

Build your first Angular app

This tutorial consists of lessons that introduce the Angular concepts you need to know to start coding in Angular.

You can do as many or as few as you would like and you can do them in any order.

Before you start

For the best experience with this tutorial, review these requirements to make sure you have what you need to be successful.

Your experience

The lessons in this tutorial assume that you have experience with the following:

1. Created an HTML web page by editing the HTML directly.
2. Programmed web site content in JavaScript.
3. Read Cascading Style Sheet (CSS) content and understand how selectors are used.
4. Used command-line instructions to perform tasks on your computer.

Your equipment

These lessons can be completed by using a local installation of the Angular tools or by using StackBlitz in a web browser. Local Angular development can be completed on Windows, MacOS or Linux based systems.

Conceptual preview of your first Angular app

The lessons in this tutorial create an Angular app that lists houses for rent and shows the details of individual houses. This app uses features that are common to many Angular apps.

**Acme Fresh Start Housing**

Chicago, IL

[Learn More >](#)**A113 Transitional Housing**

Santa Monica, CA

[Learn More >](#)**Warm Beds Housing Support**

Juneau, AK

[Learn More >](#)**Homesteady Housing**

Chicago, IL

[Learn More >](#)**Happy Homes Group**

Gary, IN

[Learn More >](#)**Hopeful Apartment Group**

Oakland, CA

[Learn More >](#)**Seriously Safe Towns**

Oakland, CA

[Learn More >](#)**Hopeful Housing Solutions**

Oakland, CA

[Learn More >](#)**Seriously Safe Towns**

Oakland, CA

[Learn More >](#)**Capital Safe Towns**

Portland, OR

[Learn More >](#)

Local development environment

Perform these steps in a command-line tool on the computer you want to use for this tutorial.


Step 1 - Identify the version of `node.js` that Angular requires

Angular requires an active LTS or maintenance LTS version of Node. Let's confirm your version of `node.js`. For information about specific version requirements, see the engines property in the [package.json file](#).

From a **Terminal** window:

1. Run the following command: `node --version`
2. Confirm that the version number displayed meets the requirements.

Step 2 - Install the correct version of `node.js` for Angular

If you do not have a version of `node.js` installed, please follow the [directions for installation on nodejs.org](#) 

Step 3 - Install the latest version of Angular



With `node.js` and `npm` installed, the next step is to install the `Angular CLI` which provides tooling for effective Angular development.

npm install -g @angular/cli

From a **Terminal** window run the following command: `npm install -g @angular/cli`.

Step 4 - Install integrated development environment (IDE)

You are free to use any tool you prefer to build apps with Angular. We recommend the following:

1. [Visual Studio Code](#) 
 2. As an optional, but recommended step you can further improve your developer experience by installing the [Angular Language Service](#) 
-

Lesson review

In this lesson, you learned about the app that you build in this tutorial and prepared your local computer to develop Angular apps.

Next steps

- [First Angular app lesson 1 - Hello world](#)
-

More information

For more information about the topics covered in this lesson, visit:

- [What is Angular](#)
- [Angular CLI Reference](#)

Last reviewed on Wed Jul 12 2023



Lesson 1: Hello world



This first lesson serves as the starting point from which each lesson in this tutorial adds new features to build a complete Angular app. In this lesson, we'll update the application to display the famous text, "Hello World".

Estimated time: ~10 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

The updated app you have after this lesson confirms that you and your IDE are ready to begin creating an Angular app.

Step 1 - Test the default app

In this step, after you download the default starting app, you build the default Angular app. This confirms that your development environment has what you need to continue the tutorial.

In the **Terminal** pane of your IDE:

1. In your project directory, navigate to the `first-app` directory.
2. Run this command to install the dependencies needed to run the app.

```
npm install
```

npm install

3. Run this command to build and serve the default app.

```
ng serve
```

The app should build without errors.

4. In a web browser on your development computer, open `http://localhost:4200`.
5. Confirm that the default web site appears in the browser.
6. You can leave `ng serve` running as you complete the next steps.

Step 2 - Review the files in the project

In this step, you get to know the files that make up a default Angular app.

In the **Explorer** pane of your IDE:

1. In your project directory, navigate to the `first-app` directory.
2. Open the `src` directory to see these files.
 - a. In the file explorer, find the Angular app files (`/src`).
 - i. `index.html` is the app's top level HTML template.
 - ii. `style.css` is the app's top level style sheet.
 - iii. `main.ts` is where the app start running.
 - iv. `favicon.ico` is the app's icon, just as you would find in any web site.
 - b. In the file explorer, find the Angular app's component files (`/app`).
 - i. `app.component.ts` is the source file that describes the `app-root` component. This is the top-level Angular component in the app. A component is the basic building block of an Angular application. The component description includes the component's code, HTML template, and styles, which can be described in this file, or in separate files.

In this app, the styles are in a separate file while the component's code and HTML template are in this file.
 - ii. `app.component.css` is the style sheet for this component.
 - iii. New components are added to this directory.
 - c. In the file explorer, find the image directory (`/assets`) that contains images used by the app.
 - d. In the file explorer, find the files and directories that an Angular app needs to build and run, but they are not files that you normally interact with.
 - i. `.angular` has files required to build the Angular app.
 - ii. `.e2e` has files used to test the app.
 - iii. `.node_modules` has the node.js packages that the app uses.
 - iv. `angular.json` describes the Angular app to the app building tools.
 - v. `package.json` is used by `npm` (the node package manager) to run the finished app.
 - vi. `tsconfig.*` are the files that describe the app's configuration to the TypeScript compiler.

After you have reviewed the files that make up an Angular app project, continue to the next step.

Step 3 - Create Hello World

In this step, you update the Angular project files to change the displayed content.

In your IDE:

1. Open `first-app/src/index.html`.

2. In `index.html`, replace the `<title>` element with this code to update the title of the app.

Replace in src/index.html

```
<title>Homes</title>
```

Then, save the changes you just made to `index.html`.

3. Next, open `first-app/src/app/app.component.ts`.

4. In `app.component.ts`, in the `@Component` definition, replace the `template` line with this code to change the text in the app component.

Replace in src/app/app.component.ts

```
template: '<h1>Hello world!</h1>'
```

5. In `app.component.ts`, in the `AppComponent` class definition, replace the `title` line with this code to change the component title.

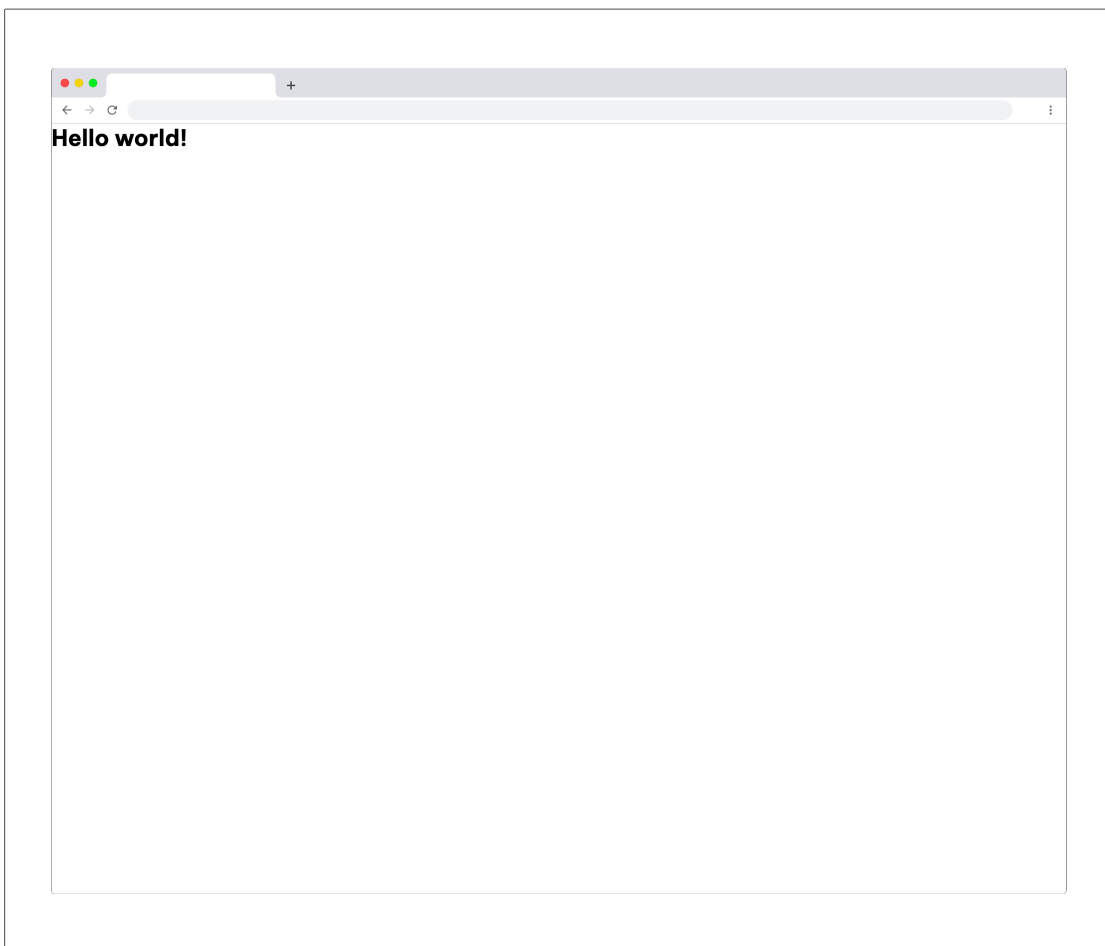
Replace in src/app/app.component.ts

```
title = 'homes';
```

Then, save the changes you made to `app.component.ts`.

6. If you stopped the `ng serve` command from step 1, in the **Terminal** window of your IDE, run `ng serve` again.

7. Open your browser and navigate to `localhost:4200` and confirm that the app builds without error and displays *Hello world* in the title and body of your app:



Lesson review

In this lesson, you updated a default Angular app to display *Hello world*. In the process, you learned about the `ng serve` command to serve your app locally for testing.

If you have any trouble with this lesson, review the completed code for it in the [live example](#) / [download example](#).

Next steps

First Angular app [lesson 2 - Creating Components](#)

More information

For more information about the topics covered in this lesson, visit:

- [Angular Components](#)
- [Creating applications with the Angular CLI](#)



Lesson 2: Create Home component



This tutorial lesson demonstrates how to create a new component for your Angular app.

Estimated time: ~10 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

Your app has a new component: HomeComponent.

Conceptual preview of Angular components

Angular apps are built around components, which are Angular's building blocks. Components contain the code, HTML layout, and CSS style information that provide the function and appearance of an element in the app. In Angular, components can contain other components. An app's functions and appearance can be divided and partitioned into components.

In Angular, components have metadata that define its properties. When you create your `HomeComponent`, you use these properties:

- `selector`: to describe how Angular refers to the component in templates.
- `standalone`: to describe whether the component requires a `NgModule`.
- `imports`: to describe the component's dependencies.
- `template`: to describe the component's HTML markup and layout.
- `styleUrls`: to list the URLs of the CSS files that the component uses in an array.

selector.

template:
style urls

[Learn more about Components.](#)

Step 1 - Create the HomeComponent

In this step, you create a new component for your app.

In the **Terminal** pane of your IDE:

1. In your project directory, navigate to the `first-app` directory.

2. Run this command to create a new `HomeComponent`

```
ng generate component home --standalone --inline-template --skip-tests
```

3. Run this command to build and serve your app.

```
ng serve
```

4. Open a browser and navigate to `http://localhost:4200` to find the application.

5. Confirm that the app builds without error.

It should render the same as it did in the previous lesson because even though you added a new component, you haven't included it in any of the app's templates, yet.

