**Creating an App**

You'll learn how to create an Angular app with a few commands, run the server, and turn it off.

**We'll cover the following**

* [Scaffolding a new project locally](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5950260637073408#Scaffolding-a-new-project-locally)
* [Starting/stopping the server](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5950260637073408#Startingstopping-the-server)
* [Educative example](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5950260637073408#Educative-example)

We’re going to start things off by scaffolding a new project. You’ll learn how to start a project both locally and on Educative.

**Scaffolding a new project locally**

We installed the CLI in the previous lesson. We’re going to use it to create a new project. In the terminal, we’ll need to run the ng new command.

ng new reverse-phrase

After inputting the command, we’ll need to provide a name for the project. The first project we’ll create will reverse a string based on an input. Hence, the reason we’re calling it the *reverse phrase*.

The CLI will ask us a couple of questions about the project we’d like to generate. These are just options to customize the project further. Generally, we’ll be asked if we’d like to turn on routing. Routing is a topic we’ll cover in the future, but we don’t need it right now. Select *No*.

Then, it will ask us to select a CSS tool. Plain CSS will work.

Just hit *enter* for everything. We’ll be using the default settings. We won’t be doing anything super complex, so the default settings will work just fine.

After running the command, the CLI will start downloading the starter files and install its dependencies.

**Starting/stopping the server**

Angular will provide us with a development server to locally preview the app in the browser. It will even watch the files for changes and refresh the application for us. The best part is that we don’t need to configure the server. It works out of the box. This lets us focus on developing the application.

There are two commands to start the server: npm start and ng serve. You will find that most tutorials will use either one. However, they’re 100% identical. There aren’t any differences between the two. Behind the scenes, the npm start command will run the ng serve command. Feel free to use either one. In the terminal, run the following command:

ng serve

Angular will begin to compile your project. It will then watch your files for changes. Anytime you make a change, Angular will recompile the project. You should be provided with a link to view your project in the browser.

By default, you can view the project over at [http://localhost:4200](http://localhost:4200/) or [http://127.0.0.1:4200](http://127.0.0.1:4200/)

You may see something different depending on when you’re taking this course. The default start page usually changes in future versions. Regardless, you should be able to view the application in the browser if everything worked.

The server can be stopped by pressing CTRL + C on your keyboard.

## Educative example

Throughout this course, you’ll come across Educative’s widget for running code. You’ll be able to modify the files in the widget freely. In fact, I encourage you to do so to help you better understand Angular.

You can run the example by pressing the Run button. This will generate a preview of the output and logs in the terminal. There’s even a link if you’d like to see the fully functional app in a separate tab. Keep in mind that the link will not work unless you press the Run button first.

The following example is scaffolded when you run the ng new command.

**Reviewing the Files**

Let’s review some of the files to feel more comfortable with the project.

**We'll cover the following**

* [Root Files](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#Root-Files)
  + [.browserslistrc](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#browserslistrc)
  + [.editorconfig](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#editorconfig)
  + [.gitignore](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#gitignore)
  + [angular.json](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#angularjson)
  + [karma.config.json](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#karmaconfigjson)
  + [package.json](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#packagejson)
  + [tsconfig.json](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#tsconfigjson)
* [Directories](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#Directories)
  + [e2e](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#e2e)
  + [src](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#src)
* [Tunnel visioning](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#Tunnel-visioning)
  + [Convention over configuration](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#Convention-over-configuration)
  + [Focusing on two files](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5744686926397440#Focusing-on-two-files)

The starter project by Angular can be overwhelming at first. The CLI downloads a bunch of files that it can be confusing as to where to start. We’ll review some of the files to become more comfortable with the project. This review won’t be super comprehensive. We’ll be briefly looking at things.

## Root Files

We’ll start with the files in the root directory of the project.

### .browserslistrc

Angular is built with various tools, such as Webpack. Most tools can be configured to output different results for specific browsers. It can be tedious to configure each tool for different browsers. This is where the .browserslistrc file comes into play.

