JOHN BRYCE
Leading in IT Education
a matrix company

Angular Environment

@angular/cli - From Development to Production

Our Goal

JOHN BRYCE
Leading in IT Education
a matrix company

- Create a todo application with a single screen
- The screen display a todo list taken from the server
- The server address is:

  https://nztodo.herokuapp.com/api/task/?format=json

Todo list

Task 1
Task 2
Task 3

The data is taken from the server

That’s Too Easy!

JOHN BRYCE
Leading in IT Education
a matrix company

- Here’s the catch - the app should be production ready
  - AOT compiled
  - minified
  - cookie hash
  - node express server
  - server side rendering
  - gzipped

Problem - Creating New Angular App

JOHN BRYCE
Leading in IT Education
a matrix company

When creating a new Angular app, there are a lot of repetitive tasks we need to do in our project:

- Install webpack
- Install typescript
- Configure webpack and typescript to work together
- Install test framework
- Environment variables

Scaffolding a new Angular project and setting the above list can take a lot of time

Angular Scaffolding Tool - @angular/cli

JOHN BRYCE
Leading in IT Education
a matrix company

- Angular CLI is used to start a new Angular project
- We install Angular CLI with npm and usually install it as global npm package

  > npm install -g @angular/cli
- You can type **ng -v** to verify that it's installed
- You can type **ng help** if you want to see help on the available commands

@angular/cli - New Project

JOHN BRYCE
Leading in IT Education
a matrix company

- With @angular/cli you can start a new Angular project
- To start a new project:
  - **ng new <project-name>**
- The above command will create a new folder
- Let’s run the command and examine what is created.

What Our New Project Contains

JOHN BRYCE
Leading in IT Education
a matrix company

- **e2e -** we can create end to end tests using **protractor**. the tests we put in this folder
- **unit testing** - karma and jasmine are configured to run unit tests. unit tests file has a **spec.ts** extension
- **src/app** the application code is located in this folder
- **src/environments -** used for supplying changing variables based on environment
- **lint -** used to alert us on code styling, rules are set to apply for Angular code styling

Running Our App

JOHN BRYCE
Leading in IT Education
a matrix company

- To run our app
  - **ng serve**
- This will run our app at http://localhost:4200
- Additional flags
  - **--port, --host, --environment**
- You can check additional options for the command by running:
  - **ng serve --help**
- App will auto reload when changing files

Generating Components

JOHN BRYCE
Leading in IT Education
a matrix company

- We can use Angular CLI to create Angular architecture components
- During the course we won't use this functionality in order to better understand how to do things ourselves
- The syntax is:
  - **ng generate <type of architecture component> <name>**

Build

JOHN BRYCE
Leading in IT Education
a matrix company

- You can use Angular CLI to build your project
- The command is:
  - **ng build**
- A dist directory with the application will be prepared
- You can add **--prod** if you want the output minimized and tree shaking optimization

How @angular/cli Works

JOHN BRYCE
Leading in IT Education
a matrix company

- Behind the scenes @angular/cli use a bundling tool called webpack
- To understand webpack and how it works, let's try and review the problem we have with web applications

Problem - Many Files and Tasks

JOHN BRYCE
Leading in IT Education
a matrix company

We have many files in our web application and many steps we need to do with those files before serving them to the user:

- Style files
  - css which we need to minify
  - less, sass, scss which we need to compile to css and minify
  - push css files to CDN change the urls to point to the CDN
  - deal with caching
- Image files
  - we have to compress the image files and push them to CDN
  - after pushing to CDN we have to change the references to point to CDN
  - deal with caching
- src files
  - if our code is written in version >= ES6 we have to transpile our code to vanilla
  - TypeScript files have to be compiled
  - compiled code need to be pushed to CDN and references have to be changed
  - deal with caching
- Combine all the files of our web app to the minimum files needed

Possible Solutions

JOHN BRYCE
Leading in IT Education
a matrix company

- gulp / grunt
  - harder to combine src files to single file
  - hard to deal with tree shaking
- system.js
  - compile code on runtime is slow
  - doesn’t deal well with asset files
- Rollup
  - works similar to webpack
  - less plugins and community support
  - better tree shaking
- webpack
  - very popular in the open source community
  - a lot of contributors and plugins
  - easy to use and configure
  - The most popular solution for working with Angular

What is Webpack?

