



# MODULE 5

---

*Angular - A Single Page Application Framework*

# WHAT IS ANGULAR?

---

- Front-end framework
- Use HTML, CSS, JavaScript (additionally TypeScript) to build UI for an app
- Used to build an app as a Single Page Application (SPA)
  - Some alternative frameworks are React (very popular), Vue, Ember and Knockout
- Supports most browsers including IE9+
  - Requires polyfills (provided with framework) for older browsers

# FEATURES OF ANGULAR

---

- Plain HTML used for templating
  - Templating provides a way to specify HTML separate from the data associated with a page. When required templates allow HTML and data to be combined and presented
  - Many other frameworks require separate template engine
- Powerful data binding
  - Data binding is a concept that enables keeping DOM (HTML page) in sync with data associated with JS code
  - Simplifies code - eliminates need for lot of custom code that involves DOM manipulation and event handling

# FEATURES OF ANGULAR

---

- Modular
  - Makes it easy to reuse content and maintain code
  - Angular provides building blocks to structure code
    - Components
    - Services
    - Pipes etc.
  - Angular framework is itself modular
    - Forms module
    - Browser module
    - Router module etc.

# FEATURES OF ANGULAR

---

- Complete SPA support
  - No need for any other library
  - Components for building views
  - Routing
- Back-end integration
  - Service makes it easy to communicate with server APIs

# WHAT IS SPA

---

- SPA = Single Page Applications (e.g. GMail)
- Traditional web apps
  - Multiple HTML pages for multiple views
  - Page refreshes when new view loads
  - Communication with server for HTML, data etc.
  - Page rendered with data on server-side
- SPA
  - App has only a single page!
  - Views load by manipulating DOM
  - Communication is usually for data only
  - Page rendered by JS running in browser

# IMPORTANT ANGULAR APPLICATION BUILDING BLOCKS

---

- **Modules**
  - Organize application code feature-wise
- **Components**
  - The view and view interactions
- **Services**
  - Business logic
  - Server communication
- **Pipes and Filters**
  - Data transformation before rendering
- **Routing, routing guards**
  - Handles page transitions and deep linking
- **Directives - element and attribute directives**
  - Custom HTML components and attributes

# WHAT WE SHALL BUILD - THE APPLICATION OUTLINE

---

- We shall build a simple Product Management Application for a store in this course
- Pages
  - Dashboard (home page)
  - Product listing page
  - Product Details page
- Other components
  - Rating component - reusable
  - App component - to tie pages together
- Service to communicate with backend
- Some others like custom pipes, routing guards etc.

# BEFORE GETTING STARTED

---

- Angular depends on Node.js. Install Node.
  - <https://nodejs.org/en/>
- You need to install Angular CLI to create Angular apps easily
  - <https://cli.angular.io/>
- Make sure to use an IDE with support for Angular
  - <https://code.visualstudio.com/>

# ANGULAR CLI

---

- Scaffold Angular apps
- Generate building block boilerplate code
- Build Angular apps
  - Transpilation, minification, bundling etc. happen during the build step
  - Production builds compile code too
  - Uses tools like System JS (module loader) and Webpack (Module bundler and build tool)
- Comes with a built-in server to serve apps (live reload)

# NODE AND NPM

---

- Node.js (or Node) is a JS runtime usually used to build server-side of web apps
- Node comes with a package manager called npm (node package manager)
- npm is a very popular way to install dependencies for both client-side and server-side apps
  - `npm install <dependency> --save`
  - `package.json` maintains list of saved dependencies
  - Angular relies on npm to install Angular framework files and third-party dependencies
  - We will use npm too. Example: to install Bootstrap.

# JAVASCRIPT LANGUAGE FEATURES

---

- Many browsers support only ES5. Very old ones only ES3.
- ES2015 supported in modern browsers (formerly ES6)
- ES2015 has features like
  - Classes
  - Arrow functions
  - Modules (this is different from modules in Angular)
  - Iterables and for..of loops
  - Backtick strings and string interpolation

# TYPESCRIPT

---

- TypeScript is a superset of ES2015
- TypeScript code is transpired to ES5/ES3 in a build step by Angular CLI - so browser support is not an issue.
- TypeScript provides typed variables
- ES6 features and some ES7 features supported
- Non-JS features like interfaces also supported
- Code is more robust when using TypeScript
- VSCode has good support for TypeScript
  - Intellisense
  - Type-definition files for third-party libraries including Angular

