Angular

Getting MEAN 2 – A Practical Workshop

Sharif Malik | 2017

TABLE OF CONTENT

INTRODUCTION

What is Angular?

- Angular is a platform that makes it easy to build applications with the web.
- Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges.
- Angular empowers developers to build applications that live on the web, mobile, or the desktop.
- Angular is a framework for building client applications in HTML and either JavaScript or a language like TypeScript that compiles to JavaScript.
- The framework consists of several libraries, some of them core and some optional.
- You write Angular applications by composing HTML *templates* with Angularized markup, writing *component* classes to manage those templates, adding application logic in *services*, and boxing components and services in *modules*

Why Angular?

- There are many front-end JavaScript frameworks to choose from today, each with its own set of trade-offs. Many people were happy with the functionality that Angular 1.x afforded them.
- Angular 2 improved on that functionality and made it faster, more scalable and more modern. Organizations that found value in Angular 1.x will find more value in Angular 2.

Angular 1 vs Angular 2 vs Angular 4

- **Angular 1** = Angular JS: very popular JS framework, released couple of years ago which allows to create reactive Single page applications (SPAs).

- **Angular 2** = Angular: Complete rewrite of Angular 1 and the future of Angular and was released in Sep, 2016

- **Angular 4** = Simply an update to "Angular 2"

(Angular 3 was skipped due to version number conflicts)

Single Page Application

Explanation

- ❖ In a traditional Web app, every time the app calls the server, the server renders a new HTML page. This triggers a page refresh in the browser. If you've ever written a Web Forms application or PHP application, this page lifecycle should look familiar.
- ❖ In an SPA, after the first page loads, all interaction with the server happens through AJAX calls. These AJAX calls return data—not mark-up—usually in JSON format. The app uses the JSON data to update the page dynamically, without reloading the page.

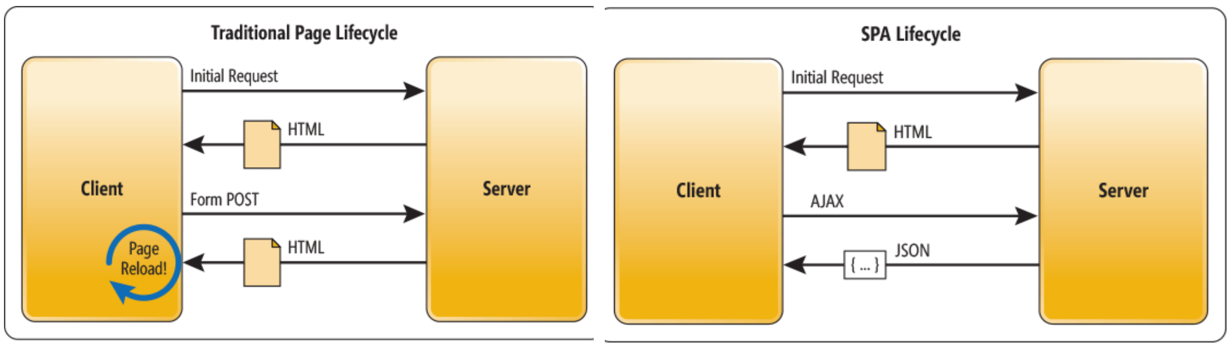
What is Single Page Application?

- ❖ Single-Page Applications (SPAs) are Web apps that load a single HTML page and dynamically update that page as the user interacts with the app.
- ❖ SPAs use AJAX and HTML5 to create fluid and responsive Web apps, without constant page reloads.

Popular JavaScript Frameworks for Building SPAs

- ❖ The more interactivity that happens on the client-side, the more JavaScript code is needed to make those interactive pieces function well. And the more code is written, the more important it is to have a clean and well-architected codebase. And this is exactly the problem JavaScript frameworks help solve each with its own approach.

