Angular Application Development

Lab Manual

Lab 1

Getting Started with Angular

© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Get a simple Angular application up and running
- Explore different application features

Lab Overview

In this lab, you'll work with an Angular “Hello World” application and get it up and running on your machine. You’ll also explore some of the core Angular concepts.

Estimated Completion Time

25 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Setting Up the Lab Environment

In this exercise, you'll get lab environment setup.

1. Install the required applications and frameworks if they're not already installed on your machine:

    a. **Node.js** – Install from **http://nodejs.org**. You can install the latest version.
    b. **VS Code** – Install from **http://code.visualstudio.com**.
    c. **Course Files** - Download and extract the course files .zip file (refer to the course manual for the link) and extract it directly to your desktop. If you're on Windows, see the first note below.

If you're on Windows it's recommended that you right-click on the .zip file, select Properties from the menu and then "unblock" the .zip before extracting it to your desktop.

VS Code is used throughout the labs because it provides a flexible cross-platform editor and makes it easy to access the command prompt from a project, refactor code and more. However, any text editor can be used.

2. Open a command-prompt and go to the root of the course files for that should now be on your desktop.

3. Run the following command:

```
npm install
```

4. Run the following command to setup the lab files:

```
npm run setup
```

5. You should see output displayed in the console window about folders that were copied.

© Wahlin Consulting: All rights reserved

6. Close the command-prompt window.

7. Continue to the next exercise.

© Wahlin Consulting: All rights reserved

Exercise 2: Running the Application

In this exercise, you'll get the “Hello World” application dependencies installed and run it in the browser.

1. As you work with each lab, the name of the lab (“Getting Started with Angular” in this case) will always match with a folder in the lab files source code. For example, for this lab you’ll find a folder named, **Labs/Getting Started with Angular** in the source code that you extracted to your desktop in the previous exercise. Inside of that folder you’ll find a **Begin** folder where you’ll do the lab work.

2. Open this lab’s **Begin** folder in VS Code.

    a. Windows: File → Open Folder
    b. Mac: File → Open

    
    For this lab, you'll want to open the **Labs/Getting Started with Angular/Begin** folder using the technique above.

3. Open the **package.json** file in the editor and take a moment to look through the different dependencies. Notice that there are several Angular dependencies listed such as:

    a. @angular/common
    b. @angular/compiler
    c. @angular/core

4. Additional Angular dependencies and others such as **RxJS**, **SystemJS** and **Zone.js** are also listed. You’ll learn more about these scripts and the role that they play throughout the course.

5. Locate the **scripts** property in **package.json** and take a moment to look through the different commands such as **tsc**, **tsc:w**, **start**, etc. These commands are used to compile TypeScript code and run the application.

6. Open a command window by **right-clicking BELOW the last file** in the **VS Code Explorer** window (where all the files are displayed on the left) and selecting the appropriate option based on your operating system:

© Wahlin Consulting: All rights reserved

Windows: Open in Command Prompt

Mac: Open in Terminal

Note: You'll be using this procedure several times in future labs so it's something you'll want to get comfortable doing!

7. Ensure that the command prompt is in this lab's **Begin** directory before continuing.

8. Run the following command to install the application dependencies defined in **package.json**:

```
npm install
```

Note: If you get an error about not being able to find package.json then you're running the command in the wrong directory. Double-check that you followed the steps above correctly and that you're running the command from the Begin directory.

Note: If you see colored messages displayed they're normally warning messages that can be ignored.

9. After the installation completes run the following command to build the TypeScript code for the application:

```
npm run tsc:w
```

10. Take a moment to view the output generated in the console window. This command converts the TypeScript code in the application to ES5. It then watches for any files changes.

11. Press **ctrl + c** to exit the current process that's running in the command window.

12. Run the following command to build the code and run the application in the browser:

© Wahlin Consulting: All rights reserved

```
npm start
```

13. Notice that a browser opens but only a header section displays. Content isn't shown because the root component element for the application hasn't been defined in **index.html** (you'll fix that in a moment) and "bootstrapped".

14. Leave the browser open and the console window running.

15. Continue to the next exercise.

© Wahlin Consulting: All rights reserved

Exercise 3: Modifying the Application

In this exercise, you'll walk through the application to learn about key aspects of Angular that will be covered throughout the course. You'll also add root component and bootstrap code.

1. Open **src/app/app.component.ts** in the editor. This code represents an Angular component.

2. Take a moment to explore the following features:
   - The code imports **Component** from **@angular/core**.
   - **Component** is used to define **selector** and **template** properties. Note that the selector property is assigned a value of **app-component** (this will be important later!).
   - A class named **AppComponent** is defined.

