Introduction to Angular

CST 365 – Web Applications
Michael Ruth, Ph.D.
Associate Professor
Computer Science & I.T.
mruth@roosevelt.edu

Objectives

- Discuss client-side web development using client-side frameworks like Angular
- Describe the development environment for Angular including setup of Node.js
- Discuss Angular concepts including modules, components, views, data binding, and services
- Explain modules, components, services, dependency injection, and decorators
- Discuss Single Page Application (SPA) concepts including directory and application structure

Client-Side Web Development

- Client-side code is written using **HTML**, **CSS**, and **JavaScript**
  - Runs inside the web browser and has little or no access to the underlying OS
  - Since browsers are responsible for actually displaying the web pages, we can't control what ever user will see
    - Browsers provide inconsistent levels of compatibility with client-side features
    - Part of the challenge here is handling those differences gracefully

Client-Side Frameworks

- JavaScript is an essential part of the web, used on 95% of all websites and client-side frameworks are built on JS
- The advent of modern JavaScript frameworks has made it much easier to build highly dynamic, interactive applications.
- A **framework** is a library that offers opinions about how software gets built.
  - These opinions allow for predictability and homogeneity in an application;
  - predictability allows software to scale to an enormous size and still be maintainable;
  - predictability and maintainability are essential for the health and longevity of software.

Angular

- Angular is an application design framework and development platform for creating efficient and sophisticated single-page apps
  - A single page application (SPA) are a fairly common choice for front-end frameworks
    - Once loaded, further interactions involve only requesting/receiving data (the pages are already loaded)

Angular Advantages

- Custom Components:
  - declaratively build components that have functionality along with rendering logic into reasonable size parts
- Data Binding:
  - seamlessly move data from core JS code to the view and react to the view without dealing with glue code
- Dependency Injection:
  - write modular services and have them injected where necessary
- Comprehensive
  - Angular is a full-fledged framework and provides out-of-the-box solutions for server communication, routing and much more

Development Environment

- Most of Angular (2.0+) uses Node.js for a large part of its build environment, so even though we won't use Node.js, we need to install it:
  - Note: you need npm (package manager)
  - Setup Environment:
    - https://angular.io/guide/setup-local
  - Node.js Download:
    - https://nodejs.org/en/
  - Test:
    - Run **npm** from the command line

Development Environment (2)

- Install the Angular CLI
  - You use the Angular CLI to create projects, generate application and library code, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.
  - To install the Angular CLI, open a terminal window and run the following command:
    - **npm install -g @angular/cli**

Development Environment (3)

- Now, we can test by creating a new workspace and initial starter app:
  - Run the CLI command ng new and provide the name my-app, as shown here:
    - **ng new helloworld**
  - The ng new command prompts you for information about features to include in the initial app. Accept the defaults by pressing the Enter or Return key.
  - The Angular CLI installs the necessary Angular npm packages and other dependencies. This can take a few minutes.
  - The CLI creates a new workspace and a simple Welcome app, ready to run.

Development Environment (4)

- Now, we can run it to make sure everything works:
  - The Angular CLI includes a server, so that you can build and serve your app locally.
  - Navigate to the workspace folder, such as my-app.
  - Run the following command:
    - cd my-app
    - ng serve --open
  - The ng serve command launches the server, watches your files, and rebuilds the app as you make changes to those files.
  - The --open (or just -o) option automatically opens your browser to http://localhost:4200/

Angular Concepts

- Angular is a platform and framework for building single-page client applications using HTML and TypeScript.
  - Angular is written in TypeScript.
  - It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.

Angular Concepts (2)

- The basic building blocks of the Angular framework are Angular **components** that are organized into ***NgModules***.
  - NgModules collect related code into functional sets; an Angular app is defined by a set of NgModules.
  - An app always has at least a ***root module*** that enables bootstrapping, and typically has many more ***feature modules***.
    - Components define ***views***, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
    - Components use ***services***, which provide specific functionality not directly related to views.
      - Service providers can be ***injected*** into components as ***dependencies***, making your code modular, reusable, and efficient.

Angular Concepts (3)

- Modules, components and services are classes that use ***decorators***.
  - These decorators mark their type and provide metadata that tells Angular how to use them.
  - The metadata for a component class associates it with a template that defines a view.
    - A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display.
  - The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).
  - An app's components typically define many views, arranged hierarchically.
    - Angular provides the **Router** service to help you define navigation paths among views.
      - The router provides sophisticated in-browser navigational capabilities.