Instead of configuring each tool, you can outsource the list of browsers to support in one file. The current configuration will support the latest browser but not IE. You can click on the link at the top of the file for various queries that can be added for more browser support. You can also run the following command to get the list of browsers the configuration will support:

npx browserslist

### .editorconfig

The first file is the .editorconfig file. If you’re working on a team, chances are everyone will be using a different editor. Their editors may be configured differently than yours, which can cause issues when the code is committed.

The .editorconfig file can be detected by most code editors. If present, the settings in the file will override the current settings of the editor. This saves you from having to manually make the changes. It’s a great way to keep formatting consistent across multiple editors.

At the top of the file, you’ll find a link to the documentation for various configuration options. We won’t be using this file for the course. You can safely ignore or delete it.

### .gitignore

The .gitignore file will tell Git what files to ignore when committing files. Angular has adjusted this file to include files and folders that don’t need to be committed. We won’t be modifying this file.

### angular.json

The angular.json file is how we can configure the workspace and project. The CLI will use it to determine how it should build the project for development and production. We’ll be revisiting this file throughout the course.

### karma.config.json

The karma.config.json file is the configuration for the Karma test runner tool. Karma allows us to run tests in an environment of our choosing. You can safely ignore this file for the time being. It won’t impact how we write code.

### package.json

The package.json file will contain a list of dependencies for our project. It will also have commands for interacting with the project using the Angular CLI.

### tsconfig.json

There are three configuration files for TypeScript: tsconfig.app.json, tsconfig.json, and tsconfig.spec.json. The tsconfig.app.json file is used when the application is compiled. The tsconfig.spec.json file is used when the testing code is compiled. Both files extend the tsconfig.json file.

We aren’t going to be modifying the TypeScript configuration. It’s been optimized to best suit Angular. However, if you do need to change things, then you are more than welcome to do so.

## Directories

Let’s explore the directories created in the project.

### e2e

The e2e directory is where we can define tests for end-to-end testing. There’s not much else to it.

### src

The src directory is where we’ll be working a majority of the time. It’s where the application code will mainly sit. The CLI will compile the code inside this directory, bundle it, and serve the build.

## Tunnel visioning

One of the biggest challenges of learning Angular is trying to understand everything it creates all at once. If you’ve ever used React or Vue, then you know it’s possible to get a project going by loading one or two files. This makes it easier to get started with either library.

However, Angular is different. The files required for a basic project are comprehensive. This is because Angular is meant for large applications that can scale. Unlike other frameworks, it’s not meant for building small widgets on a page. It’s a convention over configuration framework.

### Convention over configuration

There are two types of frameworks: configuration over convention and convention over configuration.

A configuration over convention framework is when a framework lets you decide how to structure your application. It provides you with the functions necessary to build an app but doesn’t dictate how you name your files, functions, or variables. It’s completely flexible. For this reason, they’re prevalent because of their beginner-friendly approach. React and Vue falls into this category.

A convention over configuration framework is when a framework is much more opinionated. It requires an application to be structured in a certain way and has rules for how you should define classes, functions, and variables. These frameworks have a higher learning curve but are much more suitable for large applications. It’s easier for teams to use because everyone has to adhere to standards set by the framework. Angular falls into this category.

### Focusing on two files

In the next lesson, we’re going to focus on two files: src/app/app.component.html and src/app/app.component.ts. We’ll be ignoring everything else.

**Why?**

That’s a fair question. The approach we’ll be taking to learning Angular is to focus on a small set of files and expand as we go along. We won’t worry about why the files are named or structured in a specific way. We’ll learn about the other files as the course progresses. For now, I want you to pretend as if the app.component.html and app.component.ts files are the only files in the src directory.

# Event Binding

We'll become familiar with the template syntax features available in Angular. The first template syntax feature we'll learn is event binding.

**We'll cover the following**

* [Listening for events](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5584934107873280#Listening-for-events)
* [Component classes](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5584934107873280#Component-classes)

In this lesson, we’re going to learn about **template syntax**. Templates in Angular are the HTML in your application, i.e., the presentation part of the application. We’re not limited to tags and attributes in a template. There’s additional syntax available to make the template more dynamic.

