

Introduction to Angular

Lab Manual



Copyright © 2020-2022

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.

Introduction to Angular	1
About this Lab Manual	4
Lab 1: Creating a New Project	7
Lab 2: Running Your Project	11
Lab 3: Styles: Using a CSS Framework	14
Lab 4: Your First Component	18
Lab 5: Creating Data Structures	22
Lab 6: Passing Data into a Component	29
Lab 7: Looping Over Data	32
Lab 8: Formatting Data for Display	34
Lab 9: More Reusable Components	36
Lab 10: Responding to an Event	40
Lab 11: Create a Form to Edit Your Data	44
Lab 12: Communicating from Child to Parent Component	48
Lab 13: Hiding and Showing Components	53
Lab 14: Preventing a Page Refresh	55
Lab 15: More Component Communication	58
Lab 16: Forms I Binding	63
Lab 17: Forms I Saving	70
Lab 18: Forms I Validation	75
Lab 19: Forms I Refactor	79
Lab 20: Services & Dependency Injection	82
Lab 21: Setup Backend REST API	86
Lab 22: HTTP GET	89
Lab 23: HTTP Error Handling	92
Lab 24: HTTP PUT	96

Lab 25: Showing a Loading Indicator	100
Lab 26: Router Navigation	104
Lab 27: Route Parameters	112
Lab 28: Custom Pipe	118
Lab 29: Build & Deploy	122
Appendix A: How to Skip Labs	128
Appendix B: REST Review	131

About this Lab Manual

This lab manual provides a series of hands-on exercises for learning how to build web applications using Angular.

Conventions

Each hands-on exercise in this manual will consist of a series of steps to accomplish a learning objective.

Code Blocks

- All paths are relative to the **project-manage** directory.

So the file below will be found at:

AngularCourse\code\labs\working\project-manage\src\app.module.ts

- **Highlighted code** indicates code that has changed. If the code is not highlighted it should already exist from a previous step.
- Code with a **Strikethrough** should be removed.
- **...** Indicates code has been omitted for formatting and clarity but you should leave these sections of code in your running application.

src\app.module.ts

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

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

snippets\lab00-step00.txt

- Most code snippets are short and easy to type but some are longer so a file with the contents of the code to add is provided in the folder.

AngularCourse\code\labs\snippets

- If a code snippets is provided for a code block the file path will appear below the code block as show above.

Commands

These commands should be run in a command-prompt (Windows) or terminal (Mac).

```
ng -v
```

Sidebars

The boxes are sidebars and should be read.

The boxes with blue borders are information and tips.

The boxes with red borders are alerts.

Completion

At the end of each lab you will see:

- ✓ You have completed Lab ...

Lab 1: Creating a New Project

Objectives

- Verify the Angular CLI is installed
- Create a new Angular project
- Open the new project
- Review the default project structure

Steps

Verify the Angular CLI is installed

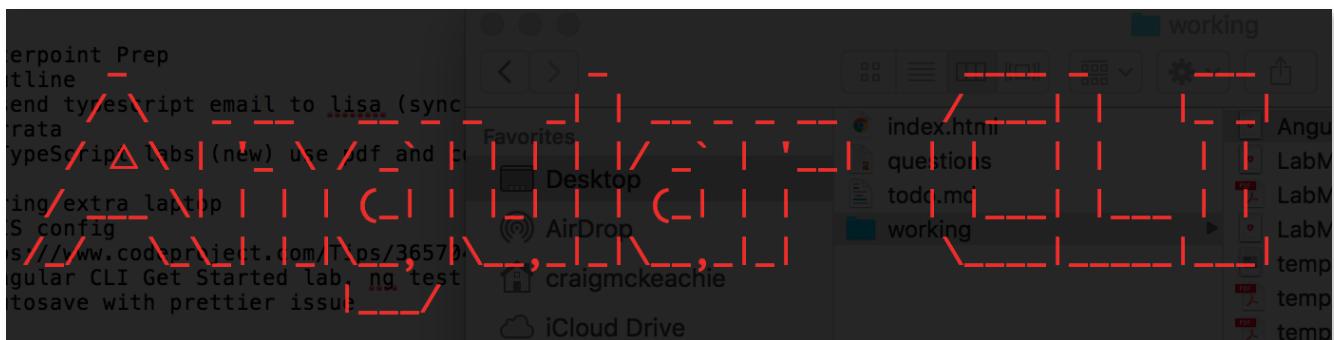
1. Open a command prompt (Windows) or terminal (Mac).

You can be in *any* directory when you run the command because the Angular CLI is installed globally.

2. Run the command.

3. Verify the output.

```
ng v
```



Create a new Angular project

4. Change the current directory to **AngularCourse\code\labs\working**.

5. **Run** the command.

```
ng new project-manage
```

6. You will receive the following prompt. **Type y** to answer yes.

7. You will receive another prompt. Hit **enter** to accept the default of CSS.

```
? Would you like to add Angular routing? (y/N)
```

8. A new Angular project will be created for you.

```
? Which stylesheet format would you like to use? (Use arrow keys)
```

```
> CSS
```

```
SCSS [ http://sass-lang.com ]
```

```
SASS [ http://sass-lang.com ]
```

```
LESS [ http://lesscss.org ]
```

```
Stylus [ http://stylus-lang.com ]
```

This could take a several minutes and requires an internet connection to install Angular and the other libraries from **npmjs.com**.

Adding Angular routings tells the Angular CLI to create a routing module where we can configure our routes.

Choosing CSS tells the CLI we want don't want to use a preprocessor for our styles.

Open the new project

9. Change the current directory (cd) to the directory you created in the last step.

```
cd project-manage
```

10. Open the **project-manage** directory in your **editor** of choice.

If you are using Visual Studio Code you can run following command:

```
code .
```

...**code** refers to Visual Studio Code and **.** means current directory.

Review the default project structure

11. Take a few minutes to go over the **default project structure** with your instructor. Below are some things to discuss.
- a. Open **package.json** and review the **dependencies** (JavaScript libraries) installed as well as the **scripts**.
 - b. Understand each of the files involved in **bootstrapping** (starting) an Angular application.
 1. app.component.html | app.component.ts
 2. index.html
 3. app.module.ts
 4. main.ts

✓ You have completed Lab 1

Lab 2: Running Your Project

Objectives

- Run the project
 - Make a change and see the app update
-

Steps

Run the project

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.

```
ng serve --open
```

The flag **--open** automatically opens your default web browser with the application running in it.

2. **Run** the command.
3. The project will:
 - build and bundle the code
 - open a development web server
 - open your default web browser

4. The page should display an Angular logo and the text shown below.

Welcome to project-manage!

In earlier versions of the Angular CLI (before 7.5.x), you need to uncomment some lines in the `polyfills.ts` file to allow the page to render in **Internet Explorer 11** but this is no longer required. See this article to understand what steps were required in previous versions.

<https://blog.angularindepth.com/angular-and-internet-explorer-5e59bb6fb4e9>

For more information about browser support see:

<https://angular.io/guide/browser-support>

Make a change and see the app update

5. Open and edit the file:

```
src\app\app.component.ts
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'project-manage';
  title = 'awesome';
}
```

6. Save your changes.
7. The browser should automatically reload and display this at the top of the page.

```
awesome app is running!
```

✓ You have completed Lab 2

IE 11 or lower:

Since **differential builds** are **enabled by default** in the Angular CLI to get the project to display in **IE 11 or lower** you would need to turn off differential builds by following the steps in **Lab 29: Step 19 a & b**.

Lab 3: Styles: Using a CSS Framework

Objectives

- Install a CSS framework
- Stop and restart the build and web server
- Verify styles are working in app

Steps

Install a CSS framework

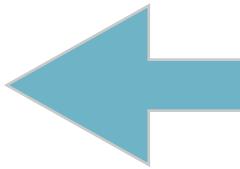
1. Open a new (*leave ng serve running*) command prompt (Windows) or terminal (Mac). Set the directory to project-manage.
2. Run the command.

```
npm install mini.css@3.0.0
```

- The JavaScript package manager npm automatically adds mini.css as a dependency.

package.json

```
"dependencies": {  
...  
  "core-js": "^2.5.4",  
  "mini.css": "^3.0.0",  
  "rxjs": "^6.0.0",  
  "zone.js": "^0.8.26"  
},  
...  
}
```



Mini.css is a **minimal**, responsive, style-agnostic **CSS framework**. Mini.css is similar to Bootstrap but lighter and requires fewer CSS classes so you can focus on learning Angular but still get a professional look.

3. Include the framework's **stylesheet** in the Angular CLI's configuration.

angular.json

```
"projects": {  
  "project-manage": {  
    "root": "",  
    "sourceRoot": "src",  
    "projectType": "application",  
    "prefix": "app",  
    "schematics": {},  
    "architect": {  
      "build": {  
        "builder": "@angular-devkit/build-angular:browser",  
        "options": {  
          "outputPath": "dist/project-manage",  
          "index": "src/index.html",  
          "main": "src/main.ts",  
          "polyfills": "src/polyfills.ts",  
          "tsConfig": "src/tsconfig.app.json",  
          "assets": [  
            "src/favicon.ico",  
            "src/assets"  
          ],  
          "styles": [  
            "node_modules/mini.css/dist/mini-default.min.css",  
            "src/styles.css"  
          ],  
          "scripts": []  
        },  
        ...  
      },  
      ...  
    },  
    ...  
  },  
  ...  
}
```

In WebStorm files with a *.min.css extension are hidden under the un-minified version of the file.

Stop and restart the build and web server

4. Focus your cursor in the **command prompt** (Windows) or **terminal** (Mac). and use **[Ctrl+C]** to stop the build and web server.

Windows users will be prompted if it is OK to terminate the process and should answer **[y+enter]**.

5. Run the command.

```
ng serve --open
```

Your current directory should still be set to **project-manage** or the above command will not work.

6. The application will **build** and **open a browser**.

Verify styles are working in app

7. Open the file **app\app.component.html**.
8. Delete all **contents** from the file.
9. Add the following quote.

```
src\app\app.component.html
```

```
<blockquote cite="Benjamin Franklin">
    Tell me and I forget, teach me and I may remember, involve me and
    I learn.
</blockquote>
```

```
snippets\lab03-step09.html
```

10. Save your changes.
11. The browser should automatically reload and display the quote with the CSS styles shown below.

“

Tell me and I forget, teach me and I may remember, involve me and I learn.

— Benjamin Franklin

- ✓ You have completed Lab 3

Lab 4: Your First Component

Objectives

- Create a Feature Module
- Create a Component

Steps

Create a Feature Module

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
2. **Run** the command.

```
ng generate module projects --routing --module=app
```

The **--routing** flag tells the Angular CLI to generate a module to hold our project related routes (**projects\projects-routing.module.ts**). We will use this module later in the course when we cover routing so you can safely ignore it for now.

The **--module** flag tells the Angular CLI to import the feature module for projects (**projects\projects.module.ts**) to the root module as shown in the next step.

3. Review the **root module (AppModule)** and note that the **feature module (ProjectsModule)** has been **imported** into it because of the `--module` flag used in the previous step.

```
src\app\app.module.ts
```

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

Create a Component

4. Run the command.

```
ng g component projects/projects-container
```

The Angular CLI (`ng`) `g` command is short for generate and by default will create files under the `src\app` directory. If you generate with a path prefix as we did in the example above (`projects / ...`), the CLI will create files in that location and create the directories if they don't already exist.

