CS2003 W10
Client side frameworks

Özgür Akgün

Slides adapted from David Morrison

Reminder

Coursework:

Web2 practical released and on Studres.

https://studres.cs.st-andrews.ac.uk/CS2003/Practicals/Web2/CS2003%20Web2.pdf

Due 9pm on Wednesday the 25th November

Submit via MMS

This week

- Client side JavaScript Frameworks
  - jQuery
  - Angular
  - React
  - Vue
- Server Side JavaScript with Node.js
- Quick discussion of ECMAScript 6
- React JS
- Then there will be a chance to work on Web 2 and get help with it.

JavaScript Frameworks

- JavaScript in its original form can be a little cumbersome
  - You may have found this out yourselves already!
- Frameworks allow accelerated application development via
  - Libraries for particular tasks
  - Abstractions over JavaScript to make programming easier
  - Providing an environment suitable for a particular kind of task

Why not learn them on this course?

- JavaScript frameworks are themselves written in JavaScript.
  - It's important to understand what is going on underneath!
- Frameworks constantly change in popularity
  - Next year the most popular framework will quite possibly be something completely new!
  - This course doesn't teach any framework, but you should try them out!
  - You should now have enough knowledge to understand the framework basics yourself!

jQuery

- Very widely used.
- Not so trendy any more.
- Makes it easier to
  - Navigate the DOM
  - Select DOM elements
  - Create Animations
  - Handle Events
  - Make HTTP Requests.
- Uses the $ symbol everywhere!

jQuery Selectors

- The key feature of jQuery is the idea of a DOM element selector.
- These are similar to CSS selectors.

```
// normal JS DOM code
let button = document.getElementById('continue');
button.onclick = function() {
  console.log("button clicked");
};
```

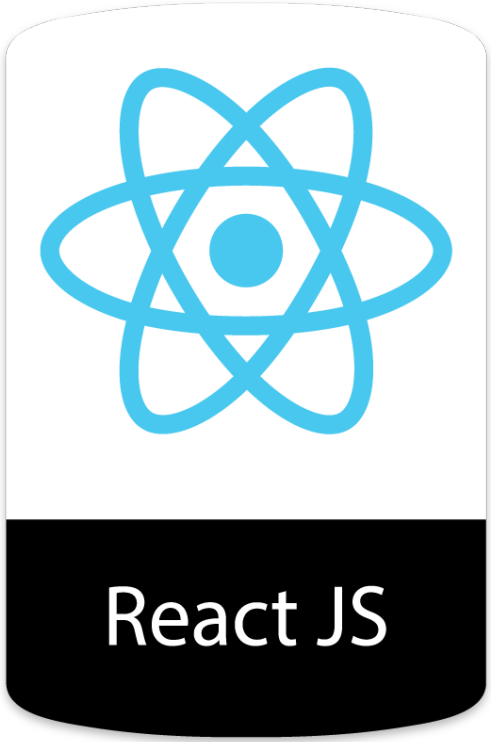
```
// jquery
$("button#continue").click(function() {
  console.log("button clicked");
});
```

Angular JS

- Very popular framework
- Principally maintained by Google
- Provides new pre-built components to JavaScript
- Primarily aimed at single page web applications
- Works by binding to HTML tags which it then modifies
- Got a reputation as being a bit complicated.

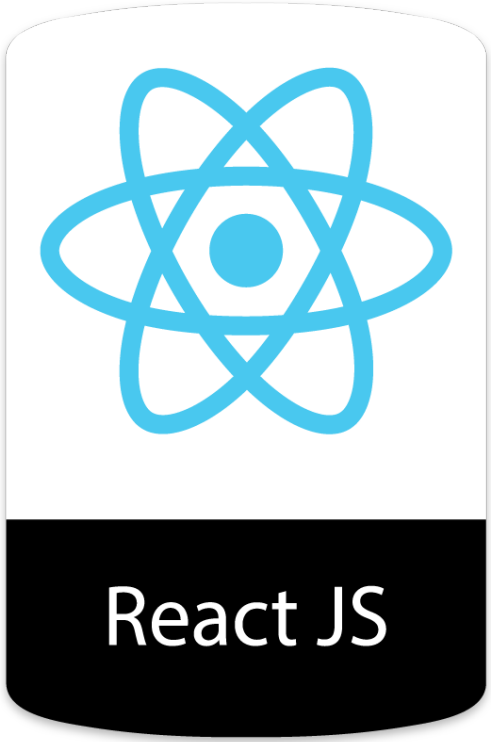
React

- React is a still quite trendy framework for JS.
- Built by Facebook.
- Designed around a data-driven views.
- Data is transformed into Views.
- Views are defined in JSX



React

- Based on the principle of one-way data flow
  - Properties are passed to components
  - Components use properties but cannot modify them.
  - Callbacks can modify the state, which modifies the properties
  - When the state is changes, all the components are redrawn.