The template for the application can be found in the src/app directory under the file name app.component.html. It’ll contain the following:

Below the comments are hundreds of lines of HTML. As the comment suggests, we’re more than free to replace the template completely. Let’s replace it with a simple heading.

If we press the *Run* button, then we’ll see that the page has been completely replaced with the heading in the app.component.html file. This file is responsible for outputting the template of the application. Angular will refer to the template as the view. In this course, I’ll be using the words *template* and *view* interchangeably.

**Listening for events**

The application we’ll build will output a string in reverse. The first thing we’ll need to do is listen for the event when a new string is submitted. Only then will we be able to reverse the string. In the app.component.html file, we’ll add the following:

1

2

3

4

5

6

7

8

<div>

  <label>Text</label>

  <input />

</div>

<div>

  <button (click)="onClickReverse()">Submit</button>

</div>

There are two sets of <div> tags. The first is wrapped around a <label> and <input> element. The second is wrapped around a <button> element.

The button is listening for an event called click. In Angular, we can listen for events by adding the name of the event as an attribute to the element wrapped in parentheses. In this case, we’re listening to the click event.

The value for the event will be code we’d like to run when the event is emitted. In this example, we’re running a function called onClickReversed().

This is considered invalid syntax in HTML5. However, thanks to Angular, it can help the browser understand the strange syntax we’ve added to our <button> element.

So, that leads us to the question: Where do we define the function?

**Component classes**

Every template will be connected to a component class. A component class is an object of methods and properties that are exposed to the template. It can be used to control and interact with the template. We can find the component class in the app.component.ts file.

Here’s what it looks like at the moment:

1

2

3

4

5

6

7

8

9

10

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

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'reverse-phrase';

}

We’re going to ignore everything about this file except for the class that’s being exported. This is because I want to focus solely on how to create templates. We’ll dive deeper into what the rest of the syntax is later in the course. The only thing you need to know about the @Component decorator is that it helps Angular identify the class as a component class.

The purpose of the class being exported is to expose methods and properties to the template. By default, it will have a property called title. The CLI scaffolded this, so we can safely remove it.

Let’s update it to include the onClickReverse() method.

**Note:** The terminal in the widget will output the logs of the terminal on the server. If you’d like to see the logs from the client, you’ll need to click the link to open the app on a separate page and open the developer tools.

If we click the Run button, we’ll be able to click the button. The button is listening for the click event. If triggered, the event will run the onClickReverse() method, which will log a message in the console.

Great! We’ve learned how to listen for events in Angular. We’re using what’s called **event binding syntax**.

One thing you’ll notice is that we’re not using the this keyword or referring to the class name, AppComponent, in the template. Angular is capable of directly referencing the methods and properties in a component class. This allows us to call the function by its name onClickReverse(). We don’t have to use:

<button (click)="this.onClickReverse()">Submit</button>

or

<button (click)="AppComponent.onClickReverse()">Submit</button>

**What are components?** I mentioned the word component a lot in this lesson. A component is a custom HTML element that the browser can understand. We’ll dive into components in-depth in the next section. For now, don’t worry so much about what components are. The main focus of this section is to learn about the special syntax we can use in our templates.

# Property Binding

Learn how to bind properties to an attribute.

**We'll cover the following**

* [Storing the input](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4534795947737088#Storing-the-input)
* [Disabling the button](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4534795947737088#Disabling-the-button)
  + [Alternate solution](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/4534795947737088#Alternate-solution)

We have a function that will run whenever a button is clicked. All it does is log a message. We’ll want to make it more useful by reversing some input text. First, we’ll need to store the value from the input.

Here’s how we’ll approach things:

1. We’ll create a property to store the input’s value.
2. We’ll listen to the input event on the <input> element.
3. If the input event is triggered, we’ll update the property.
4. The button will need to be disabled if the input is empty.

The last step will require us to do some property binding.

## Storing the input

In the app.component.ts component class file, we’ll define a property called text. It will be set to an empty string.

export class AppComponent {  
  text = '';  
}

The goal is to update the text property whenever the user starts to type into the input. In the template, we can listen to the input event on the <input> element, which is triggered every time the input changes. We’ll apply this change in the app.component.html file.