- ❖ There are a lot of open source JavaScript frameworks that help with building SPAs, such as:
  Angular, React, Ember, Vue.js, Backbone etc.
  Example: LinkedIn, Gmail web-application etc.



Angular CLI

What is Angular CLI?

The Angular CLI is a command line interface tool that can create a project, add files, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.

Usage:
ng help

Generating new project
ng new PROJECT-NAME

Serving project via a development server
cd PROJECT-NAME
ng serve

Navigate to **http://localhost:4200/**. The app will automatically reload if you change any of the source files.

Generating components, Directives, Pipes and services

You can use the ng generate (or just ng g) command to generate Angular components:

Scaffold	**Usage**
Component	ng g component my-new-component
Directive	ng g directive my-new-directive
Pipe	ng g pipe my-new-pipe
Service	ng g service my-new-service
Class	ng g class my-new-class
Guard	ng g guard my-new-guard
Interface	ng g interface my-new-interface
Enum	ng g enum my-new-enum
Module	ng g module my-module

- ❖ Angular-cli will add reference to components, directives and pipes automatically in the **app.module.ts** file.
- ❖ If you need to add this references to another custom module, follow this steps:
  ng g module new-module

Quick Start: Hello World App

- ❖ Good tools make application development quicker and easier to maintain than if you did everything by hand.
- ❖ The **Angular CLI** is a command line interface tool that can create a project, add files, and perform a variety of ongoing development tasks such as testing, bundling, and deployment.
- ❖ The goal in this guide is to build and run a simple Angular application in TypeScript, using the Angular CLI while adhering to the Style Guide recommendations that benefit every Angular project.

❖ Step 1. Set up the Development Environment

You need to set up your development environment before you can do anything.

Install Node.js® and npm if they are not already on your machine.
Then install the Angular CLI globally.

```
npm install –g @angular/cli
```

❖ Step 2. Create a new project

Open a terminal window. Generate a new project and skeleton application by running the following commands:

```
ng new my-app
```

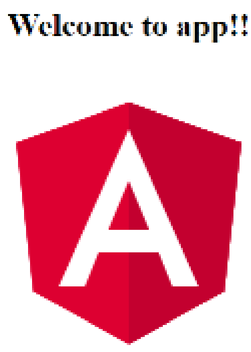
Note: Patience please! It takes time to set up a new project, most of it spent installing npm packages.

Step 3: Serve the application

Go to the project directory and launch the server.

```
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.
Using the --open (or just -o) option will automatically open your browser on *http://localhost:4200/*.
Your app greets you with a message:



Welcome to app!!

Step 4: Edit your first Angular component

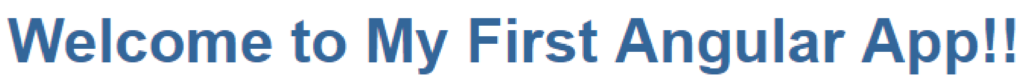
The CLI created the first Angular component for you. This is the root component and it is named app-root. You can find it in ./src/app/app.component.ts.

Open the component file and change the title property from Welcome to app!! to Welcome to My First Angular App!!:

```
export class AppComponent {
title = 'My First Angular App';
}
```

The browser reloads automatically with the revised title. That's nice, but it could look better.
Open src/app/app.component.css and give the component some style.

