NUXT.JS SUCCINCTLY

BY **ED FREITAS**

SUCCINCTLY EBOOK SERIES

Syncfusion®

Nuxt.js Succinctly

By

Ed Freitas

Foreword by Daniel Jebaraj

ISBN: 978-1-64200-225-6

Technical Reviewer: James McCaffrey
Copy Editor: Courtney Wright
Acquisitions Coordinator: Tres Watkins, VP of content, Syncfusion, Inc.
Proofreader: Jacqueline Bieringer, content producer, Syncfusion, Inc.

Table of Contents

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, CEO
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn’t everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors’ tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click,” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!

About the Author

Ed Freitas is a consultant on business process automation and a software developer focused on customer success.

He likes technology and enjoys learning, playing soccer, running, traveling, and being around his family.

Ed is available at https://edfreitas.me.

Acknowledgments

A huge thank you to the fantastic Syncfusion team that helped this book become a reality—especially Jacqueline Bieringer, Tres Watkins, and Graham High.

The manuscript manager and technical editor thoroughly reviewed the book's organization, code quality, and overall accuracy—Jacqueline Bieringer from Syncfusion and James McCaffrey from Microsoft Research. Thank you both.

I dedicate this book to my dear Uncle Tony, who recently passed away. A veteran, he devoted his life to his loved ones, his beloved country—the United States of America—and his work.

He performed his work with utmost passion, integrity, and with a smile; he never missed a day, and he always gave his best to his customers and anyone who sought his wisdom and advice. Your love, character, and everything you did will always remain with everyone you loved. Rest in peace.

This book is also dedicated to the all the victims of international aggression and violence. I hope that one day (in the not-so-distant future), we will become an advanced species that cares for our pale blue dot, practices love and compassion instead of hatred, and realizes that all we have is each other. To create a better future and world, we must learn to cherish and appreciate the lives of others as much as our own.

Introduction

Let's begin by explaining what Nuxt.js is. To do that, we first need to talk about Vue.js.

Vue.js is a progressive (meaning incrementally adoptable) front-end JavaScript framework that you can use to build highly engaging user interfaces and single-page applications.

Nuxt.js is an open-source JavaScript library based on Vue.js that uses Node.js, Webpack, and Babel.js under the hood. Nuxt.js takes inspiration from Next.js, a framework of similar purpose that's based on React.js.

Nuxt.js takes Vue.js development to the next level and builds upon Vue.js. Think of it as a *framework for a framework*—adding two significant features to Vue.js: server-side rendering; and easy Vue.js application configuration and routing through folders and files.

To make it simple to understand, the goal of Nuxt.js is to make it easier for developers to create and optimize Vue.js applications.

One of the main features of Nuxt.js is its ability to add server-side rendering to a Vue.js application. Server-side rendering essentially means that we can build a Vue.js application to prerender pages on the server before serving them to the user.

Server-side rendering has excellent advantages for search engine optimization (SEO) and can significantly speed up your web app.

Traditional Vue.js applications require that developers create routes; however, with Nuxt.js, explicitly creating routes is no longer necessary, as routes are inferred by the folder and file structure of your Nuxt.js project.

In a nutshell, Nuxt.js simplifies the development of Vue.js applications and makes the experience of writing Vue.js apps even more exciting and fun without adding any overhead to the shipped bundle, and instead, optimizing the app for production.

If you are starting your journey with Vue.js, it's probably better that you get up to speed with Vue.js first, rather than jumping straight into Nuxt.js. For that, the *Succinctly* series has you covered with *Vue.js Succinctly*.

If you are not new to Vue.js, it will be okay to start your journey with Nuxt.js directly, and you'll feel at home right off the bat.

Nuxt.js adds a nice layer of sugar coating around Vue.js, and if you are a Vue.js developer, it is a tool that will boost your productivity.

So, without further ado, let's explore what this technology has to offer.

Chapter 1 Getting Started

Client-side vs. server-side rendering

Before we begin, there's one crucial concept that I would like you to have crystal clear in your mind, and that's server-side rendering.

Nuxt.js should not be confused with a templating engine running on the server—it's not a replacement for templating engines such as EJS or Handlebars.

To understand server-side rendering, we need to take a step back and look at how traditional Vue.js applications (which are single-page applications) work—which is another reason you should first get up-to-speed with Vue.js before reading this book.

If you place UI components into your application pages rendered by some other server-side service or technology, such as PHP or ASP.NET, you don't need Nuxt.js.

On the other hand, Vue.js is a client-side framework for traditional single-page applications. The single-page application mechanism works for a Vue.js app because the user sends a request by entering a URL, and the server sends back an HTML file.

In that case, the HTML file sent by the server contains barely any HTML code, but it does include your single-page application logic—essentially, all the scripts that need to be loaded to start the Vue.js app, which runs entirely in the browser.

The loaded Vue.js app is responsible for rendering the UI. Therefore, the view adds all the HTML elements to the page, and it is also responsible for routing and catching any URLs that you might visit within the app.

With a traditional Vue.js application, you never receive a second HTML page from the server as a response when you visit a new URL route within that application (as long as you stay within the application's routes).

However, the HTML file could be served from the server and rendered there. In the traditional Vue.js application approach, the HTML file received doesn't contain the HTML that the user sees on the screen, as this content is all created on the client side through JavaScript.

The traditional Vue.js approach is not great for search engine optimization, especially if you need to load data asynchronously before rendering something onto the screen. The following figure illustrates this process.

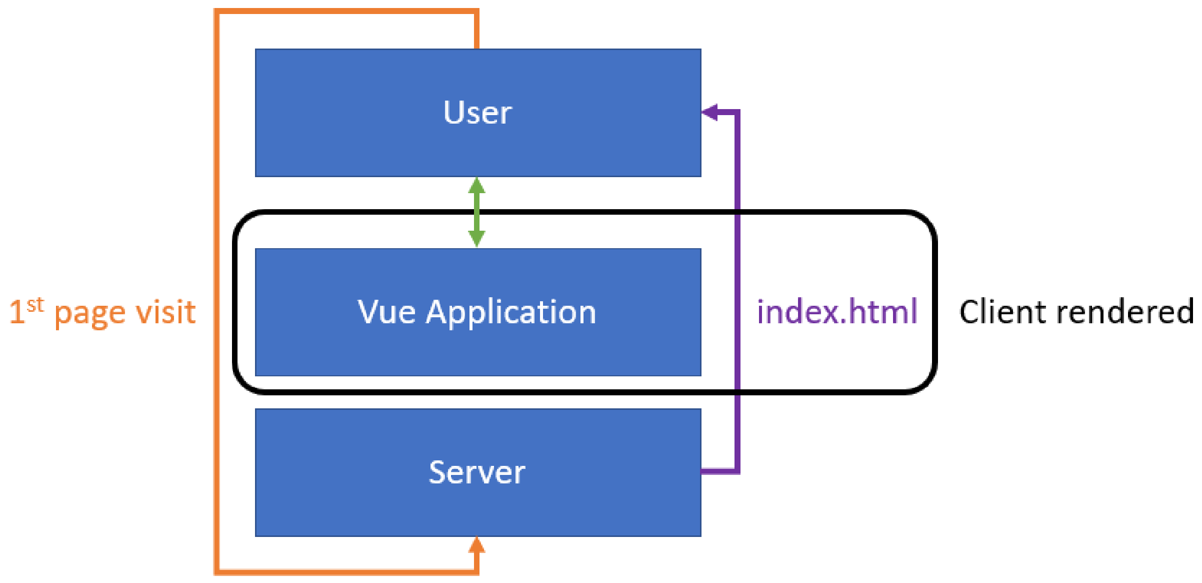


Figure 1-a: Traditional Client-Side Rendering (Single-Page Application)

In Figure 1-a, the index.html file is rendered on the client after the server returns it following a user request.

So, imagine for a second that the Google web crawler is indexing your site or application created with Vue.js. The crawler will not wait for your page content to load; instead, it will see an empty page. On the other hand, this traditional single-page approach is not well suited for sites that require fast loading times, such as e-commerce applications.

What Nuxt.js gives us is the ability to solve this problem by rendering that first page on the server. Therefore, that first page the user visits for any given URL within the application scope (independent of the root URL or another app URL) can be prerendered on the server, on the fly, when the user requests it.

With Nuxt.js, you get a regular Vue.js application, but that app gets prerendered dynamically or even statically on the server. In other words, with Nuxt.js, when a user accesses your page, the page is sent from the server prerendered and loaded. The following figure illustrates this process.

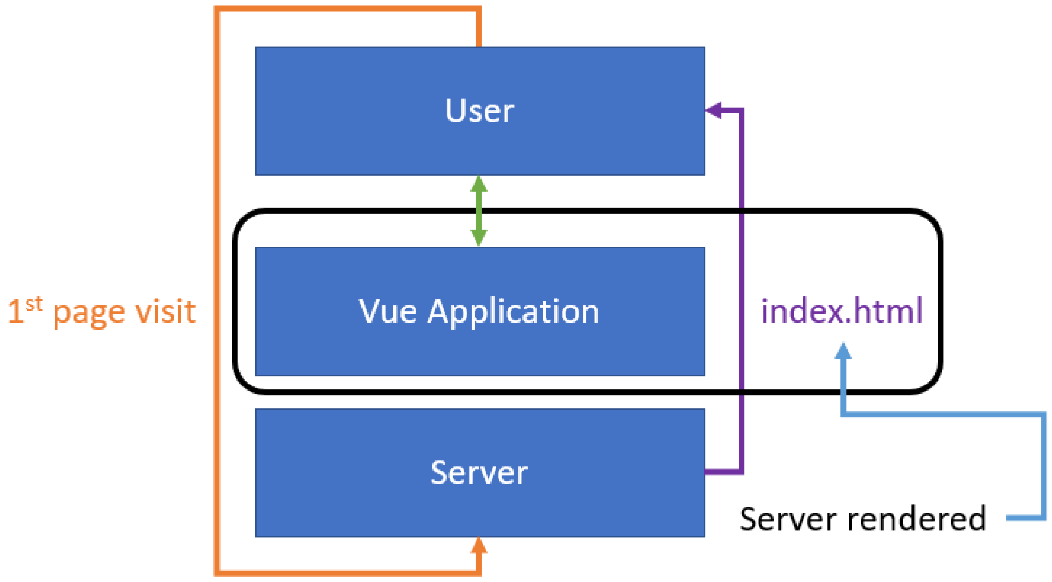


Figure 1-b: Server-side rendering

The difference, in this case, is that although the Vue.js application still manages the **index.html** file, the server can send back a non-empty index.html file on the first request that has been prerendered.

From there onward, we are back in single-page application mode, and no secondary HTML file gets rendered—as Vue.js handles all navigation actions from that point.

Nuxt.js helps in that initial load, pre-rendered on the server, which results in a non-empty HTML response, thus improving page load performance and SEO. This is what server-side rendering is all about.

Nuxt.js server-side rendering

Now that I've covered the difference between client-side and server-side rendering, I'd like to bring another essential item to your attention.

If you've worked extensively with Vue.js, you might already know that it's possible to implement server-side rendering without using a framework like Nuxt.js. This topic is officially covered in the Vue.js documentation. It's well covered there, and there's a complete guide that goes through all the steps required to implement it. However, it's not an easy thing to do.

Even though that guide exists, Nuxt.js abstracts most of that complexity away. The advantage of using Nuxt.js instead of implementing server-side rendering using Vue.js is that you get it ready to go, out of the box, and highly optimized.

With Nuxt.js, to get server-side rendering working, we need to create a new project—that's it (as we'll see shortly). There's no need to fiddle with Vue.js configuration.

So, to put things into perspective, Nuxt.js didn't invent server-side rendering and is not reinventing the wheel. All Nuxt.js does is make it super easy for any Vue.js application to have highly optimized server-side rendering out of the box—making development with Vue.js a cleaner and better experience.

Installing Node.js

To install Nuxt.js, you first need to have Node.js installed. If you don't have Node.js installed, you can download it from the official site.

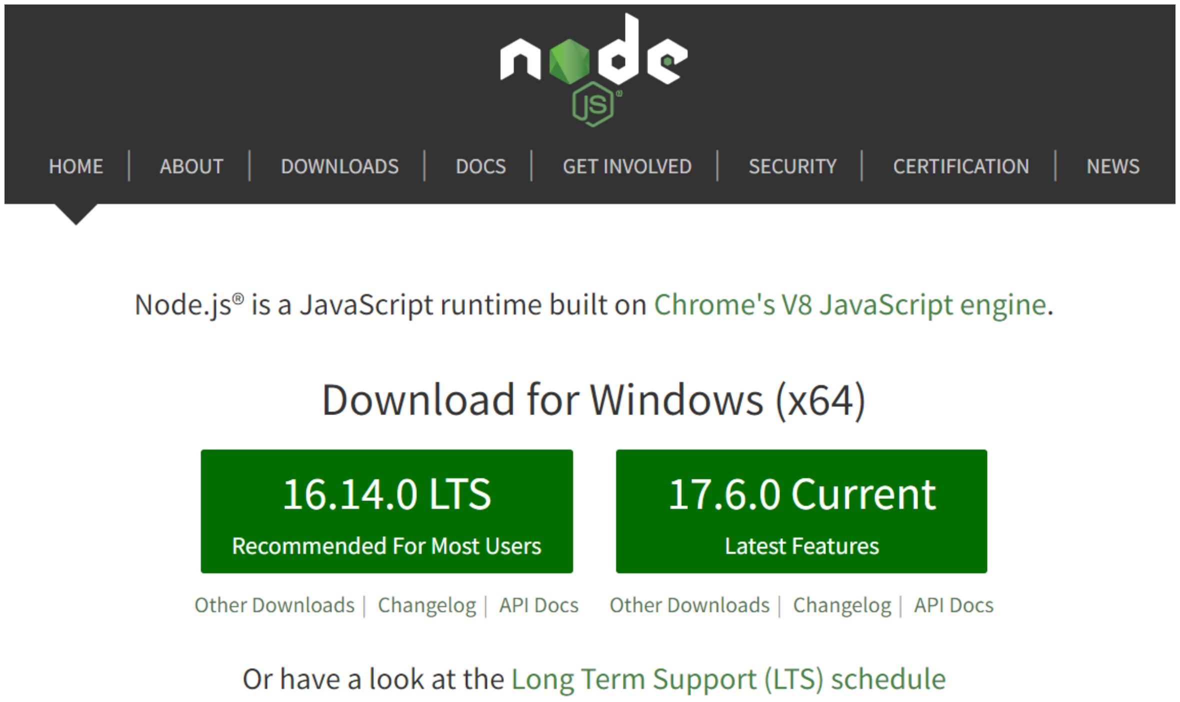

HOME
ABOUT
DOWNLOADS
DOCS
GET INVOLVED
SECURITY
CERTIFICATION
NEWS
Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.
Download for Windows (x64)
16.14.0 LTS
Recommended For Most Users
17.6.0 Current
Latest Features
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs
Or have a look at the Long Term Support (LTS) schedule

Figure 1-c: Node.js Official Website

You can choose to download either the long-term support (LTS) or the current version—either is fine. I will install the LTS version. Once you have downloaded Node.js, execute the installer file.

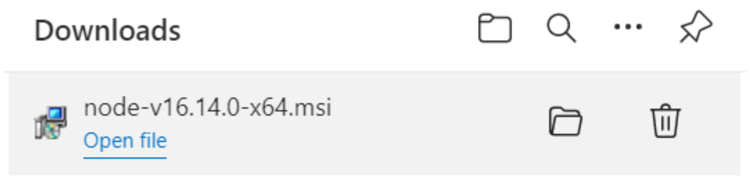
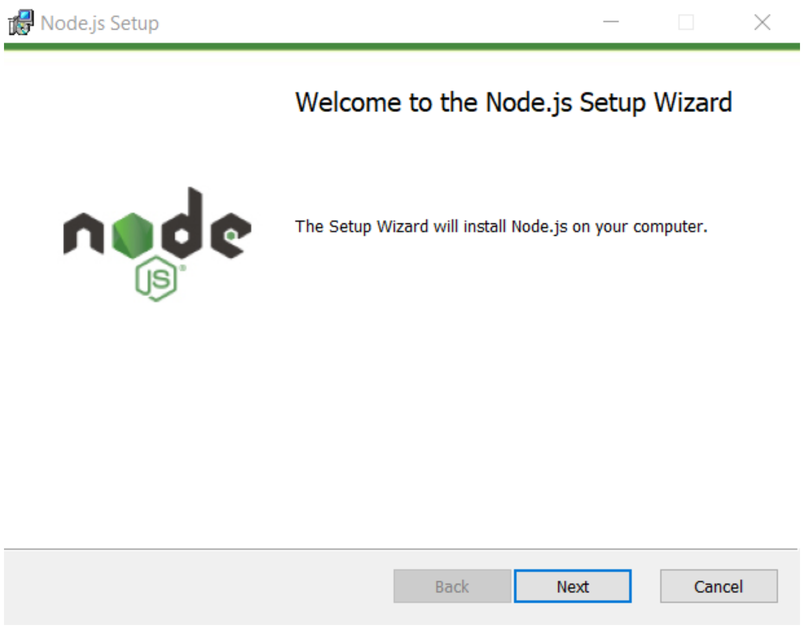


Figure 1-d: Node.js Installer

Once you have executed the installer, you'll see the following screen. Click **Next** to continue the installation process.

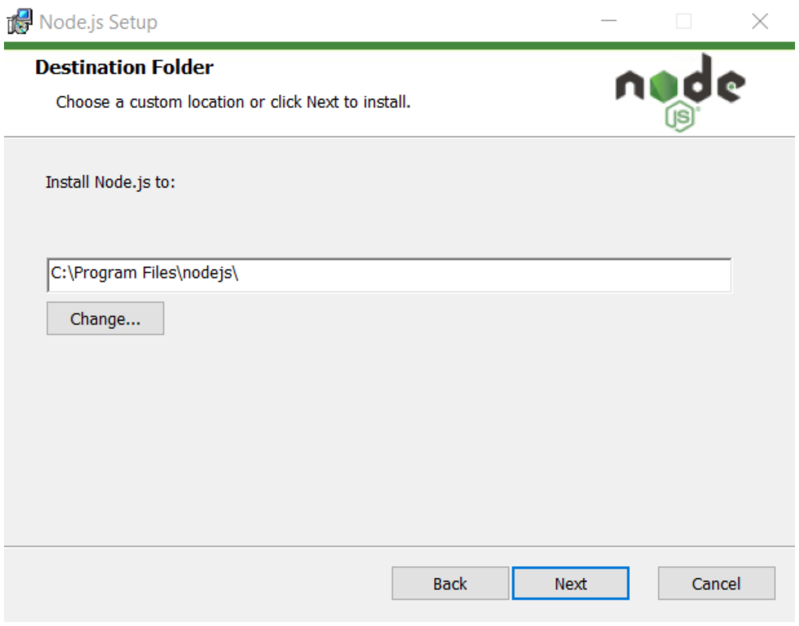


Welcome to the Node.js Setup Wizard

The Setup Wizard will install Node.js on your computer.

Figure 1-e: Initial Node.js Installation Screen

Then, you'll be asked to accept the license terms. Click **Next** to carry on with the installation. At this stage, you'll be presented with a screen where you can select the Node.js installation folder.



Destination Folder

Choose a custom location or click Next to install.

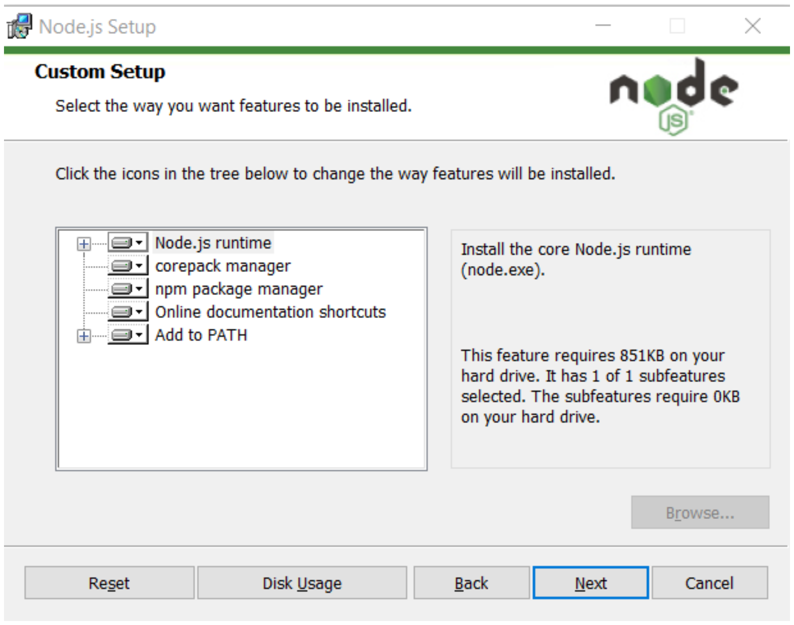
Install Node.js to:

C:\Program Files\nodejs\

Figure 1-f: Node.js Installation (Destination Folder Screen)

I usually leave the default installation folder and click **Next**; however, you can choose a different folder if you prefer. With that done, click **Next** to continue the installation.

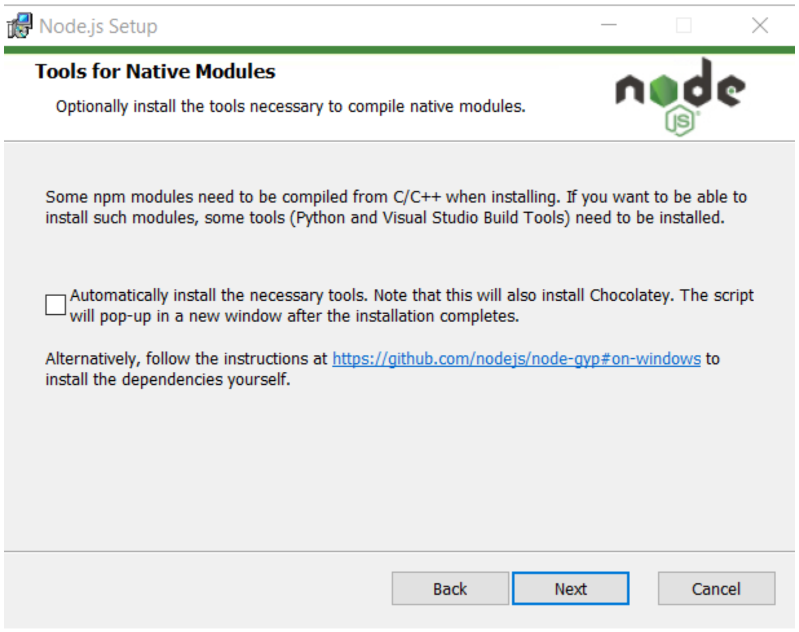
At this point, you'll see the Custom Setup screen. In my case, I always use the default options, as you can see in the following figure.



Node.js Setup
Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
corepack manager
npm package manager
Online documentation shortcuts
Add to PATH
Install the core Node.js runtime (node.exe).
This feature requires 851KB on your hard drive. It has 1 of 1 subfeatures selected. The subfeatures require 0KB on your hard drive.
Browse...
Reset
Disk Usage
Back
Next
Cancel

Figure 1-g: Node.js Installation (Custom Setup Screen)

To continue the installation, click **Next**. You should see the following screen.



Node.js Setup
Tools for Native Modules
Optionally install the tools necessary to compile native modules.
Back
Next
Cancel

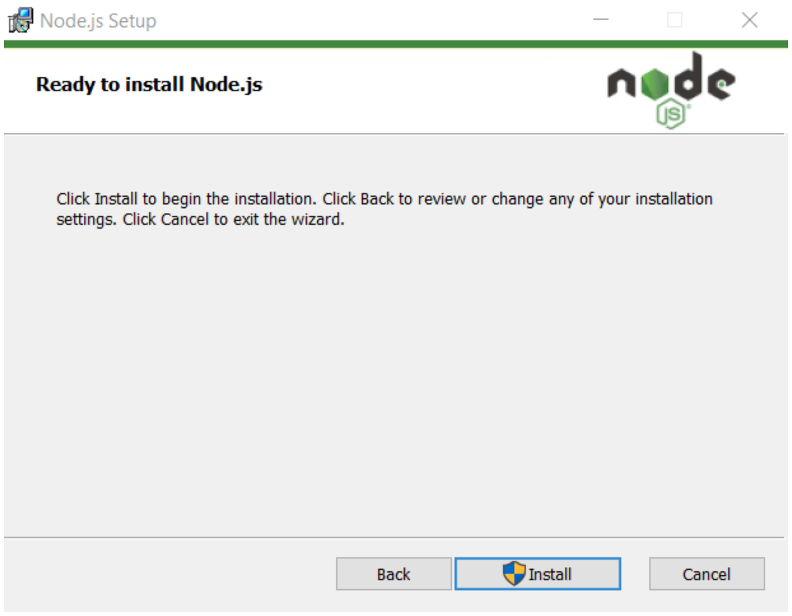
Some npm modules need to be compiled from C/C++ when installing. If you want to be able to install such modules, some tools (Python and Visual Studio Build Tools) need to be installed.

☐ Automatically install the necessary tools. Note that this will also install Chocolatey. The script will pop-up in a new window after the installation completes.

Alternatively, follow the instructions at https://github.com/nodejs/node-gyp#on-windows to install the dependencies yourself.

Figure 1-h: Node.js Installation (Tools for Native Modules Screen)

You may choose to select the option **Automatically install the necessary tools** at this stage. When this option is selected, it allows the Node.js installer to install any other dependency needed using Chocolatey. To continue the installation, click **Next**.



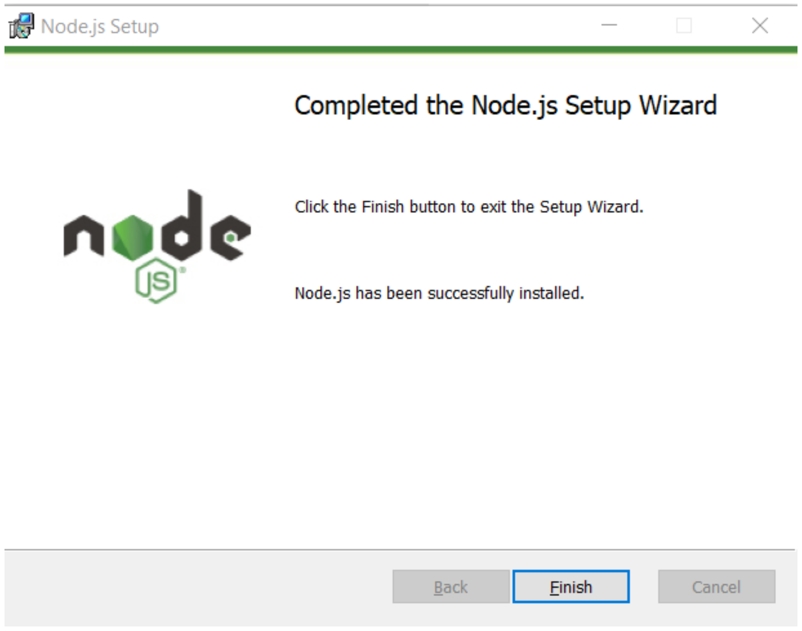
Ready to install Node.js

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Figure 1-i: Node.js Installation (Ready to install Node.js Screen)

We need to click **Install** at this stage. Doing that will deploy the Node.js runtime and files on the installation folder previously selected. The process is usually quick.

If a previous version of Node.js exists on the machine, that version gets removed before the newer version is deployed. Once the new files have been installed, you'll see the following screen.



Completed the Node.js Setup Wizard

Click the Finish button to exit the Setup Wizard.

Node.js has been successfully installed.

Figure 1-j: Node.js Installation (Node.js Setup Wizard Screen—Finish)

To finalize the installation, all we need to do is click **Finish**. Now Node.js is installed, and we can install Nuxt.js.

Getting started with Nuxt.js

At the time of writing this book, the latest version of Nuxt.js is version 3 (in beta)—which is the version that we'll be working with throughout this book, as it includes support for the latest Vue.js features released with version 3.

Installing Nuxt.js is straightforward. From the terminal, command line, or from the built-in terminal within Visual Studio Code (VS Code)—which is my editor of choice—type in the following command. Feel free to use another editor; however, I recommend VS Code so you can follow along easily.