<div>  
  <label>Text</label>  
  <input (input)="onInputText($event.target.value)" />  
</div>

We’re using the same syntax we used before: a pair of parentheses with the name of the event inside. Angular supports all browser events on any element in the DOM.

In this example, we’re running a function, called onInputText(), when the input event is emitted. It’s being passed in the $event.target.value property.

Angular will define $event object for you. It’s accessible in the template. This object represents the JavaScript event object. We have access to the same properties and methods that we would find in the vanilla event object. The target property contains information about the element on which the event was triggered. If the target element is an <input> element, we can access the value through the value property.

We’ll update the component class to include the method.

export class AppComponent {  
  text = '';  
  
  onInputText (value: string) {  
    this.text = value;  
  }  
}

The onInputText() function has one parameter: value. We’re assigning a type to the parameter. We want to make sure we’re receiving a string.

**Note:** JavaScript will always treat the input’s value as a string.

Inside the function, we’re updating the text property with the value parameter.

## Disabling the button

The final step is to disable the button. Buttons can be disabled using the disabled attribute. However, we’ll want to toggle this attribute dynamically. We can do so by binding it to a property. In the app.component.html file, we’ll update the button to the following:

<div>  
  <button (click)="onClickReverse()" [disabled]="!text.length">Submit</button>  
</div>

What we’re performing is called **property binding syntax**. We can bind an attribute by wrapping the attribute with square brackets. This will tell Angular to interpret the value of the attribute as an expression.

**Note**: Expressions are lines of code that get evaluated into a single value.

In this example, we’re checking if the text property in the component class has a length. If it doesn’t, this will evaluate to true. This will cause Angular to add the disabled attribute to the element.

Once again, just like methods, we don’t have to access properties via the this keyword or by the class name AppComponent. The template will have access to the properties and methods in the component class.

### Alternate solution

Instead of binding an attribute to a property, we can bind it to a function that will return a value. For example, the disabled attribute can be updated to the following:

<div>  
  <button (click)="onClickReverse()" [disabled]="disableButton()">Submit</button>  
</div>

Then, in the component class, we can add the disableButton() function.

export class AppComponent {  
  text = '';  
    
  disableButton () {  
    return !text.length;  
  }  
  
  onInputText (value: string) {  
    this.text = value;  
  }  
}

This is one approach. For our purposes, we’ll be going with the first approach because it’s simpler.

After running the code, the button is disabled by default because the text property is set to an empty string. It will have a length of 0, which evaluates to false.

If we begin to type into the input, the button is enabled. This happens automatically. Angular will keep track of the properties in the component. If a property changes, any bindings that use the property in the template are re-evaluated. In this case, the disabled attribute was removed because the text property had its length property changed.

This is the benefit of using Angular. The component class and template are tied together, which means any changes to the properties are automatically reflected in the template.

# Interpolation

Learn how to output a property in plain text.

**We'll cover the following**

* [Terminology](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6600833560477696#Terminology)
  + [Expressions](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6600833560477696#Expressions)
  + [Interpolation](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6600833560477696#Interpolation)
* [Reversing the text](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6600833560477696#Reversing-the-text)
  + [Interpolation in the template](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/6600833560477696#Interpolation-in-the-template)

It’s time to reverse the string and output the results. This will involve a process called interpolation.

## Terminology

Angular loves its terminology. Some words, such as interpolation and expressions, come up frequently. Let’s explore what these are.

### Expressions

In JavaScript, expressions are single lines of code that evaluate to a single value. The value can be a number, string, or logical value. There are all kinds of expressions. Arithmetic operations, basic string manipulation, and logical comparisons are all considered expressions. Here are some examples of expressions:

*// Number Expression*

2 + 2; *// Evaluates to 4*

*// String Expression*

"hello".toUpperCase(); *// Evaluates to "HELLO"*

*// Logical Expression*

100 === 100 *// Evalutes to true*

To add a bit more clarity, here are some examples that aren’t considered expressions: variable declarations, function declarations, and conditionals statements.