```
h1 { color: #369;
font-family: Arial, Helvetica, sans-serif;
font-size: 250%;
}
```



Welcome to My First Angular App!!

You can download source code from here (https://github.com/virtualSharif/gettingMEAN2/tree/master/angular/examples/example1)

The src folder

Your app lives in the src folder. All Angular components, templates, styles, images, and anything else your app needs go here. Any files outside of this folder are meant to support building your app.

File	Purpose
app/app.component.{ts,html,css,spec.ts}	Defines the AppComponent along with an HTML template, CSS stylesheet, and a unit test. It is the root component of what will become a tree of nested components as the application evolves
app/app.module.ts	Defines AppModule, the root module that tells Angular how to assemble the application. Right now it declares only the AppComponent. Soon there will be more components to declare.
assets/*	A folder where you can put images and anything else to be copied wholesale when you build your application.
environments/*	This folder contains one file for each of your destination environments, each exporting simple configuration variables to use in your application. The files are replaced on-the-fly when you build your app. You might use a different API endpoint for development than you do for production or maybe different analytics tokens. You might even use some mock services. Either way, the CLI has you covered.
favicon.ico	Every site wants to look good on the bookmark bar. Get started with your very own Angular icon.
index.html	The main HTML page that is served when someone visits your site. Most of the time you'll never need to edit it. The CLI automatically adds all js and css files when building your app so you never need to add any <script> or <link> tags here manually.
main.ts	The main entry point for your app. Compiles the application with the JIT compiler and bootstraps the application's root module (AppModule) to run in the browser. You can also use the AOT compiler without changing any code by passing in --aot to ng build or ng serve.
polyfills.ts	Different browsers have different levels of support of the web standards. Polyfills help normalize those differences. You should be pretty safe with core-

	js and zone.js, but be sure to check out the Browser Support guide for more information.
styles.css	Your global styles go here. Most of the time you'll want to have local styles in your components for easier maintenance, but styles that affect all of your app need to be in a central place.
test.ts	This is the main entry point for your unit tests. It has some custom configuration that might be unfamiliar, but it's not something you'll need to edit.
tsconfig.{app|spec}.json	TypeScript compiler configuration for the Angular app (tsconfig.app.json) and for the unit tests (tsconfig.spec.json).

The root folder

The src/ folder is just one of the items inside the project's root folder. Other files help you build, test, maintain, document, and deploy the app. These files go in the root folder next to src/.

File	Purpose
e2e/	Inside e2e/ live the end-to-end tests. They shouldn't be inside src/ because e2e tests are really a separate app that just so happens to test your main app. That's also why they have their own tsconfig.e2e.json
node_modules/	Node.js creates this folder and puts all third party modules listed in package.json inside of it.
.angular-cli.json	Configuration for Angular CLI. In this file you can set several defaults and also configure what files are included when your project is built. Check out the official documentation if you want to know more.
.editorconfig	Simple configuration for your editor to make sure everyone that uses your project has the same basic configuration. Most editors support an .editorconfig file. See http://editorconfig.org for more information.
.gitignore	Git configuration to make sure autogenerated files are not commited to source control.
karma.conf.js	Unit test configuration for the Karma test runner, used when running ng test.

package.json	npm configuration listing the third party packages your project uses. You can also add your own custom scripts here.
protractor.conf.js	End-to-end test configuration for Protractor, used when running ng e2e.
README.md	Basic documentation for your project, pre-filled with CLI command information. Make sure to enhance it with project documentation so that anyone checking out the repo can build your app!
tsconfig.json	TypeScript compiler configuration for your IDE to pick up and give you helpful tooling.
tslint.json	Linting configuration for TSLint together with Codelyzer, used when running ng lint. Linting helps keep your code style consistent.

Modules

- ❖ Angular apps are modular and Angular has its own modularity system called NgModules.
- ❖ Every Angular app has at least one NgModule class, the root module, conventionally named AppModule.
- ❖ While the root module may be the only module in a small application, most apps have many more feature modules, each a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities.
- ❖ An NgModule, whether a root or feature, is a class with an @NgModule decorator.

Note: Decorators are functions that modify JavaScript classes. Angular has many decorators that attach metadata to classes so that it knows what those classes mean and how they should work. Learn more about decorators on the web(https://medium.com/google-developers/exploring-es7-decorators-76ecb65fb841)

- ❖ The most important properties are:

    declarations - the view classes that belong to this module. Angular has three kinds of view classes: components, directives, and pipes.
    exports - the subset of declarations that should be visible and usable in the component templates of other modules.
    imports - other modules whose exported classes are needed by component templates declared in this module.

providers - creators of services that this module contributes to the global collection of services; they become accessible in all parts of the app.
bootstrap - the main application view, called the root component, that hosts all other app views. Only the root module should set this bootstrap property.

❖ Example:

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

@NgModule(
{
imports: [ BrowserModule ],
providers: [ Logger ],
declarations: [ AppComponent ],
exports: [ AppComponent ],
bootstrap: [ AppComponent ] })
export class AppModule { }
```

