**Angular**

Typescript:

Typescriptlang.org/docs/handb…

* To install globally via npm: (but preferably install locally)

In the Terminal of your file (test.ts) in VS Code:

npm install -g typescript / npm install typescript -g

npm uninstall typescript -g (removes the package)

* To install locally

npm init (to initialize the project first) => package.json is created

npm install typescript => node\_modules is created

in package.json under dependencies: typescript is shown

* To check your version of the installed typesctipt:

tsc --version

* To compile your code: (compilation output is plain JavaScript => test.js)

tsc ./test.ts (typescript compiler on test.ts file)

* When installed locally to get access to tsc you need to use npx instead of npm (which has locally installed typescript that will run it)

npx tsc test.ts

Or use: ./node\_modules/.bin/tsc ./test.ts

Or in package.json scripts: “build”: “tsc test.ts” and then npm run build

* When compiling we can set certain configurations for the output file. Those can be stated in the Terminal command or all in a separate tsc config file

Angular:

Creating a new app

* Install globally via npm

npm install -g @angular/cli@latest=> creates ng command that can be used further

ex. ng --version

* Create new project

ng new some-app

cd some-app (enter the directory)

code .(open the app in VS Code)

* Start dev server on port 4200

ng serve / ng s => localhost:4200

* Building the app in production environment

ng build -c production

* When we need to refresh vs code, press ctrl+P, in the field type “>reload window” (if not working install Angular language service plug-in) (angular.json “strict:true”)
* Continue: angular project

Files in Angular project:

* tsconfig.json – settings/configuration how the typescript files are compiled, also how the angular is compiling tsc to js
* tsconfig.app.json – it is extending the tsconfig.json file and is used for our app
* tsconfig.spec.json – it is extending the tsconfig.json file and is used for running tests of our app
* angular.js – configuration for the angular cli, how our project is build, if any additional scripts to be included in the bundle; external libraries we list in scripts, like google maps example. File replacement is mostly used for “environments” viriables
* editorconfig – for the IDE
* browserslistcr – for which browsers is our app building
* app – the main component
* app -> styles.css – if we want a style to be applied globally, this is where we write it
* app -> main.ts – shows in what environment is our app working (browser, server, other)
* app -> pollyfills.ts – tracks async operations in our app
* Index.html – the starting point of our app
* In the Terminal ng build => scripts are built in index.html > scripts (under app-root) and our app is bootstrapped
* App -> app.module.ts – a class with Ng decorator that connects all the elements in this module
  + declarations(stores all the components that we want to use in the module)
  + imports – browser module for browser app, that contains browser specific things that we use to run our app
  + providers – array of different service providers
  + bootstrap – the component that we want to use to bootstrap or to start the app

**import { NgModule } from '@angular/core';**

**import { BrowserModule } from '@angular/platform-browser';**

**import { AppRoutingModule } from './app-routing.module';**

**@NgModule({**

**declarations: [**

**AppComponent**

**],**

**imports: [**

**BrowserModule,**

**AppRoutingModule**

**],**

**providers: [],**

**bootstrap: [AppComponent]**

**})**

**export class AppModule { }**

* continue

Using Google maps example: by importing the google.maps api

* In angular.json => “scripts”: [“ ” ->/\*link for the google maps location js file\*/]
* In index.html => after body we can add the script that will load the location

Components:

* A component is a class with Component decorator and contains 3 properties (selector, templateUrl, syleUrls). Or it can be called a directive with a specific template

**import { Component } from '@angular/core';**

**@Component({**

**selector: 'app-root',**

**templateUrl: './app.component.html',**

**styleUrls: ['./app.component.css']**

**})**

**export class AppComponent {**

**title = 'Article Site';**

**}**

* Creating a component:
  + when manually creating component files (test.component.ts, .html, .css) it has to be registered in app.module.ts> NgModule > declarations > TestComponent(the exported class name in the component.ts file) => this way we can use <app-test> selector in the .html files
  + when using the Angular CLI: ng generate component home (cd is articles-app that was created with ng new articles-app > cd articles-app)

a new folder is created scr/app/home and the cli directly imports the component in the app module

Data binding: in the .html file

* Using interpullation with {{}}, ex. {{title}} – used only for showing text somewhere in our template, i.e a property that has string value
* Using [] for property values, ex. <input [value] = “title”> </input>, i.e we give value to the input that is taken from the title property of the class in the .component.ts file
* Using () for event change, ex. <button (click)=”buttonClickHandler()”>Click Here</button>, where in the class in the .component.ts file we need to create the function for this event, i.e. buttonClickHandler(): void{ console.log(‘Button was clicked’);}

Templates – a form of HTML that tells Angular how to render the component. \*ngIf and \*ngFor come from the app.module.ts>imports>BrowserModule, that contains all common modules such as ngIf and ngFor and others

* To render nested properties of an object
* Using \*ngFor to render array properties

**export class GamesComponent {**

**games : Game[];  
 constructor() {**

**this.games = [ *// Array of games* ]**

**}**

**}**

**<h1>Games List</h1>**

**<p>Pick a game to Buy</p>**

**<ul>**

**<li \*ngFor="let game of games">**

**{{game.title}}**

**</li>**

**</ul>**

* Using \*ngIf for condition statements

**<h1>Games List</h1>**

**<p>Pick a game to Buy</p>**

**<ul>**

**<li \*ngFor="let game of games">**

**<div>**

**{{game.title}}**

**</div>**

**<span \*ngIf="game.price >= 100">**

**Price: {{game.price}}**

**</span>**

**</li>**

**</ul>**

* To attach events and handle them in the component

**<button (click)="showContent($event)">Show Content</button>**

**export class GamesComponent {**

**public games: Game[];**

**showContent: boolean;**

**constructor() {**

**this.games = [ *// Array of games* ]**

**}**

**showAdditionalContent($event) {**

**this.showContent = true;**

**}**

**}**