COMP-3411 / 4704
Web Programming II

Lecture 11 - Introduction to TypeScript and Angular

Daniel Pittman, Ph.D., CISSP

02/08/2022

General Points of Order

- At this point in the class we will be changing our focus from the **server** folder to the **client** folder!
- The rest of this course will be dedicated to covering Angular, Bootstrap, and Unit/Integration testing
- The UI we will be building in the labs over the next few weeks will use the API we have developed as of Lab 09
- Please make sure that the API is working in your local project!
    - Some issues I've seen:
        - Some students have not changed server/config/environment/index.js to **enable MongoDB**
            - **This prevents the API defined in Lab09 from working!**
    - If you think you're code is working locally, make sure you have committed all of your changes
        - From the root of your project run:
            - **git commit -am "Committing latest changes"**
            - **git push -u origin**
        - Need to see if you have files outstanding? Run **git status** from the root of your project

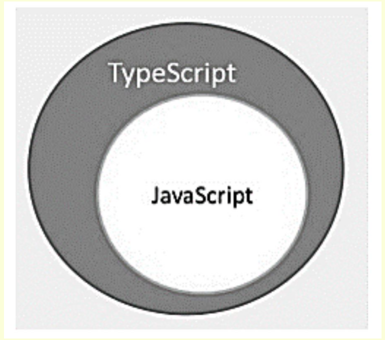
TypeScript Overview - What is TypeScript?

- Simply put, TypeScript is a **superset** of JavaScript, that **compiles** to plain JavaScript
  - "TypeScript is JavaScript for application-scale development"
- With Node.js, we introduced the ability to run JavaScript on the server, which is great!
- Unfortunately, JavaScript lacks many of the features that prevent it from succeeding at the enterprise level, such as:
  - Object Orientation
  - Strong type checking
  - Compile-time error checks
- TypeScript was created to bridge that gap in functionality

https://www.tutorialspoint.com/typescript/typescript_overview.htm

TypeScript Overview - What is TypeScript?

- Typescript is a **strongly-typed, object-oriented, compiled language**
- It was originally designed by Anders Hejlsberg (who also designed C#) at Microsoft
- TypeScript is both a **language** and a **set of tools**
- Being a **superset** of JavaScript, it contains all of the abilities of JavaScript, plus some additional features!



https://www.tutorialspoint.com/typescript/typescript_overview.htm

TypeScript Overview - What is TypeScript?

- **TypeScript is just JavaScript**
  - All you need to know in order to write TypeScript is JavaScript
  - All TypeScript code is converted to its JavaScript equivalent for the purposes of execution
- **TypeScript supports JavaScript libraries**
  - Compiled TypeScript code can be imported by any other JavaScript code
  - TypeScript-created JavaScript can import and use any existing JavaScript frameworks, tools, or libraries
- **JavaScript is TypeScript**
  - Any existing **.js** file you have can be renamed to **.ts** and be compiled with other TypeScript files
- **TypeScript is portable**
  - TypeScript can run on any environment JavaScript can, without the need of a dedicated VM

https://www.tutorialspoint.com/typescript/typescript_overview.htm

TypeScript and ECMAScript

- TypeScript is aligned with the **ECMAScript 6 (ES6)** specification
- The basic language features of TypeScript are in line with ECMAScript 5 (ES5) - the official specification for JavaScript
- TypeScript features such as **classes, Modules, etc...** are in line with ES6
- TypeScript also includes features such as **generics and type annotations**, however, which are not part of the ES6 spec

https://www.tutorialspoint.com/typescript/typescript_overview.htm

Why Use TypeScript?

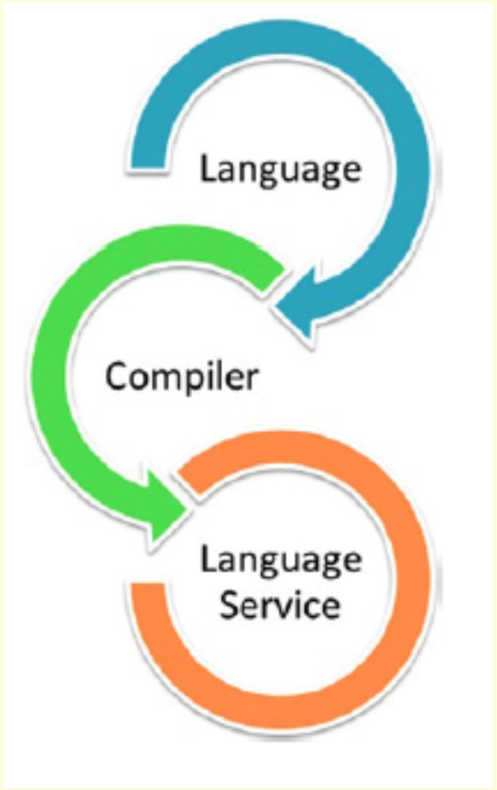
https://www.tutorialspoint.com/typescript/typescript_overview.htm

