# Node.js

* It is free and open source JavaScript interpreter/server environment (written in c++) that allows us to run JavaScript outside of a web browser
* Platform independent
* Made from google chromium V8 engine with added event loop and low level I/O API

## CLI Command

* Node -v – Check the current version of your Node.js installed in your system

# NPM

* Node Package Manager
  + It is included with your Node.js
* It consists of three components
  + Website – Discovers packages, set up profiles, and manage access to public or private packages
  + CLI – commands in terminal that allows us to use npm
  + Registry – public database of JavaScript packages comprised of software and metadata
* Similar Maven, it can mange our dependencies and version, this is done through package.json (like Pom.xml)
* It will install our packages in a folder called node\_module in our Angular application

## CLI Command

* Npm -v – Check the current version of your NPM installed in your system
* Npm install -g typescript -Install TypeScript in your system
* Npm install -g @angular/cli -Install Angular CLI in your system

# So why do we need these tools to start Angular?

* Angular is a front-end framework that uses TypeScript to create web application.
* Unfortunately, our browsers only understand JavaScript code, so we need to **transpile** our TypeScript code into JavaScript code.
  + Will transpile TypeScript into JavaScript **NOT** compiling
* Node.js and NPM will perform compilation of the JavaScript and then we can deploy them in production.

# TypeScript

* It is an open-source language developed by Microsoft
* It supports object-oriented programming language
  + It cannot polymorphism that well (method overloading)
* It is a superset of JavaScript
  + All this means it is like JavaScript except it has more methods and uses different way of development that you don’t have to otherwise JavaScript
* It is strictly typed.
  + Now in JavaScript that is not the case.
* **Be sure to study on the different variable types in TypeScript that we can use (like undefined, null, any, void (used for functions), and never (also used for functions))**

## CLI commands

* Tsc -v -Checks your current version of **T**ype**S**cript **C**ompiler
* Tsc [filepath] – will transpile TS file into a JS file
* Tsc [filepath] -w – Will transpile TS file into JS file every time the TS file changes
* Tsc -t es2015 [filepath] -w – Will transpile TS into a specific ES version, default is ES5

# Angular

* It is an open-source framework TypeScript based front-end framework
* AngularJS (JavaScript based) but they switch to TypeScript based on Angular 2
  + Angular JS did not support dynamic loading of the pages or server-side programming language
  + Angular 2 also have better support for mobile devices
  + They also changed their architecture from MVC design to Component and Directive design

## Component Structure

* A grouping of files that is comprised of html, TS, and CSS file
* Optionally you can also add a SPECK file for testing purpose.
* Each structure have their own separate file structure.

## Single Page Application

* Allows us to route pre-loaded components (HTML,TS,and CSS) onto the same page without reloading the window
* Allows us to navigate in the same page without refreshing the whole page
* Advantages:
  + It was fast and responsive. It only updates the required components instead of the entire page.
  + Provides user pleasant experiences especially on mobiles devices.
  + Caching capabilities, since it retrieve everything at once and saves it to your device locally, if you lost connection, you can still navigate through the page and once connection has been reestablished, your local data will synchronize.
* Disadvantages:
  + Doesn’t perform well with SEO (Search Engine Optimization)
    - Angular
  + Angular dynamically fill in the content by using JavaScript that is defined at the bottom of the .html file.
    - Mobile users with slow internet will experience a blank screen for a while since it needs to load a single HUGE JavaScript.
    - Most users will close a browser after experiencing a blank page for more than 3 seconds

## Webpack – bundling

* So it is inefficient to load multiple script tags/files into a HTML page since it reduces the page speed since the browser have to request each script file separately
* This can be solved by Bundling several files together into one file to be downloaded by the browser in one single request.
* Angular will automatically does Webpack-bunding in the background so we don’t have to do it.

