Angular

The "Batteries Included" Enterprise SPA Framework

What Is Angular?

- A Full Component-Based SPA Framework /Platform (not just a library)
- Developed By a Team Led by Google
- Open-Source (MIT License)
- Complete Rewrite of AngularJS
- Large Community
- Many High-Quality Third-Party Libraries
- Documentation and Training Widely Available

What Is Angular?

- Written in and usually used with Typescript:
  - A statically typed, object-oriented language with generics
  - A superset of ES2015, transpiles to ES5
- Component-Based Architecture
- Modular (both ES2015 and Angular-specific)

What Does Angular Provide?

- Angular CLI to generate project and add features
- Canonical Project Structure
- Ability to create Angular libraries and publish as NPM Modules
- RxJS everywhere
- Server-side Rendering (Angular Universal)
- Native Mobile Applications via NativeScript integration
- Testing (Jasmine / Karma / Protractor)

What Does Angular Provide?

- Components, Directives, Pipes, Services
- Dependency Injection
- Routing
- Http Client
- HTML Templates with CSS Encapsulation
- Data Binding (One-Way or Two-Way)
- Forms - Template-Driven or Reactive
- and More...

Is Angular a Good Fit For Your Development Team?

Angular May Be a Good Fit For You If:

- You have green-field development projects
  - Angular is not easily added to an existing application incrementally
- You develop medium to large sized applications
  - especially if you develop multiple applications: modularization makes it easy to share modules among applications

Is Angular a Good Fit For Your Development Team?

Angular May Be a Good Fit For You If:

- Your Development Team is not heavily invested in pure JS, and wants the infrastructure decisions already made
  - "Batteries Included" means that many decisions are made for you, so you may not have as much control as your team would like.
  - The Typescript decision has already been made
- Your Development Team is already productive with an Object-Oriented back-end framework, e.g. Java/Spring, .Net/C#
  - The similarities cut the learning curve significantly, and can be synergistic

Creating An Angular App (angular-cli)

- If you don’t have Node, install it from https://nodejs.org

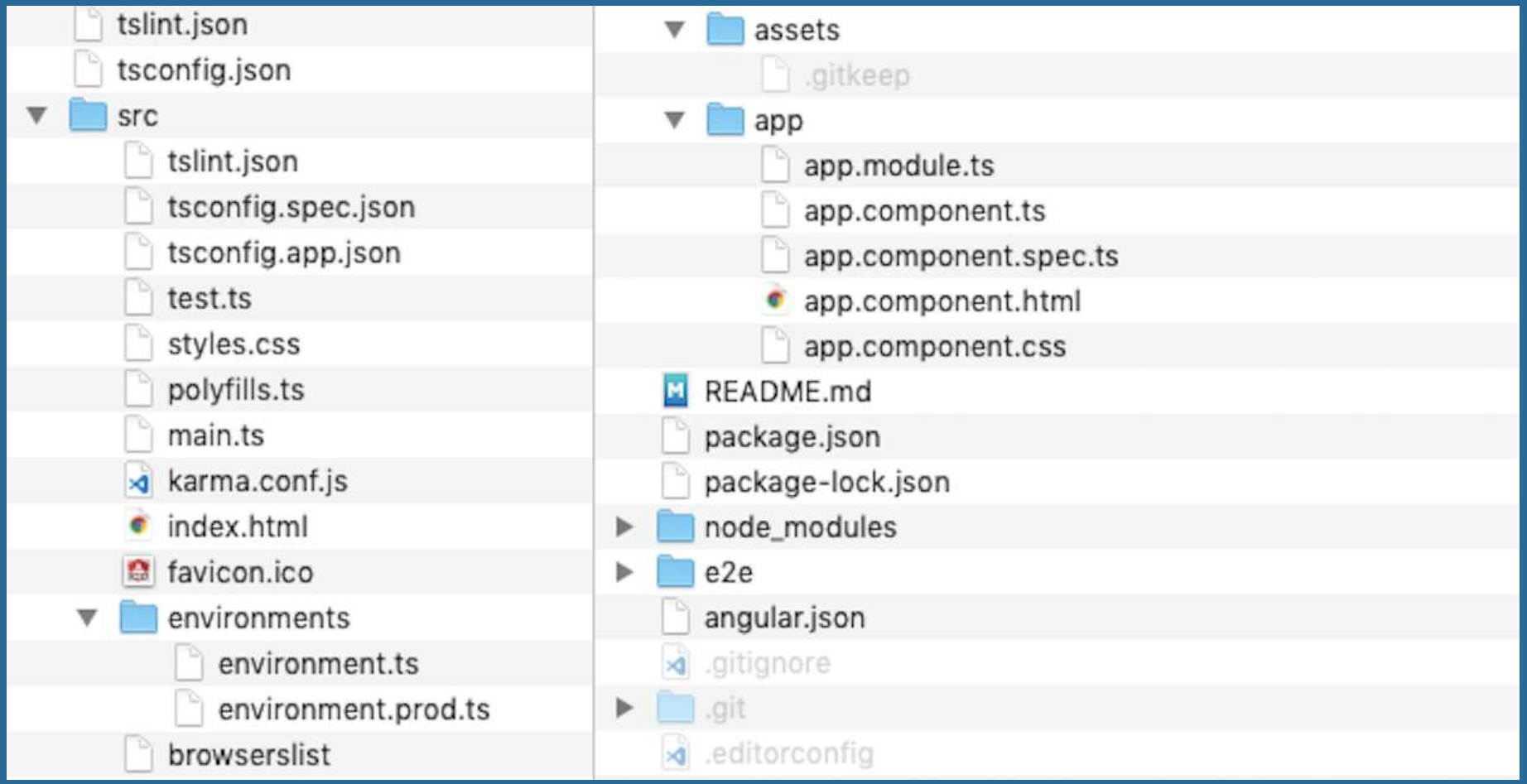
- If you don't have the Angular CLI installed, install it globally (one time only)