Modules

- An NgModule declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities.
- An NgModule can associate its components with related code, such as services, to form functional units.
- Every Angular app has a root module, conventionally named **AppModule**, which provides the bootstrap mechanism that launches the application.
  - An app typically contains many functional modules.
- Like JavaScript modules, NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules.
  - For example, to use the router service in your app, you import the Router NgModule.
- Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability.
- In addition, this technique lets you take advantage of lazy-loading—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

Components

- Every Angular application has at least one component, the **root** component that connects a component hierarchy with the page document object model (DOM).
- Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.
- The **@Component()** decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Templates, directives, and data binding

- A template combines HTML with Angular markup that can modify HTML elements before they are displayed.
  - Template ***directives*** provide program logic, and ***binding markup*** connects your application data and the DOM.
  - There are two types of data binding:
    - ***Event binding*** lets your app respond to user input in the target environment by updating your application data.
    - ***Property binding*** lets you interpolate values that are computed from your application data into the HTML.
  - Before a view is displayed, Angular evaluates the directives and resolves the binding syntax in the template to modify the HTML elements and the DOM, according to your program data and logic.
  - Angular supports ***two-way data binding***, meaning that changes in the DOM, such as user choices, are also reflected in your program data.
  - Your templates can use ***pipes*** to improve the user experience by transforming values for display.
    - For example, use pipes to display dates and currency values that are appropriate for a user's locale.
    - Angular provides predefined pipes for common transformations, and you can also define your own pipes.

Services and dependency injection

- For data or logic that isn't associated with a specific view, and that you want to share across components, you create a service class.
  - A service class definition is immediately preceded by the **@Injectable()** decorator.
    - The decorator provides the metadata that allows other providers to be injected as dependencies into your class.
  - Dependency injection (DI) lets you keep your component classes lean and efficient.
    - They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.

How We'll Proceed

- Today, we'll create a "HelloWorld" application & review initial assets
  - We'll immediately see the structure of a typical SPA…
    - Note: all starts with **index.html**
  - Finally, although we'll work with typescript, everything works via **transpiled** javascript???
    - **ng serve** command transpiles everything into working javascript files

SPA Structure (src)

- e2e
- node_modules
- src
  - app
    - app.component.css
    - app.component.html
    - app.component.spec.ts
    - **app.component.ts -> root component**
    - **app.module.ts -> main module**
  - assets
  - environments
  - **index.html -> Root HTML**
  - **main.ts -> entry point**
- **JSON files:**
  - There are a few… these are config files for CLI

Root HTML (index.html)

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Helloworld</title>
    <base href="/">
    <meta name="viewport" content="width=device-width,
    initial-scale=1">
    <link rel="icon" type="image/x-icon"
    href="favicon.ico"></head>
  <body>
    <!- root component for our angular application -->
    <app-root></app-root>
  </body>
</html>
```


Entry Point (main.ts)

```
import { enableProdMode } from '@angular/core';

import { platformBrowserDynamic } from '@angular/platform-
browser-dynamic';

import { AppModule } from './app/app.module';

import { environment } from './environments/environment';

if (environment.production) {        enableProdMode();
}

//bootstraps the main module AppModule
platformBrowserDynamic().bootstrapModule(AppModule)
        .catch(err => console.error(err));
```

Main Module (app.module.ts)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({ //marks class defn as module
   declarations: [
       AppComponent  //marks which modules are usable
   ],
   imports: [
       BrowserModule  //importing is how we add functionality
   ],
   providers: [],
   bootstrap: [AppComponent] //entry point for application
})
export class AppModule { }
```


Root Component (app.component.ts)

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

@Component({ //marks defn as component
   //dom selector that gets translated into an instance of
   //this component
   selector: 'app-root',
   //the HTML template backing this component
   templateUrl: './app.component.html',
   //any component specific styling
   styleUrls: ['./app.component.css']
})

export class AppComponent {  //component class
   title = 'helloworld';     //members and functions
}
```


Default HTML Template (app.component.html)

```
<!-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * -->
<!-- * * * * * * * * * * The content below * * * * * * * * * * -->
<!-- * * * * * * * * * * is only a placeholder * * * * * * * * * * -->
<!-- * * * * * * * * * * and can be replaced. * * * * * * * * * * -->
<!-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * -->
<!-- * * * * * * * * * * Delete the template below * * * * * * * * * * -->
<!-- * * * * * * * * to get started with your project! * * * * * * * * -->
<!-- * * * * * * * * * * * * * * * * * * * * * * * * * * * * * -->

… defaults must go …
```

New HTML Template (app.component.html)

- Copy the following into a text editor to replace the data in the file:
  - `<h1>{{title}}</h1>`
  - And re-run serve!

App Overview

- Eventually, our app will have the basic components:
  - Index.html (root component)
    - Shows a list of students (IDs & Names)
  - detail.html
    - Shows an individual student (detail)
  - AddStudent.html
    - Allows us to add a student
  - EditStudent.html (maybe)
    - Allows us to edit a student

Summary

- Discussed client-side web development using client-side frameworks like Angular
- Described the development environment for Angular including setup of Node.js
- Discussed Angular concepts including modules, components, views, data binding, and services
- Explained modules, components, services, dependency injection, and decorators
- Discussed Single Page Application (SPA) concepts including directory and application structure

For More Information

- For this presentation, I primarily used Wikipedia and Spring's main site for most of the details:
  - Angular Guide for Getting Started:
    - https://angular.io/guide/setup-local
  - Angular Architecture:
    - https://angular.io/guide/architecture

Questions?