Angular Helicopter View

Flash Introduction

Hello everyone. Vasco here from the Angular University, how is it going ? I'm super excited to bring you this book on Angular. I would love to help you on your Angular learning journey.

So let's get started right now exploring the extended Angular, Typescript and RxJs Ecosystem together, there are some exercises along the way and a code repository.

You can find the source code for this ebook in the following Github repository:

https://github.com/angular-university/courses

The goal of this ebook is for you to learn the Angular framework from the point of view of an Application Developer – why should you use the Angular framework and in which way does it help you build better and more feature-rich applications ?

Why Angular ?

We are going to take an innovative approach for learning a web framework: we are going to present the features of the framework of course, but our main points of focus will be:

- Why is a feature built it a certain way ?
- What problems is it trying to solve ?

- When should we use each feature and why ?
- How much do I need to know about the internals of Angular in order to be proficient using it ?
- Is a given feature commonly used in day to day application development, or is it meant mostly for third party library development ?

Let's give some practical and easy to understand explanations about to the multiple concepts around Angular, Typescript and Single Page Applications in general.

The goal is to give you a high-density of those "Aha!" moments, that are really what learning is all about.

So in a Nutshell, let's summarize the content of this book:

- We are going to start by giving you a convincing explanation on how single page applications work, and why would you want to build an application that way

- Then we are going to guide you through the Angular framework core module, covering everything you need to know to use it effectively: Components, Directives, Pipes, Services – this is the bread and butter of Angular on which everything else is based upon

- We are then going to build a simple service layer together, and cover some of the design principles that we need to use to be able to design Angular services effectively

- We will then cover the Angular Router, and learn how to use it in practice to build the navigation system of our applications

- One of the biggest use cases of Angular is for building form intensive applications. We are going to see how we can use Angular to build forms using the Forms module

- We will end by showing how to test the Components, Directives and Forms that we have been building so far

- We will be using the Angular CLI for our examples

I hope this makes it clear what the content of this book will be about.

More than covering a feature or concept, the goal of the book is to help understand not only the concept / feature, but also what are the common use cases, when should we use it and why, and what are the advantages.

Let's start with the Single Page Application concept itself:

Did you ever wondered what is the advantage of a building an application as a Single Page Application, or SPA ?

Let's start by answering that fundamental question – because that is one of the main reasons why we would want to learn Angular, right ?

I want to thank you so much for choosing this book , and so whitout further ado: let's get started learning the Angular Ecosystem together.

Thanks and Kind Regards,
Vasco | Angular University

Why a Single Page Application, What are the Benefits ? What is a SPA ?

Let's start with the first question: Why Single Page Applications (SPAs) ? This type of applications have been around for years, but they have not yet become widespread in the public Internet.

And why is that ?

There is both a good reason for that, and a even better reason on why that could change in the near future.

SPAs did get adopted a lot over the last few years for building the private dashboard part of SaaS (Software as a Service) platforms or internet services in general, as well as for building enterprise data-driven and form-intensive applications.

But does Angular then enforce building our applications as a SPAs ?

The answer is no, it does not, but that is an interesting possibility that it brings, given the many benefits of building an application that way. Which brings us to a key question:

Why Would You Want to Build an Application as a SPA?

This is something that is often taken for granted. Everybody is building applications this way, but what is the big advantage of that ? There has to be a at least a couple of killer features, right ?

Let's start with a feature that does not get mentioned very often: Production Deployment.

Understanding the advantages of SPA use in production will actually also answer the key question:

What is a Single Page Application?

Sometimes names in Software Development are not well chosen, and that can lead to a lot of confusion. That is certainly not the case with the term SPA: a single page application literally has only one page !!

If you want to see a Single Page Application in action, I invite you to head over to the AngularUniv https://angular-university.io and start clicking on the home page on the list of latest courses, and on the top menu.

If you start navigating around, you will see that the page does not fully reload – only new data gets sent over the wire as the user navigates through the application – that is an example of a single page application.

Let me give you an an explanation for what is the advantage of building an application like that and how does that even work.

Using the Chrome Dev Tools, let's inspect the index.html of the AngularUniv website that gets downloaded on the home page (or any other page, if you navigate and hit refresh).

```
<meta charset="UTF-8">
<head>
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" re
```

```
    <link href="//maxcdn.bootstrapcdn.com/font-awesome/4.6.2/css/font-aweso
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon"/>
    <link rel="stylesheet"
          href="https://angular-university.s3.amazonaws.com
                /bundles/bundle.20170418144151.min.css">
</head>

<body class="font-smoothing">

<app>... </app>

<script
        src="https://angular-university.s3.amazonaws.com
             /bundles/bundle.20170418144151.min.js"></script>

</body>

```

01.html hosted with by GitHub view raw

The page that gets downloaded is the very first request that you will see in the Chrome Network tab panel – that is the literally the Single Page of our Application.

Let's notice one thing – this page is almost empty ! Except for the tag, there is not much going on here.