Code Listing 1-a: Command to Create a New Nuxt.js Project

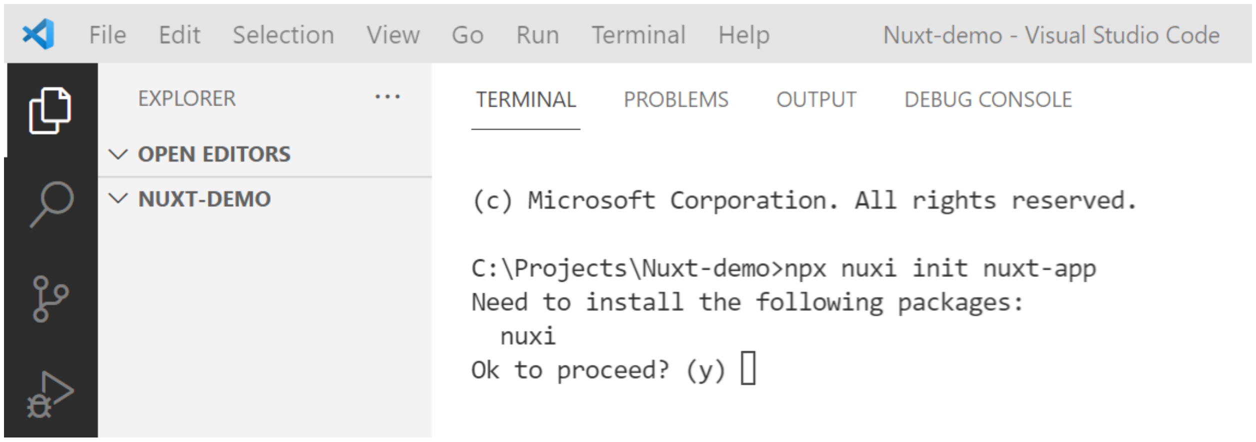
```
npx nuxi init nuxt-app
```

Note: The name of the Nuxt.js application being created is *nuxt-app****; however, you may choose a different name. I suggest using the same name to follow along easily.***

Tip: For more information on how the *npx* ***command works, please check out the official NPM documentation.***

Once you have executed this command, you'll be asked if you want to install the Nuxi NPM package, which is the new Nuxt.js CLI experience.

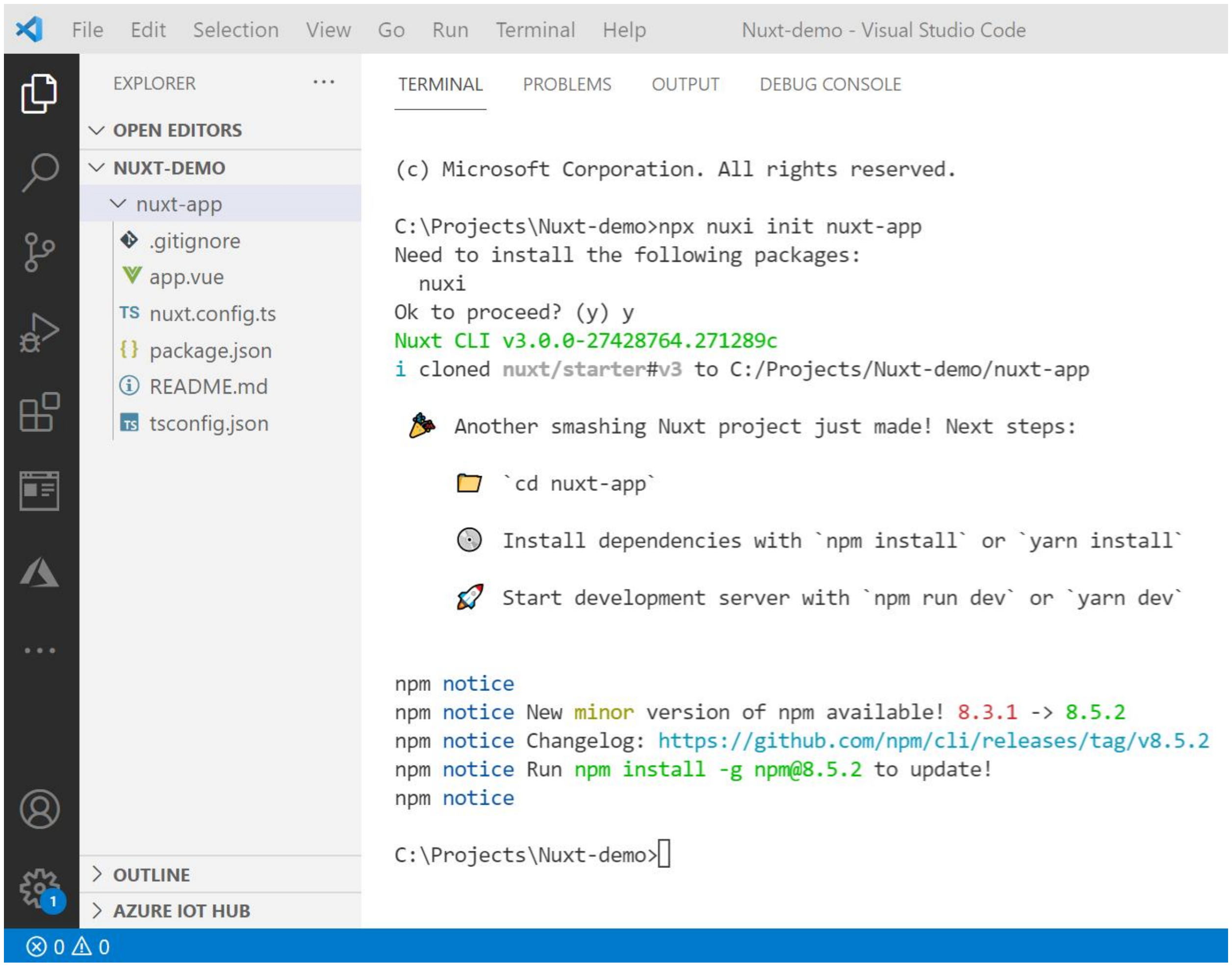
Note: NPM stands for node package manager.



```
(c) Microsoft Corporation. All rights reserved.

C:\Projects\Nuxt-demo>npx nuxi init nuxt-app
Need to install the following packages:
  nuxi
Ok to proceed? (y)
```

Figure 1-k: VS Code Integrated Terminal (Installing Nuxt.js)

To continue, type in **y** and press enter—this will install the CLI and scaffold a new Nuxt.js project, as you can see in the following figure.



```
(c) Microsoft Corporation. All rights reserved.

C:\Projects\Nuxt-demo>npx nuxi init nuxt-app
Need to install the following packages:
  nuxi
Ok to proceed? (y) y
Nuxt CLI v3.0.0-27428764.271289c
i cloned nuxt/starter#v3 to C:/Projects/Nuxt-demo/nuxt-app

 🎉 Another smashing Nuxt project just made! Next steps:

     📁 `cd nuxt-app`

     💿 Install dependencies with `npm install` or `yarn install`

     🚀 Start development server with `npm run dev` or `yarn dev`


npm notice
npm notice New minor version of npm available! 8.3.1 -> 8.5.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.5.2
npm notice Run npm install -g npm@8.5.2 to update!
npm notice

C:\Projects\Nuxt-demo>
```

Figure 1-l: VS Code Integrated Terminal (Nuxt.js Installed)

With Nuxt.js installed and the project scaffolded, we can go into the newly created Nuxt.js application folder, called nuxt-app, with the following command.

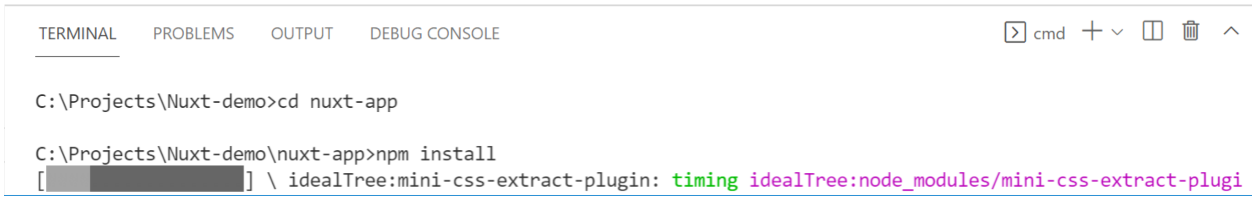
Code Listing 1-b: Command to Change to the Project Folder

```
cd nuxt-app
```

Once we are inside the nuxt-app folder, we can enter the **`npm install`** command at the prompt to install all the required packages and dependencies that our project will need.

Code Listing 1-c: Command to Install Project Dependencies

```
npm install
```



```
C:\Projects\Nuxt-demo>cd nuxt-app

C:\Projects\Nuxt-demo\nuxt-app>npm install
[                  ] \ idealTree:mini-css-extract-plugin: timing idealTree:node_modules/mini-css-extract-plugi
```

Figure 1-m: Installing Project Dependencies (VS Code)

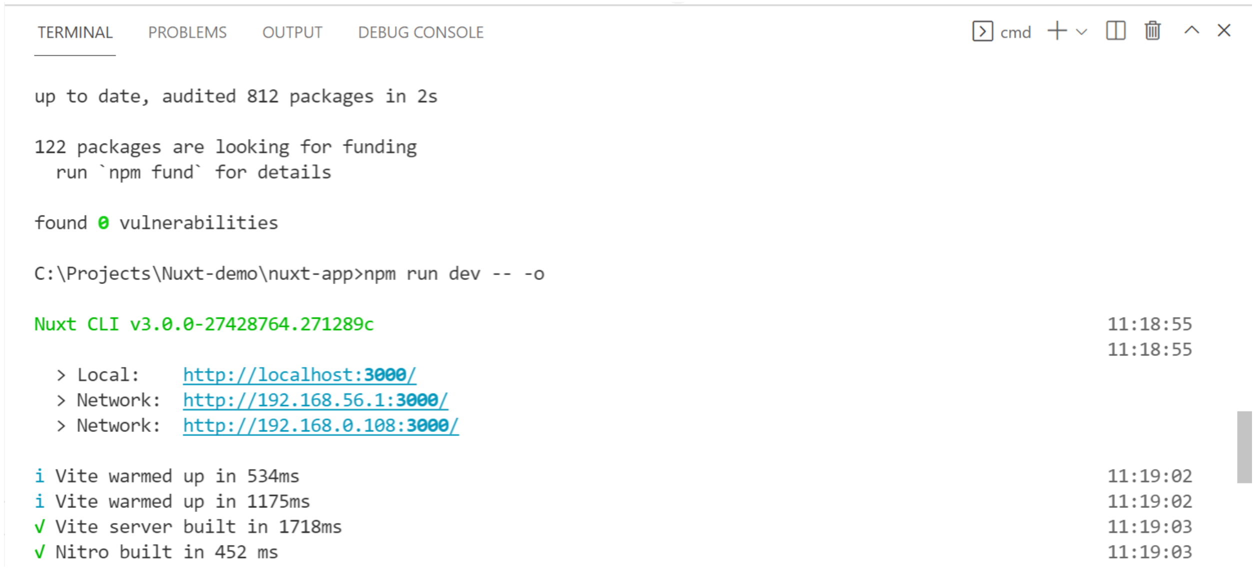
Note: Although it is possible to use *yarn* ***instead of*** *npm****, throughout this book, I'll be using*** *npm****. You may well choose to use*** *yarn****, though.***

With that done, let's run the scaffolded application in development mode and have a look. You can do this by executing the following command.

Code Listing 1-d: Command to Execute the Project in Dev Mode

```
npm run dev -- -o
```

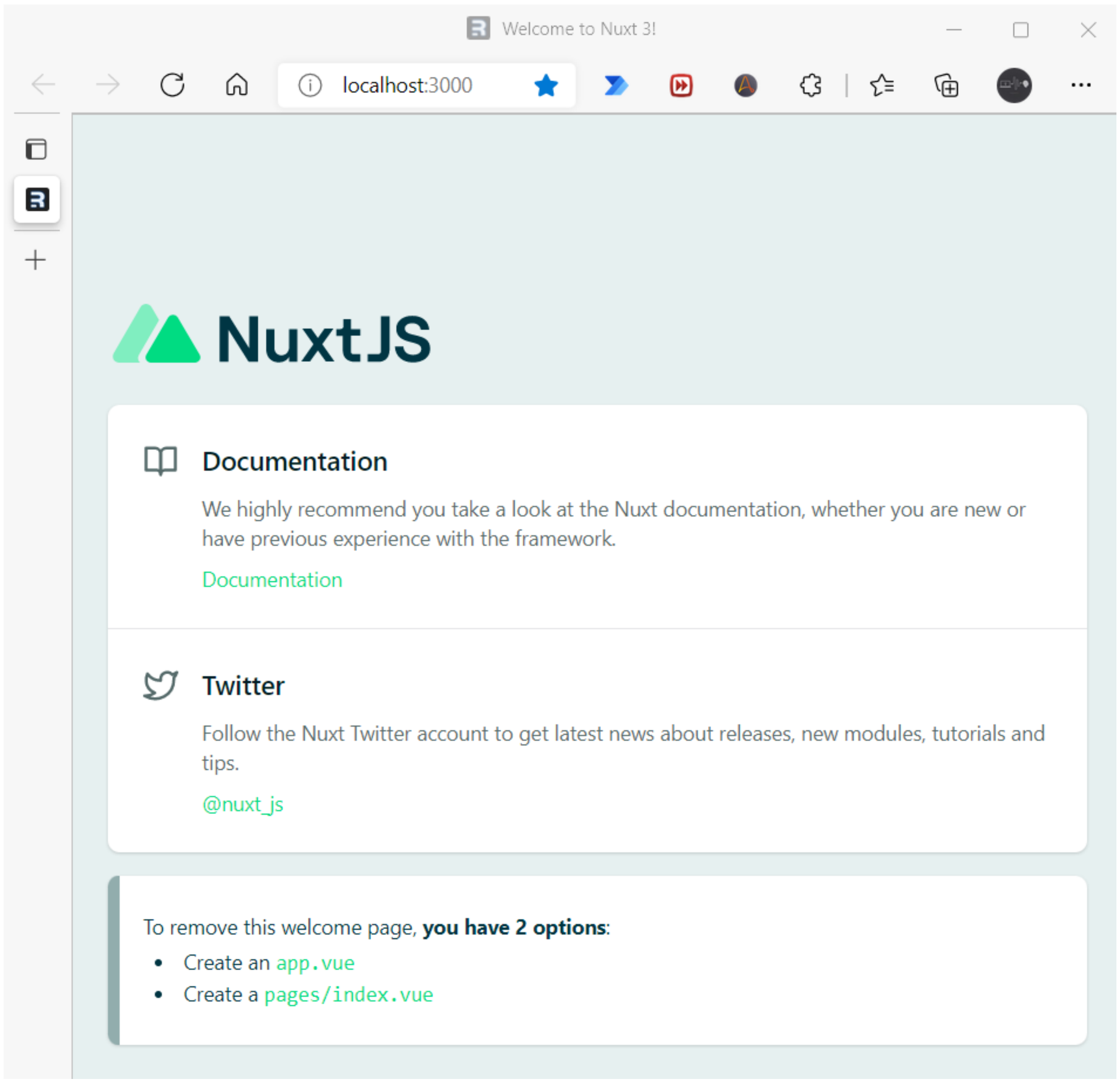
After executing this command, you should see the following within the built-in terminal in VS Code. The `--` as an argument on its own means further arguments should be treated as positional arguments, not options.



```
up to date, audited 812 packages in 2s

122 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Projects\Nuxt-demo\nuxt-app>npm run dev -- -o

Nuxt CLI v3.0.0-27428764.271289c                                          11:18:55
                                                                          11:18:55
  > Local:    http://localhost:3000/
  > Network:  http://192.168.56.1:3000/
  > Network:  http://192.168.0.108:3000/

i Vite warmed up in 534ms                                                 11:19:02
i Vite warmed up in 1175ms                                                11:19:02
√ Vite server built in 1718ms                                             11:19:03
√ Nitro built in 452 ms                                                   11:19:03
```

Figure 1-n: Execution of the Project

To visualize the Nuxt.js project running, you'll need to open it in a modern browser, which you can do by typing the local URL, or clicking directly on the URL link, as seen in the preceding figure.


Welcome to Nuxt 3!

localhost:3000

NuxtJS

Documentation

We highly recommend you take a look at the Nuxt documentation, whether you are new or have previous experience with the framework.

Documentation

Twitter

Follow the Nuxt Twitter account to get latest news about releases, new modules, tutorials and tips.

@nuxt_js

To remove this welcome page, **you have 2 options**:

- Create an `app.vue`
- Create a `pages/index.vue`

Figure 1-o: Nuxt.js Starter Project Running

To stop the execution of the application, you can press **Ctrl+C** within the integrated VS Code terminal.

Summary

At this stage, we have successfully installed Nuxt.js and scaffolded a basic application, which, as you have seen, was straightforward.

The next chapter will explore the project structure and explain how pages, views, and routing work together.

Chapter 2 Project Structure

Initial project structure

An essential part of understanding Nuxt.js is knowing a project's folder structure. That's what this chapter is all about.

Let's begin by looking at the scaffolded project's folder structure. If you have VS Code open, ensure that the EXPLORER panel is visible. It displays the list of folders and files that are part of the project.

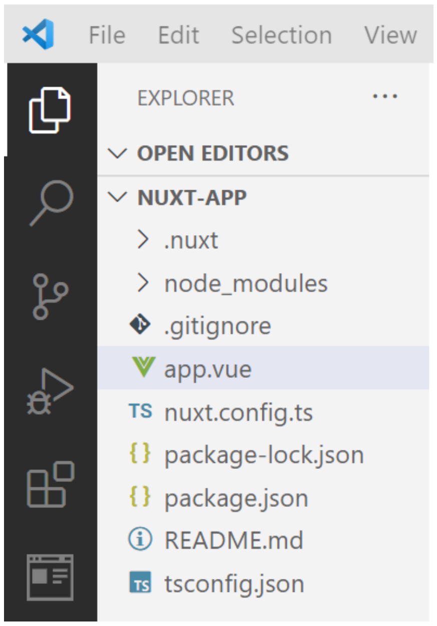


Figure 2-a: VS Code (EXPLORER—Project Folders and Files)

Notice that the project doesn't (yet) include any project-specific folders. It consists only of the .nuxt folder, which contains all the core Nuxt.js engine and configuration, and the node_modules folder, which includes the dependencies required by Nuxt.js.

So, there aren't any project-related folders specific to the Nuxt.js app itself. This means that this project we created is a barebones Nuxt.js app, which is nothing more than an empty shell, and we would need to create project-specific folders manually.

This is the path we will take—to create the folder and file structure ourselves as we go. However, there's an alternative way that I'd like to show you.

Create Nuxt app

There's a tool that works with Nuxt.js version 2 called create-nuxt-app, which can scaffold a Nuxt.js (version 2) application with a complete folder structure in no time.

At the time of writing this book, **`create-nuxt-app`** does not yet support Nuxt.js version 3; nevertheless, I'd like to cover it briefly.

Creating a Nuxt.js (version 2) application with **`create-nuxt-app`** is straightforward. You need to execute the following command and replace **`<project-name>`** with the name of your app, where **`npx`** stands for "node package execute."

Code Listing 2-a: Command to Install create-nuxt-app and Scaffold a Nuxt.js (v2) App

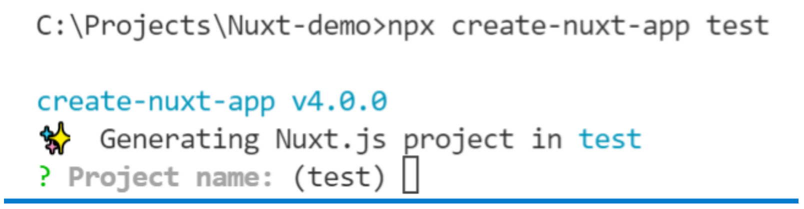
```
npx create-nuxt-app <project-name>
```

Let's go through these steps using VS Code. I will type the following command using the built-in VS Code terminal, within my root directory.

Code Listing 2-b Create the test Nuxt.js (v2) App

```
npx create-nuxt-app test
```

After I press Enter, **`create-nuxt-app`** asks to confirm the name of the application.

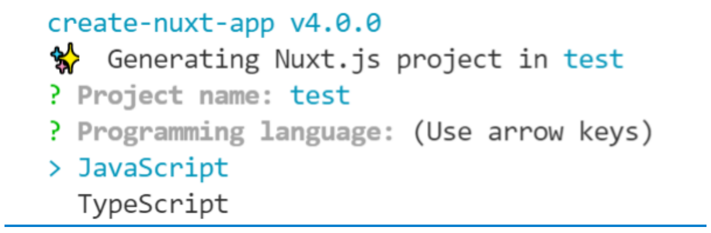


```
C:\Projects\Nuxt-demo>npx create-nuxt-app test

create-nuxt-app v4.0.0
✨  Generating Nuxt.js project in test
? Project name: (test)
```

Figure 2-b: Project Name— create-nuxt-app (Built-in Terminal—VS Code)

At this stage, if I press Enter, the project name will be set as **test**. However, I could also choose to type in a different name for the project. I'll stick with test and press Enter.

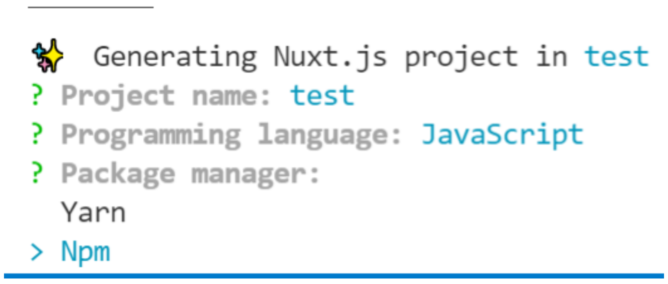
Then, **`create-nuxt-app`** requests the programming language we want to use for our Nuxt.js (version 2) application. We can change the programming language by using the arrow keys. In my case, I'll choose JavaScript. To select it, I press Enter.



```
create-nuxt-app v4.0.0
✨  Generating Nuxt.js project in test
? Project name: test
? Programming language: (Use arrow keys)
> JavaScript
  TypeScript
```

Figure 2-c: Programming language— create-nuxt-app (Built-in Terminal—VS Code)

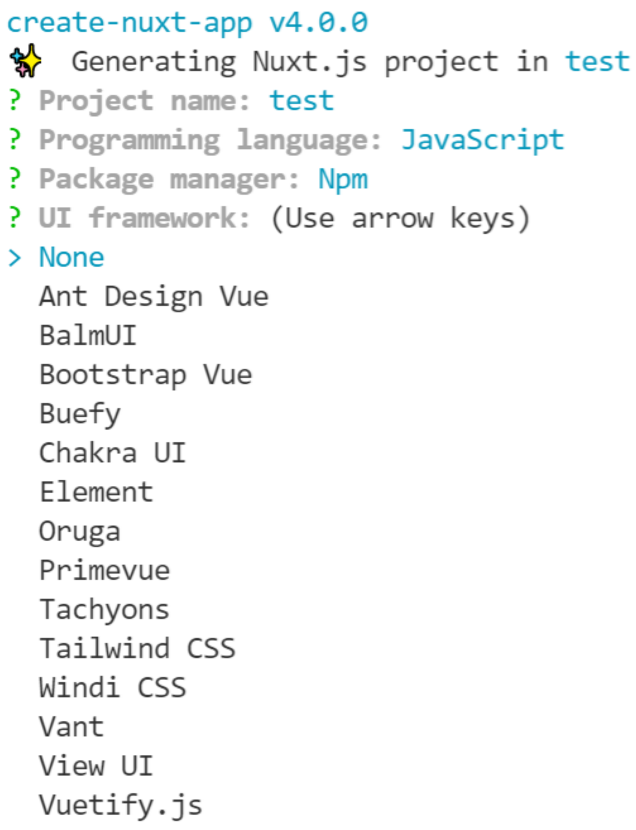
The next step is to choose the package manager our application will use. Although **`Yarn`** is also valid, I will select the **`Npm`** option using the down arrow key, and then press Enter.



```
✨  Generating Nuxt.js project in test
? Project name: test
? Programming language: JavaScript
? Package manager:
  Yarn
> Npm
```

Figure 2-d: Package Manager— create-nuxt-app (Built-in Terminal—VS Code)

Next, there's the option to select one of several UI frameworks. I won't be choosing any UI framework, as the goal is to walk you through these steps. However, if you were using **`create-nuxt-app`** to scaffold an app you're creating, you could choose a UI framework that would best fit your project.

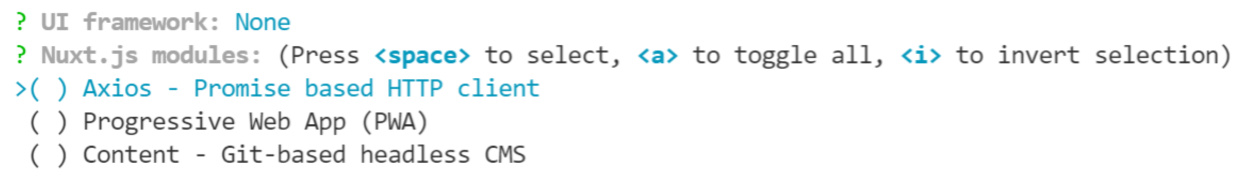


```
create-nuxt-app v4.0.0
✨  Generating Nuxt.js project in test
? Project name: test
? Programming language: JavaScript
? Package manager: Npm
? UI framework: (Use arrow keys)
> None
  Ant Design Vue
  BalmUI
  Bootstrap Vue
  Buefy
  Chakra UI
  Element
  Oruga
  Primevue
  Tachyons
  Tailwind CSS
  Windi CSS
  Vant
  View UI
  Vuetify.js
```

Figure 2-e: UI Framework—create-nuxt-app (Built-in Terminal—VS Code)

Next, there's the option to add additional modules, such as the [Axios]() HTTP client library, or add [progressive web app]() (PWA) capabilities to the Nuxt.js application.

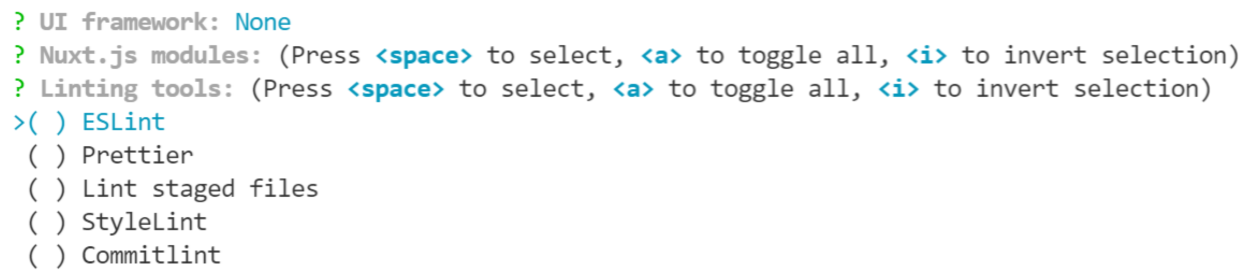
For most applications, you would probably need to choose Axios—which can be done by pressing the space key. Since this is just a walkthrough, I won't select an option. To continue, I'll press Enter.



```
? UI framework: None
? Nuxt.js modules: (Press <space> to select, <a> to toggle all, <i> to invert selection)
>( ) Axios - Promise based HTTP client
 ( ) Progressive Web App (PWA)
 ( ) Content - Git-based headless CMS
```

Figure 2-f: Nuxt.js Modules— create-nuxt-app (Built-in Terminal—VS Code)

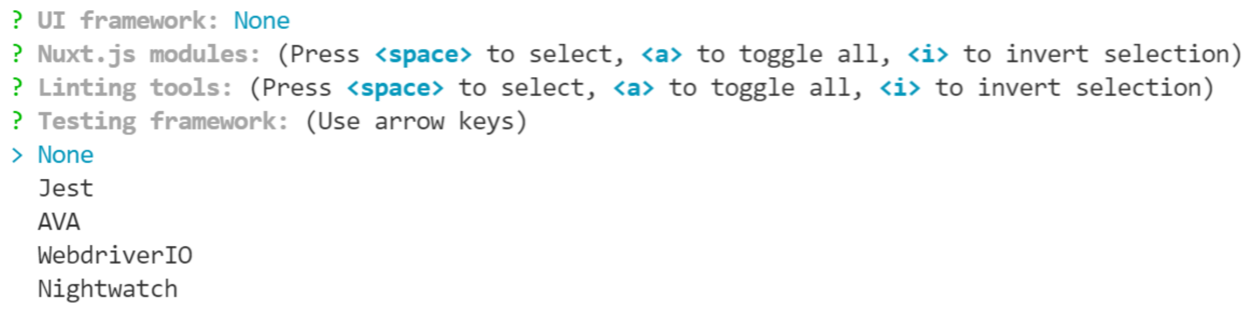
Next, there's the option to select one or more Linting tools. To continue, I press Enter.



```
? UI framework: None
? Nuxt.js modules: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Linting tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
>( ) ESLint
 ( ) Prettier
 ( ) Lint staged files
 ( ) StyleLint
 ( ) Commitlint
```

Figure 2-g: Linting Tools— create-nuxt-app (Built-in Terminal—VS Code)

Next, it is possible to choose a Testing framework. I will leave it set to None and press Enter.