## CLI Command

* Npm install -g @angular/cli -Will install the CLI of Angular
* Ng version –Will check your current Angular version.
* Ng new [project name] –Will create an Angular project for you
* Ng serve -Will run your Angular application to a localhost
* Ng generate component [component’s name] – Will auto generate a component for you
* Ng generate pipe [pipe’s name]- Will auto generate a pipe for you
* Ng generate service [service’s name]- Will auto generate a service for you

# Angular Project Files

* The ones in **BOLD** are the most important files you should know about in Angular
* **E2e folder** – Contains source files for a set of end to end testing and testing configuration files
* **Node\_modules** – hold all our modules (“packages”) for the entire workspace
  + You will see that we are only taking a couple of these packages in our current Angular application
* **Src** – contain all the source files which give information about application’s logic, data and assets.
  + **App folder** – contains the app component files (HTML, CSS, TS, and optionally a speck file)
  + **App.module.ts** – defines the root module and helps Angular assemble the application
    - It will dictate all the dependencies (modules) that your Angular application needs to have to run all of your components
  + Assets folder – contains images, audio, video or whatever files you need for you Angular application
  + Environment folder – contains build configuration options for particular target environments
* **Index.html**
  + It is the main HTML page that is served when someone visits your site.
  + Angular will automatically add all the JavaScript (Webpack - bundling) and CSS files when you build you app.
* Main.ts – the main entry point of an application
  + It compiles the application with the JIT compiler and bootstrap the application’s roo model to run the browser
* Polyfills.ts – provides polyfill scripts for older browser support
* Style.css – provides the CSS files that applies to your project
* Test.ts – Main entry point for unit tests used in the application
* .editorconfig – The file contains configuration for code editors
* .gitignore – Git will ignore certain files to not include in repo (like node\_modules)
* Angular.json – holds CLI configuration defaults for all projects in the workspace. It includes configuration options for the build, serve, and test tools.
* Browserlist – used to configure the sharing of target browsers and Node.js versions among various front-end tools
* Karma.conf.js – it contains application-specific Karma configuration
* Package-lock.json – this provides version information for all packages installed into node\_modules by the npm client
* **Package.json –** Used to configure npm packages dependencies that are available to all projects in the workspace
* README.md – introductory documentation
* Tsconfig.app.json – it holds the application-specific TypeScript configuration, including TypeScript and Angular template compiler options.
* Tsconfig.json – holds default TypeScript configuration for projects in the workspace
* Tslint.json – holds default TSLint configuration for projects in the workspace. TSLint is an extensible static analysis tool that checks TypeScript code for readability, maintainability, and functionality errors.

# Angular Decorator

* A decorator is similar to a Java notation.
* It adds a special metadata to whatever you attach to
  + You can do it on a class, method, variable.
* Defined with @expression syntax

## Class Decorator

* It is declared before a class declaration that applied to the constructor of the class

## Method Decorator

* It is declared before a method declaration.
* Used to observe, modify, and replace a method definition

## Property Decorators

* It is declare before a property declaration
* Used to change default definition or modify our object instance
  + Ex: to add new properties or change data
  + Lets say you have a password property (variable) to be longer than 8 characters

## Accessor Decorators

* It is declared before an accessor declaration
* Used to observe, modify, and replace accessor definition
* Note: check angular docs to see specific usage of Accessor decorators

# Steps to create Component

1. Create component structure (html, css, ts) in a folder inside the app folder in your Angular project.
2. In the component’s TS file, create a class with a @component class decorator and fill in the properties (selector, templateUrl, styleUrls)
3. Go to app.module.ts and add the component under the @NgModule decorator under declarations
4. Reference the component using its html tag (should be what you put on the selector property of the @Component decorator) in the app.component.html
5. Ng Serve the Angular application and you should see your first component.

Note: if you do not see anything, edit the app.component.html to only have your component’s tag. Also add a simple header or paragraph inside your component’s html file.