> *Note: on the actual website the page downloaded also has HTML that was pre-rendered on the server using Angular Universal*

Notice the names of the CSS and Javascript bundles: All the CSS and even all the Javascript of the application (which includes Angular) is not even coming from the same server as the `index.html` – this is coming from a completely different static server:

> *In this case the two bundles are actually coming from an Amazon S3 bucket !*

Also, notice that the name of the bundles is versioned: it contains the timestamp at which the deployment build was executed that deployed this particular version of the application.

The Production Deployment Advantages of Single Page Applications

The scenario we have seen above is actually a pretty cool advantage of single page applications:

> *Single Page Applications are super easy to deploy in Production, and even to version over time !*

A single page application is super-simple to deploy if compared to more traditional server-side rendered applications: its really just one `index.html` file, with a CSS bundle and a Javascript bundle.

These 3 static files can be uploaded to any static content server like Apache, Nginx, Amazon S3 or Firebase Hosting.

Of course the application will need to make calls to the backend to get its data, but that is a separate server that can be built if needed with a completely different technology: like Node, Java or PHP.

Actually if we would build our REST API or other type of backend in Node, we could even build it in Typescript as well, and so be able to use the same language on both the server and the client, and even share some code between them.

Single Page Application Versioning in Production

Another advantage of deploying our frontend as a single page application is versioning and rollback. All we have to do is to version our build output (that produces the CSS and JS bundles highlighted in yellow above).

We can configure the server that is serving our SPA with a parameter that specifies which version of the frontend application to build: its as simple as that !

This type of deployment scales very well: static content servers like Nginx can scale for a very large number of users.

Of course this type of deployment and versioning is only valid for the frontend part of our application, this does not apply to the backend server. But still its an important advantage to have.

But is production deployment the only advantage of single page applications ? Certainly not, here is another important advantage:

Single Page Applications And User Experience

If you have ever used a web application that is constantly reloading everything from the server on almost every user interaction, you will know that that type of application gives a poor user experience due to:

- the constant full page reloads

- also due to the network back and forth trips to the server to fetch all that HTML.

In a single page application, we have solved these problem, by using a fundamentally different architectural approach:

On a SPA, after the initial page load no more HTML gets sent over the network. Instead only data gets requested from the server (or sent to the server).

So while a SPA is running, only data gets sent over the wire, which takes a lot less time and bandwidth than constantly sending HTML. Let's have a look at the type of payload that goes over the wire typically in a SPA.

For example, in the AngularUniv SPA if you click on a course, no HTML will be sent over the wire. Instead we get a Ajax request that receives a JSON payload with all the course data:


```
{
  "summary": {
    "id": 9,
    "url": "angular2-http",
    "description": "Angular RxJs Jumpstart",
    "longDescription": "long description goes here",
    "totalLessons": 15,
    "comingSoon": false,
    "isNew": false,
    "isOngoing": false,
    "visibleFrom": "1970-01-31T23:00:00.000Z",
    "iconUrl": "https://angular-university.s3.amazonaws.com/thumbnails/angu
    "courseListIcon": "https://angular-academy.s3.amazonaws.com/course-logo
  },
  "lessons": [...]
}
```

```
18
```

02.json hosted with by GitHub view raw

As we can see, the HTML version of a course would be much larger in size when compared to a plain JSON object due to all the opening and closing tags, but also there is a lot of duplicate HTML if we constantly loading similar pages over and over.

For example, things like the headers and the footers of a page are constantly being sent to the browser but they have not really changed.

So with this, we have here a second big advantage of a single page application: a much improved user experience due to less full page reloads and a better overall performance because less bandwidth is needed.

So Why Don't We Use SPAs Everywhere Then ?

If SPAs have so many advantages, why haven't they been adopted in a larger scale on the public internet ? There is a good reason for that.

Until relatively recently, search engines like Google had a hard time indexing correctly a single page application. But what about today ?

In the past there where some recommendations to use a special Ajax Crawling scheme that have been meanwhile deprecated. So is Google now able to fully render Ajax ?

In an official announcement, we have the information that Google search is now generally able to crawl Ajax, but there are some reports

that its yet not completely able to do so.

And the recommendation is to use Progressive enhancement instead. So what does that mean, at this stage is it possible to use SPAs in the public internet or not yet?

Search Engine Friendly SPAs ?

Its possible to have the performance and user experience of a SPA together with good SEO properties: the use of the Angular Universal pre-rendering engine allows us to:

- pre-render an application on the backend
- ship the HTML to the browser together with Angular
- and have Angular bootstrap on the client side and take over the page a a SPA

Will SPAs Become More Frequent In the Future ?

Imagine a SEO friendly version of Amazon that would not refresh itself at each page reload and with a much improved performance and user experience: that would have likely a huge positive impact on the time customers spend on the site !

So the technical benefits of SEO-friendly SPAs are significant and even more so on mobile, and we would expect that these type of SEO friendly SPA applications would become more frequent in the future also on the public internet.

But this leaves the last question of this section still unanswered. You must be thinking at this stage after reading the last few sections:

If only minimal HTML is loaded from the server at startup time, then how does the page keep changing over time ?

Which leads us to the last question of this section, and maybe the most important:

How Do Single Page Application Even Work?

Indeed, how can they work because we only loaded very little HTML from the server. Once the application is started, only data goes over the wire. So how does the new HTML come from ?