5. Because you generated the component with a **projects/** path prefix in the previous step the component will automatically be added to the declarations of the **ProjectsModule**. In order to use the component in the **AppModule** you need to list it in the **exports** of the **ProjectsModule**.

```
src\app\projects\projects.module.ts
```

```
...
@NgModule({
  imports: [CommonModule, ProjectsRoutingModule],
  declarations: [ProjectsContainerComponent], [ProjectsContainerComponent]
  exports: [ProjectsContainerComponent]
})
export class ProjectsModule {}
```

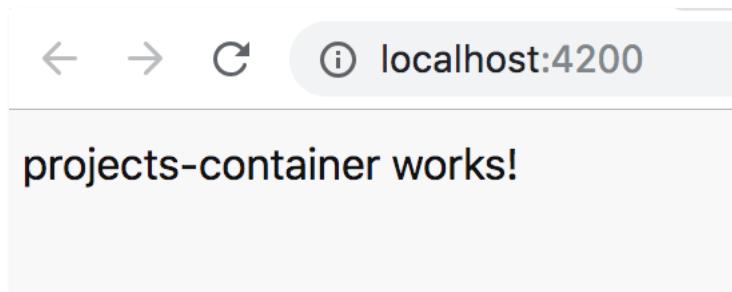
6. Make the following changes so Angular knows where to render the component.

```
src\app\app.component.html
```

```
<blockquote cite="Benjamin Franklin">
  Tell me and I forget, teach me and I may remember, involve me and
  I learn.
</blockquote>

<app-projects-container></app-projects-container>
```

7. **Save** your changes to the code.
8. Your **browser** should automatically **reload** and display the component as shown below.



✓ You have completed Lab 4

Lab 5: Creating Data Structures

Objectives

- Create a Model or Entity Object
 - Add hard-coded mock data
 - Display the data
-

Overview

Create a model or entity object to use as a data structure in your application.

Steps

Create a Model or Entity Object

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
2. **Run** the command.

```
ng g class projects/shared/project --skipTests=true --type=model
```

3. Open the file and add a constructor.

src\app\projects\shared\project.model.ts

```
export class Project {  
  constructor(  
    public id: number,  
    public name: string,  
    public description: string,  
    public imageUrl: string,  
    public contractTypeId: number,  
    public contractSignedOn: Date,  
    public budget: number,  
    public isActive: boolean,  
    public editing: boolean  
  ) {}  
  
}
```

snippets\lab05-step04.txt

Add hard-coded mock data

4. Close your editor and **copy** all images in this directory:

- **snippets\Lab5\assets**

into the \code\ labs\working\project-manage\src\assets directory

You may need to close your editor or
stop ng serve to copy the files.

The **assets** directory has images we will use throughout the course.

5. Create the **file** mock-projects in the directory shown below and copy the mock data from the snippet file. Be sure to **copy** the mock data from the **snippets** directory as only one object is shown below.

src\app\projects\shared\mock-projects.ts

```
import { Project } from './project.model';
export const MOCK_PROJECTS: Project[] = [
  new Project(
    1,
    'Scarlet Weeknight',
    'Fusce quis quam eget sapien sodales iaculis. Curabitur aliquet at erat sed cursus. In hendrerit.',
    'assets/placeimg_500_300_arch7.jpg',
    5,
    new Date(2015, 1, 2),
    30100,
    true,
    false
  ),
  ...
]
```

snippets\lab05-step05.txt

Display the data

6. Create a **projects** property, strongly type it as an **array of projects**, and assign the imported **mock project data**.

```
src\app\projects\projects-container\projects-container.component.ts
```

```
import { Component, OnInit } from '@angular/core';
import { MOCK_PROJECTS } from '../shared/mock-projects';
import { Project } from '../shared/project.model';

@Component({
  selector: 'app-projects-container',
  templateUrl: './projects-container.component.html',
  styleUrls: ['./projects-container.component.css']
})
export class ProjectsContainerComponent implements OnInit {
  projects: Project[] = MOCK_PROJECTS;
  constructor() {}

  ngOnInit() {}
}
```

7. Delete the current contents of the template. Display the array of projects data in the template.

```
<h1>Projects</h1>
{{projects}}
```

- #### 8. Verify the output.

Projects

[object Object],[object Object],[object Object],[object Object],[object Object],[object Object],[object Object]

9. Format the data using a pipe to serialize the array as a string.

```
<h1>Projects</h1>
{{projects | json}}
```

- #### 10. Verify the output.

Projects

11. Wrap the formatted JSON data in a <pre> tag to preserve the whitespace.

```
src\app\projects\projects-container\projects-container.component.html
```

```
<h1>Projects</h1>
<pre>
{{projects | json}}
</pre>
```

12. Verify the output.

Projects

```
[  
 {  
   "id": 1,  
   "name": "Scarlet Weeknight",  
   "description": "Fusce quis quam eget sapien sodales iaculis.",  
   "imageUrl": "assets/placeimg_500_300_arch7.jpg",  
   "contractTypeId": 5,  
   "contractSignedOn": "2015-02-02T05:00:00.000Z",  
   "budget": 30100,  
   "isActive": true,  
   "editing": false  
,  
 {  
   "id": 2,  
   "name": "Tipus",  
 }
```

Sending output to a json pipe and wrapping it in an HTML <pre> tag is a useful debugging tip.

- ✓ You have completed Lab 5

Lab 6: Passing Data into a Component

Objectives

- Create a presentation component
 - Pass data into the presentation component
-

Steps

Use property binding and the @Input decorator to pass data into a component.

Create a presentation component

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
2. **Run the following command to generate a presentation component.**

```
ng g component projects/project-list
```

3. Add an **input property** to the new component.

```
src\app\projects\project-list\project-list.component.ts

import { Component, OnInit, Input } from '@angular/core';
import { Project } from '../shared/project.model';

@Component({
  selector: 'app-project-list',
  templateUrl: './project-list.component.html',
  styleUrls: ['./project-list.component.css']
})
export class ProjectListComponent implements OnInit {
  @Input()
  projects: Project[] = [];
  constructor() {}

  ngOnInit() {}
}
```

4. Remove the generated HTML and display the **input property data** in the template.

```
src\app\projects\project-list\project-list.component.html

<p>
  project-list works!
</p>
<pre>
  {{projects | json}}
</pre>
```

Pass data into the presentation component

5. Open the parent container component **template** file and use the new presentation component's **selector** to display the data.

```
src\app\projects\projects-container\projects-container.component.html
```

```
<h1>Projects</h1>
<pre>
{{projects | json}}
</pre>
<app-project-list [projects]="projects"></app-project-list>
```

6. Verify the result is the same as the previous lab.

Projects

```
[  
 {  
   "id": 1,  
   "name": "Scarlet Weeknight",  
   "description": "Fusce quis quam eget sapien sodales  
   "imageUrl": "assets/placeimg_500_300_arch7.jpg",  
   "contractTypeId": 5.  
 }]
```

Although the results are the same we are beginning to break our UI into encapsulated, re-usable pieces.

- ✓ You have completed Lab 6

Lab 7: Looping Over Data

Objectives

- Loop over data

Steps

Loop over data

1. Open the following template and **loop** through the array of data **using** an **ngFor** directive. Use **interpolation** and **property binding** to display the data.

```
src\app\projects\project-list\project-list.component.html
```

```
<pre>
  {{projects | json}}
</pre>


<div class="cols-sm" *ngFor="let project of projects">
    <div class="card">
      <img [src]=" project.imageUrl " [alt]="project.name">
      <section class="section dark">
        <h5 class="strong">
          <strong>{{project.name}}</strong>
        </h5>
        <p>{{project.description}}</p>
        <p>Budget : {{project.budget}}</p>
      </section>
    </div>
  </div>


```

```
snippets\lab07-step01.html
```

2. Verify the result.

Projects

**Scarlet Weeknight**

Fusce quis quam eget sapien sodales iaculis. Curabitur aliquet at erat sed cursus. In hendrerit.

Budget : 30100

**Tipus**

Aliquam rhoncus, libero eget feugiat rutrum, tortor sem posuere elit, scelerisque eleifend mi mi sit amet.

Budget : 52378

**Kaylux**

Pellentesque eu mauris vel sem laoreet blandit ac ac erat. Morbi lorem justo, commodo at faucibus vitae, consequat.

Budget : 72500

**Dusty Epsilon**

Nunc cursus purus malesuada, dignissim augue ac, ullamcorper turpis. Nullam efficitur odio id nulla eleifend.

Budget : 42400



Not all boxes will be the same height at this point in the labs. We will fix this in a later lab.

✓ You have completed Lab 7

Lab 8: Formatting Data for Display

Objectives

- Format data using Angular's built-in currency pipe

Steps

Format data using Angular's built-in currency pipe

1. Open the following template and format the budget amount as currency.

```
src\app\projects\project-list\project-list.component.html
```

```
<div class="row">
  <div class="cols-sm" *ngFor="let project of projects">
    <div class="card">
      <img [src]=" project.imageUrl " [alt]=" project.name ">
      <section class="section dark">
        <h5 class="strong">
          <strong>{{project.name}}</strong>
        </h5>
        <p>{{project.description}}</p>
        <p>Budget : {{project.budget}}</p>
        <p>Budget : {{project.budget | currency : 'USD': 'symbol': '0.0-2'}}</p>
      </section>
    </div>
  </div>
</div>
```

2. Verify the result.

Projects



Scarlet Weeknight

Fusce quis quam eget sapien sodales iaculis. Curabitur aliquet at erat sed cursus. In hendrerit.

Budget : \$30,100

Tipus

Aliquam rhoncus, libero rutrum, tortor sem pos scelerisque eleifend mi

Budget : \$52,378

- ✓ You have completed Lab 8

Lab 9: More Reusable Components

Objectives

- Create a presentation component for each project
 - Pass a project into the presentation component
-

Steps

Create a presentation component for each project

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
2. **Run** the following **command** to generate a **presentation component**.

```
ng g component projects/project-card
```

3. Add an **input property** to the new component.

```
src\app\projects\project-card\project-card.component.ts
```

```
import { Component, OnInit, Input } from '@angular/core';
import { Project } from '../shared/project.model';

@Component({
  selector: 'app-project-card',
  templateUrl: './project-card.component.html',
  styleUrls: ['./project-card.component.css']
})
export class ProjectCardComponent implements OnInit {
  @Input()
  project: Project;
  constructor() {}

  ngOnInit() {}
}
```

4. Cut the HTML from the list template and paste it into the card template.

```
src\app\projects\project-list\project-list.component.html
```

```
<div class="row">
  <div class="cols-sm" *ngFor="let project of projects">
    <div class="card">
      <img [src]="project.imageUrl" [alt]="project.name">
      <section class="section dark">
        <h5 class="strong">
          <strong>{{project.name}}</strong>
        </h5>
        <p>{{project.description}}</p>
        <p>Budget : {{project.budget | currency : 'USD': 'symbol': '0.0-2'}}</p>
      </section>
    </div>
  </div>
</div>
```

```
src\app\projects\project-card\project-card.component.html
```

```
<p>
  —project-card works!
</p>

<div class="card">
  <img [src]="project.imageUrl" [alt]="project.name">
  <section class="section dark">
    <h5 class="strong">
      <strong>{{project.name}}</strong>
    </h5>
    <p>{{project.description}}</p>
    <p>Budget : {{project.budget | currency : 'USD': 'symbol': '0.0-2'}}</p>
  </section>
</div>
```

Pass a project into the presentation component

5. Open the parent list component template file and use the new card presentation component to display the data.

```
src\app\projects\projects-list\projects-list.component.html
```

```
<div class="row">
  <div class="cols-sm" *ngFor="let project of projects">
    <app-project-card [project]="project"></app-project-card>
  </div>
</div>
```

6. Verify the result is the same as the previous lab.

Projects



Scarlet Weeknight

Fusce quis quam eget sapien sodales iaculis. Curabitur aliquet at erat sed cursus. In hendrerit.

Budget : \$30,100



Tipus

Aliquam rhoncus, libero rutrum, tortor sem pos scelerisque eleifend mi

Budget : \$52,378

- ✓ You have completed Lab 9

Lab 10: Responding to an Event

Objectives

- Use event binding to respond to a user event

Steps

Use event binding to respond to a user event

```
src\app\projects\project-card\project-card.component.ts
```

```
...
export class ProjectCardComponent implements OnInit {
  @Input()
  project: Project;
  constructor() {}

  ngOnInit() {}

  onEditClick(project: Project, event: Event) {
    event.preventDefault();
    console.log(project);
  }
}
```

We will explain `event.preventDefault()` in a future lab. We actually don't need to call it yet but we will do it in a later lab so we are adding it now to prepare.

