Demo Manual



Copyright © 2019-2021

Funny Ant, LLC

All rights reserved.

No part of this book may be reproduced in any form or by any electronic or mechanical means including information storage and retrieval systems, without permission from the author.

Table of Contents

- Demo Manual
  - Table of Contents
  - Understanding this Manual
  - Overview
    - Getting Started
    - Reset
    - External vs. Inline Styles
  - Components
    - First & Templates (Internal vs. External)
    - Nesting
  - Modules
    - Declarations
    - Imports & Exports (Feature Modules)
  - Components (continued...)
    - JSON Pipe
    - ngFor
    - Interpolation
    - Property Binding
    - Input Properties
    - Event Binding
    - Pipes
    - Output Events
    - Component Styling
    - ngIf
    - ngSwitch
  - Forms
    - Reactive Forms Binding
    - Reactive Forms Validation
      - UpdateOn
    - Reactive Forms Validation Messages
    - Reactive Forms Custom Validator
    - ngClass
      - Displaying Validation Messages
  - Services
  - RxJS
    - Observables
      - Creating a Stream of DOM Events
      - Listening for keyup events
    - Observers
    - Subcriptions
    - Operators
      - map

        - tap
        - filter
    - Practical Example
        - debounceTime
        - distinctUntilChanged
        - switchMap
- Http
    - Get
    - Error Handling
    - Retry
- Routing
    - Routing Basics
    - Routing Navigation
        - Code Review
        - Routing Changes
- Routing (Advanced)
    - Child Routes
    - Lazy Loading
- Components (Advanced)
    - Change Detection
        - Checkout
        - ChangeDetectionStrategy.Default
        - ChangeDetectionStrategy.OnPush
- Appendixes
    - Redux (NgRx)
        - Setup
        - Installation
        - Store
        - Actions
        - Reducer
        - Component
    - Template-driven Forms
        - Template Forms Binding
        - Template Forms Validation
            - Displaying Validation Messages
        - Two-way Binding
    - Bootstrapping an Application
- Setup
    - Creating this project
    - Creating the http-start branch
- Feedback

Understanding this Manual

Overview

This is the demo manual and is **not provided to attendees until the end of class**. It's purpose is to provide step-by-step directions for instructor demonstrations (demos) used in class.

Attendees do, however, receive the finished code for each demonstration at the beginning of class as part of the course files zip.

It is **recommended that the instructor print a hard copy** of this demo manual and use it as a reference to complete the demonstrations.

Getting Started

1. Open `code\demos\start` as the top level folder in your editor.
2. Open a command-prompt or terminal in `code\demos\start` and run the command.

```
npm install
```

3. Build and start the demo application by running the following command.

```
ng serve -o
```

You can leave `ng serve` running when changing between demos or writing code and it will automatically rebuild your code and reload the browser so you can see your changes.

Reset

After starting a demo you will need to reset the code before beginning the next demo. You can do this by running the following commands.

```
git checkout start -f //checks out the start branch
git clean -df // removes untracked directories and files df
```

For more information see this link about git-clean

External vs. Inline Styles

IMPORTANT

The demos project uses inline templates and inline styles (the html and css is write in the ts file). This is intentional as it greatly helps students see the connection between code in the html and the ts file.

In addition, we have found that when learning...less files equals less confusion.

The official Angular Style Guide recommends to **Extract templates and styles to their own files** (i.e. external templates and styles).

> This recommendation IS followed in the student labs to teach best practices.

> This recommendation is NOT followed in the demos to facilitate learning.

> Reference: Angular Style Guide: Extract templates and styles to their own files
>
> - https://angular.io/guide/styleguide#extract-templates-and-styles-to-their-own-files

Accordingly, this project was created using the following command and it can be useful to explain this before doing the first demo.

```
ng new start --routing --inline-style --inline-template --skip-tests
```

Components

First & Templates (Internal vs. External)

```
ng g component hello-world -d
```

app.component.ts

```
@Component({
  selector: 'app-root',
  template: `
    <app-hello-world></app-hello-world>
  `,
  styles: []
})
export class AppComponent {}
```

create the file hello-world/hello-world.component.html

```
<p>hello-world still works!</p>
```

app.component.ts

```
@Component({
  selector: 'app-hello-world',
+  templateUrl: `./hello-world.component.html`,
  styles: []
})
export class HelloWorldComponent implements OnInit {
  constructor() {}

  ngOnInit() {}
}
```

Nesting

```
ng g component my-button
```

Point out that it was added to app.module.ts declarations

`hello-world/hello-world.component.html`

```
<p>
  hello-world still works!
+ <app-my-button></app-my-button>
</p>
```

Modules

Declarations

```
ng g c tag-one
```

Show how component was added to the `declarations` in the app module.

`app.module.ts`

```
+ import { TagOneComponent } from './tag-one.component';
 ...
@NgModule({
  declarations: [AppComponent,
+               TagOneComponent]
   ...
})
```

```
ng g c tag-one
ng g c tag-two
ng g c tag-three
```

Add new component selectors/tags to the app.component template.

```
@Component({
  selector: 'app-root',
  template: `
    <app-tag-one></app-tag-one>
    <app-tag-two></app-tag-two>
    <app-tag-three></app-tag-three>
  `,
  styles: []
})
export class AppComponent {}
```

Imports & Exports (Feature Modules)

```
ng g module orders
ng g c orders/order-list
ng g c orders/order-detail
ng g c orders/order-form
ng g service orders/shared/order
//note that the service is injected in root we'll talk about that
later
ng g pipe orders/shared/order-number
```

import orders module in app module

```
+ import { OrdersModule } from './orders/orders.module';

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

- Exports add `project-detail` to app component template

`app.component.ts`

```
@Component({
  selector: 'app-root',
  template: `
+   <app-order-detail></app-order-detail>
  `,
  styles: []
})
export class AppComponent {}
```

- doesn't work yet

- export project-detail component

`orders\orders.module.ts`

```
@NgModule({
  imports: [CommonModule],
  declarations: [
    OrderListComponent,
    OrderDetailComponent,
    OrderFormComponent,
    OrderNumberPipe
  ],
+  exports: [OrderDetailComponent]
})
export class OrdersModule {}
```

- Now it should render `order detail works`.

Components (continued...)

JSON Pipe

Just start with the solution branch and remove and add back the couple lines showing the `json pipe`.