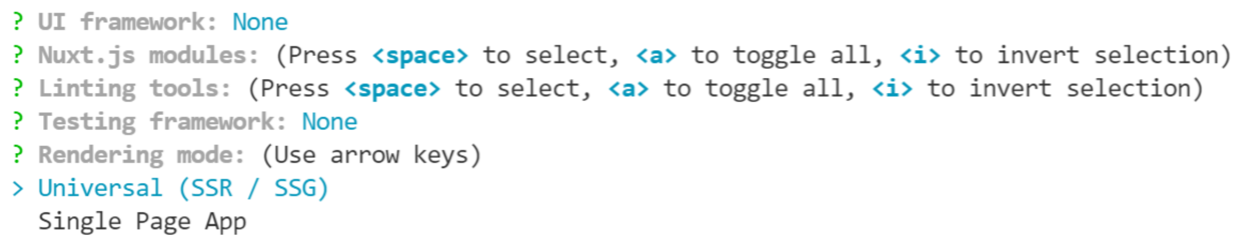


```
? UI framework: None
? Nuxt.js modules: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Linting tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Testing framework: (Use arrow keys)
> None
  Jest
  AVA
  WebdriverIO
  Nightwatch
```

Figure 2-h: Testing Framework— create-nuxt-app (Built-in Terminal—VS Code)

Then, it is possible to select the application's rendering mode. I will choose **Universal**, which includes server-side rendering (SSR)—which is the whole purpose of using Nuxt.js in the first place.

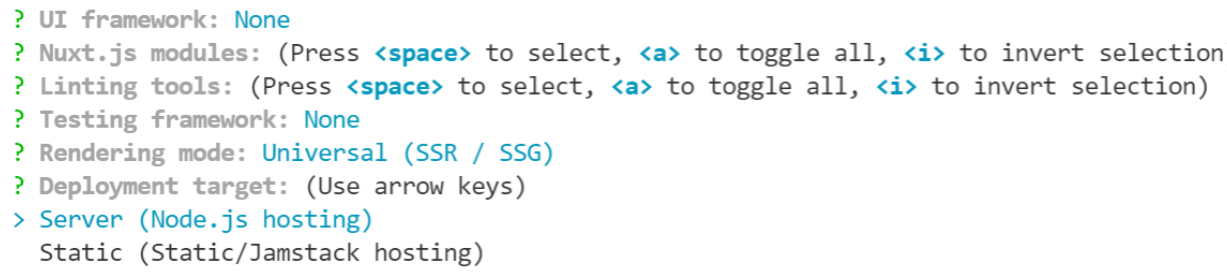
It is also possible to select **Single Page App**, but that doesn't add additional value; for that purpose, we can use Vue.js instead of Nuxt.js.



```
? UI framework: None
? Nuxt.js modules: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Linting tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Testing framework: None
? Rendering mode: (Use arrow keys)
> Universal (SSR / SSG)
  Single Page App
```

Figure 2-i: Rendering Mode— create-nuxt-app (Built-in Terminal —VS Code)

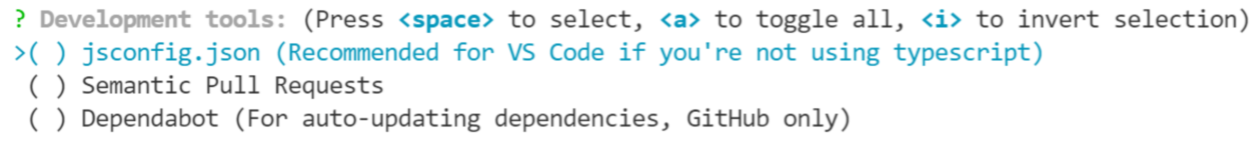
Following that, it is possible to select the deployment target. I will go with the **Server** option, which allows us to use Node.js hosting as a backend rather than generating a static site.



```
? UI framework: None
? Nuxt.js modules: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Linting tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
? Testing framework: None
? Rendering mode: Universal (SSR / SSG)
? Deployment target: (Use arrow keys)
> Server (Node.js hosting)
  Static (Static/Jamstack hosting)
```

Figure 2-j: Deployment Target—create-nuxt-app (Built-in Terminal—VS Code)

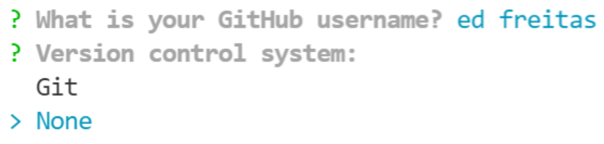
Next, it is possible to choose additional development tools. I won't select any options. So, to continue, I press Enter.



```
? Development tools: (Press <space> to select, <a> to toggle all, <i> to invert selection)
>( ) jsconfig.json (Recommended for VS Code if you're not using typescript)
 ( ) Semantic Pull Requests
 ( ) Dependabot (For auto-updating dependencies, GitHub only)
```

Figure 2-k: Development Tools— create-nuxt-app (Built-in Terminal—VS Code)

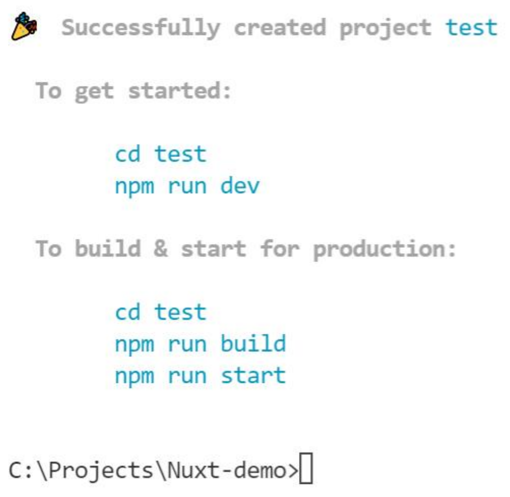
Finally, I will indicate my GitHub username, and as the Version control system, select **None**. In a real-world project (and not a walkthrough), I would suggest you choose Git.



```
? What is your GitHub username? ed freitas
? Version control system:
  Git
> None
```

Figure 2-l: GitHub Username and Version Control System— create-nuxt-app (Built-in Terminal—VS Code)

We can see that the project is now successfully created.



```
🎉  Successfully created project test

  To get started:

        cd test
        npm run dev

  To build & start for production:

        cd test
        npm run build
        npm run start


C:\Projects\Nuxt-demo>
```

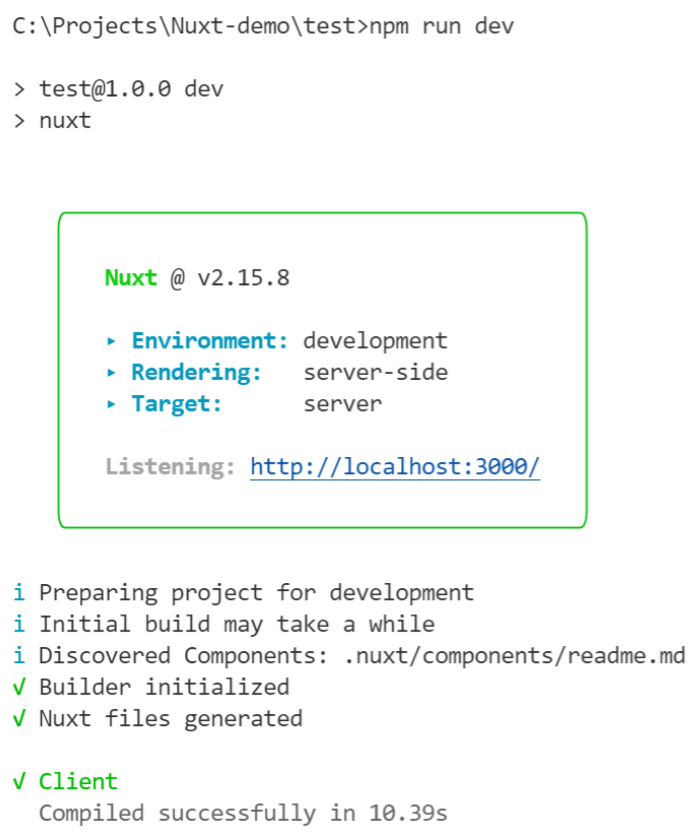
Figure 2-m: Project Created— create-nuxt-app (Built-in Terminal—VS Code)

We can use the following commands to run the scaffolded application.

Code Listing 2-c: Execute the Test Nuxt.js (v2) App

```
cd test
npm run dev
```

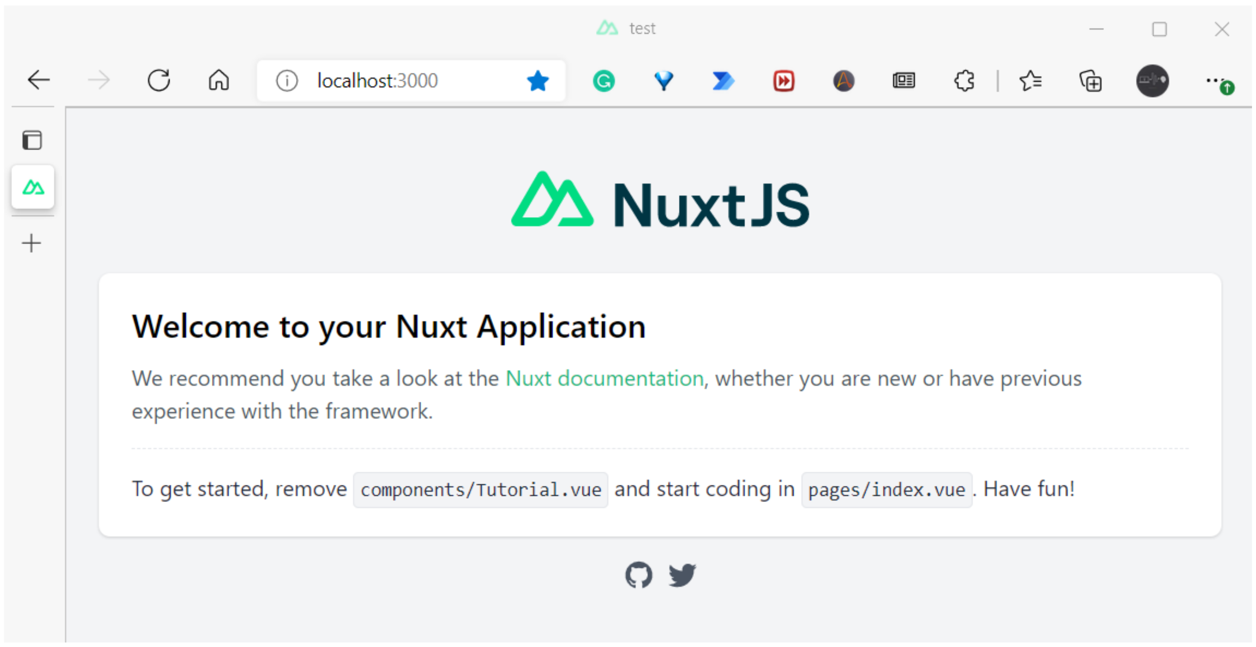
After executing these commands, we can see the scaffolded test application running.



```
C:\Projects\Nuxt-demo\test>npm run dev

> test@1.0.0 dev
> nuxt


   Nuxt @ v2.15.8

   ▸ Environment: development
   ▸ Rendering:   server-side
   ▸ Target:      server

   Listening: http://localhost:3000/


i Preparing project for development
i Initial build may take a while
i Discovered Components: .nuxt/components/readme.md
√ Builder initialized
√ Nuxt files generated

√ Client
  Compiled successfully in 10.39s
```

Figure 2-n: Project Executing—create-nuxt-app (Built-in Terminal—VS Code)

If we open the localhost URL, we can see the scaffolded application running.



Welcome to your Nuxt Application

We recommend you take a look at the Nuxt documentation, whether you are new or have previous experience with the framework.

To get started, remove `components/Tutorial.vue` and start coding in `pages/index.vue`. Have fun!

Figure 2-o: Scaffolded App Running

To stop the execution of the application, press Ctrl+C from the built-in terminal within VS Code.

To learn more about working with the **`create-nuxt-app`** tool, you can find additional details in the official documentation.

Project structure (create-nuxt-app)

Now that we have explored how the **`create-nuxt-app`** tool works, let's look at the project structure that has been created.

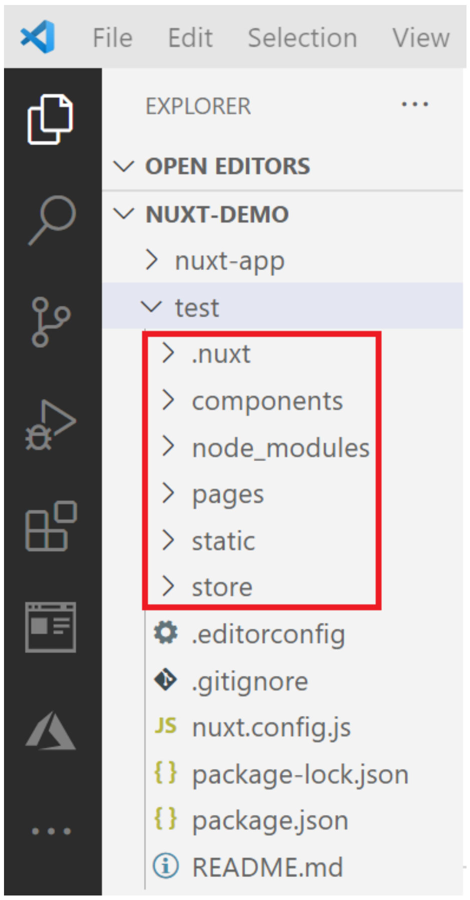


Figure 2-p: Project Folders

We can see that the .nuxt and node_modules have been added, just like when we scaffolded the Nuxt.js 3 application. However, notice that the components, pages, static, and store folders have been added to the project.

As its name implies, the components folder is used for adding Vue.js components to the Nuxt.js application.

The pages folder is how Nuxt.js addresses routing, and each file within that folder constitutes a page of a Nuxt.js application. Vue.js and Nuxt.js applications are made up of pages, and each page can contain one or more components.

The static folder stores the application's static resources, such as JavaScript, CSS files, and images. On the other hand, the store folder is not required, and it is used for working with Vuex application state storage files.

Summary

Even though we won't be using the **`create-nuxt-app`** tool (as we'll focus on Nuxt.js 3 rather than Nuxt.js 2), I just wanted to show you the resultant folder structure, which is practical and speeds up development.

In the next chapter, we will continue with the original Nuxt.js 3 application we created, explore how to work with it by creating pages, and lay the foundations for the application we'll build throughout this book.

Chapter 3 App Foundations

Getting started

It’s time to look at some code and put in place the foundations of the application we’ll be building, which will be an app to manage a list of favorite books.

The application we will build has been inspired by these excellent GitHub repositories: nuxt-blog-starter-kit and online-store-nuxt3.

With VS Code open, let's open the nuxt-app project we created for Nuxt.js 3. To do so, click **File** > **Open Folder** and select the **nuxt-app** folder where it resides.

With the nuxt-app project open, let’s click the **app.vue** file to make our first change.

Code Listing 3-a: Original app.vue File

```
<template>
  <div>
    <NuxtWelcome />
  </div>
</template>
```

The first thing I'm going to do is replace **`<NuxtWelcome />`** with **`<NuxtPage />`**. What this does is replace the boilerplate code **(`<NuxtWelcome />`**) with the content of the app's current page (**`<NuxtPage />`**).

Code Listing 3-b: Modified app.vue File

```
<template>
  <div>
    <NuxtPage />
  </div>
</template>
```

Once you’ve made this change, press **Ctrl+S** to save this change or click **File** > **Save**.

Pages folder

The next thing we are going to do is create a pages folder. We can do this by clicking on the **New Folder** icon within the VS Code EXPLORER panel.

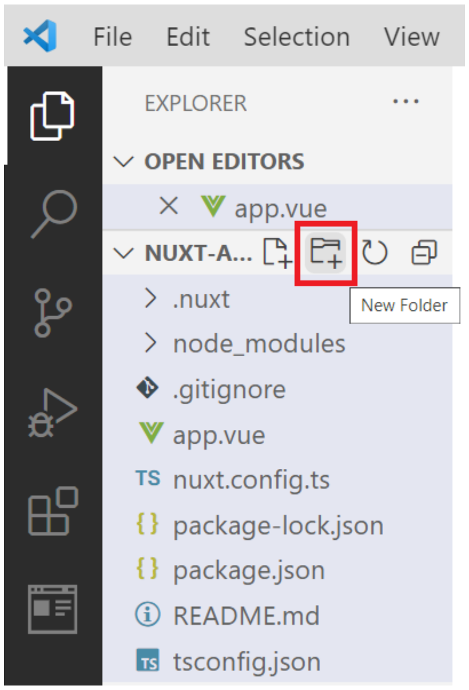


Figure 3-a: New Folder Button (VS Code—EXPLORER)

We can place the app's pages within the pages folder. After creating the pages folder, let's select it, and then click the **New File** icon to add a new file to that folder, which we will call index.vue.

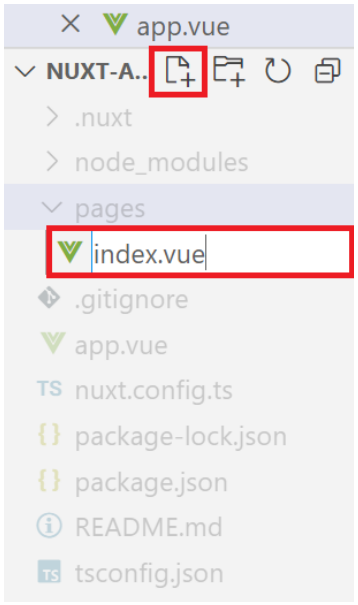
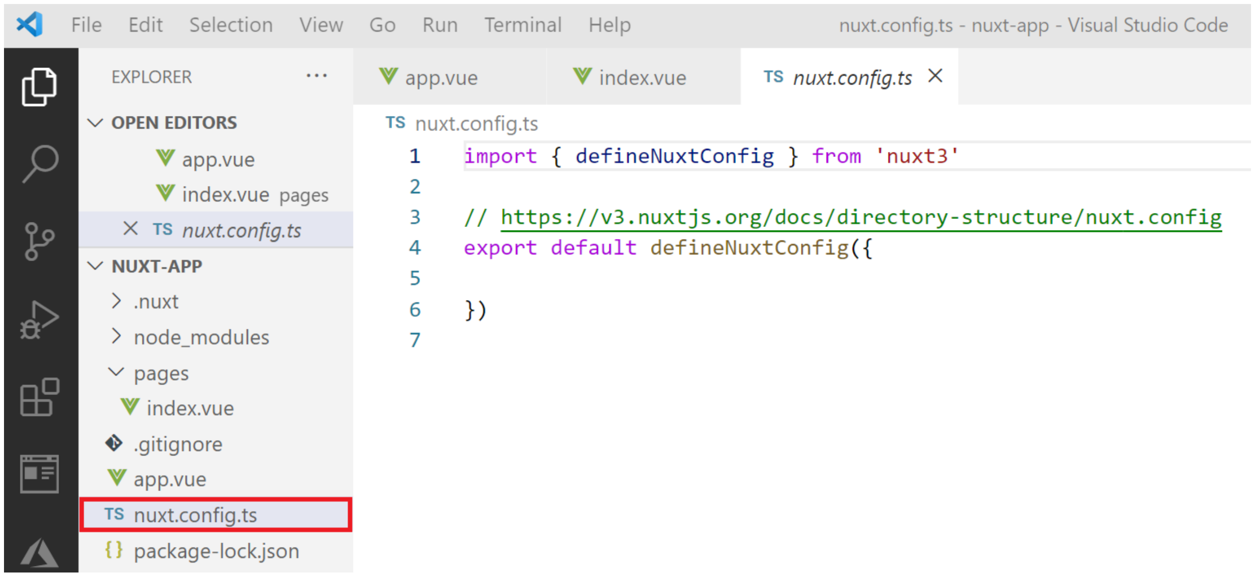


Figure 3-b: New File (VS Code—EXPLORER)

We'll come back to index.vue, as this will be our starting point. For now, let's switch gears and install the latest version of Bootstrap, which is one of the world's most popular front-end toolkits.

Installing Bootstrap

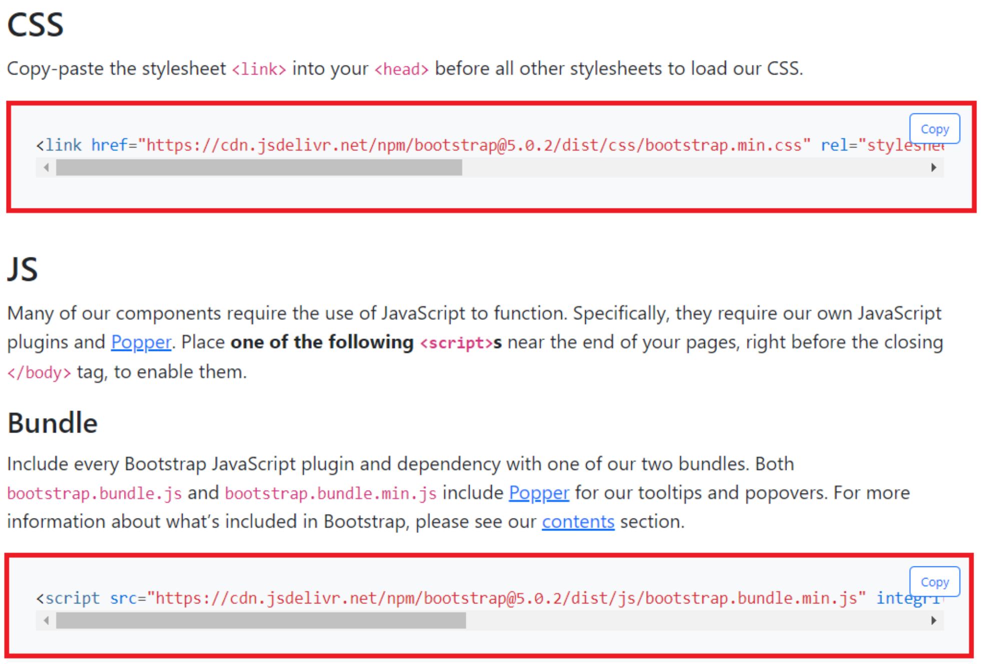
Installing the Bootstrap toolkit is straightforward. Let's open the nuxt.config.ts file using the EXPLORER panel within VS Code.



```
import { defineNuxtConfig } from 'nuxt3'

// https://v3.nuxtjs.org/docs/directory-structure/nuxt.config
export default defineNuxtConfig({

})
```

Figure 3-c: The nuxt.config.ts File—VS Code

First, we need to copy two links from the official documentation, one for the Bootstrap CSS styles, and another for the Bootstrap JavaScript Bundle, as you can see highlighted in **red** in the following figure.



CSS

Copy-paste the stylesheet `<link>` into your `<head>` before all other stylesheets to load our CSS.

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="styleshe
```

JS

Many of our components require the use of JavaScript to function. Specifically, they require our own JavaScript plugins and Popper. Place **one of the following `<script>`s** near the end of your pages, right before the closing `</body>` tag, to enable them.

Bundle

Include every Bootstrap JavaScript plugin and dependency with one of our two bundles. Both `bootstrap.bundle.js` and `bootstrap.bundle.min.js` include Popper for our tooltips and popovers. For more information about what's included in Bootstrap, please see our contents section.

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integri
```

Figure 3-d: Bootstrap CSS and JavaScript Bundle (Bootstrap Official Documentation)

First, manually copy the CSS link (just the string contained within **src**, and not using the Copy button) and paste it (highlighted in **bold**) into the modified nuxt.config.ts file, which looks as follows.

Code Listing 3-c: Modified nuxt.config.ts File (Includes Bootstrap CSS)

```
import { defineNuxtConfig } from 'nuxt3'

// https://v3.nuxtjs.org/docs/directory-structure/nuxt.config
export default defineNuxtConfig({
    meta: {
        link: [
            {
                rel: 'stylesheet',
                href: 'https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/
                 dist/css/bootstrap.min.css'
            }
        ]
    }
})
```

Note: I've written the** CSS **link in two lines (for readability); however, when you paste it in your nuxt.config.ts file, do not split it into two lines—make sure it remains in a single line.

Next, let's add the Bootstrap JavaScript Bundle to the nuxt.config.ts file. Manually copy the Bundle link (just the string contained within `src`, and not using the Copy button) and paste it (highlighted in **bold**) into the modified nuxt.config.ts file. This will look as follows.

Code Listing 3-d: Modified nuxt.config.ts File (Includes Bootstrap CSS & JavaScript Bundle)

```
import { defineNuxtConfig } from 'nuxt3'

// https://v3.nuxtjs.org/docs/directory-structure/nuxt.config
export default defineNuxtConfig({
    meta: {
        link: [
            {
                rel: 'stylesheet',
                href: 'https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/
                 dist/css/bootstrap.min.css'
            }
        ],
        script: [
            {
                type: 'text/javascript',
                src: 'https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/
                      dist/js/bootstrap.bundle.min.js'
            }
        ]
    }
})
```

Note: I've written the** Bundle **link in two lines (for readability); however, when you paste it in your** nuxt.config.ts **file, do not split it into two lines—make sure it remains in a single line.

Once you're done, save the changes to the nuxt.config.ts file. Now, Bootstrap has been installed and is ready to be used.

Components folder

Now that we have Bootstrap, we need to create a **components** folder that we will use to add various components our application will utilize. We can do this by clicking the **New Folder** icon within the VS Code EXPLORER panel, as seen in Figure 3-a.

Once the folder has been created, select it, and click the **New File** icon within the VS Code EXPLORER panel. Let's name the new file **NavBar.vue**.

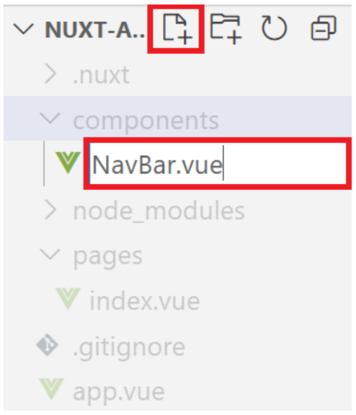


Figure 3-e: Creating NavBar.vue (VS Code—EXPLORER)

After you've created the file, click to open it in VS Code.

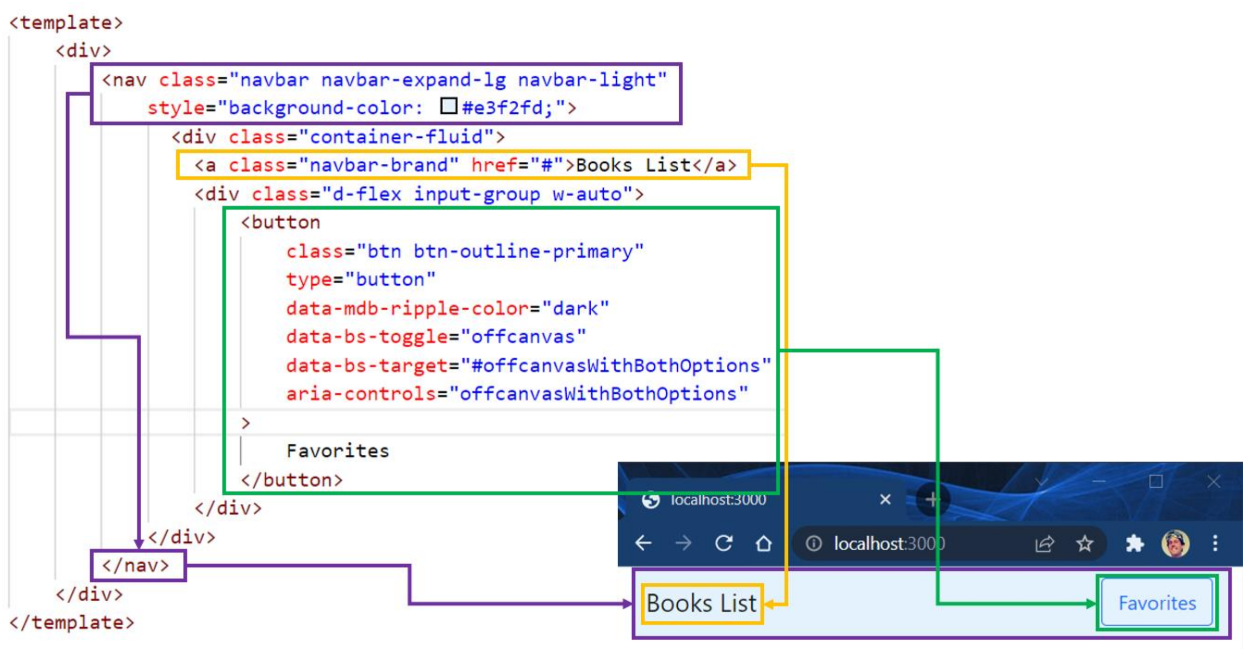
Navbar component

The next thing we want to do is write the **`navbar`** component code. Open **NavBar.vue** and paste in the following code.

