bucks new university

CO550 – WEB APPLICATIONS

UNIT 11 – WIDER CONTEXT OF WEB APPLICATIONS, PROGRESSIVE WEB APPS, SEO, AND UX

WIDER CONTEXT OF WEB APPLICATIONS

- **Progressive Web Apps** (PWAs) and **Single Page Applications** (SPAs)
- SEO Considerations
- UX (User Experience)

WHAT ARE PROGRESSIVE WEB APPS?

Google's definition…

Progressive Web Apps are user experiences that have the reach of the web, and are:

- **Reliable** - Load instantly and never show the downasaur, even in uncertain network conditions.
- **Fast** - Respond quickly to user interactions with silky smooth animations and no janky scrolling.
- **Engaging** - Feel like a natural app on the device, with an immersive user experience.

Source: https://developers.google.com/web/progressive-web-apps/

GOOGLE'S PWA CHECKLIST

https://developers.google.com/web/progressive-web-apps/checklist

Some of the baseline requirements:

- Site is served over HTTPS
- Pages are responsive on tablets & mobile devices
- All app URLs load while offline
- Metadata provided for Add to Home screen
- Site works cross-browser
- Each page has a URL (deep linking)
- Etc…

bucks
new university

HOW DO WE BUILD A PROGRESSIVE WEB APP?

bucks
new university

IN ASP.NET…

Web API + Single Page Application = SPA

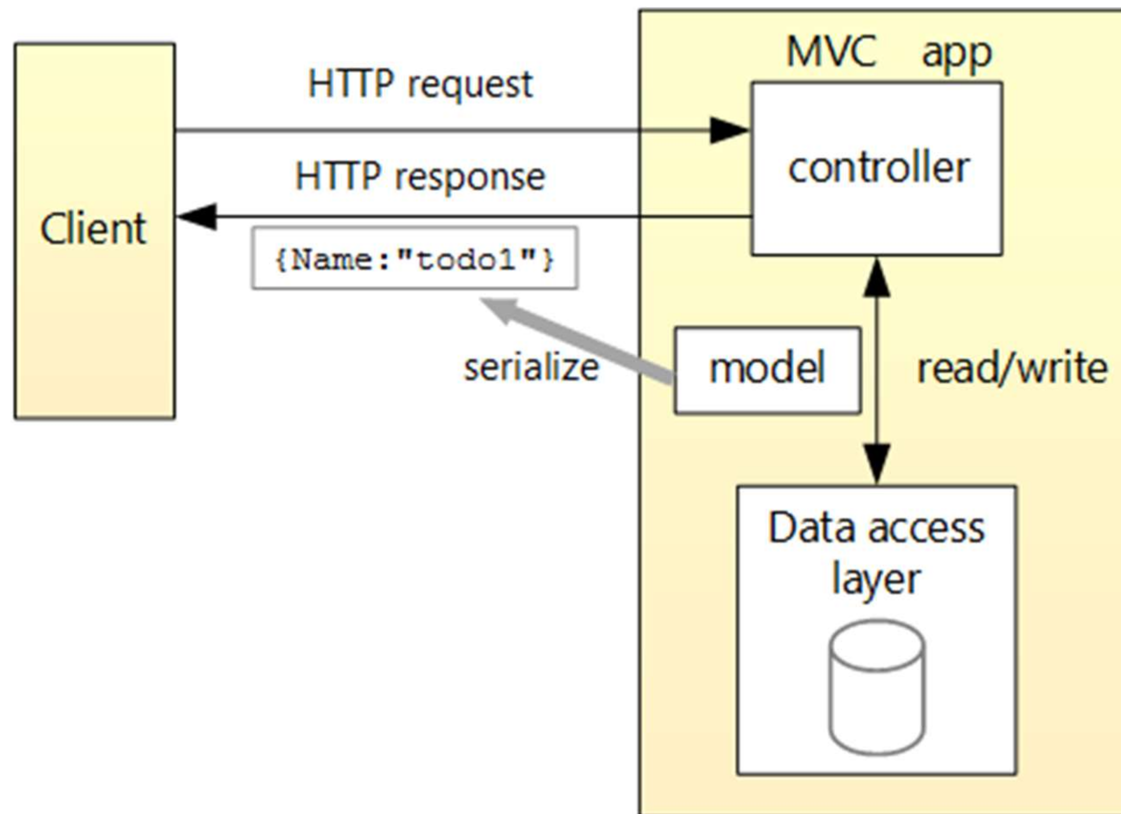
(almost a PWA)