3. Open **src/index.html** in the editor.

4. Notice that several scripts are included in the head section of the document. One of these scripts is **System.js** which handles loading ES2015 modules (you'll learn more about modules later in the course).

5. Locate the custom script element that has the **System.import()** call in it. This code starts the application by loading the **app** module.

   
   Where is the "app" module defined? It refers to a folder named "app" that you'll find at src/app in the folder structure. We'll talk more about how System.js and modules work later in the course.

6. Locate the **Root Component Element Goes Here** comment in **index.html** and add the following HTML below it:

```
<app-component>
  Loading...
</app-component>
```


© Wahlin Consulting: All rights reserved

This will cause Angular to load the root component you looked at earlier when the page is displayed in the browser.

7. Save **index.html**.

8. Go back to the browser. Did the page load successfully?

9. The answer is "No"! Although the root component has been added to the page it hasn't been "bootstrapped" yet by Angular.

10. Open **src/app/app.module.ts** in the editor. This is the root module (in other words the main container) for the application. Notice that it imports **AppComponent** toward the top of the code and declares it as the "bootstrap" component for the application using a "bootstrap" property.

11. Open **src/app/main.ts**. This code does the following:

    - Imports several objects from Angular modules using ES2015 import syntax.
    - Imports the custom root module named **AppModule** that you looked at in the previous step.

12. Locate the **Bootstrap Code Goes Here** comment and add the following code after it to "bootstrap" **AppModule**:

```
platformBrowserDynamic().bootstrapModule(AppModule)
  .then((success: any) => console.log('App bootstrapped'))
  .catch((err: any) => console.error(err));
```

The "bootstrap" process lets Angular know about the root component to load when the application runs.

13. Save **main.ts**.

14. Go back to the browser.

© Wahlin Consulting: All rights reserved

15. Now that the application component is "bootstrapped" into Angular it can be used in the index.html page. The page should now display the “Hello World” content correctly below the header. Close the browser when you’re done.

16. Press **ctrl + c** in the command window to stop the server.

17. Close the command window.

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved

Lab 2

Getting Started with TypeScript (Instructor-Led)

Angular Application Development
© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Write a TypeScript class
- Add class members
- Instantiate and use the class

Lab Overview

In this lab, you'll work with key TypeScript features used in Angular applications.

Estimated Completion Time

20 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Creating a TypeScript Class with Members

In this exercise, you'll create a TypeScript class using the TypeScript playground at http://typescriptlang.org.

1. Open your browser and navigate to **http://typescriptlang.org**.

2. Click on **Playground** in the menu bar.

3. Perform the following tasks:
    a. Create a **Person** class.
    b. Add **firstName** and **lastName** properties (both of type **string**).
    c. Add a constructor that takes **fname** and **lname** parameters (both of type **string**). Assign the parameter values to the appropriate properties.
    d. Add a **getFullName()** function that returns **firstName** and **lastName** (concatenate them together with a space between them).
    e. Create a new instance of the **Person** class and pass any name values you'd like into the constructor for first and last name.
    f. Call the person instance's **getFullName()** function and display the value in an alert.
    g. Refactor the constructor so that the **firstName** and **lastName** properties are automatically generated (hint: use public or private in the constructor).

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved

Lab 3

Exploring the Angular JumpStart Application

© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Explore the Angular JumpStart applications that will be used throughout the course
- Install application dependencies
- View the application in the browser and explore functionality

Lab Overview

In this lab, you'll get the Angular JumpStart application up and running and explore some of the key features it offers.

Estimated Completion Time

20 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Exploring and Running the JumpStart Application

In this exercise, you'll explore the Angular JumpStart application and get it up and running.

1. We haven't learned enough about Angular yet to start building a "real" application. However, by walking through some of the highlights of the framework and getting early exposure to key concepts and code that will be covered later in the course, you'll gain a better understanding about how the framework works as we dive into the details later.

   The goal of this lab is to simply explore an application that will be used throughout the rest of the course and to get it up and running. There's no expectation that you understand all of the concepts and code that you'll see at this point (it's only an exploratory lab). You'll learn more about how the application is built throughout the rest of the course.

2. Open the **Angular-JumpStart-master** folder in VS Code. You'll find this folder in the location where you extracted the course lab files and ran the setup process.