Code Listing 3-e: The NavBar Component (NavBar.vue)

```
<template>
    <div>
        <nav class="navbar navbar-expand-lg navbar-light"
            style="background-color: #e3f2fd;">
              <div class="container-fluid">
                <a class="navbar-brand" href="#">Books List</a>
                <button
                    class="navbar-toggler"
                    type="button"
                    data-mdb-toggle="collapse"
                    data-mdb-target="#navbarTogglerDemo02"
                    aria-controls="navbarTogglerDemo02"
                    aria-expanded="false"
                    aria-label="Toggle navigation"
                >
                    <i class="fas fa-bars"></i>
                </button>
                <div class="collapse navbar-collapse"
                    id="navbarTogglerDemo02">
                    <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                    </ul>
                    <div class="d-flex input-group w-auto">
                        <button
                            class="btn btn-outline-primary"
                            type="button"
                            data-mdb-ripple-color="dark"
                            data-bs-toggle="offcanvas"
                            data-bs-target="#offcanvasWithBothOptions"
                            aria-controls="offcanvasWithBothOptions"
                        >
                            Favorites
                        </button>
                    </div>
                </div>
            </div>
        </nav>
    </div>
</template>
```

To understand this code and how it relates to the finished **navbar** component, let's look at the following diagram.



```
<template>
    <div>
        <nav class="navbar navbar-expand-lg navbar-light"
            style="background-color: □#e3f2fd;">
            <div class="container-fluid">
              <a class="navbar-brand" href="#">Books List</a>
              <div class="d-flex input-group w-auto">
                  <button
                      class="btn btn-outline-primary"
                      type="button"
                      data-mdb-ripple-color="dark"
                      data-bs-toggle="offcanvas"
                      data-bs-target="#offcanvasWithBothOptions"
                      aria-controls="offcanvasWithBothOptions"
                  >
                      Favorites
                  </button>
              </div>
            </div>
        </nav>
    </div>
</template>
```

Figure 3-f: Navbar Code and UI Relationship

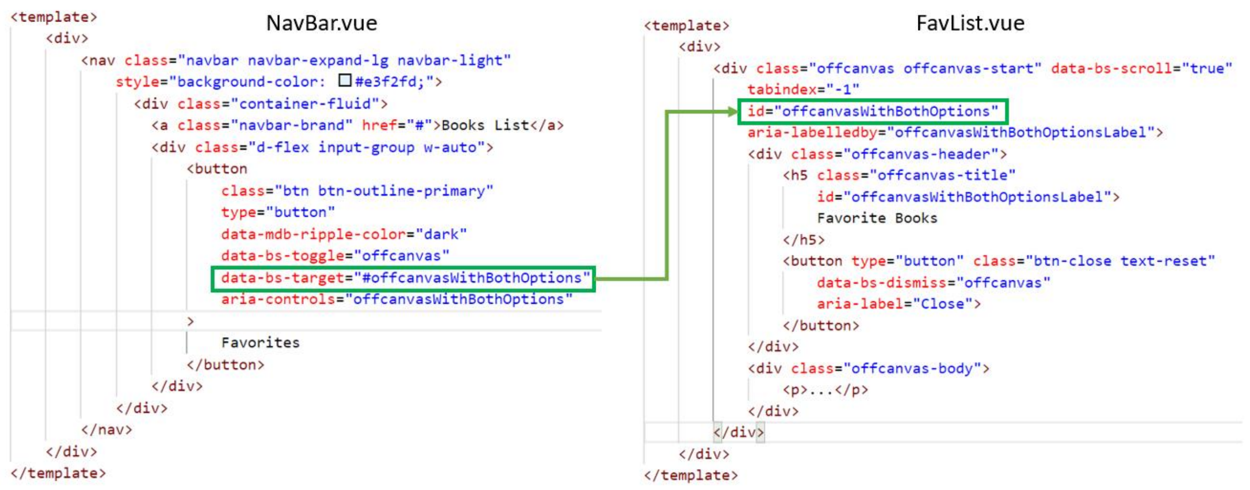
We define a **`nav`** component wrapped around the main **`div`** using regular Bootstrap CSS classes (highlighted in purple). This **`nav`** contains an anchor (**`a`**) tag (highlighted in yellow) and a Favorites **`button`** component (highlighted in green).

Like Vue.js, Nuxt.js HTML code must be embedded within the **`template`** tags. As you have seen, that wasn't difficult at all.

Now, here comes the exciting part. Notice that the **`button`** component includes the **`data-bs-target`** attribute. What does this attribute do? It’s related to another Bootstrap component called offcanvas, which is a hidden sidebar that we'll use to keep a list of favorite books.

In essence, the **`data-bs-target`** attribute is used to reference an **`offcanvas`** component that will appear when the **Favorites** button is clicked.

To keep our code clean, let's create the **`offcanvas`** functionality in a separate component that we'll name **`FavList`** (**FavList.vue**). The following diagram illustrates how both components are related using the **`offcanvasWithBothOptions`** attribute name.


NavBar.vue

```
<template>
  <div>
    <nav class="navbar navbar-expand-lg navbar-light"
      style="background-color: #e3f2fd;">
      <div class="container-fluid">
        <a class="navbar-brand" href="#">Books List</a>
        <div class="d-flex input-group w-auto">
          <button
            class="btn btn-outline-primary"
            type="button"
            data-mdb-ripple-color="dark"
            data-bs-toggle="offcanvas"
            data-bs-target="#offcanvasWithBothOptions"
            aria-controls="offcanvasWithBothOptions"
          >
            Favorites
          </button>
        </div>
      </div>
    </nav>
  </div>
</template>
```

FavList.vue

```
<template>
  <div>
    <div class="offcanvas offcanvas-start" data-bs-scroll="true"
      tabindex="-1"
      id="offcanvasWithBothOptions"
      aria-labelledby="offcanvasWithBothOptionsLabel">
      <div class="offcanvas-header">
        <h5 class="offcanvas-title"
          id="offcanvasWithBothOptionsLabel">
          Favorite Books
        </h5>
        <button type="button" class="btn-close text-reset"
          data-bs-dismiss="offcanvas"
          aria-label="Close">
        </button>
      </div>
      <div class="offcanvas-body">
        <p>...</p>
      </div>
    </div>
  </div>
</template>
```

Figure 3-g: NavBar—FavList Relationship (Using the offcanvasWithBothOptions Attribute Name)

FavList component

Now that we have the **navbar** component ready, we need to create a new FavList.vue file. We can do this by selecting the **components** folder and clicking the **New File** icon within the VS Code EXPLORER panel.

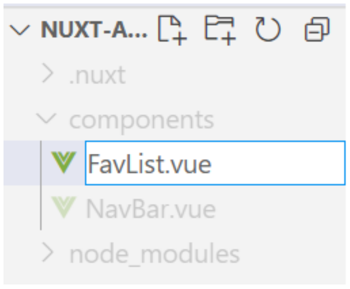


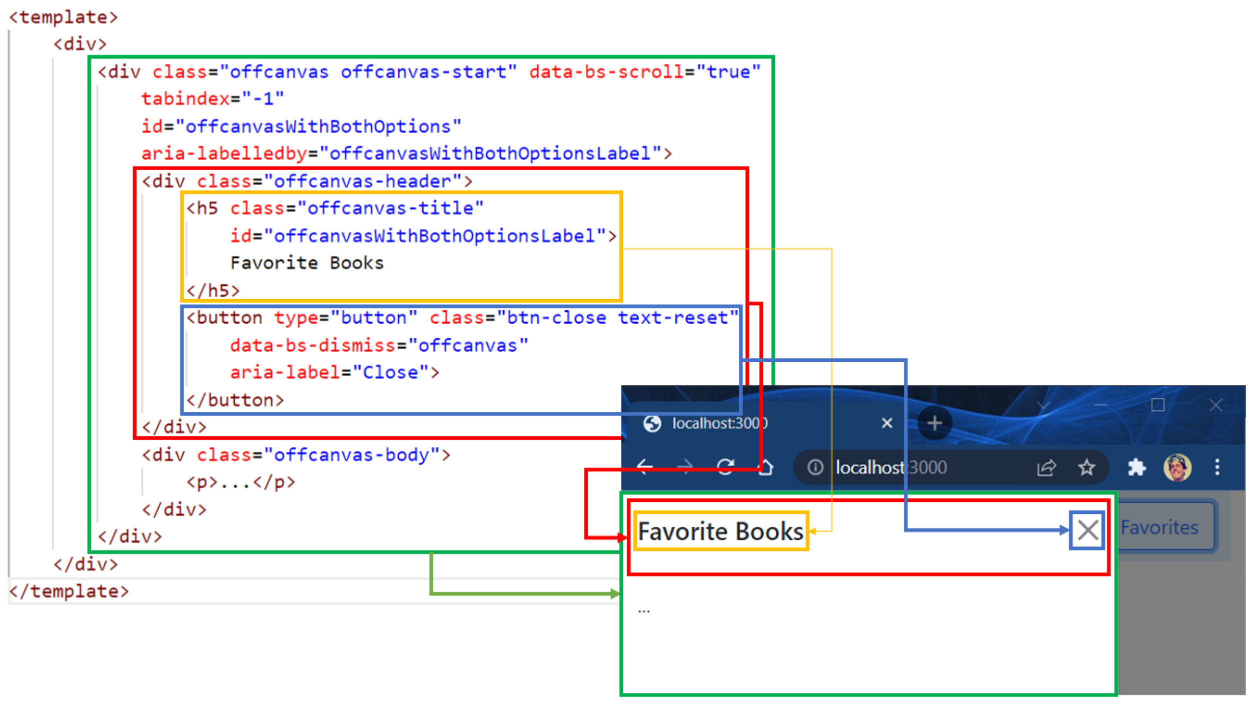
Figure 3-h: Creating FavList.vue (VS Code—EXPLORER)

After you've created the file, click to open it in VS Code, and paste in the following code.

Code Listing 3-f: The FavList Component (FavList.vue)

```
<template>
    <div>
        <div class="offcanvas offcanvas-start" data-bs-scroll="true"
            tabindex="-1"
            id="offcanvasWithBothOptions"
            aria-labelledby="offcanvasWithBothOptionsLabel">
            <div class="offcanvas-header">
                <h5 class="offcanvas-title"
                    id="offcanvasWithBothOptionsLabel">
                    Favorite Books
                </h5>
                <button type="button" class="btn-close text-reset"
                    data-bs-dismiss="offcanvas"
                    aria-label="Close">
                </button>
            </div>
            <div class="offcanvas-body">
                <p>...</p>
            </div>
        </div>
    </div>
</template>
```

To understand this code and how it relates to the finished **`offcanvas`** component functionality, let's look at the following diagram.



```
<template>
    <div>
        <div class="offcanvas offcanvas-start" data-bs-scroll="true"
            tabindex="-1"
            id="offcanvasWithBothOptions"
            aria-labelledby="offcanvasWithBothOptionsLabel">
            <div class="offcanvas-header">
                <h5 class="offcanvas-title"
                    id="offcanvasWithBothOptionsLabel">
                    Favorite Books
                </h5>
                <button type="button" class="btn-close text-reset"
                    data-bs-dismiss="offcanvas"
                    aria-label="Close">
                </button>
            </div>
            <div class="offcanvas-body">
                <p>...</p>
            </div>
        </div>
    </div>
</template>
```

Figure 3-i: FavList Code and UI Relationship

We see that we have the **`div`** with the **`id offcanvasWithBothOptions`** (highlighted in green) contained within the main **`div`**.

This **`div`** contains two child **`div`** tags—one represents the **`offcanvas-header div`** (highlighted in **red**), and the other represents the **`offcanvas-body div`** (for which we'll add the HTML code later).

The **offcanvas-header div** contains an **h5** tag (highlighted in yellow) with the title of the offcanvas and a **button** that closes the offcanvas when clicked (highlighted in blue).

As you have seen, that wasn't difficult to implement either.

Initial objects

Now that we have the application's main UI components partially ready, it's time to add some hardcoded objects to represent the book data our application will display. As you might have guessed, this book data will be about the *Succinctly* series books.

Let’s open the **index.vue** file contained within the pages folder using the EXPLORER panel in VS Code and add the following code.

Code Listing 3-g: The Initial Objects Data (index.vue)

```
<script>
    export default {
        data() {
            return {
                books: [
                    {
                        uuid: 'cde4664e-afb8-47b2-b4e0-ecedad4ebbf6',
                        name: 'Razor Components Succinctly',
                        description: 'Razor components are specific
                          building blocks within the Blazor framework.',
                        author: 'Ed Freitas',
                        url: 'https://www.syncfusion.com/succinctly-free-
                              ebooks/razor-components-succinctly',
                        picUrl:
                             'https://cdn.syncfusion.com/content/images/
                              downloads/ebook/ebook-cover/
                              Razor-Components-Succinctly.png'
                    },
                    {
                        uuid: 'e3d063c7-57bf-4df6-a626-137bfc658e04',
                        name: 'Azure Virtual Desktop Succinctly',
                        description: 'Put simply, Azure Virtual Desktop
                          is a way to serve Windows resources
                          over the internet.',
                        author: 'Marco Moioli',
                        url: 'https://www.syncfusion.com/succinctly-free-
                              ebooks/azure-virtual-desktop-succinctly',
                        picUrl: 'https://cdn.syncfusion.com/content/
                                 images/downloads/ebook/ebook-cover/
```

```
                                  azure-virtual-desktop-succinctly.png'
                    },
                    {
                        uuid: '86af8093-7ba9-4178-9c19-0eeb9e75bcf0',
                        name: 'Ansible Succinctly',
                        description: 'Ansible is an open-source software,
                          automation engine, and automation language.',
                        author: 'Zoran Maksimovic',
                        url: 'https://www.syncfusion.com/succinctly-free-
                              ebooks/ansible-succinctly',
                        picUrl: 'https://cdn.syncfusion.com/content/
                                 images/downloads/ebook/
                                 ebook-cover/ansible-succinctly.png'
                    }
                ]
            }
        }
    }
</script>
```

Note: I've written some of the strings in multiple lines per attribute (for readability); however, if you paste these in your** index.vue **file, do not split these into numerous lines—make sure each value remains in a single line.

The book data contains a **`books`** array returned by the **`data`** method. The **`uuid`** is a universally unique ID, sometimes called a GUID.

Books UI

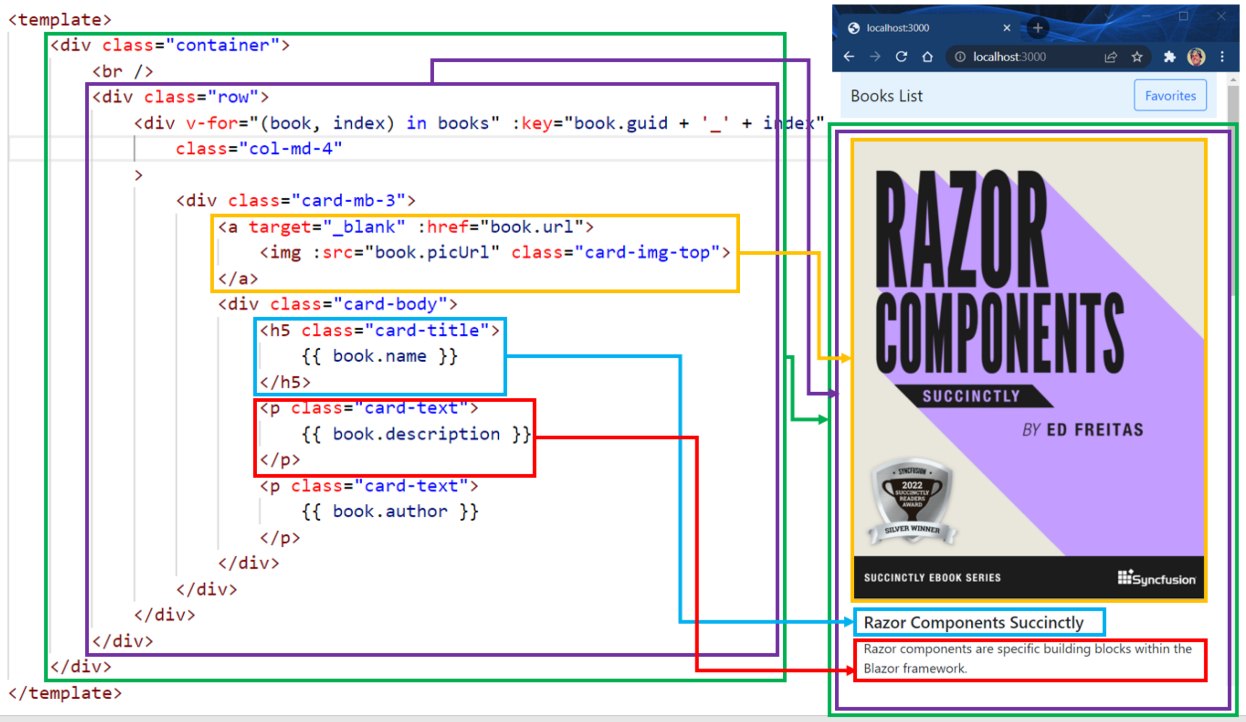
Now that we have the book information ready, let's create the UI to display the book data, as seen in the following listing.

Code Listing 3-h: Books UI (index.vue)

```
<template>
    <div class="container">
        <br />
        <div class="row">
           <div v-for="(book, index) in books"
                 :key="book.uuid + '_' + index"
                 class="col-md-4"
            >
                <div class="card-mb-3">
```

```
                        <a target="_blank" :href="book.url">
                            <img :src="book.picUrl" class="card-img-top">
                        </a>
                        <div class="card-body">
                            <h5 class="card-title">
                                {{ book.name }}
                            </h5>
                            <p class="card-text">
                                {{ book.description }}
                            </p>
                            <p class="card-text">
                                {{ book.author }}
                            </p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
</template>
```

To understand this better, let's look at the following diagram.



```
<template>
    <div class="container">
        <br />
        <div class="row">
            <div v-for="(book, index) in books" :key="book.guid + '_' + index"
                class="col-md-4"
            >
                <div class="card-mb-3">
                    <a target="_blank" :href="book.url">
                        <img :src="book.picUrl" class="card-img-top">
                    </a>
                    <div class="card-body">
                        <h5 class="card-title">
                            {{ book.name }}
                        </h5>
                        <p class="card-text">
                            {{ book.description }}
                        </p>
                        <p class="card-text">
                            {{ book.author }}
                        </p>
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>
```

Razor Components Succinctly

Razor components are specific building blocks within the Blazor framework.

Figure 3-j: Books Code and UI Relationship

We can see that we have the main **div** using Bootstrap's **container** CSS class (highlighted in green). This container **div** contains a **row** (highlighted in purple), and within it, we find another **div** used to display a list of **books**.

The list of **books** is displayed by iterating over them using the **v-for** directive from Vue.js. Notice that each book rendered using the **v-for** directive requires a unique **key**: the concatenation of the **book.uuid** and the **index** value.

Then, we have an anchor (**a**) tag (highlighted in yellow) that points to the book's URL (**book.url**). It contains the book's cover image that uses the **book.picUrl** as its source (**src**).

When clicking on the book cover's image, we are directed to the book's website on a separate browser tab/window—which is why the anchor's **target** is set to **_blank**.

Following that, we find an **h5** tag that displays the **book.name** (highlighted in green), a **p** tag that shows the **book.description** (highlighted in red), and finally, another **p** tag that displays the **book.author**.

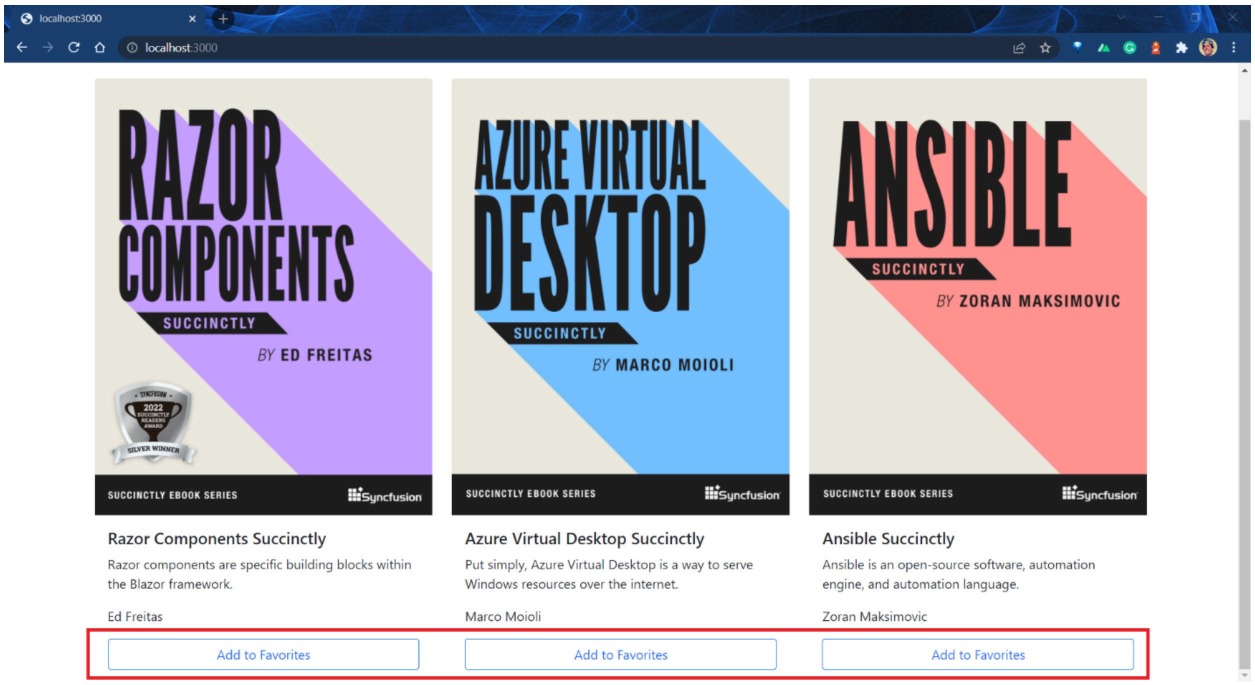
Favorites button

Now that we have most of the book UI ready, I'd like to add an Add to Favorites button for each book displayed, which will allow a book to be added to the **offcanvas** component that will contain the list of favorite books. The new code is highlighted in **bold**.

Code Listing 3-i: Updated Books UI (index.vue)

```
<template>
    <div class="container">
        <br />
        <div class="row">
            <div v-for="(book, index) in books"
                 :key="book.uuid + '_' + index"
                 class="col-md-4"
            >
                <div class="card-mb-3">
                    <a target="_blank" :href="book.url">
                        <img :src="book.picUrl" class="card-img-top">
                    </a>
                    <div class="card-body">
                        <h5 class="card-title">
                            {{ book.name }}
                        </h5>
                        <p class="card-text">
                            {{ book.description }}
                        </p>
                        <p class="card-text">
                            {{ book.author }}
                        </p>
                        <div class="d-grid">
                            <button class="btn btn-outline-primary">
                                Add to Favorites
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>
```

The updated UI would look as follows—with the Add to Favorites button appearing after each author name.


localhost:3000
localhost:3000
RAZOR COMPONENTS SUCCINCTLY BY ED FREITAS
AZURE VIRTUAL DESKTOP SUCCINCTLY BY MARCO MOIOLI
ANSIBLE SUCCINCTLY BY ZORAN MAKSIMOVIC
SUCCINCTLY EBOOK SERIES
Syncfusion

Razor Components Succinctly
Razor components are specific building blocks within the Blazor framework.
Ed Freitas
Add to Favorites

Azure Virtual Desktop Succinctly
Put simply, Azure Virtual Desktop is a way to serve Windows resources over the internet.
Marco Moioli
Add to Favorites

Ansible Succinctly
Ansible is an open-source software, automation engine, and automation language.
Zoran Maksimovic
Add to Favorites

Figure 3-k: Finished Books UI (Main Screen)

You'll be able to verify this shortly in your browser, but first, let's wrap up the remaining changes for index.vue.

Code Listing 3-j: Books UI (index.vue)—Full Code

```
<template>
    <div class="container">
```

```
        <br />
        <div class="row">
            <div v-for="(book, index) in books"
                :key="book.uuid + '_' + index"
                class="col-md-4"
            >
                <div class="card-mb-3">
                    <a target="_blank" :href="book.url">
                        <img :src="book.picUrl" class="card-img-top">
                    </a>
                    <div class="card-body">
                        <h5 class="card-title">
                            {{ book.name }}
                        </h5>
                        <p class="card-text">
                            {{ book.description }}
                        </p>
                        <p class="card-text">
                            {{ book.author }}
                        </p>
                        <div class="d-grid">
                            <button class="btn btn-outline-primary">
                                Add to Favorites
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>

<style scoped>
    .img {
        width: 80%;
        height: auto;
    }

    .center {
        display: block;
        margin-left: auto;
        margin-right: auto;
        width: 100%;
    }
```

```
</style>

<script>
    export default {
        data() {
            return {
                books: [
                    {
                        uuid: 'cde4664e-afb8-47b2-b4e0-ecedad4ebbf6',
                        name: 'Razor Components Succinctly',
                        description: 'Razor components are specific
                          building blocks within the Blazor framework.',
                        author: 'Ed Freitas',
                        url: 'https://www.syncfusion.com/succinctly-free-
                              ebooks/razor-components-succinctly',
                        picUrl: 'https://cdn.syncfusion.com/content/
                                 images/downloads/ebook/ebook-cover/
                                 Razor-Components-Succinctly.png'
                    },
                    {
                        uuid: 'e3d063c7-57bf-4df6-a626-137bfc658e04',
                        name: 'Azure Virtual Desktop Succinctly',
                        description: 'Put simply, Azure Virtual Desktop
                                      is a way to serve Windows
                                      resources over the internet.',
                        author: 'Marco Moioli',
                        url: 'https://www.syncfusion.com/succinctly-free-
                              ebooks/azure-virtual-desktop-succinctly',
                        picUrl:
                             'https://cdn.syncfusion.com/content/images/
                              downloads/ebook/ebook-cover/
                              azure-virtual-desktop-succinctly.png'
                    },
                    {
                        uuid: '86af8093-7ba9-4178-9c19-0eeb9e75bcf0',
                        name: 'Ansible Succinctly',
                        description: 'Ansible is an open-source software,
                          automation engine, and automation language.',
                        author: 'Zoran Maksimovic',
                        url: 'https://www.syncfusion.com/succinctly-free-
                              ebooks/ansible-succinctly',
                        picUrl: 'https://cdn.syncfusion.com/content/
                                 images/downloads/ebook/
                                 ebook-cover/ansible-succinctly.png'
                    }
                ]
            }
        }
    }
</script>
```

Note: I've written some of the strings in multiple lines per attribute (for readability); however, if you paste these in your *index.vue* ***file, do not split these into numerous lines—make sure each value remains in a single line.***

Notice that I've also included some scoped CSS styles (highlighted in **bold**). Even though we are not using these CSS classes, I've included them if you want to style the HTML of index.vue later.

This will open the browser and allow you to view the changes implemented. (If you're not running the application, execute the **`npm run dev`** command from the built-in VS Code terminal to run the app.)