- The term **single-page application (SPA)** is a broadly applied term… but generally speaking, an SPA is a web application whose initial content is delivered as a combination of HTML and JavaScript and whose subsequent operations are performed using a RESTful web service that delivers data via JSON in response to Ajax requests.
- This differs from a Razor Pages app, for example, where operations performed by the user result in new HTML documents being generated in response to synchronous HTTP requests – we can call this type of application a **round-trip application** (RTA).

bucks new university

ASP.NET CORE WEB API

Source: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.2



ADVANTAGES OF SINGLE PAGE APPLICATIONS

The advantages of a SPA are that…

- less bandwidth is required
- the user receives a smoother experience

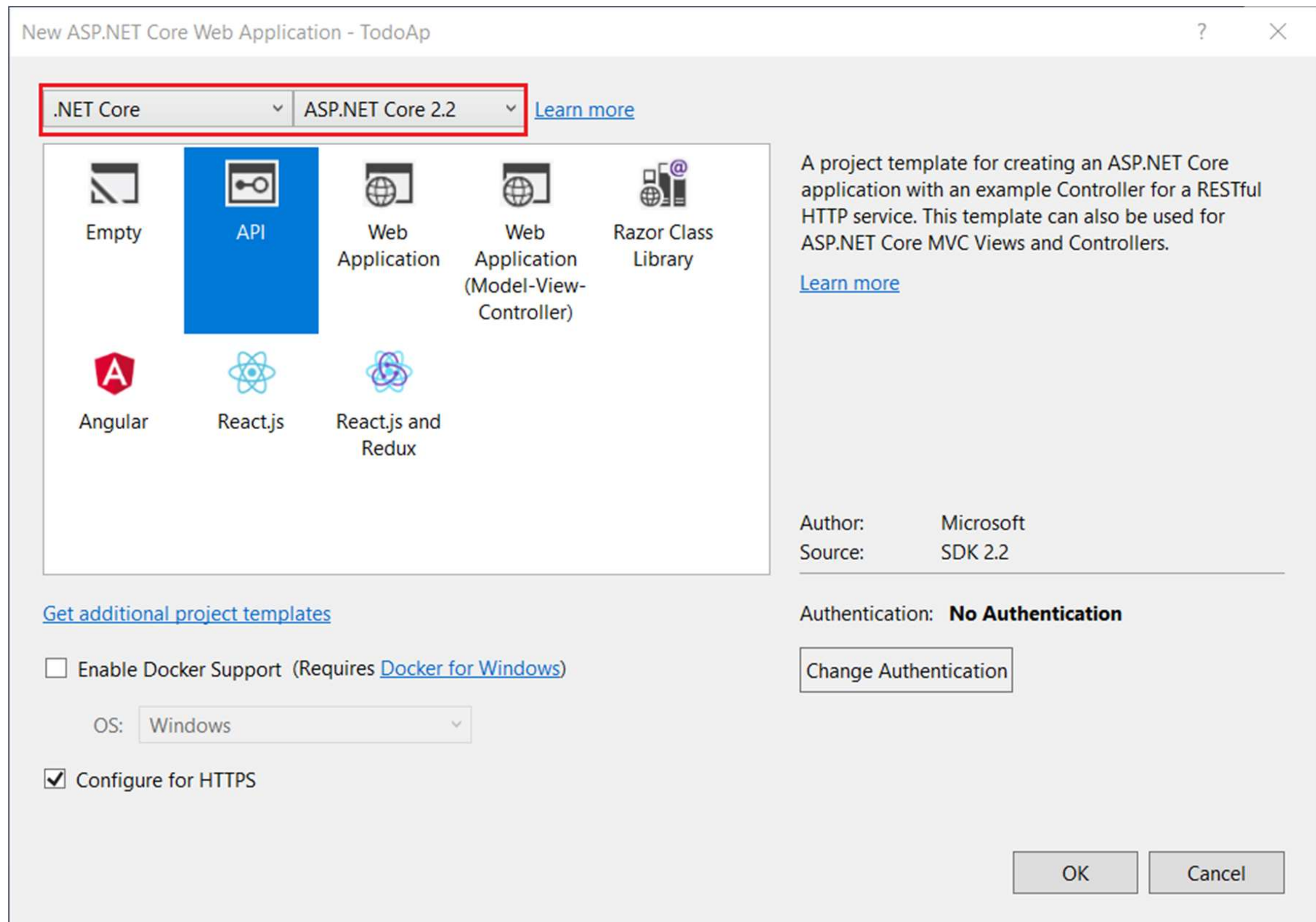
Disadvantages

- the smoother experience can be hard to achieve and that the complexity of the JavaScript code required for a SPA demands careful design and testing.