*// Variable Declarations*

const a;

*// Function Declarations*

function greet (message) {

    console.log(message);

}

*// Conditional Statements*

if (a === b) {

*// Do something*

}

### Interpolation

Angular will search through your template for expressions. It will interpolation on the result of an expression, which is the process of replacing placeholders with string values. Let’s look at how interpolation works and get a better understanding of it.

## Reversing the text

There are three things we’ll need to do to reverse the text.

1. Store the reversed text.
2. Update the onClickReverse() function to reverse the text property when the button is clicked. We’ll store the result.
3. Output the reversed text.

We can handle the first two steps in the component class. In the app.component.ts file, update the class to the following:

export class AppComponent {

  text = '';

  reversedText = '';

  onClickReverse () {

    this.reversedText = this.text.split('').reverse().join('');

  }

  onInputText (value: string) {

    this.text = value;

  }

}

In the example above, we’re creating a property called reversedText. Its initial value will be an empty string.

Next, we’re updating the onClickReverse() function to update the reversedText property. We’re chaining a couple of functions onto the text property to get the desired results. Here’s a rundown of what these functions do:

1. The split() method splits a string into an array of strings by separating the string into substrings.
2. The reverse() method reverses an array. The first array element becomes the last, and the last array element becomes the first. This is applied to the result of the split() method.
3. The join() method joins all elements of an array into a string.

The final result of all this is the text in reverse.

### Interpolation in the template

The final step is to output the reversedText property in the template. We’ll update the app.component.html file to include the following at the very bottom:

<div>

  Reversed Text: {{ reversedText }}

</div>

We can output properties using double curly braces ({{ }}). The code inside the curly braces **must** be an expression. The result of the expression will be interpolated, and Angular will replace the curly braces with the evaluated value. This is interpolation.

Press the Run button. If we begin to type in the input, the button will be enabled. If we press it, the reversedText property will be updated. Angular will update the text in the <div> tag with the newly updated value.

The curly brackets syntax is a way to output the value of an expression. We’ve successfully reversed the text. Congrats! In the next couple of lessons, we’ll polish things up.

# Structural Directives

In this lesson, we'll learn about how to use directives to conditionally render elements.

**We'll cover the following**

* [Directives](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5918992369188864#Directives)
  + [Structural directives](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5918992369188864#Structural-directives)
  + [Attribute directives](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5918992369188864#Attribute-directives)
* [ngIf](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5918992369188864#ngIf)

There’s one final touch we’ll add to the application. Below the button, the text Reversed Text appears regardless if there’s text to reverse or not. It would be ideal if we could hide the element until there’s something to reverse. We can do just that using directives.

## Directives

A directive is a special attribute used to change the appearance or behavior of an element. There are two types of directives you’ll commonly use in your template.

### Structural directives

The first type is structural directives. These directives have the power to change the layout of the DOM by adding/removing elements. They can be identified by the \* character before the name of the attribute.

### Attribute directives

The second type is attribute directives. They can change the appearance or behavior of a DOM element.

## ngIf

Let’s use a directive in action. We can hide an element with the ngIf directive, which is a structural directive. We’ll apply it to the <div> element wrapped around the reversedText expression in the app.component.html file.

<div \*ngIf="reversedText.length">  
  Reversed Text: {{ reversedText }}  
</div>

The format for using structural directives is an asterisk (\*), followed by the name of the directive. The value for the directive will be an expression that can evaluate to either true or false. The ngIf directive will add an element to the document if the expression evaluates to true. Otherwise, it will hide the element. It’s similar to how conditional statements work in JavaScript. The main difference is that we can apply this logic directly on an element.

In this example, we’re checking the length property on the reversedText property. The length property of strings will return 0 if there are no characters in the string. This is the same as false. A a result, the element is removed from the document.

Directives are a great way to make a document interactive. Let’s see them in action. In the widget below, the <div> element is hidden. We won’t be able to see it until we type in the input and click the button.

# Deploying the App

We'll learn how to deploy the application to the web.