- Starting with Angular version 2, the recommended language to use is TypeScript
  - All examples online, and documentation, will show TypeScript
  - It is **possible** to write Angular 2+ using vanilla JavaScript, but you will be mostly **on your own** looking for documentation if you do!
- Beyond that, there are several good features that TypeScript brings over other languages such as CoffeeScript or Dart:
  - TypeScript is **extended JavaScript**, whereas CoffeeScript and Dart are completely new languages requiring a language-specific execution environment
  - TypeScript is a **compiled language**. JavaScript is an **interpreted language**. JavaScript code has to be run in order to test if its valid, whereas the TypeScript transpiler can check for errors at compile-time
  - TypeScript supports **strong static typing**. JavaScript is not strongly typed. TypeScript comes with an optional static typing and type inference system through the TypeScript Language Service (TLS)
    - The type of a variable, if declared with no type, can be inferred by the TLS based on its assigned value
  - TypeScript supports **Object Oriented Programming** concepts such as classes, interfaces, inheritance, etc...

Components of TypeScript

https://www.tutorialspoint.com/typescript/typescript_overview.htm

- TypeScript consists of three core components:
  - **Language:** The syntax, keywords, and type annotations of the TypeScript language
  - The **TypeScript compiler (tsc):** Converts the instructions provided in TypeScript to its JavaScript equivalent
  - The **TypeScript Language Service (TLS):** The TLS provides an editor-like layer around the core compiler, providing features such as statement completions, signature help, code formatting and outlining, colorization, etc...



TypeScript Declaration Files

- When you compile a TypeScript file there is an option to create a **declaration file** (file extension **.d.ts**) that acts as an **interface** to the components within the compiled TypeScript
- This is similar to the concept of a header file in C/C++
- The declaration files provide **intellisense** for types, function calls, and variable support for existing JavaScript libraries such as jQuery

https://www.tutorialspoint.com/typescript/typescript_overview.htm

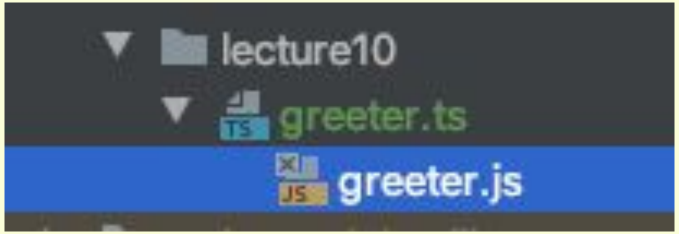
TypeScript in 5 Minutes

- Let's go through a quick tutorial to get more familiar with TypeScript syntax!
- This tutorial is based on "TypeScript in 5 minutes" available at

  https://www.typescriptlang.org/docs/handbook/typescript-tooling-in-5-minutes.html
- First, let's create a folder in our project named **lectures/lecture10**
- In that folder, download the following files from Canvas and put them in the folder:
  - **greeter.ts**
  - **greeter.final.ts**
  - **greeter.html**

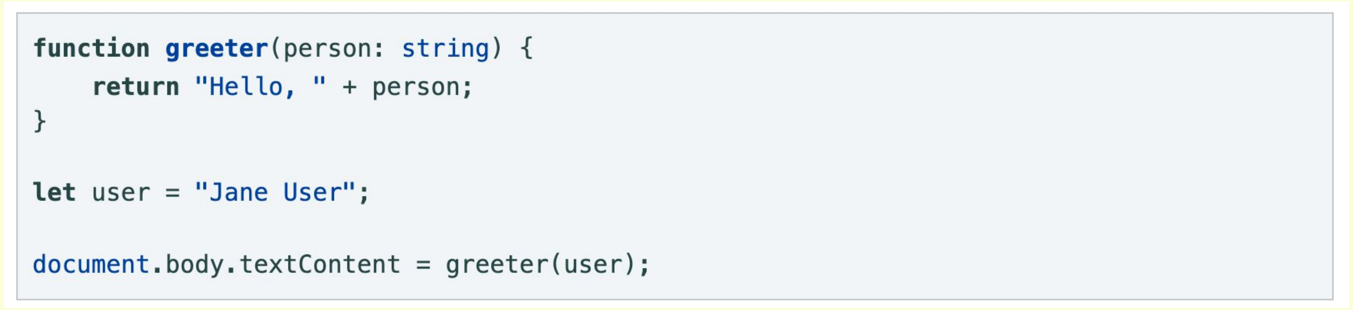
TypeScript in 5 Minutes

- Let's start off by looking at **greeter.ts**
- As you can see, in its current form it's just JavaScript! We could have named this file with **.js** and it would have worked
- Let's compile the file and see what's created
- **npx tsc greeter.ts**
- This will create a file named **greeter.js** that will have the same code as is in **greeter.ts**
- Notice how IntelliJ is nice enough to visually show **greeter.js** is created from **greeter.ts**