Many applications mix and match SPA and RTA techniques, where each major functional area of the application is delivered as a SPA, and navigation between functional areas is managed using standard HTTP requests that create a new HTML document.

bucks
new university

CREATING A WEB API PROJECT

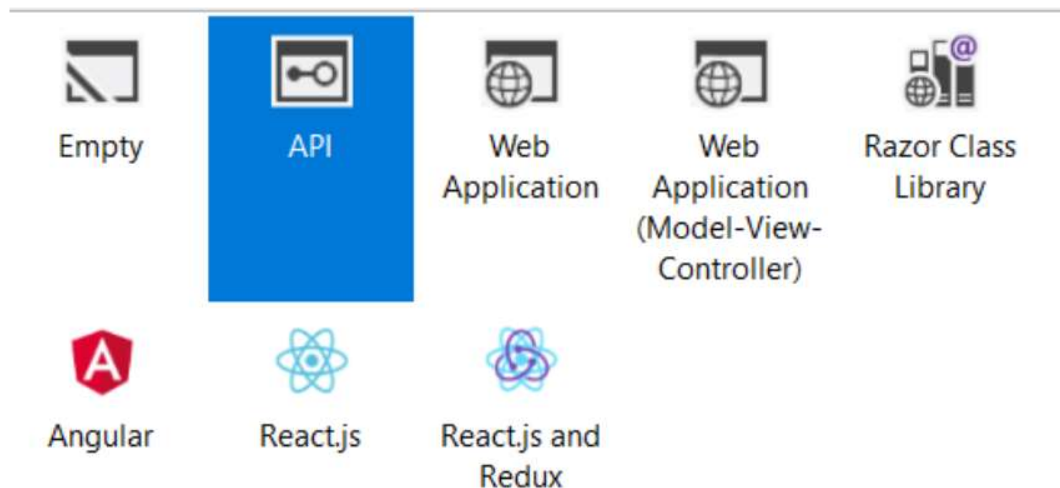


SINGLE PAGE APPLICATIONS

Web API + client-side framework (or you could use raw JS)

JS client-side frameworks…

- ReactJS - https://reactjs.org/
- AngularJS - https://angularjs.org/
- Knockout - https://knockoutjs.com/



bucks new university

SINGLE PAGE APPLICATIONS…

- The transition to a single-page application puts more of a burden on the browser because I need to preserve application state at the client.
- I need a data model that I can update, a series of logic operations that I can perform to transform the data and a set of UI elements that allows the user to trigger those operations.
- In short, I need to recreate a miniature version of the MVC pattern in the browser.
- The library that Microsoft has adopted for single-page applications is **Knockout**, which creates a JavaScript implementation of the MVC pattern (or, more accurately, the **MVVM** pattern)