# Data Binding

* It is a process in where we share/project values from the component’s TS into the component’s HTML file or vice verse
* It is a way to communicate or share data between our TS and our HTML inside the component
* There are different ways for data binding but we will just focus on one-way data binding and two-way data binding

## One-way Data binding

* Allows us to bind data from component to the view or view to the component
* It is uni-directional.
  + Meaning if we change a variable defined in the class it will get reflected on the html
* Interpolation
  + Syntax is using “{{}}”
  + It evaluates whatever is inside the expression in the way you can reference variable in the component TS file or do some simple operations.
  + Any changes to the component’s TS file will get reflected on the html
* Property Binding
  + Syntax is using “[]”
  + It binds the attributes of an HTML element to the component’s variable/state
  + Any changes to the TS file will get reflected on the HTML file
* Event Binding
  + Syntax is using “()”
  + It binds the DOM event such as keystrokes, mouse clicks, mouse overs, etc. to a function of your TS file

## Two-way Data binding

* It allows us to bind data from both the component to the view (HTML) and the view to the component.
* It is achieved by combining both property binding and event binding together
* Syntax it uses is “[()]”
* Angular uses ngModel directive to achieve two-way data binding on HTML <forms> element
  + It is a directive that binds the user value (from the forms) in a variable which we can use later in the TS file.
* **NOTE**: you must import FormsModule module in app.module.ts

# Angular Directives

* It allows us to manipulate our DOM or even change its structure
* It acts as a marker on the DOM element that tells Angular to change that DOM element either by appearance, behavior, and layout.
* Most Angular directives will be indicated by “ng”
* You can also create your custom Directives in Angular
  + You must use the @Directive class decorator

## Structure Directive

* This will change/manipulate the structure of the DOM element
* \*ngIf – This will either remove or show a DOM element based on a condition
  + You will not be able to see it in the HTML elements if the Boolean condition is false
* \*ngFor – This will iterate an array in Component’s TS file and then list it (Like a foreach in Java)
* \*ngSwitch (similar to a Java switch)

## Attribute Directive

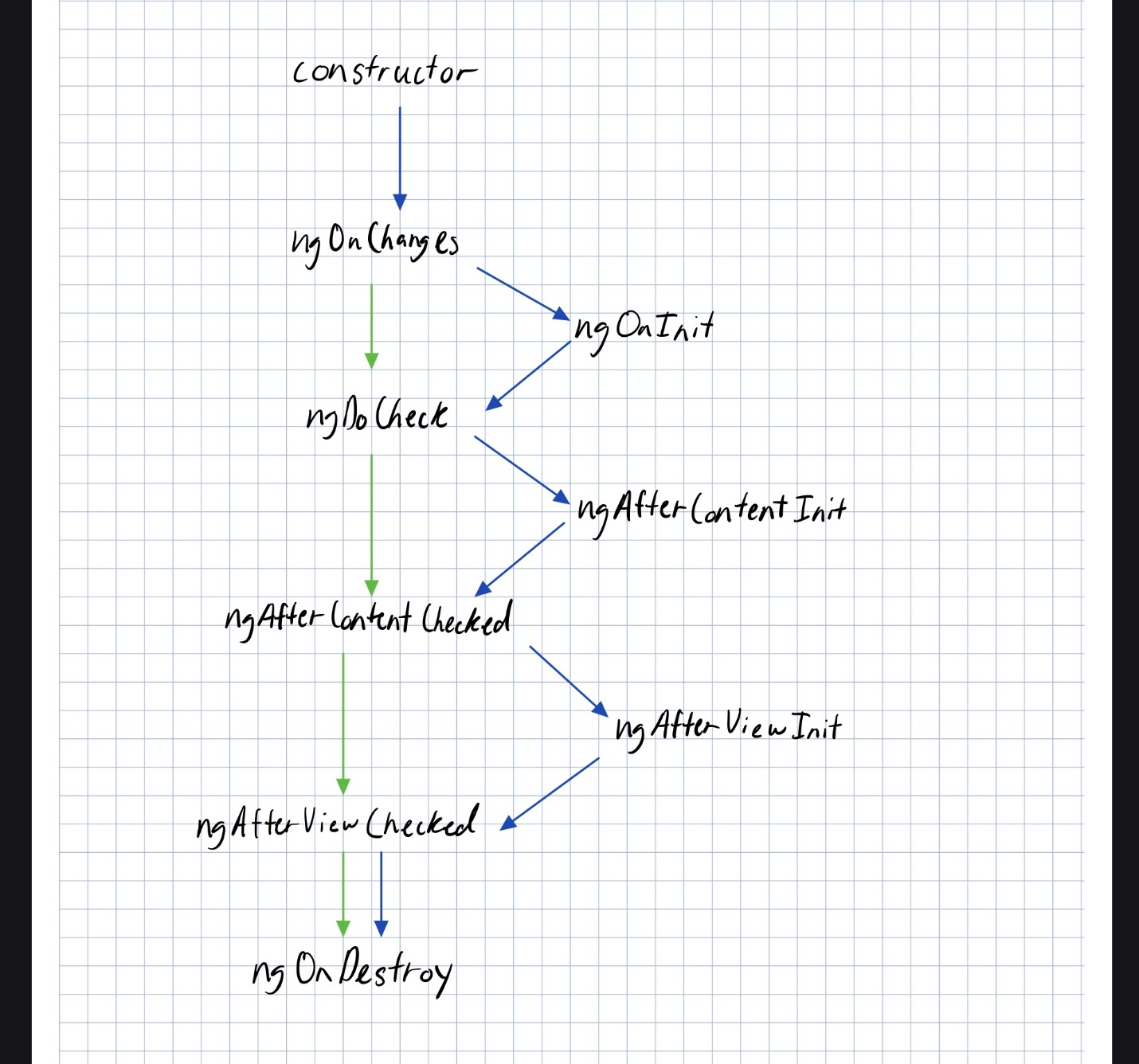
* It is used to change the look and behavior of the DOM elements
* ngClass – Adds multiple CSS **classes** dynamically on the element
* ngStyle – Will check on a Boolean condition and will change the CSS **style** of that element based on that condition