TypeScript in 5 Minutes

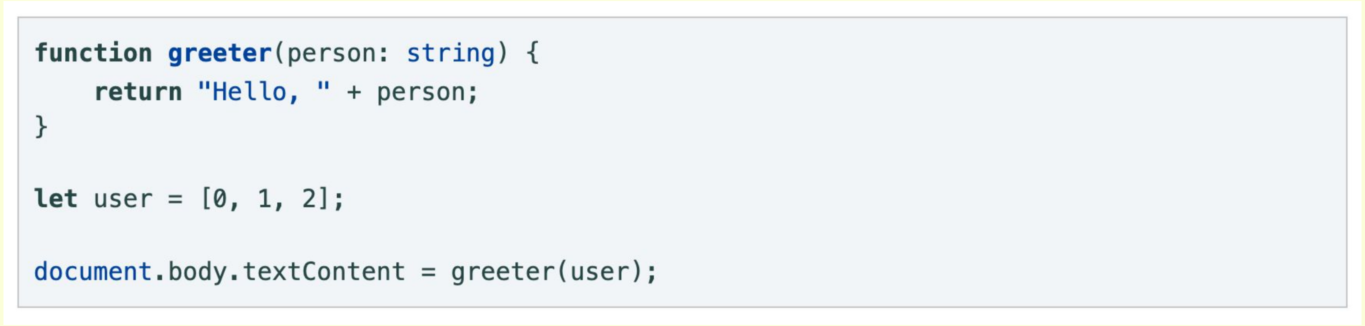
- OK! We've compiled our first TypeScript file! Now let's add some TypeScript features to it
- Let's add a **: string** type annotation to the **person** function
- A **Type Annotation** in TypeScript is a way to annotate the **intended contract** of a function or variable
- In this case, we are intending the function **greeter** to be called with a single **string** parameter
- Notice when we recompile the **.ts** file that the type is removed in the **.js** file!
    - The type information is compile-time only - JavaScript doesn't understand it!



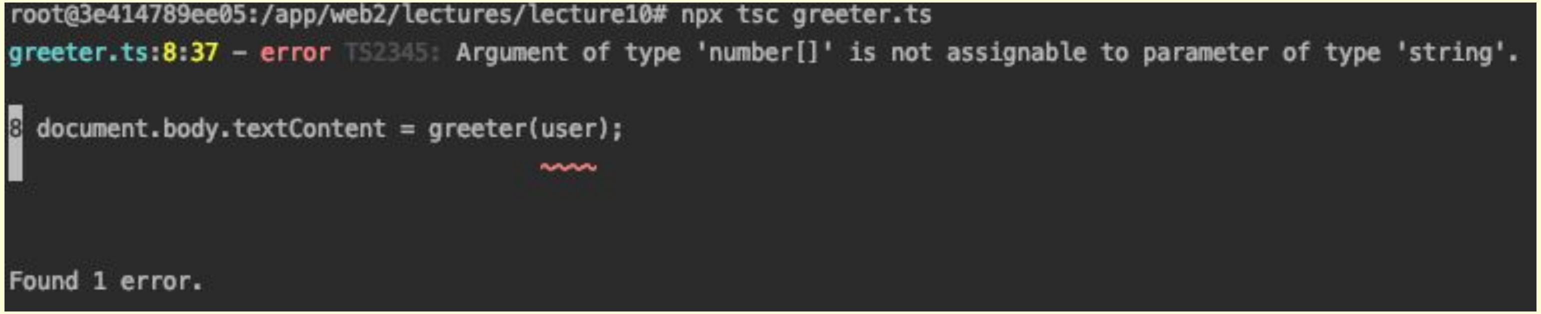
```
function greeter(person: string) {
    return "Hello, " + person;
}

let user = "Jane User";

document.body.textContent = greeter(user);
```

TypeScript in 5 Minutes

- Let's try to call the greeter function passing an array of values, and see what the compiler says:



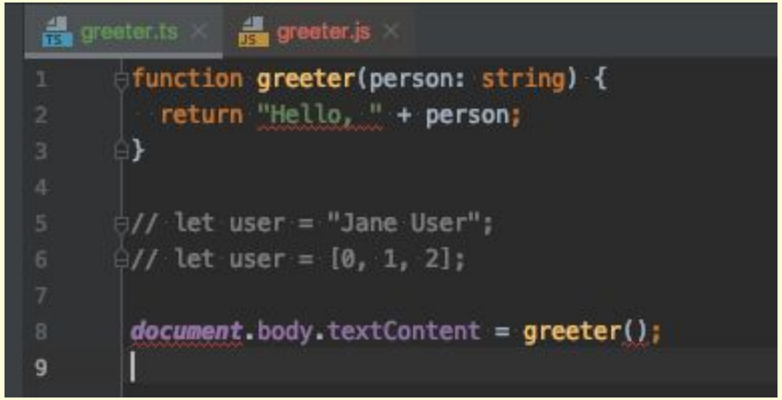
```
function greeter(person: string) {
    return "Hello, " + person;
}

let user = [0, 1, 2];

document.body.textContent = greeter(user);
```