JOHN BRYCE
Leading in IT Education
a matrix company

- Bundling tool - takes your assets and combine them to minimum files needed
- During the bundling process webpack can perform actions on the assets before combining them
  - for example webpack can compile our scss files before creating a single css file
- Webpack build is not on run time
- Webpack is configured with a configuration file. The configuration files tells webpack
  - what is the entry point file
  - where to output the results
  - how to process the files
- Webpack starts with the entry point files and looks at the import / require statements to build a dependency trees of files in our application.
- From the dependency tree and the configuration file telling webpack how to compile the files it can go over the tree and create our app bundle

Why Use Webpack?

JOHN BRYCE
Leading in IT Education
a matrix company

- Multiple requests to load scripts reduce loading time
- Order of script loading can cause problems
- Script files css and images are only loaded when needed
- Webpack can compile also scss and less
- Can transform es6 code to vanilla
- Can compile Typescript

Webpack ex.

JOHN BRYCE
Leading in IT Education
a matrix company

- let's create a typescript app with two files:
  - **main.ts**
  - **person.ts**
- **person.ts** defines a **Person** class which **main.ts** is importing and using
- we will use webpack to create a compiled single **JS** file from these files.

Install Webpack

JOHN BRYCE
Leading in IT Education
a matrix company

- You can install webpack using npm
  - **npm install webpack --save-dev**
- This will expose the command **webpack** and you can verify webpack is installed by typing
  - **webpack -v**

Webpack Configuration File

JOHN BRYCE
Leading in IT Education
a matrix company

- Create new file: **webpack.config.js (**default name otherwise have to use --config flag**)**
- Configuration is in the block: **module.exports = {...}**
- Configuration keys:
  - **entry : string -** entry point file to compile
  - **Output : Object -** defines the output file configuration
    - **filename : string -** name of output file
    - **path : string -** the path to put the output file
- To run webpack using the configuration file type: **webpack**
- To run with watch for changes: **webpack -w**
- Let's try and configure webpack to bundle an hello world app which uses import statement

Webpack Loaders

JOHN BRYCE
Leading in IT Education
a matrix company

- Loaders instruct webpack how to process the files before bundling them
- With loaders we can tell webpack to compile our typescript files, to compile our scss files, to optimize our images etc.
- Loaders we use need to be installed with npm
- The loaders has to be specified in the config files
- In the config file the loaders configuration looks like this:

```
module: {
    rules: [
      { test: /\.css$/, use: 'css-loader' },
      { test: /\.ts$/, use: 'ts-loader' }
    ]
  }
```

- Let's see how we use some common loaders

Typescript Loader

JOHN BRYCE
Leading in IT Education
a matrix company

- We can tell webpack to preprocess all the files with **.ts** extension and compile them to **JS** before bundling them
- We can use the **awesome-typescript-loader**
  - **npm install awesome-typescript-loader --save-dev**
- You will have to also install typescript
  - **npm install typescript --save-dev**
- We will also need to add a typescript config file, the default one will do
  - **tsc --init**

Tree Shaking

JOHN BRYCE
Leading in IT Education
a matrix company

- Tree shaking is the process of dead code elimination
- We want to remove the code that will never run
- let’s try and add a function to the **person.ts** that will never be called
- try to compile your code in production mode
- is the function included in the created bundle?

Webpack Plugins

JOHN BRYCE
Leading in IT Education
a matrix company

- Webpack plugins can hook to webpack build process
- They can optimize files, change them, add new files
- Plugins can do everything that loaders can't and large part of webpack is built with plugins
- Plugins are objects with **apply** method, that method is called with webpack compiler
- Using the compiler you can access certain build steps
- Mostly we won't write plugins of our own but use community plugins
- Let's see an example of using plugins

extract-text-webpack-plugin

JOHN BRYCE
Leading in IT Education
a matrix company

- Extract text from a bundle to separate file
- We can use this plugin to separate our styling to a css file
- This plugin is special since we also need to use the plugin as a loader to instruct webpack that our style file will be used with this plugin
- We will combine this plugin with **sass-loader** to process our **scss** files
- After sass-loader process the scss to css we need to process the css with **css-loader**
- To chain loader we use the **!** mark and the order is from right to left
- We will also need to install **node-sass**
- Let’s try to create our scss styling project