1. Create a **method** on a component **to handle** an event.
2. Add a **button** and use event binding to **wire it up** to the **event handler** method you created in the last step.

```
src\app\projects\project-card\project-card.component.html
```

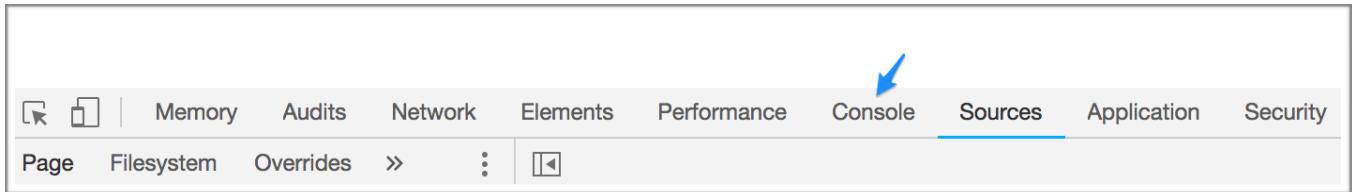
```
<div class="card">
  <img [src]=" project.imageUrl " [alt]=" project.name " >
  <section class="section dark">
    <h5 class="strong">
      <strong>{{project.name}}</strong>
    </h5>
    <p>{{project.description}}</p>
    <p>
      Budget :
      {{project.budget | currency : 'USD': 'symbol': '0.0-2'}}</p>
    <button class=" bordered" (click)="onEditClick(project, $event)">
      <span class="icon-edit "></span>
      Edit
    </button>
  </section>
</div>
```

```
snippets\lab10-step02.html
```

3. Verify the code is working by following these steps.
 - a. **Save** the file.
 - b. The browser will **automatically reload** the application.
 - c. In your browser, **open** the **Chrome DevTools** by hitting **F12**

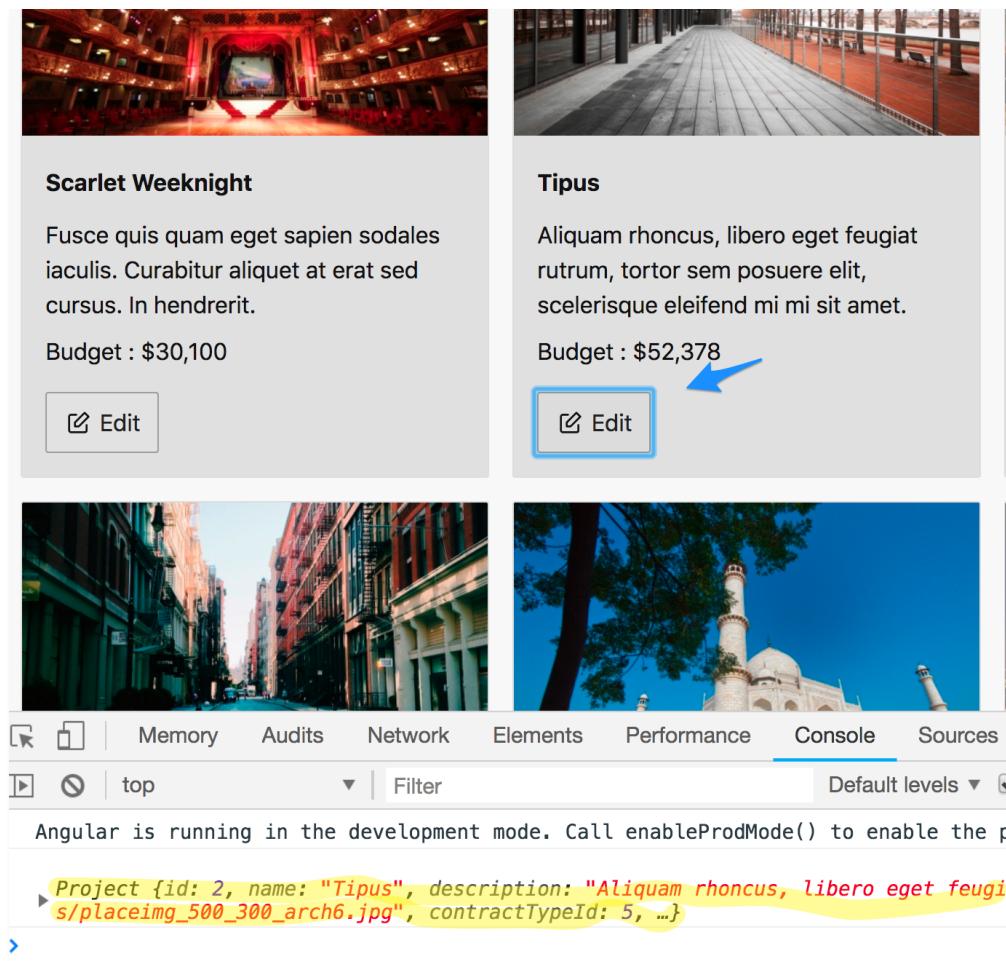
On a laptop you may need to hold down your function key while hitting F12 [fn+F12].

- d. Switch the **Chrome DevTools** to the **Console** tab by clicking on it.



- e. **Click on the edit button** for one of the projects.

- f. Verify the project object is logged to the DevTools console.



✓ You have completed Lab 10

Lab 11: Create a Form to Edit Your Data

Objectives

- Create a form component
 - Render a form component
 - Style a form component
-

Steps

Create a form component

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
2. **Run** the following **command** to **generate** a **form** component.

```
ng g c projects/project-form
```

The **c** in the command **ng g c** is short for component and can be used to save some typing.

3. Add the markup provided below to render an HTML form.

```
src\app\projects\project-form\project-form.component.html
```

```
<form class="input-group vertical">

  <label for="name">Project Name</label>
  <input type="text" name="name" placeholder="enter name">

  <label for="description">Project Description</label>
  <textarea type="text" name="description"
    placeholder="enter description">
  </textarea>

  <label for="budget">Project Budget</label>
  <input type="number" name="budget" placeholder="enter budget">

  <label for="isActive">Active?</label>
  <input type="checkbox" name="isActive">

  <div class="input-group">
    <button class="primary bordered medium">Save</button>
    <span></span>
    <a href="">cancel</a>
  </div>

</form>
```

```
snippets\lab11-step03.html
```

Render a form component

4. Add the form component **selector** to the list component.

```
src\app\projects\project-list\project-list.component.html
```

```
<div class="row">
  <div class="cols-sm" *ngFor="let project of projects">
    <app-project-card [project]="project"></app-project-card>
    <app-project-form></app-project-form>
  </div>
</div>
```

Style a form component

5. Add a component style to set the minimum width of the form.

```
src\app\projects\project-form\project-form.component.css
```

```
form {
  min-width: 300px;
}
```

At this point, there is no change to the layout as the result of the min-width style being but we are adding it here because we will need it later.

6. Verify the form displays as shown below.

The image shows two side-by-side screenshots of a web-based project management application. Each screenshot displays a project card with a thumbnail image at the top, followed by the project name, a brief description, a budget amount, an edit button, and a detailed view section below.

Project Card (Left): Scarlet Weeknight

- Thumbnail:** An ornate theater interior with red curtains and balcony seating.
- Name:** Scarlet Weeknight
- Description:** Fusce quis quam eget sapien sodales iaculis. Curabitur aliquet at erat sed cursus. In hendrerit.
- Budget:** \$30,100
- Edit:**

Project Card (Right): Tipus

- Thumbnail:** A modern building with large glass windows and a paved walkway.
- Name:** Tipus
- Description:** Aliquam rhoncus, libero eget feugiat rutrum, tortor sem posuere elit, scelerisque eleifend mi mi sit amet.
- Budget:** \$52,378
- Edit:**

Detailed View Section (Left):

- Project Name:**
- Project Description:**
- Project Budget:**
- Active?**
- Buttons:** Save | cancel

Detailed View Section (Right):

- Project Name:**
- Project Description:**
- Project Budget:**
- Active?**
- Buttons:** Save | cancel

✓ You have completed Lab 11

Lab 12: Communicating from Child to Parent Component

Objectives

- Create a custom events in the child
- Listen for the custom event in the parent

Steps

Create a custom event in the child

```
src\app\projects\project-card\project-card.component.ts

import { Component, OnInit, Input,
          Output, EventEmitter } from '@angular/core';
...
export class ProjectCardComponent implements OnInit {
  @Input()
  project: Project;
  @Output()
  edit = new EventEmitter<any>();

  constructor() {}
  ngOnInit() {}

  onEditClick(project: Project, event: Event) {
    event.preventDefault();
    console.log(project);
    this.edit.emit({ editingProject: project });
  }
}
```

In the last step make sure that **EventEmitter** is coming from the **correct import path** as shown in the code snippet.

1. **Create custom events**, make them available on the tag, and **emit** the **event**.

Note that **no code changes** are **needed** for this step, it is just a review so you can follow the flow of events in the component hierarchy.

```
src\app\projects\project-card\project-card.component.html
```

```
...
<button class=" bordered" (click)="onEditClick(project, $event)">
  <span class="icon-edit"></span>
  Edit
</button>
...
```

2. Open the template to **review** how **onEditClick** is being triggered...with the click of the edit button in the card.

Listen for the custom event in the parent

3. Edit the **parent component** and **create an event handler** that assigns the project being edited into a property and logs which project is being edited.

```
src\app\projects\project-list\project-list.component.ts
```

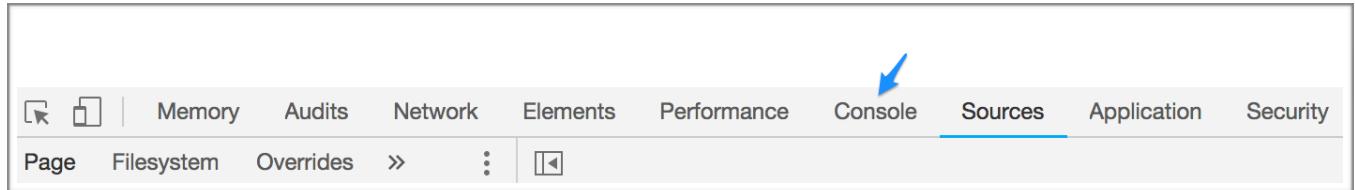
```
export class ProjectListComponent implements OnInit {  
  @Input()  
  projects: Project[] = [];  
  editingProject: Project;  
  
  onEdit(event: any) {  
    this.editingProject = event.editingProject;  
    console.log(this.editingProject);  
  }  
  
  constructor() {}  
  
  ngOnInit() {}  
}
```

4. **Subscribe your event handler to the custom event.** Note that \$event will be the **custom event object** you emitted in the child component.

```
src\app\projects\project-list\project-list.component.html
```