# Component Lifecycle

* Each component is managed by the Angular framework
* Creates the component
* Renders the component
* And then it will create/render any child components
* Process changes to the data bound properties
* Destroy that component

## Lifecycle Hooks

1. Constructor is the first to get executed before any of the other lifecycle hook events
2. **ngOnChanges()** – this is called whenever one or more data bound properties change
3. **ngOnInit()** – This is called only once to initialize the component and set the input properties
4. ngDoCheck() – This is called during all change-detection that Angular can’t detect on its own
5. ngAfterContentInit() – This is called only once after Angular perform any content projection onto the component’s view
6. ngAfterContentChecked() – Invoked after each time Angular checks for content project into the component
7. ngAfterViewInit() – invoked after Angular initialized the component’s view and child’s view
8. ngAfterViewCheckout() – invoked after each time Angular checks for the content projected into the components
9. **ngOnDestroy()** – invoked after Angular destroys the directive or component



# Angular Universe

* It is a pre-rendering solution of Angular
  + By default, Angular will dynamically fill in the content of your webpage so at the very beginning it is just a blank page until everything is fully loaded
* It will render the initial HTML and CSS shown to the user Ahead Of Time either at build time or when the user request the page
* It uses Ahead Of Time Compiler (AOT) to achieve pre-rendering on server-side
* Solves the issue of seeing a blank screen in the initial load of the web application
  + Fun Fact: 53% of users leave the page within 3 seconds of looking a blank page

## Ahead of Time Compiler (AOT)

* Angular needs a compilation process before they can run in a browser. AOT will compile your application during the build process for a faster rendering of the browser for the end-user
* It will convert your HTML and TS file code into efficient JavaScript code during the build phase
  + So before the browser downloads and runs that code.
* It uses decorators to extract metadata to interpret the parts of the application that Angular is suppose to manage.