**We'll cover the following**

* [Installing Vercel](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5385056664682496#Installing-Vercel)
* [Deploying with Vercel](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5385056664682496#Deploying-with-Vercel)
* [We’re finished!](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5385056664682496#Were-finished)
* [The build command](https://www.educative.io/module/page/El5jyzfkAngPpgpAB/10370001/5017901150502912/5385056664682496#The-build-command)

At the moment, we’re the only ones that can preview the application. We can’t share it with anyone. That’s a shame. We’ve worked very hard on this app. I think it would be a good idea to share it with the world.

The first thing we need to do is pick a hosting service. There are dozens of services available. Every single one of them has its pros and cons. You will need to do some research to find out which service best suits you. Some of the most popular services are Heroku, Digital Ocean, AWS, and Netlify. I consider them all to be great choices.

For this application, we’re going to deploy to a service called Vercel. Vercel is a service for deploying/hosting your applications. It supports dozens of frameworks and languages out of the box, including Angular. It’s my personal favorite. I use it for clients and personal projects. It’s quick and painless. Hopefully, by the end of this lesson, you’ll love it too.

You can sign up here: <https://vercel.com/>.

Signing up is free and easy. You don’t need a credit card for an account. They allow you to upload hobby projects for free. You don’t have to pay anything for them to host your project.

## Installing Vercel

Once you’ve got an account, let’s begin deploying the project.

**Note:** If you’ve been using the Educative widgets to run the examples, you can download a copy of the files [here](https://github.com/jaskokoyn/reverse-phrase).

Vercel comes with a tool for deploying projects with a single command. We’ll need to install the tool if we want to deploy the project.

In your terminal, run the following command.

npm i -g vercel

It’s essential that we install this as a global command. This way, we can use it with any project.

Vercel is a command tool for interacting with Vercel. It helps you deploy a project to Vercel. The next thing we need to do is connect our account. The Vercel tool won’t know where to upload the files if it doesn’t know which account to use. We can connect an account by running the following command:

vercel login

After running the command, you’ll be prompted to provide your email. A confirmation email will be sent. Verify your email before continuing.

## Deploying with Vercel

The application can be deployed with the following command:

vercel

1. It will ask us a few questions about the project. Let’s run through them together.
2. The first question it will ask you is if you’re sure about deploying your project. Select Yes.
3. The next question will ask you which account you’d like to upload to. Your name should appear in the list. Select it by pressing enter.
4. Then, it will ask you if you would like to link this project with an existing project in Vercel. This is a brand new project, so we’ll select No.
5. Afterward, it will ask you for the name of the project. By default, it will use the name of the directory the command is running in. We’ll use the default.
6. Up next, it will ask for the directory in which the project is created. Vercel is compatible with dozens of frameworks. It can even build projects for you. You don’t have to build it yourself. It will detect what framework you’re using and run the appropriate commands for building the project. If you would like to let it build the project for you, you can leave this at the default value, which is the root directory of the project. We’ll let Vercel build the project.
7. It will detect that we’re using Angular as our framework and will determine which commands to run. It should be able to identify the build command as: ng build. If it is, proceed onto the next step.

It will begin to deploy the project to your account. This will take a few moments.

## We’re finished!

This project has been successfully deployed. You can view the project by clicking on the link in the log. Vercel will provide you with a custom URL for viewing your project.

Here’s the URL to my deployment of the project: <https://reverse-phrase.now.sh/>.

Hopefully, I’ve convinced you that Vercel is a service you should consider for deploying your projects. It can offer so much more. We’ve only scratched the surface.

## The build command

So far, we’ve been using the ng serve command to build our project. The ng serve command will create a development server and watch our files for any changes. If any changes are made, it will compile the project with the changes. After the code’s been compiled, the page on the browser reloads. This is great for development, but it’s not what we’ll want to use for production.

The ng serve command doesn’t fully optimize the build for production. It’s only meant for development. If we want to build the project for production, we’ll need to use the ng build command. Unlike the ng serve command, this command will compile and minify the project. It will remove development dependencies, optimize the build, and deliver your files in a single directory. This is the command that Vercel used to build the project before deploying it.

The compiled project can be found in a directory called dist. The files you’ll find in this directory are the files you’ll want to deploy to the host. You don’t need to deploy anything outside of this.