



# Angular Course

In this course you will get a hands-on of the major features of Angular

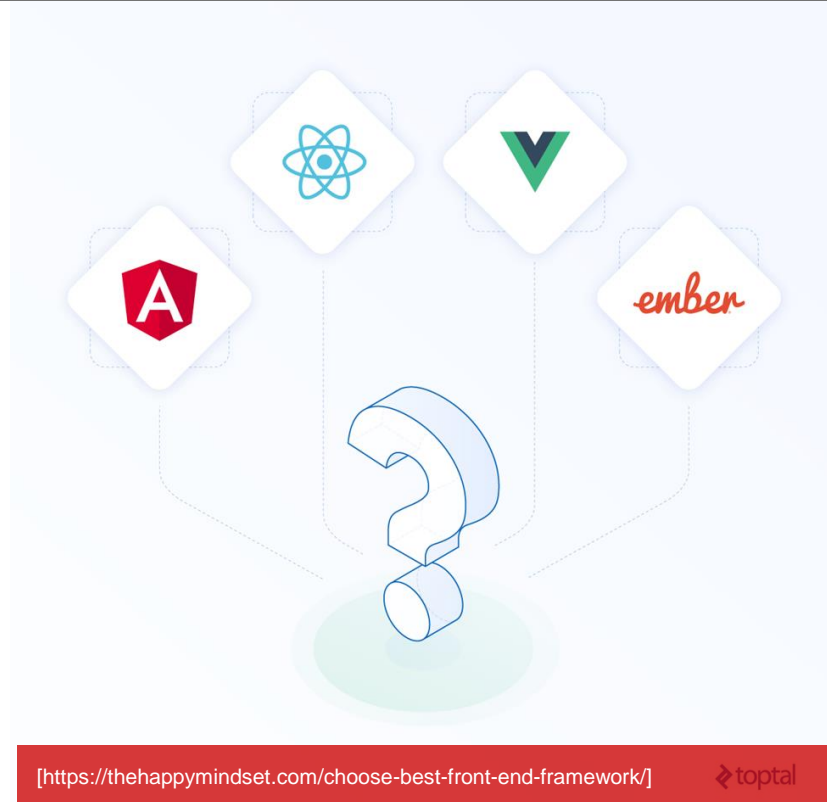
**David Oliveira**

Senior Consultant

23th January, 2018

# Why Angular?

- Availability of Learning Resources
- Popularity
- Core Features
- Usability
- Ease of Integration (with Other Libraries)
- Outputs: Web, Mobile, Desktop



- Components
- Forms
- Directives
- Pipes
- Services
- Routing
- Observables



- App\_Initializer
- Redux in Angular
- Unit Testing
- The mechanics of DOM updates in Angular (View encapsulation)
- Analyze an Angular App's Bundle Size
- Etc.



```
function doSomething(someData)
{
    // do something
}
```

**JS**  
**ES6**

TypeScript

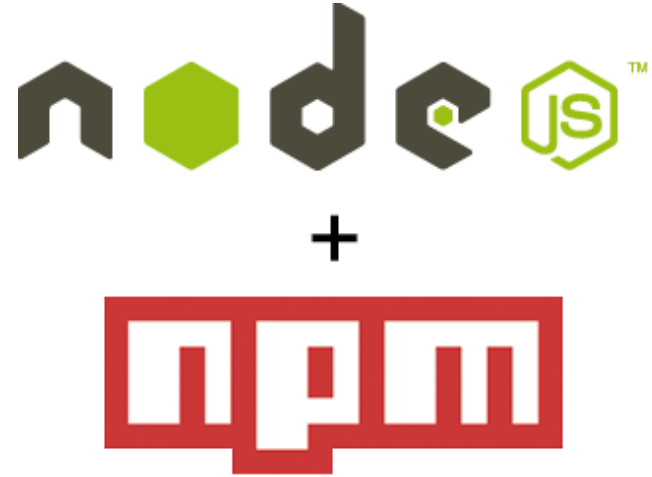
Type and Properties of someData?