Because there has got to be new HTML being generated somewhere, as its the only way that the browser will change what its displaying, right ?

The answer is simple, and it has to do with the way that single page applications actually work when compared to traditional server-based applications.

On a traditional application (which includes the vast majority of today's public internet), the data to HTML transformation (or rendering) is being done on the server side.

On the other hand, Single page applications do it in a much different way:

> *In a SPA after application startup, the data to HTML transformation process has been moved from the server into the client – SPAs have the equivalent of a template engine running in your browser !*

And so with this information in hand, let's wrap up this section by summarizing the key points about single page applications.

Section Summary – Advantages of SPAs and the Key Concept About How They Work

Building our application as a SPA will give us a significant number of benefits:

- We will be able to bring a much improved experience to the user
- The application will feel faster because less bandwidth is being used, and no full page refreshes are occurring as the user navigates through the application
- The application will be much easier to deploy in production, at least certainly the client part: all we need is a static server to serve a minimum of 3 files: our single page index.html, a CSS bundle and a Javascript bundle.
- We can also split the bundles into multiple parts if needed using code splitting.
- The frontend part of the application is very simple to version in production, allowing for simplified deployment and rollbacks to previous version of the frontend if needed

And this just one possible deployment scenario of SPAs in production.

Other Production Scenarios

Other scenarios include pre-rendering large parts of the application and upload everything to a static hosting server, in-memory caching on the server of only certain pages, do versioning using DNS, etc.

Its today simpler than ever to make SEO-friendly SPAs, so they will likely have increased adoption in the future, in scenarios where SPAs have not been frequently used.

The Way SPAs Work

The way that single page applications bring these benefits is linked to the way that they work internally:

- After the startup, only data gets sent over the wire as a JSON payload or some other format. But no HTML or CSS gets sent anymore over the wire after the application is running.

The key point to understand how single page applications work is the following:

> *instead of converting data to HTML on the server and then send it over the wire, in a SPA we have now moved that conversion process from the server to the client.*

The conversion happens last second on the client side, which allow us to give a much improved user experience to the user.

I hope that this convinces you of the advantages of SPAs. If not please let me know and why that is not the case so I can better develop some of the points.

Also, at this stage I want to show you some code straight away. Let's build a small Hello World SPA in Angular, and take it from there. We are going to see how the concepts that we have introduced map to the small application that we are about to build.

This is just the beginning so let's get going. I hope you are enjoying this book !

Getting Started With Angular - Development Environment Best Practices With Yarn, the Angular CLI, Setup an IDE

This post is a guide for setting up a solid development environment, for having the best Angular learning and working experience.

Let's make sure that we have the best development experience possible and don't run into constant development environment problems.

This might be one of the largest obstacles for someone just arriving at the Angular Ecosystem, maybe even more so than the reactive concepts – getting that environment setup right from the start is crucial.

We are going to setup up a development environment that is easily upgradeable and causes minimum problems over time due to things like semantic versioning.

In this post we are going to:

- Setup a Node development environment
- Learn and install the Yarn Package Manager
- Setup the Angular CLI
- Scaffold our first Hello World Angular application
- Setup a IDE, we are going to be using Webstorm

Setting Up A Node Development Environment

In order to get the best possible development experience, and if you don't have node yet installed:I advise you to go the nodejs.org website and install the latest version of node:

Please make sure not to use the long-term support version (LTS), but the latest version instead. We will be using node for frontend tooling purposes and for running our development server, and for that specific purpose you will usually run into less issues if you use the latest version.

But, if you already have node installed (any version almost), there is a much better way to upgrade your node version than to run an installer - why you might not be able to in a company computer.

To avoid running into any issues with constant reinstallations of node, I suggest the following: instead of overwriting your current version of node with the latest one, lets use a simple command line tool that easilly allows to switch node versions.

Why use a command line node versioning tool ?

Using a tool like that has several advantages:

- It's very useful to be able to quickly change node versions if for example we have multiple projects to maintain on the same machine, and each project needs different versions of node.

- with such tool, upgrading to newer versions of node in the future will be much easier – we will not have to run an installer in your machine again

So first before starting, let's make sure that you have at least some version of node installed on your machine.

Installing The Nave Command Line Tool

If you are using a Linux or Mac environment, then let's go ahead and install the nave command line tool, which is available on npm:

The command to install this tool is:

```
npm install -g nave
```

Please remember that you might have to add "sudo" at the beginning of some of the commands in this book, if you are not an administrator on this machine.

If you are on Windows, you can instead install the nvm-windows tool that gives you equivalent functionality:

With a node version changing tool in place, lets then install a given version of node. For example, lets bring the version 7.9.0 to our development machine, and start a shell with it:

```
> nave use 7.9.0
##################################################
100.0%
installed from binary
```

```
> node -v
v7.9.0
```

If using nvm-windows, the command would then be:

```
nvm use 7.9.0
```

In both cases the result would be the same, we have just started a new shell with the chosen version of on the path.

> *if you are on Windows and looking for a Bash shell that does not take administration privileges to install, have a look at Git Bash*

We will now be able in the future to easilly upgrade versions – thats a great start ! Now let's continue laying the foundation of our development environment.

