Google | طويق

ReactJS (Components)

أكاديمية طويق
TUWAIQ ACADEMY

وزارة الاتصالات وتقنية المعلومات
MINISTRY OF COMMUNICATIONS AND INFORMATION TECHNOLOGY

الاتحاد السعودي للأمن السيبراني والبرمجة والدرونز
SAUDI FEDERATION FOR CYBERSECURITY, PROGRAMMING & DRONES

مهارات المستقبل

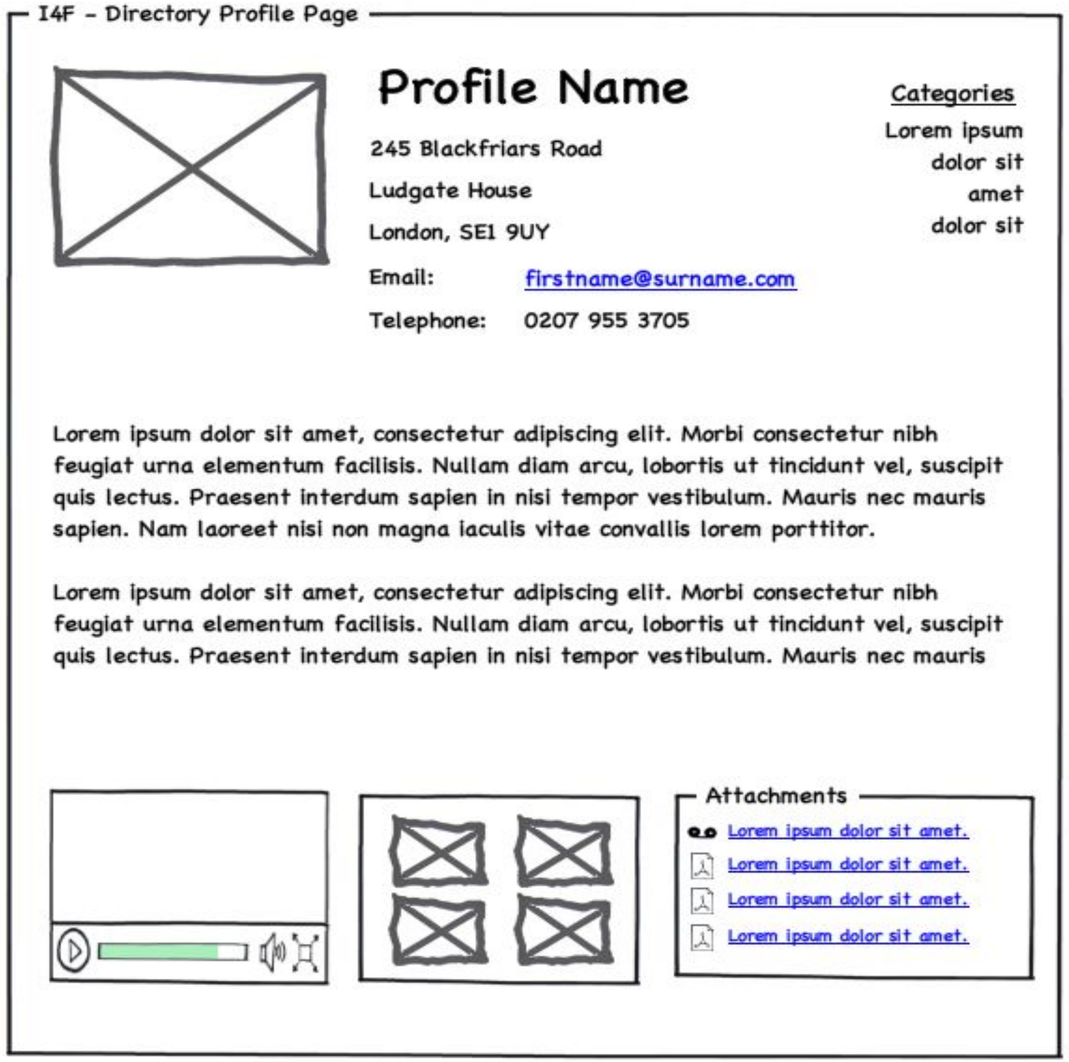
stc

Objective

1. UI design
2. Creating React Components
3. What Data Should be Stored in What State?
4. Props vs state
5. The app.js component
6. Dynamic React Components
7. Render Components using if Statements
8. Render Multiple Components Using Lists

UI Design

- Before you begin writing front-end code, you need to design the user interface (UI).
- There are various ways to do this including, wireframing:



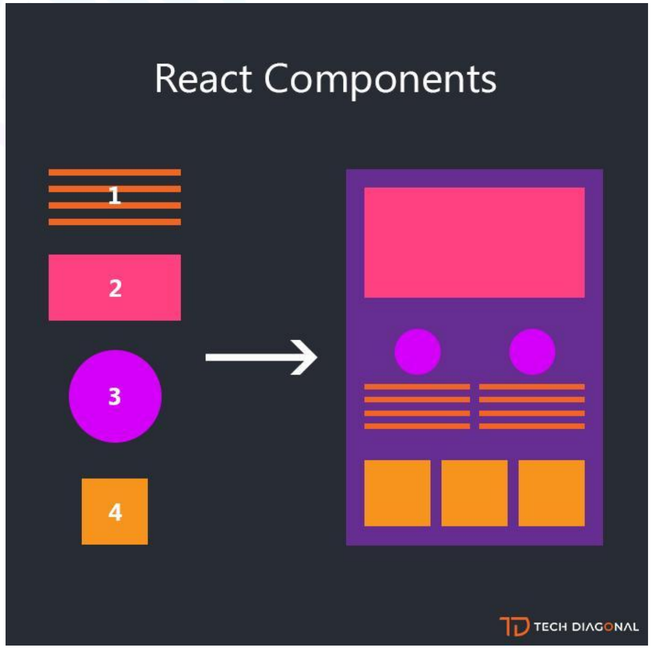
UI Design

- As you design your UI, you will notice that there are certain elements that will be used over and over again.
- According to Reacts' documentation: "A good rule of thumb is that if a part of your UI is used several times (Button, Panel, Avatar), or is complex enough on its own (App, FeedStory, Comment), it is a good candidate to be a reusable component."
- A component can contain other components

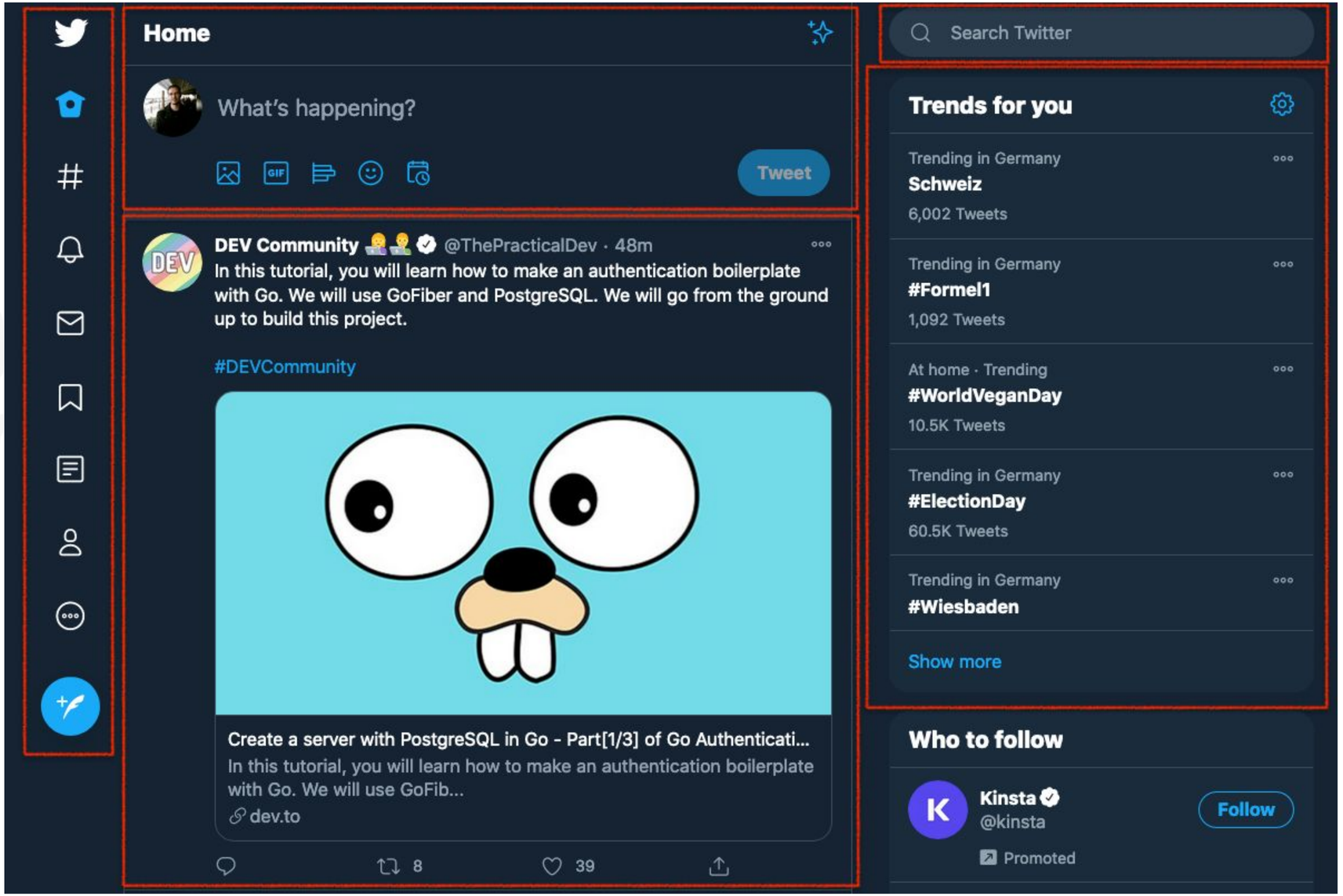
Creating React Components

- “React components are small, reusable pieces of code that return a React element to be rendered to the page.” React

- What is the difference between a React element and a React component?



