

Angular

Lab Manual



Copyright © 2018-2020

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.

About this Lab Manual	3
Lab 30: Search	6
Lab 31: Search using RxJS	10
Unit Testing Lab 1: First Test	13
Unit Testing Lab 2: Component Test	17
Unit Testing Lab 3: Component with Input & Output	24
Unit Testing Lab 4: Component with Service	30
Unit Testing Lab 5: Service Mocking Http	35
Unit Testing Lab 6: Pipe	40
Appendices: Optional Labs	42
E2E Testing Lab 1: First Test	43
E2E Testing Lab 2: Page Objects	47
E2E Testing Lab 3: Loading Data	49
E2E Testing Lab 4: Saving Data	51
Appendix A: How to Skip Labs	54

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 in the are relative to the **project-manage** directory.

So the file below will be found at:

AngularCourse\code\labs\working\project-manage\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.
- 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 below.

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],
})
```

snippets\lab00-step00.html

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 30: Search

Objectives

- ☐ Add the ability to search for projects

Steps

Add the ability to search for projects

1. Add a **listByName** method to the **ProjectService**.

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

```
...
export class ProjectService {
  ...
  list(): Observable<Project[]> { ... }

  listByName(name: string): Observable<Project[]> {
    if (!name.trim()) {
      return this.list(); // if no name was provided, list all
    }
    const url = `${this.projectsUrl}?name_like=${name}`;
    return this.http.get<Project[]>(url).pipe(
      catchError((error: HttpErrorResponse) => {
        console.error(error);
        return throwError('An error occurred searching the projects.');
```

snippets\lab30-step01.txt

2. Add an **onSearch** method and a **search** method. Invoke **search** in **ngOnInit**.

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

```
export class ProjectsContainerComponent implements OnInit {  
  ...  
  ngOnInit() {  
    this.loading = true;  
    this.projectService.list().subscribe(  
      ...  
    );  
    this.search('');  
  }  
  
  onSearch(term: string) {  
    this.search(term);  
  }  
  
  search(term: string) {  
    this.loading = true;  
    this.projectService.listByName(term).subscribe(  
      data => {  
        this.loading = false;  
        this.projects = data;  
      },  
      error => {  
        this.loading = false;  
        this.errorMessage = error;  
      }  
    );  
  }  
  ...  
}
```

snippets\lab30-step02.txt

3. **Add** a **search input** to the template and **call onSearch** on the **keyup** event.

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

```
<h1>Projects</h1>
<div class="row">
  <div class="col-sm-12">
    <div class="input-group fluid">
      <input #searchBox type="text" name="searchBox"
        placeholder="Search" (keyup)="onSearch(searchBox.value)">
    </div>
  </div>
</div>
<div class="row">
```


snippets\lab30-step03.txt

4. Verify


- a. **Save** your **code** changes.
- b. **Click** on **Projects** in the navigation if you aren't at that route already.
- c. **Type "group"** in the search input.

- d. The **projects** should be **filtered** to ones with “**group**” in their name.


Projects



Wisozk Group
Centralized interactive application



Pollich Group
Ergonomic heuristic firmware



Kutch Group
Decentralized explicit case

Notice that the screen flashes with your every keystroke resulting in a poor user experience. We will fix this in the next lab using RxJS and Observables.

✓ You have completed Lab 30

Lab 31: Search using RxJS

Objectives

- ☐ Improve the user experience when searching
-

Steps

Steps begin on the next page.

1. Refactor ProjectsContainerComponent to use an observable Subject.

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

```
...
import { Subject, Observable, Subscription } from 'rxjs';
import { debounceTime, distinctUntilChanged, switchMap } from 'rxjs/operators';

export class ProjectsContainerComponent implements OnInit, OnDestroy {
  projects: Project[];
  errorMessage: string;
  loading: boolean;
  private searchTerms = new Subject<string>();
  private subscription: Subscription;

  constructor(private projectService: ProjectService) {}

  ngOnInit() {
    this.observeSearchTerms();
    this.searchTerms.next('');
  }

  onSearch(term: string) {
    this.searchTerms.next(term);
  }

  observeSearchTerms() {
    this.subscription = this.searchTerms
      .pipe(
        // wait 300ms after each keystroke before considering the term
        debounceTime(300),

