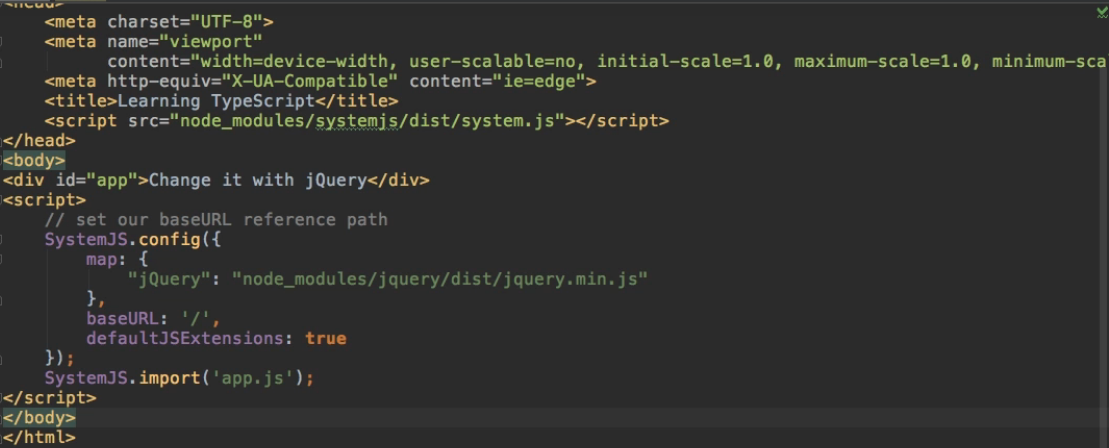
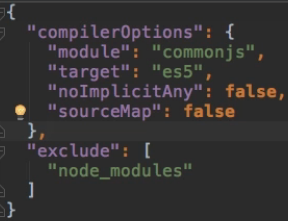
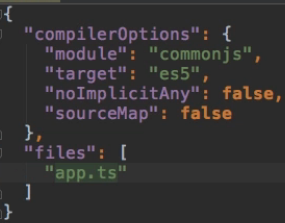
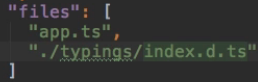
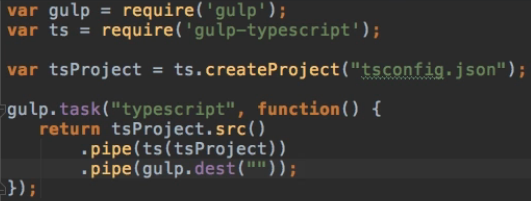
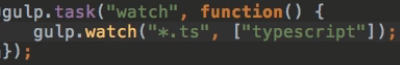
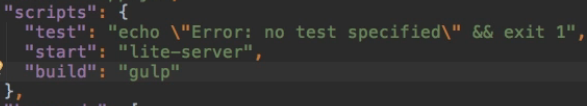
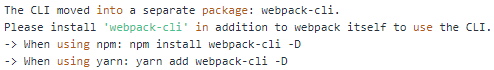
**Introduction**  
\* In this section, I want to stay in the world of TypeScript project management or projects, workflows and so on.  
\* We’re going to have a look at how we may integrate TypeScript into various workflows, including a pure TypeScript compiler usage workflow where we only use the TypeScript compiler and see how we can configure that to work really nicely, how to integrate it in a Gulp workflow or how to integrate TypeScript with Webpack.

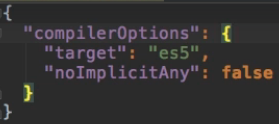
**Using “tsc” and the tsconfig File**  
**pure TypeScript workflow**  
\* So without a task runner like Gulp or a bundler or a dependency management systme like Webpack.  
\* This might be all you need for a lot of projects.  