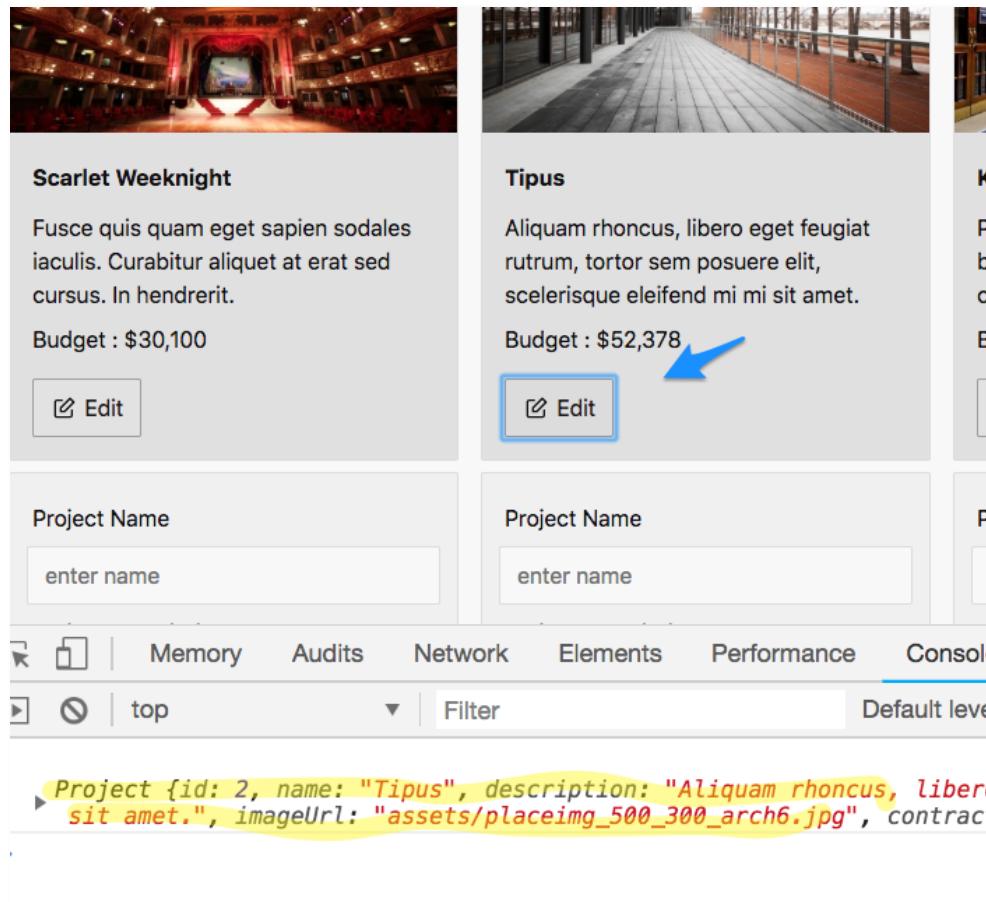
```
<div class="row">  
  <div class="cols-sm" *ngFor="let project of projects">  
    <app-project-card [project]="project" (edit)="onEdit($event)">  
    </app-project-card>  
    <app-project-form></app-project-form>  
  </div>  
</div>
```

5. Verify the code is working by following these steps.
 - a. **Save** the file.
 - b. The browser will **automatically reload** the application.
 - c. In your browser, **open** the **Chrome DevTools** by hitting **F12**.
 - d. Switch the **Chrome DevTools** to the **Console** tab by clicking on it.



- e. **Click on the edit button** for one of the projects.

- f. Verify the project object is still being **logged** to the DevTools **console**.



You may remember that logging was happening in a previous lab. In the previous lab, the logging was occurring in the child component. In this lab, we have removed that logging and are raising an event back up to the parent list component. This will allow the card component to be easily reused in another part of the application.

- ✓ You have completed Lab 12

Lab 13: Hiding and Showing Components

Objectives

- Hide and show a component using **ngIf**

Steps

Now that current project being edited is being set into the **editingProject** property we can use an **ngIf** directive in the template to **hide and show** the card and form when appropriate.

Hide and show a component using **ngIf**

1. Show and hide the **form** when **edit** is clicked.

```
src\app\projects\project-list\project-list.component.html
```

```
<div class="row">
  <div class="cols-sm" *ngFor="let project of projects">
    <app-project-card [project]="project" (edit)="onEdit($event)">
      *ngIf="project !== editingProject">
        </app-project-card>
        <app-project-form
          *ngIf="project === editingProject">
            </app-project-form>
      </div>
    </div>
```

2. Verify the form is hiding and showing by:
 - a. Save your changes.
 - b. Clicking on the various edit buttons on the page.

At this point, clicking save will not do anything (we'll implement this in a later lab). Also, clicking cancel actually refreshes the entire page which we do not want to happen so we will fix that in the next lab.

✓ You have completed Lab 13

Lab 14: Preventing a Page Refresh

Objectives

- Prevent the default web browser behavior; attempting to load a page

Overview

Click cancel and notice that the entire page reloads which should not happen in a single-page application (SPA). In this lab we will learn how to prevent this default behavior.

Steps

Prevent the default web browser behavior; attempting to load a page

```
src\app\projects\project-form\project-form.component.html
```

```
<form class="input-group vertical">  
...  
<div class="input-group">  
  <button class="primary bordered medium">Save</button>  
  <span></span>  
  <a href="#" (click)="onCancelClick($event)">cancel</a>  
</div>  
</form>
```

Since we are handling the above event with an event handler method that we don't create until the next step your editor will underline it and give you the message **Unknown method onCancelClick**.

1. Subscribe an event handler to the cancel link's click event.

2. Implement the event handler method.

```
src\app\projects\project-form\project-form.component.ts
```

```
...
export class ProjectFormComponent implements OnInit {
  constructor() {}
  ngOnInit() {}

  onCancelClick(event: Event) {
    event.preventDefault();
  }
}
```

3. Verify the code is working following these steps:

- a. Click the edit button for a project.
- b. On the form that displays click the cancel link.
- c. Prior to us preventing the default browser behavior of loading a page when a link is clicked, this caused a reload of the entire page. Now clicking cancel will do nothing (but no longer reload the page).

Note that the form will not be removed because we haven't told the parent list that the child has cancelled editing. We will do this in the next lab.

✓ You have completed Lab 14

Lab 15: More Component Communication

Objectives

- Create a custom event in the child
 - Listen for the custom event in the parent
-

Overview

In this lab, you the child form component will emit a custom event to the parent list component. This event will notify the list there is no longer a project being edited.

This lab is very similar to the previous component communication lab so consider it an optional lab to do only if time permits.

You will need to follow the directions on how to skip a lab before continuing to the next lab to maintain continuity in the labs.

Steps

Create a custom event in the child

1. Create a custom cancel event and emit it.

```
src\app\projects\project-form\project-form.component.ts
```

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

@Component({
  selector: 'app-project-form',
  templateUrl: './project-form.component.html',
  styleUrls: ['./project-form.component.css']
})
export class ProjectFormComponent implements OnInit {
  @Output()
  cancel = new EventEmitter<void>();

  constructor() {}
  ngOnInit() {}

  onCancelClick(event: Event) {
    event.preventDefault();
    this.cancel.emit();
  }
}
```

Listen for the custom event in the parent

2. **Subscribe** to the custom event in the parent list with an event handler.

```
src\app\projects\project-list\project-list.component.html
```

```
<div class="row">
  <div class="cols-sm" *ngFor="let project of projects">
    <app-project-card [project]="project" (edit)="onEdit($event)"
      *ngIf="project !== editingProject">
      </app-project-card>
    <app-project-form *ngIf="project === editingProject"
      (cancel)="onCancel()">
      </app-project-form>
    </div>
  </div>
```

The invocation of the onCancel() method will have a red line under it with the error “[Angular] Unknown method ‘onCancel’” if you have the Angular Language Service extension that comes as part of the Angular Essentials Extension for Visual Studio Code . It is safe to ignore this message as we will create the onCancel method in the next step but it is good to know Angular can alert you to these errors in the its templates.

3. Implement the event handler method to take the project out of edit mode.

```
src\app\projects\project-list\project-list.component.ts
```

```
export class ProjectListComponent implements OnInit {
...
  editingProject: Project;
...
  onCancel() {
    this.editingProject = null;
  }
}
```

4. Verify the code is working.

- Save** all your files.
- Click** the **edit** button on a project and the **form** **should show** in place of the card.

Projects



Scarlet Weeknight

Fusce quis quam eget sapien sodales iaculis. Curabitur aliquet at erat sed cursus. In hendrerit.

Budget : \$30,100

Edit



Tipus

Aliquam rhoncus, libero eget feugiat rutrum, tortor sem posuere elit, scelerisque eleifend mi mi sit amet.

Budget : \$52,378

Edit

Project Name	<input type="text" value="enter name"/>
Project Description	<input type="text" value="enter description"/>
Project Budget	<input type="text" value="enter budget"/>
Active?	<input type="checkbox"/>
<input type="button" value="Save"/> <input type="button" value="cancel"/>	



Dusty Epsilo

Nunc cursus dignissim aug Nullam efficit

Budget : \$42

Edit



- c. Click the cancel link and the form should be removed and be replaced again by the card.

Projects



Scarlet Weeknight

Fusce quis quam eget sapien sodales iaculis. Curabitur aliquet at erat sed cursus. In hendrerit.

Budget : \$30,100

[Edit](#)



Tipus

Aliquam rhoncus, libero eget feugiat rutrum, tortor sem posuere elit, scelerisque eleifend mi mi sit amet.

Budget : \$52,378

[Edit](#)



Kaylux

Pellentesque eu mauris vel sem laoreet blandit ac ac erat. Morbi lorem justo, commodo at faucibus vitae, consequat.

Budget : \$72,500

[Edit](#)



Dusty Epsilon

Nunc cursus dignissim au Nullam effici

Budget : \$42,000

[Edit](#)

✓ You have completed Lab 15

Lab 16: Forms I Binding

Objectives

- Create a reactive binding between HTML elements and FormControl objects
- Observe the reactive binding

Steps

Create a reactive binding between HTML elements and FormControl objects

1. Import the **ReactiveFormsModule**.

```
src\app\projects\projects.module.ts
```

```
...
```

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

```
@NgModule({
  imports: [
    CommonModule,
    ProjectsRoutingModule,
    ReactiveFormsModule
  ],
  declarations: [
    ProjectsContainerComponent,
    ProjectListComponent,
    ProjectCardComponent,
    ProjectFormComponent]
})
export class ProjectsModule { }
```

Check to make sure your **import** of the **ReactiveFormsModule** in the last step is coming from the **correct path**. Some editors automatically import this from a longer incorrect path.

2. Create the **FormGroup** and **FormControl** objects and initialize them to the values in the project passed in to the control via the **project input property**.

src\app\projects\project-form\project-form.component.ts

```
import { Component, OnInit, Output, EventEmitter, Input } from
'@angular/core';
import { Project } from '../shared/project.model';
import { FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'app-project-form',
  templateUrl: './project-form.component.html',
  styleUrls: ['./project-form.component.css']
})
export class ProjectFormComponent implements OnInit {
  @Input()
  project: Project;
  @Output()
  cancel = new EventEmitter<void>();
  projectForm: FormGroup;

  constructor() {}
  ngOnInit() {
    this.projectForm = new FormGroup({
      name: new FormControl(this.project.name),
      description: new FormControl(this.project.description),
      budget: new FormControl(this.project.budget),
      isActive: new FormControl(this.project.isActive)
    });
  }
  ...
}
```

snippets\lab16-step02.txt

3. Update the list control **template** to **set the project** (created in the last step) into the input property **using property binding syntax**.

```
src\app\projects\project-list\project-list.component.html
```

```
<div class="row">
  <div class="cols-sm" *ngFor="let project of projects">
    <app-project-card [project]="project" (edit)="onEdit($event)"
      *ngIf="project !== editingProject">
      </app-project-card>
      <app-project-form [project]="project"
        *ngIf="project === editingProject" (cancel)="onCancel()">
        </app-project-form>
    </div>
  </div>
```

4. Annotate your form with Angular's **form directives** and **output the values**.