Webpack Code Splitting

JOHN BRYCE
Leading in IT Education
a matrix company

- Entry value in the config can point to multiple files
- In the config file entry can be an object with the key as names and value as path to the different file paths
- Output can also be an object with filename as variable:
  - **output: {path: path.join(__dirname, 'build'), filename: '[name].bundle.js'}**
- Another useful variable is **[hash]** we can use that for caching

Webpack Caching

JOHN BRYCE
Leading in IT Education
a matrix company

- We want to add the **[hash]** variable to cache our files
- This place a random string on the file
- We don’t know what file names to include in our **index.html** files since the filenames keep changing
- We don’t want to manually change the **index.html**
- We can use **html-webpack-template** which will automatically add our scripts to the bottom of the index file

Webpack Caching

JOHN BRYCE
Leading in IT Education
a matrix company

- We want to add the **[hash]** variable to cache our files
- This place a random string on the file
- We don’t know what file names to include in our **index.html** files since the filenames keep changing
- We don’t want to manually change the **index.html**
- We can use **html-webpack-template** which will automatically add our scripts to the bottom of the index file

Webpack Debugging Our Code

JOHN BRYCE
Leading in IT Education
a matrix company

- Problem: it will be hard to debug our code with a generated distribution file
- Source-map are files that map from generated code to our source files
- We can add in the config:
  - **devtool: 'source-map'**
- Webpack will create source map for the generated files
- We can use chrome developer tools to place breakpoints in our source code

webpack-dev-server

JOHN BRYCE
Leading in IT Education
a matrix company

- Opens a server which tracks the files from webpack and will reload the browser when the files change
- To install: **npm install webpack-dev-server --save-dev**
- To launch: **webpack-dev-server**
- @angular/cli use webpack-dev-server when we run **ng serve**

Ex.

JOHN BRYCE
Leading in IT Education
a matrix company

- to practice what we learned about Webpack, try to install the **extract-text-webpack-plugin** and try to make webpack compile scss files to external css file
- try to install **html-webpack-template** and create an **index.html** which webpack will use to insert the hashed scripts it compiles

@angular/cli ejecting webpack config

JOHN BRYCE
Leading in IT Education
a matrix company

- @angular/cli use webpack as the build tool
- @angular/cli also uses **webpack-dev-server** and will reload the browser when source file change
- By default @angular/cli will hide the webpack configuration file and won't let you directly add configuration there
- @angular/cli does let you change options using the **.angular-cli.json** but not to modify the webpack configuration files
- Sometimes you will need to customize Angular CLI webpack configuration file, to do this you can run the command:
  - **ng eject**
- This will output the webpack configuration file and you can edit that file
- After ejecting build and run are done through the package.json scripts and not through the **ng** command

Building Our App for Production

JOHN BRYCE
Leading in IT Education
a matrix company

- As mentioned before we can build our app with @angular/cli using the command
  - **ng build**
  - will output: vendor.bundle.js 2.44 MB
- We can also add a flag **--aot** to the build
  - **ng build --aot**
  - will output: vendor.bundle.js 1.17 MB

More than 50% reduction in file size

What is the AOT flag?

Ahead of Time (AOT) Compiler

JOHN BRYCE
Leading in IT Education
a matrix company

- AOT compiler converts your Angular HTML and typescript code into efficient JavaScript code
- The build needs to be done before the browser downloads the code
- The Angular compiler can also compile the code during run time after the browser download the code (JIT - just in time compiler which is the default)
- The benefits on AOT
  - browser fast rendering - the browser doesn’t need to compile the code it downloads
  - fewer requests - css and templates are inlined
  - small file sizes - we don’t need to ship our app with Angular compiler
  - detect template errors
  - better security

Building Our App for Production

JOHN BRYCE
Leading in IT Education
a matrix company

- For production we want to build our code the following way
  - minified
  - tree shaking
  - build optimizer
  - AOT
- You can pass the flag --prod
  - **ng build --prod**
  - main.6443d4f6c6f512d596e8.bundle.js: 154kb