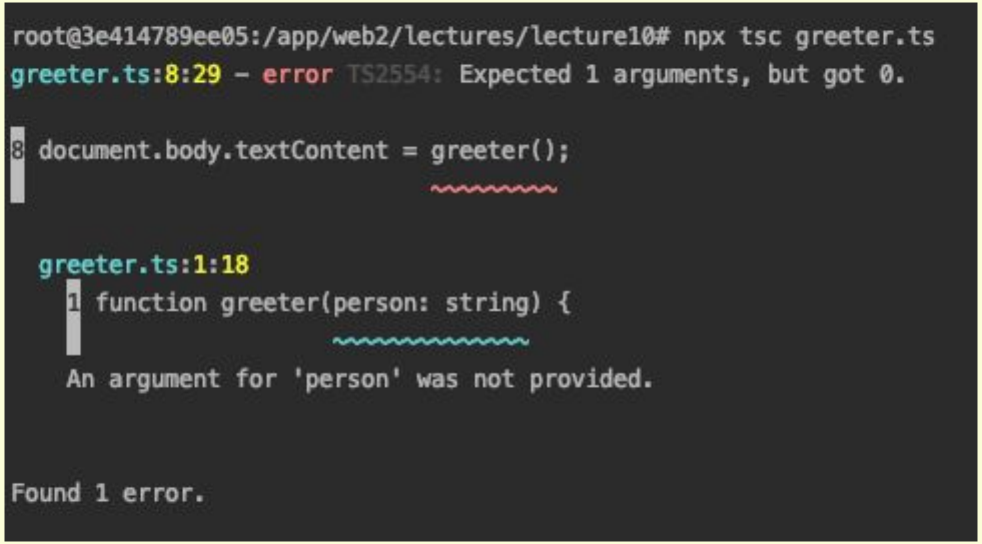
```
root@3e414789ee05:/app/web2/lectures/lecture10# npx tsc greeter.ts
greeter.ts:8:37 - error TS2345: Argument of type 'number[]' is not assignable to parameter of type 'string'.

8 document.body.textContent = greeter(user);
                                      ~~~~


Found 1 error.
```

TypeScript in 5 Minutes

- We can also try removing all of the parameters from the **greeter** function call:



```
function greeter(person: string) {
  return "Hello, " + person;
}

// let user = "Jane User";
// let user = [0, 1, 2];

document.body.textContent = greeter();

```



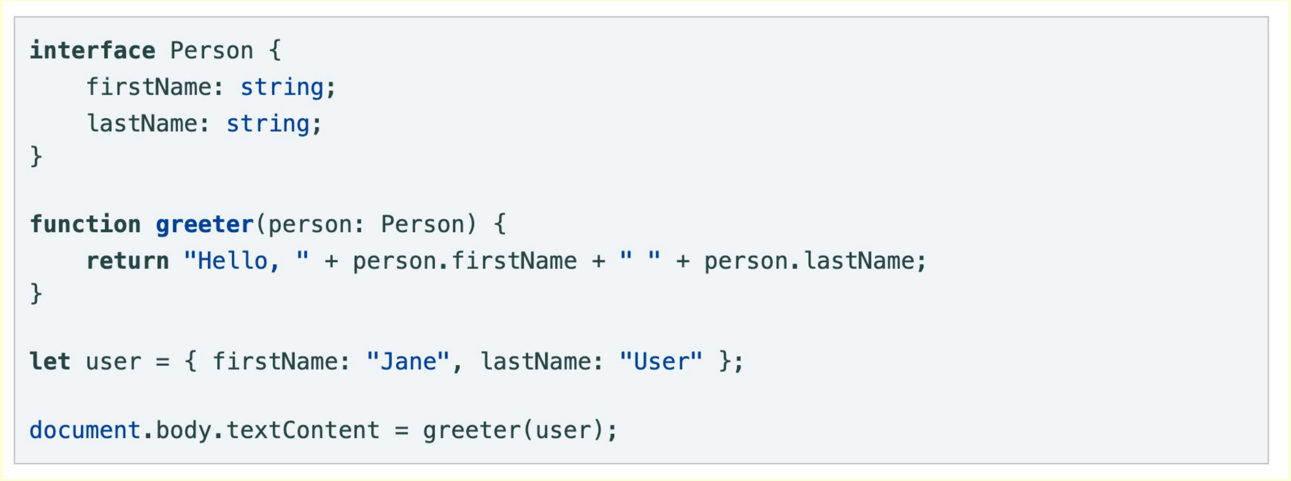
```
root@3e414789ee05:/app/web2/lectures/lecture10# npx tsc greeter.ts
greeter.ts:8:29 - error TS2554: Expected 1 arguments, but got 0.

8 document.body.textContent = greeter();
                              ~~~~~~~~~


  greeter.ts:1:18
    1 function greeter(person: string) {
                       ~~~~~~~~~~~~~~
    An argument for 'person' was not provided.


Found 1 error.
```

TypeScript in 5 Minutes

- In both error cases, TypeScript provided **static code analysis** based on both the structure of the code itself, and the annotations we provided
- This enables us to "fail fast" during development, not having to run the code to see there's something wrong with it
- You might also notice that, despite there being an error in the .ts code, a .js file was still created
- You can use TypeScript even when there are errors in your code, but TypeScript will warn you that your code most likely will not run as expected!
- In our project template, however, the **gulp build** task will not complete if you have errors - a protection against publishing bad code!