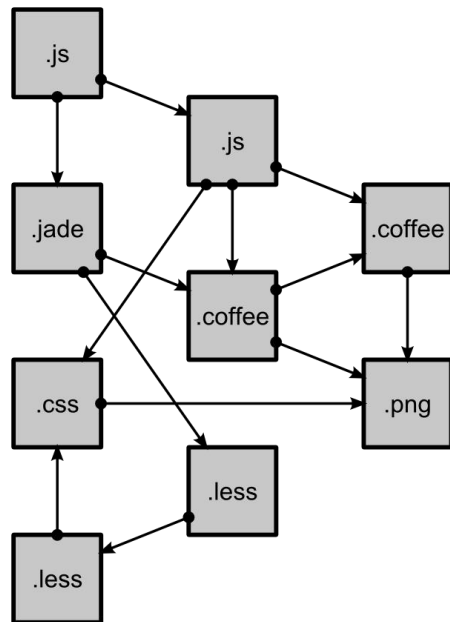
## Node JS

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

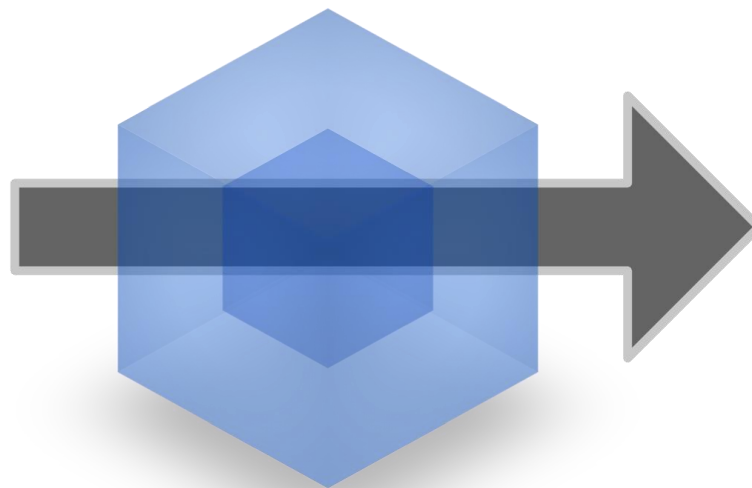
## NPM

Node.js' package ecosystem, [npm](https://www.npmjs.com/), is the largest ecosystem of open source libraries in the world.

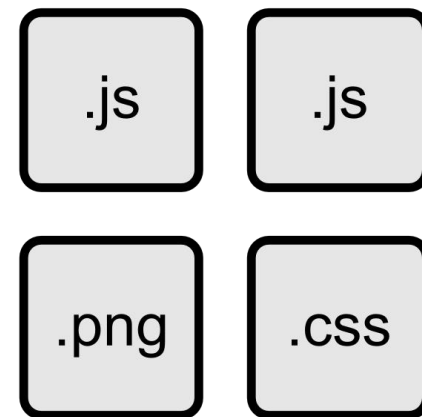




modules  
with dependencies



**webpack**  
MODULE BUNDLER



static  
assets

**The Angular CLI is a tool to initialize, develop, scaffold and maintain Angular applications**

```
npm install -g @angular/cli
```

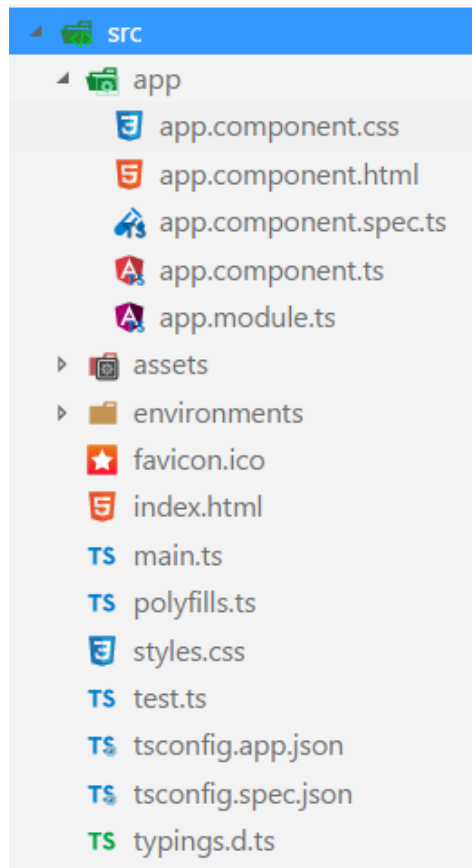
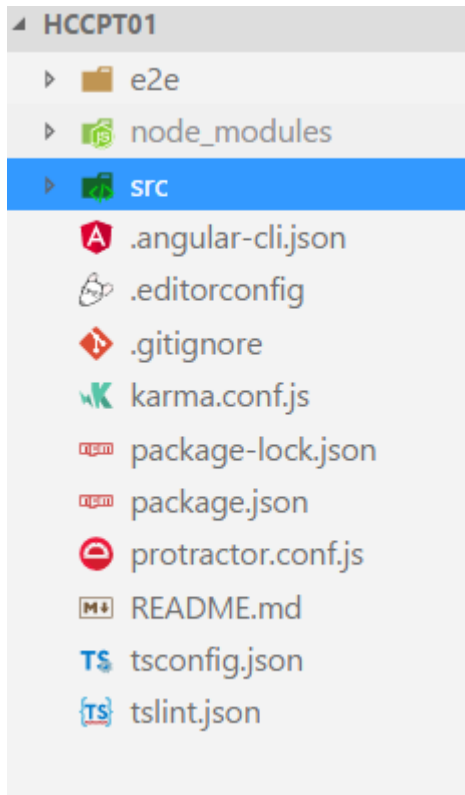
```
ng new hccpt  
cd hccpt  
ng serve
```