```
git checkout json-pipe -f
```

```
@Component({
  selector: 'app-root',
  template: `
    <div>
      <p>Without JSON pipe:</p>
      <p>{{ object }}</p>
      <p>With JSON pipe (no pre tag):</p>
      <p>{{ object | json }}</p>
      <p>With JSON pipe (and pre tag):</p>
      <pre>{{ object | json }}</pre>
    </div>
  `,
  styles: []
})
export class AppComponent {
  object: Object = {
    foo: 'bar',
    baz: 'qux',
    nested: { xyz: 3, numbers: [1, 2, 3, 4, 5] }
  };
}
```

ngFor

```
@Component({
  selector: 'app-root',
  template: `
    <ul>
      <li *ngFor="let fruit of fruits">{{ fruit }}</li>
    </ul>
  `,
  styles: []
})
export class AppComponent {
  fruits = ['Apple', 'Orange', 'Plum'];
}
```

```
+  <li *ngFor="let fruit of fruits; let i = index;">
     {{i + 1}}.
     {{fruit}}
   </li>
```

The Angular logo used for this demonstration is available in the `src\assets` directory of the start branch.

Interpolation

```
@Component({
  selector: 'app-root',
  template: `
    <h2>{{ image.name }}</h2>
    <p>{{ image.path }}</p>
  `,
  styles: []
})
export class AppComponent {
  image = {
    path: '../assets/angular_solidBlack.png',
    name: 'Angular Logo'
  };
}
```

Property Binding

```
@Component({
  selector: 'app-root',
  template: `
    <img [src]="image.path" [alt]="image.name"
[title]="image.name" />
  `,
  styles: []
})
export class AppComponent {
  image = {
    path: '../assets/angular_solidBlack.png',
    name: 'Angular Logo'
  };
}
```

Input Properties

```
git checkout ngFor -f
```

Generate a list component.

```
ng g component fruit-list
```

```
// fruit-list/fruit-list.component.ts

import { Component, OnInit, Input } from '@angular/core';

@Component({
  selector: 'app-fruit-list',
  template: `
    <ul>
      <li *ngFor="let fruit of fruits">{{ fruit }}</li>
    </ul>
  `,
  styles: []
})
export class FruitListComponent implements OnInit {
  @Input()
  fruits: string[];
  constructor() {}

  ngOnInit() {}
}
```

app.component.ts

```
@Component({
  selector: 'app-root',
  template: `
    <app-fruit-list [fruits]="data"></app-fruit-list>
  `,
  styles: []
})
export class AppComponent {
  data: string[] = ['Apple', 'Orange', 'Plum'];
}
```

Event Binding

```
@Component({
  selector: 'app-root',
  template: `
    <a href="" (click)="onClick($event)">Click Me!</a>
    <p [innerText]="message"></p>
  `,
  styles: []
})
export class AppComponent {
  message = '';

  onClick(event) {
    event.preventDefault();
    this.message = 'clicked';
  }
}
```

Pipes

```
git checkout pipes -f
```

```
@Component({
  selector: 'app-root',
  template: `
    <table>
      <tr>
        <th>Pipe Expression</th>
        <th>Formatted Output</th>
      </tr>
      <tr>
        <td>amount | currency: 'USD': "symbol": '2.1-2'</td>
        <td>{{ amount | currency: 'USD':'symbol':'2.1-2' }}</td>
      </tr>
      <tr>
        <td>releaseDate | date : 'MM/dd/yyyy'</td>
        <td>{{ releaseDate | date: 'MM/dd/yyyy' }}</td>
      </tr>
      <tr>
        <td>amount | number: '3.3-4'</td>
        <td>{{ amount | number: '3.3-4' }}</td>
      </tr>
      <tr>
        <td>percentOfGross | percent: '2.2'</td>
        <td>{{ percentOfGross | percent: '2.2' }}</td>
      </tr>
    </table>
  `,
  styles: []
})
export class AppComponent {
  amount = 47.341;
  releaseDate: Date = new Date(1975, 4, 25);
  percentOfGross = 0.3245;
}
```

Output Events

```
ng g component email-subscribe
```

```
// email-subscribe.ts

import { Component, OnInit, Output, EventEmitter } from
'@angular/core';

@Component({
  selector: 'app-email-subscribe',
  template: `
    <input type="text" #email placeholder="email" />
    <button (click)="onClick(email.value)">Subscribe</button>
  `,
  styles: []
})
export class EmailSubscribeComponent implements OnInit {
  @Output()
  subscribe = new EventEmitter<string>();
  constructor() {}

  ngOnInit() {}
  onClick(email: string) {
    this.subscribe.emit(email);
  }
}
```

```
// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <app-email-subscribe
      (subscribe)="onSubscribe($event)"
    ></app-email-subscribe>
    {{ message }}
  `,
  styles: []
})
export class AppComponent {
  message: string;
  onSubscribe(email) {
    this.message = `Successfully subscribed. Please check your
```

```
  email ${email} and click link.`;
    }
  }
```

Component Styling

- Inline `app.component.ts`

```
styles: [
  `
    h1 {
      color: rgb(255, 165, 0);
    }
  `
];
```

- External `app.component.ts`

```
@Component({
  selector: 'app-root',
  template: `
    <h1>Welcome to {{ title }}!</h1>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'playground';
}
```

`app.component.css`

```
h1 {
  color: rgb(255, 165, 0);
}
```

ngIf

```
@Component({
  selector: 'app-root',
  template: `
    <div>
      <input type="text" placeholder="username" />
      <input type="text" placeholder="password" />
      <button (click)="signIn()">Sign In</button>
    </div>

    <div>Welcome back friend.</div>
  `,
  styles: []
})
export class AppComponent {
  isSignedIn = false;

  signIn() {
    this.isSignedIn = true;
  }
}
```

```
  <div
+  *ngIf="isSignedIn">
  Welcome back friend.
  </div>
```

```
  <div
+  *ngIf="!isSignedIn">
  <input type="text" placeholder="username">
   ...
```

- nglf; else

```
@Component({
  selector: 'app-root',
  template: `
  <div
+ *ngIf="!isSignedIn; else signedIn">
  <input type="text" placeholder="username">
  <input type="text" placeholder="password">
  <button (click)="signIn()">Sign In</button>
  </div>

+ <ng-template #signedIn>
  <div>
  Welcome back friend.
  </div>
+ </ng-template>
  `,
  styles: []
})
```

ngSwitch

```
git checkout ngFor -f
```

```
<div [ngSwitch]="fruits.length">
  <p *ngSwitchCase="0">No records returned.</p>
  <p *ngSwitchCase="1">1 record returned.</p>
  <p *ngSwitchDefault>{{fruits.length}} records were returned.
</p>
</div>
```

Forms

Reactive Forms Binding

`app.module.ts`

```
+ import { ReactiveFormsModule } from '@angular/forms';

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

> Note: You may have to manually type the import

> It is easier to understand to type the component code first, then the template.