Npm is the official node package manager, but we will be using a different alternative. Let's learn why we would want to use the Facebook Yarn Package Manager instead.

Why Use The Yarn Package Manager instead of NPM?

While using npm, you might notice that sometimes you run into the following situations:

- You run npm install, and you get an error on the terminal, then you run the same command again and the error is gone. This is

sometime caused by proxies and network issues, but a lot of times its not the case

- you have a project running on on your machine in one folder, but you checkout the code to another folder and install it there, and its somehow not working there due to dependencies issues

- your production or staging environment build fails due to dependency related errors that you don't have on your local machine

- Another developer working on the same project is running into dependencies issues, but on your machine its working great: or the other way around

All these issues are related to a couple of inherent problems of the way that the npm package manager works by default.

The biggest difference between Yarn and NPM

Npm uses semantic versioning, which means that we will take upon installation a dependency version of a library that is on a certain range of versions.

While on paper this sounds like a good idea to enable getting continuous patch updates of dependencies, in practice it doesn't work as well as one could think.

Facebook used to run into these issues too in their internal builds, and so they have created an alternative package manager that they have first tested extensively internally and then open sourced: The Yarn Package Manager.

So let's try it out and learn the advantages along the way. To install Yarn, let's do the following:

```
npm install -g yarn
```

Yes I know, we used npm LOL I will likely be one of the last times though.

So we now have installed Yarn globally. Lets now use it as much as possible, because it will unlike npm "freeze" the dependency tree, and ensure that the code that runs in production and in other developers machine is the exact same code that runs on your machine as well.

So how does Yarn Work Then ?

Its actually quite simple, upon project installation Yarn will freeze the dependencies by making note of what libraries where installed in a file which is called `yarn.lock`.

So to install a file and generate a lock file, we simply have to run the following command on the same directory as your `package.json` is:

```
yarn
```

This does the same as an an npm install, meaning:

- it will inspect the package.json and calculate a tree of library dependencies

- it will download all project dependencies and write them under the `node_modules` folder

This is so far what npm does as well, so why use Yarn instead ?

Two immediate Advantages of Yarn

At this point, if nothing else yarn would really be bringing us two very important advantages:

- Yarn is much faster than npm, especially for projects with a lot of dependencies, like for example projects that are scaffolded using the Angular CLI

- Using Yarn you will not run anymore into that very frequent scenario of having to either run npm install multiple times until it completes (with a couple of cache cleans in the middle), or having to delete `node_modules` and reinstall

The Biggest Advantage Of Yarn

The improved speed and reliability alone would already make Yarn a better alternative. But those might not be the biggest advantage. The biggest thing about Yarn is that during the installation it will keep track about all the dependencies that it has installed.

Yarn will write the exact version numbers of each library that is has downloaded and it will put those in a text file, called a lock file. You will see it right next to your package.json once the installation completes.

A couple of things about the yarn.lock file

This is just a plain text file, here are some important points about it:

- we should not edit it manually, only Yarn should modify its file in result of a command that we execute in the command line

- we need to commit this file to source control

So what is the big advantage of having a lock file, and of the Yarn approach in general ?

Committing the file ensures that any other developer using the same code base will have the same exact dependencies than you. This is really convenient because problems due to library differences are much more prevalent than one could think.

And what about when we commit something and the integration build fails due to differences between libraries in the continuous server machine and our local machine, how are we going to debug that ?

Imagine the headache that such situation could cause, because you probably don't have access to the file system of the integration server.

Yarn advantages in a Nutshell

Keeping a lock file and using the same exact dependency tree everywhere avoids so many time-consuming problems. In a Nutshell, and to summarize the advantages of Yarn we can say that:

> *Yarn gives us faster, more reliable and most of all reproducible builds, meaning that we will never have to debug issues due to having different libraries on different machines*

About Npm Shrinkwrap

One final note about npm, its also possible to freeze dependencies in npm, but the process is reported to be non-deterministic, have a look here at the Yarn docs regarding shrinkwrap:

> *If you are using an npm-shrinkwrap.json file right now, be aware that you may end up with a different set of dependencies. Yarn does not support npm shrinkwrap files as they don't have enough information in them to power Yarn's more deterministic algorithm.*

Also have a look at this mention on the same docs about the Yarn lock file:

> *It's similar to npm's npm-shrinkwrap.json, however it's not lossy and it creates reproducible results.*

So I hope that this convinces you why we should use Yarn instead of npm. Let me know what you think of this approach, and lets continue setting up our development environment.

The next steps are: installing the CLI (and configure it to use Yarn), and installing an IDE.

Installing the Angular Command Line Interface

Now that we have a package manager in place, let's start using it to install everything that we need. To install the Angular CLI, which is a command line tool that we can use to scaffold Angular applications, we can run the following command:

```
yarn global add @angular/cli
```

At this stage, if everything went well we should have the Angular CLI available at the command line. If we run this command we should have:

```
>ng --version
```

This should return you the CLI version that you just installed.