6. Leave `ng serve` running as you complete the next steps.

Step 2 - Add the new component to your app's layout

In this step, you add the new component, `HomeComponent` to your app's root component, `AppComponent`, so that it displays in your app's layout.

In the **Edit** pane of your IDE:

1. Open `app.component.ts` in the editor.

2. In `app.component.ts`, import `HomeComponent` by adding this line to the file level imports.

Import HomeComponent in src/app/app.component.ts

```
import { HomeComponent } from './home/home.component';
```

3. In `app.component.ts`, in `@Component`, update the `imports` array property and add `HomeComponent`.

Replace in src/app/app.component.ts

```
imports: [  
  HomeComponent,  
],
```

4. In `app.component.ts`, in `@Component`, update the `template` property to include the following HTML code.

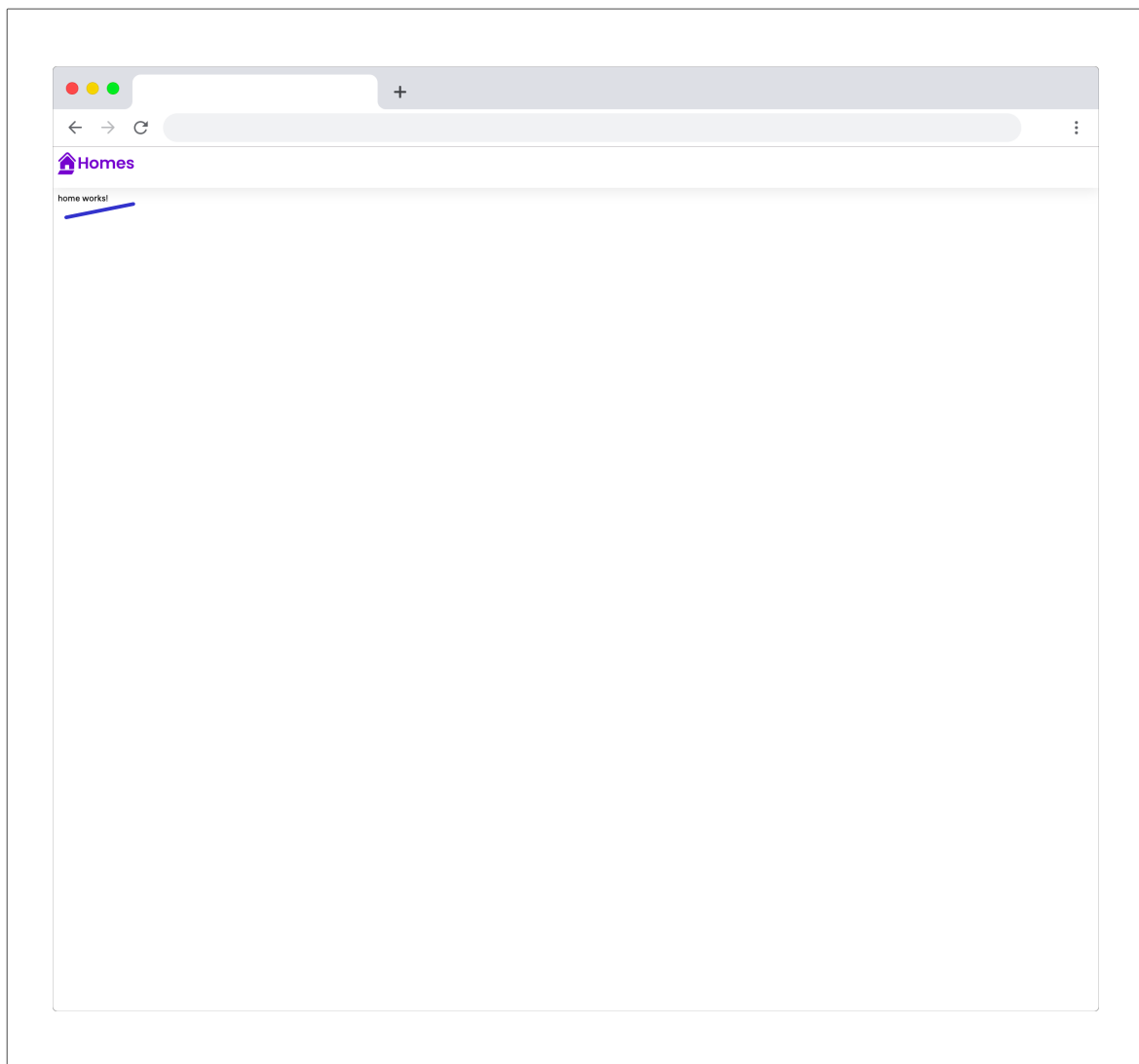
Replace in src/app/app.component.ts

```
template: `  
  <main>  
    <header class="brand-name">  
        
    </header>  
    <section class="content">  
      <app-home></app-home>  
    </section>  
  </main>  
`
```

5. Save your changes to `app.component.ts`.

6. If `ng serve` is running, the app should update. If `ng serve` is not running, start it again. *Hello world* in your app should change to *home works!* from the `HomeComponent`.

7. Check the running app in the browser and confirm that the app has been updated.



Step 3 - Add features to HomeComponent

In this step you add features to `HomeComponent`.

In the previous step, you added the default `HomeComponent` to your app's template so its default HTML appeared in the app. In this step, you add a search filter and button that is used in a later lesson. For now, that's all that `HomeComponent` has. Note that, this step just adds the search elements to the layout without any functionality, yet.

In the **Edit** pane of your IDE:

1. In the `first-app` directory, open `home.component.ts` in the editor.
2. In `home.component.ts`, in `@Component`, update the `template` property with this code.

Replace in `src/app/home/home.component.ts`

```
template: `
  <section>
    <form>
      <input type="text" placeholder="Filter by city">
      <button class="primary" type="button">Search</button>
    </form>
  </section>
`;
```

3. Next, open `home.component.css` in the editor and update the content with these styles.

Replace in `src/app/home/home.component.css`

```
results {
  display: grid;
  column-gap: 14px;
  row-gap: 14px;
  grid-template-columns: repeat(auto-fill, minmax(400px, 400px));
  margin-top: 50px;
  justify-content: space-around;
}

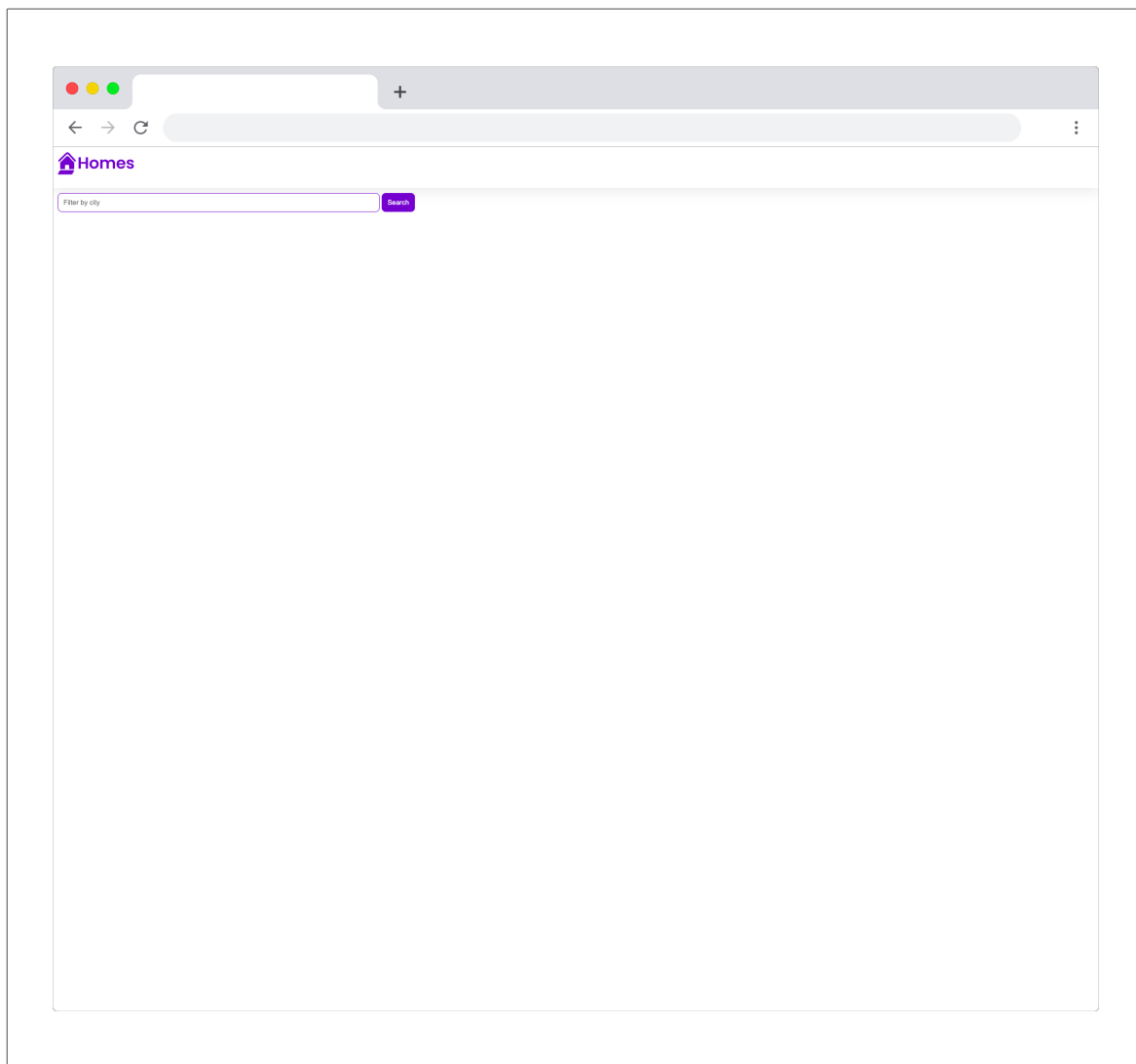
input[type="text"] {
  border: solid 1px var(--primary-color);
  padding: 10px;
  border-radius: 8px;
  margin-right: 4px;
  display: inline-block;
  width: 30%;
}

button {
  padding: 10px;
  border: solid 1px var(--primary-color);
  background: var(--primary-color);
  color: white;
  border-radius: 8px;
}

@media (min-width: 500px) and (max-width: 768px) {
  .results {
    grid-template-columns: repeat(2, 1fr);
  }
  input[type="text"] {
    width: 70%;
  }
}

@media (max-width: 499px) {
  .results {
    grid-template-columns: 1fr;
  }
}
```

4. Confirm that the app builds without error. You should find the filter query box and button in your app and they should be styled. Correct any errors before you continue to the next step.



Lesson review

In this lesson, you created a new component for your app and gave it a filter edit control and button.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [First Angular app lesson 3 - Create the application's HousingLocation component](#)

More information

For more information about the topics covered in this lesson, visit:

- [ng generate component](#)
- [Component](#) reference
- [Angular components overview](#)



Lesson 3: Create the application's HousingLocation component

This tutorial lesson demonstrates how to add the `HousingLocation` component to your Angular app.

Estimated time: ~10 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

- Your app has a new component: `HousingLocationComponent` and it displays a message confirming that the component was added to your application.

Step 1 - Create the HousingLocationComponent

In this step, you create a new component for your app.