bucks
new university

MVVM

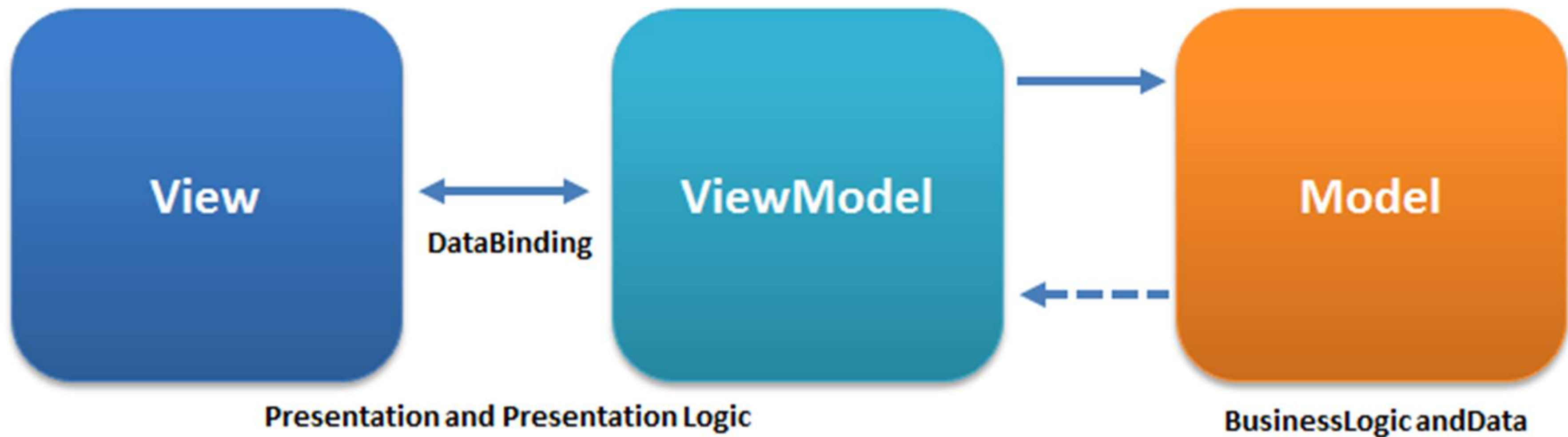
Model-View-ViewModel

- **Model** = the data or business logic, completely UI independent, that stores the state and does the processing (exactly as it is in MVC)
- **View** = the visual elements, the buttons, graphics and more complex controls of a GUI (again, as in MVC)
- **ViewModel** = "Model of a View“ and can be thought of as abstraction of the view, but it also provides a specialization of the Model that the View can use for data-binding. The ViewModel contains data-transformers that convert Model types into View types, and it contains Commands the View can use to interact with the Model.

Goes all the way back to 2005:
https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/

bucks
new university

MVVM



https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

bucks
new university

MVVM

A key aspect of the MVVM approach is **data binding**:

- In simple examples, the View is data **bound** directly to the Model. Parts of the Model are simply displayed in the view by one-way data binding.
- Other parts of the model can be edited by directly binding controls two-way to the data. For example, a boolean in the Model can be data bound to a CheckBox, or a string field to a TextBox.
- In practice however, only a small subset of application UI can be data bound directly to the Model... The Model is very likely to have a data types that cannot be mapped directly to controls.
- The UI may want to perform complex operations that must be implemented in code which doesn't make sense in our strict definition of the View but are too specific to be included in the Model.
- Finally we need a place to put view state such as selection or modes.