@angular/cli Environments

JOHN BRYCE
Leading in IT Education
a matrix company

- Environments let's you set specific data per environment
  - example you might want different backend url for dev and prod
- By default when you start a project @angular/cli creates a **dev** and **production** environments
- Those environment configuration files are located at the environments folder at the src folder
- To use that data you import the **environment.ts** and the data will be replaced according to the proper environment you choose
- You can add additional environments by modifying the **.angular-cli.json** file

.angular-cli.json

JOHN BRYCE
Leading in IT Education
a matrix company

- json files that holds @angular/cli configuration
- You can set your project here to work with scss
  - also you can run: **`ng set defaults.styleExt scss`**
- You can set options about generated components
  - always have a template inline
  - always have a change strategy on push
- You can control how the test and lints work
- Add global style files
- And more can be checked at the official docs:
  - https://github.com/angular/angular-cli/wiki/angular-cli

Serving Our Angular App

JOHN BRYCE
Leading in IT Education
a matrix company

- We saw earlier that we can server our app using @angular/cli
  - **ng serve**
- The server that will run is **webpack-dev-server**
- The server is good for development but you can't use it to serve your app on production.
- Time for us to build a server for our app and what better technology to use then node and express

Express

- Expressjs is a web framework to create server applications
- Expressjs is minimal, unopinionated framework
- The flow of writing an express app is the following:
  - create express app instance: **const app = express()**
  - make the app listen to a port: **app.listen(3000, () => {console.log('listening on port 3000')})**
  - attach routes to the app: **app.get('/', (req, res) => res.send('hello world'));**
  - attach middlewares: **app.use(cookieParser())**

Express - Hello World

JOHN BRYCE
Leading in IT Education
a matrix company

- Let’s create our first express app which prints “hello world”
    * import express
    * create and app
    * define a route and add implementation to send a string of hello world
    * listen to port

Express - Middlewares

JOHN BRYCE
Leading in IT Education
a matrix company

- Middlewares functions are functions that have access to the, **request object (req), response object (res),** and the **next** function, in application's request-response cycle.
- the **next** function will execute the next middleware
- middlewares can do the following:
  - Execute any code
  - Make changes to the request response objects
  - End the request-response cycle
  - Call the next middleware
- you can use **app.use** to load the middleware, notice that the order you place it is important

Express static middleware

JOHN BRYCE
Leading in IT Education
a matrix company

- to serve static files such as images, CSS files, and JS files, use the **express.static** built-in middleware function in Express
- **> express.static(root, [options])**
  - **root -** the root directory from which to serve static assets
  - **options -** options to add like headers
- we can use by attaching it as middleware to our app:
  - **> app.use('/static', express.static('public'))**

Express - Serving Our Angular App

JOHN BRYCE
Leading in IT Education
a matrix company

- To serve an SPA application we can static serve the index.html file and static js files that @angular/cli created when we built the app
- To serve a file we can use: **res.sendFile('absolute path')**
- To serve the static js files we can use **express.static** middleware
  - app.get(path, express.static('absolute path'))
- Our SPA needs to work with Angular router and also when we reload at a certain route we still need to serve the index.html file and return to the same page
- What will distinguish the routes to serve index.html and the routes to serve the static files?
- It's better practice to not use express to serve the static files rather use reverse proxy like nginx, varnish, or upload the files to CDN
- We can also use gzip middleware to compress the data we send back
- Let's create an express server to serve our Angular app

Express Templates

JOHN BRYCE
Leading in IT Education
a matrix company

- a template makes it easier to design an HTML page
- Using a template we can replace variables with actual values in our HTML
- express can work with many template engines, to set up an engine you need to do the following
  - load the engine with **app.engine('html', <template engine cb>)**
  - set express to work with that engine: **app.set('view engine', 'html')**
  - tell express where the template directory is located: **app.set('views', path.resolve(__dirname, 'dist'))**
- after setting the template engine, the **response object (res)** will have a **render** method that gets a path to the template to render
- we can look at our angular **index.html** as a template engine where instead of injecting variables we inject components

Server Side Rendering

JOHN BRYCE
Leading in IT Education
a matrix company

- With the server that we just create it let's run the server and examine our app in the browser
- Right click the page after it loads and choose to view the source file
- Our index.html contains minimal text and almost all the logic of the app is in our js files
- So? what's wrong with that?

