React.js

🎉🎉

gerrit.neven@plymouth.ac.uk

Today

- Component declaration: class & function
- Props & state
- Passing down props
- Break, + build something
- Modifying state

React.js

- *"A JavaScript library for building user interfaces" - reactjs.org*
- React initially build by Facebook
- Three core principles
  1. Declarative
  2. Component-based
  3. Learn once, write anywhere

Setup instructions (from scratch, see next slide for easy way)

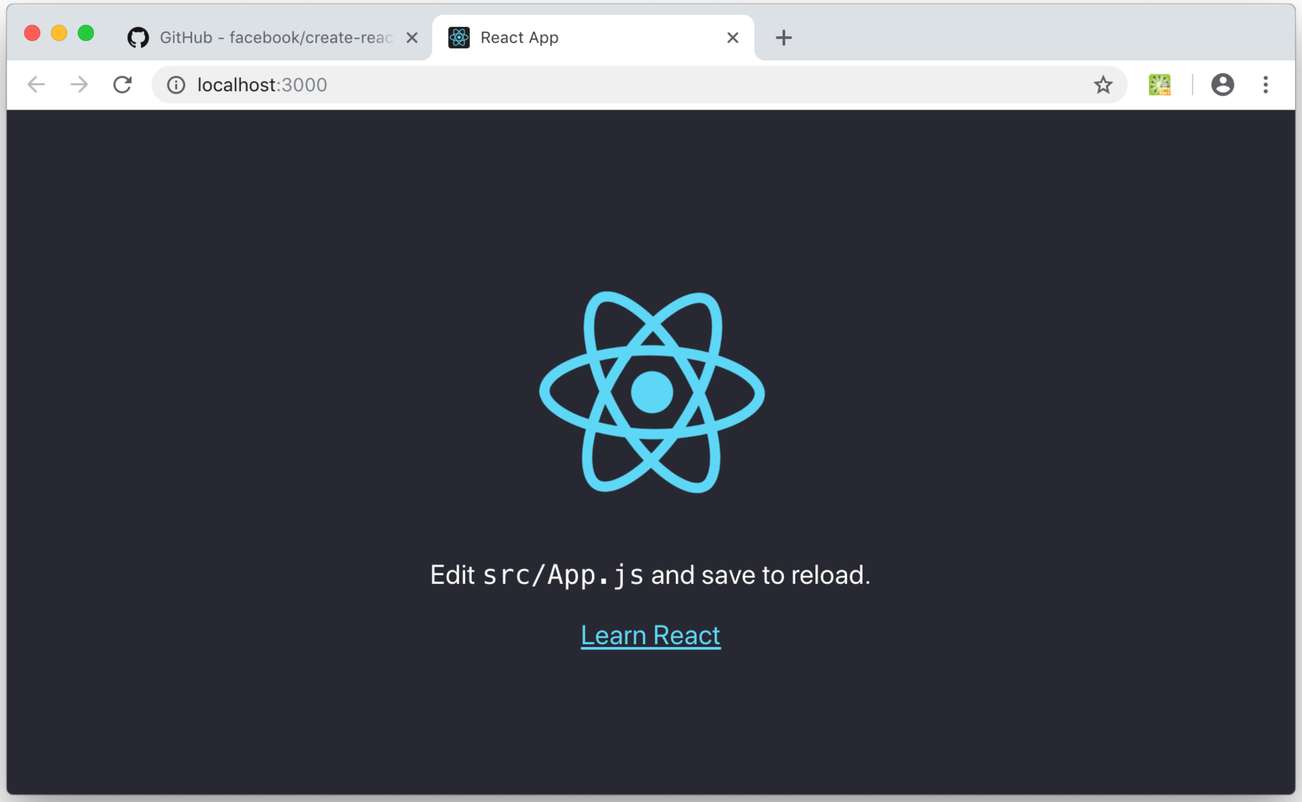
- npm install create-react-app
- Uni computers:
  - node_modules/.bin/create-react-app app-name
- Personal computers:
  - create-react-app app-name
- cd app-name
- npm install
- npm start
- Go to localhost:3000

This has a bunch of things we don't need, so there is a stripped down version at github.com/gerbyzation/dat504-react-starter

Setup: easy way

- git clone github.com/gerbyzation/dat504-react-starter or go to repo page and download zip
- Rename to something more descriptive
- npm install
- npm start (from within the project)
- Go to localhost:3000

This has a bunch of things we don't need, so there is a stripped down version at github.com/gerbyzation/dat504-react-starter


GitHub - facebook/create-reac
React App
localhost:3000
Edit src/App.js and save to reload.
Learn React

Component declaration

- All these components do the same (excuse the typo)
- First two are called `functional components` (they are pure functions), shorter way to declare 'simple' components
- Differences:
  - Functional components can't contain state
  - Class based components have access to lifecycle events (we'll get to these later)

```
function Thing(props) {
  return <h1>This is a component</h1>
}

const Thing = (props) => {
  return <h1>This is a component</h1>
}

class Thiing extends Component {
  render () {
    return <h1>This is a component</h1>
  }
}
```