3. Take a moment to open and explore the following files:

   a. **package.json** – Contains all of the dependencies needed by the application. Notice that the dependencies are similar to the "Hello World" application you explored earlier.
   b. **src/systemjs.config.js** – Loads required scripts including SystemJS. For now, just look through the file. We'll discuss what's going on soon.
   c. **src/app/customers/customers.component.ts** – Renders the overall "homepage" for the application. Note that core TypeScript features are used such as:
      i. ES2015 modules (several modules are "imported" at the top of the file).
      ii. A decorator named **@Component** (more on this later in the course).
      iii. A class named **CustomersComponent**
   d. **src/app/customers/customers.component.html** – This is the HTML template used to render the "homepage". Take a moment to look at the different

© Wahlin Consulting: All rights reserved

Angular specific binding syntax used such as **[ ]**, **( )**, **{{ }}** and the custom component elements like **<cm-filter-textbox>**, **<cm-customers-card>**, **<cm-customers-grid>**, **<cm-map>** and **<cm-pagination>**. We'll be diving into this binding syntax as well as the various components later in the course.

4. Install the application dependencies and start the application server by performing the following tasks:

    a. Right-click in the **VS Code Explorer** area (below or on the last file) and select the open in terminal/command window option as you did in an earlier lab.
    b. Type the following into the command window:

    ```
    npm install
    ```

    c. After the dependencies are installed type the following command to compile the TypeScript down to ES5 and start the application server:

    ```
    npm start
    ```

5. The application should automatically open in your default browser. If the browser doesn't open, enter the following URL in Chrome:

    ```
    http://localhost:3000
    ```

If Chrome isn't set as your default browser go into its Settings screen and select it as the default. It'll be used throughout the class for debugging purposes.

---

6. Take a few moments to explore the application:
    a. Click the **Card View**, **List View** and **Map View** items to see the different customer views.
    b. Click **Card View** and select a customer by clicking their name. Notice that you're then taken to another view that allows you to see customer details, orders and lets you edit the customer information.
    c. Try to edit a customer and notice that you're taken to a login screen.

© Wahlin Consulting: All rights reserved

d. Login using any email address and a password with at least 6 characters (one of those should be a number).
e. Once you're taken to the customer edit view, make any changes you'd like to the customer and then save them. Click "Customers" on the top navigation bar. The changes you made to the customer should be displayed.
f. Page through the customers by clicking the pager control under the customers.
g. Type into the **filter textbox** in the top-right and notice how the customer data is filtered.

7. Close the browser.

8. Press **ctrl + c** in the command window to stop the server.

9. Close the command window.

10. Throughout the rest of the course you'll be building parts of the Angular JumpStart application to learn more about how Angular components, modules, services and routing work.

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved

Lab 4

Components and Modules

© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Work with Angular component classes and decorators
- Import functionality from modules
- Implement an Angular abstract class
- Use an Input property

Lab Overview

In this lab, you'll practice importing from modules and learn how to work with components, modules, input properties and more.

Estimated Completion Time

30 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Working with Modules and Components

In this exercise, you'll import objects from a module and work with different component features used in the Angular JumpStart application.

1. Open this lab's **Begin** folder in **VS Code**.

Note that if you get stuck at all in the lab, need help with any of the steps, or just want to copy/paste code, you can find the finished source code in the Angular-JumpStart-master folder at the root of the course files.

2. Open a command window for this project using the built-in VS Code functionality. Ensure your command window is pointed to this lab's **Begin** folder.

Hint: Remember that you can right-click in the VS Code Explorer to open a command window as you've done in previous labs.

3. Install the dependencies defined in **package.json** using the **npm** command-line tool.

You've performed this step several times in previous labs. If you don't remember the command to use refer to Lab 1 for details.

4. Run the following command in the command window to start the TypeScript compiler and watch for changes:

```
npm run tsc:w
```


© Wahlin Consulting: All rights reserved

Note: You will see errors displayed at this point. You'll fix those in the following steps.

5. To help you focus on the code, avoid switching back and forth between the lab manual and the code editor, and minimize the amount of typing you must do, the following steps rely on **TODO** comments embedded in the lab files. Please note the following:

    a. Each TODO task is clearly marked in the respective file with "TODO".
    b. Some files have multiple TODO tasks so be careful to ensure that you complete all the tasks before moving on to the next file.

6. Here's an example of what a TODO task will look like (don't do anything with this task – it's just an example ☺).

```
/*

TODO: Adding a Component Decorator

1. Add a Component decorator below this TODO task.

2. Add the following properties into the decorator:

moduleId:    module.id
selector:    'cm-app-component'
templateUrl: 'app.component.html'

*/
```

Note that it's recommended that you leave the TODO comments in the files **even after you've completed them** so that you can refer to them later (if necessary) during the lab to fix any code issues that may come up.