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';
@Component({
  selector: 'app-root',
  template: `
    <h1>Forms</h1>
    <form [formGroup]="loginForm" (submit)="onSubmit()">
      <input formControlName="username" type="text"
name="username" /> <br />
      <input formControlName="password" type="text"
name="password" /> <br />
      <button>Sign In</button>
    </form>
    <pre>
    {{ loginForm.value | json }}
  </pre
    >
  `,
  styles: []
})
export class AppComponent implements OnInit {
  loginForm: FormGroup;
  ngOnInit(): void {
    this.loginForm = new FormGroup({
      username: new FormControl(),
      password: new FormControl()
    });
  }
  onSubmit() {
    console.log(this.loginForm.value);
  }
}
```

Reactive Forms Validation

app.component.ts

```
export class AppComponent implements OnInit {
loginForm: FormGroup;
ngOnInit(): void {
  this.loginForm = new FormGroup(
    {
      username: new FormControl(null,
+ Validators.required),
      password: new FormControl()
    });
}
onSubmit() {
  console.log(this.loginForm.value);
}
}
```

app.component.html

```
 ...
<pre *ngIf="loginForm.get('username').invalid">
{{loginForm.get('username').errors | json}}
</pre>
```

1. See validation errors on refresh.
2. Enter username.
3. Error does not display.
4. Delete username see error displayed again.

UpdateOn

app.component.ts

```
  export class AppComponent implements OnInit {
  loginForm: FormGroup;
  ngOnInit(): void {
    this.loginForm = new FormGroup(
      {
        username: new FormControl(null, Validators.required),
        password: new FormControl()
      },
+     { updateOn: 'blur' }
    );
  }
  onSubmit() {
    console.log(this.loginForm.value);
  }
  }
```

Reactive Forms Validation Messages

`app.component.html`

```
...
<div
  *ngIf="loginForm.get('username').dirty &&
loginForm.get('username').invalid &&
loginForm.get('username').touched"
>
  <div *ngIf="loginForm.get('username').hasError('required')">
    Username is required.
  </div>
</div>
```

> Creating the inner `div` first and then wrapping with the outer `div` can facilitate understanding.

Follow these steps to display the validation message. This is intentional but can be confusing.

- Enter some text in the email field
- Delete the text
- Tab out of the email input or cause the email input to lose focus in some way

Reactive Forms Custom Validator

Validate password doesn't contain the phrase password "password".

- Start with this branch

```
git checkout reactive-forms-validation
```

- Output password validation errors

```
<pre *ngIf="loginForm.get('password').invalid">
 {{loginForm.get('password').errors | json}}
 </pre
>
```

- Create Custom validator

Do this in the `app.component.ts` file to make it easier to follow the code.

```
export class CustomValidators {
  static forbiddenPhrase(control: AbstractControl):
ValidationErrors | null {
    if (control.value) {
      if (control.value.toLowerCase() === 'password') {
        return { forbiddenPhrase: true };
      }
    }
    return null;
  }

  // create method signature
  // paste existing validation function, convert inner function
to arrow by removing name and adding => before opening curly
brace
  // add semi-colon at the end
  // remove hard-coded phrase
  static forbiddenPhraseValidatorFn(phrase: string): ValidatorFn
{
    return (control: AbstractControl): ValidationErrors | null =>
{
      if (control.value) {
        if (control.value.toLowerCase() === phrase) {
          return { forbiddenPhrase: true };
        }
      }
```

```
        return null;
      };
    }
  }
```

- Add custom validator. app.component.ts

```
import { Component, OnInit } from '@angular/core';
import {
  FormGroup,
  FormControl,
  Validators,
  AbstractControl,
  ValidatorFn,
  ValidationErrors
} from '@angular/forms';
...
export class AppComponent implements OnInit {
  loginForm: FormGroup;
  ngOnInit(): void {
    this.loginForm = new FormGroup(
      {
        username: new FormControl(null, Validators.required,
        password: new FormControl(
          null,
+
CustomValidators.forbiddenPhraseValidatorFn('password')
        )
      }
+       // { updateOn: 'blur' }
    );
  }
  onSubmit() {
    console.log(this.loginForm.value);
  }
}
```

ngClass

```
@Component({
  selector: 'app-root',
  template: `
    <p>We need to button up our approach out of the loop ... </p>
  `,
  styles: [
    `
      .highlight {
        background-color: #ffff00;
      }
    `
  ]
})
export class AppComponent {}
```

```
@Component({
  selector: 'app-root',
  template: `
+    <p (click)="onClick()" [class.highlight]="isHighlighted">
     We need to button ...
     </p>
  `,
  styles: [
    `
      .highlight {
        background-color: #ffff00;
      }
    `
  ]
})
export class AppComponent {
+  isHighlighted = false;
+  onClick() {
+    this.isHighlighted = !this.isHighlighted;
+  }
}
```

```
@Component({
  selector: 'app-root',
  template: `
    <p (click)="onClick()" [class.highlight]="isHighlighted"
+   [class.underline]="true">
    We need to button ...
    </p>
  `,
  styles: [
    `
      .highlight {
        background-color: #ffff00;
      }
+     .underline {
+       text-decoration: underline;
+     }
    `
  ]
})
export class AppComponent {
  isHighlighted = false;
  onClick() {
    this.isHighlighted = !this.isHighlighted;
  }
}
```

```
@Component({
  selector: 'app-root',
  template: `
    <p (click)="onClick()"
+    [ngClass]="calculateClasses()">
    We need to button up ...
    </p>
  `,
  styles: [
    `
      .highlight {
        background-color: #ffff00;
      }
      .underline {
        text-decoration: underline;
      }
    `
  ]
})
export class AppComponent {
  isHighlighted = false;

  onClick() {
```

```
    this.isHighlighted = !this.isHighlighted;
  }

+  calculateClasses() {
+    return {
+      highlight: this.isHighlighted,
+      underline: true
+    };
+ }
}
```

- ngStyle