In the **Terminal** pane of your IDE:

1. In your project directory, navigate to the `first-app` directory.

2. Run this command to create a new `HousingLocationComponent`

```
ng generate component housingLocation --standalone --inline-template --skip-tests
```

3. Run this command to build and serve your app.

```
ng serve
```

4. Open a browser and navigate to `http://localhost:4200` to find the application.

5. Confirm that the app builds without error.

It should render the same as it did in the previous lesson because even though you added a new component, you haven't included it in any of the app's templates, yet.

6. Leave `ng serve` running as you complete the next steps.

Step 2 - Add the new component to your app's layout

In this step, you add the new component, `HousingLocationComponent` to your app's `HomeComponent`, so that it displays in your app's layout.

In the Edit pane of your IDE:

1. Open `home.component.ts` in the editor.
2. In `home.component.ts`, import `HousingLocationComponent` by adding this line to the file level imports.

Import HousingLocationComponent in src/app/home/home.component.ts

```
import { HousingLocationComponent } from '../housing-location/housing-location.component';
```

3. Next update the `imports` property of the `@Component` metadata by adding `HousingLocationComponent` to the array.

Add HousingLocationComponent to imports array in src/app/home/home.component.ts

```
imports: [  
  CommonModule,  
  HousingLocationComponent  
],
```

4. Now the component is ready for use in the `template` for the `HomeComponent`. Update the `template` property of the `@Component` metadata to include a reference to the `<app-housing-location>` tag.

Add housing location to the component template in src/app/home/home.component.ts

```
template: `  
  <section>  
    <form>  
      <input type="text" placeholder="Filter by city">  
      <button class="primary" type="button">Search</button>  
    </form>  
  </section>  
  <section class="results">  
    <app-housing-location></app-housing-location>  
  </section>  
`;
```

Step 3 - Add the styles for the component

In this step, you will copy over the pre-written styles for the `HousingLocationComponent` to your app so that the app renders properly.

1. Open `src/app/housing-location/housing-location.css`, and paste the styles below into the file:

Add CSS styles to housing location to the component in `src/app/housing-location/housing-location.component.css`

```
.listing {
  background: var(--accent-color);
  border-radius: 30px;
  padding-bottom: 30px;
}

.listing-heading {
  color: var(--primary-color);
  padding: 10px 20px 0 20px;
}

.listing-photo {
  height: 250px;
  width: 100%;
  object-fit: cover;
  border-radius: 30px 30px 0 0;
}

.listing-location {
  padding: 10px 20px 20px 20px;
}

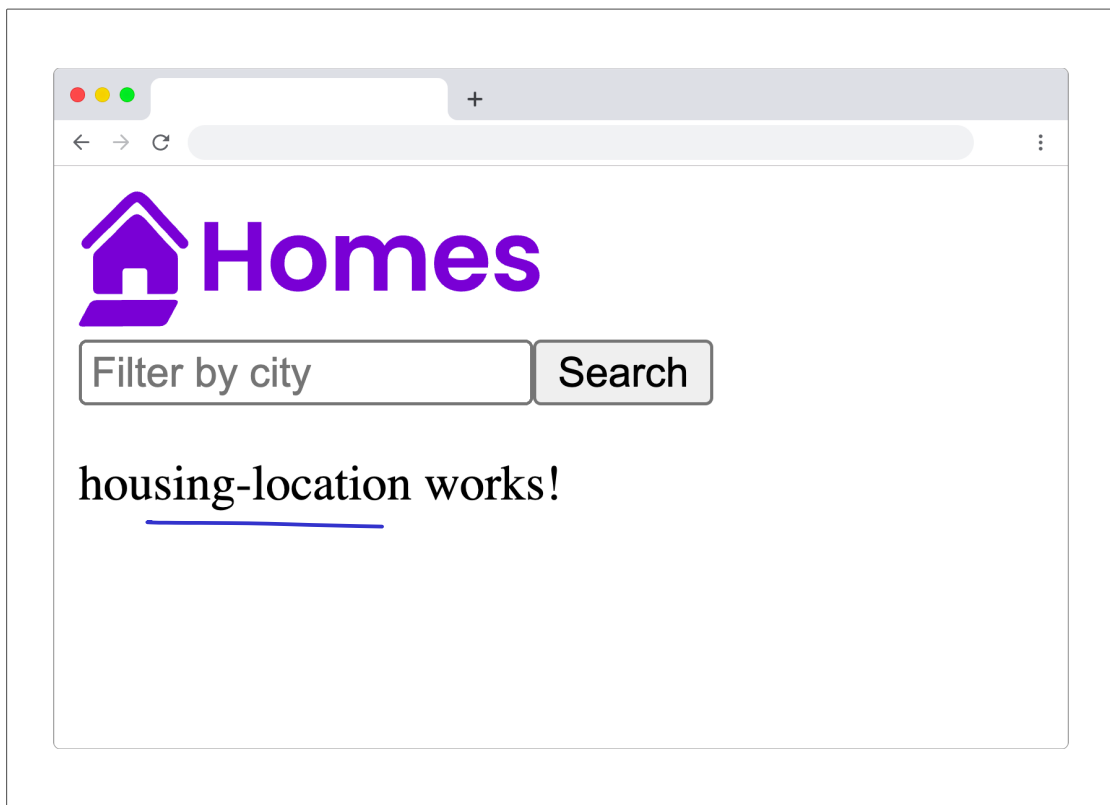
.listing-location::before {
  content: url("/assets/location-pin.svg") / "";
}

section.listing a {
  padding-left: 20px;
  text-decoration: none;
  color: var(--primary-color);
}

section.listing a::after {
  content: "\203A";
  margin-left: 5px;
}
```

content: "\203A"

2. Save your code, return to the browser and confirm that the app builds without error. You should find the message "housing-location works!" rendered to the screen. Correct any errors before you continue to the next step.



Lesson review

In this lesson, you created a new component for your app and added it to the app's layout.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [First Angular app lesson 4 - Add a housing location interface to the application](#)

Last reviewed on Tue Jul 11 2023



First Angular app lesson 4 - Creating an interface



This tutorial lesson demonstrates how to create an interface and include it in a component of your app.

Estimated time: ~10 minutes

Starting code: [live example](#) / [download example](#)


Completed code: [live example](#) / [download example](#)

What you'll learn

- Your app has a new interface that it can use as a data type.
- Your app has an instance of the new interface with sample data.

mock ..

Conceptual preview of interfaces

Interfaces  are custom data types for your app.

*interface: custom data type part
of app*

Angular uses TypeScript to take advantage of working in a strongly typed programming environment. Strong type checking reduces the likelihood of one element in your app sending incorrectly formatted data to another. Such type-mismatch errors are caught by the TypeScript compiler and many such errors can also be caught in your IDE.

In this lesson, you'll create an interface to define properties that represent data about a single housing location.

Step 1 - Create a new Angular interface

This step creates a new interface in your app.

In the **Terminal** pane of your IDE:

1. In your project directory, navigate to the `first-app` directory.
2. In the `first-app` directory, run this command to create the new interface.

```
ng generate interface housinglocation
```

3. Run `ng serve` to build the app and serve it to `http://localhost:4200`.
4. In a browser, open `http://localhost:4200` to see your app.
5. Confirm that the app builds without error. Correct any errors before you continue to the next step.

Step 2 - Add properties to the new interface

This step adds the properties to the interface that your app needs to represent a housing location.

1. In the **Terminal** pane of your IDE, start the `ng serve` command, if it isn't already running, to build the app and serve it to `http://localhost:4200`.
2. In the **Edit** pane of your IDE, open the `src/app/housinglocation.ts` file.
3. In `housinglocation.ts`, replace the default content with the following code to make your new interface to match this example.

Update `src/app/housinglocation.ts` to match this code

```
export interface HousingLocation {  
  id: number;  
  name: string;  
  city: string;  
  state: string;  
  photo: string;  
  availableUnits: number;  
  wifi: boolean;  
  laundry: boolean;  
}
```

```
export interface to {  
  id: number;  
  name: string;  
  wifi: boolean;  
}
```

4. Save your changes and confirm the app does not display any errors. Correct any errors before you continue to the next step.

At this point, you've defined an interface that represents data about a housing location including an id, name, and location information.

Step 3 - Create a test house for your app

You have an interface, but you aren't using it yet.

In this step, you create an instance of the interface and assign some sample data to it. You won't see this sample data appear in your app yet. There are a few more lessons to complete before that happens.

1. In the **Terminal** pane of your IDE, run the `ng serve` command, if it isn't already running, to build the app and serve your app to `http://localhost:4200`.
2. In the **Edit** pane of your IDE, open `src/app/home/home.component.ts`.
3. In `src/app/home/home.component.ts`, add this import statement after the existing `import` statements so that `HomeComponent` can use the new interface.

Import HomeComponent in src/app/home/home.component.ts

```
import { HousingLocation } from '../housinglocation';
```

4. In `src/app/home/home.component.ts`, replace the empty `export class HomeComponent {}` definition with this code to create a single instance of the new interface in the component.

Add sample data to src/app/home/home.component.ts

```
export class HomeComponent {  
  readonly baseUrl = 'https://angular.io/assets/images/tutorials/faa';  
  
  housingLocation: HousingLocation = {  
    id: 9999,  
    name: 'Test Home',  
    city: 'Test city',  
    state: 'ST',  
    photo: `${this.baseUrl}/example-house.jpg`,  
    availableUnits: 99,  
    wifi: true,  
    laundry: false,  
  };  
}
```

readonly baseUrl = '...';
housingLocation: H... = {
 (obj)
 3;

5. Confirm that your `home.component.ts` file matches like this example.

src/app/home/home.component.ts

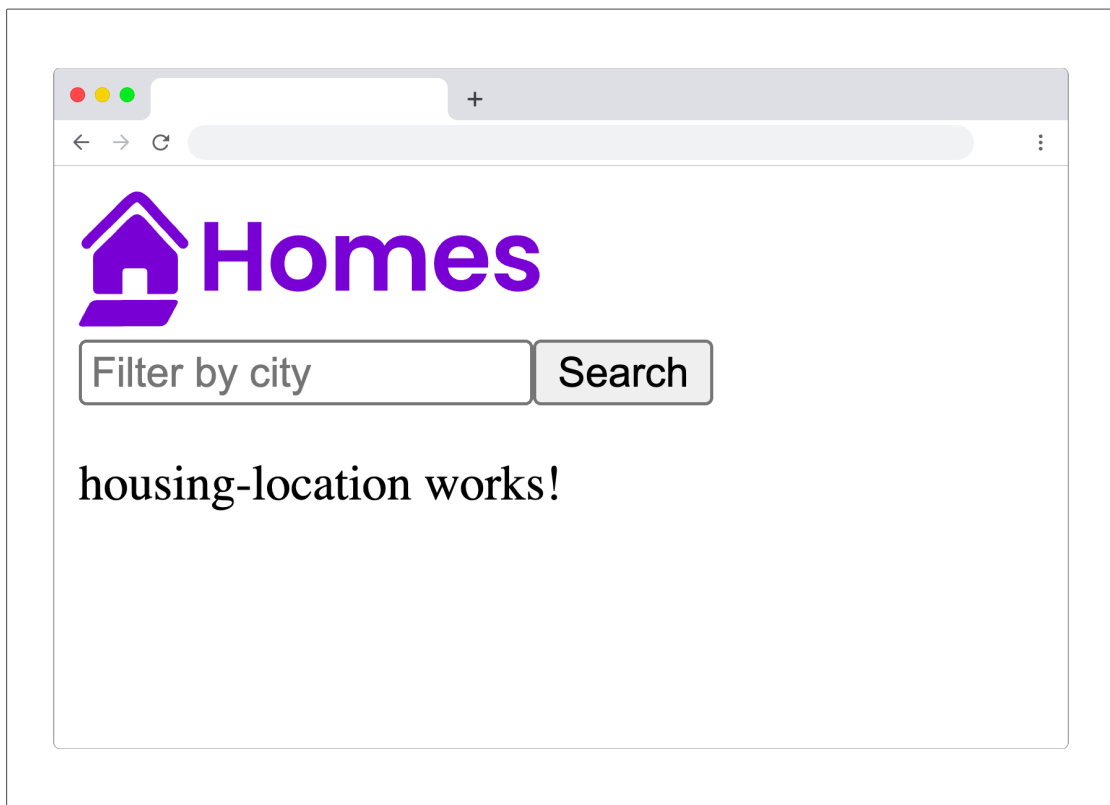
```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HousingLocationComponent } from '../housing-location/housing-
location.component';
import { HousingLocation } from '../housinglocation';

@Component({
  selector: 'app-home',
  standalone: true,
  imports: [
    CommonModule,
    HousingLocationComponent
  ],
  template: `
    <section>
      <form>
        <input type="text" placeholder="Filter by city">
        <button class="primary" type="button">Search</button>
      </form>
    </section>
    <section class="results">
      <app-housing-location></app-housing-location>
    </section>
  `,
  styleUrls: ['./home.component.css'],
})
export class HomeComponent {
  readonly baseUrl = 'https://angular.io/assets/images/tutorials/faa';

  housingLocation: HousingLocation = {
    id: 9999,
    name: 'Test Home',
    city: 'Test city',
    state: 'ST',
    photo: `${this.baseUrl}/example-house.jpg`,
    availableUnits: 99,
    wifi: true,
    laundry: false,
  };
}
```