Navigate to <http://localhost:4200/>.

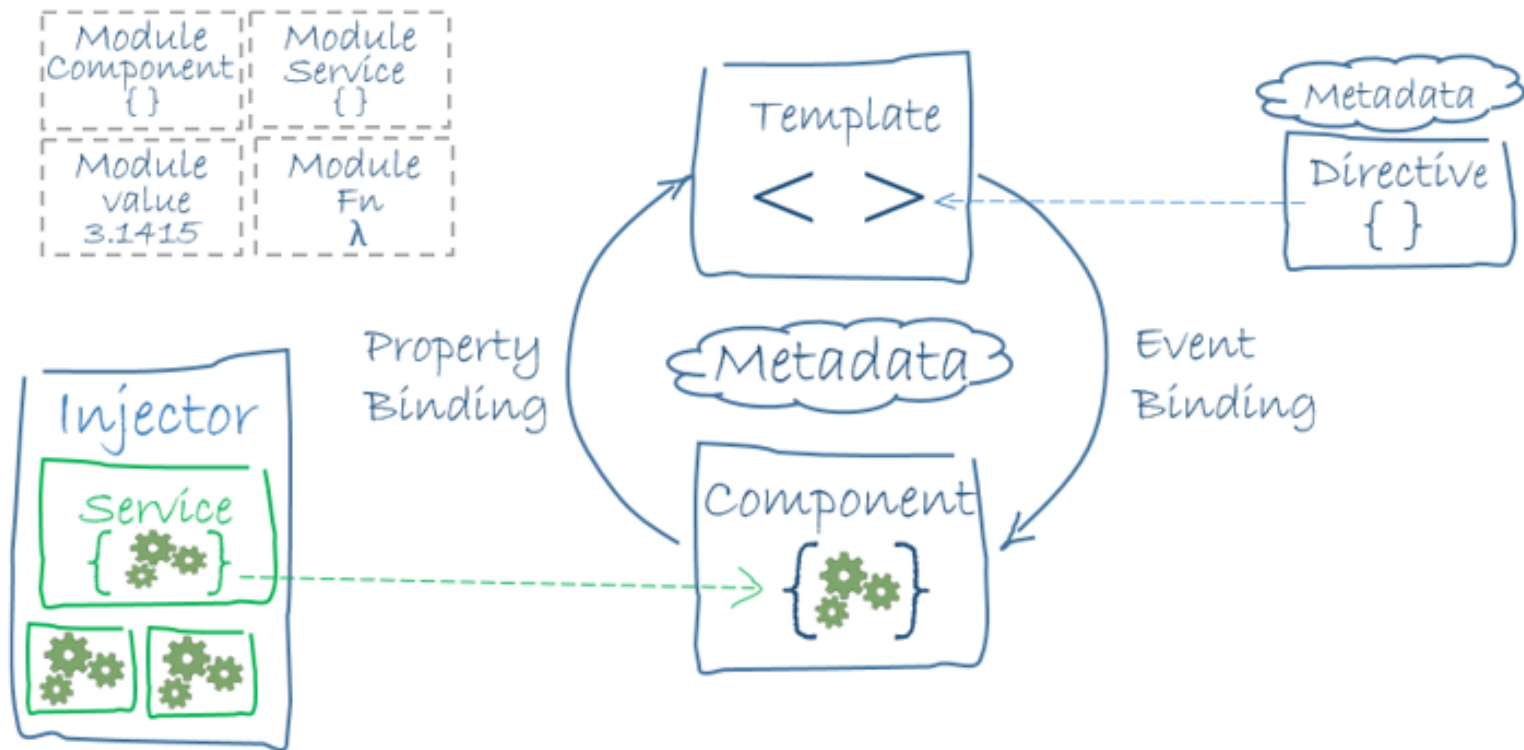
The app will automatically reload if you change any of the source files.