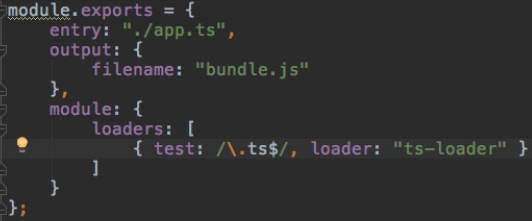
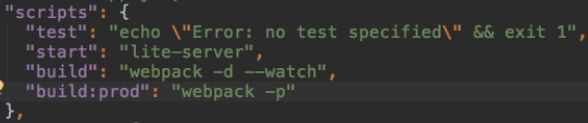
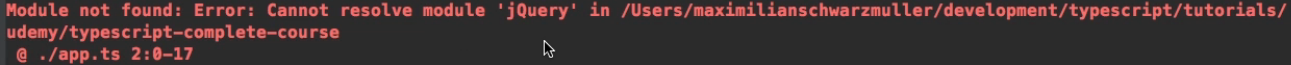
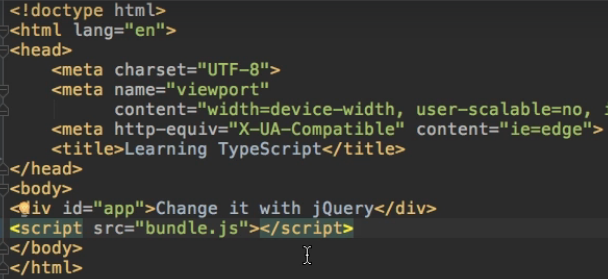
  
**tsc** => **compile our TypeScript code => compiles app.ts to app.js which then gets loaded in the index.html file.**  
\* By using “tsc”, we’re basically using TypeScript as our only workflow tool.  
\* We don’t have a task runner watching for any changes and then recompiling or something like this.  
**tsc -w => enter watch mode, very helpful.**  
=> **We’re now watching all TypeScript files for changes and whenever we make such a change, we’ll re-trigger compilation**.  
\* That’s something you definitely want to use when only using TypeScript.  
\* **Pure TypeScript workflow also means that you have to understand how the tsconfig.json file - the place where you configure your TypeScript project, actually works**.  
  
\* **I’m not talking about the compiler options here**.  
\* **What I want to consider is this exclude here or generally how we can control with the tsconfig.json which TypeScript files in your project should get compiled**.  
**How TypeScript resolves Files using the tsconfig.json File**  
**tsc** => when we run it like this, it basically loosk for the tsconfig.json file and then reads it to learn how it should behave.  
\* Here we have some compilerOptions and then the exclude.  
\* **This file tells the TypeScript compiler command that it should compile all TypeScript files in this folder except for any files in the node\_modules folder**.  
\* **So all other files in all other folders in this project folder will be considered during compilation**.  
\* This is the reason why in our app.ts file, we automatically detect the jQuery typings which livei n the typings folder. We never import the index.d.ts file which refers to the jQuery typings, it gets found automatically by TypeScript simply because we only specify what not to include.  
\* It won’t compile the .d.ts files because it recognizes that .d.ts is not a file that should be compiled but a file which holds some definitions and should be taken into account during compilation.  
\* So this is how it knows how to work with your directory, with your project and which files to take into account.  
\* Why do we exclude the node\_modules folder?  
=> Because there are a lot of Third-party libraries or packages by other creators and we probably don’t want to recompile those. Those packages should ship in a finished mode, in a finished version.  
\* We are including certain things like jQuery explicitly with the import statement in app.ts which then is considered when compiling but it’s not re-compiled because we don’t want to do this.  
\* **An alternative to excluding would be to instead explicitly tell it what to include**.  
  
\* **If we now run “tsc”, we get an error about $ jQuery because we only take the app.ts file into account because we’re not automatically scanning for all the .ts or d.ts files anymore**.  
\* We can make it work again by adding:  
  
\* So this is how you can control which files are taken into account when you run the “tsc” command.  
**“files”:  
“exclude”:**  
\* You control what you compile and what to include in the compilation step or what not to look at.  
<http://www.typescriptlang.org/docs/handbook/tsconfig-json.html>  
\* There are some additional examples and the “exclude” and “files” node is explained.  
\* Generally you should keep in mind that if you don’t specify the files node here, it will take all files into account.  
\* Then you an bit a bit more precise by adding “exclude” and for example excluding the node\_modules.  
\* Generally, it will always look for .ts and .d.ts files when compiling your project.

**More on “tsc” and the tsconfig file**  
\* There’s 1 other approach to compile files.  
\* **You can run “tsc” and then target a specific file**.  
**tsc app.ts**  
\* **If you do this, your tsconfig file will not be used**.  
\* **You can use the compilerOptions with -- prefixes in that command**.  
  
\* What if I want to run “tsc” but my tsconfig.json file is not positioned in the root folder?  
  