        // ignore new term if same as previous term
        distinctUntilChanged(),

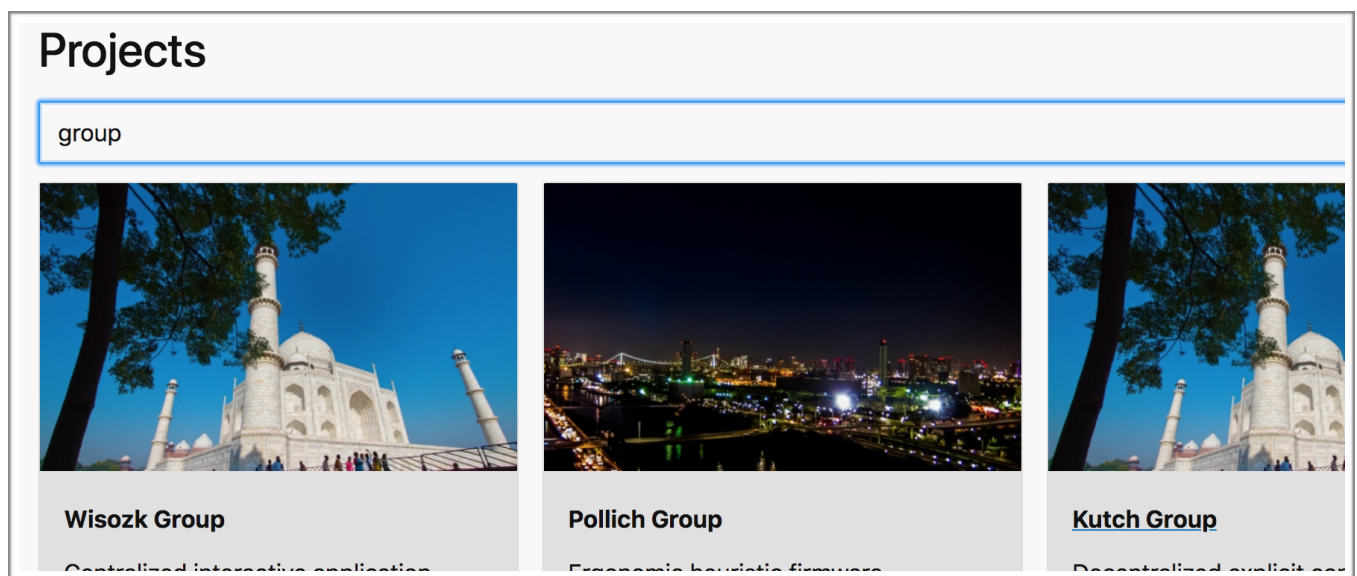
        // switch to new search observable each time the term changes
        switchMap(
          (term: string): Observable<Project[]> => {
            this.loading = true;
            return this.projectService.listByName(term);
          }
        )
      )
      .subscribe(
        data => {
          this.loading = false;
          this.projects = data;
        },
        error => {
          this.loading = false;
          this.errorMessage = error;
        }
      );
  }

  ngOnDestroy(): void {
    this.subscription.unsubscribe();
  }
}
```

snippets\lab31-step01.txt

2. Verify

- a. **Save** your **code** changes.
- b. **Click** on **Projects** in the navigation if you aren't at that route already.
- c. **Type “group”** in the search input.
- d. The **projects** should be **filtered** to ones with **“group”** in their name.



Notice that the screen no longer flashes with your every keystroke resulting in a significantly improved user experience.

✓ You have completed Lab 31

Unit Testing Lab 1: First Test

Objectives

- ☐ Write your first JavaScript unit test
 - ☐ Debug a unit test
-

Steps

Write your first JavaScript unit test

1. Close all your current editors and command prompts / terminals.
2. Open the following directory in your editor as the top level directory. This will be your starting point and working directory for all the unit testing labs.
 - `code\labs\unit-lab00\complete\project-manage`

The code is the completed Angular labs up to this point. In addition, the unit test files (.spec) generated by the Angular CLI have been commented out where the tests are failing but the boiler-plate setup code for testing has been left to save us typing. We will get all the unit tests passing in the upcoming labs.

3. **Open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
4. **Run** the following **command** to install all JavaScript dependencies in this folder

```
npm install
```

5. **Run** the following **command** to build the Angular project and run the tests in both the Karma console test runner and the Jasmine HTML test runner.

```
ng test
```

6. A Chrome browser will open and run the unit tests in the Jasmine HTML test runner. Karma will run the tests and display the results at the command prompt or terminal.