# Angular Solution File Structure



# Angular Architecture Overview





# Angular Course

In this course you will get a hands-on of the major features of Angular

**David Oliveira**

Senior Consultant

30<sup>th</sup> January, 2018

# Our App to build with Angular

We will build an application to manage a Condominium Administration with the following features:

- Add/Edit Contacts (Owners and Suppliers)
- Add/Edit Payments
- Dashboard with Charts about Payments



<https://github.com/davidoliveira/AngularCourse>

```
ng new KondominioApp --routing --style=scss
```

```
cd KondominioApp
```

```
npm install bootstrap@next --save
```

```
npm install --save @ng-bootstrap/ng-bootstrap
```

```
ng serve
```

<https://github.com/angular/angular-cli/wiki/generate>

<https://github.com/angular/angular-cli/wiki/stories>

<https://angular.io/guide/ngmodules>

<https://angular.io/guide/ngmodule-vs-jsmodule>

```
@NgModule({  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    BrowserModule,  
    AppRoutingModule,  
    NgbModule.forRoot()  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```

*declarations*—this application's lone component.

*imports*—import [BrowserModule](#) to have browser specific services such as DOM rendering, sanitization, and location.

*providers*—the service providers.

*bootstrap*—the *root* component that Angular creates and inserts into the index.html host web page.

**A declarable can only belong to one module, so only declare it in one `@NgModule`. When you need it elsewhere, import the module that has the declarable you need in it.**

ng g component Login

ng g module Contacts --routing

- ng g component Index
- ng g component Detail

ng g module Payments --routing

- ng g component Index
- ng g component Detail

<https://angular.io/guide/router>

<https://angular.io/guide/lazy-loading-ngmodules>

```
const routes: Routes = [
  { path: 'login', component: LoginComponent },
  {
    path: 'contacts',
    loadChildren: 'app/contacts/contacts.module#ContactsModule'
  },
  {
    path: 'payments',
    loadChildren: 'app/payments/payments.module#PaymentsModule'
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```





# Angular Course

In this course you will get a hands-on of the major features of Angular

**David Oliveira**

Senior Consultant

5<sup>th</sup> February, 2018

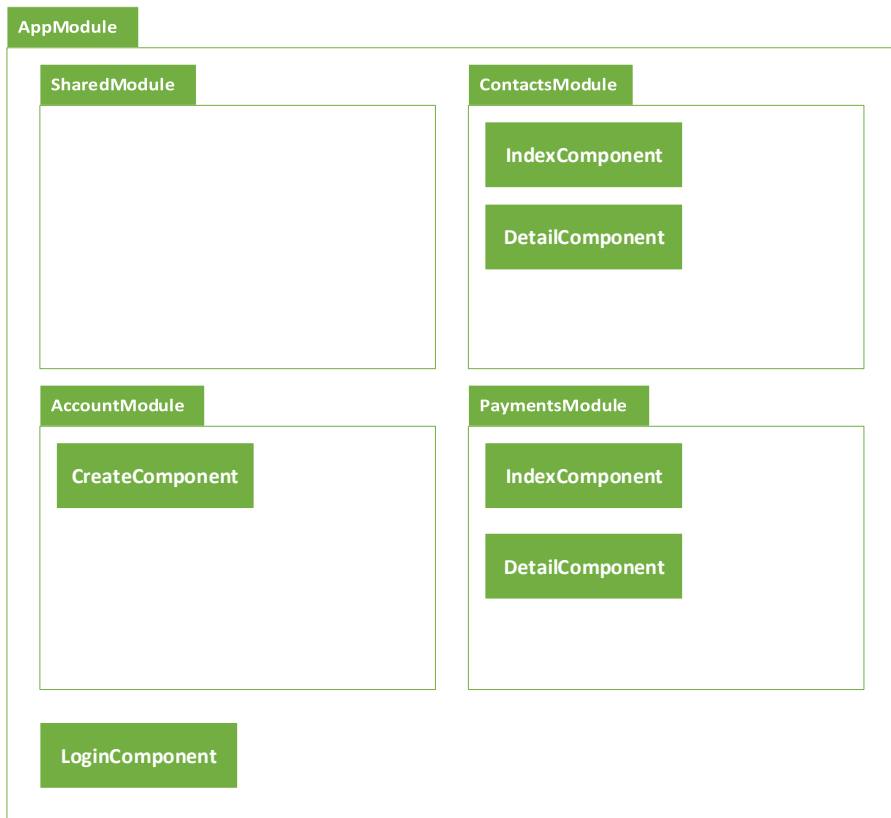
## Last Week

- Setup our App to manage a Condominium
- Add Modules and Components for each Module
- Setup Routing