TypeScript in 5 Minutes

- Let's make **greeter.ts** a little more complex by adding an **interface** that describes objects with a **firstName** and **lastName** field
- In TypeScript, two types are considered **compatible** if their internal structure is compatible
- This allows us to implement an interface by just creating an object with the attributes that are required (no **implements** keyword needed!)

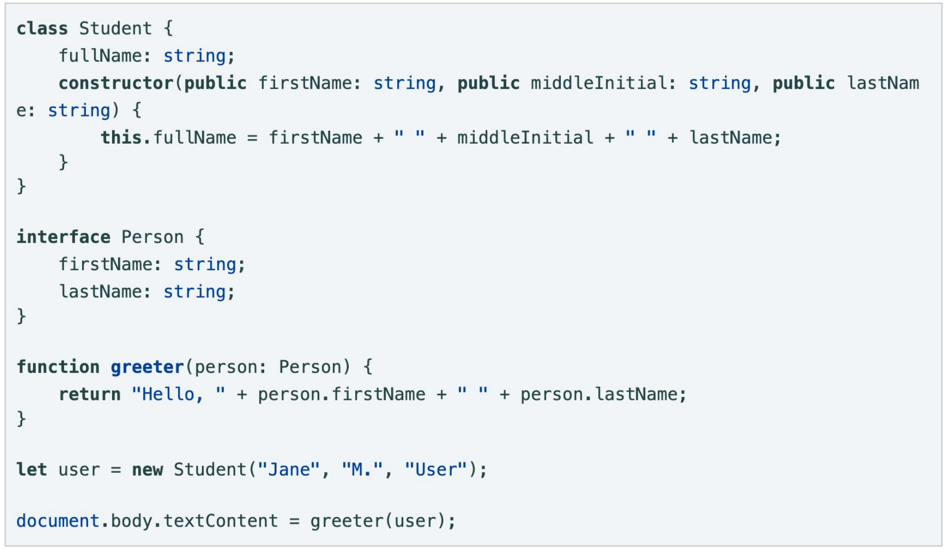


```
interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = { firstName: "Jane", lastName: "User" };

document.body.textContent = greeter(user);
```

TypeScript in 5 Minutes

- Finally, let's augment our example one last time to include **classes**
- TypeScript supports ES6 JavaScript features, including class-based Object-Oriented programming techniques
- Let's create a **Student** class with a **constructor** and a few **public fields**
    - Notice how the **class** and **interface** work nicely together
    - Also note the **public** on constructor arguments, which is a shorthand in TypeScript to create properties in the class with the same name!

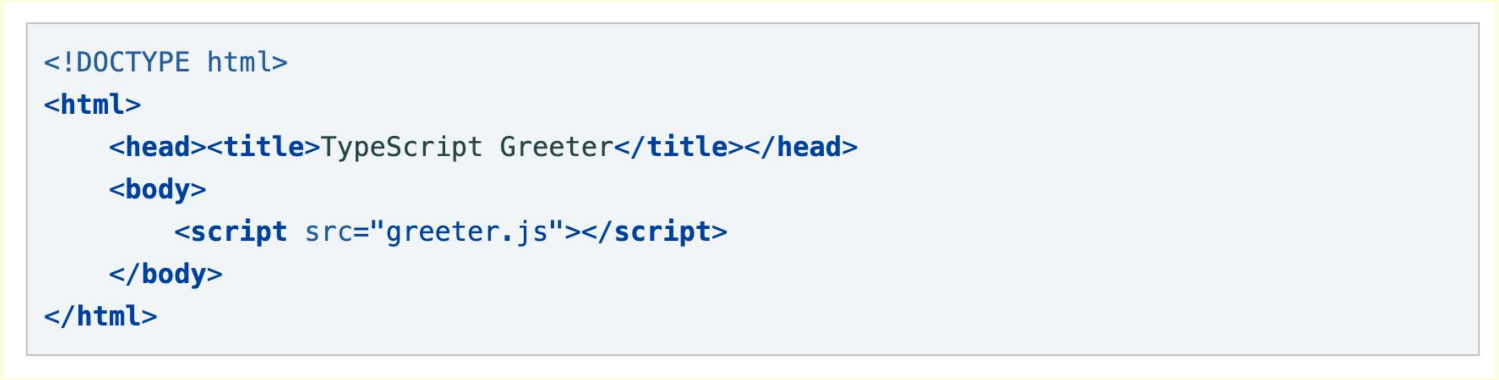
This code is available on Canvas in **greeter.final.ts**



```
class Student {
    fullName: string;
    constructor(public firstName: string, public middleInitial: string, public lastNam
e: string) {
        this.fullName = firstName + " " + middleInitial + " " + lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person: Person) {
    return "Hello, " + person.firstName + " " + person.lastName;
}

let user = new Student("Jane", "M.", "User");

document.body.textContent = greeter(user);
```

TypeScript in 5 Minutes

- Now that we've compiled our final form of **greeter.ts**, we can see the differences in the generated JavaScript
- Classes in TypeScript are just a shorthand for a **prototype-based object** model that is frequently used in JavaScript
- Using the basic HTML in **greeter.html**, we can open our page in a browser, and see our simple TypeScript application!



```
<!DOCTYPE html>
<html>
    <head><title>TypeScript Greeter</title></head>
    <body>
        <script src="greeter.js"></script>
    </body>
</html>
```