```
Executed 8 of 8 SUCCESS
```

Both processes will watch for change to files with .spec in their name and run again whenever you save a change. Note that running “npm test” runs the “ng test” command. Running either command is equivalent.

7. **Create** the following **spec file** and **add** the following **code**.

```
src\app\smoke-test.spec.ts
```

```
describe('Smoke Test', () => {  
  it('should run a passing test', () => {  
    expect(true).toEqual(false);  
  });  
});
```

8. **Save** the file and you should **see** a **failure message** similar to the one shown below.

```
Smoke Test should run a passing test FAILED  
  Expected true to equal false.  
  ...
```

9. **Change** false to true in the test.

```
src/app/smoke-test.spec.ts

describe('Smoke Test', () => {
  it('should run a passing test', () => {
    expect(true).toEqual(true);
  });
});
```

10. **Save** the file and you should **see** the following **success message**.

```
Executed 9 of 9 (SUCCESS)
```

Debug a unit test

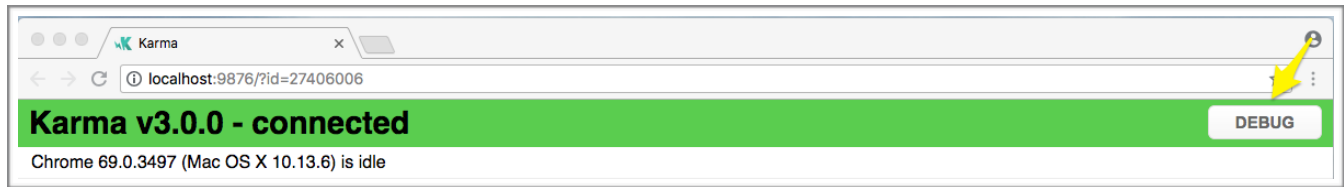
11. **Add** a **debugger statement** to the unit test as shown below.

```
src/app/smoke-test.spec.ts

describe('Smoke Test', () => {
  it('should run a passing test', () => {
    debugger;
    expect(true).toEqual(true);
  });
});
```

Your linter (tslint) will display an error that use of debugger statements is forbidden. You can safely ignore this error. It is trying to prevent you from accidentally leaving this line in production code and causing a defect. In this case, we are using it to make it easier to break into the test instead of searching for the file in the Chrome DevTools source tab.

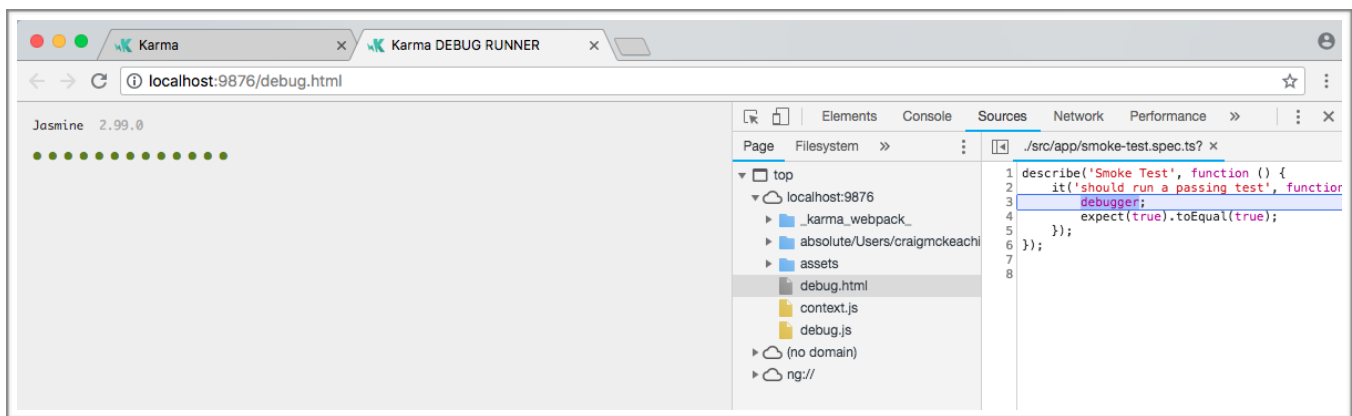
12. Find the karma browser window and click the **DEBUG** button in the upper right corner.



13. A new browser tab opens and re-runs the tests.

14. **Open** the Chrome browser's **DevTools** (F12).

15. **Refresh** the browser...and it **stops** at the **debugger** breakpoint.



16. Click the **continue** button or **F8** to let the script finish.



17. **Remove** the **debugger** statement from the test.

✓ You have completed Unit Testing: Lab 1

Unit Testing Lab 2: Component Test

Objectives

- ☐ Test a simple component
 - ☐ Understand how to detect changes in a component
-

Steps

Test a simple component

1. As mentioned previously, your working directory for all the unit testing labs should be:
 - `code\labs\unit-lab00\complete\project-manage`
2. *If not already running*, **run** the command **ng test** in the working directory.

Steps continue on the next page.

3. **Add a variable** to hold the header element. **Query** the component for the **header element**. Write a test to **verify** the **value** of the **header element**.