## Angular CLI

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

# Our App Structure



```
const routes: Routes = [
  { path: 'login', component: LoginComponent },
  {
    path: 'account',
    loadChildren: 'app/account/account.module#AccountModule'
  },
  {
    path: 'contacts',
    loadChildren: 'app/contacts/contacts.module#ContactsModule'
  },
  {
    path: 'payments',
    loadChildren: 'app/payments/payments.module#PaymentsModule'
  }
];
```

Routing configuration to navigate through our application.  
Check Angular documentation about it:

<https://angular.io/guide/feature-modules>  
<https://angular.io/guide/lazy-loading-ngmodules>



# Angular Course

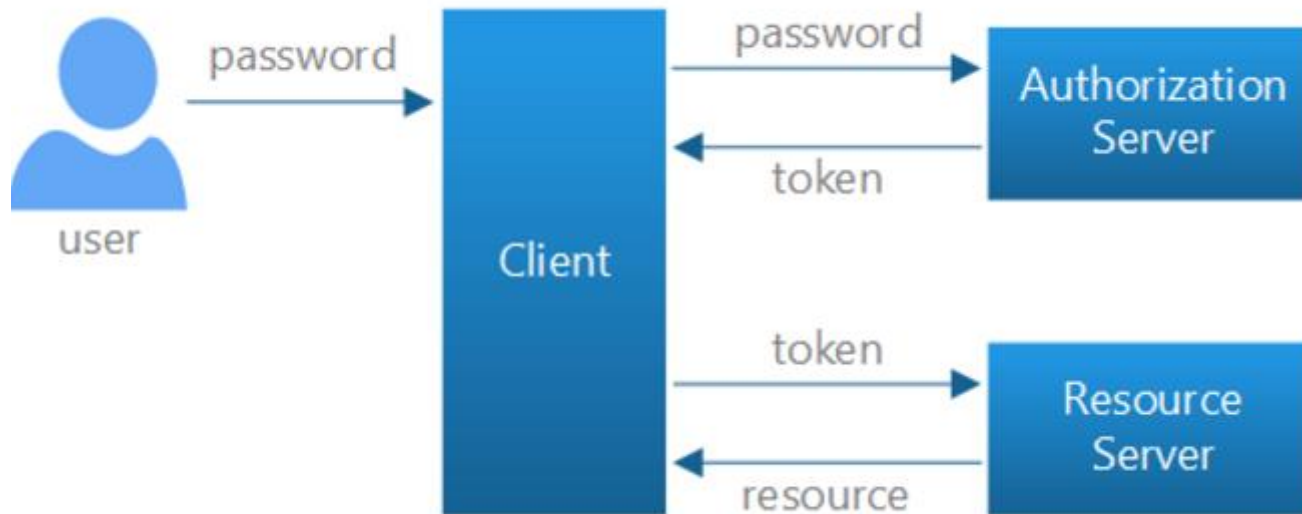
In this course you will get a hands-on of the major features of Angular

**David Oliveira**

Senior Consultant

27<sup>th</sup> February, 2018

# Secure our App



Angular's route guards are interfaces which can tell the router whether or not it should allow navigation to a requested route. They make this decision by looking for a true or false return value from a class which implements the given guard interface. There are five different types of guards and each of them is called in a particular sequence.

- **CanActivate** to mediate navigation *to* a route.
- CanActivateChild to mediate navigation *to* a child route
- CanDeactivate to mediate navigation *away* from the current route.
- CanLoad to perform route data retrieval *before* route activation.
- Resolve to mediate navigation *to* a feature module loaded *asynchronously*.

<https://angular.io/guide/router#milestone-5-route-guards>

Intercepts HttpRequest and handles them.

Any interceptor that we want to create needs to implement the `HttpInterceptor` interface. This means that our new class must have a method called `intercept` with `HttpRequest` and `HttpHandler` parameters. Using interceptors is all about changing outgoing requests and incoming responses, but we can't tamper with the original request—it needs to be immutable. To make changes we need to clone the original request.

The interceptor needs to be added to the `HTTP_INTERCEPTORS` array. This is done by making the existing `HTTP_INTERCEPTORS` array use the new class we've created. Add this in the providers array for our application's module.