Summary

In this chapter, we have laid out the foundation of our app, implemented the application's main UI, and enabled the usage of Bootstrap's **`offcanvas`** component—which we will use to keep our favorite books.

We still have to add all the necessary logic for the application to work and finish the UI of the favorites list—which only makes sense to complete once we have the app's logic laid out. That's what we are going to look at next.

Chapter 4 Client Logic

Overview

With most of the application's UI done, it's now time to focus on the application's client-side logic. I'll mainly use the Options API from Vue.js rather than the Composition API (new with Vue.js 3), which uses the **`setup`** method.

The reason is that I don't want to introduce too many changes for developers coming from Vue.js version 2, so we can focus on Nuxt.js rather than also having to focus on different Vue.js 3 changes.

Add to favorites method

The first step in adding logic to our application is to add a method that allows us to add a book to the list of favorite books. The following listing shows the method that achieves that, called **`addToFavs`**.

Code Listing 4-a: Methods (index.vue)

```
<script>
    export default {
        data() {
            return {
                favorites: [],
                books: [
                ...
                ]
            }
        },
        methods: {
            addToFavs(book) {
                let inFavs = false;

                for (const fav of this.favorites) {
                    if (fav.uuid === book.uuid) {
                        inFavs = true;
                        break;
                    }
                }

                if (!inFavs) {
                    this.favorites.push({...book});
                }
            }
        }
    }
</script>
```

For the sake of readability, I've explicitly excluded the book data (which is why you'll notice this instead `...`). However, that doesn't mean that this data has been removed from the code.

Let's now explore what the **`addToFavs`** method does. The **`addToFavs`** method begins by declaring an **`inFavs`** variable initialized to **`false`**.

We assume that no books have been added to the list of **`favorites`** at this stage. Notice that the list of favorite books is initialized as an empty array within the **`data`** method (**`favorites: []`**).

Following that, we loop through the **`favorites`** array (using a **`for`** loop) and check if the current item (**`fav`**) on the **`favorites`** list has an **`uuid`** equal to the **`uuid`** of the **`book`** being inserted into the **`favorites`** list.

If the **`uuid`** of the current item has the same value as the **`uuid`** of the book being inserted, then **`inFavs`** is set to **`true`**, and the loop finalizes.

If the **`book`** being inserted does not exist on the list of **`favorites`**, then it is added to the list with this instruction: **`this.favorites.push({...book})`**.

Note: We are using the JavaScript [spread](#) syntax (*…book****) when calling the*** *push* ***method of the*** *favorites* ***array, as we want to add all the properties of the*** *book* ***object as one element of the array.***

By doing this, we ensure that the same book is not added more than once to the **`favorites`** list.

Click event binding

Now that we have the **`addToFavs`** method ready, let's enable it by binding it to a **`click`** event, as seen in bold in the following code.

Code Listing 4-b: Binding the addToFavs Methods (index.vue)

```
<button @click="addToFavs(book)" class="btn btn-outline-primary">
    Add to Favorites
</button>
```

This works because we use the **`v-on`** directive (shorthanded as **`@`**) to bind the **`addToFavs`** method to the **`click`** event. To the **`addToFavs`** method, we pass the **`book`** that we want to insert to the **`favorites`** list. Here is the updated code for index.vue.

Code Listing 4-c: Updated index.vue

```
<template>
    <div class="container">
        <br />
        <div class="row">
            <div v-for="(book, index) in books"
                :key="book.uuid + '_' + index"
                class="col-md-4"
            >
                <div class="card-mb-3">
                    <a target="_blank" :href="book.url">
                        <img :src="book.picUrl" class="card-img-top">
                    </a>
                    <div class="card-body">
                        <h5 class="card-title">
                            {{ book.name }}
                        </h5>
                        <p class="card-text">
                            {{ book.description }}
                        </p>
                        <p class="card-text">
                            {{ book.author }}
                        </p>
                        <div class="d-grid">
                            <button @click="addToFavs(book)"
                                class="btn btn-outline-primary">
                                Add to Favorites
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>

<style scoped>
    .img {
        width: 80%;
        height: auto;
    }

    .center {
        display: block;
        margin-left: auto;
        margin-right: auto;
        width: 100%;
    }
</style>

<script>
    export default {
        data() {
            return {
                favorites: [],
                books: [
                    {
                        uuid: 'cde4664e-afb8-47b2-b4e0-ecedad4ebbf6',
                        name: 'Razor Components Succinctly',
                        description: 'Razor components are specific
                          building blocks within the Blazor framework.',
                        author: 'Ed Freitas',
                        url: 'https://www.syncfusion.com/succinctly-free-
                          ebooks/razor-components-succinctly',
                        picUrl:
                          'https://cdn.syncfusion.com/content/images/
                           downloads/ebook/ebook-cover/
                           Razor-Components-Succinctly.png'
                    },
                    {
                        uuid: 'e3d063c7-57bf-4df6-a626-137bfc658e04',
                        name: 'Azure Virtual Desktop Succinctly',
                        description: 'Put simply, Azure Virtual Desktop
                                      is a way to serve Windows
                                      resources over the internet.',
                        author: 'Marco Moioli',
                        url: 'https://www.syncfusion.com/succinctly-free-
                          ebooks/azure-virtual-desktop-succinctly',
                        picUrl:
                          'https://cdn.syncfusion.com/content/images/
                           downloads/ebook/ebook-cover/
                           azure-virtual-desktop-succinctly.png'
                    },
                    {
                        uuid: '86af8093-7ba9-4178-9c19-0eeb9e75bcf0',
                        name: 'Ansible Succinctly',
                        description: 'Ansible is an open-source software,
                          automation engine, and automation language.',
                        author: 'Zoran Maksimovic',
                        url: 'https://www.syncfusion.com/succinctly-free-
                          ebooks/ansible-succinctly',
                        picUrl:
                          'https://cdn.syncfusion.com/content/images/
                           downloads/ebook/ebook-cover/
                           ansible-succinctly.png'
                    }
                ]
            }
        },
        methods: {
            addToFavs(book) {
                let inFavs = false;

                for (const fav of this.favorites) {
                    if (fav.uuid === book.uuid) {
                        inFavs = true;
                        break;
                    }
                }

                if (!inFavs) {
                    this.favorites.push({...book});
                }
            }
        }
    }
</script>
```

Note: I've written some of the strings in multiple lines per attribute (for readability); however, if you paste these in your index.vue file, do not split these into numerous lines—make sure each value remains in a single line.

Now we're ready to work on the FavList.vue file.

Moving the Favorites list

As you have noticed, the **`favorites`** array has been declared within the **`data`** method of the index.vue file (as seen previously).

Although this has worked so far, it poses a problem to us going forward. We need to have the data contained within the **`favorites`** array available within the FavList.vue file.

We could pass the **`favorites`** array to the FavList.vue file by calling the component **`FavList`** and passing the **`favorites`** array as a property: **`<FavList favs="favorites"/>`**.

This could work, but the problem with this approach is that besides repeating the data, we would have one copy within index.vue and another copy in FavList.vue—which we would also need to keep in sync.

A better way would be to move the **`favorites`** array to FavList.vue and then reference the **`favorites`** array from index.vue. By doing this, we keep one copy of the data.

We can do this by changing the name of the **`favorites`** array to **`favlist`** (to be declared in FavList.vue) and then referencing it within index.vue as **`<FavList ref="favlist"/>`**.

Refactoring index.vue

To begin, let's refactor index.vue (which no longer uses the **`favorites`** array) as follows. The newest changes are emphasized in **bold**.

Code Listing 4-d: Refactored index.vue

```
<template>
    <div class="container">
        <FavList ref="favlist"/>
        <br />
        <div class="row">
            <div v-for="(book, index) in books"
                 :key="book.uuid + '_' + index"
                class="col-md-4"
            >
                <div class="card-mb-3">
                    <a target="_blank" :href="book.url">
                        <img :src="book.picUrl" class="card-img-top">
                    </a>
                    <div class="card-body">
                        <h5 class="card-title">
                            {{ book.name }}
                        </h5>
                        <p class="card-text">
```

```
                            {{ book.description }}
                        </p>
                        <p class="card-text">
                            {{ book.author }}
                        </p>
                        <div class="d-grid">
                            <button @click="addToFavs(book)"
                                class="btn btn-outline-primary">
                                Add to Favorites
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>

<style scoped>
    .img {
        width: 80%;
        height: auto;
    }

    .center {
        display: block;
        margin-left: auto;
        margin-right: auto;
        width: 100%;
    }
</style>

<script>
    import FavList from '../components/FavList';

    export default {
        components: { FavList },
        data() {
            return {
                books: [
                    {
                        uuid: 'cde4664e-afb8-47b2-b4e0-ecedad4ebbf6',
                        name: 'Razor Components Succinctly',
                        description: 'Razor components are specific
```

```
                          building blocks within the Blazor framework.',
                        author: 'Ed Freitas',
                        url: 'https://www.syncfusion.com/succinctly-free-
                            ebooks/razor-components-succinctly',
                        picUrl:
                            'https://cdn.syncfusion.com/content/images/
                            downloads/ebook/ebook-cover/
                            Razor-Components-Succinctly.png'
                    },
                    {
                        uuid: 'e3d063c7-57bf-4df6-a626-137bfc658e04',
                        name: 'Azure Virtual Desktop Succinctly',
                        description: 'Put simply, Azure Virtual Desktop
                                      is a way to serve Windows
                                      resources over the internet.',
                        author: 'Marco Moioli',
                        url: 'https://www.syncfusion.com/succinctly-free-
                            ebooks/azure-virtual-desktop-succinctly',
                        picUrl: 'https://cdn.syncfusion.com/content/
                                 images/downloads/ebook/ebook-cover/
                                 azure-virtual-desktop-succinctly.png'
                    },
                    {
                        uuid: '86af8093-7ba9-4178-9c19-0eeb9e75bcf0',
                        name: 'Ansible Succinctly',
                        description: 'Ansible is an open-source software,
                          automation engine, and automation language.',
                        author: 'Zoran Maksimovic',
                        url: 'https://www.syncfusion.com/succinctly-free-
                            ebooks/ansible-succinctly',
                        picUrl: 'https://cdn.syncfusion.com/content/
                                 images/downloads/ebook/
                                 ebook-cover/ansible-succinctly.png'
                    }
                ]
            }
        },
        methods: {
            addToFavs(book) {
                let inFavs = false;
                const favs = this.$refs.favlist;

                for (const fav of favs.favorites) {
                    if (fav.uuid === book.uuid) {
                        inFavs = true;
                        break;
                    }
                }

                if (!inFavs) {
                    favs.favorites.push({...book});
                }
            }
        }
    }
</script>
```

Note: I've written some of the strings in multiple lines per attribute (for readability); however, if you paste these in your index.vue file, do not split these into numerous lines—make sure each value remains in a single line.

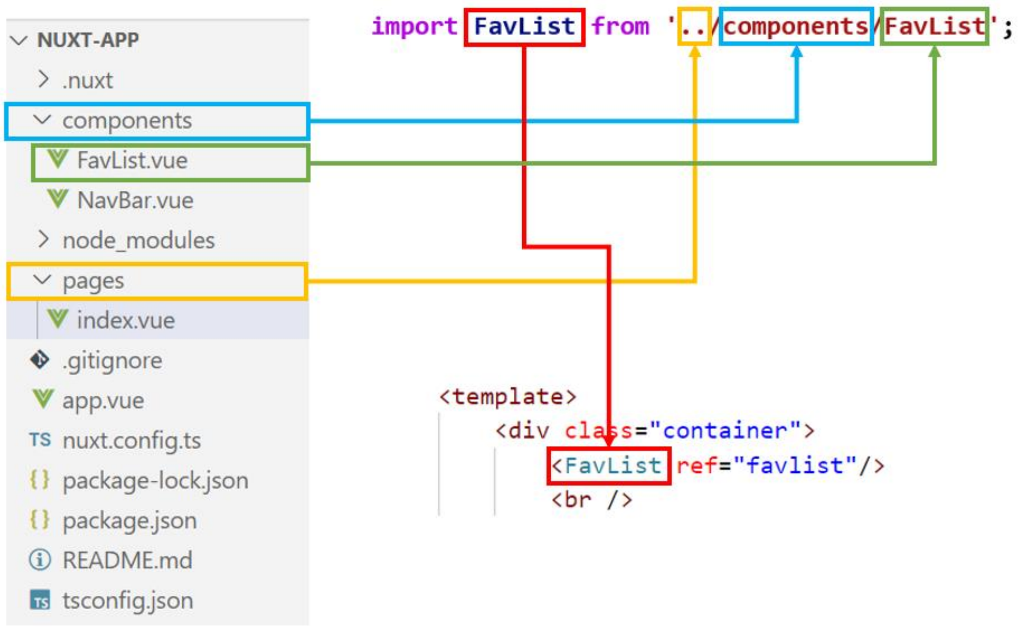
Let's go over the changes. First, we've included (within the HTML markup) a reference to the **`FavList`** component as follows **`<FavList ref="favlist"/>`**.

Notice that **`favlist`** (which is the original **`favorites`** array renamed) is being passed as reference (**`ref`**). This means that although it is part of FavList.vue (as we will see shortly), we can access and manipulate it from index.vue.

Next, we can see that the **`FavList`** component is being imported within the **`script`** section as follows.

```
import FavList from '../components/FavList';
```

To understand how the **`import`** statement works, let's look at the following diagram.



```
import FavList from '../components/FavList';
```

```
<template>
    <div class="container">
        <FavList ref="favlist"/>
        <br />
```

Figure 4-a: Relationship between Import Statement and FavList Component

The preceding diagram shows how the **`FavList`** relates to the **`FavList`** component referenced in the HTML markup (highlighted in red).

To get to the file path where the component is located, we need to come out of the pages folder, go one level up (**`..`**)—highlighted in yellow—and then go into the components folder (highlighted in blue).

The **`FavList`** component is contained within **`FavList`** (which is the FavList.vue file), highlighted in green. Notice how the .vue extension is not required when importing a component.

Next, we can see that we have included **`FavList`** within the **`components`** property of index.vue, as follows: **`components: { FavList }`**.

Following that, within the **`addToFavs`** method, we can access **`favlist`** array from the **`FavList`** component by using Vue.js template refs, as follows.

```
const favs = this.$refs.favlist;
```

Then, the favorites array (**`favlist`**) is accessed as **`favs.favorites`**.

FavList.vue

With index.vue refactored, let's open FavList.vue and copy and paste the following code.

Code Listing 4-e: FavList.vue

```
<template>
    <div>
        <div class="offcanvas offcanvas-start" data-bs-scroll="true"
            tabindex="-1"
            id="offcanvasWithBothOptions"
            aria-labelledby="offcanvasWithBothOptionsLabel">
            <div class="offcanvas-header">
                <h5 class="offcanvas-title"
                    id="offcanvasWithBothOptionsLabel">
                    Favorite Books
                </h5>
                <button type="button" class="btn-close text-reset"
                    data-bs-dismiss="offcanvas"
                    aria-label="Close">
                </button>
            </div>
            <div class="offcanvas-body">
                <div v-for="(fav, index) in favorites"
                    :key="fav.uuid + '_' + index"
                    class="card-mb-3">
                    <div class="row">
                        <div class="col-md-4">
                            <img :src="fav.picUrl"
                                class="img-fluid rounded-start">
                        </div>
                        <div class="col-md-8">
                            <div class="card-body">
                                <h5 class="card-title">
                                    {{ fav.name }}
                                </h5>
                                <p class="card-title">
                                    {{ fav.author }}
                                </p>
                                <div class="d-grid">
                                    <button @click="delFav(fav)"
                                      class="btn btn-outline-secondary">
                                        Remove from list
                                    </button>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>

<script>
    export default {
        name: 'FavList',
        data() {
            return {
                favorites: []
            }
        },
        methods: {
            delFav(fav) {
                const favs = this.favorites;
                const idx = favs.findIndex(
                                 item => item.uuid === fav.uuid);
                if (idx >= 0) {
                    favs.splice(idx, 1);
                }
            }
        }
    }
</script>
```

Within the **`template`** section, we can see that most of the HTML code is used for rendering the **`offcanvas`** Bootstrap component. There are two sections: the header (**`offcanvas-header`**) and body (**`offcanvas-body`**).

The header section contains the title used by the **`offcanvas`** component and a button to close it. On the other hand, the body section includes a picture of the book (**`fav.picUrl`**), the book's title (**`fav.name`**), the book's author (**`fav.author`**), and a button (**`button @click="delFav(fav)`**) to remove the book from the favorites list (**`favorites`**).

The body section repeats itself for all the books that have been added to the favorites list (**`favorites`**)—by executing this instruction: **`v-for="(fav, index) in favorites"`**.

Regarding the **`script`** part of the preceding code, we can see that the **`data`** method returns the **`favorites`** array—previously declared within index.vue.

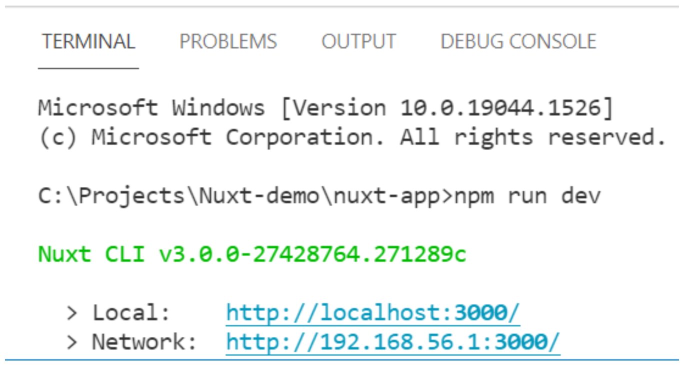
The **`delFav`** method, which receives the current or selected favorite book as a parameter (**`fav`**), removes that book from the list of favorite books by executing **`favs.splice(idx, 1)`**, which means remove **`1`** item at position **`idx`**.

To retrieve the current or selected favorite book, we use the **`favs.findIndex`** method. The chosen book is determined by comparing the value of **`uuid`** the current item to the value of the book (**`fav`**) passed as a parameter to the **`delFav`** method.

Executing the app

Excellent—we've got a working app at this stage. It's a small application and doesn't do that much, but it does achieve the goal that we set out to accomplish: to display a list of books and add and remove favorites from that list.

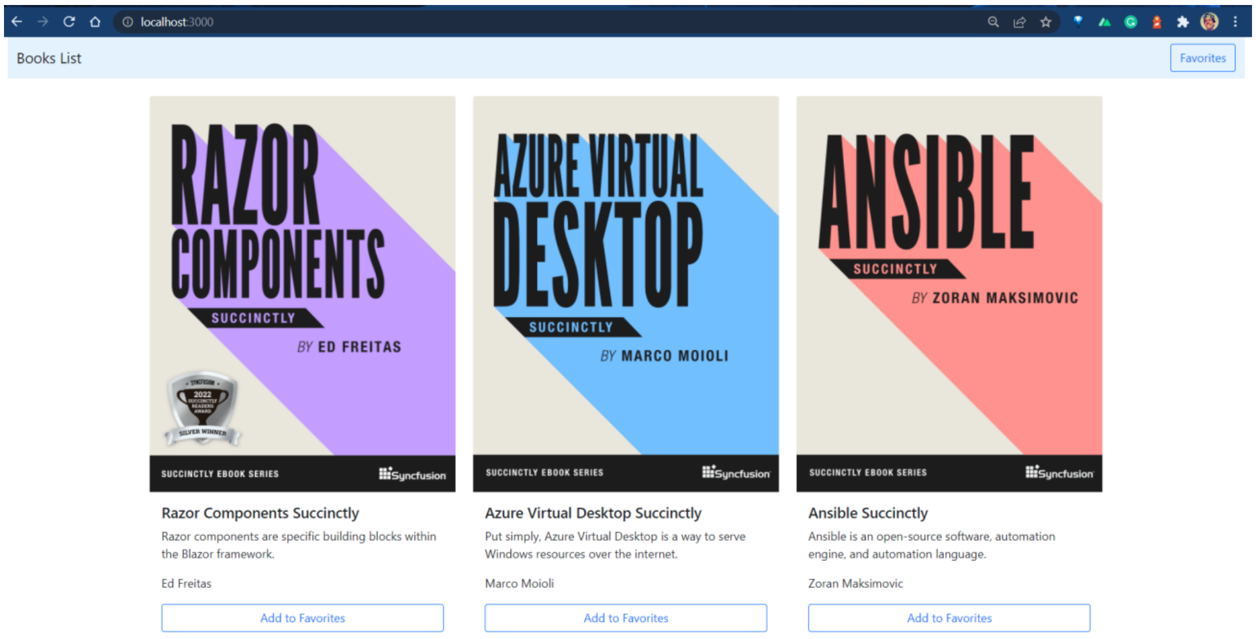
To see the application running, we can execute it. To do that, run the **`npm run dev`** command from the built-in terminal in VS Code.



```
Microsoft Windows [Version 10.0.19044.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Projects\Nuxt-demo\nuxt-app>npm run dev

Nuxt CLI v3.0.0-27428764.271289c

  > Local:    http://localhost:3000/
  > Network:  http://192.168.56.1:3000/
```

Figure 4-b: Executing the App (Built-in Terminal—VS Code)

If we open the browser and navigate to the application's URL, we should see the following.



Razor Components Succinctly
Razor components are specific building blocks within the Blazor framework.
Ed Freitas

Azure Virtual Desktop Succinctly
Put simply, Azure Virtual Desktop is a way to serve Windows resources over the internet.
Marco Moioli

Ansible Succinctly
Ansible is an open-source software, automation engine, and automation language.
Zoran Maksimovic

Figure 4-c: Application Running (1)

If we click each of the **Add to Favorites** buttons (below each of the book images) from left to right, and then click the **Favorites** button, we should see the following list of favorite books.

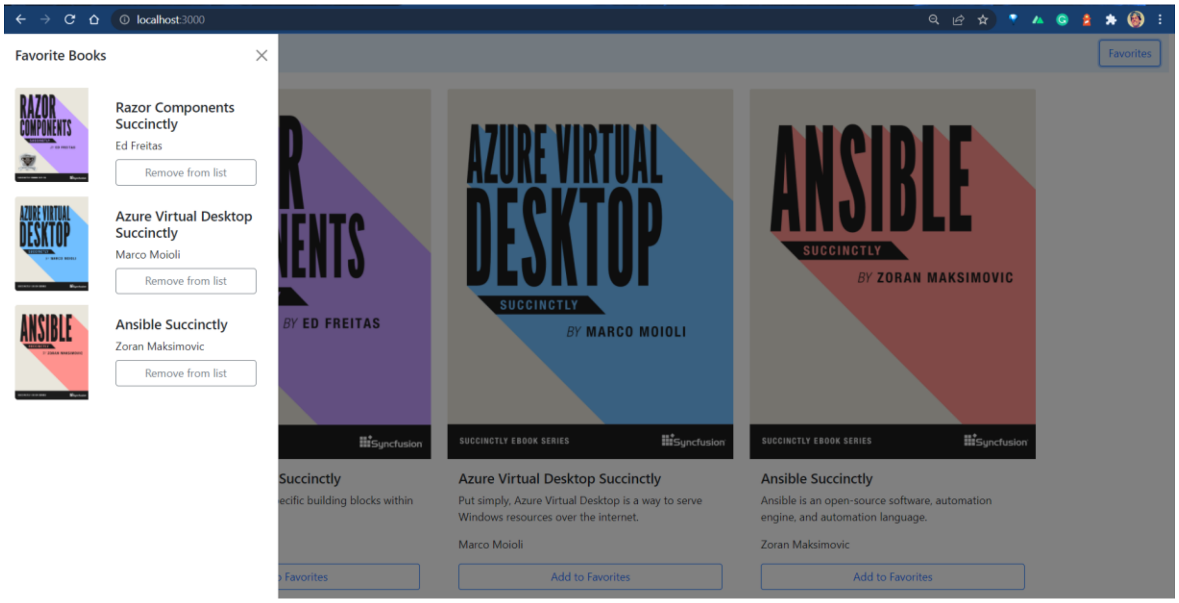

localhost:3000
Favorites
Favorite Books
Razor Components Succinctly
Ed Freitas
Remove from list
Azure Virtual Desktop Succinctly
Marco Moioli
Remove from list
Ansible Succinctly
Zoran Maksimovic
Remove from list
Azure Virtual Desktop Succinctly
Put simply, Azure Virtual Desktop is a way to serve Windows resources over the internet.
Marco Moioli
Add to Favorites
Ansible Succinctly
Ansible is an open-source software, automation engine, and automation language.
Zoran Maksimovic
Add to Favorites

Figure 4-d: Application Running (2)

Looking at the list of favorite books, we can see that the books have been added to the list in the same order as each of the Add to Favorites buttons were clicked.

If we click the **Remove from list** button for the middle book (*Azure Virtual Desktop Succinctly*), we should see the following.

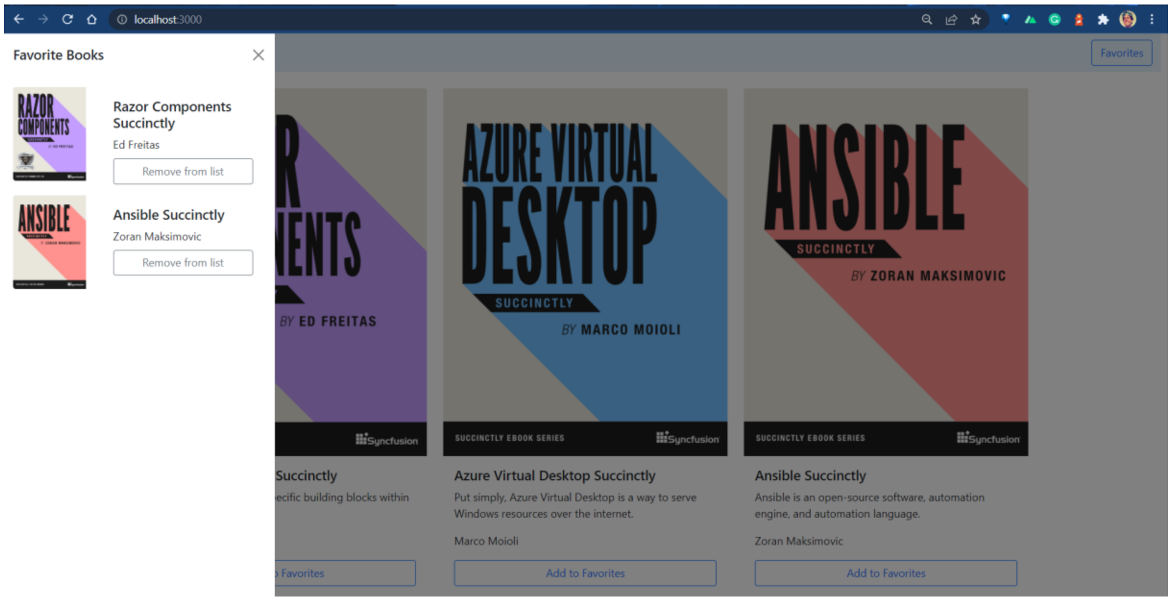

localhost:3000
Favorites
Favorite Books
Razor Components Succinctly
Ed Freitas
Remove from list
Ansible Succinctly
Zoran Maksimovic
Remove from list
Azure Virtual Desktop Succinctly
Put simply, Azure Virtual Desktop is a way to serve Windows resources over the internet.
Marco Moioli
Add to Favorites
Ansible Succinctly
Ansible is an open-source software, automation engine, and automation language.
Zoran Maksimovic
Add to Favorites

Figure 4-e: Application Running (3)

As we can see, the *Azure Virtual Desktop Succinctly* book has been successfully removed from the list of favorite books.

Summary

Congrats on what you've achieved so far! We have created all the application's client front-end logic. The next step is to work on the server-side logic and persist the data—which is what the next chapter is all about.

Chapter 5 Firebase

Overview

We are ready to start with the server-side setup of our application. Although we could write server-side code with previous versions of Nuxt.js, with version 3, we can now write endpoints just like writing an API using Node.js, but more straightforward.

To achieve data persistence and server-side functionality, we will use Firebase, a platform supported by Google for creating and monitoring mobile and web applications.

Getting started with Firebase

Getting started with Firebase is very easy. You need to have a Google Workspace or Gmail account to use it.