```
src\app\home\home-container\home-container.component.spec.ts
```

```
...
describe('HomeControllerComponent', () => {
  let component: HomeControllerComponent;
  let fixture: ComponentFixture<HomeControllerComponent>;
  let h1: HTMLElement;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [HomeControllerComponent]
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(HomeControllerComponent);
    component = fixture.componentInstance;
    h1 = fixture.debugElement.nativeElement.querySelector('h1');
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should render title in a h1 tag', () => {
    expect(h1.textContent).toEqual('Home');
  });
});
```

4. **Save** the file and the **test** will **automatically run**.
5. **Verify** you see an additional successful test.

```
Executed 10 of 10 (SUCCESS)
```

Understand how to detect changes in a component

6. **Update** the **component** to **dynamically set a title property** into the header.

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

```
...
@Component({
  selector: 'app-home-container',
  templateUrl: './home-container.component.html',
  styleUrls: ['./home-container.component.css']
})
export class HomeContainerComponent implements OnInit {
  title = '';
  constructor() {}

  ngOnInit() {}
}
```

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

```
<h1>Home</h1>
<h1>{{title}}</h1>
```

7. **Change** the “should render title...” **test** to **expect** an **empty string**.

```
src\app\home\home-container\home-container.component.spec.ts
```

```
...
describe('HomeContainerComponent', () => {
  let component: HomeContainerComponent;
  let fixture: ComponentFixture<HomeContainerComponent>;
  let h1: HTMLElement;

  ...

  it('should render title in a h1 tag', () => {
    expect(h1.textContent).toEqual('');
  });
});
```

8. **Save** the file to run the tests again and **verify** they all pass.

```
Executed 10 of 10 (SUCCESS)
```

9. **Add** another **test** that **sets** the **title** property on the component.

```
src\app\home\home-container\home-container.component.spec.ts
```

```
...
describe('HomeContainerComponent', () => {
  let component: HomeContainerComponent;
  let fixture: ComponentFixture<HomeContainerComponent>;
  let h1: HTMLElement;

  ...

  it('should render title in a h1 tag', () => {
    expect(h1.textContent).toEqual('');
  });

  it('changing title, updates h1', () => {
    const title = 'Home';
    component.title = title;
    expect(h1.textContent).not.toContain(title, 'before detectChanges');
    fixture.detectChanges();
    expect(h1.textContent).toContain(title);
  });
});
```

Notice that calling **detectChanges** on the fixture causes the component to render again and that prior to calling **detectChanges** the **h1** is not yet updated.

10. Verify the new test passes.

```
Executed 11 of 11 (SUCCESS)
```

✓ You have completed Unit Testing: Lab 2

Unit Testing Lab 3: Component with Input & Output

Objectives

- ☐ Test an input property
 - ☐ Test an output property
-

Steps

Test an input property

1. *If not already running, run the command **ng test**.*
2. Uncomment the contents of the file:

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

Highlight a block and choose **Ctrl+ /** (Windows) or **Cmd+ /** (Mac) to comment or uncomment a block of code.

After uncommenting the code you will get test failures. Continue with the next steps to resolve these failures.

3. **Add TruncateStringPipe** to the testing module's declarations since it is used in the components's template. **Import** the **RouterTestingModule** since the component's template uses the **routerLink** directive.

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

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';
import { ProjectCardComponent } from './project-card.component';
import { RouterTestingModule } from '@angular/router/testing';
import { TruncateStringPipe } from 'src/app/shared/truncate-string.pipe';
import { Project } from '../shared/project.model';
```

```
describe('ProjectCardComponent', () => {
  let component: ProjectCardComponent;
  let fixture: ComponentFixture<ProjectCardComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      imports: [RouterTestingModule],
      declarations: [ProjectCardComponent, TruncateStringPipe]
    }).compileComponents();
  }));

  ...
  // code that goes here appears in the next code block
});
```