❖ **Note**: The export of AppComponent is just to show how to export; it isn't necessary in this example. A root module has no reason to export anything because other components don't need to import the root module.

Launch an application by bootstrapping its root module. During development, you're likely to bootstrap the AppModule in a main.ts file like this one.

Example:

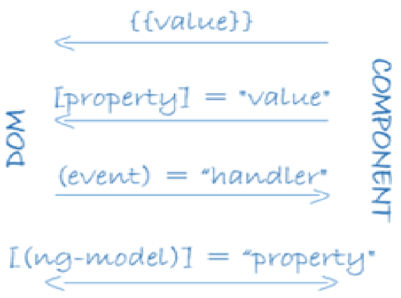
```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

Components

- ❖ A component controls a patch of screen called a view.
- ❖ Angular creates, updates, and destroys components as the user moves through the application.
- ❖ Your app can take action at each moment in this lifecycle through optional lifecycle hooks (https://angular.io/guide/lifecycle-hooks), like ngOnInit() declared above.

Data binding



- ❖ Without a framework, you would be responsible for pushing data values into the HTML controls and turning user responses into actions and value updates.
- ❖ Writing such push/pull logic by hand is tedious, error-prone, and a nightmare to read as any experienced jQuery programmer can attest.
- ❖ Angular supports data binding, a mechanism for coordinating parts of a template with parts of a component.
- ❖ As the diagram shows, there are four forms of data binding syntax. Each form has a direction — to the DOM, from the DOM, or in both directions.

Two-way data binding

- ❖ Two-way data binding is an important fourth form that combines property and event binding in a single notation, using the ngModel directive.
- ❖ In two-way binding, a data property value flows to the input box from the component as with property binding. The user's changes also flow back to the component, resetting the property to the latest value, as with event binding.
- ❖ Angular processes all data bindings once per JavaScript event cycle, from the root of the application component tree through all child components.
- ❖ Data binding plays an important role in communication between a template and its component.
- ❖ Data binding is also important for communication between parent and child components.