<https://angular.io/api/common/http/HttpInterceptor>





# Angular Course

In this course you will get a hands-on of the major features of Angular

**David Oliveira**  
Senior Consultant  
6<sup>th</sup> March, 2018



ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences.

It extends the observer pattern to support sequences of data and/or events and adds operators that allow you to compose sequences together declaratively while abstracting away concerns about things like low-level threading, synchronization, thread-safety, concurrent data structures, and non-blocking I/O.

## Why Use Observables?

The ReactiveX Observable model allows you to treat streams of asynchronous events with the same sort of simple, composable operations that you use for collections of data items like arrays. It frees you from tangled webs of callbacks, and thereby makes your code more readable and less prone to bugs.

<http://reactivex.io/intro.html>

<https://hackernoon.com/understanding-creating-and-subscribing-to-observables-in-angular-426dbf0b04a3>



# Angular Course

In this course you will get a hands-on of the major features of Angular

**David Oliveira**

Senior Consultant

21<sup>th</sup> March, 2018

- Pass data from parent to child with input binding
- Intercept input property changes with a setter
- Intercept input property changes with ngOnChanges()
- Parent listens for child event
- Parent interacts with child via local variable
- Parent calls an @ViewChild()
- Parent and children communicate via a service

# Components Interaction: Exercise

KONDOMINIO






PAGAMENTOS

CONDÓMINOS

FORNECEDORES

Olá Administrator! SAIR

Pagamentos

Data	Valor	Descrição	
2018-01-08	€100.53	Quotização de Janeiro de 2018 do 17º	
2018-09-08	€100.53	Quotização de Setembro de 2018 do 17º	
2018-02-08	€100.53	Quotização de Fevereiro de 2018 do 17º	
2018-10-08	€100.53	<b>Quotização de Outubro de 2018 do 17º</b>	
2018-03-08	€100.53	Quotização de Março de 2018 do 17º	

Adicionar/Editar

Origem

Destino

Quotização de Outubro de 2018 do 17º

2018-10-08T00:00:00

100.53

Atualizar

Remover

- Develop the remove action at Payments page, at Index and Detail component
- Despite of the source, the payment should be deleted from the list at Index component
- API <https://kondominioapi.herokuapp.com/explorer/>

<https://angular.io/guide/component-interaction#component-interaction>



# Angular Course

In this course you will get a hands-on of the major features of Angular

**David Oliveira**

Senior Consultant

5<sup>th</sup> May, 2018

A pipe is to manipulate data, while a directive is more for DOM manipulation.

```
<tr *ngFor="let contact of contacts">  
  <td>{{contact.name | firstnameLastname}}</td>  
</tr>
```

```
@Pipe({  
  name: 'firstnameLastname'  
})  
export class FirstnameLastnamePipe implements PipeTransform {  
  transform(value: any, args?: any): any {  
    const words = value.split(' ');  
    if (words.length === 1) {  
      return words[0];  
    }  
    if (words.length >= 2) {  
      return words[0] + ' ' + words[words.length - 1];  
    }  
    return '';  
  }  
}
```

Angular comes with some common pipes, like date and upper case and lower case. You can also write your own pipes to handle custom scenarios that fit your application needs. Pipes are a great way to change data in a reusable way, without having to embed the transform logic within component classes and without having to modify the data just for display purposes.

<https://angular.io/guide/pipes>  
<https://angular.io/guide/attribute-directives#attribute-directives>

In Angular, a component is actually a directive with a template. Directives provide functionality and can transform the DOM. There are two types of directives:

- **Structural:** structural directives modify layout by altering elements in the DOM. Structural Directives change the structure of the view. Two examples are NgFor and NgIf.
- **Attribute:** change the appearance or behavior of an element, component, or another directive.

```
constructor(  
  private templateRef: TemplateRef<any>,  
  private viewContainer: ViewContainerRef) { }
```

```
constructor(el: ElementRef) {  
  el.nativeElement.style.backgroundColor = 'yellow';  
}
```

<https://angular.io/guide/pipes>  
<https://angular.io/guide/attribute-directives#attribute-directives>





# Angular Course

In this course you will get a hands-on of the major features of Angular

**David Oliveira**

Senior Consultant

15<sup>th</sup> May, 2018



The Angular CLI downloads and install everything you need to test an Angular application with the Jasmine, Karma and Protractor.

[Jasmine](#) is the framework we are going to use to create our tests. It has a bunch of functionalities to allow us the write different kinds of tests.

[Karma](#) is a task runner for our tests. It uses a configuration file in order to set the startup file, the reporters, the testing framework, the browser among other things.