4. Set the **input property** to a project. Verify the **HTML header** is updated.

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

```
...
describe('ProjectCardComponent', () => {
  ...
  beforeEach(() => {
    fixture = TestBed.createComponent(ProjectCardComponent);
    component = fixture.componentInstance;
    component.project = new Project(
      1,
      'Mission Impossible',
      'This is really difficult.',
      'assets/placeimg_500_300_arch7.jpg',
      5,
      new Date(2015, 1, 2),
      30100,
      true,
      false
    ),
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('header should be project name', () => {
    const h5 = fixture.nativeElement.querySelector('h5');
    expect(h5.textContent).toEqual(component.project.name);
  });
});
```

```
snippets\unit-test-lab03-step04.txt
```

5. Verify the new test passes.

```
Executed 13 of 13 (SUCCESS)
```

Test an output property

6. Act as if the test is the parent component receiving the output event.
Subscribe to the output event and trigger it from the component test.

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

```
...
import { By } from '@angular/platform-browser';

describe('ProjectCardComponent', () => {
  ...

  it('should raise event when edit clicked', () => {
    let projectBeingEdited: Project;

    component.edit.subscribe(
      (event: any) => (projectBeingEdited = event.editingProject)
    );

    const editButtonDebugElement = fixture.debugElement.query(By.css('button'));
    editButtonDebugElement.triggerEventHandler('click', {
      preventDefault: () => {}
    });
    expect(projectBeingEdited).toEqual(component.project);
  });
});
```

```
snippets\unit-test-lab03-step06.txt
```

Check to ensure the import path for **By** is:

```
import { By } from '@angular/platform-browser';
```

And not:

```
import { By } from 'selenium-webdriver';
```

The second argument passed to **triggerEventHandler** is a stub of your browser's **event** object. Because we only call one method `preventDefault` on the event object we just need to stub that method and do nothing in the implementation.

7. Verify the new test passes.

Executed 14 of 14 (SUCCESS)

✓ You have completed Unit Testing: Lab 3

Unit Testing Lab 4: Component with Service

Objectives

- ☐ Test a component with an observable async service
-

Steps

Test a component with an observable async service

1. *If not already running, run the command **ng test**.*
2. Uncomment the contents of the file:

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

Highlight a block and choose **Ctrl+ /** (Windows) or **Cmd+ /** (Mac) to comment or uncomment a block of code.

After uncommenting the code you will get test failures. Continue with the next steps to resolve these failures.

3. As shown in the next code block, since you are trying to only test the **ProjectListContainerComponent** create a **stub** for the:
 - a. **ProjectListComponent** and **add** it to the testing module **declarations**
 - b. **ProjectService** and **add** it to the testing module **providers**

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

```
import { async, ComponentFixture, TestBed } from '@angular/core/testing';
import { ProjectsContainerComponent } from '../projects-container.component';
import { Component, Input } from '@angular/core';
import { Project } from '../../shared/project.model';
import { Observable, of } from 'rxjs';
import { PROJECTS } from '../../shared/mock-projects';
import { ProjectService } from '../../shared/project.service';
```

```
@Component({ selector: 'app-project-list', template: '' })
class ProjectListStubComponent {
  @Input()
  projects: Project[] = [];
}
```

```
export class ProjectServiceStub {
  listByName(): Observable<Project[]> {
    return of(PROJECTS);
  }
}
```

```
describe('ProjectsContainerComponent', () => {
  let component: ProjectsContainerComponent;
  let fixture: ComponentFixture<ProjectsContainerComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ProjectsContainerComponent, ProjectListStubComponent],
      providers: [{ provide: ProjectService, useClass: ProjectServiceStub }]
    }).compileComponents();
```

```
snippets\unit-test-lab04-step03.txt
```

4. **Add a test** to verify the data is available after the asynchronous search completes because **detectChanges** triggers **ngOnInit** which sends an empty

string as a search term.

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

```
describe('ProjectsContainerComponent', () => {  
  ...  
  // works if no debounceTime because of is a synchronous observable  
  // it('should have projects sync', () => {  
  //   expect(component.projects.length).toEqual(7);  
  // });  
  
  it('should have projects', async(() => {  
    fixture.whenStable().then(() => {  
      expect(component.projects.length).toEqual(7);  
    });  
  }));  
  
});
```

```
snippets\unit-test-lab04-step04.txt
```

Review the **commented code** above which demonstrates that because we stubbed the service using the **of** creation function in RxJS which is **synchronous**. **Stubs can make asynchronous** methods that normally make an AJAX request into **synchronous** operations so wrapping the function in **async** becomes unnecessary. In this case however, the **component uses debounceTime** to wait 300ms after each keystroke before considering the next search term. And **debounceTime** internally **uses** JavaScript's **setTimeout** method **which is asynchronous** so we need to **wrap** our **expectation** in a **async** function and check **whenStable** so we know that the callback has returned and the data is available.