- ❖ Example:
  You can download the source code from here (https://github.com/virtualSharif/gettingMEAN2/tree/master/angular/examples/example2)

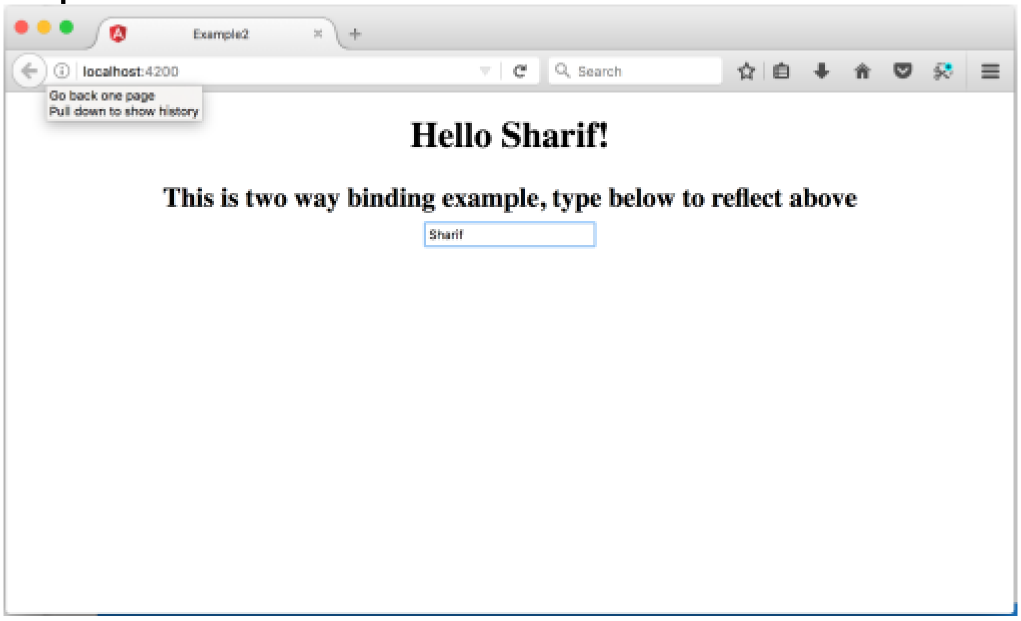
Step1: Install all the dependencies

npm install

Step 2: Run the application
ng serve

Step 3: Open the browser,
go to http://localhost:4200 to the see the output of two-way binding

Output:



Hello Sharif!

This is two way binding example, type below to reflect above

Sharif

Services

- ❖ Service is a broad category encompassing any value, function, or feature that your application needs.
- ❖ Almost anything can be a service. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.
- ❖ Examples include:
  logging service, data service, message bus, tax calculator, application configuration.
- ❖ Services are everywhere: Component classes should be lean. They don't fetch data from the server, validate user input, or log directly to the console. They delegate such tasks to services.
- ❖ A component's job is to enable the user experience and nothing more. It mediates between the view (rendered by the template) and the application logic (which often includes some notion of a model).

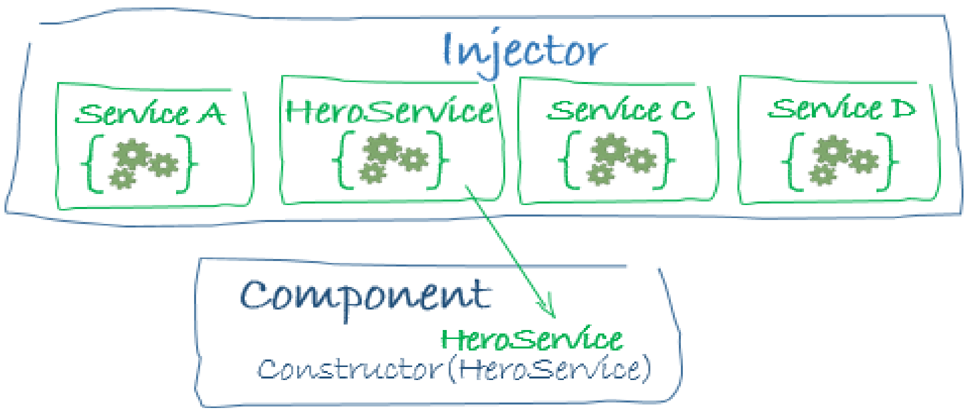
- A good component presents properties and methods for data binding. It delegates everything nontrivial to services.
- Angular doesn't enforce these principles. It won't complain if you write a "kitchen sink" component with 3000 lines.
- Angular does help you follow these principles by making it easy to factor your application logic into services and make those services available to components through dependency injection.

Dependency injection

- Dependency injection is a way to supply a new instance of a class with the fully-formed dependencies it requires. Most dependencies are services.
- Angular uses dependency injection to provide new components with the services they need.
- Angular can tell which services a component needs by looking at the types of its constructor parameters.
- For example, the constructor of your UserListComponent needs a UserService.