By adding the `housingLocation` property of type `HousingLocation` to the `HomeComponent` class, we're able to confirm that the data matches the description of the interface. If the data didn't satisfy the description of the interface, the IDE has enough information to give us helpful errors.

6. Save your changes and confirm the app does not have any errors. Open the browser and confirm that your application still displays the message "housing-location works!"



7. Correct any errors before you continue to the next step.

Lesson review

In this lesson, you created an interface that created a new data type for your app. This new data type makes it possible for you to specify where `HousingLocation` data is required. This new data type also makes it possible for your IDE and the TypeScript compiler can ensure that `HousingLocation` data is used where it's required.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [Lesson 5 - Add an input parameter to the component](#)

More information

For more information about the topics covered in this lesson, visit:

- [ng generate interface](#)
- [ng generate](#)

Lesson 5: Add an input parameter to the component

This tutorial lesson demonstrates how to create a component `@Input()` and use it to pass data to a component for customization.

Estimated time: ~10 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

Padre ^{info} → hijo

What you'll learn

Your app's `HousingLocationComponent` template has a `HousingLocation` property to receive input.

Conceptual preview of Inputs

Inputs allow components to share data. The direction of the data sharing is from parent component to child component.

In this lesson, you'll define `@Input()` properties in the `HousingLocationComponent` component which will enable you to customize the data displayed in the component.

Learn more in the [Sharing data between child and parent directives and components](#) guide.

Step 1 - Import the Input decorator

This step imports the `Input` decorator into the class.

In the code editor:

1. Navigate to `src/app/housing-location/housing-location.component.ts`
2. Update the file imports to include `Input` and `HousingLocation`:

Import `HousingLocationComponent` and `Input` in `src/app/housing-location/housing-location.component.ts`

```
import { Component, Input } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HousingLocation } from '../housinglocation';
```

Step 2 - Add the Input property

1. In the same file, add a property called `housingLocation` of type `HousingLocation` to the `HousingLocationComponent` class. Add an `!` after the property name and prefix it with the `@Input()` decorator:
decorator:

```
Import HousingLocationComponent and Input in src/app/housing-location/housing-location.component.ts
```

```
export class HousingLocationComponent {  
  @Input() housingLocation!: HousingLocation;  
}
```

@Input() housingLocation!;
↳ not null

You have to add the `!` because the input is expecting the value to be passed. In this case, there is no default value. In our example application case we know that the value will be passed in - this is by design. The exclamation point is called the non-null assertion operator and it tells the TypeScript compiler that the value of this property won't be null or undefined.

2. Save your changes and confirm the app does not have any errors.
3. Correct any errors before you continue to the next step.

Lesson review

In this lesson, you created a new property decorated with the `@Input()` decorator. You also used the non-null assertion operator to notify the compiler that the value of the new property won't be `null` or `undefined`.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [Lesson 6 - Add a property binding to an component's template](#)

For more information about the topics covered in this lesson, visit:

- [Sharing data between child and parent directives and components](#)

Last reviewed on Tue Jul 11 2023



Lesson 6 - Add a property binding to a component's template



This tutorial lesson demonstrates how to add property binding to a template and use it to pass dynamic data to components.

Estimated time: ~10 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

- Your app has data bindings in the `HomeComponent` template.
- Your app sends data from the `HomeComponent` to the `HousingLocationComponent`.

Conceptual preview of Inputs

In this lesson, you'll continue the process of sharing data from the parent component to the child component by binding data to those properties in the template using property binding.

Property binding enables you to connect a variable to an `Input` in an Angular template. The data is then dynamically bound to the `Input`.

For a more in depth explanation, please refer to the [Property binding](#) guide.

Step 1 - Update tag in the `HomeComponent` template

This step adds property binding to the `<app-housing-location>` tag.

In the code editor:

1. Navigate to `src/app/home/home.component.ts`
2. In the template property of the `@Component` decorator, update the code to match the code below:

Add housingLocation property binding

```
<app-housing-location [housingLocation]="housingLocation"></app-housing-  
location>
```

When adding a property binding to a component tag, we use the `[attribute] = "value"` syntax to notify Angular that the assigned value should be treated as a property from the component class and not a string value.

The value on the right handside is the name of the property from the `HomeComponent`.

Step 2 - Confirm the code still works

1. Save your changes and confirm the app does not have any errors.
2. Correct any errors before you continue to the next step.

Lesson review

In this lesson, you added a new property binding and passed in a reference to a class property. Now, the `HousingLocationComponent` has access to data that it can use to customize the component's display.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [Lesson 7 - Add an interpolation to a component's template](#)

For more information about the topics covered in this lesson, visit:

- [Property binding](#)



Lesson 7 - Add an interpolation to a component's template

This tutorial lesson demonstrates how to add interpolation to Angular templates in order to display dynamic data in a template.

Estimated time: ~10 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

- Your app will display interpolated values in the `HousingLocationComponent` template.
- Your app will render a housing location data to the browser.

Conceptual preview of interpolation

In this step you will display values (properties and `Input` values) in a template using interpolation.

Using the `{{ expression }}` in Angular templates, you can render values from properties, `Inputs` and valid JavaScript expressions.

For a more in depth explanation, please refer to the [Displaying values with interpolation](#) guide.

Step 1 - Update `HousingLocationComponent` template to include interpolated values

This step adds new HTML structure and interpolated values in the `HousingLocationComponent` template.

In the code editor:

1. Navigate to `src/app/housing-location/housing-location.component.ts`
2. In the template property of the `@Component` decorator, replace the existing HTML markup with the following code:

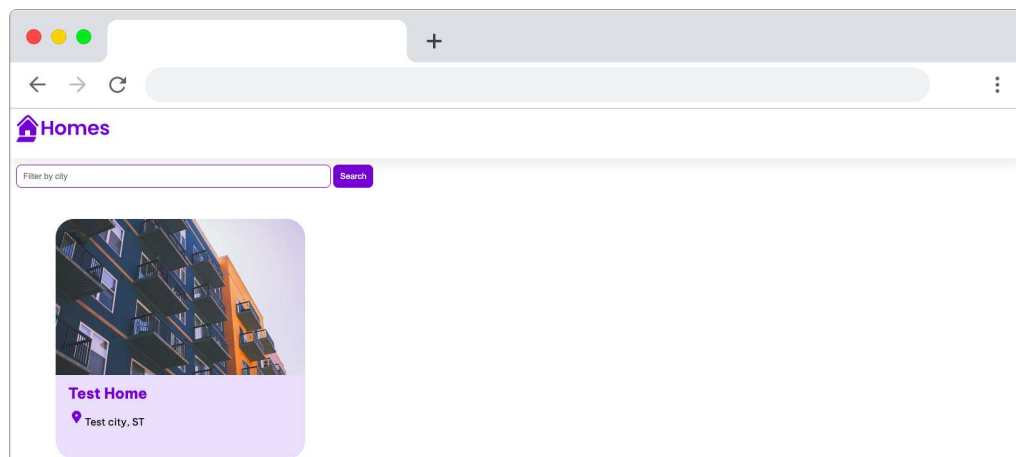
Update HousingLocationComponent template

```
template: `
  <section class="listing">
    <img class="listing-photo" [src]="housingLocation.photo" alt="Exterior photo
of {{housingLocation.name}}">
    <h2 class="listing-heading">{{ housingLocation.name }}</h2>
    <p class="listing-location">{{ housingLocation.city}}, {{housingLocation.state
}}</p>
  </section>
`;
```

In this updated template code you have used property binding to bind the `housingLocation.photo` to the `src` attribute. The `alt` attribute uses interpolation to give more context to the alt text of the image. You use interpolation to include the values for `name`, `city` and `state` of the `housingLocation` property.

Step 2 - Confirm the changes render in the browser

1. Save all changes.
2. Open the browser and confirm that the app renders the photo, city and state sample data.



Lesson review

In this lesson, you added a new HTML structure and used Angular template syntax to render values in the `HousingLocation` template. Now, you have two important skills:

- [passing data](#) to components
- [Interpolating](#) values into a template

With these skills, your app can now share data and display dynamic values in the browser. Great work so far.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [Lesson 8 - Use *ngFor to list objects in component](#)

For more information about the topics covered in this lesson, visit:

- [Displaying values with interpolation](#)
- [Template syntax](#)

Last reviewed on Tue Jul 11 2023



Lesson 8: Use *ngFor to list objects in component



This tutorial lesson demonstrates how to use `ngFor` directive in Angular templates in order to display dynamically repeated data in a template.

Estimated time: ~10 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

- You will have added a data set to the app
- Your app will display a list of elements from the new data set using `ngFor`

Conceptual preview of ngFor

In Angular, `ngFor` is a specific type of `directive` used to dynamically repeat data in a template. In plain JavaScript you would use a for loop - ngFor provides similar functionality for Angular templates.

You can utilize `ngFor` to iterate over arrays and even asynchronous values. In this lesson, you'll add a new array of data to iterate over.

For a more in depth explanation, please refer to the [Built-in directives](#) guide.

Step 1 - Add housing data to the HomeComponent

In the `HomeComponent` there is only a single housing location. In this step, you will add an array of `HousingLocation` entries.

