event.target.value**

```
<input
  type="text"
  value={this.state.celsius}
  onChange={event => this.setState({
    ...this.state,
    celsius: event.target.value
  })}/>
```

# Lift the state up!

Visit: <https://reactjs.org/docs/lifting-state-up.html>

- To pass a **shared state** between **components**, move it to their **common ancestor**
- **Define** the **state** in the common ancestor
- Pass it as **props** to the **original** components
- Pass a **setter** function as **change handler**

# This session

- Single-page applications
- Intro to React
- JSX
- Props, state, events

# Next session

- React projects
- NodeJS, npm
- Enhanced function components  
Hooks
- API calls

# Final notes

- Read **React** tutorials

<https://reactjs.org/docs/hello-world.html>

<https://reactjs.org/tutorial/tutorial.html>

- Register for **interview** sessions for **phase 2**