7. Open **src/app/app.component.ts** file and complete all the **TODO** tasks listed. You'll do the following:

© Wahlin Consulting: All rights reserved

- Import the Component decorator symbol
- Define a **@Component** decorator
- Create an **AppComponent** class

8. Open **src/app/app.component.html** file and take a moment to explore the HTML content. The **AppComponent** template acts as the host for the application. It relies on the following components:

    - **<cm-navbar>** provides the navbar at the top of the application
    - **<router-outlet>** provides an area where components can be loaded as the user interacts with the application. It will be discussed more in the routing chapter
    - **<cm-growler>** provides a way to display messages to users as they perform actions
    - **<cm-modal>** provides a way to display a custom modal dialog message to users

9. Open **src/app/app.module.ts** file and complete all the **TODO** tasks listed. You'll do the following:

    - Import symbols
    - Enhance the **@NgModule** properties

10. Open **src/app/main.ts** file and note how **AppModule** is imported and bootstrapped.

11. Open **src/index.html** file and complete all the **TODO** tasks listed. You'll do the following:

    - Add the **AppComponent** selector into the body

12. Open **src/app/customers/customers.component.ts** file and complete all the **TODO** tasks listed. You'll do the following:

    - Add a **templateUrl** property into the **@Component** decorator
    - Implement **OnInit** on the **CustomersComponent** class

© Wahlin Consulting: All rights reserved

- Add code into **ngOnInit()**

14. Ensure that you've saved all your work before continuing.

15. View the TypeScript build output in the console window. Ensure that no errors are displayed before continuing to the next step.

16. Once the project builds successfully in the console window, press **ctrl + c** to stop the current process that's running.

17. Run the following command:

```
npm start
```

18. The customer cards should display successfully in the browser.

If the page doesn't display correctly, right-click on the page and select **Inspect** to get to the Chrome dev tools. Look for any errors displayed in the console (look for a red icon in the upper-right corner of the dev tools window) and fix them as appropriate.

---

19. Close the browser and stop the server (**ctrl + c** in the command window).

20. Close the command window.

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved

Lab 5

Interpolation, Expressions and Pipes

© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Work with interpolation to bind data into a view
- Work with pipes
- Build a custom pipe

Lab Overview

In this lab, you'll bind data into views using interpolation. You'll also work with pipes and see how they can be used to format data.

Estimated Completion Time

30 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Working with Interpolation and Pipes

In this exercise, you'll use interpolation to bind data into a view. You'll also use pipes to format data.

2. Open this lab's **BeginPipesAndBindings** folder in **VS Code**.

3. Install the application dependencies using **npm**.

4. Open the **src/app/app.component.ts** file in the editor. Take a moment to look at the existing code in the component and template.

5. Perform the following tasks in the component template:

    a. Use interpolation to bind any of the component properties into the template.
    b. Experiment with using expressions in the template such as: {{ price + 1 }}
    c. Use pipes to format data. Add the following items into the template to see the various pipes in action. Experiment with changing the values passed to the currency or date pipes (see the Angular docs for additional information on the pipes).
        i. CurrencyPipe {{ price | currency:'USD':true }}
        ii. DatePipe {{ birthday | date:'shortDate' }}
        iii. UpperCasePipe {{ message | uppercase }}
        iv. LowerCasePipe {{ message | lowercase }}
        v. PercentPipe {{ percentage | percent:'2.2' }}

6. Run **npm start** to see the application in the browser. Ensure that the data binding expressions and pipes are working properly in the browser.

7. Close the browser and console windows if everything is working correctly and continue to the next exercise.

© Wahlin Consulting: All rights reserved

Exercise 2: Creating a Custom Pipe

In this exercise, you'll create a custom Angular pipe that can be used to capitalize the first letter of a string and explore other custom pipes used in the application.

1. Open this lab's **Begin** folder in VS Code

 Note that this folder is different from the one used in the previous exercise.

---

2. Install the application dependencies using **npm**.

3. Open **data/customers.json** and notice how some of the **firstName** and **lastName** properties aren't cased properly. Also, notice how some of the data values in the file have spaces around them.

   For this application, we'll assume that the data is provided by another department or company and can't be changed. To fix these problems you'll work with custom pipes in this lab so you can see how to build them and how they can be applied.

4. Open the **src/app/shared/pipes/capitalize.pipe.ts** file and complete the TODO tasks. You'll do the following:

   a. Import **Pipe** and **PipeTransform** from **@angular/core**.
   b. Create a **CapitalizePipe** class.
   c. Add a **transform()** function that converts the first letter of words to uppercase.
   d. Add the Pipe decorator to the custom pipe class.