1. In `src/app/home/home.component.ts`, remove the `housingLocation` property from the `HomeComponent` class.

2. Update the `HomeComponent` class to have a property called `housingLocationList`. Update your code to match the following code:

Add housingLocationList property

```
export class HomeComponent {  
  readonly baseUrl = 'https://angular.io/assets/images/tutorials/faa';  
  
  housingLocationList: HousingLocation[] = [  
    {  
      id: 0,  
      name: 'Acme Fresh Start Housing',  
      city: 'Chicago',  
      state: 'IL',  
      photo: `${this.baseUrl}/bernard-hermant-CLKGGwIBTaY-unsplash.jpg`,  
      availableUnits: 4,  
      wifi: true,  
      laundry: true  
    },  
    {  
      id: 1,  
      name: 'A113 Transitional Housing',  
      city: 'Santa Monica',  
      state: 'CA',  
      photo: `${this.baseUrl}/brandon-griggs-wR11KBaB86U-unsplash.jpg`,  
      availableUnits: 0,  
      wifi: false,  
      laundry: true  
    },  
    {  
      id: 2,  
      name: 'Warm Beds Housing Support',  
      city: 'Juneau',  
      state: 'AK',  
      photo: `${this.baseUrl}/i-do-nothing-but-love-lAyXd11-Wmc-unsplash.jpg`,  
      availableUnits: 1,  
      wifi: false,  
      laundry: false  
    },  
    {  
      id: 3,  
      name: 'Homesteady Housing',  
      city: 'Chicago',  
      state: 'IL',  
      photo: `${this.baseUrl}/ian-macdonald-W8z6aiwfi1E-unsplash.jpg`,  
      availableUnits: 1,  
      wifi: true,  
      laundry: false  
    }  
  ]  
}
```

```
},
{
  id: 4,
  name: 'Happy Homes Group',
  city: 'Gary',
  state: 'IN',
  photo: `${this.baseUrl}/krzysztof-hepner-978RAXoXnH4-unsplash.jpg`,
  availableUnits: 1,
  wifi: true,
  laundry: false
},
{
  id: 5,
  name: 'Hopeful Apartment Group',
  city: 'Oakland',
  state: 'CA',
  photo: `${this.baseUrl}/r-architecture-JvQ0Q5IkeMM-unsplash.jpg`,
  availableUnits: 2,
  wifi: true,
  laundry: true
},
{
  id: 6,
  name: 'Seriously Safe Towns',
  city: 'Oakland',
  state: 'CA',
  photo: `${this.baseUrl}/phil-hearing-IYfp2Ixe9nM-unsplash.jpg`,
  availableUnits: 5,
  wifi: true,
  laundry: true
},
{
  id: 7,
  name: 'Hopeful Housing Solutions',
  city: 'Oakland',
  state: 'CA',
  photo: `${this.baseUrl}/r-architecture-GGupkreKwxA-unsplash.jpg`,
  availableUnits: 2,
  wifi: true,
  laundry: true
},
{
  id: 8,
  name: 'Seriously Safe Towns',
  city: 'Oakland',
```

```
        state: 'CA',
        photo: `${this.baseUrl}/saru-robert-9rP3mxf8qWI-unsplash.jpg`,
        availableUnits: 10,
        wifi: false,
        laundry: false
    },
    {
        id: 9,
        name: 'Capital Safe Towns',
        city: 'Portland',
        state: 'OR',
        photo: `${this.baseUrl}/webaliser-_TPTXZd9m0o-unsplash.jpg`,
        availableUnits: 6,
        wifi: true,
        laundry: true
    }
];
```

Do not remove the `@Component` decorator, you will update that code in an upcoming step.

Step 2 - Update the HomeComponent template to use `ngFor`

Now the app has a dataset that you can use to display the entries in the browser using the `ngFor` directive.

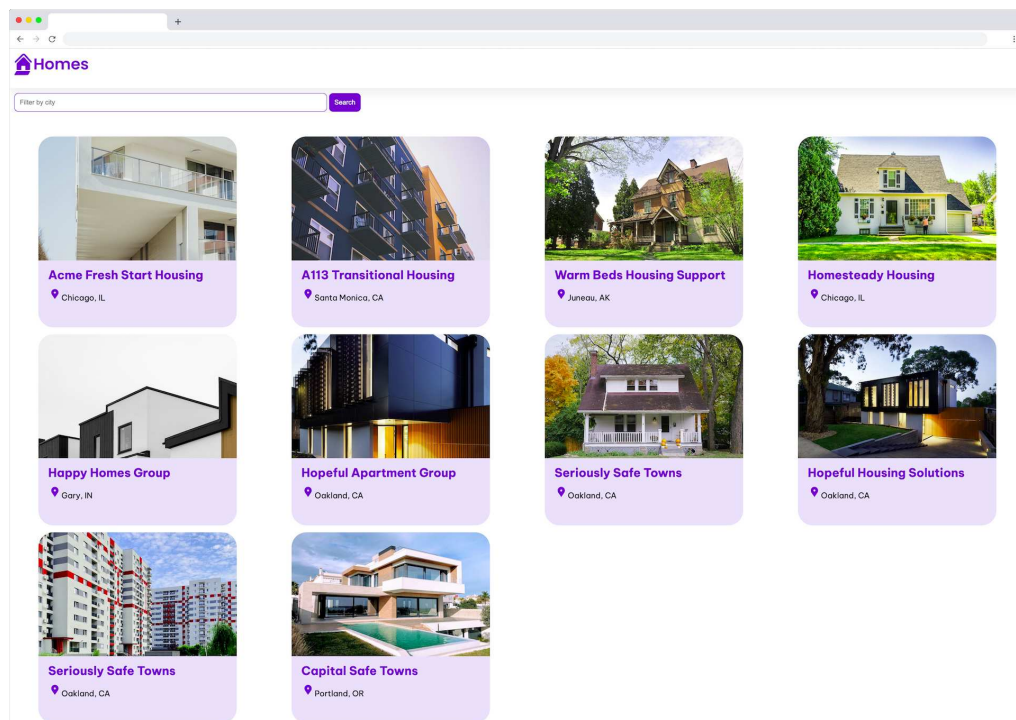
1. Update the `<app-housing-location>` tag in the template code to this:

Add ngFor to HomeComponent template

```
<app-housing-location
  *ngFor="let housingLocation of housingLocationList"
  [housingLocation]="housingLocation">
</app-housing-location>
```

Note, in the code `[housingLocation] = "housingLocation"` the `housingLocation` value now refers to the variable used in the `ngFor` directive. Before this change, it referred to the property on the `HomeComponent` class.

2. Save all changes.
3. Refresh the browser and confirm that the app now renders a grid of housing locations.



Lesson review

In this lesson, you used the `ngFor` directive to repeat data dynamically in Angular templates. You also added a new array of data to be used in the Angular app. The application now dynamically renders a list of housing locations in the browser.

The app is taking shape, great job.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [Lesson 9 - Add a service to the application](#)

For more information about the topics covered in this lesson, visit:

- [Structural Directives](#)
- [ngFor guide](#)
- [ngFor](#)

Last reviewed on Tue Jul 11 2023



Lesson 09: Angular services



This tutorial lesson demonstrates how to create an Angular service and use dependency injection to include it in your app.

Estimated time: ~15 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

Your app has a service to serve the data to your app. At the end of this lesson, the service reads data from local, static data. In a later lesson, you'll update the service to get data from a web service.

Conceptual preview of services

This tutorial introduces Angular services and dependency injection.

Angular services

Angular services provide a way for you to separate Angular app data and functions that can be used by multiple components in your app. To be used by multiple components, a service must be made injectable. Services that are injectable and used by a component become dependencies of that component. The component depends on those services and can't function without them.

Dependency injection

Dependency injection is the mechanism that manages the dependencies of an app's components and the services that other components can use.

Step 1 - Create a new service for your app

This step creates an injectable service for your app.

In the **Terminal** pane of your IDE:

1. In your project directory, navigate to the `first-app` directory.
2. In the `first-app` directory, run this command to create the new service.

```
ng generate service housing --skip-tests
```

--skip-tests

3. Run `ng serve` to build the app and serve it to `http://localhost:4200`.
4. Confirm that the app builds without error. Correct any errors before you continue to the next step.

Step 2 - Add static data to the new service

This step adds some sample data to your new service. In a later lesson, you'll replace the static data with a web interface to get data as you might in a real app. For now, your app's new service uses the data that has, so far, been created locally in `HomeComponent`.

In the Edit pane of your IDE:

1. In `src/app/home/home.component.ts`, from `HomeComponent`, copy the `housingLocationList` variable and its array value.
2. In `src/app/housing.service.ts`:
 - a. Inside the `HousingService` class, paste the variable that you copied from `HomeComponent` in the previous step.
 - b. Inside the `HousingService` class, paste these functions after the data you just copied. These functions allow dependencies to access the service's data.

Service functions in `src/app/housing.service.ts`

```
getAllHousingLocations(): HousingLocation[] {  
  return this.housingLocationList;  
}  
  
getHousingLocationById(id: number): HousingLocation | undefined {  
  return this.housingLocationList.find(housingLocation => housingLocation.id  
    === id);  
}
```

You will need these functions in a future lesson. For now, it is enough to understand that these functions return either a specific `HousingLocation` by id or the entire list.

- c. Add a file level import for the `HousingLocation`.

Import `HousingLocation` type in `src/app/housing.service.ts`

```
import { HousingLocation } from './housinglocation';
```

3. Confirm that the app builds without error. Correct any errors before you continue to the next step.

Step 3 - Inject the new service into HomeComponent

This step injects the new service into your app's `HomeComponent` so that it can read the app's data from a service. In a later lesson, you'll replace the static data with a live data source to get data as you might in a real app.

In the Edit pane of your IDE, in `src/app/home/home.component.ts`:

1. At the top of `src/app/home/home.component.ts`, add the `inject` to the items imported from `@angular/core`. This will import the `inject` function into the `HomeComponent` class.

Update to `src/app/home/home.component.ts`

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

2. Add a new file level import for the `HousingService`:

Add import to `src/app/home/home.component.ts`

```
import { HousingService } from '../housing.service';
```

3. From `HomeComponent`, delete the `housingLocationList` array entries and assign `housingLocationList` the value of empty array (`[]`). In a few steps you will update the code to pull the data from the `HousingService`.
4. In `HomeComponent`, add the following code to inject the new service and initialize the data for the app. The `constructor` is the first function that runs when this component is created. The code in the `constructor` will assign the `housingLocationList` the value returned from the call to `getAllHousingLocations`.

Initialize data from service in `src/app/home/home.component.ts`

```
housingLocationList: HousingLocation[] = [];  
housingService: HousingService = inject(HousingService);  
  
constructor() {  
  this.housingLocationList = this.housingService.getAllHousingLocations();  
}
```

5. Save the changes to `src/app/home/home.component.ts` and confirm your app builds without error. Correct any errors before you continue to the next step.

Lesson review

In this lesson, you added an Angular service to your app and injected it into the `HomeComponent` class. This compartmentalizes how your app gets its data. For now, the new service gets its data from a static array of data. In a later lesson, you'll refactor the service to get its data from an API endpoint.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [Lesson 10 - Add routes to the application](#)
-

More information

For more information about the topics covered in this lesson, visit:

- [Creating an injectable service](#)
- [Dependency injection in Angular](#)
- [ng generate service](#)
- [ng generate](#)

Last reviewed on Sat Jul 15 2023



Lesson 10: Add routes to the application



This tutorial lesson demonstrates how to add routes to your app.

Estimated time: ~15 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

At the end of this lesson your application will have support for routing.

Conceptual preview of routing

This tutorial introduces routing in Angular. Routing is the ability to navigate from one component in the application to another. In [Single Page Applications \(SPA\)](#), only parts of the page are updated to represent the requested view for the user.

The [Angular Router](#) enables users to declare routes and specify which component should be displayed on the screen if that route is requested by the application.

In this lesson, you will enable routing in your application to navigate to the details page.

Step 1 - Create a default details component

1. From the terminal, enter the following command to create the `DetailsComponent`:

```
ng generate component details --standalone --inline-template --skip-tests
```

This component will represent the details page that provides more information on a given housing location.