Components

- Independent
- Reusable
- Learn once, update anywhere

State & props

- Props (short for properties) are the given properties that influences what or how a component displays.
- props are given **by the parent**, a child can't change properties themselves
- Props can also be functions, which will be useful in a bit

TodoList (parent) gives ‘task’ and ‘done’ as props to TodoItem

result:

- [x] read eloquent javascript
- [ ] build react app

```
const TodoList = (props) => {
  return (
    <ul>
      <TodoItem task={'read eloquent javascript'} done={true} />
      <TodoItem task={'build react app'} done={false} />
    </ul>
  )
}

const TodoItem = (props) => {
  return (
    <li>
      <input type='checkbox' defaultChecked={props.done} /> {props.task}
    </li>
  );
}
```

Lets make that a bit more dynamic

- Using .map, we create a list of TodoItem stored under the variable task
- We insert the task list into html by wrapping it in curly braces

- [x] read eloquent javascript
- [ ] extend addressbook API with businesses
- [ ] come up with idea for project

```
const todos = [
  { task: 'read eloquent javascript', done: true },
  { task: 'extend addressbook API with businesses', done: false },
  { task: 'come up with idea for project', done: false }
];

const TodoList = (props) => {
  const tasks = todos.map(item =>
    <TodoItem task={item.task} done={item.done} />);
  return (
    <ul>
      {tasks}
    </ul>
  )
}
```

But how can we modify properties? We make state!

- We define todos in state, and pass this as a prop (!) to TodoList

```
class App extends Component {
  state = {
    todos: [
      { task: 'read eloquent javascript', done: true },
      { task: 'extend addressbook API with businesses', done: false },
      { task: 'come up with idea for project', done: false }
    ]
  }

  render () {
    return <TodoList todos={this.state.todos} />
  }
}

const TodoList = (props) => {
  const tasks = props.todos.map(item => <TodoItem task={item.task} done={item.done} />);
  return (
    <ul>
      {tasks}
    </ul>
  )
}
```

What can we do with state

- State is modifiable via this.setState()
  - **Always** use setState(), don't modify state directly
- State can be passed down as props
- To modify state, create methods that use setState() (always use arrow functions for this)

```
class Person extends Component {
  state = {
    name: 'Sandra',
    age: 32,
    gender: 'female',
  }

  changeName = (newName) => {
    this.setState({
      name: newName,
    })
  }
}
```

State & props

- Props come from parent element
- State is part of a specific component
- Children can’t modify parent state directly, but can be given functions through props that can update parent state
- Unidirectional: data flows down

```
class App extends Component {
  state = {
    active: false
  }

  toggleStatus = () => {
    this.setState({
      active: !this.state.active // set active to opposite of current value
    })
  }

  render() {
    return <Status toggleStatus={this.toggleStatus} active={this.state.active} />
  }
}

const Status = (props) => {
  let status;
  if (props.active == true) status = 'on'
  else status = 'off'
  return (
    <div>
      <h1>the switch is <span style={{textTransform: 'uppercase'}}>{status}</span></h1>
      <button onClick={props.toggleStatus}>Toggle status</button>
    </div>
  );
}
```

State, props & setState are the core concepts of React

There's some more to show you, but that's for specific use cases. If you know this well you understand most of react! 🎉

Exercise & break time!

Exercise

Build a Counter component that displays a minus button, current count and add button (eg like “ - 12 +”).

Need hints? Download the slides and look at the next slide

Exercise hints

- The markup would look something like this:

```
<div>
  <h1>Counter</h1>
  <button>-</button>
  <span class="count">12</span>
  <button>+</button>
</div>
```

- Keep track of the count value in state
- Create an increment and decrement method in your component class
- A button element has an onClick event handler that executes a function when the button is pressed

```
function buttonClicked() {
  console.log('this button was pressed!')
}

const Carrot = () => {
  return <button onClick={buttonClicked}>Click me!</button>
}
```

Forms

- Uncontrolled
  - Ref(ference)
  - When you need the value, find the element via it's ref and grab it's value
- Controlled
  - Keep track of it's value at all time

```
class UncontrolledForm extends Component {
  state = {
    name: undefined,
  }
  input = React.createRef();

  greet = (event) => {
    this.setState({
      name: this.input.current.value
    })
  }

  render () {
    if (this.state.name) {
      return <h1>Hi {this.state.name}, how are you?</h1>
    } else {
      return (
        <div>
          <input type='text' ref={this.input} placeholder="What's your name" />
          <button type='submit' onClick={this.greet}>Send</button>
        </div>
      )
    }
  }
}
```

Controlled form

```
class PhoneNumberField extends Component {
  state = {
    number: '',
  }

  onChange = (event) => {
    this.setState({
      number: event.target.value,
    })
  }

  render() {
    return (
      <div>
        <p>Number: {this.state.number}</p>
        <input type='text' onChange={this.onChange} value={this.state.number}
placeholder="phone number" />
      </div>
    )
  }
}
```

Form exercises

1. Build a simple app that allows you to make a list. All it needs is an input field, a button that adds it to the list and a list of items. Break it up in separate components so you should end up with 4 (input, button, list and list-item)
2. Build an input for email address that validates the structure while you type

Tomorrow

- Components that communicate with server
- Navigation
- node.js authentication?