5. Verify the new tests pass.

```
Executed 16 of 16 (SUCCESS)
```

6. Verify the data renders to the screen properly we test the **ProjectListComponent**.

The **ProjectListComponent** tests use similar techniques to ones we have already learned so I have provided the implementation.

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

Replace the current contents of this file with the snippet below.

```
snippets\unit-test-lab04-step06.txt
```

We are testing the child **ProjectListComponent** by setting its **input property** to an array of projects and stubbing its child component **ProjectCard**.

Bonus exercise: If you finish the lab early try removing the export keyword on the ProjectCardStubComponent.

```
export class ProjectCardStubComponent
```

The test will continue to pass but you will receive an error.

Why are you receiving the error?

7. Verify the new tests pass.

```
Executed 19 of 19 (SUCCESS)
```

✓ You have completed Unit Testing: Lab 4

Unit Testing Lab 5: Service Mocking Http

Objectives

- ☐ Test a service by mocking the HttpClient service
-

Steps

Test a component with an observable async service

1. *If not already running, run the command **ng test**.*
2. Uncomment the contents of the file:
`src\app\projects\shared\project.service.spec.ts`

Highlight a block and choose **Ctrl+ /** (Windows) or **Cmd+ /** (Mac) to comment or uncomment a block of code.

After uncommenting the code you will get test failures. Continue with the next steps to resolve these failures.

3. **Setup** the HttpClientTestingModule and the HttpTestingController so XML HTTP Requests (XHR) are mocked.

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

```
import { TestBed } from '@angular/core/testing';
import { ProjectService } from '../project.service';
import {
  HttpClientTestingModule,
  HttpTestingController
} from '@angular/common/http/testing';
import { HttpClient } from '@angular/common/http';
import { PROJECTS } from '../mock-projects';
import { environment } from 'src/environments/environment';

describe('ProjectService', () => {
  let httpClient: HttpClient;
  let httpTestingController: HttpTestingController;
  let service: ProjectService;
  let projectsUrl: string;

  beforeEach(() => {
    TestBed.configureTestingModule({ imports: [HttpClientTestingModule] });
    httpClient = TestBed.get(HttpClient);
    httpTestingController = TestBed.get(HttpTestingController);
    service = TestBed.get(ProjectService);
    projectsUrl = environment.backendUrl + '/projects/';
  });

  it('should be created', () => {
    const service: ProjectService = TestBed.get(ProjectService);
    expect(service).toBeTruthy();
  });
});
```

snippets\unit-test-lab05-step03.txt

4. **Write a test** to verify the http request is made when listing projects. After each test, **verify** there are **no more pending requests**.

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

```
describe('ProjectService', () => {  
  let httpClient: HttpClient;  
  let httpTestingController: HttpTestingController;  
  let service: ProjectService;  
  let projectsUrl: string;  
  
  beforeEach(() => { ... });  
  
  it('should list projects', () => {  
    service.list().subscribe(data => expect(data).toEqual(PROJECTS));  
    const request = httpTestingController.expectOne(projectsUrl);  
    request.flush(PROJECTS);  
  });  
  
  afterEach(() => {  
    httpTestingController.verify();  
  });  
  
});
```

```
snippets\unit-test-lab05-step04.txt
```

5. **Save** the file and **verify** that the **test passes** successfully.

```
Executed 21 of 21 (SUCCESS)
```

6. **Test** that a **user friendly error** is returned **when an HTTP error** occurs on the server.