Step 2 - Add routing to the application

1. In the `src/app` directory, create a file called `routes.ts`. This file is where we will define the routes in the application.

2. In `main.ts`, make the following updates to enable routing in the application:

a. Import the routes file and the `provideRouter` function:

Import routing details in src/main.ts

```
import { provideRouter } from '@angular/router';  
import routeConfig from './app/routes';
```

b. Update the call to `bootstrapApplication` to include the routing configuration:

Add router configuration in src/main.ts

```
bootstrapApplication(AppComponent,  
  {  
    providers: [  
      provideProtractorTestingSupport(),  
      provideRouter(routeConfig)  
    ]  
  })  
).catch(err => console.error(err));
```

3. In `src/app/app.component.ts`, update the component to use routing:

- a. Add a file level import for `RouterModule`:

Import RouterModule in src/app/app.component.ts

```
import { RouterModule } from '@angular/router';
```

- b. Add `RouterModule` to the `@Component` metadata imports

Import RouterModule in src/app/app.component.ts

```
imports: [  
  HomeComponent,  
  RouterModule,  
],
```

- c. In the `template` property, replace the `<app-home></app-home>` tag with the `<router-outlet>` directive and add a link back to the home page. Your code should match this code:

Add router-outlet in src/app/app.component.ts

```
template: `  
  <main>  
    <a [routerLink]="['/']">  
      <header class="brand-name">  
          
      </header>  
    </a>  
    <section class="content">  
      <router-outlet></router-outlet>  
    </section>  
  </main>  
`,
```

Step 3 - Add route to new component

In the previous step you removed the reference to the `<app-home>` component in the template. In this step, you will add a new route to that component.

1. In `routes.ts`, perform the following updates to create a route.

- a. Add a file level imports for the `HomeComponent`, `DetailsComponent` and the `Routes` type that you'll use in the route definitions.

Import components and Routes

```
import { Routes } from '@angular/router';
import { HomeComponent } from '../home/home.component';
import { DetailsComponent } from '../details/details.component';
```

- b. Define a variable called `routeConfig` of type `Routes` and define two routes for the app:

Add routes to the app

```
const routeConfig: Routes = [
  {
    path: '',
    component: HomeComponent,
    title: 'Home page'
  },
  {
    path: 'details/:id',
    component: DetailsComponent,
    title: 'Home details'
  }
];

export default routeConfig;
```

The entries in the `routeConfig` array represent the routes in the application. The first entry navigates to the `HomeComponent` whenever the url matches `''`. The second entry uses some special formatting that will be revisited in a future lesson.

2. Save all changes and confirm that the application works in the browser. The application should still display the list of housing locations.

Lesson review

In this lesson, you enabled routing in your app as well as defined new routes. Now your app can support navigation between views. In the next lesson, you will learn to navigate to the "details" page for a given housing location.

You are making great progress with your app, well done.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [Lesson 11 - Integrate details page into application](#)
-

More information

For more information about the topics covered in this lesson, visit:

- [Routing in Angular Overview](#)
- [Common Routing Tasks](#)

Last reviewed on Tue Jul 11 2023



Lesson 11 - Integrate details page into application



This tutorial lesson demonstrates how to connect the details page to your app.

Estimated time: ~15 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

At the end of this lesson your application will have support for routing to the details page.

Conceptual preview of routing with route parameters

Each housing location has specific details that should be displayed when a user navigates to the details page for that item. To accomplish this goal, you will need to use route parameters.

Route parameters enable you to include dynamic information as a part of your route URL. To identify which housing location a user has clicked on you will use the id property of the `HousingLocation` type.

Step 1 - Create a new service for your app

In lesson 10, you added a second route to `src/app/routes.ts` which includes a special segment that identifies the route parameter, `id`:

```
'details/:id'
```

In this case, `:id` is dynamic and will change based on how the route is requested by the code.

1. In `src/app/housing-location/housing-location.component.ts`, add an anchor tag to the `section` element and include the `routerLink` directive:

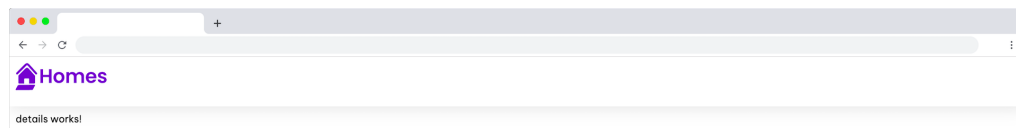
Add anchor with a routerLink directive to `housing-location.component.ts`

```
template: `
  <section class="listing">
    <img class="listing-photo" [src]="housingLocation.photo" alt="Exterior photo
of {{housingLocation.name}}">
    <h2 class="listing-heading">{{ housingLocation.name }}</h2>
    <p class="listing-location">{{ housingLocation.city}}, {{housingLocation.state
}}</p>
    <a [routerLink]="['/details', housingLocation.id]">Learn More</a>
  </section>
`
```

The `routerLink` directive enables Angular's router to create dynamic links in the application. The value assigned to the `routerLink` is an array with two entries: the static portion of the path and the dynamic data.

For the `routerLink` to work in the template, add a file level import of `RouterLink` and `RouterOutlet` from '@angular/router', then update the component `imports` array to include both `RouterLink` and `RouterOutlet`.

2. At this point you can confirm that the routing is working in your app. In the browser, refresh the home page and click the "Learn More" button for a housing location.



Step 2 - Get route parameters

In this step, you will get the route parameter in the `DetailsComponent`. Currently, the app displays `details works!`. Next you'll update the code to display the `id` value passed using the route parameters.

1. In `src/app/details/details.component.ts` update the template to import the functions, classes and services that you'll need to use in the `DetailsComponent`:

Update file level imports

```
import { Component, inject } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ActivatedRoute } from '@angular/router';
import { HousingService } from '../housing.service';
import { HousingLocation } from '../housinglocation';
```

2. Update the `template` property of the `@Component` decorator to display the value `housingLocationId`:

```
template: `<p>details works! {{ housingLocationId }}</p>`,
```

3. Update the body of the `DetailsComponent` with the following code:

```
export class DetailsComponent {
  route: ActivatedRoute = inject(ActivatedRoute);
  housingLocationId = -1;
  constructor() {
    this.housingLocationId = Number(this.route.snapshot.params['id']);
  }
}
```

This code gives the `DetailsComponent` access to the `ActivatedRoute` router feature that enables you to have access to the data about the current route. In the `constructor`, the code converts the `id` parameter acquired from the route from a string to a number.

4. Save all changes.
5. In the browser, click on one of the housing location's "Learn More" links and confirm that the numeric value displayed on the page matches the `id` property for that location in the data.

Step 3 - Customize the DetailComponent

Now that routing is working properly in the application this is a great time to update the template of the `DetailsComponent` to display the specific data represented by the housing location for the route parameter.

To access the data you will add a call to the `HousingService`.

1. Update the template code to match the following code:

Update the DetailsComponent template in src/app/details/details.component.ts

```
template: `
<article>
  <img class="listing-photo" [src]="housingLocation?.photo"
    alt="Exterior photo of {{housingLocation?.name}}"/>
  <section class="listing-description">
    <h2 class="listing-heading">{{housingLocation?.name}}</h2>
    <p class="listing-location">{{housingLocation?.city}},
    {{housingLocation?.state}}</p>
  </section>
  <section class="listing-features">
    <h2 class="section-heading">About this housing location</h2>
    <ul>
      <li>Units available: {{housingLocation?.availableUnits}}</li>
      <li>Does this location have wifi: {{housingLocation?.wifi}}</li>
      <li>Does this location have laundry: {{housingLocation?.laundry}}</li>
    </ul>
  </section>
</article>
`
```

Notice that the `housingLocation` properties are being accessed with the optional chaining operator `?`.

This ensures that if the `housingLocation` value is null or undefined the application doesn't crash.

2. Update the body of the `DetailsComponent` class to match the following code:

Update the DetailsComponent class in src/app/details/details.component.ts

```
export class DetailsComponent {

  route: ActivatedRoute = inject(ActivatedRoute);
  housingService = inject(HousingService);
  housingLocation: HousingLocation | undefined;

  constructor() {
    const housingLocationId = Number(this.route.snapshot.params['id']);
    this.housingLocation =
    this.housingService.getHousingLocationById(housingLocationId);
  }

}
```

Now the component has the code to display the correct information based on the selected housing location. The `constructor` now includes a call to the `HousingService` to pass the route parameter as an argument to the `getHousingLocationById` service function.

3. Copy the following styles into the `src/app/details/details.component.css` file:

Add styles for the DetailsComponent

```
.listing-photo {
  height: 600px;
  width: 50%;
  object-fit: cover;
  border-radius: 30px;
  float: right;
}

.listing-heading {
  font-size: 48pt;
  font-weight: bold;
  margin-bottom: 15px;
}

.listing-location::before {
  content: url('/assets/location-pin.svg') / '';
}

.listing-location {
  font-size: 24pt;
  margin-bottom: 15px;
}

.listing-features > .section-heading {
  color: var(--secondary-color);
  font-size: 24pt;
  margin-bottom: 15px;
}

.listing-features {
  margin-bottom: 20px;
}

.listing-features li {
  font-size: 14pt;
}

li {
  list-style-type: none;
}

.listing-apply .section-heading {
  font-size: 18pt;
  margin-bottom: 15px;
}

label, input {
  display: block;
}

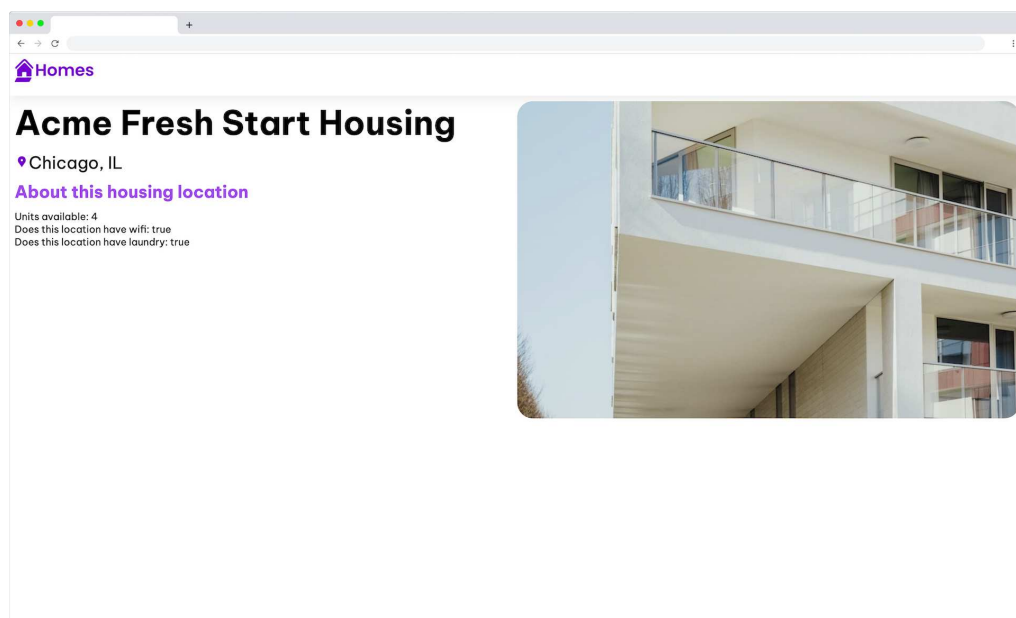
label {
  color: var(--secondary-color);
  font-weight: bold;
  text-transform: uppercase;
  font-size: 12pt;
}

input {
  font-size: 16pt;
  margin-bottom: 15px;
}
```

```
padding: 10px;
width: 400px;
border-top: none;
border-right: none;
border-left: none;
border-bottom: solid .3px;
}
@media (max-width: 1024px) {
  .listing-photo {
    width: 100%;
    height: 400px;
  }
}
```

4. Save your changes.

5. In the browser refresh the page and confirm that when you click on the "Learn More" link for a given housing location the details page displays the correct information based on the data for that selected item.