> *WARNING: Yarn is a relatively new tool. If you run into issues installing global dependencies, such as for example you can't find the CLI executable, you can always do one more use of npm:*

```
npm install -g @angular/cli
```

Over time, this should not be necessary though. Now that we have the CLI installed, let's start using it.

Scaffolding Our First Angular Application Using the Angular CLI

The CLI will create a standard structure for our project, and setup a working build. The build needs dependencies, that will be downloaded by default using the npm package manager.

But its advisable to configure the CLI from the beginning to use the Yarn Package Manager instead:

```
ng set --global packageManager=yarn
```

Now if we use the multiple commands of the Angular CLI, we are going to be using yarn instead of npm. So things will go much faster and we

will have reproducible builds for our project. Lets then scaffold our first application:

```
ng new hello-world-app
```

This is going to take a while, but it will create a new project structure and it will install all the needed dependencies in one go. We now have a ready to use project ! We can run our application by simply doing:

```
cd hello-world-app


ng serve
```

The ng serve command should start a development server on your locahost port 4200, so if you go to your browser and enter the following url:

```
http://localhost:4200
```

You should see in the browser a blank screen with the message “app works!”.

We are going to review this application step-by-step in an upcoming post. But wouldn’t it be better to review the application already in our final development environment ?

So let’s get that in place first, and we will take it from there.

Setting Up an IDE – Webstorm or Visual Studio Code

There are a couple of great IDEs out there. For example there is the Microsoft Visual Studio Code IDE, which is available here and is free:

https://code.visualstudio.com

There is also the Webstorm IDE, with a couple of versions that you can try out, here is the free trial version:

https://www.jetbrains.com/webstorm

And here is the free Early Access Program Edition, with all the latest features not released yet of the next upcoming version. These early access versions are very stable despite the name:

https://confluence.jetbrains.com/display/WI/WebStorm+EAP

After installing the IDE, we just have to open the folder with the Angular CLI project in it, and a new project will be created.

Webstorm will automatically detect the Typescript version that you are using inside `node_modules`, and will use that to compile the code and show any errors.

An important feature of our IDE environment

This means there won't the need to configure Typescript manually, and end up accidentally having different compiler behavior between the command line and the IDE for example.

This is what we want to avoid especially if just getting started with the Angular ecosystem, we want an initial experience where installations go smoothly and the IDE just works.

Webstorm also has some great Angular integration, we can for example jump from the template directly to a component class method.

The installation of Webstorm should go smoothly, and with this we have a development environment in place that will the basis for for the exploring of the Angular framework that we are about to do.

Conclusion and What's Next

With this section you should have a solid development environment in place. If you didn't then please tell me in the comments what went wrong.

So let's continue by reviewing the application that was generated: What are all those files ?

You must be thinking, that's a lot of files and steps for only one Hello World application, right ? So let's address that using the generated application, because its a very important question to answer:

> *What is the advantage of using Angular when compared to jQuery for example ?*

Let's find out, because already in this simple application there is one killer Angular feature in action, that is probably the biggest differentiator towards previous generation technologies.

Why Angular? Angular vs jQuery - A Beginner-Friendly Explanation on the Main Advantages of Angular and MVC

When setting up your initial Angular development environment using the Angular CLI, you will realize how much the CLI has now become essential.

This is because we need so much more setup today, when compared to what we had before - and this is the case not only for Angular but also for any other equivalent ecosystem.

A couple of key questions

In this new world of module loaders, advanced build systems that we no longer setup ourselves from scratch, and where everything is transpiled down to plain ES5 Javascript from higher level languages, it would not be surprising if at a given point you would ask yourself:

> *Do i really need all these tools and dependencies ? Isn't there a simpler way, what is the advantage of doing thing this way ? Why Angular ?*

You might even ask yourself, why just not do everything in something like jQuery instead ?

There are many use cases that are still today well solved by that approach, but that lead us to the question:

> *In which way is Angular better than for example jQuery (and other equivalent previous generation technologies) ?*

What problems does Angular solve in a fundamentally better way, and why is it better ?

Everything happens under the hood

Let's find out ! The key thing about the main advantages of Angular and MVC, is that everything happens so much under the hood that its sometimes hard to realize that the advantages are even there.

In order to understand the benefits of the MVC approach, we are going to build a small application in jQuery, and its equivalent in Angular - and the differences between both will quickly become apparent.

An example of a jQuery Application

To understand the advantages that both Angular and MVC bring, let's give a simple example. Let's start with a simple application that just displays a list of lessons on the screen.

The lessons will be loaded via an HTTP GET request from this url. Actually this data is coming from Firebase via its alternative HTTP API (learn more about it in the Firebase Crash Course).

And this is what the list of lessons will look like:

```
{
    "-KgVwEC-dwWkLO4ZsQ9e": {
        "description": "Angular Tutorial For Beginners - Build Your First A
        "longDescription": "This is step by step guide to create your first
        "tags": "BEGINNER",
        "url": "angular-hello-world-write-first-application",
```

```
        "videoUrl": "https://www.youtube.com/embed/du6sKwEFrhQ"
    },
    "-KgVwEC0vq_chg0dvlrb": {
        "courseId": "-KgVwEBq5wbFnjj7O8Fp",
        "description": "Building Your First Angular Component - Component C
        "duration": "2:07",
        "longDescription": "In this lesson we are going to see how to inclu
        "tags": "BEGINNER",
        "url": "angular-build-your-first-component",
        "videoUrl": "https://www.youtube.com/embed/VES1eTNxi1s"
    }
  ....
}
```

01.json hosted with by GitHub view raw

What is the Model in MVC?