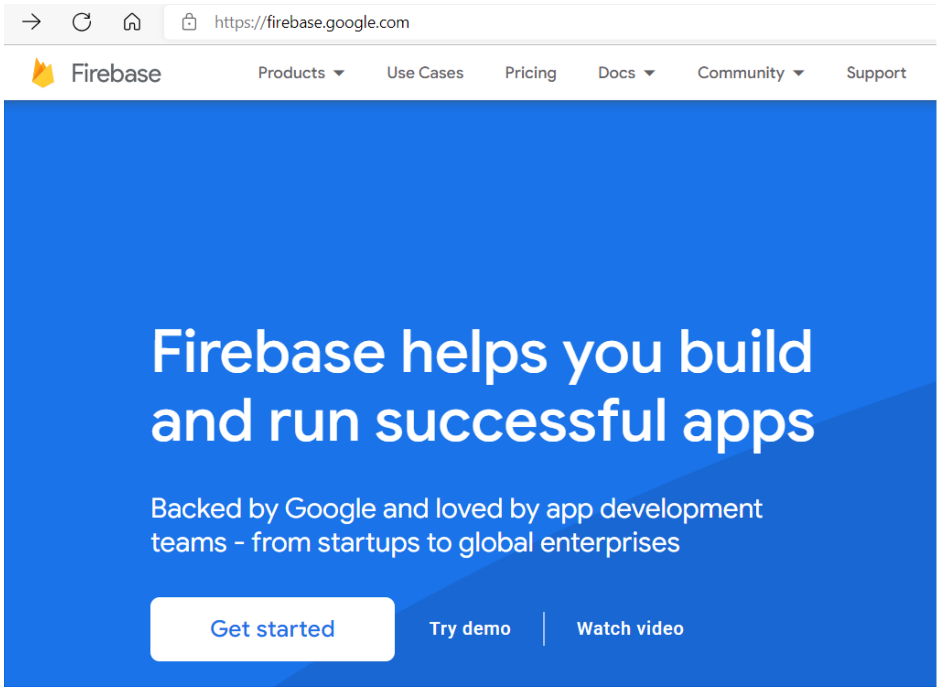

https://firebase.google.com
Firebase
Products
Use Cases
Pricing
Docs
Community
Support
Firebase helps you build and run successful apps
Backed by Google and loved by app development teams - from startups to global enterprises
Get started
Try demo
Watch video

Figure 5-a: Firebase Home Page

Note: The Firebase home page and website might change over time, but you should still easily be able to continue with the steps provided.

Once you're on the Firebase home page, click **Get started** > **Continue**.

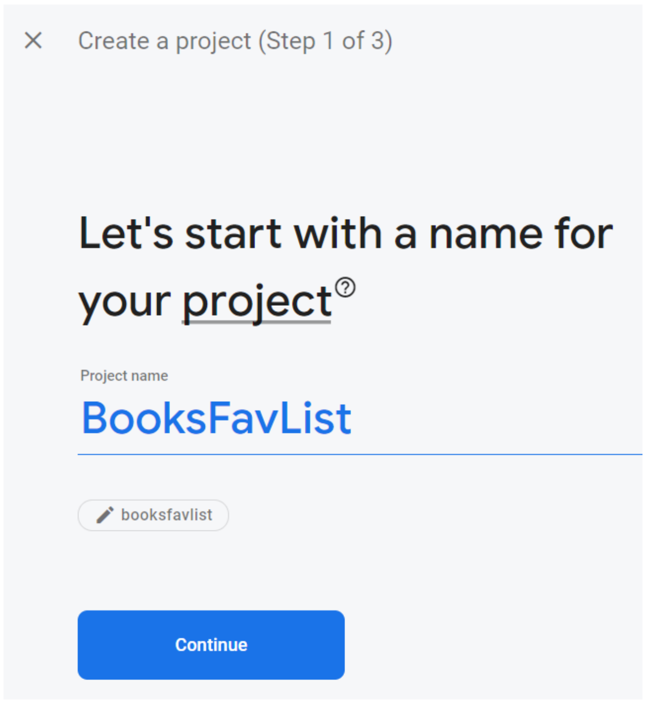
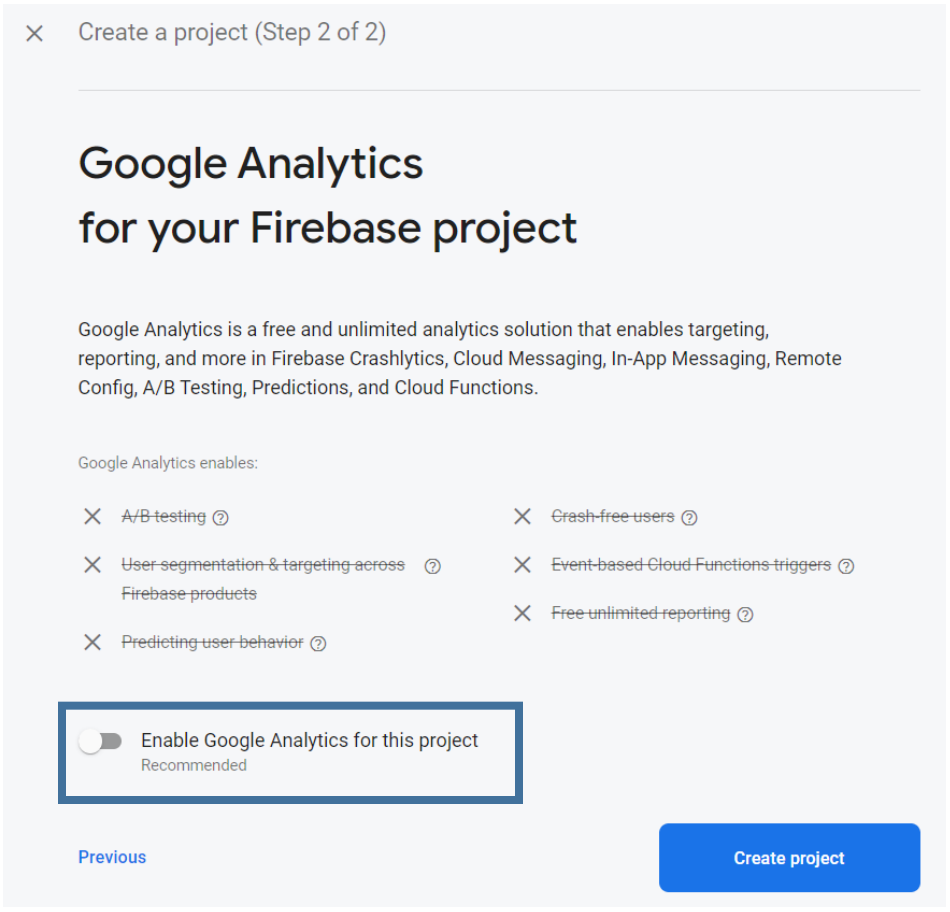

Create a project (Step 1 of 3)
Let's start with a name for your project
Project name
BooksFavList
booksfavlist
Continue

Figure 5-b: Creating a Firebase Project (Step 1 of 3)

Next, we'll see the following screen for setting up Google Analytics.


Create a project (Step 2 of 2)

Google Analytics
for your Firebase project

Google Analytics is a free and unlimited analytics solution that enables targeting, reporting, and more in Firebase Crashlytics, Cloud Messaging, In-App Messaging, Remote Config, A/B Testing, Predictions, and Cloud Functions.

Google Analytics enables:

~~A/B testing~~
~~User segmentation & targeting across Firebase products~~
~~Predicting user behavior~~
~~Crash-free users~~
~~Event-based Cloud Functions triggers~~
~~Free unlimited reporting~~

Enable Google Analytics for this project
Recommended

Figure 5-c: Creating a Firebase Project (Step 2 of 3)

At this stage, I would suggest disabling the option Enable Google Analytics for this project, which I'm going to do. We don't need to use Google Analytics for our purposes; however, it's an option to consider for larger projects.

Next, click **Create project** to initiate the creation of the project and provision the necessary resources.

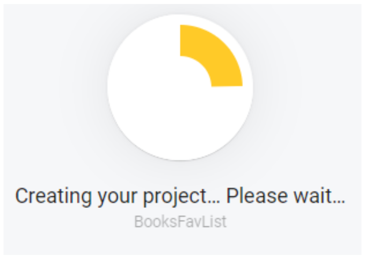

Creating your project… Please wait…
BooksFavList

Figure 5-d: Provisioning Resources (Step 3 of 3 - Creating the Firebase Project)

Once the project has been provisioned, you should see the following screen.

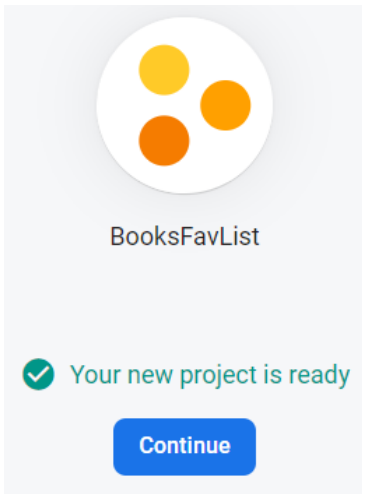


Figure 5-e: Firebase Project Created

Click **Continue**, which will lead us to the project dashboard.

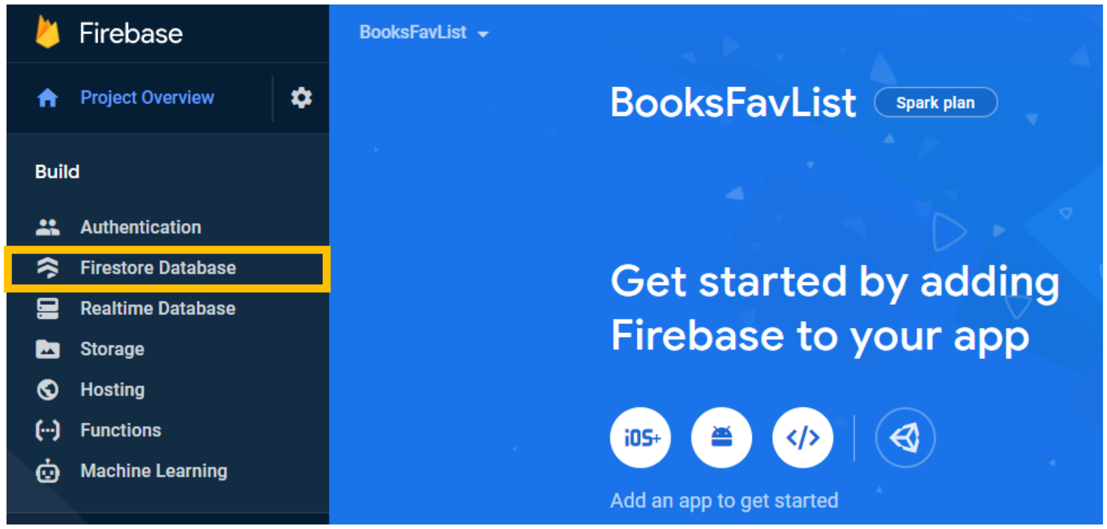


Figure 5-f: Firebase Project Dashboard

With the Firebase project created, we need to create a Firestore database, which you can do by clicking the navigation option highlighted in yellow in Figure 5-f, leading us to the following screen.

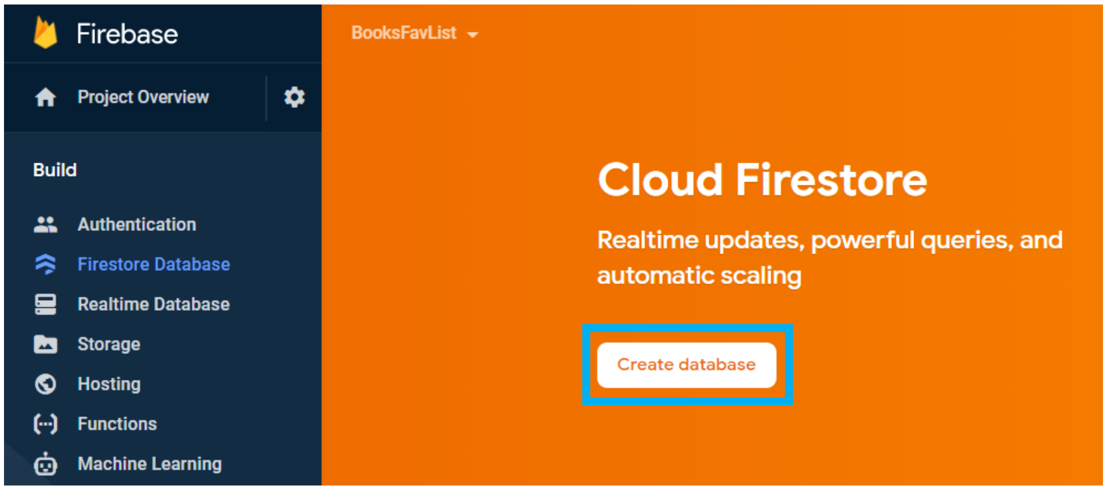
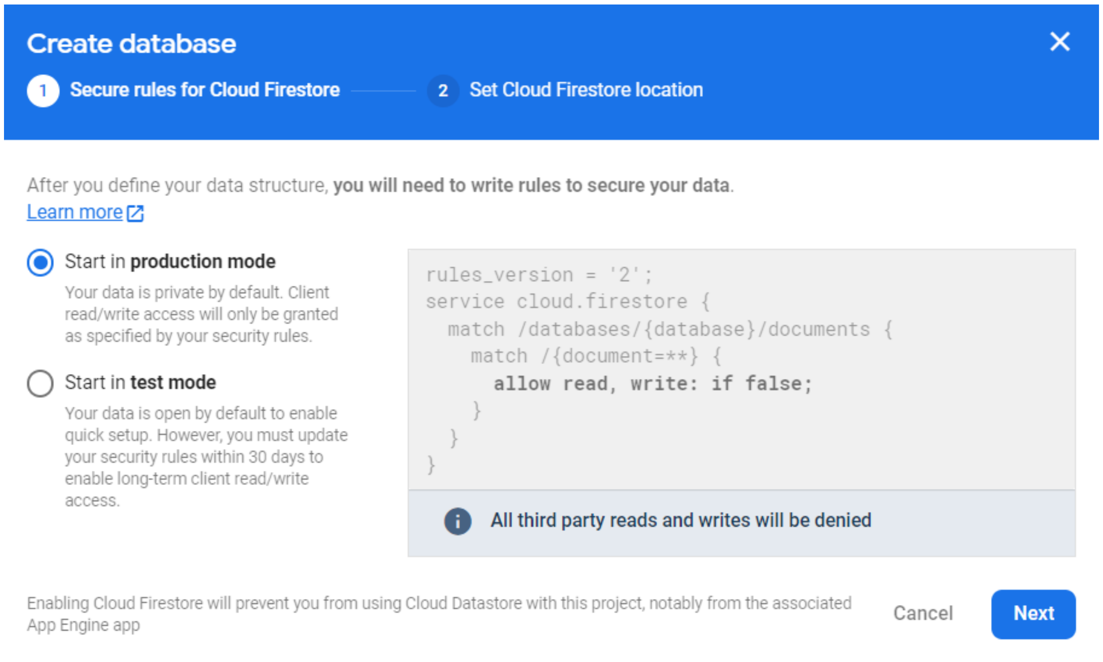


Figure 5-g: Cloud Firestore Main Page (Firestore Database)

Here, we can click the **Create database** button to continue creating the Firestore database; this will display the following window.


Create database
1 Secure rules for Cloud Firestore
2 Set Cloud Firestore location

After you define your data structure, **you will need to write rules to secure your data**.
Learn more

[✓ Start in **production mode**]
Your data is private by default. Client read/write access will only be granted as specified by your security rules.

Start in **test mode**
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

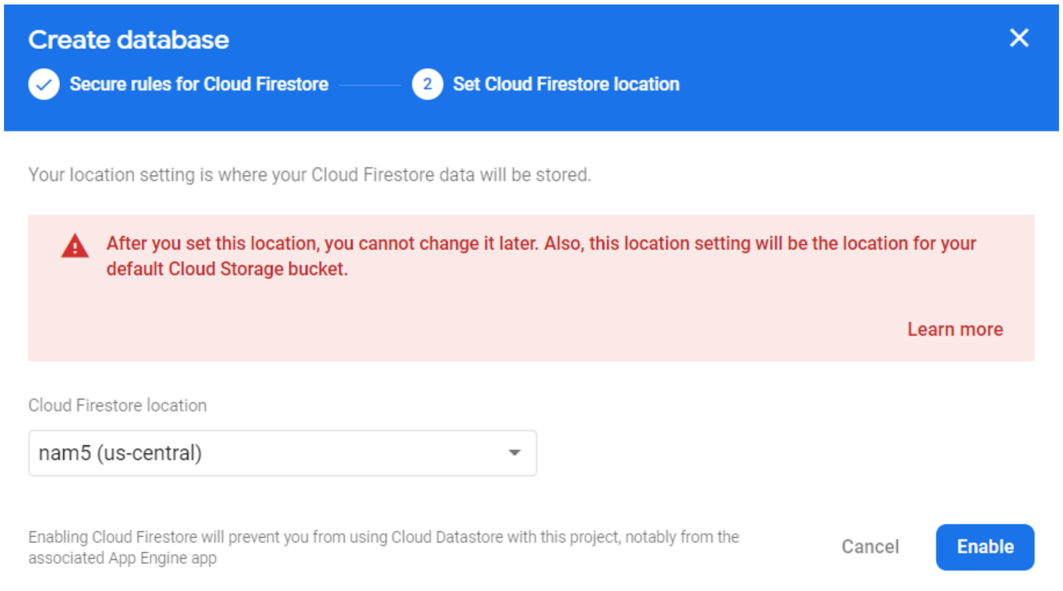
```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
    }
  }
}
```

All third party reads and writes will be denied

Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app

Figure 5-h: Create Database Window—Step 1 (Firestore Database)

Although I would usually recommend choosing the Start in test mode option, in this case, for the sake of simplicity and avoiding additional configuration later on (as the focus of this course is not Firebase), let's go with the **Start in production mode** option. So, let's leave the default option and click **Next**.


Create database

Secure rules for Cloud Firestore — 2 Set Cloud Firestore location

Your location setting is where your Cloud Firestore data will be stored.

After you set this location, you cannot change it later. Also, this location setting will be the location for your default Cloud Storage bucket.

Learn more

Cloud Firestore location

nam5 (us-central)

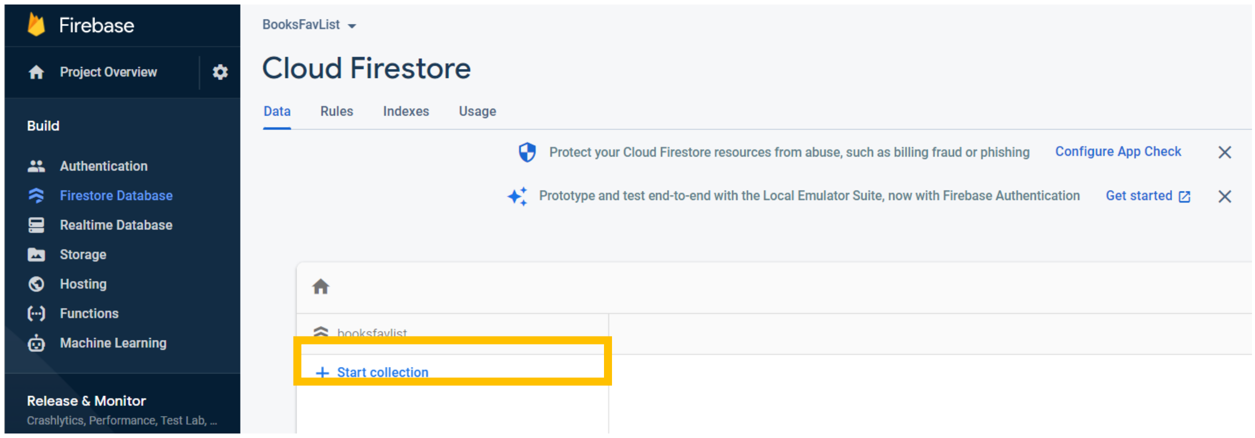
Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app

Figure 5-i: Create Database Window—Step 2 (Firestore Database)

Then, we can choose the Cloud Firestore location. I'll go with the default option (**nam5 (us-central)**), but feel free to select a location closer to where you are physically located.

So, to finish creating the Cloud Firestore database, click the **Enable** button to provision the required database resources.

The Cloud Firestore database is ready to be used, as we can see in Figure 5-j.



Cloud Firestore

Protect your Cloud Firestore resources from abuse, such as billing fraud or phishing

Configure App Check

Prototype and test end-to-end with the Local Emulator Suite, now with Firebase Authentication

Get started

booksfavlist

+ Start collection

Figure 5-j: Cloud Firestore Database Ready

Now we can create a collection and add some data, which will replace the hardcoded book data we added to our client-side code.

Adding data

Instead of having data hardcoded into the application like we currently have, let's move that data to Cloud Firestore, instead, and modify our application to retrieve the data from it.

To do that, click the **+ Start collection** option highlighted in the preceding figure—this will display the following window.

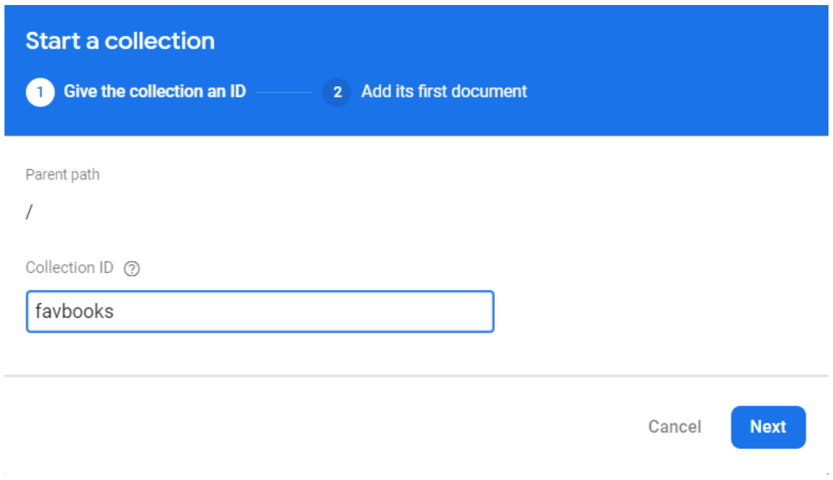


Figure 5-k: Start a collection (Step 1)—Firestore Database

I'll give the Collection ID the name favbooks and click **Next** to display the following window.

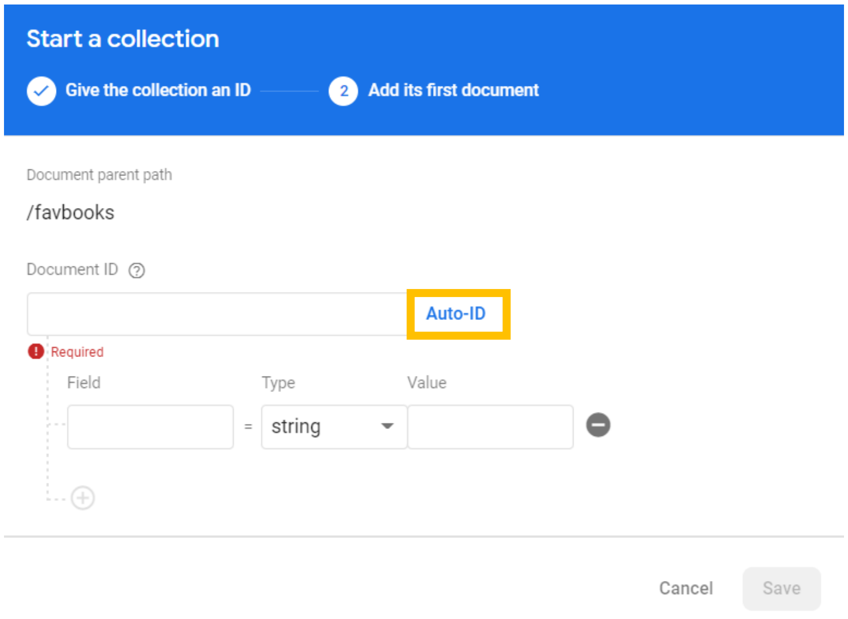
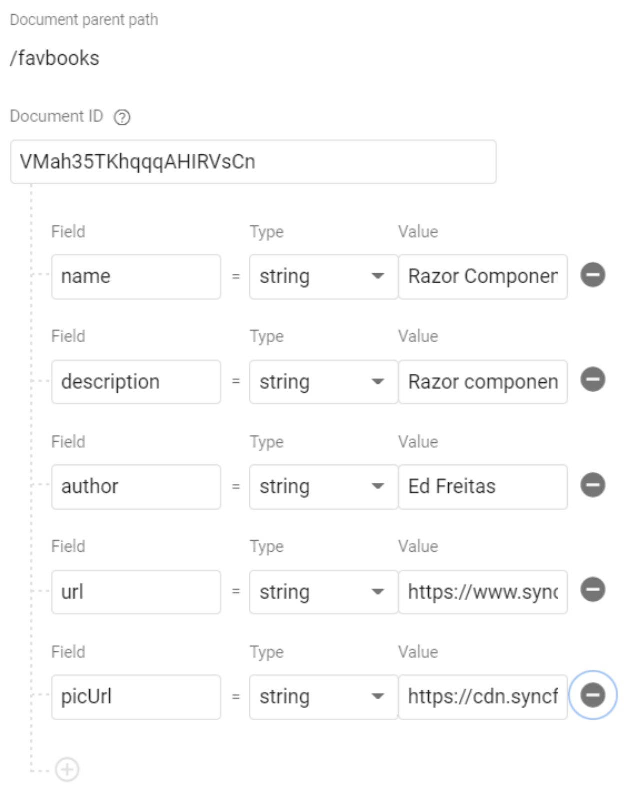


Figure 5-l: Start a collection (Step 2)—Firestore Database

Following that, click the **Auto-ID** button to generate the **Document ID**. Then, let's enter each field value using the same hardcoded values from the index.vue file. For the first document (seen in the following figure), we are adding the field values for *Razor Components Succinctly*.



Document parent path

/favbooks

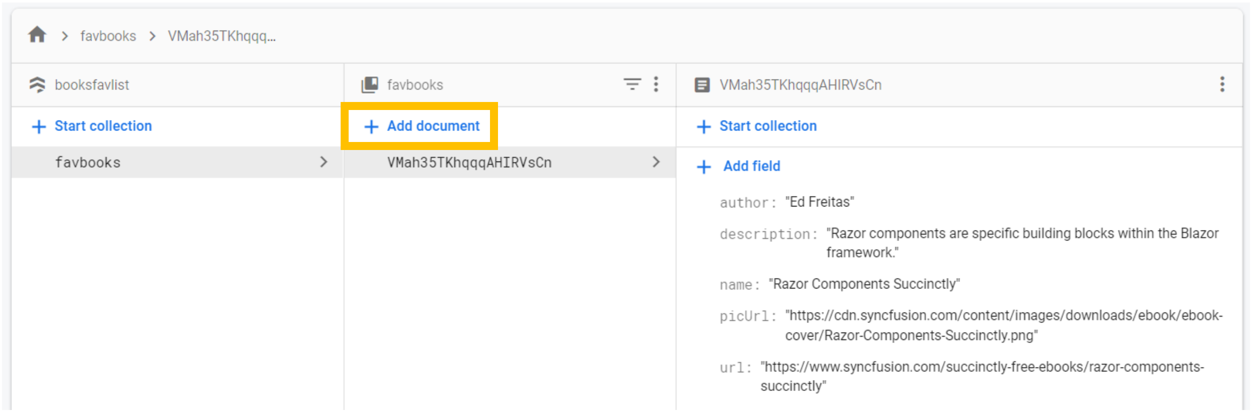
Document ID

VMah35TKhqqqAHIRVsCn

Field	Type	Value
name	string	Razor Componer
description	string	Razor componen
author	string	Ed Freitas
url	string	https://www.sync
picUrl	string	https://cdn.syncf

Figure 5-m: Start a collection (Adding a Document)

Click **Save** at the bottom of the screen (not visible in the preceding figure) to save the document.



```
author: "Ed Freitas"
description: "Razor components are specific building blocks within the Blazor framework."
name: "Razor Components Succinctly"
picUrl: "https://cdn.syncfusion.com/content/images/downloads/ebook/ebook-cover/Razor-Components-Succinctly.png"
url: "https://www.syncfusion.com/succinctly-free-ebooks/razor-components-succinctly"
```

Figure 5-n: Document Added

Repeat the same steps for adding the data for *Azure Virtual Desktop Succinctly* and *Ansible Succinctly*. To do that, click the **+ Add document** button.