Step 4 - Add navigation to the HomeComponent

In a previous lesson you updated the `AppComponent` template to include a `routerLink`. Adding that code updated your app to enable navigation back to the `HomeComponent` whenever the logo is clicked.

1. Confirm that your code matches the following:

Add routerLink to AppComponent

```
template: `
  <main>
    <a [routerLink]="['/']">
      <header class="brand-name">
        
      </header>
    </a>
    <section class="content">
      <router-outlet></router-outlet>
    </section>
  </main>
`,
```

Your code may already be up-to-date but confirm to be sure.

Lesson Review

In this lesson you updated your app to:

- use route parameters to pass data to a route
- use the `routerLink` directive to use [dynamic data to create a route](#)
- use route parameter to retrieve data from the `HousingService` to display specific housing location information.

Really great work so far.

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next Steps

- [Lesson 12 - Adding forms to your Angular application](#)

More information

For more information about the topics covered in this lesson, visit:

- [Route Parameters](#)
- [Routing in Angular Overview](#)
- [Common Routing Tasks](#)
- [Optional Chaining Operator](#) [↗](#)



Lesson 12: Adding a form to your Angular app



This tutorial lesson demonstrates how to add a form that collects user data to an Angular app. This lesson starts with a functional Angular app and shows how to add a form to it.

The data that the form collects is sent only to the app's service, which writes it to the browser's console. Using a REST API to send and receive the form's data is not covered in this lesson.

Estimated time: ~15 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

- Your app has a form into which users can enter data that is sent to your app's service.
- The service writes the data from the form to the browser's console log.

Step 1 - Add a method to send form data

This step adds a method to your app's service that receives the form data to send to the data's destination. In this example, the method writes the data from the form to the browser's console log.

In the Edit pane of your IDE:

1. In `src/app/housing.service.ts`, inside the `HousingService` class, paste this method at the bottom of the class definition.

Submit method in src/app/housing.service.ts

```
submitApplication(firstName: string, lastName: string, email: string) {  
  console.log(`Homes application received: firstName: ${firstName}, lastName: ${lastName}, email: ${email}.`);  
}
```

2. Confirm that the app builds without error. Correct any errors before you continue to the next step.

Step 2 - Add the form functions to the details page

This step adds the code to the details page that handles the form's interactions.

In the Edit pane of your IDE, in `src/app/details/details.component.ts`:

1. After the `import` statements at the top of the file, add the following code to import the Angular form classes.

Forms imports in src/app/details/details.component.ts

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

2. In the `DetailsComponent` decorator metadata, update the `imports` property with the following code:

imports directive in src/app/details/details.component.ts

```
imports: [  
  CommonModule,  
  ReactiveFormsModule  
],
```

3. In the `DetailsComponent` class, before the `constructor()` method, add the following code to create the form object.

template directive in src/app/details/details.component.ts

```
applyForm = new FormGroup({  
  firstName: new FormControl(''),  
  lastName: new FormControl(''),  
  email: new FormControl('')  
});
```

In Angular, `FormGroup` and `FormControl` are types that enable you to build forms. The `FormControl` type can provide a default value and shape the form data. In this example `firstName` is a `string` and the default value is empty string.

4. In the `DetailsComponent` class, after the `constructor()` method, add the following code to handle the **Apply now** click.

template directive in src/app/details/details.component.ts

```
submitApplication() {  
  this.housingService.submitApplication(  
    this.applyForm.value.firstName ?? '',  
    this.applyForm.value.lastName ?? '',  
    this.applyForm.value.email ?? ''  
  );  
}
```

This button does not exist yet - you will add it in the next step. In the above code, the `FormControl`s may return `null`. This code uses the nullish coalescing operator to default to empty string if the value is `null`.

5. Confirm that the app builds without error. Correct any errors before you continue to the next step.

Step 3 - Add the form's markup to the details page

This step adds the markup to the details page that displays the form.

In the Edit pane of your IDE, in `src/app/details/details.component.ts`:

1. In the `DetailsComponent` decorator metadata, update the `template` HTML to match the following code to add the form's markup.

template directive in `src/app/details/details.component.ts`

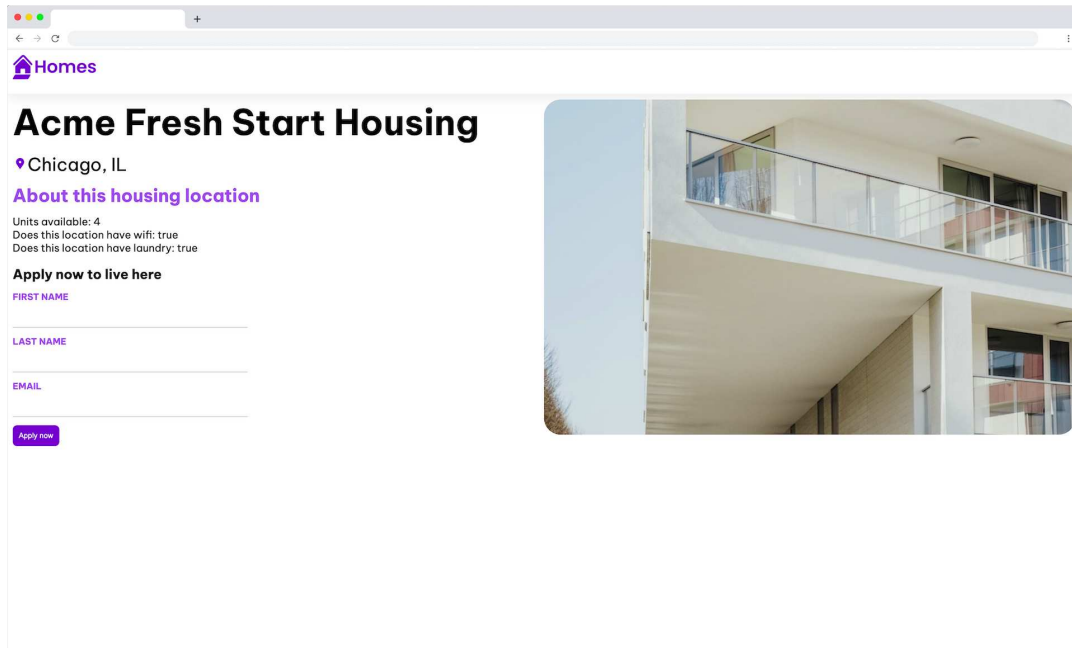
```
template: `
  <article>
    <img class="listing-photo" [src]="housingLocation?.photo"
      alt="Exterior photo of {{housingLocation?.name}}"/>
    <section class="listing-description">
      <h2 class="listing-heading">{{housingLocation?.name}}</h2>
      <p class="listing-location">{{housingLocation?.city}},
{{housingLocation?.state}}</p>
    </section>
    <section class="listing-features">
      <h2 class="section-heading">About this housing location</h2>
      <ul>
        <li>Units available: {{housingLocation?.availableUnits}}</li>
        <li>Does this location have wifi: {{housingLocation?.wifi}}</li>
        <li>Does this location have laundry: {{housingLocation?.laundry}}</li>
      </ul>
    </section>
    <section class="listing-apply">
      <h2 class="section-heading">Apply now to live here</h2>
      <form [formGroup]="applyForm" (submit)="submitApplication()">
        <label for="first-name">First Name</label>
        <input id="first-name" type="text" formControlName="firstName">

        <label for="last-name">Last Name</label>
        <input id="last-name" type="text" formControlName="lastName">

        <label for="email">Email</label>
        <input id="email" type="email" formControlName="email">
        <button type="submit" class="primary">Apply now</button>
      </form>
    </section>
  </article>
`;
```

The template now includes an event handler `(submit)="submitApplication()"`. Angular uses parentheses syntax around the event name to define events in the template code. The code on the right hand side of the equals sign is the code that should be executed when this event is triggered. You can bind to browser events and custom events.

2. Confirm that the app builds without error. Correct any errors before you continue to the next step.



Step 4 - Test your app's new form

This step tests the new form to see that when the form data is submitted to the app, the form data appears in the console log.

1. In the **Terminal** pane of your IDE, run `ng serve`, if it isn't already running.
2. In your browser, open your app at `http://localhost:4200`.
3. Right click on the app in the browser and from the context menu, choose **Inspect**.
4. In the developer tools window, choose the **Console** tab. Make sure that the developer tools window is visible for the next steps
5. In your app:
 - a. Select a housing location and click **Learn more**, to see details about the house.
 - b. In the house's details page, scroll to the bottom to find the new form.
 - c. Enter data into the form's fields - any data is fine.
 - d. Choose **Apply now** to submit the data.
6. In the developer tools window, review the log output to find your form data.

Lesson review

In this lesson, you updated your app to:

- add a form using Angular's forms feature
- connect the data captured in the form to a component using an event handler

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [Lesson 13 - Add the search feature to your application](#)

More information

For more information about the topics covered in this lesson, visit:

- [Angular Forms](#)
- [Event Handling](#)

Last reviewed on Wed Jul 12 2023



Lesson 13: Add the search feature to your app



This tutorial lesson demonstrates how to add a search functionality to your Angular app.

The app will enable users to search through the data provided by your app and display only the results that match the entered term.

Estimated time: ~15 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

- Your app will use data from a form to search for matching housing locations
- Your app will display only the matching housing locations

Step 1 - Update the home component properties

In this step, you'll update the `HomeComponent` class to store data in a new array property that you will use for filtering.

1. In `src/app/home/home.component.ts`, add new property to the class called `filteredLocationList`.

Add the filtered results property

```
filteredLocationList: HousingLocation[] = [];
```

The `filteredLocationList` hold the values that match the search criteria entered by the user.

2. The `filteredLocationList` should contain the total set of housing locations values by default when the page loads. Update the `constructor` for the `HomeComponent` to set the value.

Set the value of filteredLocationList

```
constructor() {  
  this.housingLocationList = this.housingService.getAllHousingLocations();  
  this.filteredLocationList = this.housingLocationList;  
}
```

Step 2 - Update the home component template

The `HomeComponent` already contains an input field that you will use to capture input from the user. That string text will be used to filter the results.

1. Update the `HomeComponent` template to include a template variable in the `input` element called `#filter`.

Add a template variable to HomeComponent's template

```
<input type="text" placeholder="Filter by city" #filter>
```

This example uses a `template reference variable` to get access to the `input` element as its value.

2. Next, update the component template to attach an event handler to the "Search" button.

Bind the click event

```
<button class="primary" type="button"
(click)="filterResults(filter.value)">Search</button>
```

By binding to the `click` event on the `button` element, you are able to call the `filterResults` function.

The argument to the function is the `value` property of the `filter` template variable. Specifically, the `.value` property from the `input` HTML element.

3. The last template update is to the `ngFor` directive. Update the `ngFor` value to iterate over values from the `filteredLocationList` array.

Update the ngFor directive value

```
<app-housing-location *ngFor="let housingLocation of filteredLocationList"
[housingLocation]="housingLocation"></app-housing-location>
```

Step 3 - Implement the event handler function

The template has been updated to bind the `filterResults` function to the `click` event. Next, your task is to implement the `filterResults` function in the `HomeComponent` class.

