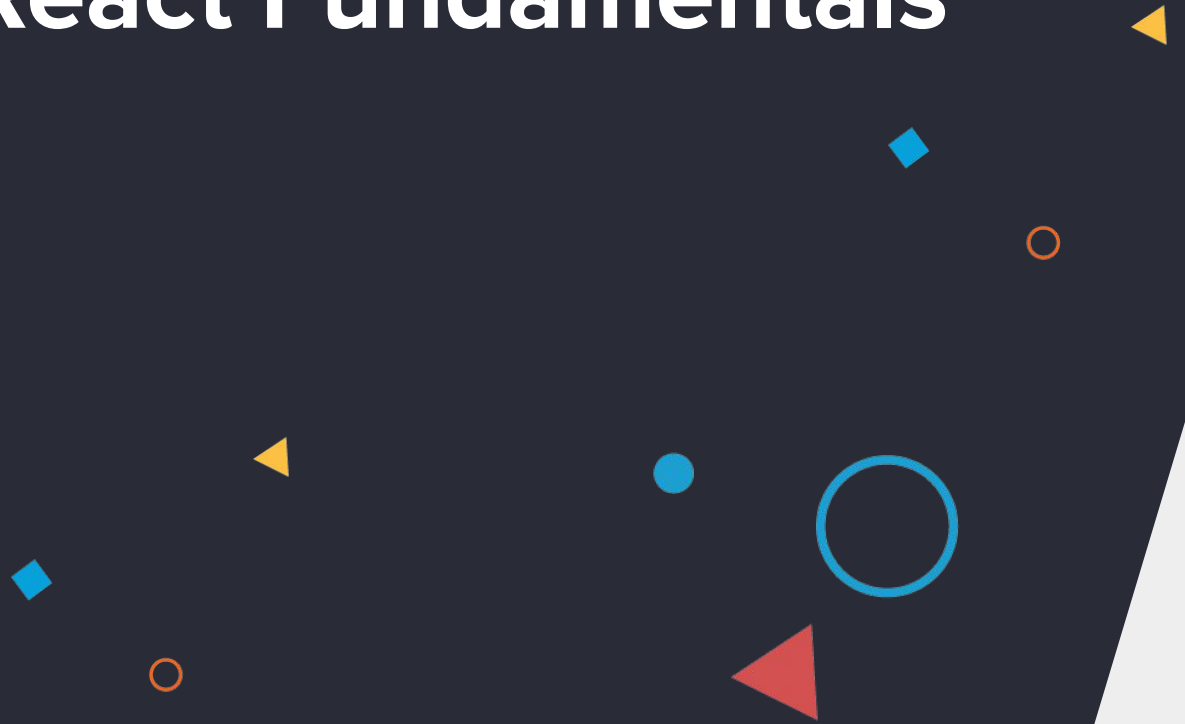
React Fundamentals



LIGHTHOUSE LABS

AGENDA

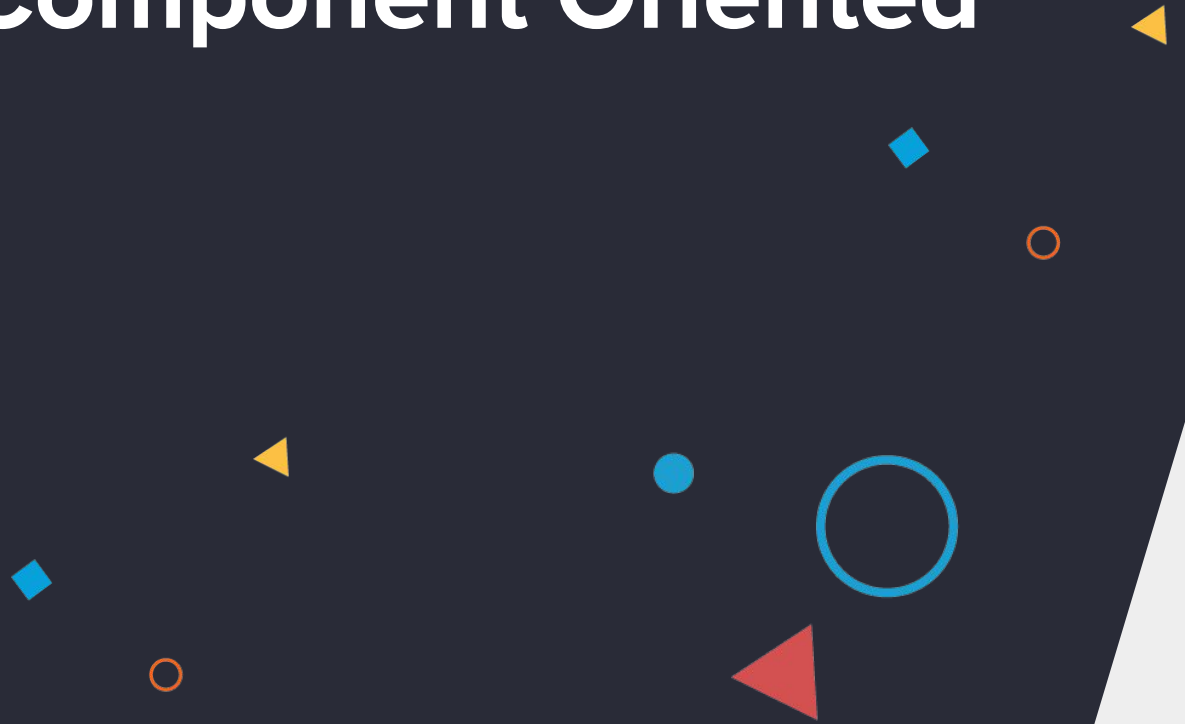
Component Oriented

Templates: JSX

State & Props

Simple React App Demo

Component Oriented



LIGHTHOUSE LABS

HTML Element vs Component

Similarities

- Both have a tree hierarchy
- Both affect the layout of the resulting DOM
-

Differences

- React Components are like “super” elements
- React Components have functional behaviour
- React Components automatically re-render as their state changes

In React, Everything is a Component

- React applications are made up of nested React Components
- Each time a Component is `rendered`, whatever children of that Component are `rendered`

```
function Form(props) {
    return (<form>
        <TextInput name="username" />
        <PasswordInput name="password" />
    </form>);
}
```

```
function TextInput(props) {
    return (<input name={props.name} />);
}
```

LIGHTHOUSE LABS

JSX - React Templating

LIGHTHOUSE LABS

React is a Frontend Framework

- React allows us to **build dynamic frontends** for our websites by composing Components
- Data is still managed by a backend system (for example: express, RoR, flask, etc)
- AJAX (*axios* or *jQuery* or *fetch api*) is used to fetch data for frontend
- Unlike jQuery, **React has a built in templating engine**: JSX
- Unlike anything we've seen before, **React chooses when to render**

JSX is a template language

- JSX is a lot like EJS
  - Instead of being stored in a separate “template” file
  - Every component returns JSX
  - Thus, JSX is by made up of nested partials
  - The JSX return will be re-evaluated whenever a component needs to be rendered

```
function Form(props) {
  return (<form>
    <TextInput name="username" />
    <PasswordInput name="password" />
  </form>);
}
```

jsx

```
function TextInput(props) {
  return (<input name={props.name} />);
}
```

jsx

LIGHTHOUSE LABS

State & Props

LIGHTHOUSE LABS

Component-Local State

