

VuePress Quick Start Guide

Build blazing-fast static websites with the power of Vue.js



Packt>

www.packt.com

Sufyan bin Uzayr

VuePress Quick Start Guide

Build blazing-fast static websites with the power of Vue.js

Sufyan bin Uzayr



BIRMINGHAM - MUMBAI

VuePress Quick Start Guide

Copyright © 2018 Packt Publishing All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Commissioning Editor: Kunal Chaudhari

Acquisition Editor: Reshma Raman

Content Development Editor: Roshan Kumar

Technical Editor: Shweta Jadhav

Copy Editor: Safis Editing

Project Coordinator: Hardik Bhide

Proofreader: Safis Editing

Indexer: Mariammal Chettiyar

Graphics: Alishon Mendonsa

Production Coordinator: Shraddha Falebhai First published: September 2018

Production reference: 1270918

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-78953-575-4

www.packtpub.com

For my father

– Sufyan bin Uzayr



mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

Packt.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.packt.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the author

Sufyan bin Uzayr is a writer, teacher, and developer with 10+ years' experience in the industry. He is an open source enthusiast and specializes in a wide variety of technologies. He holds four masters' degrees and has authored multiple books.

Sufyan is an avid writer. He regularly writes about topics related to coding, tech, politics, and sports. He is a regular columnist for various publications and magazines.

Sufyan is the CEO of Parakozm, a software development company catering to a global clientele. He is also the CTO at Samurai Servers, a web server management company focusing mainly on enterprise-scale audiences.

In his spare time, Sufyan teaches coding and English to young students.

Learn more about his works at *sufyanism (dot) com*.

There are many people who deserve to be on this page. I am especially grateful to Faisal Fareed and Sadaf Fareed, my siblings, for helping with things back home, as well as Mathew, Bogdan, and Bilal, for reviewing this book. The team at Packt, Douglas P, Reshma R, Roshan K, Shweta J, Hardik B, and others, for ensuring that the book remains perfect throughout. Everyone at Parakozm, and the VuePress community, for their help and support.

About the reviewers

Mathew Rooney is a coder with multiple years of experience in the web development industry. He works with PHP, JavaScript, and offers custom-coded WordPress themes and plugins.

Mathew is a firm believer in open source software and has been working as a developer for nearly 7 years now.

Mathew Rooney is also an expert in creating native web apps in React and VueJS. He works on integration and testing of JS apps and has recently been experimenting with Vue.js integration in existing web projects.

I would like to thank my family and friends for their never-ending support. I am also thankful to Packt Publishing and the Vue community, without whom none of this would be possible. I'd also like to thank Sufyan bin Uzayr, author of the book, for putting together a very helpful guide about VuePress and static sites; as well as Hardik Bhinde for giving me the opportunity to review this amazing book.

Bilal Shahid is a full-stack developer with several years of experience with JavaScript MVCs, Java Spring framework, and PHP along with CMS development, including WordPress and Shopify. He is passionate about the open source movement, and his interests include eCommerce, the real-time web, and code optimization. He also speaks at developer meetups and workshops, and teaches web development through his online courses and tutorials. In his spare time, he likes to read books on topics including history, biographies, and politics. He is currently serving as a frontend architect at one of the major global online retailers, Utopia Deals, where he builds solutions for effective online retailing.

Bogdan Balc, a Tech Lead and Architect with a passion for UI technologies, has worked with JavaScript for the past 12 years, from the emergence of jQuery and Ajax to state-of-the-art solutions such as React, Angular, Ember.js, and Vue.js. When not evaluating the latest technologies and paradigms in the UI world, he spends his time watching sports or movies with friends and family.

Nowadays he channels most of his efforts into making we3interactive one of the most successful and creative startups in Cluj.

His passion for Vue.js pushed him to collaborate on great books such as *Learning Vue.js*, *Vue.js 2 Cookbook*, and *Vue.js by Example*.

I would like to thank my family and friends for supporting me in putting the time and effort into helping develop this book into a great resource for anyone wanting to start building websites with Static Site Generators, in this case VuePress.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

[Title Page](#)

[Copyright and Credits](#)

[VuePress Quick Start Guide](#)

[Dedication](#)

[Packt Upsell](#)

[Why subscribe?](#)

[Packt.com](#)

[Contributors](#)

[About the author](#)

[About the reviewers](#)

[Packt is searching for authors like you](#)

[Preface](#)

[Who this book is for](#)

What this book covers

To get the most out of this book

Download the example code files

Download the color images

Conventions used

Get in touch

Reviews

1. Introducing Static Site Generators and VuePress

What exactly is Vue.js?

What are static site generators?

Comparing static site generators to dynamic ones – differences in methodology

Advantages of static site generators

Disadvantages of static site generators

Some major static site generators

Jekyll

Hugo

Gatsby

Hexo

Nuxt.js

Why should you use VuePress?

A primer to getting started with VuePress

Summary

2. Getting Started with VuePress

Getting started with Vue.js and VuePress

What is Vue.js?

Some basic Vue.js terminology

Why not some other JS framework?

Angular/AngularJS

React

Everything else

How is Vue.js related to VuePress?

Understanding what makes VuePress special

How does VuePress work?

What can VuePress do?

What about the alternatives?