```
src\app\projects\project-form\project-form.component.html
```

```
<form [formGroup]="projectForm" class="input-group vertical">

  <label for="name">Project Name</label>
  <input type="text" name="name" placeholder="enter name"
    formControlName="name">

  <label for="description">Project Description</label>
  <textarea type="text" name="description"
    placeholder="enter description"
    formControlName="description"></textarea>

  <label for="budget">Project Budget</label>
  <input type="number" name="budget" placeholder="enter budget"
    formControlName="budget">

  <label for="isActive">Active?</label>
  <input type="checkbox" name="isActive"
    formControlName="isActive">

  <div class="input-group">
    <button class="primary bordered medium">Save</button>
    <span></span>
    <a href="" (click)="onCancelClick($event)">cancel</a>
  </div>

  <pre style="width: 300px">
    {{projectForm.value | json}}
  </pre>

</form>
```

Observe the reactive binding

Sending the form values to a json pipe in the previous step allows us to see the two-way binding going on between the HTML elements and the FormControl objects in the next step.

5. **Observe the binding** created by following these steps:
 - a. **Save** the changes to your code and your browser will reload.
 - b. **Click the edit button** for a project.
 - c. **Change** any of the **form elements** and **see the changes** reflected in the FormGroup object's **values** shown below the form.

The screenshot shows a form for editing a project. The fields are:

- Project Name:** Dusty Epsilon Updated
- Project Description:** Nunc cursus purus malesuada, dignissim augue ac, ullamcorper turpis. Nullam
- Project Budget:** 42400
- Active?**: checked

At the bottom, there are **Save** and **cancel** buttons. Below the buttons, a JSON representation of the form values is displayed:

```
{  
  "name": "Dusty Epsilon Updated",  
  "description": "Nunc cursus purus ma",  
  "budget": 42400,  
  "isActive": true  
}
```

At this point clicking the **Save** button will **not** be working yet. You will only be able to see your changes in the values shown below the form.

6. Remove the **pre** tag that displays the FormGroup's values before continuing.

```
src\app\projects\projects-form\project-form.component.html
```

```
<form [formGroup]="projectForm" class="input-group vertical">
...
<pre style="width: 300px">
{{projectForm.value | json}}
</pre>

</form>
```

✓ You have completed Lab 16

Lab 17: Forms I Saving

Objectives

- Save the form values
-

Steps

Save the form values

There are many steps involved in communicating the updated form values from the form component up through the list component before they finally reach the container (smart) component. The architecture pattern of having one container / smart component that does the heavy lifting of talking to a backend REST API and having other presentation (dumb) components that just take inputs and emit events is common in JavaScript applications that use a component based architecture. This pattern can be found in Angular, React, and Vue.js applications.

The main advantage you will experience from architecting your applications this way is that your presentation components will be easier to reuse in other parts of your application. Consider that our form component could be used in an update scenario as we have here but also easily reused to add a new item.

The steps begin on the next page.

1. Emit a custom event in the form component.

src\app\projects\project-form\project-form.component.ts

```
export class ProjectFormComponent implements OnInit {  
  @Input()  
  project: Project;  
  @Output()  
  save = new EventEmitter<any>();  
  @Output()  
  cancel = new EventEmitter<void>();  
  ...  
  projectForm: FormGroup;  
  ...  
  onSubmit() {  
    if (this.projectForm.invalid) {  
      return;  
    }  
    const updatedProject = Object.assign(  
      {},  
      this.project,  
      this.projectForm.value  
    );  
    this.save.emit({ project: updatedProject });  
  }  
}
```

Be sure you add an @Output decorator to **both** the save and cancel properties.

snippets\lab17-step01.txt

src\app\projects\project-form\project-form.component.html

```
<form [formGroup]="projectForm" class="input-group vertical"  
  (submit)="onSubmit()">  
  ...  
</form>
```

2. **Subscribe** to the custom event in the list component and emit a new custom event.

src\app\projects\project-list\project-list.component.ts

```
import { Component, OnInit, Input,
         Output, EventEmitter } from '@angular/core';
...
export class ProjectListComponent implements OnInit {
...
  @Output()
  saveListItem = new EventEmitter<any>();
...
  onSave(event: any) {
    this.editingProject = null;
    this.saveListItem.emit({ item: event.project });
  }
...
}
```

src\app\projects\project-list\project-list.component.html

```
...
<app-project-form [project]="project"
  *ngIf="project === editingProject"
  (cancel)="onCancel()"
  (save)="onSave($event)">
</app-project-form>
...
```

3. **Subscribe** to that custom event in the container (smart) component and update the item in the project array.

```
src\app\projects\projects-container\projects-container.component.ts
```

```
...
export class ProjectsContainerComponent implements OnInit {
  projects: Project[] = MOCK_PROJECTS;

  constructor() {}
  ngOnInit() {}

  onSaveListItem(event: any) {
    const project: Project = event.item;
    const index = this.projects.findIndex(
      element => element.id === project.id
    );
    this.projects[index] = project;
  }
}
```

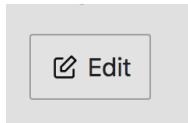
```
snippets\lab17-step03.txt
```

```
src\app\projects\projects-container\projects-container.component.html
```

```
<h1>Projects</h1>
<app-project-list [projects]="projects"
  (saveListItem)="onSaveListItem($event)">
</app-project-list>
```

4. Verify the code is working by following these steps.

a. Click the edit button for a project.



b. Change the project name in the form.

A screenshot of a project edit form. It includes fields for Project Name (containing "Dusty Epsilon Updated"), Project Description (containing placeholder text), Project Budget (containing "42400"), and Active? (with a checked checkbox). At the bottom are "Save" and "cancel" buttons.

c. Click save on the form.

d. Verify the card shows the updated data.



Note that if you refresh your browser page your changes will not persist because the updates are only happening in the browsers memory. We will get this working in a future lab when we communicate to our backend web API.

✓ You have completed Lab 17

Lab 18: Forms I Validation

Objectives

- Add form validation

Steps

Add form validation

1. Add validation functions to your controls.

```
src\app\projects\project-form\project-form.component.ts
```

```
import { FormGroup, FormControl,
         Validators } from '@angular/forms';
...
export class ProjectFormComponent implements OnInit {
...
  ngOnInit() {
    this.projectForm = new FormGroup({
      name: new FormControl(this.project.name, [
        Validators.required,
        Validators.minLength(3)
      ]),
      description: new FormControl(this.project.description),
      budget: new FormControl(this.project.budget),
      isActive: new FormControl(this.project.isActive)
    });
  }
...
}
```

2. Display the validation messages.

src\app\projects\project-form\project-form.component.html

```
<form [formGroup]="projectForm" class="input-group vertical" (submit)="onSubmit()">

  <label for="name">Project Name</label>
  <input type="text" name="name" placeholder="enter name" formControlName="name">

  <div *ngIf="projectForm.get('name').hasError('required')" class="card error">
    <p>Name is required.</p>
  </div>

  <div *ngIf="projectForm.get('name').hasError('minlength')" class="card error">
    <p>Name must be at least 3 characters long.</p>
  </div>

  ...

```

Note that it **minlength** is lowercase in the template above but the function in the TypeScript file uses *camelCase* (**minLength**).

3. Style the validation messages.

src\styles.css

```
/* You can add global styles to this file, and also import other style files */
input.ng-invalid {
  border-color: var(--input-invalid-color);
  box-shadow: none;
}

label.invalid {
  color: var(--input-invalid-color);
}
```

snippets\lab18-step03.css

4. Dynamically add the invalid CSS class to the label so it is styled as well.

src\app\projects\project-form\project-form.component.html

```
<form [formGroup]="projectForm" class="input-group vertical" (submit)="onSubmit()">

  <label for="name" [class.invalid]="projectForm.get('name').invalid">
    Project Name
  </label>

  <input type="text" name="name" placeholder="enter name" formControlName="name">
  ...

```

5. Verify the code is working by following these steps:
 - a. Click the edit button on any project
 - b. Delete the contents of the project name textbox.
 - c. The error message should display immediately and the control label will turn red.
 - d. Cause the input field to lose focus by tabbing out of it or clicking on another input field.
 - e. You should see a red border around the invalid control.

If you don't lose focus on the input field you will not see the red border because the styles applied on focus are overriding the invalid style. This is reasonable behavior but the user experience does not seem ideal. The real problem is that the validation message and red label are showing too early while the user is still working. We will fix this in the next lab.

The screenshot shows a modal dialog for editing a project. The 'Project Name' field is empty and has a red border, with a red validation message 'Name is required.' displayed below it. The 'Project Description' field contains placeholder text 'turpis. Nullam efficitur odio id nulla eleifend.' The 'Project Budget' field contains the value '42400'. The 'Active?' checkbox is checked. At the bottom are 'Save' and 'cancel' buttons.

✓ You have completed Lab 18

Lab 19: Forms I Refactor

Objectives

- Refactor the forms validation code so it is reusable
-

Steps

Refactor the forms validation code so it is reusable

1. Copy the following directory which contains a reusable component to display validation errors. We will use this component to refactor our forms validation code so it more reusable.

```
src\app\shared\validation-errors
```

Copy the snippet directory shown below to the location shown above.

```
snippets\lab19-FormsRefactor\validation-errors
```

2. Import the **ValidationErrorsComponent** into the **ProjectsModule**.

```
src\app\projects\projects.module.ts
```

```
import { ValidationErrorsComponent } from './shared/validation-
errors/validation-errors.component';
```

```
@NgModule({
  declarations: [
    ...
    ValidationErrorsComponent
  ]
})
export class ProjectsModule {}
```

3. Replace the current error messages with the new control. Remove the [class.invalid] property binding from the control's label as well.

src\app\projects\project-form\project-form.component.html

```
<form [formGroup]="projectForm" class="input-group vertical" (submit)="onSubmit()">

  <label for="name" [class.invalid]="projectForm.get('name').invalid">
    Project Name
  </label>
  <input type="text" name="name" placeholder="enter name" formControlName="name">

  <app-validation-errors [control]="projectForm.get('name')">
  </app-validation-errors>

  <div *ngIf="projectForm.get('name').hasError('required')" class="card-error">
    <p>Name is required.</p>
  </div>

  <div *ngIf="projectForm.get('name').hasError('minlength')" class="card-error">
    <p>Name must be at least 3 characters long.</p>
  </div>
  ...

```

4. Verify the code is working by following these steps:
 - a. Click the edit button on any project
 - b. Delete the contents of the project name text box.
 - c. Cause the input field to lose focus by tabbing out of it or clicking on another input field.
 - d. You should see the validation message as well as a red border around the invalid control.

The screenshot shows a modal dialog for editing a project. The 'Project Name' field is empty and has a red border, with a red validation message 'Name is required.' above it. The 'Project Description' field contains placeholder text 'turpis. Nullam efficitur odio id nulla eleifend.' which is highlighted with a blue border, indicating it is the currently focused field. Below the description is a 'Project Budget' field containing '42400'. Underneath the budget is a 'Active?' checkbox with a checked status. At the bottom of the dialog are two buttons: a blue 'Save' button and a white 'cancel' button.

Project Name

enter name

Name is required.

Project Description

turpis. Nullam efficitur odio id nulla eleifend.

Project Budget

42400

Active?

Save cancel

✓ You have completed Lab 19

Lab 20: Services & Dependency Injection

Objectives

- Create your first service
 - Inject the service into a component
-

Steps

Create your first service

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
2. **Run** the following **command** to generate a **form** component.

```
ng g service projects/shared/project
```

3. Implement a method to list all products.

```
src\app\projects\shared\project.service.ts
```

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';
import { Project } from './project.model';
import { MOCK_PROJECTS } from './mock-projects';

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

  constructor() { }

  list(): Observable<Project[]> {
    return of(MOCK_PROJECTS);
  }
}
```

The **of** function is part of the **rxjs** library and is a **creation operator** meaning it creates an **Observable**. In this case, the Observable will return the projects.

Inject the service into a component

4. Inject the **service** into the container component and **use it to access the project data**. Be sure to *remove the assignment of the projects property to the hard-coded array of mock data (MOCK_PROJECTS)*.

```
src\app\projects-container\projects-container.component.ts
```

```
...
```

```
import { ProjectService } from '../shared/project.service';
```

```
...
```

```
export class ProjectsContainerComponent implements OnInit {
  projects: Project[] = MOCK_PROJECTS;
```

```
constructor(private projectService: ProjectService) {}
```

```
ngOnInit() {
```

```
  this.projectService.list().subscribe(data => {
```

```
    this.projects = data;
```

```
  });
}
```

```
}
```

```
...
```

```
}
```

5. Verify the code is working.

- a. Save the files and the browser will automatically reload.
- b. As in previous labs, the list of projects will appear.