As we can see, the data above is simply a Plain Old Javascript Object, or POJO.

Those unique identifiers that you see are Firebase push keys, more on them on the Angular and Firebase Crash Course.

The JSOJ object above is the data of our application, also known as the Model, and we would like to display it on the screen - therefore we need to generate HTML based on this Model.

So let's start by doing so in jQuery first, and then implement the same logic in Angular.

What does a jQuery application look like?

Lets build a small jQuery application to display this data (the application is available here):

```
<div class="container">
    <div id="lessons"></div>
</div>

<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>

<script src="./index.js"></script>

```

02.html hosted with by GitHub view raw

So in our jQuery application the first thing that we need to do is to query our REST API backend by doing a jQuery Ajax request:

```
$(document).ready(function() {
      $.get( "https://final-project-recording.firebaseio.com/lessons.json",
            function( data ) {
               var lessons = Object.values(data);
               console.log(lessons);
               ...
           }
      );
});

```

03.js hosted with by GitHub view raw

Notice that we called `Object.values()`, this was so just to transform the returned result which was an object (and therefore a key-value dictionary) into a list of lessons.

> *Note: the order of the lessons is not guaranteed to be preserved*

With this we now have our Model in the browser, ready to be displayed - the list of lessons. As you can see, we did not send HTML over the wire from the server back to the client, instead we did a request to the server and returned only the data.

This is a critical notion, because the HTML is more than the data: the HTML is a particular representation of the Model, that we can call a View, and the same data could have multiple views.

The Model vs View Distinction

The same Model (the list of lessons) that we have shown above, could have multiple Views. Here are a few examples:

```
<!-- View 1 - A table with a list of lessons -->
<table class="table lessons-list">
    <thead>
    <tr>
        <th>Description</th>
    </tr>
    </thead>
    <tbody>
    <tr>
        <td>Angular Tutorial For Beginners - Build Your First App - Hello W
    </tr>
    <tr>
        <td>Building Your First Angular Component - Component Composition</
    </tr>
    </tbody>
</table>


<!-- View 2 - An aggregated view of the data  -->
<div>31 lessons</div>

```

04.html hosted with by GitHub view raw

As we can see, one View of the model is an HTML table, but another possible View could simply be the total number of lessons.

Also, each lesson itself can be used as a Model, itself with different views: we could have a lesson detail screen showing all the detail of a lesson, of just a row in a table - showing a summary of the lesson.

How to Display the View using jQuery?

We now have the data on the frontend, so we need to use it to create the multiple Views. Here is how we could generate an HTML table with a list of lessons in jQuery:

```
1
2  $.get( "https://final-project-recording.firebaseio.com/lessons.json",
3        function( data ) {
4            var lessons = Object.values(data);
5
6            var html = "<table class='table lessons-list'>' + " +
7                               "'<thead>" +
8                                   "<th>Description</th>" +
9                               "</thead>" +
10                              "<tbody>";
11
12           lessons.forEach(function(lesson) {
13                   html += '<tr>' +  +
14                               '<td>' + lesson.description + '</td>' +
15                          '</tr>';
16               });
17
18           html += '</tbody></table>';
19
20           $("#lessons").html(html);
21
22       }
23 );
24
```

05.js hosted with by GitHub view raw

Reviewing the jQuery application

As we can see, we are writing quite some code to take the model, and transform it into the multiple views of the data. We can see that this code although straightforward to write, has the following characteristics:

- This code is not very readable, its mixing several concerns
- this type of code is not very maintainable
- this is a lot of code, it makes for a significant portion of our application !
- we are building HTML by string concatenation and then passing it on to the browser, which will still have to parse it

This last point is important for the comparison in terms of performance, more on this later.

Enter Angular, how is it different?

So now let's rewrite this part of our application in Angular (the application can be found here).

This is only the key part of the application, there would be an Angular CLI folder structure around it, but its standard and generated for us. The Angular equivalent would then be the following:

```
@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html'
})
export class AppComponent implements OnInit {
  lessons: any[];

  constructor(private http:Http) {}

```

```
  ngOnInit() {
    this.http.get('https://final-project-recording.firebaseio.com/lessons.j
      .map(res => Object.values(res.json()))
      .subscribe(lessons => this.lessons = lessons);
  }
}

```

06.ts hosted with by GitHub view raw

So let's break this down and compare it to the jQuery version:

- we have here a class that knows both about the data retrieved from the backend and the HTML template needed to render it
- the HTTP request is being made via the Angular HTTP module
- the data is stored in a variable called `lessons`

> *But the biggest apparent different is that there is no trace of HTML in this code.*

The Angular View Generation Process

So how is the HTML being generated then ? Let's have a look at the `app.component.html` HTML template file:

```
<table class='table lessons-list'>
  <thead>
  <th>Description</th>
  </thead>
  <tbody>
  <tr *ngFor="let lesson of lessons">
    <td>{{lesson.description}}</td>
  </tr>
  </tbody>
</table>
```

07.html hosted with by GitHub view raw

As we can see, this is where the HTML is kept now. And its separated from the component code as it sits on a separate file altogether.