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

```
...
describe('ProjectService', () => {
  let httpClient: HttpClient;
  let httpTestingController: HttpTestingController;
  let service: ProjectService;
  let projectsUrl: string;

  beforeEach(() => { ... });

  it('should return user friendly error when listing projects', () => {
    const notFoundErrorResponse = { status: 404, statusText: 'Not Found' };
    const content = 'The requested URL was not found on the server.';
    service.list().subscribe(
      data => {},
      error => {
        expect(error).toEqual('An error occurred loading the projects.');
      }
    );
    const request = httpTestingController.expectOne(projectsUrl);
    request.flush(content, notFoundErrorResponse);
  });
  ...
});
```

```
snippets\unit-test-lab05-step06.txt
```

7. **Save** the file and **verify** that the **test passes** successfully.

```
Executed 22 of 22 (SUCCESS)
```

✓ You have completed Unit Testing: Lab 5

Unit Testing Lab 6: Pipe

Objectives

- ☐ Test a pipe
-

Steps

Test a pipe

1. *If not already running*, **run** the command **ng test**.

Steps continue on the next page.

2. Add the tests below. Also **update** the “create an instance” test to use the pipe variable instead of creating another instance.

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

```
import { TruncateStringPipe } from './truncate-string.pipe';

describe('TruncateStringPipe', () => {
  const pipe = new TruncateStringPipe();

  it('create an instance', () => {
    const pipe = new TruncateStringPipe();
    expect(pipe).toBeTruthy();
  });

  it('truncates string at given length', () => {
    expect(pipe.transform('abc', 2)).toEqual('ab ... ');
  });

  it('returns same string if length equals the length', () => {
    expect(pipe.transform('ab', 2)).toEqual('ab');
  });
});
```

```
snippets\unit-test-lab06-step02.txt
```

3. **Save** the file and **verify** the tests pass successfully.

```
Executed 24 of 24 (SUCCESS)
```

✓ You have completed Unit Testing Lab 6: Pipe

Appendices: Optional Labs

Purpose

The remainder of the labs in this manual are not *all* intended to be completed by students during the advanced course. Instructors can choose to include them as demonstrations and/or additional hands-on labs as time allows or questions arise. Students can use them as additional exercises if they finish the labs early or after the course is over to go deeper on topics. For example, some students work is focused on QA/Testing so they are interested in E2E testing with Protractor, however, often developers do not have this interest in the same topic. These optional labs are provided to allow the course to be flexible and meet these differing student needs.

E2E Testing Lab 1: First Test

Objectives

- ☐ Write your first end-to-end test
-

Steps

Write your first end-to-end test

1. Close all your current editors and command prompts/terminals.
2. Open the following directory in your editor as the top level directory. This will be your starting point and working directory for all the E2E testing labs.
 - `code\labs\e2e-lab01\begin\project-manage`

The code is the completed Angular labs up to this point.

3. **Open a command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
4. **Run** the following **command** to install all JavaScript dependencies in this folder

```
npm install
```

5. **Update** HomeComponentComponent to **set** the **title** to **'Home'**.

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

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

@Component({
  selector: 'app-home-container',
  templateUrl: './home-container.component.html',
  styleUrls: ['./home-container.component.css']
})
export class HomeComponentComponent implements OnInit {
  title = '';
  title = 'Home';
  constructor() {}

  ngOnInit() {}
}
```

6. **Delete** the default e2e test **files** generated by the Angular CLI when you create a new project.
 - a. e2e\src\app.e2e.spec.ts
 - b. e2e\src\app.po.ts
7. **Create** an e2e **spec file** to test the default home page. **Add** the following **test code**.

e2e\src\home.e2e-spec.ts

```
import { browser, by, element } from 'protractor';

describe('Home (default)', () => {
  const expectedHeader = 'Home';

  beforeEach(() => {
    browser.get('');
  });

  it('should display header', () => {
    expect(element(by.css('h1')).getText())
      .toEqual(expectedHeader);
  });
});
```

If your editor tries to import the describe or beforeEach functions from selenium-webdriver with the following import line 'import {beforeEach} from "selenium-webdriver/testing"...do not import it. If the import statement gets added simply remove it. The describe function for Jasmine does not require an import.

8. If you don't already have one open, **open** a **command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
9. If not already running, **run** the following **command** to start the backend API

```
npm run api
```

10. **Open another command prompt.**

- a. **Open** a new **command prompt** (Windows) or **terminal** (Mac). Set the directory to **project-manage**.
- b. **Run** the following **command** to run all these processes
 1. the TypeScript compiler
 2. the web server for the application
 3. Protractor

```
npm run e2e
```

11. A browser will open quickly, navigate to the page and then close the browser. You should see a success message similar to the one shown below.

```
Executed 1 of 1 spec SUCCESS ...
```

✓ You have completed E2E Testing Lab 1: First Test

E2E Testing Lab 2: Page Objects

Objectives

- ☐ Use a page object to write an end-to-end test
-

Steps

Use a page object to write an end-to-end test

1. **Create** a new **file** and create a page object for the home page by **adding** the following **code**.