Check your command prompt or terminal where `ng serve` is running to ensure you don't have any compiler errors. If you receive the error: **Property 'list' does not exist on type 'ProjectService'** you will need to stop `ng serve` **Ctrl+C** and restart the command `ng serve -o` to resolve the issue.

Although there are no visible changes to the application you have moved your data access to a reusable service so it can be shared with and used by other components.

✓ You have completed Lab 20

Lab 21: Setup Backend REST API

Objectives

- Install the backend REST API server
 - Create a custom npm script to run the REST API server
 - Start the REST API server
-

Steps

Install the backend REST API server

1. **Open** another **command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.

You should already have another command prompt or terminal open in the project-manage directory that is running your Angular application using a development web server. *Leave this server running* and start another command prompt or terminal to run the web server to serve your backend REST API.

2. **Copy** the api directory from:

- **code\labs\snippets\Lab21-REST-API\api**

into code\ labs\ working\ project-manage

3. Run the command.

```
npm install json-server@0.14.0
```

Create a custom npm script to run the REST API server

4. Add a **script** to start the **backend** REST API.

```
package.json
```

```
{  
  "name": "project-manage",  
  "version": "0.0.0",  
  "scripts": {  
    "ng": "ng",  
    "start": "ng serve",  
    "build": "ng build",  
    "test": "ng test",  
    "lint": "ng lint",  
    "e2e": "ng e2e",  
    "api": "json-server ./api/db.json"  
  },  
  ...  
}
```

Start the REST API server

5. In a **command prompt** (Windows) or **terminal** (Mac) with the current directory set to **project-manage**.
6. Run the npm script.

```
npm run api
```

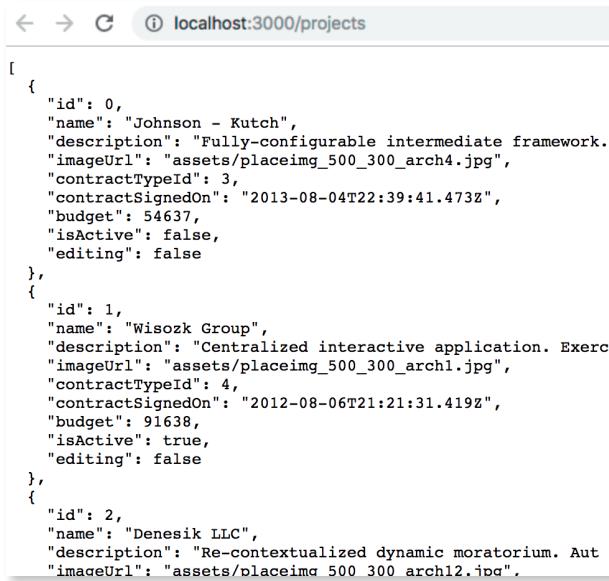
The **run** command is short for **run-script**. Running the backend json-server through an npm script ensures that we are using the local version of the server we just installed and not a previously installed global version.

7. The server should start and output similar to the following should display.

```
\{^_^\}/ hi!  
  
Loading ./api/db.json  
Done  
  
Resources  
http://localhost:3000/projects  
  
Home  
http://localhost:3000  
  
■ Type s + enter at any time to create a snapshot of the database
```

8. In your Chrome browser open: <http://localhost:3000/projects>

9. You should see JSON data being returned.



```
[  
  {  
    "id": 0,  
    "name": "Johnson - Kutch",  
    "description": "Fully-configurable intermediate framework.",  
    "imageUrl": "assets/placeimg_500_300_arch4.jpg",  
    "contractTypeId": 3,  
    "contractSignedOn": "2013-08-04T22:39:41.473Z",  
    "budget": 54637,  
    "isActive": false,  
    "editing": false  
  },  
  {  
    "id": 1,  
    "name": "Wisozk Group",  
    "description": "Centralized interactive application. Exerci",  
    "imageUrl": "assets/placeimg_500_300_arch1.jpg",  
    "contractTypeId": 4,  
    "contractSignedOn": "2012-08-06T21:21:31.419Z",  
    "budget": 91638,  
    "isActive": true,  
    "editing": false  
  },  
  {  
    "id": 2,  
    "name": "Denesik LLC",  
    "description": "Re-contextualized dynamic moratorium. Aut",  
    "imageUrl": "assets/placeimg_500_300_arch12.jpg"  
}
```

✓ You have completed Lab 21

Lab 22: HTTP GET

Objectives

- Load the data from the REST API

Steps

Load the data from the REST API

1. Import the HttpClientModule.

```
src\app\app.module.ts
```

```
import { HttpClientModule } from '@angular/common/http';

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

Sometimes your editor won't automatically bring in an import. If this happens, try typing import and use the suggested **import statement** code snippet.

2. Add the **backendUrl** to **both** environment files.

src\environments\environment.ts
AND
src\environments\environment.prod.ts

```
// This file can be replaced during build by using the `fileReplacements` array.  
// `ng build ---prod` replaces `environment.ts` with `environment.prod.ts`.  
// The list of file replacements can be found in `angular.json`.  
  
export const environment = {  
  production: false,  
  backendUrl: 'http://localhost:3000'  
};
```

3. Inject the **HttpService** and make a **GET** request.

src\app\projects\shared\project.service.ts

```
import { HttpClient } from '@angular/common/http';  
import { environment } from ' ../../../../environments/environment';  
  
@Injectable({  
  providedIn: 'root'  
})  
export class ProjectService {  
  private projectsUrl = environment.backendUrl + '/projects/';  
  
  constructor(private http: HttpClient) {}  
  
  list(): Observable<Project[]> {  
    return of(MOCK_PROJECTS);  
    return this.http.get<Project[]>(this.projectsUrl);  
  }  
}
```

4. Verify the code is working.

- a. **Save** the files and the **browser** will automatically **reload**.
 - b. **Open** `http://localhost:4200`.
 - c. As in previous labs, the **list of projects** will appear.
 - d. **Open Chrome DevTools (F12)**
 - e. Click the **Network** tab> Set the **XHR** filter> Click the **projects/** request.

The data is now being loaded from the backend REST API. **XHR** stands for **XML HTTP Request** (the formal name for AJAX).

✓ You have completed Lab 22

Lab 23: HTTP Error Handling

Objectives

- Handle a HTTP error and translate it to a user-friendly error
- Display the user friendly error

Steps

Handle a HTTP error and translate it to a user-friendly error

```
src\app\projects\shared\project.service.ts

import { Observable, of, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
...
export class ProjectService {
  private projectsUrl = environment.backendUrl + '/projects/';

  constructor(private http: HttpClient) {}

  list(): Observable<Project[]> {
    return this.http.get<Project[]>(this.projectsUrl).pipe(
      catchError((error: HttpErrorResponse) => {
        console.log(error);
        return throwError('An error occurred loading the projects.');
      })
    );
  }
}
```

1. Update the data service to handle HTTP errors.

Display the user friendly error

2. Add an error method handler in the component.

```
src\app\projects\projects-container\projects-container.component.ts
```

```
...
export class ProjectsContainerComponent implements OnInit {
  projects: Project[];
  errorMessage: string;

  constructor(private projectService: ProjectService) {}
  ngOnInit() {
    this.projectService.list().subscribe(
      data => {
        this.projects = data;
      },
      error => {
        this.errorMessage = error;
      }
    );
  }
  ...
}
```

3. Display the error in the template.

```
src\app\projects\projects-container\projects-container.component.html
```

```
<h1>Projects</h1>
<div class="row">
  <div *ngIf="errorMessage" class="card large error">
    <section>
      <p>
        <span class="icon-alert inverse "></span> {{errorMessage}}
      </p>
    </section>
  </div>
</div>
<app-project-list
[projects]="projects" (saveListItem)="onSaveListItem($event)">
</app-project-list>
```

4. Change the URL so the API endpoint cannot be reached.

```
src\app\projects\shared\project.service.ts
```

```
...
export class ProjectService {
  private projectsUrl = environment.backendUrl + '/projects/wrong';
...
}
```

5. In your browser, you should **see** the **error message** displayed.

Projects

⌚ An error occurred loading the projects.

6. **Fix the URL** to the backend API before continuing to the next lab.

```
src\app\projects\shared\project.service.ts
```

```
...
export class ProjectService {
  private projectsUrl = environment.backendUrl + '/projects/wrong';
  ...
}
```

✓ You have completed Lab 23

Lab 24: HTTP PUT

Objectives

- Communicate with the REST API to update data
-

Steps

Steps begin on the next page.

Communicate with the REST API to update data

1. Implement a method in a data service to do a **PUT** (update).

```
src\app\projects\shared\project.service.ts

...
import { HttpClient, HttpHeaders } from '@angular/common/http';

const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })
};

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

  ...
  put(project: Project): Observable<Project> {
    const url = this.projectsUrl + project.id;
    return this.http.put<Project>(url, project, httpOptions).pipe(
      catchError((error: HttpErrorResponse) => {
        console.log(error);
        return throwError('An error occurred updating the projects.');
      })
    );
  }
}

snippets\lab24-step01.txt
```

2. Invoke the method in your container component.

src\app\projects\projects-container\projects-container.component.ts

```
...
export class ProjectsContainerComponent implements OnInit {
  projects: Project[];
  errorMessage: string;

  constructor(private projectService: ProjectService) {}
  ...

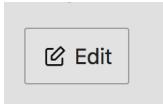
  onSaveListItem(event: any) {
    // const project = event.item;
    // const index = this.projects.findIndex(
    //   element => element.id === project.id
    // );
    // this.projects[index] = project;

    const project: Project = event.item;
    this.projectService.put(project).subscribe(
      updatedProject => {
        const index = this.projects.findIndex(
          element => element.id === project.id
        );
        this.projects[index] = updatedProject;
      },
      error => (this.errorMessage = error)
    );
  }
}
```

snippets\lab24-step02.txt

3. Verify the code is working by following these steps.

a. Click the edit button for a project.



b. Change the project name in the form.

A screenshot of a project edit form. It includes fields for Project Name (containing "Dusty Epsilon Updated" with a blue border), Project Description (containing placeholder text), Project Budget (containing "42400"), and Active? (with a checked checkbox). At the bottom are "Save" and "cancel" buttons.

c. Click save on the form.

d. Verify the card shows the updated data.

e. Refresh your browser.

f. Verify the project name is still updated.



✓ You have completed Lab 24

Lab 25: Showing a Loading Indicator

Objectives

- Show a loading indicator when HTTP requests are in flight.

Steps

1. Create a **loading** property and **set it before** issuing the **request** and then in the **next** and **error** callback **functions**.

```
src\app\projects\projects-container\projects-container.component.ts

...
export class ProjectsContainerComponent implements OnInit {
  projects: Project[];
  errorMessage: string;
  loading: boolean;

  constructor(private projectService: ProjectService) {}

  ngOnInit() {
    this.loading = true;
    this.projectService.list().subscribe(
      data => {
        this.loading = false;
        this.projects = data;
      },
      error => {
        this.loading = false;
        this.errorMessage = error;
      }
    );
  }
}
```

2. Display a loading indicator in the template if loading is true.

```
src\app\projects\projects-container\projects-container.component.html
```

```
<h1>Projects</h1>
<div *ngIf="loading" class="center-page">
  <span class="spinner primary"></span>
  <p>Loading ... </p>
</div>
<div class="row">
  ...

```

3. Add some styles to the global stylesheet to center the loading indicator on the page.