Problem

- SPA initial load can be slow
- SEO

53% of mobile site visits are abandoned if pages take longer than 3 seconds to load. (Angular docs)

How can we improve those important problems that we have with Single Page Applications?

Without Server Side Rendering

JOHN BRYCE
Leading in IT Education
a matrix company

Usually in SPA the initial HTML we get from the server looks similar to this:

```
<html>
…
<body>
        <script src=”my-spa-app.js”></script>
</body>
</html>
```

- The body is empty and contains a download of a script
- The script is running and in charge of presenting the page to the user
- Initial load can be slow - we need to grab the html and js and only after that we need to run the js to render the page
- Unfriendly to search engines

With SSR

- with server side rendering the initial HTML we get from the server looks like this:

```
<html>
    ...
    <body>
        <h1>The full page is rendered by the server</h1>
        <p>we get a full html page like it was when there was no SPA</p>
        <p>We can also use server caching so the html will be sent faster</p>
        ....
        <script src="my-spa-app.js"></script>
    </body>
</html>
```

- the first html is sent by the server, after that the spa takes control and everything is loaded dynamically
- modern frameworks/libraries like Angular2/react support SSR - we have to make sure that the code we write is universal

How SSR Works

- Server takes index.html file
- Server turns the Angular components to HTML and place them in the **app** tag in the html
- Server will turn the components to HTML based on the route
- Server will not support browser events and browser specific api's
- You can consider the server page as a quick landing page and after the Angular script are downloaded and run we will hide the server page and present the page the browser created
- The server created page will pass support for the dynamic page created by the browser

@angular/cli universal

JOHN BRYCE
Leading in IT Education
a matrix company

- If your app is created with @angular/cli, adding universal support is a breeze
- @angular/cli has a new feature that will create SSR configuration for you
  - **> ng g universal universal-app-name**
- The command will do the following:
  - create **app.server.module.ts** with the **AppModule** wrapped in a **AppServerModule**
  - create entry point file **main.server.ts**
  - add the proper configuration in the **.angular-cli.json** for the server run by the backend
  - create a **tsconfig.server.json** file with the typescript configuration for the server app
  - install additional npm packages needed to run the app in the server
  - modify the **main.ts** to load the content for the browser after the dom is finished
  - modify the **app.module.ts** to add **BrowserModule.withServerTransition**
- Run the command: **> ng build && ng build --app=universal-app-name**

ServerModule

JOHN BRYCE
Leading in IT Education
a matrix company

- In the file: **app.server.module.ts** a server module is created
- The server module wraps your app and contains low level tools that can work in the server side as well
- The module will import the following
  - AppModule
  - ServerModule
- That module will be the root module for the server
- You can also add providers that will only be used in the server side

renderModule

JOHN BRYCE
Leading in IT Education
a matrix company

- **renderModule** can turn our **AppServerModule** to **HTML** string
- Ss first argument we give the module (**AppServerModule**)
- The second argument is options dictionary containing the url and document
- Let's use the **renderModule** to create a simple js file that will be run by our server

server.js

JOHN BRYCE
Leading in IT Education
a matrix company

```
require('zone.js/dist/zone-node');

require('reflect-metadata');

const { AppServerModule } = require('./dist-server/main.bundle');

const { renderModule } = require('@angular/platform-server');



renderModule(AppServerModule, {

  url: '/',

  document: '<app-root></app-root>'

}).then(html => {

  console.log(html);

});
```

ngExpressEngine

- **npm install @nguniversal/express-engine --save**
- **ngExpressEngine** is a template engine for angular
- when settings the engine **ngExpressEngine** accepts an option dictionary with the module we want to bootstrap (in our example **AppServerModule**)
- we want to set the engine for **html** extensions to **ngExpressEngine** this way we can point the **render** method to the generated **index.html**

ngExpressEngine

JOHN BRYCE
Leading in IT Education
a matrix company

```
//create the ngExpressEngine
app.engine('html', ngExpressEngine({
    bootstrap: AppServerModule
}));

// set the engine
app.set('view engine', 'html')

// set where are views are located, in our case where the index.html is
app.set('views', path.resolve(__dirname, 'dist'));
```