Notice that this template does have some expressions and syntax that does not look like the HTML that we write every day, such as for example:

- the `ngFor` directive that loops through the list of lessons
- the `{{lesson.description}}` expression, which is how we can output data to the view

These expressions are how Angular ties the data and the view together.

The MVC Terminology

Based on the Angular application above, lets then define some terms:

- the plain JSON object is the Model of our Application
- the template (or the output of its processing) is the View
- the Angular `AppComponent` binds the View and the Model together, and to use the MVC terminology (other terminologies exist, but this is the most common) this class can be called our `Controller`

Angular vs jQuery - comparing the two versions of the application

Now that we have two applications written in both jQuery and Angular, let's start comparing them.

Already in this initial stage, we can see several differences:

- the template of Angular is much more readable than the jQuery code
- the controller code does not generate HTML, it only fetches the data from the backend

The biggest difference just looking at the code is that in the Angular version there is a separation of concerns that does not exist on the jQuery version.

In the Angular version, the Model and the View are clearly separated and interact via the Controller class, while in the jQuery version all of these concerns are mixed in the same code.

But that is not the only advantage of using an MVC framework: the differences will become much more apparent if we start modifying the data.

What if we now Modify the Model ?

The separation of concerns of the Angular version is a great thing to have, and essential in a larger application.

But there is more going on in this small example that is not yet apparent: what if we now want to update the Model ? Let's say that now we want to change the title of each lesson, to include the sequential number of the lesson.

To make it simple, let's do this in response to a button. This is what the jQuery version would look like:

```
1  (function(){
2      var lessons;
```

```
3
4       $(document).ready(function() {
5
6           $.get( "https://final-project-recording.firebaseio.com/lessons.json
7             function( data ) {
8               lessons = Object.values(data);
9
10              $("#lessons").html(generateHtml(lessons));
11          });
12
13         $('#btn').on('click',function() {
14              lessons.forEach(function(lesson, index) {
15                  lessons[index].description = index + ' - ' + lessons[index]
16              });
17
18              $("#lessons").html(generateHtml(lessons));
19
20          });
21      });
22
23  }());
```

08.js hosted with by GitHub view raw

So let's break it down what is going on in this new version of the code:

- we have extracted the code to generate the HTML into a separate function called `generateHtml()`, so that we can reuse it (see the function below)
- we have stored the lessons data inside a module, so that it does not pollute the global scope
- we have added a click handler to a button, that modifies the data and then calls again the `generateHtml()` function

The function that we created for reusing the HTML generation logic looks like the following:

```
function generateHtml(lessons) {
    var html = "<table class='table lessons-list'>" +
        "<thead>" +
        "<th>Description</th>" +
        "</thead>" +
        "<tbody>";
    lessons.forEach(function(lesson) {
        html += '<tr>' +
                '<td>' + lesson.description + '</td>' +
            '</tr>';
    });
    html += '</tbody></table>';
    return html;
}
```

08-1.js hosted with by GitHub view raw

The problem with this type of code

It might not look like it, but this code is really fragile and hard to maintain: one single quote in the wrong place and we will still have a valid Javascript string that will look completely broken when rendered by the browser, as its not valid HTML.

This is the type of code that we don't want to have to write ourselves: its really brittle, even though it might look straightforward at a first glance.

Comparing with the Angular version

Now let's have a look at the equivalent Angular version of this code:

```
@Component({
  selector: 'app-root',
  templateUrl: 'app.component.html'
})
export class AppComponent implements OnInit {
```

```
7
8     lessons: any[];
9
10    constructor(private http: Http) {}
11
12    ngOnInit() {
13      this.http.get('https://final-project-recording.firebaseio.com/lessons.j
14        .map(res => Object.values(res.json()))
15        .subscribe(lessons => this.lessons = lessons);
16    }
17
18    updateCourses() {
19      this.lessons.forEach((lesson, index) => {
20        this.lessons[index].description = index + ' - ' + this.lessons[index]
21      });
22    }
23  }
24
```

10.ts hosted with by GitHub view raw

The `updateCourses` function is being called by a button that we added to the template:

```
1
2   <button (click)="updateCourses()">Update Courses</button>
3
4   ... remaining template unchanged ...
5
```

11.html hosted with by GitHub view raw

Reviewing the Angular data modification version

Like its jQuery counterpart, this version of the application is also adding an index to the description of each course (have a look at these ngFor features for an alternative).

What we did in this Angular version was: we simply updated the Model, and Angular has automatically reflected the Model modification directly in the View for us !

We did not have to call an HTML generation function, apply the HTML in the right place of the document, etc. This is one of the killer features of Angular.

> *we did not have to write any Model to View Synchronization code manually, that synchronization has been somehow done for us under the hood !*

So how does this work ?

Angular keeps the View in sync with the model for us at all times via its transparent change detection mechanism.