```
src\styles.css
```

```
html,
body,
.container {
  height: 100%;
}

.center-page {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100%;
}
```

```
input.ng-invalid {  
  ...
}
```

```
snippets/lab25-step03.css
```

4. Implement a delay in the service so you can easily see the indicator.

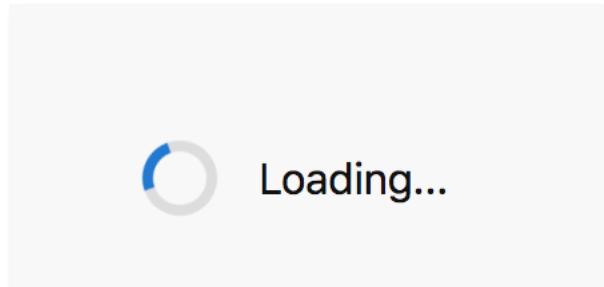
```
src\app\projects\shared\project.service.ts
```

```
import { catchError, delay } from 'rxjs/operators';
...
export class ProjectService {
  private projectsUrl = environment.backendUrl + '/projects/';

  constructor(private http: HttpClient) {}

  list(): Observable<Project[]> {
    return this.http.get<Project[]>(this.projectsUrl).pipe(
      delay(2000),
      catchError((error: HttpErrorResponse) => {
        console.log(error);
        return throwError('An error occurred ... ');
      })
    );
  }
}
```

5. Save the files and reload the application in the browser.
6. You should see a loading indicator while you are waiting for the delay to end.



7. Remove the **delay** in **ProjectService** before continuing to the next lab.

✓ You have completed Lab 25

Lab 26: Router Navigation

Objectives

- Create a Home module, component and route
 - Add a navigation menu
-

Steps

Create a Home module, component and route

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
2. **Run** the following **command** to **generate** a new feature module for components and other Angular constructs related to the **home** page.

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

If you open src\app\app.module.ts you will notice the HomeModule was automatically added to the imports of the AppModule.

3. Run the command to generate a home container component inside the home feature module you created in the last step.

```
ng g component home/home-container
```

4. Edit the component's template as follows.

```
src\app\home\home-container\home-container.component.html
```

```
<p>  
— home container works!  
</p>  
<h1>Home</h1>
```

5. Add a route that displays the component.

```
src\app\home\home-routing.module.ts
```

```
...  
import {HomeContainerComponent} from './home-container/home-  
container.component';  
  
const routes: Routes = [  
  { path: 'home', component: HomeContainerComponent },  
];  
...
```

If you have **Angular Snippets** available in your editor you can type part of **a-route-path-eager**, press **enter**, and the snippet will unfold.

To learn more visit the [documentation on Angular Snippets](#).

6. Make the home route the default route in the application.

```
src\app\app-routing.module.ts
```

```
...
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'home' }
];
...
```

Use the **a-route-path-default** snippet.

7. Make the following changes so the router knows where to output the component when it renders.

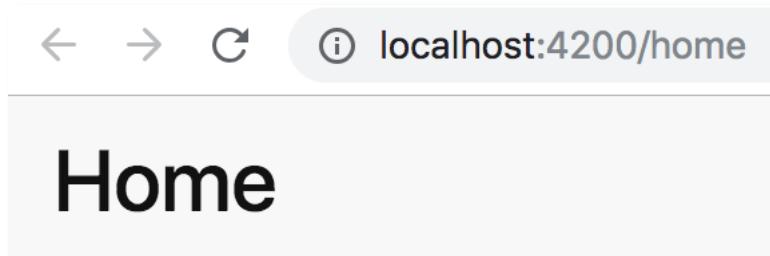
```
src\app\app.component.html
```

```
<app-projects-container></app-projects-container>
```

```
<div class="container">
  <router-outlet></router-outlet>
</div>
```

8. Save your changes and navigate to <http://localhost:4200/> in your browser.

9. The browser should be redirected to <http://localhost:4200/home> and see the **HomeContainerComponent**.



Configure a Route

10. Open the **projects-routing.module.ts** file and add a route to the **ProjectsContainerComponent**.

```
src\app\projects\projects-routing.module.ts
```

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ProjectsContainerComponent } from './projects-container/
projects-container.component';

const routes: Routes = [
  { path: 'projects', component: ProjectsContainerComponent }
];

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

Use the **a-route-path-eager** snippet.

Add a navigation menu

11. Add a navigation menu to your application.

```
src\app\app.component.html
```

```
<header class="sticky">
  <a [routerLink]=["./home"] class="logo">
    
  </a>
  <a [routerLink]=["./home"] class="button rounded"
    routerLinkActive="active">
    <span class="icon-home"></span>
    Home
  </a>
  <a [routerLink]=["./projects"] class="button rounded"
    routerLinkActive="active">
    Projects
  </a>
</header>
<div class="container">
  <router-outlet>
  </router-outlet>
</div>
```

```
snippets\lab26-step11.html
```

12. Add the following **styles** for the **navigation** menu.

```
src\styles.css
```

```
...
header {
    height: 5.1875rem;
}

a.button.active {
    border: 1px solid var(--fore-color);
}
```

```
snippets\lab26-step12.css
```

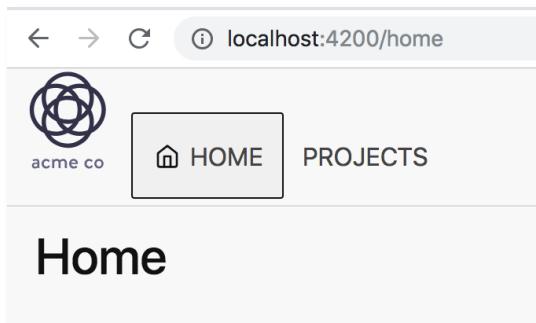
CSS variables are entities defined by CSS authors that contain specific values to be reused throughout a document. They are set using custom property notation (e.g., `--main-color: black;`) and are accessed using the `var()` function (e.g., `color: var(--main-color);`).

To learn more visit the [CSS documentation on MDN](#).

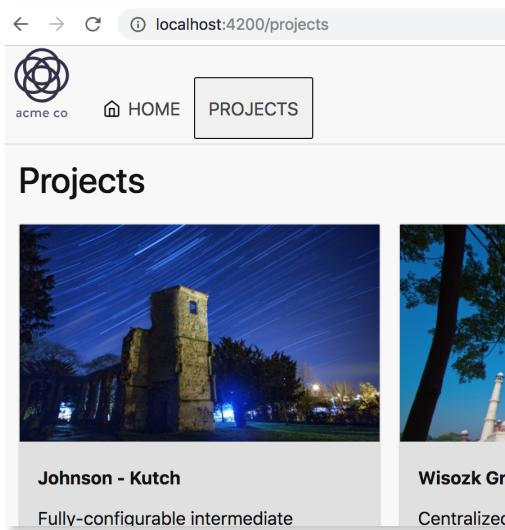
In this case, the variable `--fore-color` is defined in the file: `node_modules/mini.css/dist/mini-default.min.css` and used in `src\styles.css`.

13. Verify the menu is working by following these steps:

- a. **Save** your changes.
- b. The browser will automatically **reload**.
- c. You should see the **navigation** menu.



- d. **Click on Projects** and you should navigate to the project list.



- e. **Click on Home** and you should navigate back to home.

✓ You have completed Lab 26

Lab 27: Route Parameters

Objectives

- Navigate to a route with a parameter

Steps

Navigate to a route with a parameter

1. Add a find method to ProjectService to return a Project by Id.

```
src\app\projects\shared\project.service.ts
```

```
...
export class ProjectService {
  private projectsUrl = environment.backendUrl + '/projects/';

  constructor(private http: HttpClient) {}

  find(id: number): Observable<Project> {
    const url = this.projectsUrl + id;
    return this.http.get<Project>(url).pipe(
      catchError((error: HttpErrorResponse) => {
        console.error(error);
        return throwError('An error occurred loading the project');
      })
    );
  }
}

...
```

```
snippets\lab27-step01.txt
```

2. Copy the two directories:

- snippets\Lab27-RouteParameters\project-detail
- snippets\Lab27-RouteParameters\project-detail-container

Into the \code\labs\working\project-manage\src\app\projects directory (be sure to merge the new files into the existing files).

These directories contain some pre-built components you will use in this lab.

Take a moment to **review the code**.

3. Add the two new components to the declarations in the ProjectsModule.

src\app\projects\projects.module.ts

```
import { ProjectDetailComponent } from './project-detail/project-
detail.component';
import { ProjectDetailContainerComponent } from './project-detail-
container/project-detail-container.component';

@NgModule({
  imports: [...],
  declarations: [
    ...,
    ProjectDetailComponent,
    ProjectDetailContainerComponent
  ]
})
export class ProjectsModule {}
```

4. Add a route to display the **ProjectDetailContainer** component you just added.

```
src\app\projects\projects-routing.module.ts
```

```
...
import { ProjectDetailContainerComponent } from './project-detail-
container/project-detail-container.component';

const routes: Routes = [
  { path: 'projects', component: ProjectsContainerComponent },
  { path: 'projects/:id', component: ProjectDetailContainerComponent }
];
...
```

Be sure to route to:

ProjectDetailContainerComponent NOT ProjectDetailComponent

5. Make the card clickable by surrounding it with a link.

src\app\projects\project-card\project.card.component.html

```
<a [routerLink]="['./', project.id]">
  <div class="card">
    <img [src]=" project.imageUrl" [alt]="project.name" >
    <section class="section dark">
      <h5 class="strong">
        <strong>{{project.name}}</strong>
      </h5>
      <p>{{project.description}}</p>
      <p>
        Budget :
        {{project.budget | currency : 'USD': 'symbol': '0.0-2'}}</p>
      <button class="bordered" (click)="onEditClick(project, $event)">
        <span class="icon-edit"></span>
        Edit
      </button>
    </section>
  </div>
</a>
```

6. Verify the code works by following these steps:
 - a. Save your changes.
 - b. Click on Projects in the navigation if you aren't already at the projects route.
 - c. Click on any of the project cards.
 - d. You should see the projects detail page for the project you clicked.

The screenshot shows a web browser window with the URL `localhost:4200/projects/2`. The header contains a logo for 'acme co' and navigation links for 'HOME' and 'PROJECTS'. The main content area is titled 'Project Detail' and features a large image of a city skyline at dusk. Below the image, the project name 'Denesik LLC' is displayed in bold. A descriptive paragraph follows, stating: 'Re-contextualized dynamic moratorium. Aut nulla soluta numquam qui dolor architecto et facere dolores.' Below this, the budget is listed as '\$29,730' and the signing date is 'Signed: Jun 26, 2016'. A blue button labeled 'active' is visible at the bottom left of the content area.

- e. Click the back button in your browser to see the list of projects again.
- f. Click a different project card.
- g. You should see the projects detail page for the project you clicked.

Now that you have it working, take some time to review the code and step through it to see how all the pieces connect to provide a list to detail view.

✓ You have completed Lab 27

Lab 28: Custom Pipe

Objectives

- Create a custom pipe
 - Format data using a custom pipe
-

Steps

Create a custom pipe

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
2. **Run** the following **command** to **generate** a new shared feature module for pipes, components and directives used across the application in several different feature modules.

```
ng g module shared
```

3. Run the follow command to generate a custom pipe.

```
ng g pipe shared/truncate-string --export
```

Adding the **shared/** path before the pipe name will create the pipe in the shared folder and add it to the **declarations** of the shared module you created in the previous step. The flag **--export** tells the Angular CLI to also add the pipe to the exports of the shared module so it can be used in other feature modules if the SharedModule is imported.

4. Implement the **transform** method in the custom pipe.

src\app\shared\truncate-string.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'truncateString'
})
export class TruncateStringPipe implements PipeTransform {

  transform(value: any, length: number): any {
    if (value.length > length) {
      return value.substring(0, length) + ' ... ';
    } else {
      return value;
    }
  }
}
```

snippets\lab28-step04.txt

Format data using a custom pipe

5. Import the **SharedModule** into the feature module **ProjectModule** so it can be used in the project-card.component.html template.

```
src\app\projects\shared\projects.module.ts
```

```
...
@NgModule({
  imports: [
    CommonModule,
    ProjectsRoutingModule,
    ReactiveFormsModule,
    SharedModule
  ],
  declarations: [
    ...
  ]
})
export class ProjectsModule {}
```

6. Use the pipe in a the template for a component.

```
src\app\projects\project-card\project-card.component.html
```

```
...
<h5 class="strong">
<strong>{{project.name}}</strong>
</h5>
<p>{{project.description | truncateString: 60}}</p>
...
```

7. Verify the code is working.
 - a. Save your code changes.
 - b. Click on Projects in the navigation if you aren't at that route already.
 - c. The project descriptions should all be truncated at 60 characters and all end with an ellipsis (...) as shown below.