Connecting the app to Firebase

With the data now available in the Cloud Firestore database, it's time to connect our application to Firebase, remove the hardcoded data from index.vue, and retrieve the data from the database.

Let's click the configuration icon and then click the **Project settings** option.

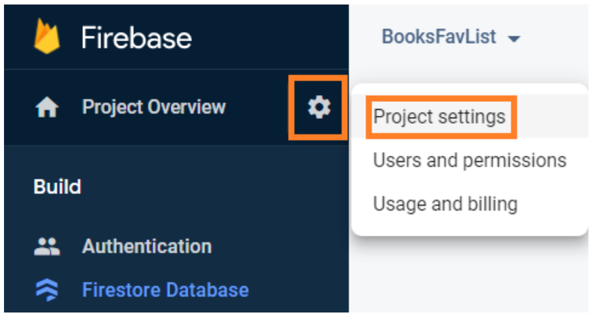
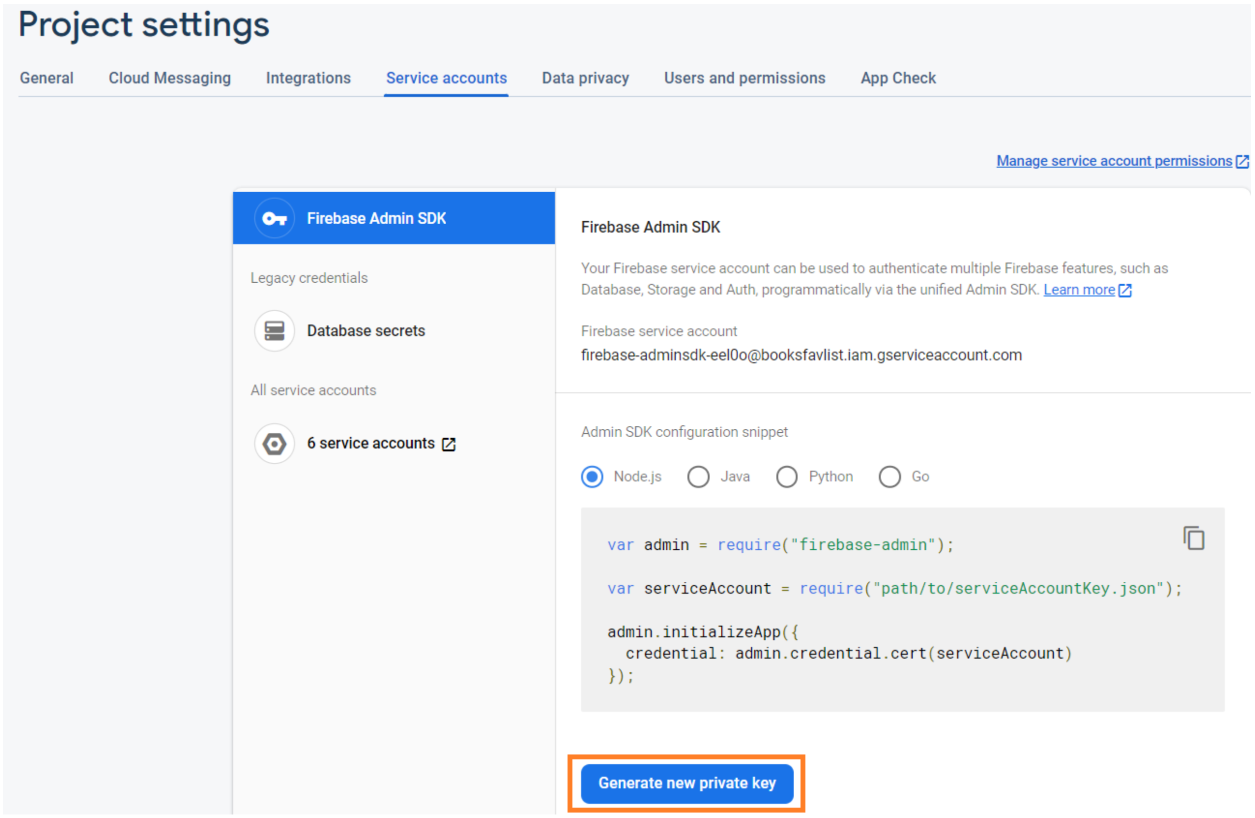


Figure 5-o: Project settings Option

Once you're in the Project settings view, click the **Service accounts** tab and the **Generate new private key** button.



Firebase Admin SDK

Your Firebase service account can be used to authenticate multiple Firebase features, such as Database, Storage and Auth, programmatically via the unified Admin SDK. Learn more

Firebase service account
firebase-adminsdk-eel0o@booksfavlist.iam.gserviceaccount.com

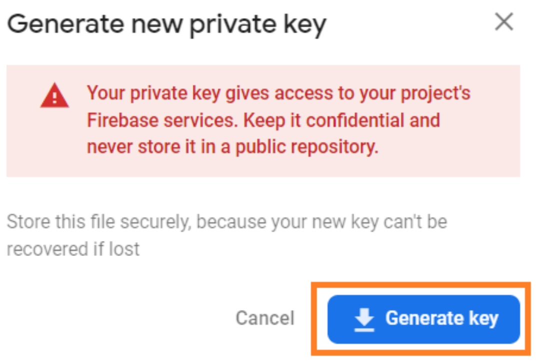
Admin SDK configuration snippet

[✓ Node.js] Java Python Go

```
var admin = require("firebase-admin");

var serviceAccount = require("path/to/serviceAccountKey.json");

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});
```

Figure 5-p: Project settings—Service accounts

Once you're done, the following window will be displayed.

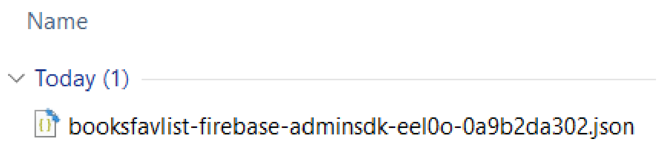

Generate new private key

Your private key gives access to your project's Firebase services. Keep it confidential and never store it in a public repository.

Store this file securely, because your new key can't be recovered if lost

Figure 5-q: Generate new private key—Service accounts

Following that, click the **Generate key** button to download the private key we will use within our Nuxt.js project. In my case, the file looks as follows.



booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json

Figure 5-r: The Generated Key File

Note: Depending on how you named your Firebase project (if it is different than the name I used), the name of the generated key file might be different than mine.

Move the downloaded file to the root folder where your Nuxt.js project resides, to ensure that it is not exposed through the client-side part of the application.

Here is how the generated key file should look within the VS Code EXPLORER panel once moved to the project's root folder.

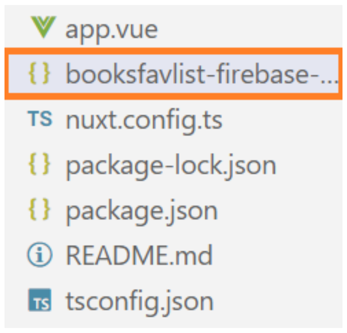


Figure 5-s: The Generated Key File (as seen in VS Code)

Installing the Firebase SDK

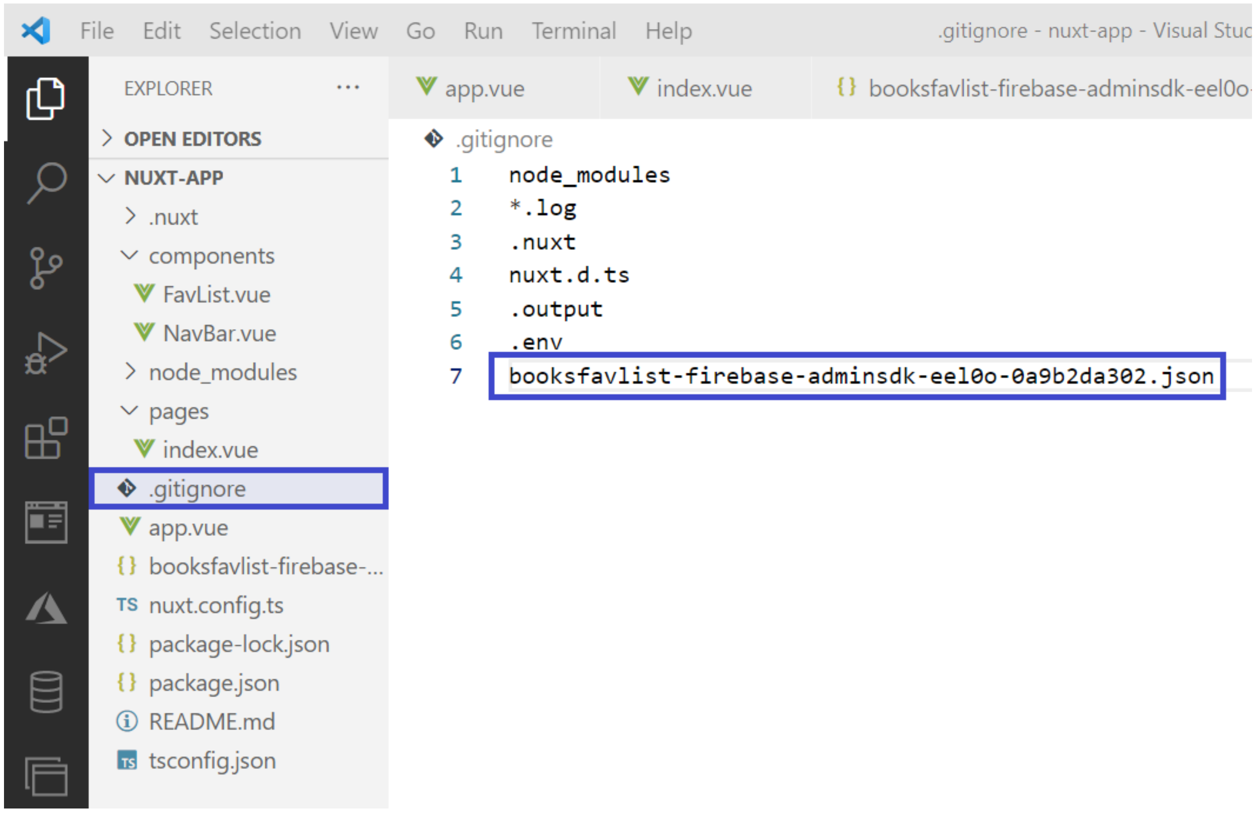
We must install the Firebase Admin SDK, the server-side version of Firebase. To do this, we need to open the built-in terminal within VS Code and type in the following command.

Code Listing 5-a: Command to Install the Firebase Admin SDK

```
npm install firebase-admin
```

Once the installation process has been finalized, make sure to add the name of the generated key file to your project's **.gitignore** file.

This prevents the file from being added to your Git repository, which is good practice and helps you avoid sharing sensitive project information by mistake.



```
.gitignore
node_modules
*.log
.nuxt
nuxt.d.ts
.output
.env
booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json
```

Figure 5-t: Adding the Generated Key File Name to .gitignore

Summary

At this stage, we have created a backend server for our project, hosted on the Firebase platform. In the next chapter, we will write our application's server-side logic, interacting directly with Firebase.

Chapter 6 Server Logic

Overview

We are ready to write the server-side logic of our application that will retrieve the data from the Cloud Firestore database.

Server logic

To begin, let's use the EXPLORER panel of VS Code to create a new folder called 'server' and a subfolder called 'api,' and within the api folder, create a new file called books.js.

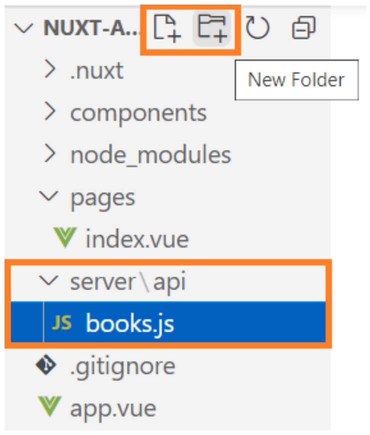


Figure 6-a: The server and api Folders with books.js (EXPLORER—VS Code)

Then, copy the following code and paste it into the books.js file.

Code Listing 6-a: books.js

```
import { initializeApp, getApps, cert } from 'firebase-admin/app';
import { getFirestore } from 'firebase-admin/firestore';

const fb_apps = getApps();

if (fb_apps.length <= 0) {
    initializeApp({
        credential:
            cert(
             './booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json'),
    });
}
```

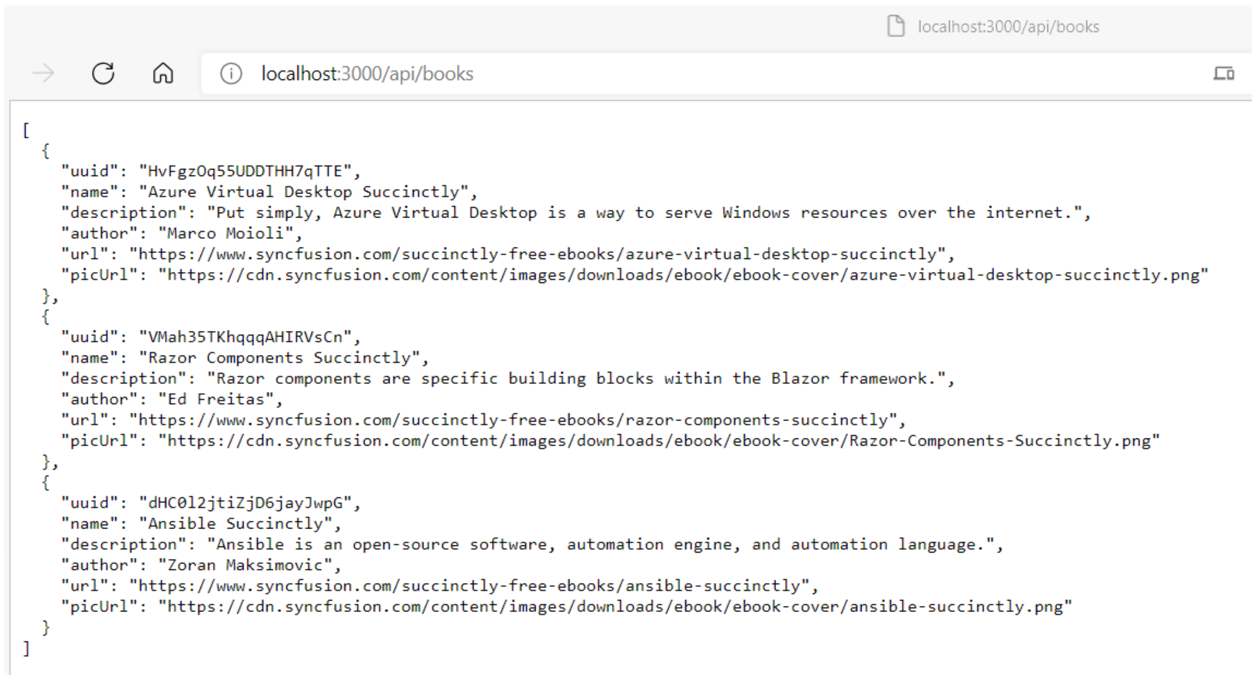
```
const func = async(rq, rs) => {
    const fs_db = getFirestore();
    const snapshot = await fs_db.collection('favbooks').get();
    const books = snapshot.docs.map(doc => {
        return {
            uuid: doc.id,
            name: doc.data().name,
            description: doc.data().description,
            author: doc.data().author,
            url: doc.data().url,
            picUrl: doc.data().picUrl
        }
    });

    return books;
}

export default func;
```

Before we go over this code, let's run the app by executing the **`npm run dev`** command from the built-in terminal within VS Code.

Once the app is running, open the browser and navigate to http://localhost:3000/api/books.

Note: Normally, Nuxt.js uses port 3000 by default. If it uses another port in your case, make sure to replace 3000 with the port number Nuxt.js is using on your machine. The *api* ***part of the URL corresponds to the api folder, and the*** *books* ***part of the URL corresponds to the books.js file.***

After navigating to that URL, you should see the following results returned by the application.



```
[
  {
    "uuid": "HvFgzOq55UDDTHH7qTTE",
    "name": "Azure Virtual Desktop Succinctly",
    "description": "Put simply, Azure Virtual Desktop is a way to serve Windows resources over the internet.",
    "author": "Marco Moioli",
    "url": "https://www.syncfusion.com/succinctly-free-ebooks/azure-virtual-desktop-succinctly",
    "picUrl": "https://cdn.syncfusion.com/content/images/downloads/ebook/ebook-cover/azure-virtual-desktop-succinctly.png"
  },
  {
    "uuid": "VMah35TKhqqqAHIRVsCn",
    "name": "Razor Components Succinctly",
    "description": "Razor components are specific building blocks within the Blazor framework.",
    "author": "Ed Freitas",
    "url": "https://www.syncfusion.com/succinctly-free-ebooks/razor-components-succinctly",
    "picUrl": "https://cdn.syncfusion.com/content/images/downloads/ebook/ebook-cover/Razor-Components-Succinctly.png"
  },
  {
    "uuid": "dHC0l2jtiZjD6jayJwpG",
    "name": "Ansible Succinctly",
    "description": "Ansible is an open-source software, automation engine, and automation language.",
    "author": "Zoran Maksimovic",
    "url": "https://www.syncfusion.com/succinctly-free-ebooks/ansible-succinctly",
    "picUrl": "https://cdn.syncfusion.com/content/images/downloads/ebook/ebook-cover/ansible-succinctly.png"
  }
]
```

Figure 6-b: The Results from Firebase

Essentially, we have created an API that can fetch data from the Cloud Firestore database using the Firebase Admin SDK.

Understanding the server logic

Let's go over the server code that we've written to understand what has been done. First, we began with the necessary imports.

```
import { initializeApp, getApps, cert } from 'firebase-admin/app';
import { getFirestore } from 'firebase-admin/firestore';
```

From the **`firebase-admin/app`** module, we imported the **`initializeApp`**, **`getApps`**, and **`cert`** methods.

Then, from the **`firebase-admin/firestore`** module, we imported the **`getFirestore`** method. All these methods are required to initialize the connection to Firebase, get the reference to the default application, and establish the connection to the database.

The next instruction within the server code is **`const fb_apps = getApps();`**. This method retrieves a list of all the initialized applications that exist within your Firebase account.

So, suppose there aren't any Firebase initialized applications (when the following condition **`fb_apps.length <= 0`** is true). In that case, we can initialize the connection to the Firebase application we created, which is done with the execution of the **`initializeApp`** method.

To make sure we initialize the correct Firebase application, we have to pass as a parameter the credentials we retrieved from Firebase (available within the generated key file)—this is done with the following instruction.

```
initializeApp({
  credential:
    cert(

      './booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json'),

  });
```

In this context, **credential** is the parameter's name, whereas the result returned by the **cert** method is the value assigned to the **credential** parameter.

What the **cert** method does in this context is read the contents of the generated key values contained within the **booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json** file.

Once we have initialized our Firebase application, we declare an asynchronous function called **func**, to which we pass request (**rq**) and response (**rs**) parameters.

The **func** function is responsible for retrieving the data from the Cloud Firestore database. To do that, we first get the connection to the Cloud Firestore database using the **getFirestore** method—this connection is assigned to the constant **fs_db**.

```
const fs_db = getFirestore();
```

Then, we invoke the **collection** method from the **fs_db** instance and pass in the name of the books collection within Firebase, which, as you might remember, we called **favbooks**.

```
const snapshot = await fs_db.collection('favbooks').get();
```

What the **get** method does is retrieve a snapshot of the data stored within the collection rather than the value of the collection itself. The reference to this snapshot is assigned to the constant with the same name: **snapshot**.

Then, the data is obtained by invoking the **docs** method from the **snapshot** instance. Given that the result obtained from the **docs** method is an array, we need to iterate through that array, which is why we use the **map** function.

```
snapshot.docs.map(...)
```

Within the **map** function, we use a **doc** variable that represents each item of the array returned by the **docs** method.

For each item within the array, we return an object that includes all the document properties stored in the Cloud Firestore database.

```
return {
  uuid: doc.id,
  name: doc.data().name,
  description: doc.data().description,
  author: doc.data().author,
  url: doc.data().url,
  picUrl: doc.data().picUrl

}
```

Notice that, except for the **`uuid`** value, all the document properties stored in the Cloud Firestore database are available through the **`data`** property of the **`doc`** instance—and this can be further optimized, as we'll see shortly.

Finally, once we've gone through all the items in the array, we can return the list of results assigned to the **`books`** variable.

```
return books;
```

Optimizing the server logic

As explained previously, except for the **`uuid`** value, all the document properties stored in the Cloud Firestore database are available through the **`data`** property of the **`doc`** instance.

So, instead of explicitly mentioning each of the properties:

```
name: doc.data().name,
description: doc.data().description,
author: doc.data().author,
url: doc.data().url,
picUrl: doc.data().picUrl
```

We can use the JavaScript spread operator (**...**) to include all the properties accessible through the **`data`** method.

```
...doc.data()
```

So, the modified and optimized code would look as follows (with the change emphasized in **bold**).

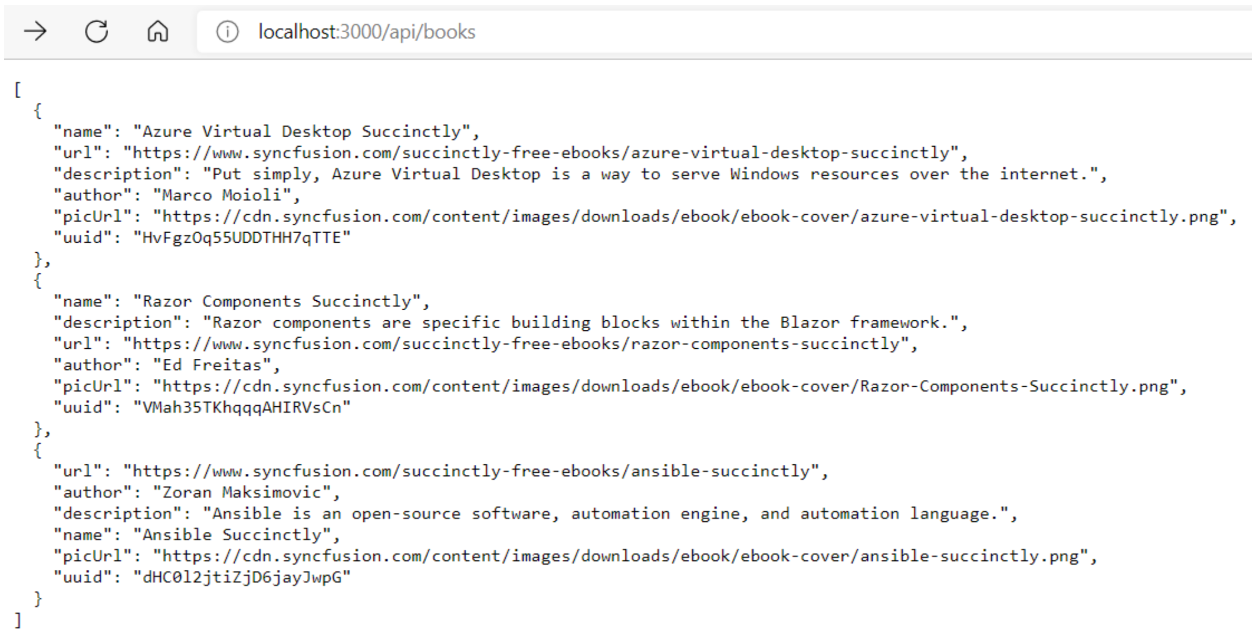
Code Listing 6-b: books.js (Optimized)

```
import { initializeApp, getApps, cert } from 'firebase-admin/app';
import { getFirestore } from 'firebase-admin/firestore';
```

```
const fb_apps= getApps();

if (fb_apps.length <= 0) {
    initializeApp({
        credential:
            cert(
              './booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json'),
    });
}

const func = async(rq, rs) => {
    const fs_db = getFirestore();
    const snapshot = await fs_db.collection('favbooks').get();
    const books = snapshot.docs.map(doc => {
        return {
            ...doc.data(),
            uuid: doc.id
        }
    });

    return books;
}

export default func;
```

If you rerun the application (`npm run dev`) from the built-in terminal within VS Code and navigate to http://localhost:3000/api/books, you will see the same results.

The only difference this time might be that the order of the books is not the same, and the `uuid` field will appear as the last property for each item.



```
[
  {
    "name": "Azure Virtual Desktop Succinctly",
    "url": "https://www.syncfusion.com/succinctly-free-ebooks/azure-virtual-desktop-succinctly",
    "description": "Put simply, Azure Virtual Desktop is a way to serve Windows resources over the internet.",
    "author": "Marco Moioli",
    "picUrl": "https://cdn.syncfusion.com/content/images/downloads/ebook/ebook-cover/azure-virtual-desktop-succinctly.png",
    "uuid": "HvFgzOq55UDDTHH7qTTE"
  },
  {
    "name": "Razor Components Succinctly",
    "description": "Razor components are specific building blocks within the Blazor framework.",
    "url": "https://www.syncfusion.com/succinctly-free-ebooks/razor-components-succinctly",
    "author": "Ed Freitas",
    "picUrl": "https://cdn.syncfusion.com/content/images/downloads/ebook/ebook-cover/Razor-Components-Succinctly.png",
    "uuid": "VMah35TKhqqqAHIRVsCn"
  },
  {
    "url": "https://www.syncfusion.com/succinctly-free-ebooks/ansible-succinctly",
    "author": "Zoran Maksimovic",
    "description": "Ansible is an open-source software, automation engine, and automation language.",
    "name": "Ansible Succinctly",
    "picUrl": "https://cdn.syncfusion.com/content/images/downloads/ebook/ebook-cover/ansible-succinctly.png",
    "uuid": "dHC0l2jtiZjD6jayJwpG"
  }
]
```

Figure 6-c: The Results from Firebase

Retrieving data from Firebase

Now that we can retrieve the data from the Cloud Firestore database, we must remove the hardcoded data within the index.vue file and load the data dynamically.

So, open the index.vue file and remove the items in the **books** property (returned by the **data** method). The changes to the updated code are emphasized in **bold** in the following code.

Code Listing 6-c: index.vue (Updated)

```
<template>
    <div class="container">
        <FavList ref="favlist"/>
        <br />
        <div class="row">
            <div v-for="(book, index) in books"
                :key="book.uuid + '_' + index"
                class="col-md-4"
            >
                <div class="card-mb-3">
                    <a target="_blank" :href="book.url">
                        <img :src="book.picUrl" class="card-img-top">
                    </a>
                    <div class="card-body">
                        <h5 class="card-title">
```

```
                                {{ book.name }}
                            </h5>
                            <p class="card-text">
                                {{ book.description }}
                            </p>
                            <p class="card-text">
                                {{ book.author }}
                            </p>
                            <div class="d-grid">
                                <button @click="addToFavs(book)"
                                        class="btn btn-outline-primary">
                                    Add to Favorites
                                </button>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
</template>

<style scoped>
    .img {
        width: 80%;
        height: auto;
    }

    .center {
        display: block;
        margin-left: auto;
        margin-right: auto;
        width: 100%;
    }
</style>

<script>
    import FavList from '../components/FavList';

    export default {
        components: { FavList },
        data() {
            return {
                books: []
            }
        },
        methods: {
            addToFavs(book) {
                let inFavs = false;
                const favs = this.$refs.favlist;

                for (const fav of favs.favorites) {
                    if (fav.uuid === book.uuid) {
                        inFavs = true;
                        break;
                    }
                }

                if (!inFavs) {
                    favs.favorites.push({...book});
                }
            }
        }
    }
</script>
```

Next, we need to add extra code to the index.vue file, allowing us to retrieve the data from the Cloud Firestore database—which we can do as follows.

```
<script setup>
    const { data: books } =
      await useAsyncData('books', () => $fetch('/api/books'));
</script>
```

This uses the **useAsyncData** function from Nuxt.js 3 to perform data fetching. So, within the setup of index.vue, the **useAsyncData** function is invoked.

The **useAsyncData** function uses the browser's Fetch API (**$fetch**) to which the server URL is passed as a parameter, retrieving the list of books.

The result of the execution of the **useAsyncData** function is a **data** object that we give the name: **books**.

By doing this, we no longer need to return the **books** array from the **data** method and can remove the **data** method altogether. The updated index.vue code is now as follows.