[Protractor](#) is an end-to-end test framework for Angular and AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.

package.json

```
41 "jasmine-core": "2.8.0",  
42 "jasmine-spec-reporter": "4.2.1",  
43 "karma": "2.0.2",  
44 "karma-chrome-launcher": "2.2.0",  
45 "karma-coverage-istanbul-reporter": "1.4.2",  
46 "karma-jasmine": "1.1.2",  
47 "karma-jasmine-html-reporter": "0.2.2",  
48 "ng2-mqtt": "0.1.2",  
49 "protractor": "5.1.2",
```

**frameworks:** this is where jasmine gets set as a testing framework. If you want to use another framework this is the place to do it.

**reporters:** this is where you set the reporters. You can change them or add new ones.

**autoWatch:** if this is set to true, the tests run in watch mode. If you change any test and save the file the tests are re-build and re-run.

**browsers:** this is where you set the browser where the test should run. By default is google but you can install and use other browsers launchers.

```
module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular/cli'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage-istanbul-reporter'),
      require('@angular/cli/plugins/karma')
    ],
    client: {
      clearContext: false // leave Jasmine Spec Runner output visible in browser
    },
    coverageIstanbulReporter: {
      reports: [ 'html', 'lcovonly' ],
      fixWebpackSourcePaths: true
    },
    angularCli: {
      environment: 'dev'
    },
    reporters: ['progress', 'kjhtml'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome'],
    singleRun: false
  });
};
```

- An environment to run angular tests is being created using all the imports at the begging of the file.
- TestBed is a powerful unit testing tool provided by angular, and it is initialized in this file.
- Finally, karma loads all the tests files of the application matching their names against a regular expression. All files inside our app folder that has “spec.ts” on its name are considered a test.

```
import 'zone.js/dist/zone-testing';
import { TestBed } from '@angular/core/testing';
import {
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting
} from '@angular/platform-browser-dynamic/testing';

declare const require: any;

// First, initialize the Angular testing environment.
getTestBed().initTestEnvironment(
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting()
);
// Then we find all the tests.
const context = require.context('./', true, /\.spec\.ts$/);
// And load the modules.
context.keys().map(context);
```

```
import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';
import { User } from './user';

describe('AppComponent', () => {
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));
  it('should create the app', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));
  it('should have as title \'app\'', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('app');
  }));
  it('should render title in a h1 tag', async(() => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('Welcome to app!');
  }));
});
```

- We import all the angular testing tools that we are going to use.
- We import all the dependencies that this component has.
- We use a “describe” to start our test block with the title matching the tested component name.
- We use an async before each. The purpose of the async is to let all the possible asynchronous code to finish before continuing.

<https://angular.io/guide/testing>

Buld an unit test for the **FirstnameLastnamePipe** with the following test cases:

- with empty, should return empty
- with null, should return empty
- with "Bla", should return "Bla"
- with "Bla Bla Bla", should return "Bla Bla"

```
] describe('FirstnameLastnamePipe', () => {  
  | let pipe: FirstnameLastnamePipe;  
  
  | beforeEach(() => {  
    | pipe = new FirstnameLastnamePipe();  
    | });  
  
  | it('create an instance', () => {  
    | expect(pipe).toBeTruthy();  
    | });  
}
```

# Unit Tests of services with http

```
describe('AccountService', () => {
  let service: AccountService;
  let httpTestingController: HttpTestingController;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [
        HttpClientTestingModule
      ],
      providers: [AccountService]
    });
    service = TestBed.get(AccountService);
    httpTestingController = TestBed.get(HttpTestingController);
  });

  it('should get a user by id', () => {
    // Respond with mock data, causing Observable to resolve.
    // Subscribe callback asserts that correct data was returned.
    const testData = new User();
    testData.id = 1;

    service.getUserById(1).subscribe((user: User) => {
      expect(user).toEqual(testData);
    });

    const req = httpTestingController.expectOne(`${environment.apiUrl}/users/1`);
    expect(req.request.method).toEqual('GET');
    req.flush(testData);

    // Finally, assert that there are no outstanding requests.
    httpTestingController.verify();
  });
});
```

- We setup the TestBed importing the HttpClientTestingModule and providing the HttpTestingController.
- You can use the HttpTestingController to mock requests and the flush method to provide dummy values as responses. As the HTTP request methods return an Observable, we subscribe to it and create our expectations in the callback methods.