## Advantages

* Startup performance improvements of our application
* Better user-experience (especially for low internet speed users)
* Better compatibility with SEO
  + Angular Universe creates a static version of your app that is easily searchable by a bot (web crawler such as google)

# Pipes

* They provide a way to transform values before it is displayed.
  + “Transform” is you can change, modify, or reformat the values into what you want it to be
* They are used in conjunction with interpolation so the syntax will be “{{targetValue | pipeType}}”
* Angular pipe api will give you a list of default pipes in Angular that already exists
* You can add multiple pipes to a single value and they will follow the order that you put it on

## Custom Pipes

* You must use the @Pipe decorator to add metadata to the class and indicate to Angular that the class will be a pipe
* Value from the left side of the pipe will be the first parameter of the transform() and the right side of the pipe will be the name of your pipe

## Async Pipes

* It will subscribe to an observable and returns the latest value it has emitted.

# Services

* It is a class that is not dependent on any component
* They are used to share data/logic/function across components
* It can be used to encapsulate external interactions (HTTPClient – a request from an external server)
* Services, if used properly, will be singletons
  + Meaning they will only have one instance exist at all times
* It will be injected into multiple components and those components can then utilize the functions defined in that service class
* You use injected decorator for services

# Dependency Injection (DI)

* It is a design pattern where we declare the dependencies in the constructor of the dependent class. However, in case of Angular Framework, the **Angular Injector** will handle the instantiating the class and its dependencies and providing the dependencies in said dependent class constructor.
* Any class with the component and injector class decorators will be handle by the **Angular Injector**
* In this case, we are giving up the control over a part of out application to the **Angular Injector**

# Routing

* It provides a way for us to navigate from one view to another view in the application.
* It also allows us to create SPA (Single Page Application)
* We must define the paths that when called upon will quickly swap in that component assigned to the path without refreshing the page.
* Using “\*\*” in the path indicates a wild card in that any other path will default go to another one (good practice to avoid blank pages).
* Note: You must add RouterModule in app.module.ts

## Lazy Loading

* By default, ngModules are **eagerly** loaded meaning as soon as you load your application, it will get all the ngModules
* This can heavily slow the loading process of your application especially with more complex and large application that have multiple routes
* Lazy loading allows you to only load the ngModules you currently need and will load more ngModules as you need them

## Route Guards

* It allows us to restrict users to not access certain routes or paths.
* Ex: Common use of route guards is if the user is logged in or not and if you are logged in you are able to go to different routes
* Types of Route Guards:
  + canActivate – will check if the user can visit a route
  + canActivateChild – Will check if the user can visit the route’s children
  + CanLoad – will decide if a route can be loaded (For lazy loading)
  + CanDeactivate – checks if the user can exit that route

# Angular HttpClient

* It is an API for Angular applicationg that handles HTTP requests in order to download/upload data and access other back-end services.
* All HttpClient methods will return an **Observable** of something