The ViewModel is responsible for all of these tasks.

bucks
new university

KNOCKOUT AS AN EXAMPLE

Some features of https://knockoutjs.com

Declarative Bindings

Easily associate DOM elements with model data using a concise, readable syntax

Automatic UI Refresh

When your data model's state changes, your UI updates automatically

bucks new university

KNOCKOUT AS AN EXAMPLE

```
Choose a ticket class:
<select data-bind="options: tickets,
                   optionsCaption: 'Choose...',
                   optionsText: 'name',
                   value: chosenTicket"></select>

<button data-bind="enable: chosenTicket,
                   click: resetTicket">Clear</button>

<p data-bind="with: chosenTicket">
    You have chosen <b data-bind="text: name"></b>
    ($<span data-bind="text: price"></span>)
</p>

<script>
    function TicketsViewModel() {
        this.tickets = [
            { name: "Economy", price: 199.95 },
            { name: "Business", price: 449.22 },
            { name: "First Class", price: 1199.99 }
        ];
        this.chosenTicket = ko.observable();
        this.resetTicket = function() { this.chosenTicket(null) }
    }
    ko.applyBindings(new TicketsViewModel());
</script>
```

Binding attributes declaratively link DOM elements with model properties

Your ***view model*** holds the UI's underlying data and behaviors

Activates Knockout

bucks new university

KNOCKOUT AS AN EXAMPLE

Represent your items as a JavaScript array, and then use a foreach binding to transform this array into a TABLE or set of DIVs. Whenever the array changes, the UI changes to match (you don't have to figure out how to inject new TRs or where to inject them).

The rest of the UI stays in sync. For example, you can declaratively bind a SPAN to display the number of items as follows:

```
There are <span data-bind="text: myItems().length"></span> items
```

Similarly, to make the 'Add' button enable or disable depending on the number of items:

```
<button data-bind="enable: myItems().length < 5">Add</button>
```

bucks
new university

KNOCKOUT – INTERACTING WITH AN API

https://knockoutjs.com/documentation/json-data.html

Knockout doesn't force you to use any one particular technique to load or save data. A commonly-used mechanism is jQuery's Ajax helper methods, such as getJSON, post, and ajax. You can fetch data from the server:

```
$.getJSON("/some/url", function(data) {
    // Now use this data to update your view models,
    // and Knockout will update your UI automatically
})
```

... or you can send data to the server:

```
var data = /* Your data in JSON format - see below */;
$.post("/some/url", data, function(returnedData) {
    // This callback is executed if the post was successful
})
```

bucks new university

KNOCKOUT AND MVVM

- **A *model***: your application's stored data. This data represents objects and operations in your business domain (e.g., bank accounts that can perform money transfers) and is independent of any UI. When using KO, you will usually make Ajax calls to some server-side code to read and write this stored model data.
- **A *view model***: a pure-code representation of the data and operations on a UI. For example, if you're implementing a list editor, your view model would be an object holding a list of items, and exposing methods to add and remove items.
  - Note that this is not the UI itself: it doesn't have any concept of buttons or display styles. It's not the persisted data model either - it holds the unsaved data the user is working with. When using KO, your view models are pure JavaScript objects that hold no knowledge of HTML.
- **A *view*:** a visible, interactive UI representing the state of the view model. It displays information from the view model, sends commands to the view model (e.g., when the user clicks buttons), and updates whenever the state of the view model changes.
  - When using KO, your view is simply your HTML document with declarative bindings to link it to the view model.

Source: https://knockoutjs.com/documentation/observables.html

bucks new university

KNOCKOUT AND MVVM

To create a view model with KO, just declare any JavaScript object. For example…

```
var myViewModel = {
    personName: 'Bob',
    personAge: 123
};
```

You can then create a very simple **view** of this view model using a declarative binding. For example, the following markup displays the personName value:

```
The name is <span data-bind="text: personName"></span>
```

bucks
new university

RESOURCES AND TUTORIALS

- Create a standalone Web API project using ASP.NET Core: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-2.2&tabs=visual-studio
- Knockout Tutorial: https://jakeydocs.readthedocs.io/en/latest/client-side/knockout.html
- https://knockoutjs.com/documentation/observables.html#mvvm_and_view_models
- ASP.NET Core React.js tutorial: https://reactjs.net/tutorials/aspnetcore.html
- How to build a good RESTful API (Video): https://www.youtube.com/watch?v=sMKsmZbpyjE