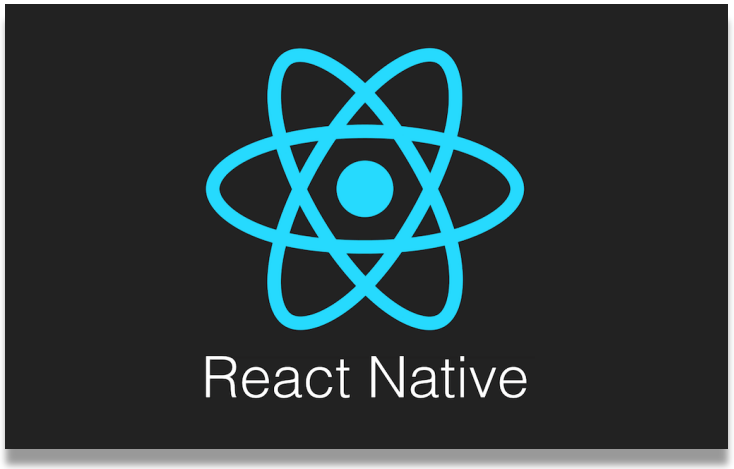
JSX

- JSX is a super set of JavaScript that includes support for XML literals.
- This means you can declare HTML tags right inside your JavaScript.
- These are translated into pure JavaScript using a build step.

```
const CardSection = (props) => {
  return (
    <View style={styles.containerStyle}>
      {props.children}
    </View>
  );
};
```

React Native

- All JavaScript apps can run as mobile apps by wrapping them in a WebView – works on Android and iOS.
- However, this prevents you getting access to native functionality.
- React Native is a system that let's you write apps for Android and iOS in JavaScript and take advantage of all the device has to offer.

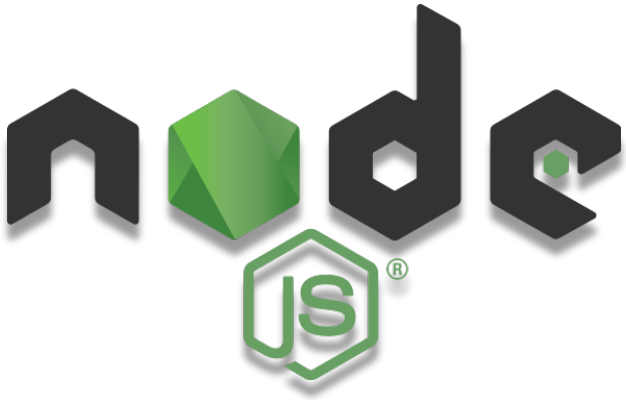


Vue

- Mixes ideas from React and Angular
- Build by the JS Community
- Based around
  - Templates
  - Reactivity
  - Components
- Very hot right now!

NodeJS

- Node is completely different JavaScript runtime environment.
- Node is basically for writing servers, particularly micro-services.
- Let's you share the language between client and server.
- It's quite fast compared to some other server side solutions.



NodeJS and Express

- Much as the browser has frameworks, so does Node.js.
- Express.js is a popular one that make it easier to make RESTful APIs.
- In this example, we have defined a route / that returns ‘Hello World!’ when we send an HTTP request to it.