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

- Use the CLI to generate a minimal Angular application

```
$ ng new awesome-project
```

Generated Project



AppComponent

```
1       @Component({                                    (1)
2         selector: 'spa-root',
3         templateUrl: './app.component.html',
4         styleUrls: [
5           './app.component.scss'
6         ]
7       })
8       export class AppComponent {
9       }
```

1 Component Decorator

AppComponent HTML Template

```
      <div class="header">
        <img class="spaLogo" src="assets/head.png">
        <span class="title">Chariot SPA Day</span>
        <img class="chariotLogo" src="assets/chariot.svg">
      </div>
      <div class="app-container">
        <div class="home">
          <div class="row">
            <div class="column">
              <h2>Welcome to the SPA</h2>
              <spa-welcome></spa-welcome>
            </div>
            <div class="column">
              <router-outlet></router-outlet> (1)
            </div>
          </div>
        </div>
        <spa-chat></spa-chat> (2)
      </div>
```

1 router outlet - routed component renders here
2 <spa-chat> renders the ChatComponent, who's selector is 'spa-chat'

Rendered WelcomeComponent (<spa-welcome>)

Default Style Encapsulation:

- Content gets extra attributes, e.g. `_ngcontent-c0`
- CSS gets re-written to apply to these attributes

```
<spa-welcome _ngcontent-c0="" _nghost-c1="">
    <div _ngcontent-c1="" class="welcomeMessage">
        <div _ngcontent-c1="">
            Just float and wait for the wind to blow you around.
            Here's another <strong _ngcontent-c1="">little happy
            Let all these little things happen. Don't fight them
            Learn to use them. No worries. No cares.
        </div>
    </div
</spa-welcome>
```

Binding In Templates

- Simple Expression Binding

```
<span>Your Login Name: {{loginName}}</span>
```

- Event Binding

```
<button (click)="onClicked()">Push Me</button>
```

- Two-Way Binding ("Banana In A Box")

```
 <input name="login" [(ngModel)]="loginName">
```

- One-Way Binding

```
<input name="login"
      [ngModel]="loginName"
      (ngModelChange)="onLoginNameChange($event)"
>
```

Event handler instead of updating the value directly

App Module

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

Router Module

```
1   import { NgModule } from '@angular/core';
2   import { Routes, RouterModule } from '@angular/router';
3    ...
4   const routes: Routes = [
5     { path: '', component: ScheduleComponent },
6     { path: 'registration/:sessionId', component: RegistrationCompo
7     { path: 'confirmation/:email/:sessionId', component: Confirmat
8     { path: '**', redirectTo: '/'}
9   ];
10
11  @NgModule({
12    imports: [RouterModule.forRoot(routes)],
13    exports: [RouterModule]
14  })
15  export class AppRoutingModule { }
```