5. Open the **src/app/shared/pipes/trim.pipe.ts** file and take a moment to explore the code. Note the name of the pipe defined in the **@Pipe** decorator as well as what the **transform()** function does.

6. Open **src/app/shared/shared.module.ts** and complete the TODO tasks. You'll do the following:

© Wahlin Consulting: All rights reserved

a. Import **CapitalizePipe** and **TrimPipe**.
b. Add **CapitalizePipe** and **TrimPipe** to the module's **exports** and **declarations** properties.

8. Continue to the next exercise.

© Wahlin Consulting: All rights reserved

Exercise 3: Using Data Binding Syntax and Pipes in Templates

In this exercise, you'll work with Angular data binding syntax and use the custom pipes discussed in the previous exercise.

1. Open **src/app/customers/customers-card.component.html** and complete the **TODO** tasks. You'll do the following:

    a. Bind component properties into the template.
    b. Use the capitalize and trim pipes that you worked with earlier.

2. Open **src/app/customers/customers-grid.component.html** and complete the TODO tasks. You'll do the following:

    a. Add pipes to data binding expressions.

3. Save all the modified files and run **npm start** to load the application in the browser.

4. Ensure that customers are displayed correctly when the page loads. Switch to the "grid" view and ensure data is displayed correctly there as well.

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved

Lab 6

Component Properties and Angular Directives

© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Work with component property and event data binding syntax
- Use Angular directives
- Use child components in a parent component

Lab Overview

In this lab, you'll use Angular data binding syntax to bind data into a template view and handle events. You'll also use Angular directives to render data. Finally, you'll add child components into a parent component template.

Estimated Completion Time

20 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Using Data Binding Syntax in Templates

In this exercise, you'll work with Angular data binding syntax and directives.

1. Open this lab's **Begin** folder in VS Code.

2. Install the application's dependencies using **npm**.

3. Open **src/app/customers/customers-card.component.ts** file and complete all the **TODO** tasks listed. You'll do the following:

    a. Add an **@Input()** property into the existing class
    b. Note the **TrackByService** that is injected into its constructor that is assigned to a property named **trackbyService**. This property will be used to enhance the functionality of the **ngFor** directive you'll work with later in this lab.

4. Open **src/app/customers/customers-card.component.html** and complete the **TODO** tasks. You'll do the following:

    a. Use the **ngFor** directive to iterate through customer data and render properties using interpolation.
    b. Add **trackBy** to the **ngFor** expression to ensure that each item has an id (doing this can improve performance when the items change).
    c. Use the **ngIf** directive to show/hide a div in the template.

5. Open **src/app/customers/customers-grid.component.ts** and take a moment to look through the code. Notice the following features:

    a. The component exposes a **customers** Input property.
    b. It has a **sort()** function that handles sorting the customer data. Sorting is performed by a **SorterService** object that is injected into the component constructor (you'll learn more about services in a later lab).

6. Open **src/app/customers/customers-grid.component.html** and complete the TODO tasks. You'll do the following:

© Wahlin Consulting: All rights reserved

a. Use event binding syntax to "hook" an event to a component function.
b. Use **ngFor** to iterate through customer data and render it in the template.
c. Add **trackBy** to the **ngFor** expression.

7. Save all the files before continuing to the next exercise.

© Wahlin Consulting: All rights reserved

Exercise 2: Using Child Components

In this exercise, you'll add the CustomersCardComponent and CustomersGridComponent components into a parent component named CustomersComponent.

1. Open **src/app/customers/customers.component.ts** and explore the code. Note the following:

    a. The component contains a **filteredCustomers** property. You'll be passing this property value to the child components later in this exercise.
    b. The component has **displayModeEnum** and **displayMode** properties defined. The **displayModeEnum** property is an enum type that can be found at the bottom of the component code. It's used to handle switching between the card, grid and map modes in the application.

2. Open **src/app/customers/customers.component.html** and complete the **TODO** tasks. You'll complete the following tasks:

    a. Add **CustomersCardComponent** and **CustomersGridComponent** as children of **CustomersComponent**.
    b. Use property binding syntax to bind to the child components' **customers** property.
    c. Use property binding syntax to bind to the **hidden** property of each child component to show/hide it as appropriate.
    d. Explore paging and mapping components used in the view.

3. Save all the files that you've modified up to this point in the lab.

4. Run **npm start** to start the server and display the application in the browser.

5. The customer cards should display successfully on the homepage. Click on the **List View** toolbar item and you should see the customers displayed in a grid.

Note: If the page doesn't display correctly then right-click in the page and select Inspect from the menu. Locate the console area and look for any errors that need to be fixed.