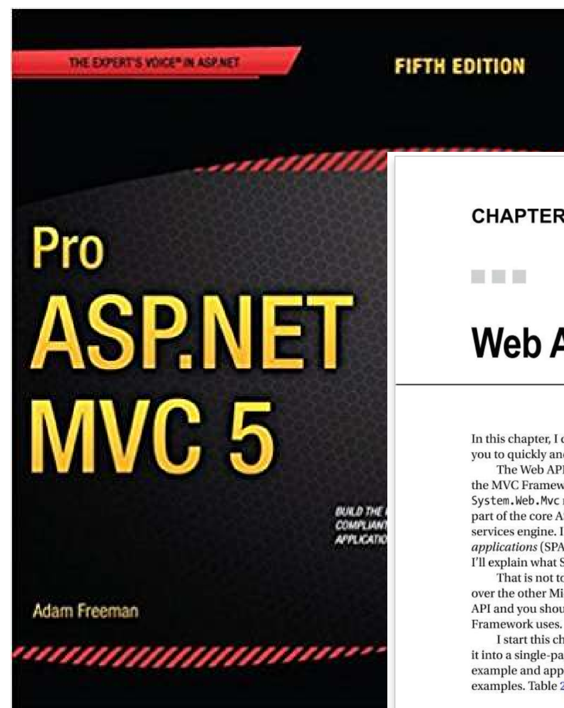
bucks new university

ANOTHER FULL TUTORIAL…

Pro ASP.NET MVC 5 (Fifth Edition)

Available online (and in print) via BNU Library

Chapter 27
Covers the Web API and SPA approach



CHAPTER 27

Web API and Single-page Applications

In this chapter, I describe the Web API feature, which is a relatively new addition to the ASP.NET platform that allows you to quickly and easily create Web services that provide an API to HTTP clients, known as *Web APIs*.

The Web API feature is based on the same foundation as the MVC Framework applications, but is not part of the MVC Framework. Instead, Microsoft has taken some key classes and characteristics that are associated with the `System.Web.Mvc` namespace and duplicated them in the `System.Web.Http` namespace. The idea is that Web API is part of the core ASP.NET platform and can be used in other types of Web applications or used as a stand-alone Web services engine. I have included Web API in this book because one of the main uses for it is to create *single-page applications* (SPAs) by combining the Web API with MVC Framework features you have seen in previous chapters. I'll explain what SPAs are and how they work later in the chapter.

That is not to take away from the way that Web API simplifies creating Web services. It is a huge improvement over the other Microsoft Web service technologies that have been appearing over the last decade or so. I like the Web API and you should use it for your projects, not least because it is simple and built on the same design that the MVC Framework uses.

I start this chapter by creating a regular MVC Framework application and then using the Web API to transform it into a single-page application. This is a surprisingly simple example, so I have treated the process like an extended example and applied some of the relevant techniques from earlier chapters because you can never have enough examples. Table 27-1 provides the summary for this chapter.

SEO CONSIDERATIONS

WHAT IS SEO ANYWAY?

- SEO = Search Engine Optimisation
- SEO *"is the practice of increasing the quantity and quality of traffic to your website through organic search engine results."*
- Google (or any search engine you're using) has a crawler that goes out and gathers information about all the content they can find on the Internet.
- The crawlers bring all those 1s and 0s back to the search engine to build an index.
- That index is then fed through an algorithm that tries to match all that data with your query.
- How does that algorithm work……..?

Source: https://moz.com/learn/seo/what-is-seo

bucks
new university

SEARCH ENGINE ALGORITHMS

- We don’t actually know exactly how the Google Search Engine Algorithm works
- But, Google (and others) give us some strong clues
- It used to be about stuffing keywords in your pages
- Now it is all about providing useful, relevant, reputable content that is presented in an easily consumable, semantic and valid manner.

Further reading:

https://www.google.com/intl/en_uk/search/howsearchworks/

bucks
new university

APPLYING SEO TO WEB APPS?

- For a lot of web apps, we may not typically need to worry about SEO because we don't need users to find the app online.
- For other web apps, like online shops, we definitely DO want search engines to find all of our content.