HttpClient

Simple Example

```
1  getSession(id: number): Observable<Session> {
2   return this.http.get<Session>(`${environment.apiUrl}/session/${id}`),
3  }
```

- Note the use of RxJS Observables.
  - The get() method returns an Observable

HttpClient

More Complex Example

```
1   getSchedule(): Observable<Session[]> {
2     return this.http.get<Session[]>(`${environment.apiUrl}/session`)
3       mergeMap( schedule =>
4          forkJoin(schedule.map(session =>
5           this.getRegistrationsForSession(session.id).pipe(
6             map((registrations) =>
7                 <Session> {...session, registrationCount: registrati
8             )
9           )
10        ))
11      )
12    );
```

- Get a list of Sessions, then get the Registrants for each Session.

Observables (subscribing)

```
1    private getSchedule() {
2      this.scheduleService.getSchedule()
3        .pipe(delay(2000)) (1)
4        .subscribe(
5          schedule => this.schedule = schedule, (2)
6          err => console.log('Oops - failed to get schedule', err) (3)
7        );
8    }
```

- (1) Subscriber introduces a delay with the rx delay operator (to simulate a slow network), by chaining the operator into the Observable's stream handling, could do other things, like map, filter, etc.
- (2) Succesful result is handled by the first function
- (3) Any error, regardless of which of the chained Http calls produced the error, is handled by the second function.

Directives

- Marked by the @Directive decorator
- Like Components (they affect the View), but do not have their own template
- Structural Directives modify the structure of the DOM
  - most prominent built-ins are NgIf (conditional) and NgFor (iterator)
- Attribute Directives modify the behavior or appearance of an element
- For instance, an attribute directive might highlight the text of an element that has a particular attribute.

Use of Structural Directives

```
1   <ng-container *ngIf="schedule;else busy"> (1)
2     <div class="schedule">
3       <div
4           class="sessionLink"
5           (click)="bookSession(session.id)"
6           *ngFor="let session of schedule" (2)
7       >
8         <div>
9           <div class="time">{{session.date | shortTime}} - {{session
10          <div class="date">{{session.date | shortDate}}</div>
11        </div>
12        <div class="attendees">{{session.registrationCount}}</div>
13        <div class="link"></div>
14      </div>
15    </div>
16  </ng-container>
```

1 *ngIf - conditional rendering
2 *ngFor - iteration

Custom Attribute Directive

Custom Attribute Directive

```
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
    constructor(el: ElementRef) {
       el.nativeElement.style.backgroundColor = 'yellow';
    }
}
```

Using the Custom Attribute Directive

```
<p appHighlight>Highlight me!</p>
```

Pipes

- Classes that transform values, ususally used in a view
- example of a built-in is the DatePipe, which takes a date and outputs a formatted string.

```
1       @Pipe({  (1)
2         name: 'shortDate'  (2)
3       })
4       export class ShortDatePipe implements PipeTransform {
5
6           static FORMAT = 'MMMM Do';
7
8           transform(value: any, args?: any): any {
9            return format(value, ShortDatePipe.FORMAT);
10          }
11      }
```

1 Pipe Decorator
2 Pipe's name

Usage of Pipes

```
1   <ng-container *ngIf="schedule;else busy">
2     <div class="schedule">
3       <div class="sessionLink" (click)="bookSession(session.id)" *ngI
4         <div>
5           <div class="time">{{session.date | shortTime}} - {{session
6           <div class="date">{{session.date | shortDate}}</div> (1)
7         </div>
8         <div class="attendees">{{session.registrationCount}}</div>
9         <div class="link"></div>
10      </div>
11    </div>
12  </ng-container>
```

1 use of shortDate pipe

Angular Take-Aways

- Angular is a full-featured SPA Framework
- Good Fit for Dev Teams without attachment to Pure JS
- Good Fit for Dev Teams currently using OO back-ends
- Good Fit for Medium-to-Large Applications
- Large Community
- Lots of Resources

Angular Take-Aways

- Code is Written in TypeScript
- Views are written (mostly) in HTML
- RxJS is used Everywhere
- Easy to create libraries from Components, Services, etc. and reuse them
- Can be used for both Web and Native Mobile Development