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(3000, () => {
  console.log('Listening on port 3000!');
});
```

Frameworks and ES6

- There are lots and lots of JavaScript frameworks!
- Many of them make use of ECMAScript 6.
- As this is an introductory course we have not covered all of ECMAScript 6's features.
- You might want to check out:
  - Template Literals
  - Destructuring Assignment
  - Modules
  - Generators
- Go and explore, there are lots of cool things.

CS2003 W10
React

Özgür Akgün

Slides adapted from David Morrison

Motivation

- We want to write a client-side single page applications in the browser.
- These might be a things such as;
  - Social media timelines
  - Chat systems
  - Text and Code Editors
  - Data Visualisations

Possible Approaches

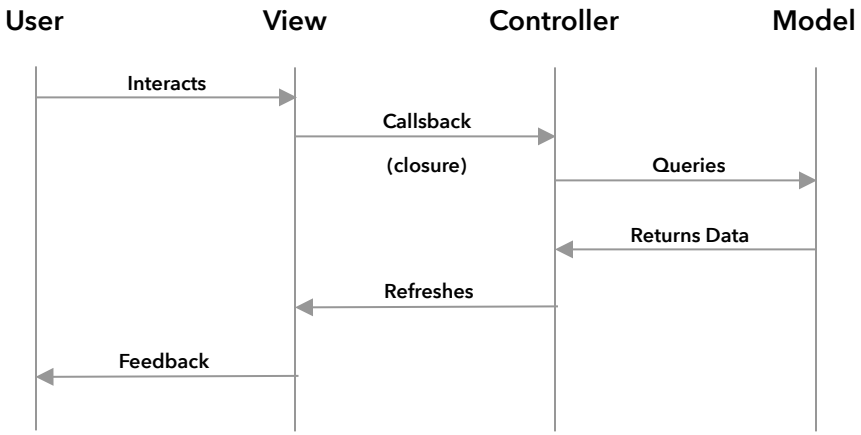
- Standard JavaScript Browser APIs
- Library like jQuery
- Framework
  - Angular
  - React
  - Vue.js
- Language with Framework
  - Elm

JQuery

- JQuery was invented in 2006 and was very popular in the late 2000s
- Addresses the following problems;
  - Not all browsers provided the same Browser APIs and implementations were inconsistent.
    - JQuery provided a layer on top that was consistent across platforms
  - Manipulating the DOM using the Browser APIs was complex
    - JQuery introduced ways of selecting and modifying DOM elements based on CSS selectors
  - The AJAX APIs (such as XMLHttpRequest) are a bit tricky to use
    - JQuery has a set of convenience functions for making HTTP requests.
- These are largely mitigated with improvements to the Browser APIs that are consistent across browsers.

Application Architecture

- So we don’t need JQuery because the Browser APIs support most of it’s functionality but do we still want to build apps this way?



System Responds to Interaction Event

Application Architecture

- Using the Browser APIs or JQuery to realise this kind of architecture suffers from two kinds of issues:
  - DOM manipulation problems
    - It's slow
    - It's complex
    - You need to know a lot about the current state in order to update it to the new one
    - Its the developers responsibility to keep the model and view states synchronized
  - Monolithic code
    - There is no enforced separation of concerns between different parts of the page
    - State ends up mixed in all over the place and not encapsulated effectively
    - This can make thing hard to follow

Frameworks

- Client-side web frameworks address these issues in different ways but they all do two things;
  - They automate the DOM manipulation but providing **declarative** ways of specifying the view state based on the model state.
  - They allow the page to be broken down in to encapsulated **components**.

React

- React is a framework for building the view component
- React lets to build all or parts of your web application from encapsulated, reusable components with local state, and compose them into complex user interfaces.
- React views are Declarative in the same way that HTML is.
  - The developer only needs to consider the current state of the view
  - They are efficiently updated to minimize DOM manipulation
  - The application view is a bit like a pure function that takes in the model and returns a set of DOM elements.
  - Data flow is uni-directional with simplifies the application logic – the data just goes one way, into the components!

Components

- It would be nice if we could declare instances of our custom components in the same way as we do HTML tags.
- For example, let's say we have a component called **HelloMessage**, that has a property called **name**, we might want to write:

```
<HelloMessage name="Taylor" />
```

- We want this component to evaluate to the string "Hello Taylor"

Components

- In React, we do the following (assuming that HelloMessage exists):

```
ReactDOM.render(
    React.createElement(HelloMessage, { name: "Taylor" }),
    document.getElementById('hello-example')
);
```

- This appends the HelloMessage component as a child of the hello-example element in the DOM.

JSX

- We can make this more like HTML using a language called JSX
- JSX is an extension of JavaScript that adds XML-like literals to the language.
- Using JSX we can rewrite our example like so:

```
ReactDOM.render(
    <HelloMessage name="Taylor" />,
    document.getElementById('hello-example')
);
```

Declaring a new class of Component

- But how do we specify the HelloWelcome component in the first place? We declare a JavaScript class that inherits from **React.Component** and has a **render** method.

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

Stateful Components

- As well as having properties, stored in the this.props field, components can also have internal state.
- This component displays a counter that counts up the number of seconds since it was mounted.
- **componentDidMount** and **componentWillUnmount** are called as when the component is first rendered and when it is removed from the Virtual DOM

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

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

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

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

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

ReactDOM.render(
  <Timer />,
  document.getElementById('timer-example')
);
```

create-react-app

- create-react-app is a command line utility that generates a simple React application that can be modified.
- It makes starting a new project easier.
- It's on GitHub here: https://github.com/facebook/create-react-app

- Let's have a go at that!

monster-zoo

- Example application from: https://github.com/davemor/monster-zoo
- Uses express to serve an API (port 5000)
- React for a UI (port 3000)

- For the server
  - npm install express
  - node server.js (port 5000)
- For the client
  - npm install create-react-app
  - Install yarn (mac: brew install yarn)
  - yarn start