Creating React Components



Creating React Components

- Notice that a React component is a piece of code (in this case a function) that accepts props as an input and returns a React element as output.

```
Function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

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

Creating React Components

Input	Processing	Output
props	React Component	React Element

`name=”Sara”`	function
	`Welcome(props) {`
	`return <h1>Hello, {props.name}</h1>;`
	`}`
	
	Components can be implemented using either **functions** or **classes**
		`<h1>Hello, Sara</h1>`

Creating React Components

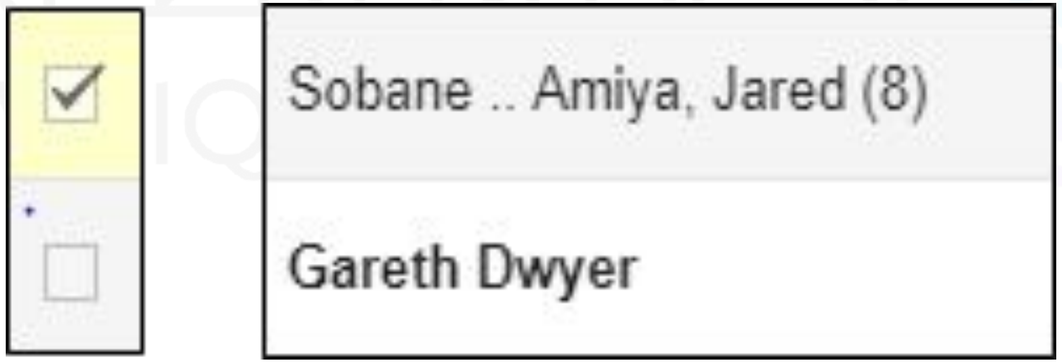
- When creating React components it is extremely important that you always adhere to the following two rules:

1. ALWAYS start component names with a capital letter. React treats components starting with lowercase letters as DOM tags.
2. A component should NEVER modify its own props.

- **Props definition:** “props are inputs to a React component. They are data passed down from a parent component to a child component.”

What Data Should be Stored in What ?State

- State is used to store data related to a component that changes over time.



What Data Should be Stored in What ?State

- Only components that are defined using classes (not functional components) can have state.
- 2 Principles to decide when to use stateful/stateless components (from Artemij Fedojev):

1. “The minority of your React components are stateful. They should be at the top of your components’ hierarchy. They encapsulate all of the interaction logic, manage the user interface state, and pass that state down the hierarchy to stateless components, using props.”
2. “The majority of your React components are stateless. They receive the state data from their parent components via this.props and render the data accordingly” (2015, Page 37).

Props vs. State

Props	State
• can be used in functional and class components. • used to pass data from parent elements to child elements. Props can only be passed from a parent to its children, not the other way around, i.e. top-down or unilateral data flow. • read-only. They should not be modified by components.	• only components defined as classes can have state • private to its component. State is not accessible to any component other than the one that owns and sets it. Data about state can be sent as props. • a component can change its state

How to Create a Stateful Component

1. Create a class component:

```
class NameOfClass extends React.Component {
  render() {
    return (
      <div>
        <h1>Look at this example of a class component!</h1>
      </div>
    );
  }
}
```

How to Create a Stateful Component

2. Add a class constructor to assign the initial state.
3. Read the state info (using this.state) and use it to render the component using the render() method.
4. Allow modification of the state using this.setState(), which will usually be called in an event handler that handles a UI interaction. Never modify the state directly because this won't re-render the component. Always use this.setState().

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() =>
this.tick(), 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```

The App.js Component

- It is a component that is created by default when using Create React App.
- The App component is the container for all other React components.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from
'./registerServiceWorker';

ReactDOM.render(<App />,
document.getElementById('root'));

registerServiceWorker();
```

Dynamic React Components

- To make truly dynamic web applications, we need to be able to use React to do some logic.
- In this task, you will learn to control how and when components are displayed on the browser using if statements and lists.

Render Components using if Statements

- One of the most basic things you want to be able to do with a React application is to decide which components to display, based on some conditions.
- Imagine you have already created the two components below:

```
function Welcome(props) {
  return <h1>Welcome back
{props.name}!</h1>;
}

function SignUp(props) {
  return <button type="button">Sign
in</button>;
}
```

Render Components using if Statements

- You could use the logic below to decide which component to show:

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <Welcome name=”John” />;
  }
  return <SignUp />;
}

ReactDOM.render(
  <Greeting isLoggedIn={true} />,
  document.getElementById('root')
);
```

Render Components using if Statements

- You could also manipulate an element inline using JSX. Remember with JSX you can embed any JavaScript using curly braces.

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' :
'not'}</b> logged in.
    </div>
  );
}
```

Render Multiple Components Using Lists

- When designing UIs, you are also more than likely going to want to be able to add multiple instances of a component to your UI.
- To do this using React, we make use of lists.


```
function NumberList(props) {
  const numbersArr = props.numbers;
  const listItems =
numbersArr.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const nums = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={nums} />,
  document.getElementById('root')
);
```