Angular Overview

- What is Angular?
  - Angular is a **MVW framework**
    - Model-View-**Whatever works for you**
      - Depending on how you use Angular, it can fall into many categories
        - Model-View-Controller (MVC)
        - Model-View-ViewModel (MVVM)
      - Bottom line: Don’t worry about it’s classification and use it however suits your problem best!
- Is there a difference between AngularJS and Angular?
  - With the release of Angular version 2, Google has differentiated between Angular 1.x and 2+ by naming convention
  - AngularJS refers to Angular version 1.x, and is available on https://angularjs.org/
  - Angular refers to Angular version 2+, and is available on https://angular.io/

Angular Overview

- What’s the difference between Angular and jQuery?
    - jQuery is a JavaScript **library** to abstract DOM manipulation between browsers into a common, manageable way
        - It does not allow for any MVC concepts such as routing, or maintaining a data model
    - Angular is a **framework** that provides similar DOM manipulation capability as jQuery, plus much more!
        - Routing
        - Components
        - Two way data binding
        - Data Models
        - Dependency injection
        - Unit tests
    - AngularJS uses its own API-compatible version of jQuery called jqLite (jQuery lite)
    - Angular does not use jQuery at all, however!

Angular Overview

- https://www.youtube.com/watch?v=VAkio68d51A&ab_channel=AndySterkowitz
- Angular key concepts:
  - **Components**
    - Each component in Angular defines a class that contains the HTML templates, CSS, business logic, and application data for that piece of functionality
    - Components also can allow you to create custom HTML tags specific to your application to encapsulate functionality!
  - **Property Binding**
    - Assigns data **from** a **domain model** to a **binding target**
  - **Event Binding**
    - Binds an HTML form element **to** a **domain model**
  - Using Property and Event Binding together, we can implement **Two-Way Data Binding**
    - The **View** updates automatically when the **Model** changes
    - The **Model** updates automatically when the **View** changes
    - No direct manipulation of DOM elements in the browser to reflect value changes!

Angular Overview

- Angular key concepts:
  - **Modules**
    - A grouping of associated components that can be included in an Angular application by using a uniquely defined name
  - **Form validation**
    - Validation rules can be declared without having to write JavaScript to validate input
  - **Server communication**
    - XMLHttpRequest (XHR) requests can be performed using built-in Angular services
  - **Dependency Injection**
    - Within the Angular application there are named resources (services, pipes, etc…) that can be used just by referencing their name in a constructor for your Component
  - **And so much more**!
    - I could spend an entire class walking through the advanced features of Angular. This course will give you an intermediate understanding of the framework, but I strongly encourage you to explore additional capabilities on your own!