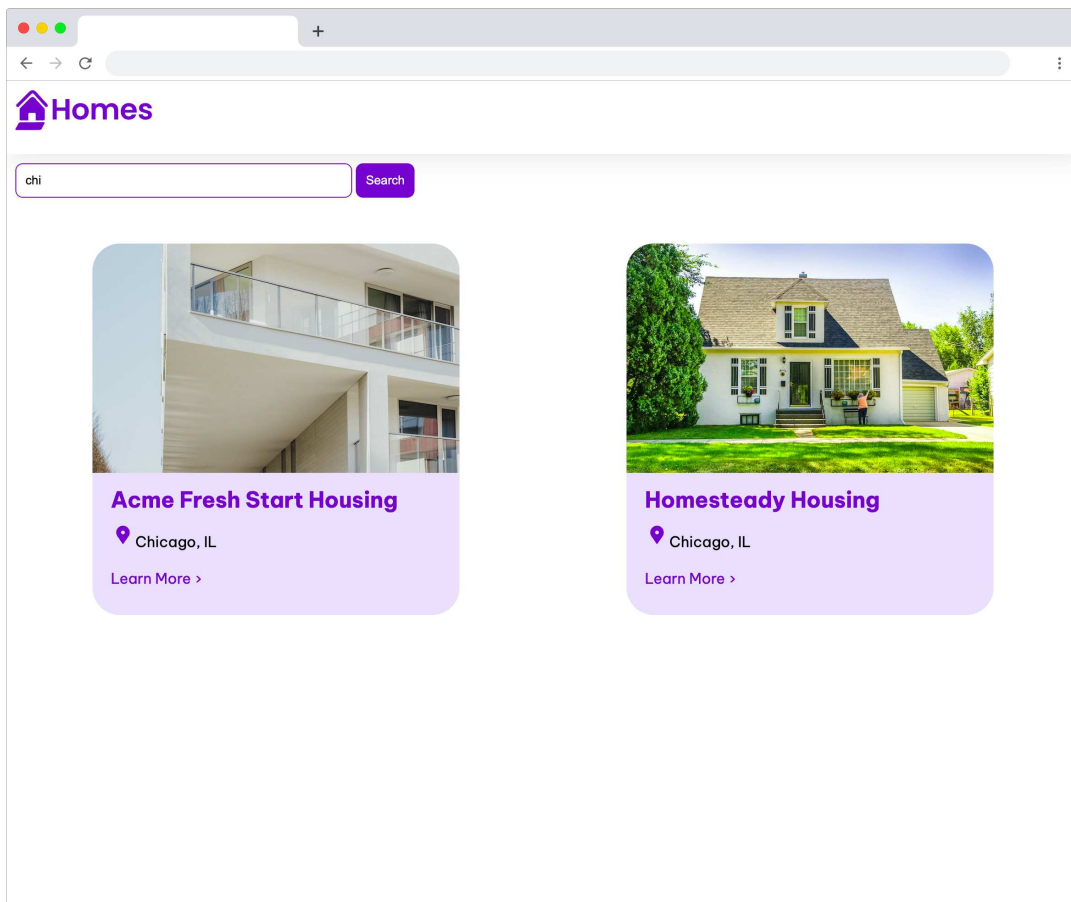
1. Update the `HomeComponent` class to include the implementation of the `filterResults` function.

Add the `filterResults` function implementation


```
filterResults(text: string) {  
  if (!text) {  
    this.filteredLocationList = this.housingLocationList;  
  }  
  
  this.filteredLocationList = this.housingLocationList.filter(  
    housingLocation =>  
    housingLocation?.city.toLowerCase().includes(text.toLowerCase())  
  );  
}
```

This function uses the `String` `filter` function to compare the value of the `text` parameter against the `housingLocation.city` property. You can update this function to match against any property or multiple properties for a fun exercise.

2. Save your code.
3. Refresh the browser and confirm that you can search the housing location data by city when you click the "Search" button after entering text.



Lesson review



In this lesson, you updated your app to:

- use template variables to interact with template values
- add search functionality using event binding and array functions

If you are having any trouble with this lesson, you can review the completed code for it in the [live example](#) / [download example](#).

Next steps

- [Lesson 14 - Add HTTP communication to your app](#)

More information

For more information about the topics covered in this lesson, visit:

- [Template Variables](#)
- [Event Handling](#)

Last reviewed on Tue Jul 11 2023



Lesson 14: Add HTTP communication to your app



This tutorial demonstrates how to integrate HTTP and an API into your app.

Up until this point your app has read data from a static array in an Angular service. The next step is to use a JSON server that your app will communicate with over HTTP. The HTTP request will simulate the experience of working with data from a server.

Estimated time: ~15 minutes

Starting code: [live example](#) / [download example](#)

Completed code: [live example](#) / [download example](#)

What you'll learn

Your app will use data from a JSON server

Step 1 - Configure the JSON server

JSON Server is an open source tool used to create mock REST APIs. You'll use it to serve the housing location data that is currently stored in the housing service.

1. Install `json-server` from npm by using the following command.

```
npm install -g json-server
```

2. In the root directory of your project, create a file called `db.json`. This is where you will store the data for the `json-server`.

3. Open `db.json` and copy the following code into the file

db.json

```
{
  "locations": [
    {
      "id": 0,
      "name": "Acme Fresh Start Housing",
      "city": "Chicago",
      "state": "IL",
      "photo": "/assets/bernard-hermant-CLKGGwIBTaY-unsplash.jpg",
      "availableUnits": 4,
      "wifi": true,
      "laundry": true
    },
    {
      "id": 1,
      "name": "A113 Transitional Housing",
      "city": "Santa Monica",
      "state": "CA",
      "photo": "/assets/brandon-griggs-wR11KBaB86U-unsplash.jpg",
      "availableUnits": 0,
      "wifi": false,
      "laundry": true
    },
    {
      "id": 2,
      "name": "Warm Beds Housing Support",
      "city": "Juneau",
      "state": "AK",
      "photo": "/assets/i-do-nothing-but-love-lAyXd11-Wmc-unsplash.jpg",
      "availableUnits": 1,
      "wifi": false,
      "laundry": false
    },
    {
      "id": 3,
      "name": "Homesteady Housing",
      "city": "Chicago",
      "state": "IL",
      "photo": "/assets/ian-macdonald-W8z6aiwfi1E-unsplash.jpg",
      "availableUnits": 1,
      "wifi": true,
      "laundry": false
    },
    {
      "id": 4,
      "name": "Happy Homes Group",
```

```
"city": "Gary",
"state": "IN",
"photo": "/assets/krzysztof-hepner-978RAXoXnH4-unsplash.jpg",
"availableUnits": 1,
"wifi": true,
"laundry": false
},
{
  "id": 5,
  "name": "Hopeful Apartment Group",
  "city": "Oakland",
  "state": "CA",
  "photo": "/assets/r-architecture-JvQ0Q5IkeMM-unsplash.jpg",
  "availableUnits": 2,
  "wifi": true,
  "laundry": true
},
{
  "id": 6,
  "name": "Seriously Safe Towns",
  "city": "Oakland",
  "state": "CA",
  "photo": "/assets/phil-hearing-IYfp2Ixe9nM-unsplash.jpg",
  "availableUnits": 5,
  "wifi": true,
  "laundry": true
},
{
  "id": 7,
  "name": "Hopeful Housing Solutions",
  "city": "Oakland",
  "state": "CA",
  "photo": "/assets/r-architecture-GGupkreKwxA-unsplash.jpg",
  "availableUnits": 2,
  "wifi": true,
  "laundry": true
},
{
  "id": 8,
  "name": "Seriously Safe Towns",
  "city": "Oakland",
  "state": "CA",
  "photo": "/assets/saru-robert-9rP3mxf8qWI-unsplash.jpg",
  "availableUnits": 10,
  "wifi": false,
```

```
"laundry": false
},
{
  "id": 9,
  "name": "Capital Safe Towns",
  "city": "Portland",
  "state": "OR",
  "photo": "/assets/webaliser-_TPTXZd9m0o-unsplash.jpg",
  "availableUnits": 6,
  "wifi": true,
  "laundry": true
}
]
```

4. Save this file.

5. Time to test your configuration. From the command line, at the root of your project run the following commands.

```
json-server --watch db.json
```

6. In your web browser, navigate to the <http://localhost:3000/locations> and confirm that the response includes the data stored in [db.json](#).

If you have any trouble with your configuration, you can find more details in the [official documentation](#).

Step 2 - Update service to use web server instead of local array

The data source has been configured, the next step is to update your web app to connect to it use the data.

1. In `src/app/housing.service.ts`, make the following changes:

- a. Update the code to remove `housingLocationList` property and the array containing the data.
- b. Add a string property called `url` and set its value to `'http://localhost:3000/locations'`

```
url = 'http://localhost:3000/locations';
```

This code will result in errors in the rest of the file because it depends on the `housingLocationList` property. We're going to update the service methods next.

- c. Update the `getAllHousingLocations` function to make a call to the web server you configured.

```
async getAllHousingLocations(): Promise<HousingLocation[]> {  
  const data = await fetch(this.url);  
  return await data.json() ?? [];  
}
```

The code now uses asynchronous code to make a GET request over HTTP.

For this example, the code uses `fetch`. For more advanced use cases consider using `HttpClient` provided by Angular.

- d. Update the `getHousingLocationsById` function to make a call to the web server you configured.

```
async getHousingLocationById(id: number): Promise<HousingLocation |  
undefined> {  
  const data = await fetch(`${this.url}/${id}`);  
  return await data.json() ?? {};  
}
```

- e. Once all the updates are complete, your updated service should match the following code.

Final version of housing.service.ts

```
import { Injectable } from '@angular/core';
import { HousingLocation } from './housinglocation';

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

  url = 'http://localhost:3000/locations';

  async getAllHousingLocations(): Promise<HousingLocation[]> {
    const data = await fetch(this.url);
    return await data.json() ?? [];
  }

  async getHousingLocationById(id: number): Promise<HousingLocation |
  undefined> {
    const data = await fetch(`${this.url}/${id}`);
    return await data.json() ?? {};
  }

  submitApplication(firstName: string, lastName: string, email: string) {
    console.log(firstName, lastName, email);
  }
}
```

Step 3 - Update the components to use asynchronous calls to the housing service

The server is now reading data from the HTTP request but the components that rely on the service now have errors because they were programmed to use the synchronous version of the service.

1. In `src/app/home/home.component.ts`, update the `constructor` to use the new asynchronous version of the `getAllHousingLocations` method.

```
constructor() {  
  this.housingService.getAllHousingLocations().then((housingLocationList:  
HousingLocation[]) => {  
    this.housingLocationList = housingLocationList;  
    this.filteredLocationList = housingLocationList;  
  });  
}
```

2. In `src/app/details/details.component.ts`, update the `constructor` to use the new asynchronous version of the `getHousingLocationById` method.

```
constructor() {  
  const housingLocationId = parseInt(this.route.snapshot.params['id'], 10);  
  
  this.housingService.getHousingLocationById(housingLocationId).then(housingLocation  
=> {  
    this.housingLocation = housingLocation;  
  });  
}
```

3. Save your code.
4. Open the application in the browser and confirm that it runs without any errors.

Lesson review

In this lesson, you updated your app to:

- use a local web server (`json-server`)
- use asynchronous service methods to retrieve data.

Congratulations! You've successfully completed this tutorial and are ready to continue your journey with building even more complex Angular Apps. If you would like to learn more, please consider completing some of Angular's other developer [tutorials](#) and [guides](#).

Last reviewed on Wed Jul 12 2023