# GETTING STARTED

---

- Generating an Angular project
- Launching it
- Making a small code change
- Understanding the files and older structure of the app
- Understanding different parts of an Angular Module
- Understanding the app build and bootstrap process
- Debugging an application using Chrome Dev Tools

# COMPONENTS

---

*Views of an SPA (pages)*

# COMPONENT

---

- A “page” in Angular is a tree of components
- Parts of a component
  - **Template** - has HTML with data bindings
  - **Encapsulated styles**
    - styles for a component don’t leak to other parts of app
  - **Class with controller logic**
    - Properties - maintain data
    - Methods - handle user interactions
    - Classes become components via the **@Component decorator**

# WORKING WITH COMPONENTS

---

- Replacing the default app component with our component code
  - Provide a simple page title and bind it
  - 2 ways to provide HTML
  - 2 ways to provide styles
- Making sure the component is declared in the app module

# PRODUCT LIST COMPONENT

---

- We build the product list page first with these features
  - Filtering for products
  - Toggle image display by clicking a button
  - Formatting of product code and currency
  - Rating stars

# STEPS TO BUILD THE PRODUCT LIST COMPONENT

---

- Create the component template (no binding)
- Define custom styles
- Create the component class
- Add the new component to the app module
- Add the component in app component (root component)
- Use Angular's interpolation syntax (different from ES2015 interpolation) to bind page title and products
  - Structural directives \*ngIf and \*ngFor
  - Part of Browser Module - so no extra import required
  - **Aside:** Template expressions can have properties, strings, expressions and can even call methods

# STEPS TO BUILD THE PRODUCT LIST COMPONENT

---

- \*ngIf
  - Adds and removes portion of the DOM based on condition
  - We use this to hide products list if list is empty
- \*ngFor
  - **Aside:** Use provided list of products
  - Define an interface for products (best practice - optional step)
  - Use let item of array syntax in \*ngFor

# STEPS TO BUILD THE PRODUCT LIST COMPONENT

---

- Property binding
  - Bind image URL to img src attribute
  - Property binding vs interpolation
    - Interpolation less efficient than property binding
    - However it is required in some cases
  - Setting image size and width using property binding
    - This can be done via styles too

# STEPS TO BUILD THE PRODUCT LIST COMPONENT

---

- Event binding
  - Add a property to maintain image display state
  - Add a method to toggle display state property value
  - Implement \*ngIf on img and event binding on the toggle button
  - Add property binding to toggle image text

# STEPS TO BUILD THE PRODUCT LIST COMPONENT

---

- Two-way data binding
  - Uses ngModel directive
  - Requires FormsModule
- Declare a property to hold filter string
- Use 2-way data binding on filter input
- Interpolate to show filter string as text too
- Add FormsModule to App module imports

# DETOUR: PIPES

---

*Transform data when binding to template*

# PIPES

---

- Pipes transform data when binding to a template
- Can be used in interpolation and property binding
  - Built-in pipes
    - uppercase, lowercase, currency, percentage, json, date etc.
  - Custom pipes created using @Pipe decorator and PipeTransform interface
    - PipeTransform interface has a method transform() that needs to be implemented
    - Make sure to add the Pipe to the App Module

# BACK TO COMPONENTS

---

*(Back from the detour on pipes)*

# USING PIPES TO TRANSFORM DATA IN PRODUCT LIST

---

- Built-in pipes
  - Add a title attribute for image that displays product name in uppercase
  - Display currency in USD with \$ symbol.
- Custom pipes
  - Display product code with hyphens replaced with space

# FILTERING THE PRODUCT LIST

---

- Filters were present in Angular JS (version 1) but removed in Angular (i.e version 2+) for performance reasons
- Now custom logic is required
  - Make property to hold filter string accessible via getter/setter
  - Declare a property to hold filtered set of products and set it whenever filter string property is set

# COMPONENT LIFECYCLE METHODS

---

- ngOnInit()
  - implements OnInit
  - called when component is initialised
  - Used to fetch data required by the component
- ngOnChanges
  - implements OnChanges
  - Called when any of component's "inputs" change
  - We will see inputs later
  - Used to react to change in input
- ngOnDestroy
  - implements OnDestroy
  - Called when component is removed from the DOM