## RXJS

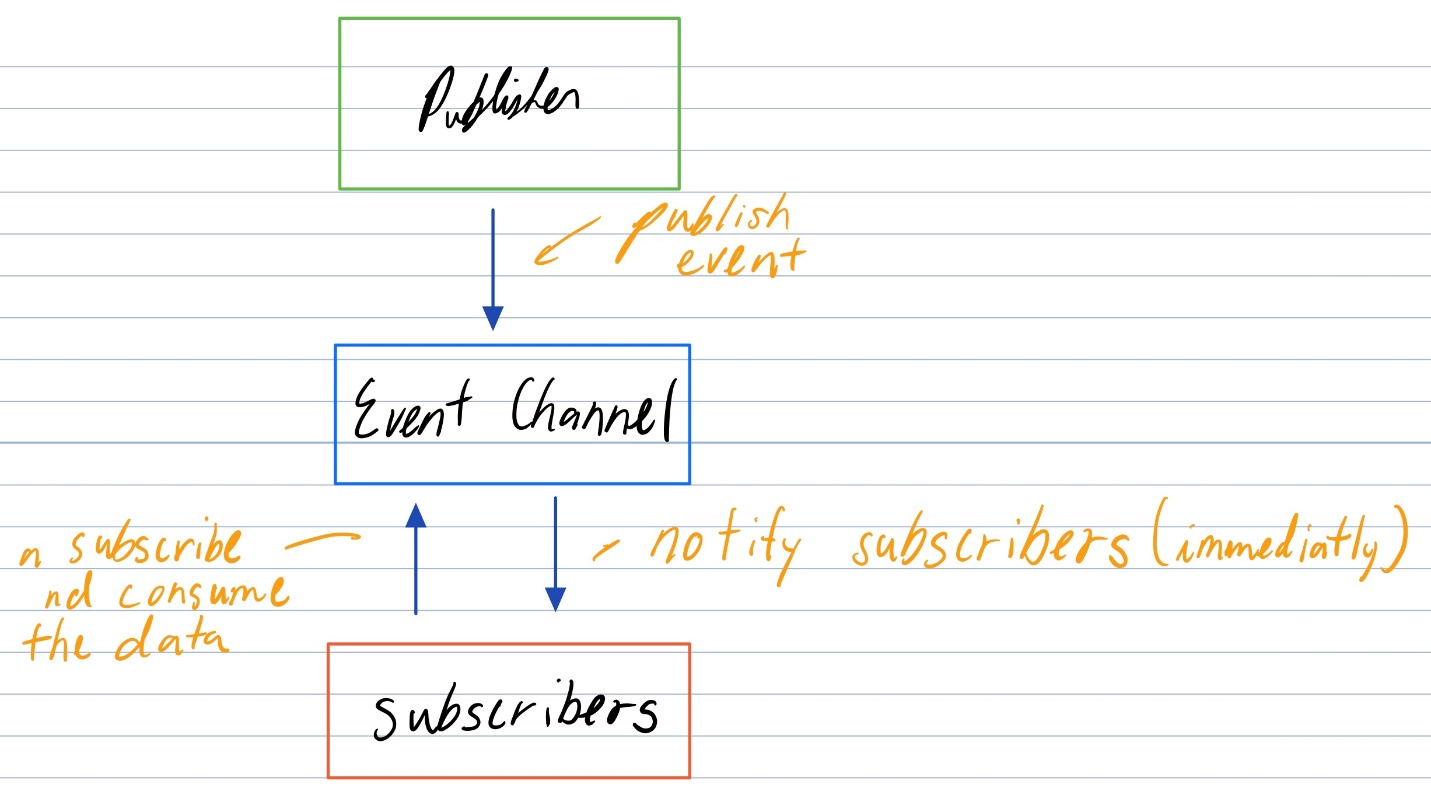
* Another framework for reactive programming using observables that makes it easier to write asynchronous code.
* **R**eactive E**x**tension **J**ava**S**cript

## Observable

* It is similar to a promise in JS
  + Promise is a place holder for a future value that handles a signle event when an asynchronous operation completes or fails
  + Not cancellable and it only has 1 activation at a time
* It follows a publisher and subscriber model
  + It defines a function for publishing values, but it will not execute until a consumer (subscriber) subscriber to it
* It can have 0 or 1 or many activations as long as the subscriber subscribes to that event channel
* It has an array that stores data that arrives asynchronous over time
* It can help manage our data such as data coming from some back-end server

## Publisher/Subscriber Design Pattern

* It describes the flow of message between application, devices, or services
* A message is published by **Publisher** to a **Channel**, any **Subscribers**monitoring that **Channel** will consume that message/value



* Usually implement in asynchronous way.

## Subjects

* Think of it as an array of Observables
* One distinction is that it can also emit data while also subscribing to an event channel
  + It is like both a subscriber and a publisher.
* They can support multiple subscriptions.

### 3 types of Subjects

* Behavior Subjects
  + It will hold the last value it receives, and any new subscribers will receive that last value.
  + Think of it as subscribing to a magazine and the moment you subscribe to that magazine you will receive the previous issue of that magazine.
* Replay Subjects
  + It is like behavior subjects BUT it can hold many past values and you decide how many past values it should keep.
  + So instead of just getting the last magazine you subscribe to let’s say the magazine company gives you the last 5 magazine that was issued before.
* Async Subjects
  + You will only get the last value when the sequence is completed
  + You will complete the sequence once you use the .complete() method to signify the sequence is completed
    - After that method is declared, all subscribers will the receive the last value Angular Form.