```
<p (click)="onClick()" [style.background-color]="'orchid'">
  We need to button up ...
</p>
```

```
<p
  [ngStyle]="{'background-color': 'lime',
    'font-size': '20px',
    'font-weight': 'bold'}"
>
  Here we go ...
</p>
```

```
// app.component.ts

import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
  <form #signinForm="ngForm" (submit)="onSubmit(signinForm)">
      <input
        type="text"
        placeholder="username"
        ngModel
+        #username="ngModel"
        name="username"
+        required
+        minlength="3"
      /><br />
+      <pre>
+        {{ username.errors | json }}
+      </pre
      >
+      <br />
+      Dirty: {{ username.dirty | json }} <br />
+      Touched: {{ username.touched | json }} <br />
      <input type="text" ngModel name="password"
placeholder="password" /><br />
      <button>Sign In</button>
    </form>
  `,
  styles: []
})
export class AppComponent {
  ...
}
```

Displaying Validation Messages

```
// app.component.ts

@Component({
  selector: 'app-root',
  template: `
    <form #signinForm="ngForm" (submit)="onSubmit(signinForm)">
      <input
        type="text"
        placeholder="username"
        ngModel
        #username="ngModel"
        name="username"
        required
        minlength="3"
      /><br />
+      <div *ngIf="username.hasError('required') &&
username.dirty">
+        Username is required.
+      </div>

-      <pre>
-        {{ username.errors | json }}
-      </pre>
-      <br />
-      Dirty: {{ username.dirty | json }} <br />
-      Touched: {{ username.touched | json }} <br />
      <input type="text" ngModel name="password"
placeholder="password" /><br />

      <button>Sign In</button>
    </form>
  `,
  styles: []
})
export class AppComponent {
...
}
```

Services

- Setup

```
git checkout ngFor
```

- Create & Register Service

```
ng g service fruit
```

```
// fruit.service.ts
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class FruitService {
  constructor() {}

  list(): string[] {
    return ['Apple', 'Orange', 'Plum'];
  }
}
```

- Inject Service

```
// app.component.ts
export class AppComponent implements OnInit {
-  fruits = ['Apple', 'Orange', 'Plum'];
+  fruits = [];

+  constructor(private fruitService: FruitService) {}

+  ngOnInit(): void {
+    this.fruits = this.fruitService.list();
+  }
}
```

- With an Observable to handle async

```
// fruit.service.ts
import { Injectable } from '@angular/core';
import { of, Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class FruitService {
  constructor() {}

  list(): Observable<string[]> {
    return of(['Apple', 'Orange', 'Plum']);
  }
}
```

```
// app.component.ts
export class AppComponent implements OnInit {
  fruits = [];

  constructor(private fruitService: FruitService) {}

  ngOnInit(): void {
    this.fruitService.list().subscribe(data => (this.fruits =
data));
  }
}
```

- See Async Happening

```
import { Injectable } from '@angular/core';
import { of, Observable } from 'rxjs';
+ import { delay } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class FruitService {
  constructor() {}

  list(): Observable<string[]> {
    return of(['Apple', 'Orange', 'Plum'])
+     .pipe(delay(4000));
  }
}
```

```
ngOnInit(): void {
    this.fruitService.list().subscribe(data => (this.fruits =
data));
+    console.log('completed OnInit');
  }
```

1. Run the application.
2. Open Chrome Devtools.
3. Because we added a 4 second delay, you will see `completed OnInit` logged before the data loads on the page.

RxJS

Observables

Observable: represents the idea of an invokable collection of future values or events.

```
import { Component, OnInit } from '@angular/core';
+ import { of } from 'rxjs';

@Component({
  selector: 'app-root',
  template: ``,
  styles: []
})
export class AppComponent implements OnInit {
+  ngOnInit(): void {
+    const observable$ = of(1, 2, 3);
+    observable$.subscribe(x => console.log(x));
+  }
}
```

OR

```
export class AppComponent implements OnInit {
+  ngOnInit(): void {
+    of(1, 2, 3).subscribe(x => console.log(x));
+  }
}
```

Result

- Open Chrome DevTools
- Switch to the Console tab
- Refresh the browser

```
1
2
3
```

Creating a Stream of DOM Events

```
import { Component, OnInit } from '@angular/core';
import { of,
+ fromEvent } from 'rxjs';

@Component({
  selector: 'app-root',
  template: `
+    <button>Click Me</button>
  `,
  styles: []
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
+    // don't do direct DOM manipulation in Angular components
+    // we'll see how to avoid this later in these demos
+    const button = document.querySelector('button');
+    const clicks$ = fromEvent(button, 'click');
+    clicks$.subscribe(x ⇒ console.log(x));
  }
}
```

Result

- Open Chrome DevTools
- Switch to the Console tab
- Refresh the browser
- Click the button on the page

```
MouseEvent {isTrusted: true, screenX: 30, screenY: 101, clientX:
30, clientY: 22, …}
MouseEvent {isTrusted: true, screenX: 30, screenY: 101, clientX:
30, clientY: 22, …}
...
```

Listening for keyup events

```
import { Component, OnInit } from '@angular/core';
import { fromEvent } from 'rxjs';

@Component({
  selector: 'app-root',
  template: `
    <input type="text" />
  `,
  styles: []
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    // again, don't do direct DOM manipulation in Angular
components
    // we'll see how to avoid this later in these demos
    const input = document.querySelector('input');
    const keyupEvents$ = fromEvent(input, 'keyup');
    keyupEvents$.subscribe((x: Event) ⇒
      console.log((<HTMLInputElement>x.target).value)
    );
  }
}
```

Result

- Open Chrome DevTools
- Switch to the Console tab
- Refresh the browser
- Enter the letters `abcdef` into the input

```
a
ab
abc
abcd
...
```

Observers

Observer: is a collection of callbacks that knows how to listen to values delivered by the Observable.

```
import { Component, OnInit } from '@angular/core';
import { of, Observer } from 'rxjs';

@Component({
  selector: 'app-root',
  template: ``,
  styles: []
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    const observable$ = of(1, 2, 3);
    const observer: Observer<any> = {
      next: x ⇒ console.log(x),
      complete: () ⇒ console.log('completed'),
      error: x ⇒ console.log(x)
    };
    observable$.subscribe(observer);
  }
}
```

Result

- Open Chrome DevTools
- Switch to the Console tab
- Refresh the browser

```
1
2
3
completed
...
```

Subcriptions

Subscription: represents the execution of an Observable, is primarily useful for cancelling the execution.