For "closed" web apps, what are the issues we need to think about?

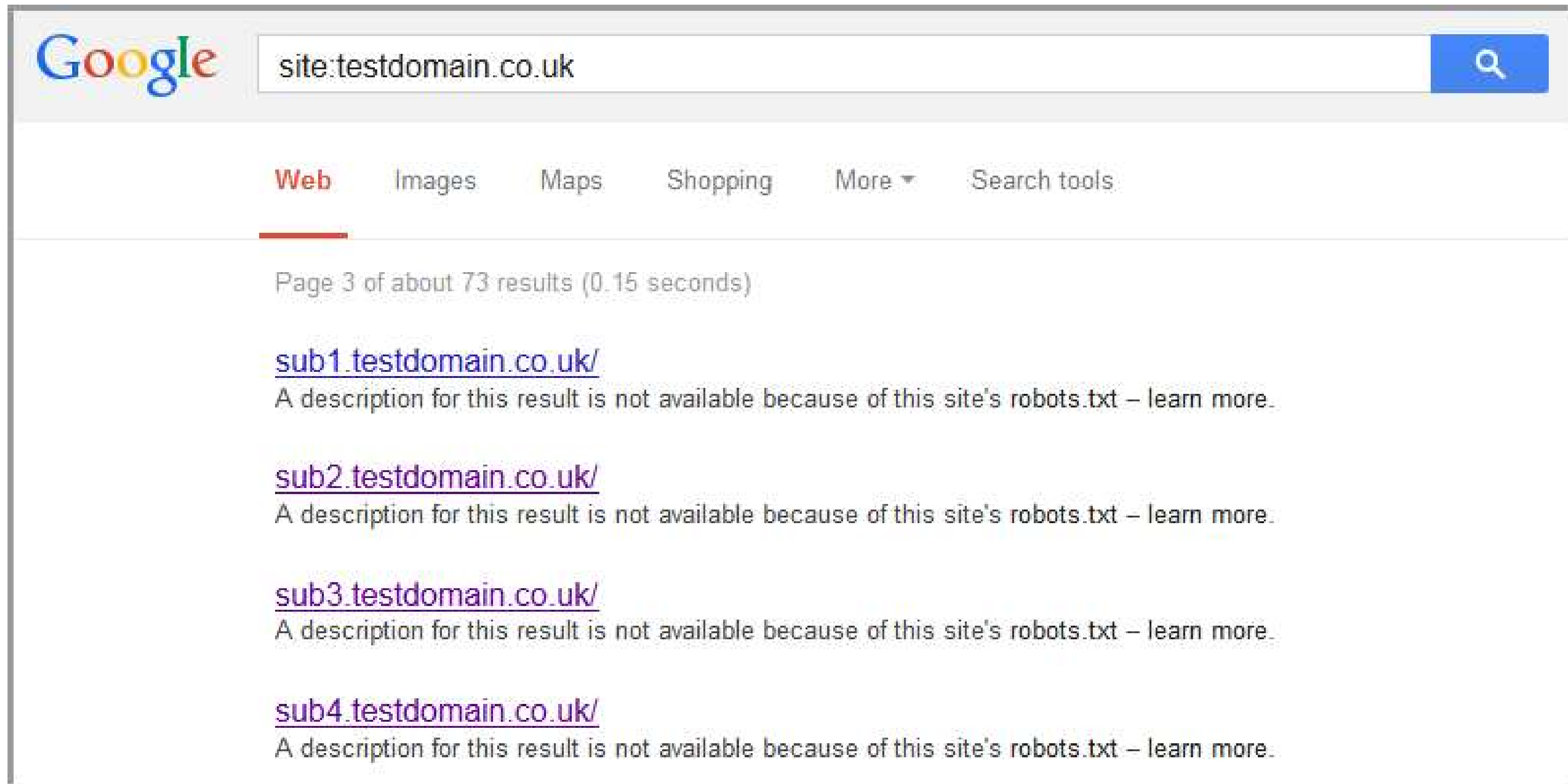
- If our web app is exposed to the public web but we don't want Google or other SEs to spider it, we can add a robots.txt file to the root of the web app with the following code inside:

```
User-agent: *
Disallow: /
```

https://moz.com/learn/seo/robotstxt

bucks new university

APPLYING SEO TO WEB APPS?