Projects



Johnson - Kutch

Fully-configurable intermediate framework. Ullam occaecati l...

Budget : \$54,637

Edit



Wisozk Group

Centralized interactive application. Exercitationem nulla ut...

Budget : \$91,638

Edit



Denesik LLC

Re-contextualized dynamic moratorium. Aut nulla soluta numqu...

Budget : \$29,730

Edit

✓ You have completed Lab 28

Lab 29: Build & Deploy

Objectives

- Build an Angular application
- Deploy the application to a web server

Steps

Build an Angular application

1. If you don't already have one open, **open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
2. **Run** the following **command to build** the application for production deployment.

```
ng build --prod
```

3. When the command completes you should see output similar but not exactly as shown below.

```
Date: 2019-07-01T18:14:52.690Z
Hash: 1486dd62053784343e95
Time: 26522ms
chunk {0} runtime-es5.741402d1d47331ce975c.js (runtime) 1.41 kB [entry] [rendered]
chunk {1} main-es5.0d657494dcf4e404aff8.js (main) 358 kB [initial] [rendered]
chunk {2} polyfills-es5.0e123feb616b379211ce.js (polyfills) 111 kB [initial] [rendered]

Date: 2019-07-01T18:15:13.333Z
Hash: 0cfcc3d9d37ae6f12d52e
Time: 20604ms
chunk {0} runtime-es2015.858f8dd898b75fe86926.js (runtime) 1.41 kB [entry] [rendered]
chunk {1} main-es2015.8e469659067ed4bc9b51.js (main) 309 kB [initial] [rendered]
chunk {2} polyfills-es2015.ab241c6a8dff4ff1bd55.js (polyfills) 36.4 kB [initial] [rendered]
chunk {3} styles.c1084cd8ee851a13732c.css (styles) 45.6 kB [initial] [rendered]
```

A `dist\project-manage` directory is created **inside your top level project-manage directory** with the files needed for deployment.

4. Run the following **command** to change your current directory.

```
cd dist\project-manage
```

Deploy the application to a web server

5. Run the following **command** to install a Node.js web server named **serve**.

```
npm install serve@10.1.1 -g
```

Assuming you would like to serve a static site, single page application or just a static file (no matter if on your device or on the local network), this package is a development web server that serves static content.

It behaves exactly like static deployments on
<https://zeit.co/now>
so it's perfect for developing your static project.

For more information see:
<https://www.npmjs.com/package/serve>

6. Run the following **command** to serve your current directory `dist\project-manage`.

```
serve
```

7. The output should be as follows.

Serving!

- **Local:** http://localhost:5000
- **On Your Network:** http://10.0.0.3:5000

Copied local address to clipboard!

8. **Open** a browser and paste the local link copied to your clipboard in the last step into the address bar.
9. You should see the **application running** in your browser.
10. **Click on projects** in the top navigation.
11. After navigating to the projects route, **refresh** your **browser**.
12. You should see a **404 error** page.

404

The requested path could not be found

13. Use **Ctrl+C** to **stop** the web server.
14. **Run** the **serve** command again but add the **-s** flag for *single-page applications*.

```
serve -s
```

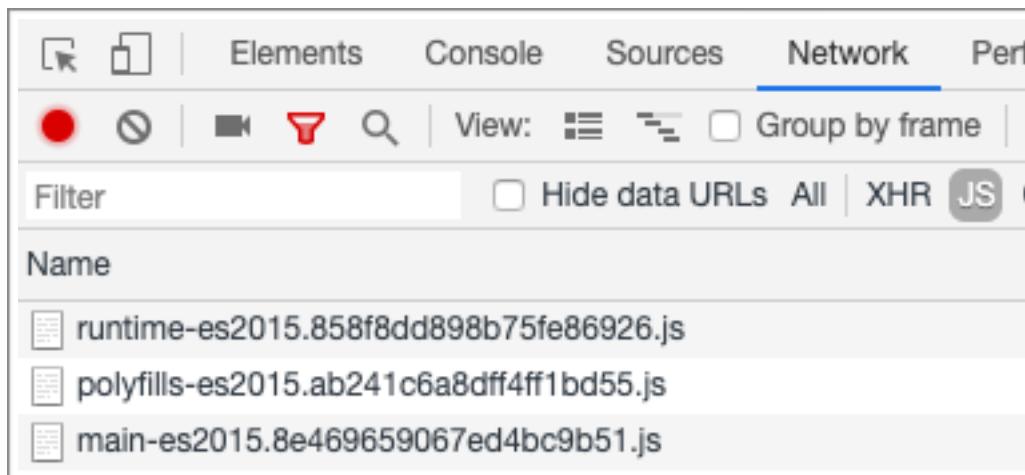
15. Follow these steps to verify the server is now redirecting to index.html when it can't find a route.

- a. You should see the **application running** in your browser.
- b. **Click on projects** in the top navigation.
- c. After navigating to the projects route, **refresh your browser**.
- d. You should see the **projects page refresh and display the projects**. Note that you are **no longer getting a 404 error**.

16. To see differential loading:

- a. If you are not already viewing the application using the Chrome browser...copy the URL: <http://localhost:5000> into Chrome
- b. Open the DevTools
- c. Switch to the network tab
- d. Refresh the page

17. Notice that only the .js files with **es2015** in the name are loaded.



18. If you have the Edge browser installed view the network traffic using its

```
src\tsconfig.json
```

```
...
"importHelpers": true,
  "target": "es2015",
  "target": "es5",
  "typeRoots": [
    "node_modules/@types"
  ],
...

```

developer tools. If you are using an Edge version that does not yet use the Chromium engine (which only insider builds do at the time of publishing) you will notice that both the es5 and es2015 scripts load and they load two to three times depending on your version.

For more information see:

<https://github.com/philipwalton/webpack-esnext-boilerplate/issues/1#issuecomment-445280691>

Fortunately those quirks with IE and Edge should go away soon now that Microsoft is finally giving up on creating a browser engine and it seems they are going to build their browser using the Chromium engine. For more information see:

<https://www.microsoftedgeinsider.com/>

19. **Optional (complete only if time allows)** - After seeing how differential loading works in the current versions of Edge you may want to turn off differential loading. Here are the steps.

- a. Enable the dead or IE browsers in the **browserslist config** file by **removing** the **not** keyword in front of them.

```
src\browserslist
...
> 0.5%
last 2 versions
Firefox ESR
not dead
not IE 9-11 # For IE 9-11 support, remove 'not'.
```

- b. Set the **target** in the **compilerOptions** to **es5**.
- c. You'll need to stop the web server, rebuild the project, and restart the web server to see the changes. Be sure you are in the **top level project-manage** directory when you rebuild the server and the **dist/project-manage** directory when you restart the web server.
- d. Notice that only one set of the js files is created (the es5 ones although the suffix -es5 is no longer necessary except for the polyfills files).

✓ You have completed Lab 29

If time permits you can follow very similar steps to deploy the application on common production web servers including Apache and IIS by following the specific directions in the Angular documentation.

<https://angular.io/guide/deployment#production-servers>

The **snippets** directory contains a **web.config** for IIS and an **.htaccess** file for **Apache** to make it easier.

Appendix A: How to Skip Labs

Labs can be skipped by attendees who:

- arrive late, leave early
 - get pulled into a meeting
 - have a doctors appointment
 - understand a topic and want to move on to a topic they don't know
 - etc...
-

Steps

1. Close any editor(s) and command prompt or terminal related to the course labs.
2. **Open a command prompt** (Windows) or **terminal** (Mac). Set the directory to the **begin\project-manage** for the lab on which you would like to start working on.
3. Run the command.

```
npm install
```

4. Run the command.
5. If you are working a lab which requires the backend api (lab 21 or later).

```
ng serve -o
```

Open another command-line or terminal. Run the command.

```
npm run api
```

For a specific example see the next page.

For example, if you want to:

- Finish | Lab 24: Http Put
- Skip | Lab 25: Showing a Loading Indicator
- Work on | Lab 26: Router Navigation

...then

- Close the project-manage folder where you were working on Lab 24: Http Put
- Open the directory below in your editor and on the command-line:
 - code\labs\lab26\begin\project-manage
- Run an npm install and after it finishes
- Run the commands
 - ng serve -o
 - npm run api
 - In separate command-line or terminal windows

Note that you:

- Won't lose your current code
- **Will work on future labs in the directory:**
 - code\labs\lab26\begin\project-manage

Appendix B: REST Review

Objectives

- Start REST API server
 - Test the REST API
-

Steps

⚠ This lab assumes you are using Visual Studio Code as your editor. If you aren't using Visual Studio Code as your editor, the instructor can complete the remainder of this lab as a demonstration. This lab is a review of RESTful web services and is optional, i.e. not required to complete other labs.

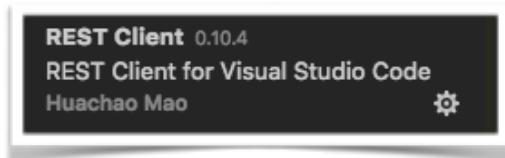
1. Setup
 - a. If not already open, re-open the code folder in your editor and navigate to the \api\test folder.
2. Start REST API server
 - a. **Open** a separate **command-prompt/terminal** and **run** the following command to start the web server running the backend REST API.

```
npm run api
```

⚠ You can skip this step if you already started the server in Lab 18 and have left it running.

3. Test the REST API

- a. If you haven't already: **Install the Visual Studio Code Plugin named REST Client.**



- b. **GET** the projects data

- **Open** the \api\test\projects-get.http file.
- If you are using Visual Studio Code
 - Choose View> Command Palette
 - Type rest s and choose Rest Client: Send Request. Note the shortcut on your operating system for future requests.



- c. The screen will split and you should see the response on the right pane.

A screenshot of Visual Studio Code showing a split interface. The left pane is a terminal window displaying the command "GET http://localhost:3000/projects/" followed by "HTTP/1.1". The right pane is titled "Response" and shows the JSON response from the server. The response includes headers such as "HTTP/1.1 200 OK 5ms", "x-powered-by: Express", "vary: Origin, Accept-Encoding", "access-control-allow-credentials: true", "cache-control: no-cache", "pragma: no-cache", "expires: -1", "x-content-type-options: nosniff", "content-type: application/json; charset=utf-8", "etag: W/"b48-A7uPukmV46/SDM+gd8EYiw\"", "content-encoding: gzip", "date: Tue, 23 Aug 2016 01:36:25 GMT", "connection: close", and "transfer-encoding: chunked". The JSON data in the response pane is as follows:

```
[{"id": 1, "name": "Matdexon", "description": "Lorem ipsum dolor sit amet,"}]
```

- d. Open the \api\test\projects-post.http file in the left pane.
- e. Run the command Rest Client: Send Request.
- f. Open the \api\test\projects-get.http file again
- g. Run the command Rest Client: Send Request.
 - Notice “Another Project” has been added at the end of the projects JSON array returned.
- h. Note the project with the id of 3 has a name of Remote Wrench.
- i. Open the \api\test\projects-put.http file on the left pane.
- j. Run the command Rest Client: Send Request
 - Notice the object with an id of 3 now has the name Remote Wrenchs.
- k. Open the \api\test\projects-delete.http file on the left side.
- l. Run the command Rest Client: Send Request.
 - Verify that the project with a name of “Another Project” has been deleted.

You have successfully completed the REST Review.