Angular Overview

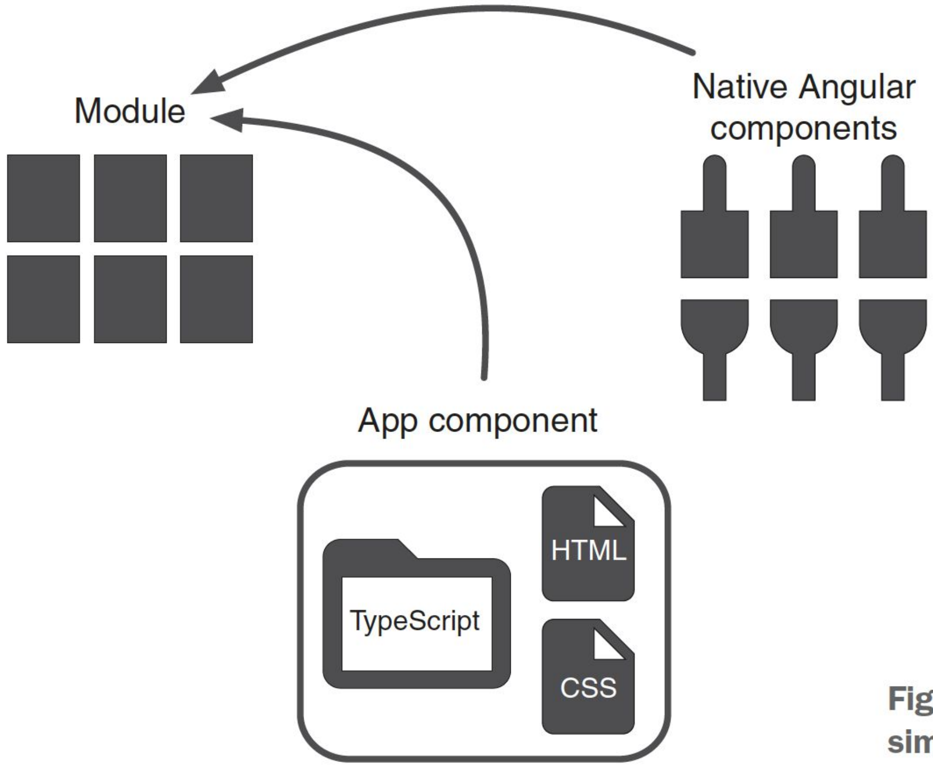
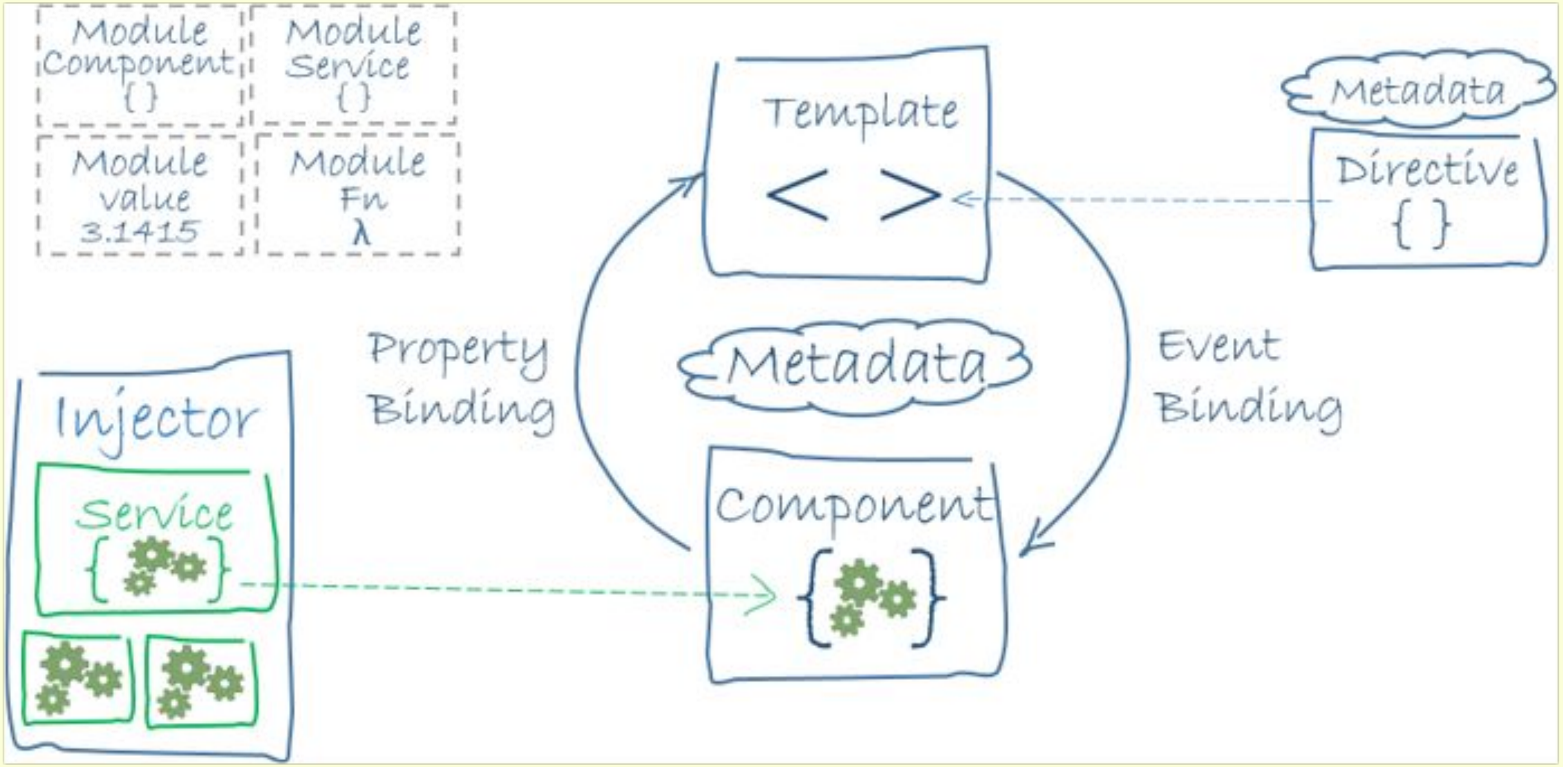


Figure 8.4 How the pieces of the simple Angular app fit together

Angular Overview



https://angular.io/guide/architecture

Client Project Structure

- High level folder structure:
  - **client/app.template.html**
    - The main HTML file for our application
  - **client/app/app.component.ts**
    - Defines the top-level **app** Component referenced in **app.template.html**
    - This component has one purpose: to create the router-outlet for Single-Page-Application (SPA) navigation
  - **client/app/app.constants.ts**
    - This file will contain any shared constants that need to be referenced within Angular Components
  - **client/app/app.module.ts**
    - The main Angular Module for our app
    - All other components will be imported here so that they can be included in the application
    - Imported by app.ts