```
e2e\src\home.po.ts
```

```
import { browser, element, by } from 'protractor';

export class HomePage {
  navigateTo() {
    return browser.get('');
  }

  getHeaderText() {
    return element(by.css('h1')).getText();
  }
}
```

2. Update the spec to use the page object.

e2e\src\home.e2e-spec.ts

```
import { browser, by, element } from 'protractor';
import { HomePage } from './home.po';

describe('Home (default)', () => {
  const expectedHeader = 'Home';
  let page: HomePage;

  beforeEach(() => {
    return browser.get('');
    page = new HomePage();
    page.navigateTo();
  });

  it('should display header', () => {
    expect(element(by.css('h1')).getText())
      .toEqual(expectedHeader);
    expect(page.getHeaderText()).toEqual(expectedHeader);
  });
});
```

3. Run the e2e tests.

```
npm run e2e
```

4. You should see the following success message again.

```
Executed 1 of 1 spec SUCCESS ...
```

✓ You have completed E2E Testing Lab 2: Page Objects

E2E Testing Lab 3: Loading Data

Objectives

- ☐ Test a page that loads data from a server
-

Steps

Test a page that loads data from a server

1. **Create** a new **file** and create a page object for the projects page by **adding** the following **code**.

e2e\src\projects.po.ts

```
import { ElementArrayFinder, by, element, browser } from 'protractor';

export class ProjectsPage {
  projectNameHeaders: ElementArrayFinder = element.all(by.css('h5.strong'));

  navigateTo() {
    return browser.get('/projects');
  }
}
```

2. **Create a spec file and use the page object** to test the page.

```
e2e\src\projects.e2e-spec.ts

import { ProjectsPage } from './projects.po';

describe('Projects', () => {
  let page: ProjectsPage;

  beforeEach(() => {
    page = new ProjectsPage();
  });

  it('should have projects', () => {
    page.navigateTo();
    expect(page.projectNameHeaders.count()).toEqual(99);
  });
});
```

3. Run the e2e tests.

```
npm run e2e
```

4. You should see the following success message again.

```
Executed 2 of 2 spec SUCCESS ...
```

✓ You have completed E2E Testing Lab 3: Loading Data

E2E Testing Lab 4: Saving Data

Objectives

- ☐ Test a page that saves data to a server

Steps

1. Update the projects page object.

e2e\src\projects.po.ts

```
import {... , ElementFinder} from 'protractor';

export class ProjectsPage {
  projectNameHeaders: ElementArrayFinder = element.all(by.css('h5.strong'));
  editButton: ElementFinder = element(by.buttonText('Edit'));
  saveButton: ElementFinder = element(by.buttonText('Save'));
  projectForm: ElementFinder = element(by.tagName('form'));
  projectNameInput: ElementFinder = element(by.name('name'));
  firstProjectHeader: ElementFinder = this.projectNameHeaders.first();

  navigateTo() {
    return browser.get('/projects');
  }

  updateName(name: string) {
    const input = this.projectNameInput;
    input.clear().then(() => {
      input.sendKeys(name);
    });
  }
}
```

2. Update the spec file to include tests that edit a project and save the data.

e2e\src\projects.e2e-spec.ts

```
import { ProjectsPage } from './projects.po';
describe('Projects', () => {
  let page: ProjectsPage;

  beforeEach(() => {
    page = new ProjectsPage();
  });

  it('should have projects', () => {
    page.navigateTo();
    expect(page.projectNameHeaders.count()).toEqual(99);
  });

  describe('when editing', () => {
    beforeEach(() => {
      page.navigateTo();
      page.editButton.click();
    });

    it('should show form', () => {
      expect(page.projectForm.isPresent()).toEqual(true);
    });

    it('should save updated project name', () => {
      const updatedName = 'updated project name';
      page.updateName(updatedName);
      page.saveButton.click();
      expect(page.firstProjectHeader.getText()).toEqual(updatedName);
    });
  });
});
```

3. Run the e2e tests.

```
npm run e2e
```

4. You should see the following success message again.

```
Executed 4 of 4 spec SUCCESS ...
```

✓ You have completed E2E Testing Lab 4: Saving Data

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.

```
ng serve -o
```

5. If you are working a lab which requires the backend api (lab 21 or later). 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`
 - `ng 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`