```
import { Component, OnInit } from '@angular/core';
import { of, Observer,
+ interval } from 'rxjs';

@Component({
  selector: 'app-root',
  template: ``,
  styles: []
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
-    const observable$ = of(1, 2, 3);
+    // Emits ascending numbers, one every second (1000ms)
+   const observable$ = interval(1000);
    const observer: Observer<any> = {
      next: x ⇒ console.log(x),
      complete: () ⇒ console.log('completed'),
      error: x ⇒ console.log(x)
    };
    const subscription = observable$.subscribe(observer);
+   setTimeout(() ⇒ subscription.unsubscribe(), 5000);
  }
}
```

Result

- Open Chrome DevTools
- Switch to the Console tab
- Refresh the browser

```
0
1
2
3
4
...
```

Operators

Continue from prior demo or...

```
git checkout rxjs-subscriptions
```

map

```
import { Component, OnInit } from '@angular/core';
import { of, Observer, interval } from 'rxjs';
+ import { map } from 'rxjs/operators';

@Component({
  selector: 'app-root',
  template: ``,
  styles: []
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    // Emits ascending numbers, one every second (1000ms)
    const observable$ = interval(1000);
    const observer: Observer<any> = {
      next: x => console.log(x),
      complete: () => console.log('completed'),
      error: x => console.log(x)
    };

+    const observableCommingOutOfThePipe$ = observable$.pipe(
+      map(x => x * 10)
+    );
+    const subscription =
observableCommingOutOfThePipe$.subscribe(observer);

-   const subscription = observable$.subscribe(observer);
-   setTimeout(() => subscription.unsubscribe(), 5000);
  }
}
```

Result

- Open Chrome DevTools
- Switch to the Console tab
- Refresh the browser

```
0
10
20
30
...
```

tap

```
import { map,
+ tap } from 'rxjs/operators';
 ...

    const observableCommingOutOfThePipe$ = observable$.pipe(
+     tap(x => console.log(x)),
      map(x => x * 10)
    );
```

Result

- Open Chrome DevTools
- Switch to the Console tab
- Refresh the browser

```
0
0
1
10
2
20
3
30
 ...
```

filter

```
import { filter } from 'rxjs/operators';
...

const observableCommingOutOfThePipe$ = observable$.pipe(
  filter(x => x % 2 === 0)
);
```

Result

- Open Chrome DevTools
- Switch to the Console tab
- Refresh the browser

```
0
0
1
10
2
20
3
30
...
```

RxJS code commonly uses the fluent syntax and chains functions.

```
ngOnInit(): void {
    // Emits ascending numbers, one every second (1000ms)
    // const observable$ = interval(1000);
    // const observer: Observer<any> = {
    //   next: x => console.log(x),
    //   complete: () => console.log('completed'),
    //   error: x => console.log(x)
    // };

    // const observableCommingOutOfThePipe$ = observable$.pipe(
    //   filter(x => x % 2 === 0)
    // );

    // observableCommingOutOfThePipe$.subscribe(observer);

    interval(1000)
      .pipe(filter(x => x % 2 === 0))
      .subscribe(x => console.log(x));
  }
```

Practical Example

Search Box

```
import { Component, OnInit } from '@angular/core';
import { Subject } from 'rxjs';

@Component({
  selector: 'app-root',
  template: `
    <input
      type="text"
      #term
      (input)="search(term.value)"
      placeholder="search"
    />
    <br />
    <p *ngFor="let message of messages">{{ message }}</p>
  `,
  styles: []
})
export class AppComponent implements OnInit {
  messages: string[] = [];
  private searchTermStream$ = new Subject<string>();

  ngOnInit(): void {
    this.searchTermStream$.subscribe(term ⇒
      this.messages.push(`http call for: ${term}`)
    );
  }

  search(term: string) {
    this.searchTermStream$.next(term);
  }
}
```

Result

- Type `angular` quickly in the searchbox
- Ouput is shown below the input on the page

```
http call for: an

http call for: ang

http call for: ang

http call for: angu

http call for: angular

http call for: angular

http call for: angular
...
```

debounceTime

```
import { Component, OnInit } from '@angular/core';
import { Subject } from 'rxjs';
+ import { debounceTime } from 'rxjs/operators';

@Component({
  selector: 'app-root',
  template: `
    <input
      type="text"
      #term
      (keyup)="search(term.value)"
      placeholder="search"
    />
    <br />
    <p *ngFor="let message of messages">{{ message }}</p>
  `,
  styles: []
})
export class AppComponent implements OnInit {
  messages: string[] = [];
  private searchTermStream$ = new Subject<string>();

  ngOnInit(): void {
    this.searchTermStream$
+     .pipe(debounceTime(300))
      .subscribe(term => this.messages.push(`http call for:
${term}`));
  }

  search(term: string) {
    this.searchTermStream$.next(term);
  }
}
```

Result

- Type `angular` quickly in the searchbox
- Ouput is shown below the input on the page

```
http call for: angular
```

distinctUntilChanged

Try

- Type `angular` quickly in the searchbox
- Delete the last letter `r` then retype the `r`
- Ouput is shown below the input on the page

Result

```
http call for: angular

http call for: angula

http call for: angular
```

```
import { Component, OnInit } from '@angular/core';
import { Subject } from 'rxjs';
import { debounceTime,
+ distinctUntilChanged } from 'rxjs/operators';

@Component({
  selector: 'app-root',
  template: `
    <input
      type="text"
      #term
      (keyup)="search(term.value)"
      placeholder="search"
    />
    <br />
    <p *ngFor="let message of messages">{{ message }}</p>
  `,
  styles: []
})
export class AppComponent implements OnInit {
  messages: string[] = [];
  private searchTermStream$ = new Subject<string>();

  ngOnInit(): void {
    this.searchTermStream$
      .pipe(
+        debounceTime(1000),
+       distinctUntilChanged()
      )
      .subscribe(term ⇒ this.messages.push(`http call for:
${term}`));
  }

  search(term: string) {
    this.searchTermStream$.next(term);
  }
}
```

Try

- Type `angular` quickly in the searchbox
- Delete the last letter `r` then retype the `r`
- Ouput is shown below the input on the page

Result