The way that this works is that Angular will automatically check the Model, to see if it has changed and if so it will reflect those changes in the view.

But besides the separation of concerns and the automated detection of changes and synchronization, there is something else fundamentally different between the jQuery and the Angular versions.

Maybe The Biggest Difference Between Angular and jQuery ?

We cannot tell just by looking at the Angular code because it all happens transparently, but one of the key differences is that unlike jQuery, Angular is doing the document modification in a much more efficient way than the jQuery version:

> *Angular is not generating HTML and then passing it to the browser to have it parsed, instead Angular is generating DOM data structures directly !*

This works much faster than building manually HTML, and then passing it to the DOM for parsing. By building DOM nodes directly, Angular is bypassing the HTML parsing step altogether. So the browser simply takes the ready to use DOM tree and renders it to the screen.

And more than that, it will do so in an optimal way, by trying to change just the HTML that absolutely needs to be changed:

> *Angular will not replace the whole DOM tree each time, it updates the DOM in an optimal way, depending on what parts of the Model have changed*

How is Angular rendering the View under the hood ?

In order to understand better what Angular is doing under the hood, let's do the following: lets go back to the jQuery version and try to do something similar to what Angular is doing - direct DOM manipulation.

The click handler code would now look something like this:

```
1
2  $('#btn').on('click',function() {
3
4      lessons.forEach(function(lesson, index) {
5          lessons[index].description = index + ' - ' + lessons[index].descrip
6      });
7
8      var lessonsDiv = document.getElementById('lessons');
9
10     // delete all child elements
11     while (lessonsDiv.firstChild) {
```

```
        lessonsDiv.removeChild(lessonsDiv.firstChild);
    }

    // apply the table directly to the DOM
    lessonsDiv.appendChild(generateDomTable(lessons));

});

```

12.js hosted with by GitHub view raw

So as we can see we are using directly the DOM API to remove the nodes under the `lessons` Div. Then we are replacing the table with a new table that was created using the `generateDomTable()` function.

And this is what that function looks like:

```

function generateDomTable(lessons) {

    var table = document.createElement('table');

    var header = table.createTHead();
    var row = header.insertRow(0);
    var cell = row.insertCell(0);
    cell.innerHTML = "Description";

    for (var i = 0; i < lessons.length; i++) {

        var row = table.insertRow(i);
        var col = row.insertCell(0);

        col.innerHTML = lessons[i].description;
    }

    return table;
}

```

13.js hosted with by GitHub view raw

If you would like to see this in action, clone and try locally the two sample applications (one application in Angular and the other in jQuery).

The Advantages of the Angular View Generation Process

Actually this is just a very rough approximation of what Angular is doing under the hood.

This function is regenerating the whole table, but what Angular does with its templates is that it will replace only the parts of the DOM tree that need to be modified, according to only the data that was modified.

So for example, if only an expression on the page title was modified, that is the only part of the DOM that will be affected, the list of lessons would remain unchanged.

Its just not practical to do manually what Angular is doing

And this is where it would become almost impossible and certainly impractical to do with jQuery the same thing that Angular is doing transparently under the hood.

Its simply not practical to write this type of Model to View DOM generation code manually ourselves like the `generateDomTable()`, despite the benefits of that approach.

For example, notice that the `generateDomTable()` function although it creates DOM nodes directly, its not optimized and generates the whole table each time.

Summary and Conclusions

So let's summarize: the advantages of using an MVC framework like Angular as we have seen are huge, so let's list them one by one.

Separation Of Concerns

We can build frontend applications with a lot less code than previous generation technologies. This is because we don't have to write code that manually synchronizes the Model and the View - that code is generated for us automatically.

The code that we do need to write will be a lot more readable and maintainable, due to the clear separation of concerns between the Model and the View.

Transparent Model to View Synchronization

The synchronization between Model and View is not only transparent but its optimized in a way that is not possible in practice to achieve manually.

UI Performance

The View is modified by generating directly DOM object trees in a cross-browser compatible way, instead of by generating HTML and then passing it to the browser for parsing - this effectively bypasses the parsing step altogether.

The HTML still needs to be parsed, but its done only once per template by Angular and by the browser each time. The parsing is done either at application startup time (Just In Time, or JIT), or ahead of time (AOT) when be build the application.

And this one of the main reasons why its so worth it to have a more advanced development environment like the CLI, because with it we can have all these features and benefits working transparently out of box.

Also, the generation and modification of the view is itself done in an optimized way, meaning that only the parts of the DOM tree that have modified data will be affected, while the rest of the page remains the same.

Why MVC has become essential for frontend development

With Angular and MVC we can focus on building applications in a more declarative way, by simply defining an HTML template just like we are used to.

We can them bind the template to the Model data, and handle the data via a Controller.

Is it doable not to use MVC ?

Choosing to not use an MVC framework means that we will have to keep the Model and the View in sync manually, which looks simple at first sight but very quickly we will end up with an unmaintainable program.

I hope that this gives you a convincing explanation of why we need an MVC framework like Angular, and the amount of problems that this type of technology solves for us in a completely transparent way.

And why it really pay off to setup a more advanced working environment, to be able to enjoy all these benefits and great developer experience.