## HttpClient Error Handling

* In case of a back-end server failing to connect to your front-end
* It will be poor user experience if they try to use our back-end server and it loads a blank screen instead.
* Another example would be doing the wrong user credentials
* Note: you must import throwError and catchError

## Template Driven Forms

* All validation, form elements are all defined in the template **HTML file**
* You use two-way data binding to bind data
* Directives:
  + ngForm – assigns a reference variable to the entire form
  + ngModel – allows us to do two-data binding between html and ts file
* Validation:
  + Angular has 3 different states of a form-control
    - If the Form has been visited
      * False – ng-untouched
      * True – ng-touched
    - Form control’s value has been changed
      * False – ng-pristine
      * True – ng-dirty
    - Form control’s value is valid (Think of password)
      * False – ng-invalid
      * True – ng-valid

## Reactive Forms

* All the form elements, user interactions, and validations are handled in the **component’s TS file / class**
* Variable Type:
  + formGroup
  + formControl
* It is easier to do data binding
* Easier to create flexible and complex requirements of the forms
* Note: You must import ReactiveFormsModule in app.module.ts
* Reactive form also have the 3 different states of form-control

# Angular Animation

* Provides an illusion of motion by changing our HTML element style over time.
* Using Angular’s built in Animation Module
  + We will be able to move, grow, shrink, or change the color of out HTML elements
* Note: You must import BrowserAnimationModule from @angular/platfrom-browser/animations

## Polyfills

* It allows you to use modern JavaScript features in older browsers (Internet Explorer)

## Angular Animation States

* Void state (void)
  + The element is not yet part of the DOM
  + Either the element is create but not yet placed in the DOM or the element is removed from the DOM (\*ngIf)
* Default state (\*)
  + The state at which the element appears at the DOM for the first time
* Custom states (You decide what syntax to reference it)
  + Ex: Think of a drop down list, it has a collapsed state and an expanded state

# Testing in Angular with Jasmine and Karma

## Behavior Driven Development (BDD)

* Development is that it occurs based of collaborations between non-technical members and technical development team.
  + Meaning a non-technical person should be able to understand what you are testing in your application
* Conversations between business analyst and Client/Product Owner will determine how the coding will occur (user stories)
  + User story is basically just the natural language that describes the feature of a software system
  + Ex: I want my web browser to display my recent purchases
* Conversations -> Test cases -> Development
  + We are doing testing first then development (TDD – Test driven development)
* Spoiler alert: Jasmine reaaaally sucks at making the unit test cases to sound like a natural language

## Jasmine

* It is a JavaScript unit testing framework that “supports” BDD
* Jasmine attempts to describe the test cases in a non-technical format
* Features to know about:
  + **Test Suite** (a collection of test cases) we use the describe() function keyword and it will be collection of individual **Test Specs** (basically similar to a unit test)
  + To create a **Test Spec** we use it() function
    - The it function will contain one or more test expectations (kinda like assertions in JUnit)
  + To create an Expectation we use expect() function
    - It is used with conjunction with matcher function that does some sort of Boolean comparison with the expected value
  + Spying is one way we can mock dependencies inside of Jasmine
    - Basically it is the same as giving no implementation detail to some function

## Setup and Teardown

* beforeAll
  + Called once before any of the test specs or the beginning of the test suite
* afterAll
  + Called once after all the test specs or the end of the test suite
* beforeEach
  + Called once before each test spec
* afterEach
  + Called one after each test spec

## Karma

* Before Karma, we had to manually run Jasmine tests by refreshing the browser in different browsers (such Google chrome, edge, firefox, etc.) every time we edit our code
* It is a tool developed to spawn browsers and run Jasmine tests all from the command line
* It will also display its results on the command line

## CLI Command

* Ng test -Will run Karma to do unit testing using Jasmine testing framework