<https://angular.io/guide/testing>

# Unit Tests of services with http

- Create a unit test for **create** method of AccountService

```
describe('AccountService', () => {
  let service: AccountService;
  let httpTestingController: HttpTestingController;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [
        HttpClientTestingModule
      ],
      providers: [AccountService]
    });
    service = TestBed.get(AccountService);
    httpTestingController = TestBed.get(HttpTestingController);
  });

  it('should get a user by id', () => {
    // Respond with mock data, causing Observable to resolve.
    // Subscribe callback asserts that correct data was returned.
    const testData = new User();
    testData.id = 1;

    service.getUserById(1).subscribe((user: User) => {
      expect(user).toEqual(testData);
    });

    const req = httpTestingController.expectOne(`${environment.apiUrl}/users/1`);
    expect(req.request.method).toEqual('GET');
    req.flush(testData);

    // Finally, assert that there are no outstanding requests.
    httpTestingController.verify();
  });
});
```

<https://angular.io/guide/testing>



# Angular Course

In this course you will get a hands-on of the major features of Angular

**David Oliveira**

Senior Consultant

24<sup>th</sup> May, 2018



## Unit/Component

- Test a single Component, Service, Pipe etc.
- Test a single specific behaviour in a controlled environment
- Mock everything you need to test this functionality
- Use code coverage analysis to make sure everything is covered
- (Usually) Improves code structure and quality
- Test edge cases on the most detailed level
- Know if a change breaks a specific behavior
- Does not test if the whole application or a process works in real life

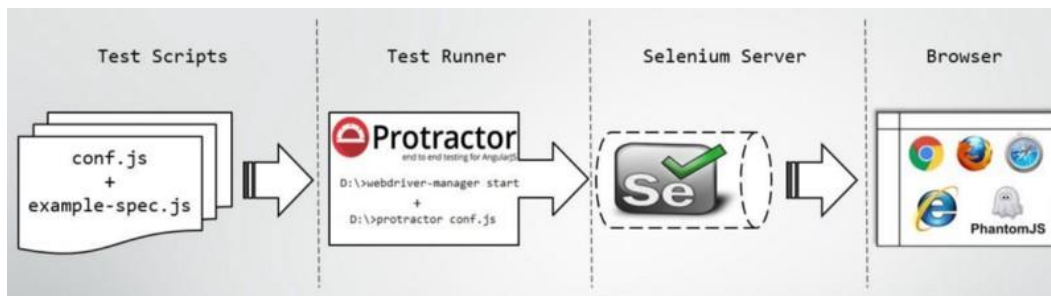
## End-2-End

- Test complete processes “End 2 End”
- Test in real environments with the whole application
- Test real live situations
- Know if most important features work with the complete application
- Focused on the overall functionality and not the specific code structure
- Does not test Edge cases
- Does not necessarily improve code quality

<https://angular.io/guide/testing>

# End to End tests with Protractor

Protractor is an official library to be used for writing E2E test suites with an Angular app. It's nothing but a wrapper over the Selenium WebDriverJS Api that translates its succinct code and methods to WebDriver JS methods.



There exists a Selenium Web Server that listens to Selenium commands in a special format called JSON Wire Protocol.

The automation script written in WebDriver JS shoots these HTTP commands to the browser driver to “drive” or “dictate” the actions it needs to perform via Selenium web server.

The server determines the steps needed to be performed by the web driver and execute them on the app being tested and send the status back to the automation script.

```
export class LoginPage {
  navigateTo() {
    return browser.get('/login');
  }

  getPageTitle() {
    return browser.getTitle();
  }

  setLoginUsernameFieldValue(value: string) {
    return element(by.id('login-email')).sendKeys(value);
  }

  getLoginUsernameFieldValue() {
    return element(by.id('login-email')).getAttribute('value');
  }
}

it('should login', () => {
  page.navigateTo();

  page.setLoginUsernameFieldValue(username);
  expect(page.getLoginUsernameFieldValue()).toEqual(username);

  page.setLoginPasswordFieldValue(password);
  expect(page.getLoginPasswordFieldValue()).toEqual(password);
});

Jasmine started

kondominio-app
  ✓ should have right title
  ✓ should login

Executed 2 of 2 specs SUCCESS in 3 secs.
```

- Prepare e2e Page Object to Access the Elements with @angular/cli, you get a e2e folder and structure that suggests to create a page object which helps you to access the elements. The page object for the default page is found in ./e2e/app.po.ts.
- Now we can write our e2e tests that use those elements to interact with the page and test its result.
- Run the tests: ng e2e



# Questions and Discussion

**HITACHI**  
Inspire the Next 