© Wahlin Consulting: All rights reserved

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved

Lab 7

Services, Providers and Http

© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Work with services, providers and modules
- Import and use Http functionality
- Work with RxJS observables and subscriptions

Lab Overview

Services play an important role in Angular applications and provide a great way to consolidate re-useable code. In this lab, you'll learn how to work with services, setup required providers in a module and use the Http object to communicate with the server. You'll also work with observables and see how components can subscribe to an observable.

Estimated Completion Time

30 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Using the Http Client

In this exercise, you'll add Http functionality into an existing service that is used throughout the application to provide data. You'll also work with RxJS observables.

1. Open this lab's **Begin** folder in VS Code.

2. Install the application dependencies using **npm**.

3. Open **src/app/shared/interfaces.ts** and take a moment to look at the included interfaces. Several of the interfaces defined in the file are used in **data.service.ts** and in components used throughout the application.

4. Open **src/app/core/services/data.service.ts** and complete the **TODO** tasks. You will do the following:

    a. Import the **Injectable**, **Http** and **Response** objects.
    b. Explore RxJS imports.
    c. Add the **Injectable** decorator.
    d. Inject **Http** into the service.
    e. Define an **Observable<T>** return type on a service function.
    f. Use the **Http** object to call the server.

5. Save your work and continue to the next exercise.

© Wahlin Consulting: All rights reserved

Exercise 2: Adding Services into the Core Module

In this exercise, you'll add services into a core module that can be used throughout the entire application.

1. Open **src/app/core/core.module.ts** and complete the **TODO** tasks. You will do the following:

    a. Import **HttpModule** so that **Http** client and other related functionality is available.
    b. Import **DataService** and add it to the module's providers.
    c. Add **HttpModule** to the decorator's **imports** and **exports** properties.
    d. Add **DataService** to the decorator's **providers** property.
    e. Explore the role of **EnsureModuleLoadedOnceGuard** in the module.

2. Open **src/app/app.module.ts** and complete the TODO tasks. You will do the following:

    a. Import **CoreModule**.
    b. Add **CoreModule** into the root module's imports property so that everything exported by **CoreModule** is available to the entire application.

3. Save your work and continue to the next exercise.

© Wahlin Consulting: All rights reserved

Exercise 3: Subscribing to Observables

In this exercise, you'll enhance add code into a component to subscribe to Observables returned from DataService.

1. Open **src/app/customers/customers.component.ts** and complete the TODO tasks. Note that you need to be in the “customers” folder (not “customer”). You will do the following:

    a. Import the **DataService** class and custom interfaces.
    b. Inject **DataService** into the component's constructor
    c. Subscribe to an observable and assign the received data to a **customers** and **filteredCustomers** property in the component.

2. Open **src/app/customer/customer-orders.component.ts**. Locate the call to **dataService.getCustomer()** and notice how the **subscribe()** function is used there as well.

3. Save your work and run **npm start** in the console to start the server and display the application in the browser.

4. The customer cards should display successfully on the homepage. If not, fix any errors that you find.

5. Right-click on the page in **Chrome** and select **Inspect** from the menu.

6. Once the **Chrome Developer Tools** window opens perform the following steps:

    a. Select the **Sources** tab.
    b. In the tree-view that displays, open the **app/core/services** folder and double-click the **data.service.ts** file to open it.
    c. Locate the **getCustomersPage()** function and set a breakpoint somewhere inside of the map() function body (click the appropriate line number to the left of the code that is in the arrow function).
    d. Refresh the page in the browser and your breakpoint should be hit as the **Http** client calls the server and returns data to the service.
    e. Take a moment to mouse over the response object you receive in the **map()** function and look at the data.

© Wahlin Consulting: All rights reserved

f. If time permits step through the code until the data is passed to the component.

7. If time permits, explore the **map()** function and other observable functions that can be used with RxJS at http://rxmarbles.com.

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved

Lab 8

Working with Routing

© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Define application routes
- Define child routes
- Link to routes using the routerLink directive
- Use the router-outlet routerLinkActive directives
- Retrieve route parameters

Lab Overview

In this lab, you'll learn how to define routes in an application (including child routes) and how they can be used to load and display different components. You'll then add the routerLink and router-outlet directives into component templates to allow users to navigate to specific routes. Finally, you'll learn how to define and retrieve route parameters.

Estimated Completion Time

20 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Defining Customers Application Routes

In this exercise, you'll define routes that are used by the application. You'll then define additional child routes on components.