Setting the express route

JOHN BRYCE
Leading in IT Education
a matrix company

- We want for every route to serve the index.html rendered by the engine we set
- We can create an express route with '*' to catch everything
- We can use **Response.render** to render the html with **ngExpressEngine**
- **render** get's the view as first argument and the second argument is options for the engine and in our case we need to pass the request

```
app.get('*', (req: Request, res: Response, next) => {
    res.render('index.html', {req});
});
```

- Try to run your app again and click the view page source to examine the html sent from the server

SSR - Good Code Bad Bode

JOHN BRYCE
Leading in IT Education
a matrix company

- Our app is now universal and this requires a certain shift in the way we write code
- Say we want to create a directive that will be placed on buttons and will add a loading spinner on buttons
- There is a jquery plugin that does that which is called **ladda**
- That plugin will require us to use **jquery**
- Let's install **jquery** and **ladda** and create our directive
- In the **.angular-cli** in the browser app you can add the scripts you installed
- You can use those scripts to create our directive

Ladda Directive in Server Side?

JOHN BRYCE
Leading in IT Education
a matrix company

- In our constructor in the directive we will initiate ladda on the **ElementRef**
- The directive will accept an input with a setter that will activate the ladda based on the boolean input
- After creating our directive try build the app and run the universal app

Bad Code


```
@Directive({
 selector: '[nzLadda]'
})
export class LaddaDirective {
 @Input('nzLadda') public set isLoading(val: boolean) {
  if (val) {
    this._l.ladda('start');
  } else {
    this._l.ladda('stop');
  }
 }
 private _l: any;
 constructor(private _element: ElementRef) {
   this._l = $(this._element.nativeElement).ladda();
 }
}
```

Bad Code

ReferenceError: $ is not defined

Good Code

JOHN BRYCE
Leading in IT Education
a matrix company

```
@Input('nzLadda') public set isLoading(val: boolean) {
 if (!isPlatformBrowser(this._platform)) {
   return;
 }
 if (val) {
   this._l.ladda('start');
 } else {
   this._l.ladda('stop');
 }
}
private _l: any;
constructor(private _element: ElementRef,
@Inject(PLATFORM_ID) private _platform: Object) {
 if (isPlatformBrowser(this._platform)) {
   this._l = $(this._element.nativeElement).ladda();
 }
}
```

Good Code

JOHN BRYCE
Leading in IT Education
a matrix company

- Our code needs to run in the server side
- If we use api's specific to the browser we need to branch our code for server and client
- To do this we can inject **PLATFORM_ID**
- We can use **isPlatformBrowser** or **isPlatformServer** to know where we are running now

Routing

- Proper web app should render the correct page when we reload it
- Routes should still work
- Our server side should render the proper page when reloading the app and create a full html for that page
- Let’s create another page called about page and check if our server can deal with routes

Server Side Optimizations

JOHN BRYCE
Leading in IT Education
a matrix company

- One of the reasons for SSR is increasing the initial load of our application
- Now that the server renders the first view we can add additional optimizations for our app.

gzip

- Helps decrease the size of the response body
- Reducing the size of the body will increase the transfer speed of the resources to the browser
- gzip can be configured as an express middleware or at the reverse proxy level like in nginx
- Let's add express gzip middleware to our express app
  - **npm install compression --save**
  - **const compression = require('compression')**
  - **app.use(compression())**

Caching

JOHN BRYCE
Leading in IT Education
a matrix company

- It’s better to handle caching with the reverse proxy
- This way a lot of the requests will not even pass to our express app
- This will save our express app to handle more important issues

Summary

JOHN BRYCE
Leading in IT Education
a matrix company

- Starting an Angular app with @angular/cli is a breeze
- Making your app production quality is a bit more challenging
- Let’s try to return to the task at the first slides and create our todo app while following these practices:
  - SSR
  - AOT
  - Tree Shaking
  - minification
  - gzipped
  - Caching (at least on the browser side you can try to upload the files to CDN as a bonus)