=> Now we get an error because by default TypeScript doesn’t scan your whole computer for this file, it only scans the folder in which you’re running the “tsc” command.  
**--project path  
-p path**  


**Adding TypeScript into a Gulp Workflow**  
**npm install --save-dev gulp gulp-typescript**  
\* **Add gulpfile.js**  
\* gulp-typescript is a wrapper around TypeScript.  
<https://www.npmjs.com/package/gulp-typescript>  
\* It wraps TypeScript so that it contains the TypeScript compiler and simply adjusts it to work in a Gulp workflow but it works the same way as the normal TypeScript works, it is the normal TypeScript, just wrapped to work with Gulp in the end.  
\* **There are some options like `watch` which are not supported because you should use gulp.watch**.  
  
\* **I don’t use any source here because I want to use the automatic file resolving of the TypeScript compiler**.  
=> We’re telling Gulp: please use the tsconfig file which is connected to the TypeScript compiler and then resolve all the files the tsconfig.json file thinks we need to compile.  
=> And then use the TypeScript compiler and compile all these files, taking the tsProject (our tsconfig.json file) into account.  
=> And then output it in the root folder “”.  
\* You don’t have to use the tsconfig file for how to resolve the files, you could also specify specific files.  
\* You can also pipe other tasks like bundling, uglifying, compiling CSS, etc.  
  
=> This will basically do the same as “tsc -w”.  
  
\* package.json:  
 **npm run build / npm start**

**Working with Webpack 4+**  
\* If you're using Webpack 4.x (check the package.json file to find out which version was installed), you'll need a slightly different setup than shown in the next lecture.  
\* Here are the adjustments you'll have to put into place:  
\* 1. Change loaders to rules.  
\* 2. In extensions[0] to "\*" from "".  
\* It's also suggested to keep the filename as ./bundle.js as a dist folder is added automatically.  
  
\* When running Webpack, you probably get an error:  
  
\* Simply install webpack-cli, thereafter it should work:  
**npm install webpack-cli --save-dev**

**Adding TypeScript into a Webpack Workflow**  
**npm install --save-dev webpack ts-loader**  
\* It’s a loader you can use in Webpack to compile TypeScript code.  
  
\* **Bundle.js will be created for us by Webpack**.  
\* I no longer need the “exclude” here because Webpack will handle all the bundling, importing and compilation. Webpack will automatically resolve for us which files it needs to bundle.  
\* Same with the source-maps.  
\* We no longer need the “module” key in tsconfig.json because we no longer need to compile our code to use any JavaScript modules because we don’t use a modue loader like systemjs anymore.  
\* So instead, Webpack will import all the files we need and bundle them for us.  
  
**webpack.config.js**

\* In the next section about ReactJS and TypeScript, you will also see a more advanced or a slightly different version of this webpack.config.js file.  
  
\* **This is the very basic setup we need**.  
\* If you’re also having a project where you are importing other TypeScript files, you might want to add Module Resolution or the File Extension Resolution to automatically search for .ts files.  
\* If you import something like jQuery and this were a jquery.ts file, you might want to add the respectice configuration here in the webpack config file to autoamtically tell Webpack that it should also look for .ts files and you will see an example of this in the next section.  
\* **package.json**  
  
**npm run build**  
  
=> The reason for this error is that the ts-loader doesn’t use our global TypeScript installation, it looks for a local one and with a local, I mean a project-specific one which has the huge advantage that if we have different versions, let’s say in our project we’re using TypeScript 1.7 and globally we have version 1.8, we’re not interfering with this, we’ll always use our local project-specific version.  
**npm install --save-dev typescript**  
**npm run build**  
  
\* Now we’re only importing jQuery in our app.ts, we’re not setting it up with System in the html file.  
  
\* By default Webpack uses the NodeJS import syntax.   
  
=> This will search through our node\_modules where we have a jquery folder and it will automatically find the file we need to use and to get this to work correctly.  
\* **Now there are no errors**.  
\* But we don’t see the color change.  
=> We need to move the bundle.js import to the end of the html page to only import it and run through it once the page has been completely loaded:  
  
\* Now it all works and we’re now in a watch mode.

**Module Summary**

**Resources**  
tsconfig.json  
<http://www.typescriptlang.org/docs/handbook/tsconfig-json.html>  
gulp-typescript  
<https://www.npmjs.com/package/gulp-typescript>