```
constructor(private userService: UserService) { }
```

- When Angular creates a component, it first asks an injector for the services that the component requires.
- An injector maintains a container of service instances that it has previously created. If a requested service instance is not in the container, the injector makes one and adds it to the container before returning the service to Angular.
- When all requested services have been resolved and returned, Angular can call the component's constructor with those services as arguments. This process is called as dependency injection.
- The process of HeroService injection looks a bit like this:



Providers

- A provider is something that can create or return a service, typically the service class itself.
- You can register providers in modules or in components.
- In general, add providers to the root module so that the same instance of a service is available everywhere.
- Example:

```
providers: [
        UserService,
        HeroService,
        Logger ],
```

Alternatively, register at a component level in the providers property of the @Component metadata:

```
@Component({
selector: user-list',
templateUrl: './user-list.component.html',
providers: [ UserService ]
})
```

- **Note**: Registering at a component level means you get a new instance of the service with each new instance of that component.

- **Points to remember about dependency injection:**
  - Dependency injection is wired into the Angular framework and used everywhere.
  - The *injector* is the main mechanism.
  - An injector maintains a *container* of service instances that it created.
  - An injector can create a new service instance from a *provider*.
  - A *provider* is a recipe for creating a service.

Forms

- Support complex data entry scenarios with HTML-based validation and dirty checking.
- Forms are the mainstay of business applications. You use forms to log in, submit a help request, place an order, book a flight, schedule a meeting, and perform countless other data-entry tasks.
- In developing a form, it's important to create a data-entry experience that guides the user efficiently and effectively through the workflow.

❖ Developing forms requires design skills (which are out of scope for this page), as well as framework support for two-way data binding, change tracking, validation, and error handling, which you'll learn about on this page.

❖ **Types of forms**
  - Template Driven Forms (https://angular.io/guide/forms)
  - Reactive Forms (https://angular.io/guide/reactive-forms)

Router

❖ Angular Router enables navigation from one view to the next as users perform application tasks.

❖ The browser is a familiar model of application navigation:
  - Enter a URL in the address bar and the browser navigates to a corresponding page.
  - Click links on the page and the browser navigates to a new page.
  - Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

❖ The Angular Router ("the router") borrows from this model. It can interpret a browser URL as an instruction to navigate to a client-generated view.

❖ It can pass optional parameters along to the supporting view component that help it decide what specific content to present.

❖ You can bind the router to links on a page and it will navigate to the appropriate application view when the user clicks a link.

❖ You can navigate imperatively when the user clicks a button, selects from a drop box, or in response to some other stimulus from any source. And the router logs activity in the browser's history journal so the back and forward buttons work as well. A routed Angular application has one singleton instance of the Router service. When the browser's URL changes, that router looks for a corresponding Route from which it can determine the component to display.

Lifecycle hooks

Tap into key moments in the lifetime of a component, from its creation to its destruction, by implementing the lifecycle hook interfaces.

HTTP

Communicate with a server to get data, save data, and invoke server-side actions with an HTTP client.

Pipes

Use pipes in your templates to improve the user experience by transforming values for display.
Consider this currency pipe expression:

```
price | currency:'USD':true
```

It displays a price of 42.33 as $42.33.

Testing

Run unit tests on your application parts as they interact with the Angular framework using the Angular Testing Platform.

Cheat sheet

Please go to this website for cheat sheet https://angular.io/guide/cheatsheet

Applications

Example3-forms:

In this example: we are trying to create a form which can be used to save new user.

You can download the source code from here (https://github.com/virtualSharif/gettingMEAN2/tree/master/angular/examples/example3-forms)

Step1: Install all the dependencies

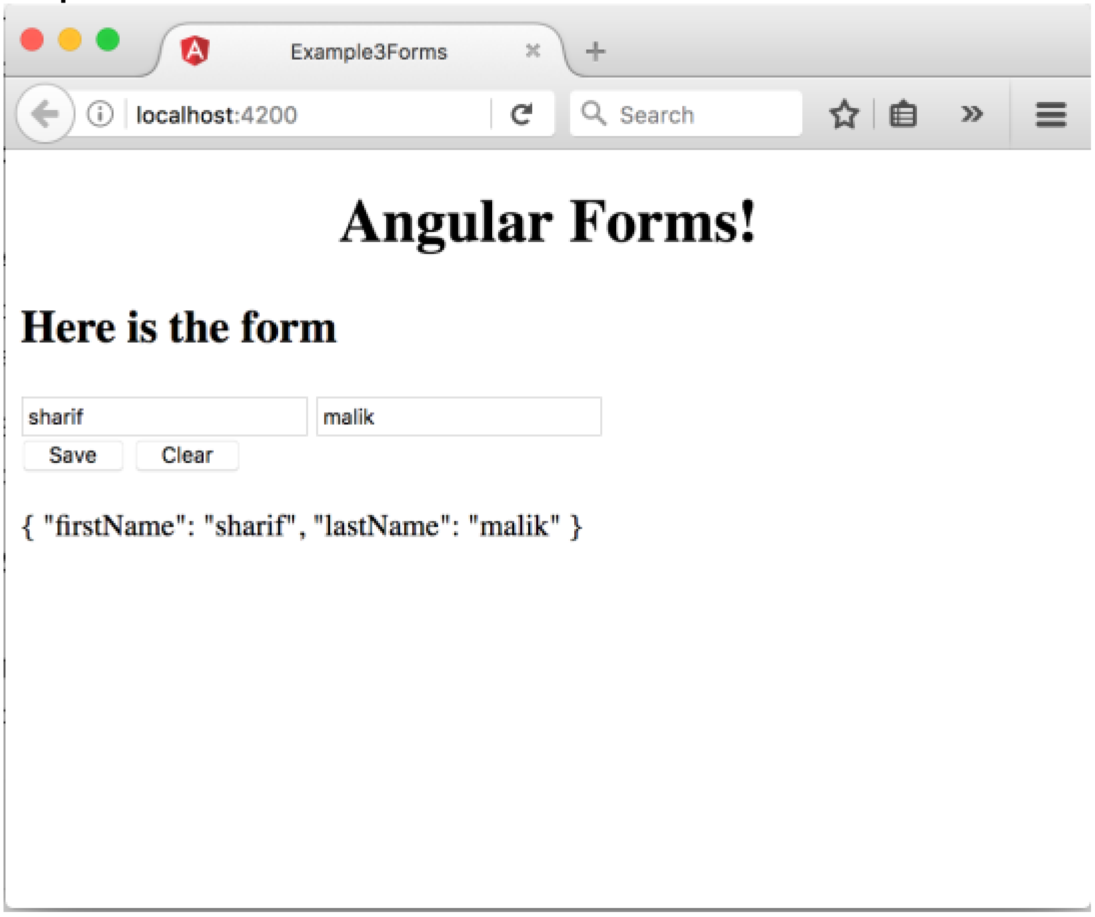
```
npm install
```

Step 2: Run the application

```
ng serve
```

Step 3: Open the browser,
go to http://localhost:4200 to the see the output of two-way binding

Output:



Angular Forms!

Here is the form

sharif malik

```
{ "firstName": "sharif", "lastName": "malik" }
```

Example4-services:

In this example: we are trying to services and inject them in the component and try to fetch data from our server.

You can download the source code from here (https://github.com/virtualSharif/gettingMEAN2/tree/master/angular/examples/example4-serivices)

Step1: Install all the dependencies

```
npm install
```

Step 2: Run the application

```
ng serve
```

Step 3: Open the browser,
go to http://localhost:4200 to the see the output of two-way binding

Example5-router:

In this example: we are trying to create two components and learn how to switch between components.

You can download the source code from here (https://github.com/virtualSharif/gettingMEAN2/tree/master/angular/examples/example5-router)

Step1: Install all the dependencies

```
npm install
```

Step 2: Run the application

```
ng serve
```

Step 3: Open the browser,
go to http://localhost:4200 to the see the output of two-way binding

References

https://angular.io/guide/quickstart

https://angular.io/guide/architecture

https://angular-university.io/