Client Project Structure

- High level folder structure:
  - **client/app/app.scss**
    - The main Sass file for our application
      - Sass is a CSS extension that provides more powerful directives
      - All .scss files are CSS compatible, so use SASS if you'd like to, or just use vanilla CSS within the .scss file!
  - **client/app/app-*.ts**
    - The main TypeScript files for our application
    - **app-local.ts** is used when running the project locally, and enables **hot module replacement**
    - **app-prod.ts** is used when deploying to Heroku, and uses **precompiled modules** for faster performance
  - **client/app/polyfills.ts**
    - File containing any **polyfills** that needed, depending on which browsers that you have to support
    - Polyfills are JavaScript files that emulate functionality that isn't implemented in certain browsers
  - **client/app/main**
    - The contents of this folder represent the "**main" Component**
      - **client/app/main/main.component.js** defines the **Component**
      - **client/app/main/main.html** defines the **HTML fragment** for the Component
      - **client/app/main/main.module.ts** defines the **Module** and **Routes**
      - **client/app/main/main.scss is the Sass** specific to the **Component**

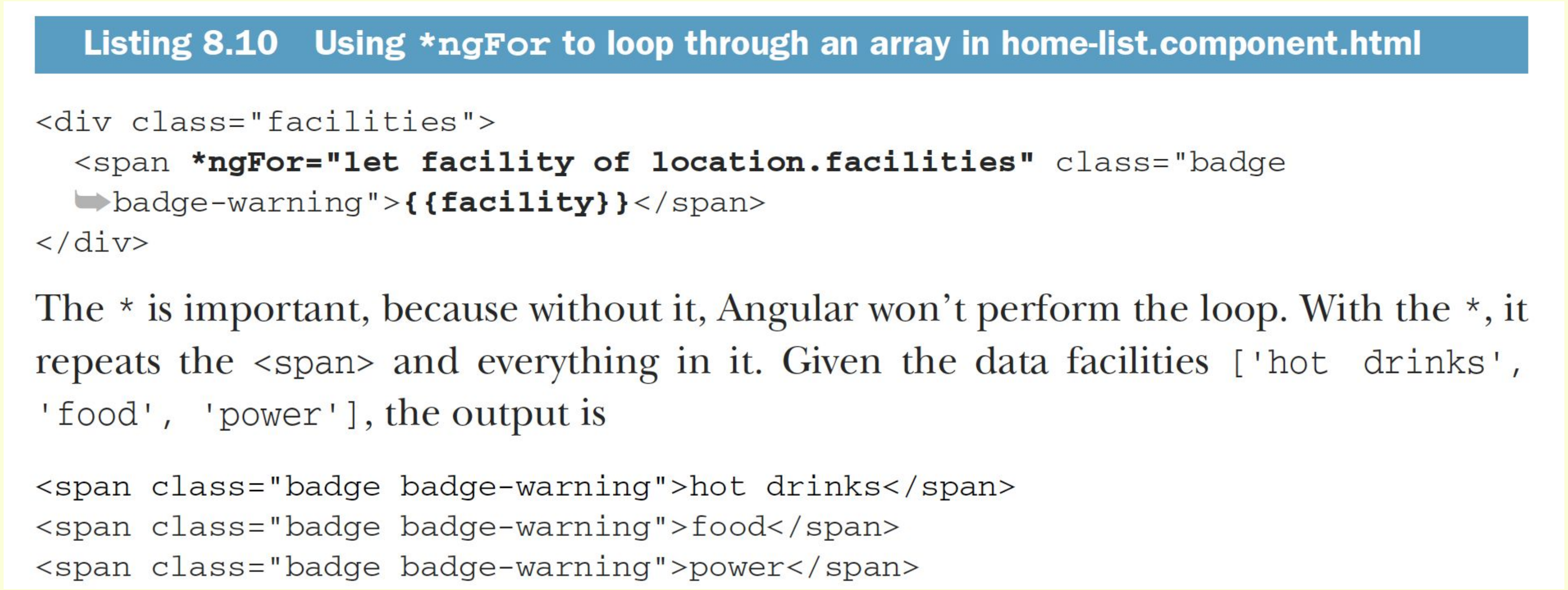
Client Project Structure

- High level folder structure:
    - **client/assets/fonts**
        - font-awesome and bootstrap fonts will be included here automatically as part of the build process, and be available to your app for use
    - **client/assets/images**
        - Any static images you want to use in your application can be placed here
    - **client/components**
        - This folder will house any reusable services, pipes, or utilities you create as part of your project
        - There is an example util service already included in **client/components/util**
            - Want to use this utility? Just **import** it! All of the functions in **util.ts** are **exported** so that other files can **import** them to use!

Live demo - client folder structure and code walkthrough

The Angular ngFor Directive

- ngFor allows you to iterate through a collection of data within your component and generate HTML fragments for each element in the collection!



Listing 8.10 Using `*ngFor` to loop through an array in home-list.component.html

```
<div class="facilities">
  <span *ngFor="let facility of location.facilities" class="badge
  ➥badge-warning">{{facility}}</span>
</div>
```

The * is important, because without it, Angular won't perform the loop. With the *, it repeats the `<span>` and everything in it. Given the data facilities `['hot drinks', 'food', 'power']`, the output is

```
<span class="badge badge-warning">hot drinks</span>
<span class="badge badge-warning">food</span>
<span class="badge badge-warning">power</span>
```

Let's get started with today's lab!

- Today's lab: Introduction to Angular
  - https://canvas.du.edu/courses/135260/assignments/1095020
  - This lab will demonstrate implementing two-way data binding in Angular, and using the ngFor directive!