bucks
new university

USABILITY AND USER EXPERIENCE

bucks
new university

USABILITY AND USER EXPERIENCE

- **UX** = User Experience
- If we are looking at developing Progressive Web Apps, Google's PWA checklist is a good place to start to cover off all the essential technical and usability requirements we would expect from a good quality PWA

- **"Usability"** is part of the broader term "user experience" and refers to the ease of access and/or use of a product or website.

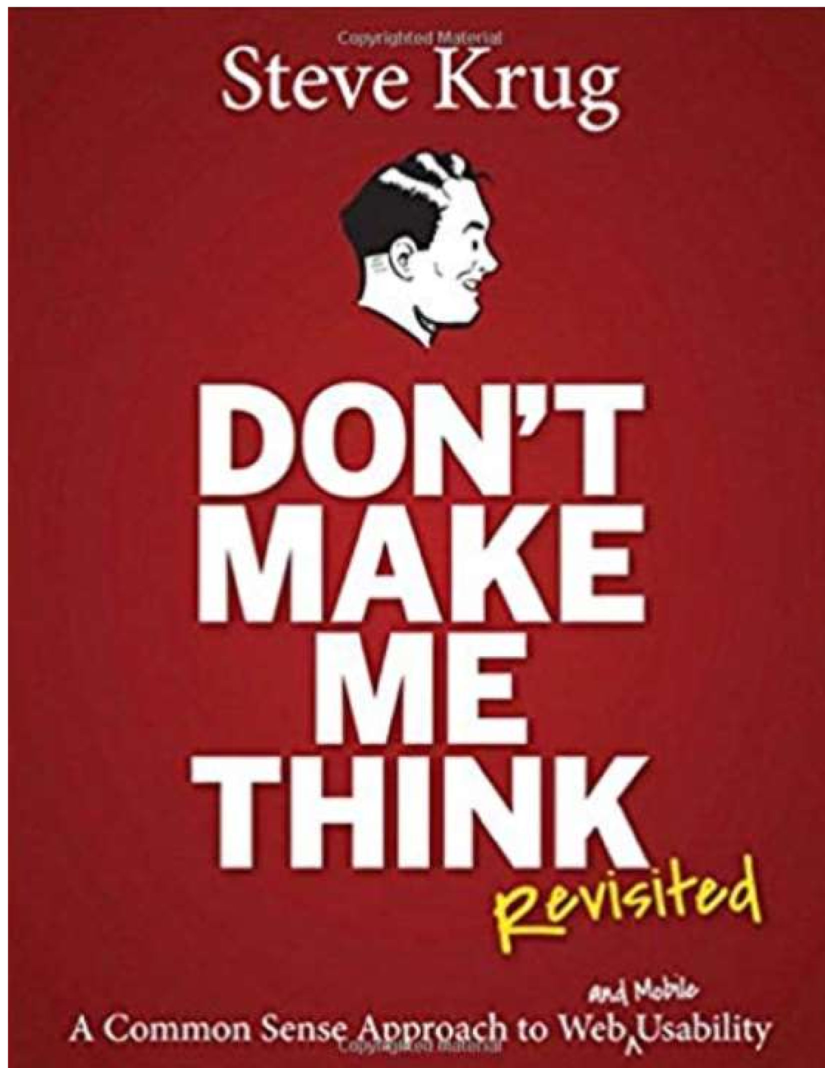
  Source: https://www.interaction-design.org/literature/topics/usability

Further reading:

https://developers.google.com/web/fundamentals/design-and-ux/ux-basics/

bucks new university

RECOMMENDED BOOK



bucks new university

HAPPY CODING!

bucks new university