Code Listing 6-d: index.vue (Updated)

```
<script setup>
    const { data: books } =
      await useAsyncData('books', () => $fetch('/api/books'));
 </script>

 <template>
     <div class="container">
         <FavList ref="favlist"/>
         <br />
         <div class="row">
             <div v-for="(book, index) in books"
                 :key="book.uuid + '_' + index"
                 class="col-md-4"
             >
                 <div class="card-mb-3">
                     <a target="_blank" :href="book.url">
                         <img :src="book.picUrl" class="card-img-top">
                     </a>
                     <div class="card-body">
                         <h5 class="card-title">
                             {{ book.name }}
                         </h5>
                         <p class="card-text">
                             {{ book.description }}
                         </p>
                         <p class="card-text">
                             {{ book.author }}
                         </p>
                         <div class="d-grid">
                             <button @click="addToFavs(book)"
                                 class="btn btn-outline-primary">
                                 Add to Favorites
                             </button>
                         </div>
                     </div>
                 </div>
             </div>
         </div>
     </div>
 </template>

 <style scoped>
     .img {
         width: 80%;
         height: auto;
     }
     .center {
         display: block;
         margin-left: auto;
         margin-right: auto;
         width: 100%;
     }
 </style>

 <script>
     import FavList from '../components/FavList';

     export default {
         components: { FavList },
         methods: {
             addToFavs(book) {
                 let inFavs = false;
                 const favs = this.$refs.favlist;

                 for (const fav of favs.favorites) {
                     if (fav.uuid === book.uuid) {
                         inFavs = true;
                         break;
                     }
                 }

                 if (!inFavs) {
                     favs.favorites.push({...book});
                 }
             }
         }
     }
 </script>
```

If you rerun the application (**`npm run dev`**) from the built-in terminal within VS Code, you should see that it works, and this time the list of books is retrieved from the Cloud Firestore database instead of being loaded locally.

You might have noticed (also depending on how fast your internet connection is) that because the list of books is being retrieved from Firebase, the application takes slightly longer to load.

In other words, the UI takes slightly longer to render, and this is because the UI is only generated after the data has been retrieved.

Lazy loading

For our example, we have hardly any data; however, in a real-world (production-ready) case, we might have much more data to retrieve, and as such, it wouldn't be recommended to have the UI blocked (without being rendered) while the data loads.

We can achieve this by using a technique known as *lazy loading*, for which Nuxt.js 3 provides a couple of functions for this purpose.

Let's refactor the index.vue code a bit further to accommodate lazy loading. The changes are emphasized in **bold** in the following code.

Code Listing 6-e: index.vue (Updated—with Lazy Loading)

```
<script setup>
    const { pending, data: books } = await useLazyFetch('/api/books');
</script>

<template>
    <div v-if="pending">
        Loading ...
    </div>
    <div v-else class="container">
        <FavList ref="favlist"/>
        <br />
        <div class="row">
            <div v-for="(book, index) in books"
                 :key="book.uuid + '_' + index"
                class="col-md-4"
            >
                <div class="card-mb-3">
                    <a target="_blank" :href="book.url">
                        <img :src="book.picUrl" class="card-img-top">
                    </a>
                    <div class="card-body">
                        <h5 class="card-title">
                            {{ book.name }}
                        </h5>
                        <p class="card-text">
                            {{ book.description }}
                        </p>
                        <p class="card-text">
                            {{ book.author }}
                        </p>
                        <div class="d-grid">
                            <button @click="addToFavs(book)"
```

```
                                    class="btn btn-outline-primary">
                                Add to Favorites
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>

<style scoped>
    .img {
        width: 80%;
        height: auto;
    }

    .center {
        display: block;
        margin-left: auto;
        margin-right: auto;
        width: 100%;
    }
</style>

<script>
    import FavList from '../components/FavList';

    export default {
        components: { FavList },
        methods: {
            addToFavs(book) {
                let inFavs = false;
                const favs = this.$refs.favlist;

                for (const fav of favs.favorites) {
                    if (fav.uuid === book.uuid) {
                        inFavs = true;
                        break;
                    }
                }

                if (!inFavs) {
                    favs.favorites.push({...book});
                }
            }
        }
    }
</script>
```

The first change is that we are using the **`useLazyFetch`** function, and as you can see, this function only requires one parameter: the URL endpoint of the data to retrieve.

```
const { pending, data: books } = await useLazyFetch('/api/books');
```

Note that this function returns a **`pending`** variable, which checks whether the data retrieval operation is still pending or has finalized.

This way, if the data retrieval operation is still taking place, we can display a loading message, which is what the following code does.

Notice how we use the **`v-if`** directive to check if the value of **`pending`** is true (which would indicate that the retrieval operation has not been finalized).

```
<div v-if="pending">
  Loading ...
</div>
```

On the other hand, if the value of **`pending`** is false, it would indicate that the data retrieval operation has been finalized. Therefore, the **`div`** containing the list of books can be shown—which is why we've added a **`v-else`** directive.

```
<div v-else class="container">
...
</div>
```

Building and generating

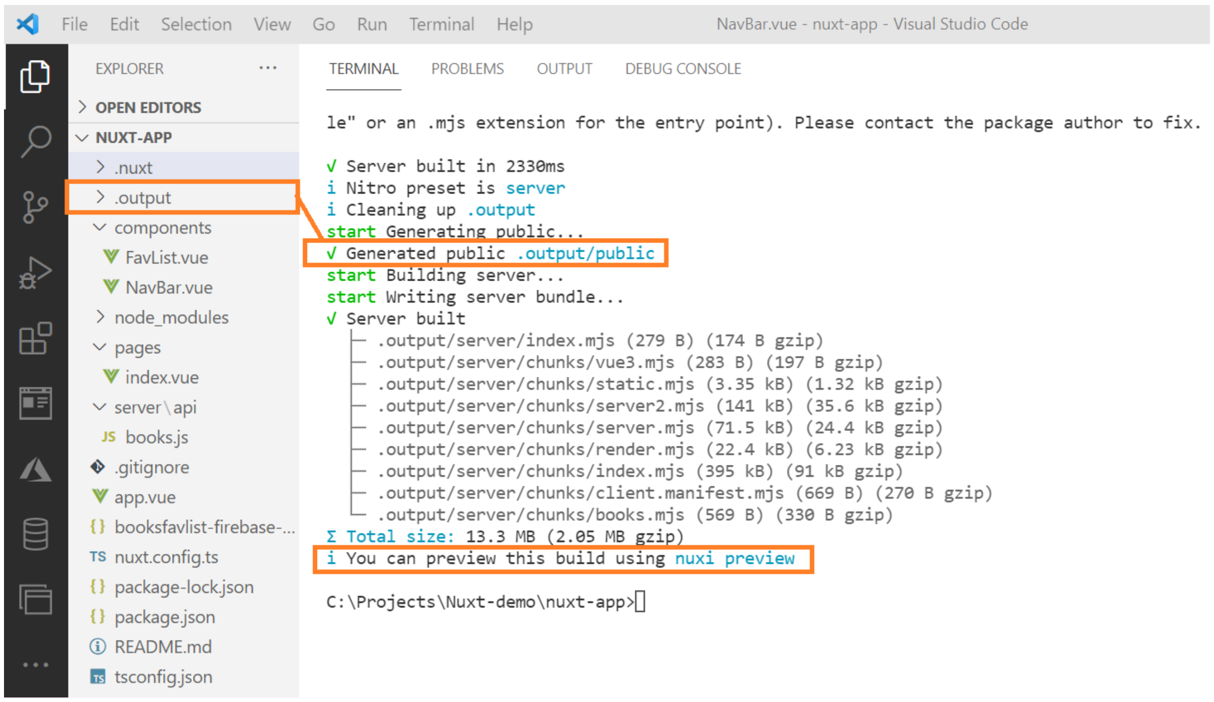
We are now done with our application, so well done if you've been following along all this time! Now it's time to generate our server-side rendered application.

We can execute the following command from the built-in terminal within VS Code.

Code Listing 6-f: Building and Generating the App

```
npm run build
```

Once this command has been executed, you should see an output similar to the following one within your terminal.



```
le" or an .mjs extension for the entry point). Please contact the package author to fix.

√ Server built in 2330ms
i Nitro preset is server
i Cleaning up .output
start Generating public...
√ Generated public .output/public
start Building server...
start Writing server bundle...
√ Server built
  ├─ .output/server/index.mjs (279 B) (174 B gzip)
  ├─ .output/server/chunks/vue3.mjs (283 B) (197 B gzip)
  ├─ .output/server/chunks/static.mjs (3.35 kB) (1.32 kB gzip)
  ├─ .output/server/chunks/server2.mjs (141 kB) (35.6 kB gzip)
  ├─ .output/server/chunks/server.mjs (71.5 kB) (24.4 kB gzip)
  ├─ .output/server/chunks/render.mjs (22.4 kB) (6.23 kB gzip)
  ├─ .output/server/chunks/index.mjs (395 kB) (91 kB gzip)
  ├─ .output/server/chunks/client.manifest.mjs (669 B) (270 B gzip)
  └─ .output/server/chunks/books.mjs (569 B) (330 B gzip)
Σ Total size: 13.3 MB (2.05 MB gzip)
i You can preview this build using nuxi preview

C:\Projects\Nuxt-demo\nuxt-app>
```

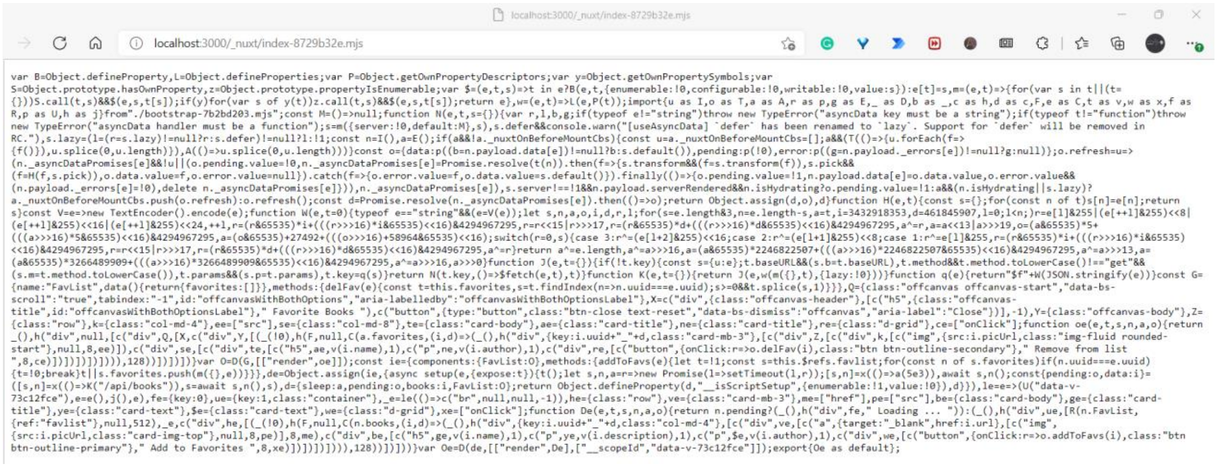
Figure 6-d: Building and Generating Output

Note that Nuxt.js has created an .output folder that contains the production-ready (minified and optimized) server-side rendered application code, which can be executed by running the following command.

Code Listing 6-g: Executing the Production-Ready App

```
npm run start
```

Once the application has started, if you right-click within your browser and click the option **View page source**, you'll see that the code has been minified and optimized for production. Here is an example from my machine.


localhost:3000/_nuxt/index-8729b32e.mjs
localhost:3000/_nuxt/index-8729b32e.mjs

```
var B=Object.defineProperty,L=Object.defineProperties;var P=Object.getOwnPropertyDescriptors;var y=Object.getOwnPropertySymbols;var
S=Object.prototype.hasOwnProperty,z=Object.prototype.propertyIsEnumerable;var $=(e,t,s)=>t in e?B(e,t,{enumerable:!0,configurable:!0,writable:!0,value:s}):e[t]=s,m=(e,t)=>{for(var s in t||(t=
{}))S.call(t,s)&&$(e,s,t[s]);if(y)for(var s of y(t))z.call(t,s)&&$(e,s,t[s]);return e},w=(e,t)=>L(e,P(t));import{u as I,o as T,a as A,r as p,g as E,_ as D,b as _,c as h,d as c,F,e as C,t as v,w as x,f as
R,p as U,h as j}from"./bootstrap-7b2bd203.mjs";const M=()=>null;function N(e,t,s={}){var r,l,b,g;if(typeof e!="string")throw new TypeError("asyncData key must be a string");if(typeof t!="function")throw
new TypeError("asyncData handler must be a function");s=m({server:!0,default:M},s),s.defer&&console.warn("[useAsyncData] `defer` has been renamed to `lazy`. Support for `defer` will be removed in
RC."),s.lazy=(l=(r=s.lazy)!=null?r:s.defer)!=null?l:!1;const n=I(),a=E();if(a&&!a._nuxtOnBeforeMountCbs){const u=a._nuxtOnBeforeMountCbs=[];a&&(T(()=>{u.forEach(f=>
{f()}),u.splice(0,u.length)}),A(()=>u.splice(0,u.length)))}const o={data:p((b=n.payload.data[e])!=null?b:s.default()),pending:p(!0),error:p((g=n.payload._errors[e])!=null?g:null)};o.refresh=u=>
(n._asyncDataPromises[e]&&!u||(o.pending.value=!0,n._asyncDataPromises[e]=Promise.resolve(t(n)).then(f=>{s.transform&&(f=s.transform(f)),s.pick&&
(f=H(f,s.pick)),o.data.value=f,o.error.value=null}).catch(f=>{o.error.value=f,o.data.value=s.default()}).finally(()=>{o.pending.value=!1,n.payload.data[e]=o.data.value,o.error.value&&
(n.payload._errors[e]=!0),delete n._asyncDataPromises[e]})),n._asyncDataPromises[e]),s.server!==!1&&n.payload.serverRendered&&n.isHydrating?o.pending.value=!1:a&&(n.isHydrating||s.lazy)?
a._nuxtOnBeforeMountCbs.push(o.refresh):o.refresh();const d=Promise.resolve(n._asyncDataPromises[e]).then(()=>o);return Object.assign(d,o),d}function H(e,t){const s={};for(const n of t)s[n]=e[n];return
s}const V=e=>new TextEncoder().encode(e);function W(e,t=0){typeof e=="string"&&(e=V(e));let s,n,a,o,i,d,r,l;for(s=e.length&3,n=e.length-s,a=t,i=3432918353,d=461845907,l=0;l<n;)r=e[l]&255|(e[++l]&255)<<8|
(e[++l]&255)<<16|(e[++l]&255)<<24,++l,r=(r&65535)*i+(((r>>>16)*i&65535)<<16)&4294967295,r=r<<15|r>>>17,r=(r&65535)*d+(((r>>>16)*d&65535)<<16)&4294967295,a^=r,a=a<<13|a>>>19,o=(a&65535)*5+
(((a>>>16)*5&65535)<<16)&4294967295,a=(o&65535)+27492+(((o>>>16)+58964&65535)<<16);switch(r=0,s){case 3:r^=(e[l+2]&255)<<16;case 2:r^=(e[l+1]&255)<<8;case 1:r^=e[l]&255,r=(r&65535)*i+(((r>>>16)*i&65535)
<<16)&4294967295,r=r<<15|r>>>17,r=(r&65535)*d+(((r>>>16)*d&65535)<<16)&4294967295,a^=r}return a^=e.length,a^=a>>>16,a=(a&65535)*2246822507+(((a>>>16)*2246822507&65535)<<16)&4294967295,a^=a>>>13,a=
(a&65535)*3266489909+(((a>>>16)*3266489909&65535)<<16)&4294967295,a^=a>>>16,a>>>0}function J(e,t={}){if(!t.key){const s={u:e};t.baseURL&&(s.b=t.baseURL),t.method&&t.method.toLowerCase()!=="get"&&
(s.m=t.method.toLowerCase()),t.params&&(s.p=t.params),t.key=q(s)}return N(t.key,()=>$fetch(e,t),t)}function K(e,t={}){return J(e,w(m({},t),{lazy:!0}))}function q(e){return"$f"+W(JSON.stringify(e))}const G=
{name:"FavList",data(){return{favorites:[]}},methods:{delFav(e){const t=this.favorites,s=t.findIndex(n=>n.uuid===e.uuid);s>=0&&t.splice(s,1)}}},Q={class:"offcanvas offcanvas-start","data-bs-
scroll":"true",tabindex:"-1",id:"offcanvasWithBothOptions","aria-labelledby":"offcanvasWithBothOptionsLabel"},X=c("div",{class:"offcanvas-header"},[c("h5",{class:"offcanvas-
title",id:"offcanvasWithBothOptionsLabel"}," Favorite Books "),c("button",{type:"button",class:"btn-close text-reset","data-bs-dismiss":"offcanvas","aria-label":"Close"})],-1),Y={class:"offcanvas-body"},Z=
{class:"row"},k={class:"col-md-4"},ee=["src"],se={class:"col-md-8"},te={class:"card-body"},ae={class:"card-title"},ne={class:"card-title"},re={class:"d-grid"},ce=["onClick"];function oe(e,t,s,n,a,o){return
_(),h("div",null,[c("div",Q,[X,c("div",Y,[(_(!0),h(F,null,C(a.favorites,(i,d)=>(_(),h("div",{key:i.uuid+"_"+d,class:"card-mb-3"},[c("div",Z,[c("div",k,[c("img",{src:i.picUrl,class:"img-fluid rounded-
start"},null,8,ee)]),c("div",se,[c("div",te,[c("h5",ae,v(i.name),1),c("p",ne,v(i.author),1),c("div",re,[c("button",{onClick:r=>o.delFav(i),class:"btn btn-outline-secondary"}," Remove from list
",8,ce)])])])])]))),128))])])])}var O=D(G,[["render",oe]]);const ie={components:{FavList:O},methods:{addToFavs(e){let t=!1;const s=this.$refs.favlist;for(const n of s.favorites)if(n.uuid===e.uuid)
{t=!0;break}t||s.favorites.push(m({},e))}}},de=Object.assign(ie,{async setup(e,{expose:t}){t();let s,n,a=r=>new Promise(l=>setTimeout(l,r));[s,n]=x(()=>a(5e3)),await s,n();const{pending:o,data:i}=
([s,n]=x(()=>K("/api/books")),s=await s,n(),s),d={sleep:a,pending:o,books:i,FavList:O};return Object.defineProperty(d,"__isScriptSetup",{enumerable:!1,value:!0}),d}}),le=e=>(U("data-v-
73c12fce"),e=e(),j(),e),fe={key:0},ue={key:1,class:"container"},_e=le(()=>c("br",null,null,-1)),he={class:"row"},ve={class:"card-mb-3"},me=["href"],pe=["src"],be={class:"card-body"},ge={class:"card-
title"},ye={class:"card-text"},$e={class:"card-text"},we={class:"d-grid"},xe=["onClick"];function De(e,t,s,n,a,o){return n.pending?(_(),h("div",fe," Loading ... ")):(_(),h("div",ue,[R(n.FavList,
{ref:"favlist"},null,512),_e,c("div",he,[(_(!0),h(F,null,C(n.books,(i,d)=>(_(),h("div",{key:i.uuid+"_"+d,class:"col-md-4"},[c("div",ve,[c("a",{target:"_blank",href:i.url},[c("img",
{src:i.picUrl,class:"card-img-top"},null,8,pe)],8,me),c("div",be,[c("h5",ge,v(i.name),1),c("p",ye,v(i.description),1),c("p",$e,v(i.author),1),c("div",we,[c("button",{onClick:r=>o.addToFavs(i),class:"btn
btn-outline-primary"}," Add to Favorites ",8,xe)])])])]))),128))])]))}var Oe=D(de,[["render",De],["__scopeId","data-v-73c12fce"]]);export{Oe as default};
```

Figure 6-e: Production-Ready Code

Deployment

Nuxt.js offers the option to deploy your application to a myriad of the most popular hosting services available on the market today. You can find detailed instructions on deploying your application to any of these services on the official Nuxt.js documentation. I invite you to explore this resource in your spare time and check which hosting solutions work best for you.

Complete app code

The listings that follow contain all the finished code for each file that is part of the application we've built throughout this book.

Code Listing 6-h: Finished App (components\FavList.vue)

```
<template>
    <div>
        <div class="offcanvas offcanvas-start" data-bs-scroll="true"
            tabindex="-1"
            id="offcanvasWithBothOptions"
            aria-labelledby="offcanvasWithBothOptionsLabel">
            <div class="offcanvas-header">
                <h5 class="offcanvas-title"
                    id="offcanvasWithBothOptionsLabel">
                    Favorite Books
                </h5>
                <button type="button" class="btn-close text-reset"
                    data-bs-dismiss="offcanvas"
                    aria-label="Close">
                </button>
            </div>
            <div class="offcanvas-body">
                <div v-for="(fav, index) in favorites"
                    :key="fav.uuid + '_' + index"
                    class="card-mb-3">
                    <div class="row">
                        <div class="col-md-4">
                            <img :src="fav.picUrl"
                                class="img-fluid rounded-start">
                        </div>
                        <div class="col-md-8">
                            <div class="card-body">
                                <h5 class="card-title">
                                    {{ fav.name }}
                                </h5>
                                <p class="card-title">
                                    {{ fav.author }}
                                </p>
                                <div class="d-grid">
                                    <button @click="delFav(fav)"
                                      class="btn btn-outline-secondary">
                                        Remove from list
                                    </button>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>

<script>
    export default {
        name: 'FavList',
        data() {
            return {
                favorites: []
            }
        },
        methods: {
            delFav(fav) {
                const favs = this.favorites;
                const idx = favs.findIndex(item =>
                                           item.uuid === fav.uuid);
                if (idx >= 0) {
                    favs.splice(idx, 1);
                }
            }
        }
    }
</script>
```

Code Listing 6-i: Finished App (components\NavBar.vue)

```
<template>
```

```
    <div>
        <nav class="navbar navbar-expand-lg navbar-light"
            style="background-color: #e3f2fd;">
              <div class="container-fluid">
                <a class="navbar-brand" href="#">Books List</a>
                <NuxtLink to="/about">About</NuxtLink>
                <div class="d-flex input-group w-auto">
                    <button
                        class="btn btn-outline-primary"
                        type="button"
                        data-mdb-ripple-color="dark"
                        data-bs-toggle="offcanvas"
                        data-bs-target="#offcanvasWithBothOptions"
                        aria-controls="offcanvasWithBothOptions"
                    >
                        Favorites
                    </button>
                </div>
            </div>
        </nav>
    </div>
</template>
```

Code Listing 6-j: Finished App (pages\index.vue)

```
<script setup>
    const { pending, data: books } = await useLazyFetch('/api/books');
</script>

<template>
    <div v-if="pending">
        Loading ...
    </div>
    <div v-else class="container">
        <FavList ref="favlist"/>
        <br />
        <div class="row">
            <div v-for="(book, index) in books"
                :key="book.uuid + '_' + index"
                class="col-md-4"
            >
                <div class="card-mb-3">
                    <a target="_blank" :href="book.url">
```

```
                        <img :src="book.picUrl" class="card-img-top">
                    </a>
                    <div class="card-body">
                        <h5 class="card-title">
                            {{ book.name }}
                        </h5>
                        <p class="card-text">
                            {{ book.description }}
                        </p>
                        <p class="card-text">
                            {{ book.author }}
                        </p>
                        <div class="d-grid">
                            <button @click="addToFavs(book)"
                                    class="btn btn-outline-primary">
                                Add to Favorites
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</template>

<style scoped>
    .img {
        width: 80%;
        height: auto;
    }

    .center {
        display: block;
        margin-left: auto;
        margin-right: auto;
        width: 100%;
    }
</style>

<script>
    import FavList from '../components/FavList';

    export default {
        components: { FavList },
        methods: {
            addToFavs(book) {
                let inFavs = false;
                const favs = this.$refs.favlist;

                for (const fav of favs.favorites) {
                    if (fav.uuid === book.uuid) {
                        inFavs = true;
                        break;
                    }
                }

                if (!inFavs) {
                    favs.favorites.push({...book});
                }
            }
        }
    }
</script>
```

Code Listing 6-k: Finished App (server\api\books.js)

```
import { initializeApp, getApps, cert } from 'firebase-admin/app';
import { getFirestore } from 'firebase-admin/firestore';

const fb_apps= getApps();

if (fb_apps.length <= 0) {
    initializeApp({
        credential:
            cert('
              ./booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json'),
    });
}

const func = async(rq, rs) => {
    const fs_db = getFirestore();
    const snapshot = await fs_db.collection('favbooks').get();
    const books = snapshot.docs.map(doc => {
        return {
            ...doc.data(),
            uuid: doc.id
        }
    });

    return books;
}

export default func;
```

Note: Keep in mind that your generated key file (in my case: booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json) may not have the same name as mine.

Code Listing 6-l: Finished App (.gitignore—root folder)

```
node_modules
*.log
.nuxt
nuxt.d.ts
.output
.env
booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json
```

Note: Bear in mind that your generated key file (in my case: booksfavlist-firebase-adminsdk-eel0o-0a9b2da302.json) may not have the same name as mine.

Code Listing 6-m: Finished App (app.vue—root folder)

```
<template>
  <div class="container-fluid">
    <NavBar />
    <NuxtPage />
  </div>
</template>
```

Code Listing 6-n: Finished App (booksfavlist-firebase-adminsdk.json—root folder)

```
{
  "type": "service_account",
  "project_id": "booksfavlist",
  "private_key_id": "0a9b2da302f6...",
  "private_key": "-----BEGIN PRIVATE KEY\n-----END PRIVATE KEY-----\n",
  "client_email": "firebase-adminsdk-eel0o@booksfavlist...",
  "client_id": "10896908522...",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url":
  "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url":
  "https://www.googleapis.com/robot/v1/metadata/x509/..."
}
```

Note: For obvious security reasons, I have removed some of the Firebase details that are linked to my account. As this is the generated key file, your details will differ from mine, in any case.

Code Listing 6-o: Finished App (nuxt.config.ts—root folder)

```
import { defineNuxtConfig } from 'nuxt3'

// https://v3.nuxtjs.org/docs/directory-structure/nuxt.config
export default defineNuxtConfig({
    meta: {
        link: [
            {
                rel: 'stylesheet',
                href:
'https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css'
            }
        ],
        script: [
            {
                type: 'text/javascript',
                src:
'https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.
js'
            }
        ]
    }
})
```

Code Listing 6-p: Finished App (package.json—root folder)

```
{
  "private": true,
  "scripts": {
    "dev": "nuxi dev",
```

```json
    "build": "nuxi build",
    "start": "node .output/server/index.mjs"
  },
  "devDependencies": {
    "nuxt3": "latest"
  },
  "dependencies": {
    "@nuxt/nitro": "^0.10.0",
    "firebase-admin": "^10.0.2",
    "mdb-ui-kit": "^3.10.2"
  }
}
```

Code Listing 6-q: Finished App (README.md—root folder)

~~~markdown
# Nuxt 3 Minimal Starter

We recommend to look at the [documentation](https://v3.nuxtjs.org).

## Setup

Make sure to install the dependencies

```bash
yarn install
```

## Development

Start the dev server on http://localhost:3000

```bash
yarn dev
```

## Production

Build the application for production:

```bash
yarn build
```
~~~

```
Checkout the [deployment
documentation](https://v3.nuxtjs.org/docs/deployment).
```

Code Listing 6-r: Finished App (tsconfig.json—root folder)

```
{
  // https://v3.nuxtjs.org/concepts/typescript
  "extends": "./.nuxt/tsconfig.json"
}
```

Note: Remember to execute the *npm install* ***command from the root folder of your project to install any (missing) dependencies.***

Next steps and final thoughts

Before closing off, I'd like to leave you with some challenges and thoughts. As some possible next steps in your Nuxt.js learning journey, two things come to mind that would be interesting for you to investigate and implement in your spare time.

The first would be to add additional pages to this application and explore the file system routing mechanism with Nuxt.js—you could add an About page, for example.

Second, and even more interesting, would be to add functionality to save the list of favorite books to the Cloud Firestore database. If you get around to implementing this feature, I'd love to hear from you.

I think Nuxt.js is a fantastic toolkit that allows Vue.js developers to code faster by worrying less about what happens under the hood and focusing more on the business aspect of the app.

If you have worked with Vue.js, I think you'll love Nuxt.js, as it will make you more productive. I hope you have enjoyed reading this book as much as I have enjoyed writing it.

Until next time—stay strong, focused, positive, and always learning. Thank you for reading and following along. All the best.