```
http call for: angular
```

Notice that I increased the debounceTime so that it's easier to retype the letters you removed.

switchMap

Cancels the orginal obervable and returns a new one

```
this.searchTermStream$
      .pipe(
        debounceTime(1000),
        distinctUntilChanged(),
+        switchMap((term: string) => {
+          return of(`new observable: ${term}`);
+        })
      )
      .subscribe(term => this.messages.push(` ${term}`));
```

Http

Get

- Checkout start branch

```
git checkout http-start
```

Which has db.json, api script, and json-server installed

- Create model

```
ng g class photo
```

Rename `photo.ts` to `photo.model.ts`

```
export class Photo {
  id: number;
  title: string;
  url: string;
  thumbnailUrl: string;
}
```

- Create service

```
ng g service photo
```

- Import HttpClientModule

```
// app.module.ts
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, AppRoutingModule,
+    HttpClientModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

- Inject HttpClient service
- Implement getAll method

```
import { Injectable } from '@angular/core';
+ import { HttpClient } from '@angular/common/http';
+ import { Photo } from './photo.model';
+ import { Observable } from 'rxjs';

@Injectable({
+  providedIn: 'root'
})
export class PhotoService {
+  constructor(private http: HttpClient) {}

+  getAll(): Observable<Photo[]> {
+    return this.http.get<Photo[]>
('http://localhost:3000/photos');
+  }
}
```

```
// app.component.ts
import { Component, OnInit } from '@angular/core';
import { PhotoService } from './photo.service';

@Component({
  selector: 'app-root',
  template: `
    <h1>Photos</h1>
    <div *ngFor="let photo of photos">
      <img [src]="photo.thumbnailUrl" alt="" />
      <p>{{ photo.title }}</p>
    </div>
  `,
  styles: []
})
export class AppComponent implements OnInit {
  pphotos: Photo[];
  constructor(private photoService: PhotoService) {}
  ngOnInit(): void {
    this.photoService.getAll().subscribe(data => (this.photos =
data));
  }
}
```

Error Handling

```
// photo.service.ts

import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from
'@angular/common/http';
import { Photo } from './photo.model';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class PhotoService {
  constructor(private http: HttpClient) {}

  getAll(): Observable<Photo[]> {
    return this.http.get<Photo[]>
('http://localhost:3000/photos/wrong').pipe(
      catchError((error: HttpErrorResponse) => {
        console.log(error);
        return throwError('An error occured loading the
photos.');
      })
    );
  }
}
```

Retry

```
git checkout http-retry -f
```

```
// see photo.service.ts
```

Routing

Routing Basics

- Start Server

```
ng serve -o
```

- Generate Components

```
ng g component home
ng g component about
ng g component contact
```

- Add Routes
  - Snippets
    - a-route-path-eager
    - a-route-path-default

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'home' },
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
];
```

- Add Navigation and the `router-outlet`

```
<nav>
  <a [routerLink]="['/home']">Home</a> |
  <a [routerLink]="['/about']">About</a> |
  <a [routerLink]="['/contact']">Contact</a>
</nav>
<router-outlet></router-outlet>
```

- Highlight Active Navigation Item

```
<nav>
  <a routerLinkActive="active" [routerLink]="['/home']">Home</a>
|
  <a routerLinkActive="active" [routerLink]="
['/about']">About</a> |
  <a routerLinkActive="active" [routerLink]="
['/contact']">Contact</a>
</nav>
<router-outlet></router-outlet>
```

Routing Navigation

- Start with routing-navigation-start branch as shown below.
- Merge the service branch.

```
git checkout routing-navigation-start
```

Code Review

All the steps in this next section have already been completed so just review/walkthrough the existing code before moving to the next section.

- Create a movies module.

```
ng g module movies --routing --module=app
```

- Create a movie model class.

```
ng g class movies/shared/movie
```

1. Rename file `movie.ts` to `movie.model.ts`.
2. Add properties to the movie model.

```
export class Movie {
  constructor(
    public id: number,
    public name: string,
    public description: string
```

```
    ) {}
}
```

- Create mock movie data.
  1. Create file `movies/shared/mock-movies.ts`
  2. Add these movies.

```
import { Movie } from './movie.model';

export const MOVIES: Movie[] = [
  new Movie(
    1,
    ' Titanic',
    'A seventeen-year-old aristocrat falls in love with a kind
but poor artist aboard the luxurious, ill-fated R.M.S. Titanic.'
  ),
  new Movie(
    2,
    ' E.T. the Extra-Terrestrial',
    'A troubled child summons the courage to help a friendly
alien escape Earth and return to his home world.'
  ),
  new Movie(
    3,
    'The Wizard of Oz',
    'Dorothy Gale is swept away from a farm in Kansas to a
magical land of Oz in a tornado and embarks on a quest with her
new friends to see the Wizard who can help her return home in
Kansas and help her friends as well.'
  ),
  new Movie(
    4,
    'Star Wars: Episode IV - A New Hope ',
    'Luke Skywalker joins forces with a Jedi Knight, a cocky
pilot, a Wookiee and two droids to save the galaxy from the
Empire/`s world-destroying battle-station while also attempting
to rescue Princess Leia from the evil Darth Vader.'
  )
];
```

- Create a movie service.

```
ng g service movies/shared/movie
```

- Add find and list methods and bring in the needed imports.

movies\shared\movie.service.ts

```
import { of } from 'rxjs';
import { Observable } from 'rxjs';
import { Injectable } from '@angular/core';

@Injectable()
export class MovieService {
  list(): Observable<Movie[]> {
    return of(MOVIES);
  }

  find(id: number): Observable<Movie> {
    const movie = MOVIES.find(m => m.id === id);
    return of(movie);
  }
}
```

- Also review the movie components code for the list and detail.
    - list
    - detail

These components were generated with the following commands:

```
ng g component movies/movie-list
```

```
ng g component movies/movie-detail
```

Routing Changes

The steps from here are changes that should be made to the code.