1. Open this lab's **Begin** folder in VS Code.

2. Install the application dependencies using **npm**.

3. Open **src/app/customers/customers-routing.module.ts** and complete the **TODO** tasks. You will do the following:

    a. Import **RouterModule** and **Routes** symbols.
    b. Define a custom route that loads **CustomersComponent**.
    c. Work with a custom routing module that uses **@NgModule**.
    d. Add related components into the module.

4. Open **src/app/customers/customers.module.ts** and complete the **TODO** tasks. You will do the following:

    a. Import **CustomersRoutingModule**.
    b. Add the routing module to the feature module's **imports** property.
    c. Add components imported into the routing module into the feature module's **declarations** property.

5. Open **src/app/customer/customer-routing.module.ts** and complete the **TODO** tasks. You will do the following:

    a. Define a custom route that includes child routes.

6. Open **src/app/customer/customer.module.ts** and note how **CustomerRoutingModule** is used in the **NgModule** decorator properties. The

© Wahlin Consulting: All rights reserved

pattern that you see in these two modules can be used in all feature modules in an application.

7. Open **src/app/app-routing.module.ts** and complete the TODO tasks. You will do the following:

    a. Import routing symbols.
    b. Add default routes.

8. Open **src/app/app.module.ts** and note that several feature modules such as **CustomersModule**, **CustomerModule** and others are imported. Each of these modules has their own routes defined in a child routing module.

9. Continue to the next exercise.

© Wahlin Consulting: All rights reserved

Exercise 2: Using Router Directives

In this exercise, you'll use router directives to allow users to navigate to different components.

1. Open **src/app/app.component.html** and complete the TODO task. You will do the following:

    a. Add **<router-outlet>** into the template.

2. Open **src/app/customer/customer.component.html** and complete the TODO tasks. You will do the following:

    a. Add the **routerLink** and **routerLinkActive** directives into the template.
    b. Add a child **router-outlet** directive into the template.
    c. Add the **routerLink** directive to allow navigation to a parent route.

3. Open **src/app/customer/customer-details.component.ts** and complete the TODO tasks. You will do the following:

    a. Inject **ActivatedRoute** into the component's constructor.
    b. Retrieve a route parameter and use it to make a service call.

4. Save all the files that you've modified up to this point in the lab.

5. Run **npm start** to start the server and display the application in the browser.

6. The customer cards should display successfully on the homepage.

7. Click on a customer name and navigate to a child route to see their details. As you interact with the application, take a moment to notice the URL that is displayed in the browser.

When you're on the homepage you should see **http://localhost:3000/customers** whereas when you select a customer you'll

© Wahlin Consulting: All rights reserved

see a route such as **http://localhost:3000/customers/1/details**. In this case "customers/1" is the root route and "details" is the child route.

8. While viewing an individual customer click the **Edit Customer** toolbar item. Are you taken to the edit screen? Currently there is no code to block the user from editing a customer even if they're not logged in. You'll see how to fix that in a later lab.

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved

Lab 9

Route Guards and Lazy Loading

© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Work with route guards
- Convert eager loaded routes to lazy loaded routes

Lab Overview

In this lab, you'll learn how to convert eager loaded routes to lazy loaded routes. You'll also work with an apply route guards to route definitions.

Estimated Completion Time

20 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Add Route Guards and Define Lazy Loaded Routes

In this exercise, you'll enhance a canActivate route guard and define lazy loaded routes in the application's root routing module.

1. Open this lab's **Begin** folder in VS Code.

2. Install the application dependencies using **npm**.

3. Open **src/app/customers/customers-routing.module.ts** and complete the **TODO** task. You will do the following:

    a. Remove the existing route path.

4. Open **src/app/customer/can-activate.guard.ts** and complete the TODO task. You will do the following:

    a. Import **CanActivate**.
    b. Implement **CanActivate** on the guard class.
    c. Add a **canActivate()** function and code to check **AuthService** to see if the user has logged in or not.

5. Open **src/app/customer/can-deactivate.guard.ts** and take a moment to review the **canDeactivate()** function. This function will determine if the user can leave the current route based on if they have unsaved (dirty) changes in the form.

6. Open **src/app/customer/customer-routing.module.ts** and complete the **TODO** tasks. You will do the following:

    a. Remove the existing route path.
    b. Add route guards to a child route.
    c. Define providers for the route guards.

© Wahlin Consulting: All rights reserved

7. Open **src/app/app-routing.module.ts** and complete the TODO tasks. You will do the following:

    a. Import routing symbols.
    b. Define lazy loaded routes.
    c. Add the routes into the app module and define a preloading strategy.