- React Components are dynamic – they change in response to data-changes!
- A Component can store data in 2 ways: State and Properties (i.e. Props)
    a. **State is data owned by this component**
    b. **Props is data owned by a parent component**
- Anytime either State or Props change, those related Components will `render`

Props passed from parent to child

- We've seen props already
- In React, we pass props just as we would set an attribute on an HTML Element
- This is often referred to as **prop-threading**
- Our parent component *`Form`* passes the "*name*" prop into the `TextInput` sub-component
- In this case, because the `Form` component passes a bare-string, **name** is static

```
function Form() {
   return (<form>
        <TextInput name="username" />
        <PasswordInput name="password" />
   </form>);
}
```

```
function TextInput(props) {
   return (<input name={props.name} />);
}
```

Props are Immutable from the Child

- Although any state may be changed by the component that owns them
- From the perspective of the child component: Props are immutable
- You **may use** a prop to
  - Render it's value
  - Conditionally change how something else is rendered
  - Invoke a function in the parent component (if a function is passed)
- You **may** never
  - Update the value of a prop

State in React is local to a Comp

- **useState** hooks allow us to **create state** in a component

```
function TextInput(props) {
  const [inputValue, setInputValue] = useState("");
  return (<input name={props.name} />);
}
```

getter

setter

LIGHTHOUSE LABS

State is mutable through setter fx

- **useState** returns an accessor and a mutator (a getter and a setter)
- Using this setter, we can change the state.

```
function TextInput(props) {
    const [inputValue, setInputValue] = useState("");
    function mutateState(e) {
        setInputValue("potato");
    }
    return (<input name={props.name} onChange={mutateState}/>);
}
```

Questions?

LIGHTHOUSE LABS