```
//movies/movie-list/movie-list.component.ts

import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-movie-list',
  template: `
  <div>
  <ul>
      <li *ngFor="let movie of movies" >
          <a
+          [routerLink]="['detail', movie.id]">
          {{movie.name}}
          </a>
      </li>
  </ul>
  </div>
  `,
  styles: []
})
export class MovieListComponent implements OnInit {
  movies: Movie[];

  constructor(private movieService: MovieService) {}

  ngOnInit() {
    this.movieService.list().subscribe(data => (this.movies =
data));
```

```
    }
}
```

- detail

```
//movies/movie-detail/movie-detail.component.ts
...
+ import { MovieService } from '../shared/movie.service';
+ import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-movie-detail',
  template: `
    <div *ngIf="movie">
      <h5>{{ movie.name }}</h5>
      <p>{{ movie.description }}</p>
    </div>
  `,
  styles: []
})
export class MovieDetailComponent implements OnInit {
  movie: Movie;

  constructor(
+    private movieService: MovieService,
+    private route: ActivatedRoute
  ) {}

+  ngOnInit() {
+    this.route.params.subscribe(p => {
+      const id = +p['id'];
+      this.movieService.find(id).subscribe(m => (this.movie =
m));
+    });
  }
}
```

- Configure routes.

```
//movies/movies-routing.module.ts
const routes: Routes = [
  { path: 'movies',component: MovieListComponent},
+ { path: 'movies/detail/:id', component: MovieDetailComponent }
];
```

- Add Navigation Item

```
<nav>
<a routerLinkActive="active" [routerLink]="
['/home']">Home</a> |
<a routerLinkActive="active" [routerLink]="
['/about']">About</a> |
<a routerLinkActive="active" [routerLink]="
['/contact']">Contact</a> |
+  <a routerLinkActive="active" [routerLink]="
['/movies']">Movies</a>
</nav>
```

Routing (Advanced)

Child Routes

Continue from the previous code (routing-navigation).

- Configure child routes.

```
// movies/movies-routing.module.ts
const routes: Routes = [
  {
    path: 'movies',
    component: MovieListComponent
+  children: [{ path: 'detail/:id', component:
MovieDetailComponent }]
  },
-  { path: 'movies/detail/:id', component: MovieDetailComponent }
];
```

- Add an outlet

```
// movies/movie-list/movie-list.component.ts
@Component({
  selector: 'app-movie-list',
  template: `
  <div>
  <ul>
      <li *ngFor="let movie of movies" >
          <a [routerLink]="['detail', movie.id]">{{movie.name}}
</a>
      </li>
  </ul>
  </div>
+  <div>
+    <router-outlet></router-outlet>
+  </div>
  `,
  styles: []
})
export class MovieListComponent implements OnInit {
...
```

Lazy Loading

- Configure Movies Module to lazy load.

> It can be useful to look at the size of `main.js` in the Chrome DevTools Network tab and record it on the board before doing this demonstration. Be sure to clear the browser cache so you get the correct file size. You can then repeat the process at the end of the lab and show that the file size is about half of the original size.

Continue from the code from the previous step (routing-navigation).

> Be sure to remove the child route and go back to a traditional route as shown in the step below or other code would need to be adjusted to demonstrate lazy-loading.

```
// movies/movies-routing.module.ts
const routes: Routes = [
  {
-    path: 'movies'
+    path: '',
    component: MovieListComponent,
+    children: [{ path: 'detail/:id', component:
MovieDetailComponent }]
  }
-  // { path: 'movies/detail/:id', component:
MovieDetailComponent }
];
```

```
// app.module.ts

  imports: [
    BrowserModule,
    AppRoutingModule,
-   MoviesModule
  ],
```

```
// app-routing.module.ts
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'home' },
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
+  {
```

```
+    path: 'movies',
+    loadChildren: '../app/movies/movies.module#MoviesModule'
+  }
];
```

Components (Advanced)

Change Detection

Checkout

```
git checkout change-detection -f
```

The following components are already created.

```
parent
child-a
child-b
grandchild-a
```

ChangeDetectionStrategy.Default

Click on each of the buttons to see what components are checked by change detection. Refreshing the page after each button click makes this easier to see.

- If an input "changes" (clicking the parent button which sets the nickname) then the entire tree is checked.
- If an event triggered then the entire tree is checked.

ChangeDetectionStrategy.OnPush

Go into each of the following child components.

- child-a
- child-b
- grandchild-a

Uncomment the line to modify the change detection strategy from:

- the default aptly named `ChangeDetectionStrategy.Default`
- to `ChangeDetectionStrategy.OnPush`

Click on each of the buttons to see what components are checked by change detection. Refreshing the page after each button click makes this easier to see.

- If a component has a changed input or raises an event, causes check on itself (component) and all ancestors
  - If an input "changes" then checks itself (component with input) and all ancestors
  - If an event happens then checks itself (component where event was raised, Ex. click) and all ancestors

Appendixes

Optional demos if class requests.

Redux (NgRx)

Documentation available at: https://ngrx.io/

Setup

```
git checkout start -f
```

Installation

Install @ngrx/schematics from npm:

```
npm install @ngrx/schematics --save-dev
```

> NgRx Schematics helps you avoid writing common boilerplate and instead focus on building your application

After installing @ngrx/schematics, install the NgRx dependencies.

```
npm install @ngrx/{store,effects,entity,store-devtools} --save
```

Store

Generate the initial state management and register it within the app.module.ts

```
ng generate @ngrx/schematics:store State --root --module
app.module.ts
```

> By adding the StoreModule.forRoot function in the imports array of your AppModule. The StoreModule.forRoot() method registers the global providers needed to access the Store throughout your application.

Actions

Generate a new file named `counter.actions.ts`

```
ng generate @ngrx/schematics:action Counter --flat
```

> The @ngrx/schematics command prefix is only needed when the default collection isn't set.

Describe the counter actions to increment, decrement, and reset its value.

```
// src/app/counter.actions.ts
// delete the generated code and replace with the code below

import { Action } from '@ngrx/store';

export enum ActionTypes {
  Increment = '[Counter Component] Increment',
  Decrement = '[Counter Component] Decrement',
  Reset = '[Counter Component] Reset'
}

export class Increment implements Action {
  readonly type = ActionTypes.Increment;
}

export class Decrement implements Action {
  readonly type = ActionTypes.Decrement;
}

export class Reset implements Action {
  readonly type = ActionTypes.Reset;
}
```

Reducer

Generate a reducer.

```
ng generate @ngrx/schematics:reducer Counter --flat --spec=false
```

Define a reducer function to handle changes in the counter value based on the provided actions.