8. Open **src/app/app.module.ts** and complete the TODO tasks. You will do the following:

    a. Remove unneeded imports
    b. Remove unneeded modules

9. Save all the files that you've modified up to this point in the lab.

10. Run **npm start** to start the server and display the application in the browser.

11. The homepage should load successfully in the browser. As the page loads, feature modules are being loaded asynchronously in the background (check out the Network tab the Chrome Developer Tools if you'd like to see this process).

12. Click on a customer name that's displayed on the homepage to go to the details for that customer.

13. Now click on the **Edit Customer** toolbar item.

14. You should be taken to the login screen by the **CanActivateGuard** since you haven't logged into the application.

15. Login using any email address and a password that's 6 characters long (include one number). After logging in you should be taken to the customer edit screen.

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved

Lab 10

Template and Reactive Forms

© Wahlin Consulting: All rights reserved

Lab Objectives

In this lab, you will:

- Use form directives and local variables
- Use the ngModel directive and "two-way" binding syntax
- Learn how to show/hide validation messages
- Learn how CSS classes can be used to highlight a control when it is valid or invalid
- Use Reactive Forms directives with FormBuilder

Lab Overview

Forms are an important part of any application that needs to capture information from end users. In this lab, you'll learn how to use template-driven and reactive forms to setup bindings between properties and form input controls and how validation errors can be displayed.

Estimated Completion Time

30 minutes

© Wahlin Consulting: All rights reserved

Lab Exercises

Exercise 1: Retrieving a Customer and Working with a Template-Driven Form

In this exercise, you'll add code to retrieve a customer that will be displayed in a template-driven form.

1. Open this lab's **Begin** folder in VS Code.

2. Install the application dependencies using **npm**.

3. Open **src/app/shared/shared.module.ts** and note that **FormsModule** is imported and exported. **SharedModule** is imported into the customer feature module (**src/app/customer/customer.module.ts**) which makes template-driven form functionality available to use throughout the feature.

4. Open **src/app/core/services/data.service.ts** and find the **getCustomer()** function. Notice that it accepts a customer ID and then makes a call to the server to retrieve the customer.

5. Open **src/app/customer/customer-edit.component.ts** and complete the **TODO** tasks. You will do the following:

    a. Examine the properties exposed by the component.
    b. Add code to grab the customer id from a route parameter.
    c. Add code to update a customer using **DataService**.
    d. Add code to navigate to the root route if the user presses the cancel button.
    e. Add code into the component that the **canDeactivate** routing guard can check to determine if a route can be navigated to or not.

© Wahlin Consulting: All rights reserved

6. Open **src/app/customer/customer-edit.component.html** and complete the TODO tasks. You will do the following:

    a. Add Angular directives to the <form> tag.
    b. Modify form input controls to support Angular template-driven form functionality and two-way binding.
    c. Add code to show/hide validation error messages.
    d. Modify buttons to handle click events and disable them when the form isn't valid.

7. Open **src/app/customer/customer-edit.component.css** and locate the **.customer-form .ng-invalid** and **.customer-form .ng-valid** classes. Notice that they will set the left border to red or green respectively.

8. Save all the files that you've modified up to this point in the lab.

9. Continue to the next exercise.

© Wahlin Consulting: All rights reserved

Exercise 2: Working with Reactive Forms

In this exercise, you'll add code to build a custom FormGroup and enhance the reactive form template for the login feature of the application.

1. Open **src/app/login/login.component.ts** and complete the TODO tasks. You will do the following:

    a. Import reactive forms functionality and a custom validation service.
    b. Inject **FormBuilder** into the constructor.
    c. Use **FormBuilder** to create a custom **FormGroup** that will be bound to the reactive forms template.

2. Open **src/app/login/login.component.html** and complete the TODO tasks. You will do the following:

    a. Add Angular directives to the <form> tag.
    b. Modify form input controls to bind them to **FormGroup** controls.
    c. Explore code to show/hide validation error messages.

3. Run **npm start** to start the server and display the application in the browser.

4. Go to the login view. Login by entering any email address and password (the password must be at least 6 digits and contain at least one number in it) and ensure that your reactive form works correctly.

5. Logout by clicking the **Logout** link in the navigation bar. Go back to the login view and attempt to enter an invalid email address (for example: test@test) and invalid password (leave out a number or add a space). Do the validation error messages display correctly?

© Wahlin Consulting: All rights reserved

6. Go back to the homepage and click one of the edit icons in the upper-right corner of any customer card. Login using any email address and a password (the password must be at least 6 digits and contain at least one number in it).

7. Once the customer edit form displays, ensure that the form works correctly and that validation errors display if an input control isn't valid. Try to leave a textbox blank and ensure that the validation message displays correctly.

This Lab is Complete - Please Stop

© Wahlin Consulting: All rights reserved