# SUB-COMPONENTS - STAR RATING COMPONENT

---

- Add template, controller files and module declaration
- One way to achieve star rating
  - Add 5 .glyphicon-star
  - Add a container of with a variable width (data-bound) and overflow: hidden set

# PARENT-CHILD COMPONENT COMMUNICATION

---

- Take star rating as `@Input()` variable
- Pass star rating from product list using property binding
- Vary the width of container as per star rating value with `ngOnChanges()`
- Define `@Output` property as an `EventEmitter<Event payload type>` object
- Emit the event (say click on stars) using `emit()` method, passing event details (payload)
- Receive the data in event-bound host attribute
- `$event` is passed as payload to handler in parent

# SERVICES

---

*(Handling App-wide Business Logic and Client-Server Communication)*

# SERVICES

---

- Built-in injector that stores singleton instances of services
- Components requiring a service will have the instance “injected” via the constructor (Dependency Injection - DI)
- Create @Injectable service class
  - Define getProducts() that return the JSON list of products
- Register a provider for the service (factory method for Service or Service itself)
  - When registered in a component, the component and nested components can use it
  - When done via a module, the entire app can use it (done to maintain singleton)
- Have it injected in product list (via its constructor)
- Use it within ngOnInit() to fetch and set filtered list

# DETOUR: OBSERVABLES

---

*Observing streams of events using the Reactive  
Extensions Library - RxJS*

# OBSERVABLES - INTRODUCTION

---

- Reactive Extensions library RxJS helps manage asynchronous streams of events
  - An Angular dependency - no need to install separately
- Feature proposed for ES2016
- Streams can be transformed via operators
  - map
  - do
  - catch

# BACK TO SERVICES

---

*(Back from the detour on observables)*

# HTTP COMMUNICATION WITH THE SERVER

---

- Use either HttpClient/HttpClientModule from @angular/common/http (v4)
- or
- Http from HttpModule/Http (depending upon v4/v2)
- For method 1
  - Inject HttpClient in service constructor
  - Use get() and pass URL to get Observable stream
  - Use RxJS operators for stream transformation
  - Use Observable.throw(message) to further throw errors from error handler (the function passed to catch )
  - Use subscribe() in product list to get a subscription and pass update, error and optionally complete handlers

# ROUTER

---

*(Handling View Navigation & Deep-linking)*

# GETTING STARTED WITH ROUTES – ADD MORE COMPONENTS (VIEWS)

---

- Add AppComponent with <router-outlet></router-outlet>
- Set <base href="/" /> in index page
- Add Home, Product detail and Page not found views with declarations (no selector in decorator)
- Add Navbar to AppComponent with bound routerLink
- Add routes

# ROUTER

---

- Angular maintains a single router instance
- Routes (i.e path-to-component mapping) are added to it using RouterModule.forRoot() and RouterModule.forChild(), passing in an array of routes
  - Some properties of a route are path, component, redirectTo and pathMatch
  - Placeholders given using :<name> (say, :id)
  - Routes matched in order
  - ‘\*\*’ is wildcard route - use for 404 page
- RouterModule registers the Router Service

# ROUTER

---

- Link product in list page to product detail page via an anchor tag
- To get id of product in detail page
  - Inject ActivatedRoute instance
  - Use ActivatedRoute's snapshot.params[] / snapshot.paramMap.get() (differs for v2/v4)
- or
- params.subscribe()
- Link back to product list page - try using Router's navigate() method instead (Router instance to be injected)

# ROUTER GUARDS

---

- Guards are pieces of code that are activated before navigation from/to a view
- Add `@Injectable()` class that implements `CanActivate`
- Depending on return value of the implemented `canActivate()` method, the routing happens/is prevented
- register in `providers[]` of the module
- Add `canActivate[]` property to route and populate with guards

# REFERENCES

---

- Angular site - <https://angular.io/> (Use this primarily)
- Angular tutorial - <https://angular.io/tutorial>
- Browser support - <https://angular.io/guide/browser-support>
- TypeScript site - <http://www.typescriptlang.org/>
- RxJS site - <http://reactivex.io/rxjs/>
- Understanding Observables - <http://rxmarbles.com/>
- Angular JS (Angular version 1) site - <https://angularjs.org/>
  - DO NOT USE THIS
  - Use only if you are working with version 1 of Angular

“

That's all folks!

*-Bugs Bunny & Friends at Looney Tunes*