```
// src/app/counter.reducer.ts
// delete the generated code and replace with the code below

import { Action } from '@ngrx/store';
import { ActionTypes } from './counter.actions';

export const initialState = 0;

export function counterReducer(state = initialState, action:
Action) {
  switch (action.type) {
    case ActionTypes.Increment:
      return state + 1;

    case ActionTypes.Decrement:
      return state - 1;

    case ActionTypes.Reset:
      return 0;

    default:
      return state;
  }
}
```

Add the count to the state interface and the reducers object and set the counterReducer to manage the state of the counter.

```
// src/app/reducers/index.ts

import {
  ActionReducer,
  ActionReducerMap,
  createFeatureSelector,
  createSelector,
```

```
  MetaReducer
} from '@ngrx/store';
import { environment } from '../../environments/environment';
+ import { counterReducer } from '../counter.reducer';

export interface State {
+  count: number;
}

export const reducers: ActionReducerMap<State> = {
+   count: counterReducer
};

export const metaReducers: MetaReducer<State>[] =
!environment.production
  ? []
  : [];
```

Component

```
ng generate component my-counter --spec=false
```

Update the `MyCounterComponent` class with a selector for the `count`, and methods to dispatch the Increment, Decrement, and Reset actions.

Then, update the `MyCounterComponent` template with buttons to call the increment, decrement, and reset methods. Use the async pipe to subscribe to the count$ observable.

```
// src/app/my-counter/my-counter.component.ts

import { Component } from '@angular/core';
+ import { Store, select } from '@ngrx/store';
+ import { Observable } from 'rxjs';
+ import { Increment, Decrement, Reset } from
'../counter.actions';

@Component({
  selector: 'app-my-counter',
   template: `
+   <div>Current Count: {{ count$ | async }}</div>
+
+   <button (click)="increment()">Increment</button>
+
+   <button (click)="decrement()">Decrement</button>
+
+   <button (click)="reset()">Reset Counter</button>
  `,
  styleUrls: ['./my-counter.component.css'],
})
export class MyCounterComponent {
+  count$: Observable<number>;
+  ngOnInit(): void {}
+  constructor(private store: Store<{ count: number }>) {
+    this.count$ = store.pipe(select('count'));
+  }

+  increment() {
+    this.store.dispatch(new Increment());
+  }

+  decrement() {
+    this.store.dispatch(new Decrement());
+  }
```

```
+  reset() {
+    this.store.dispatch(new Reset());
+  }
}
```

Add the `MyCounter` component to your `AppComponent` template. Delete the default generated html content.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
+    <app-my-counter></app-my-counter>
  `,
  styles: []
})
export class AppComponent {}
```

If not already running start the application with the following command

```
ng serve -o
```

Open `Chrome DevTools` and demonstrate the time traveling, record replay, and logging features of the `Redux DevTools` extension.

> Directions on installing this extension included as part of the setup document for the class.

Template-driven Forms

Template Forms Binding

```
// app.module.ts
+ import { FormsModule } from '@angular/forms';

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

```
// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <form #signinForm="ngForm" (submit)="onSubmit(signinForm)">
      <input type="text" ngModel name="username"
placeholder="username" /><br />
      <input type="text" ngModel name="password"
placeholder="password" /><br />
      <button>Sign In</button>
    </form>
    <pre>
      {{ signinForm.value | json }}
    </pre
    >
  `,
  styles: []
})
export class AppComponent {
  onSubmit(form) {
    console.log(form.value);
  }
}
```

Template Forms Validation

```
// app.component.ts

import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
  <form #signinForm="ngForm" (submit)="onSubmit(signinForm)">
      <input
        type="text"
        placeholder="username"
        ngModel
+        #username="ngModel"
        name="username"
+        required
+        minlength="3"
      /><br />
+      <pre>
+        {{ username.errors | json }}
+      </pre
      >
+      <br />
+      Dirty: {{ username.dirty | json }} <br />
+      Touched: {{ username.touched | json }} <br />
      <input type="text" ngModel name="password"
placeholder="password" /><br />
      <button>Sign In</button>
    </form>
  `,
  styles: []
})
export class AppComponent {
  ...
}
```

Displaying Validation Messages

```
// app.component.ts

@Component({
  selector: 'app-root',
  template: `
    <form #signinForm="ngForm" (submit)="onSubmit(signinForm)">
      <input
        type="text"
        placeholder="username"
        ngModel
        #username="ngModel"
        name="username"
        required
        minlength="3"
      /><br />
+       <div *ngIf="username.hasError('required') &&
username.dirty">
+         Username is required.
+       </div>

-       <pre>
-         {{ username.errors | json }}
-       </pre>
-       <br />
-       Dirty: {{ username.dirty | json }} <br />
-       Touched: {{ username.touched | json }} <br />
      <input type="text" ngModel name="password"
placeholder="password" /><br />

      <button>Sign In</button>
    </form>
  `,
  styles: []
})
export class AppComponent {
...
}
```

Two-way Binding

```
@Component({
  selector: 'app-root',
  template: `
    <input
      [value]="message"
      (input)="message = $event.target.value"
      type="text"
    />
    <p>{{ message }}</p>
  `,
  styles: []
})
export class AppComponent {
  message = '';
}
```

```
import { FormsModule } from '@angular/forms';

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

```
@Component({
  selector: 'app-root',
  template: `
    <input [(ngModel)]="message" type="text" />
    <p>{{ message }}</p>
  `,
  styles: []
})
export class AppComponent {
  message = '';
}
```

Bootstrapping an Application

```
import './polyfills';

// app/app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: '<h1>Hello Angular</h1>'
})
export class AppComponent {}

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

// app/app.module.ts
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  bootstrap: [AppComponent]
})
export class AppModule {}

// app/main.ts
import { platformBrowserDynamic } from '@angular/platform-
browser-dynamic';
platformBrowserDynamic().bootstrapModule(AppModule);
```

Setup

Creating this project

Not done in class just checkout start branch.

```
ng new demos --routing --inline-style --inline-template --skip-
tests --skip-install
```

> Notice the flags --inline-style --inline-template so separate html and css files are not generated for each component.

1. Open `package.json` and remove `devDependencies` related to testing.
2. Delete e2e folder.
3. npm install

Creating the http-start branch

Not done in class just checkout http-start branch.

```
git checkout start
npm install json-server
mkdir api
```

Copy db.json from here: https://jsonplaceholder.typicode.com/db into the api folder.

Add script to `package.json`

```
"scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
+   "api": "json-server ./api/db.json"
  },
```

Feedback

I appreciate all feedback. All pdf manuals are protected but are able to be annotated. Send feedback to craig@funnyant.com