Getting started with VuePress

Installation

Installing VuePress globally

Installing VuePress within an existing project

Understanding VuePress configuration

Other configurations

Summary

3. VuePress Development – Preliminary Steps

VuePress development – getting started

VuePress configuration values

Basic config values for VuePress

Title

Description

Head

ga

Locales

shouldPrefetch

serviceWorker

dest

port

host

Browser compatibility-related config values

evergreen

Build pipeline-related config values

postcss

scss

stylus

less

sass

chainWebpack

configureWebpack

What is asset handling?

URLs in VuePress

Handling public files in VuePress

What about relative URLs?

Summary

4. Creating a Site in VuePress

Building a site in VuePress

What will we be building?

Creating a VuePress site

The README.md file

Starting the dev engine

Building the home page

Adding metadata

Adding other content pages

Adding the navigation menu

Final build

Summary

5. Using Markdown in VuePress

Learning Markdown for usage in VuePress

What is Markdown?

An example

The markdown-it parser

Markdown config reference

`markdown.slugify`

`markdown.externalLinks`

`markdown.config`

`markdown.toc`

`markdown.anchor`

`markdown.lineNumbers`

Markdown extensions in VuePress

Header anchors

Links

External links

Internal links

Emoji

Adding tables

Front matter

Custom formatting options

Highlighting in code

Customized containers

Importing code snippets

Summary

6. Theme Development in VuePress

Theme development in VuePress

Working with themes in VuePress

What and how?

Config entities for theming

theme

themeConfig

Default theme config in VuePress

Home page

Custom layouts for pages

Nav bar

Nav bar links

Search bar

Native search box

Using Algolia Search

How to disable the navbar?

Previous and next links

The sidebar

Modifying header links in the sidebar

Displaying the header links of every page

Nested header links

Active header links

Organizing the sidebar

Using more than one sidebar

Adding sidebars to single pages

How to disable the sidebar?

For GitHub users

Syncing to a GitHub repository

How to hide the Edit link on some pages?

Timestamps from GitHub

CSS in VuePress theme development

Custom classes for pages

CSS override file structure

Backwards compatibility

Ejecting the theme

Custom theme development in VuePress

Customizing the default theme

Working with metadata

Other possible enhancements

Application-level enhancements

Summary

7. Doing More with VuePress!

VuePress: internationalization and doing more with it

What is internationalization?

Internationalization in VuePress

i18n config in VuePress

i18n config for the default theme

i18n config at the site level

Analyzing VuePress

What's next for VuePress?

Working with VuePress in the long run

Summary

8. Deploying VuePress to the Web

Deploying VuePress – an introduction

Prerequisites for VuePress deployment

Can we use shared hosting to run VuePress?

Getting started with VuePress deployment

Deploying VuePress to GitHub Pages

What is GitHub Pages?

Usage

Using custom domains

Deploying VuePress to Google Firebase

What is Google Firebase?

Usage

Using custom domains

Deploying VuePress to Surge

What is Surge?

Usage

Using custom domains

Deploying VuePress to Heroku

What is Heroku?

Usage

Using custom domains

Deploying VuePress to Netlify

What is Netlify?

Usage

Using custom domains

Deploying VuePress to GitLab Pages

What is GitLab Pages?

Usage

Using custom domains

Summary

Links to remote or cloud platforms

Concluding words

Other Books You May Enjoy

Leave a review - let other readers know what you think

Preface

VuePress describes itself as a *Vue-powered static site generator*. In other words, VuePress is a simple tool that lets you create static sites and **Single Page Applications (SPAs)** within minutes.

Anything and everything that you build in VuePress is search engine-friendly, fully optimized, and mobile-friendly as well as really fast due to the fact that there are no database-driven operations or external engines to work with. You can simply enter your content in Markdown, and VuePress will parse the Markdown files as valid HTML.

But there is a lot more to VuePress than just static site generation! You can customize its appearance, tweak the default minimal theme, and make use of Vue.js expertise to extend the functionality of your site. Considering the fact that Vue.js is rising in popularity rather steadily, VuePress is already standing on the shoulders of a giant and has a lot of potential for growth!

This Quick Start Guide will help you get things rolling with VuePress within no time!

Who this book is for

This book is for anyone who wishes to learn how to build static sites in VuePress. All content is rendered as HTML by VuePress and then executed as an SPA. This means everything works fast and loads in no time.

If you are a JavaScript developer or are looking to specialize in Vue.js, VuePress can prove to be a handy tool for you. This book will help you understand the functioning and methodology of VuePress, so as to help you get accustomed to the way in which Vue.js operates. Furthermore, if you are looking to put together a simple site in Markdown, such as a documentation site for your existing or upcoming projects, VuePress can prove to be the perfect solution. In that case, this Quick Start Guide is all you need to gain mastery over VuePress.

Beyond that, if you are simply looking to learn something about static sites and static site generators, then this book is meant for you too!

What this book covers

[chapter 1](#), *Introduction to Static Site Generators and VuePress*, introduces the readers to static site generators such as Jekyll, Hugo, and Hexo. Before talking about VuePress in particular, this chapter will also address the various advantages and possible disadvantages that are associated with static site generators.

[chapter 2](#), *Getting Started with VuePress*, talks about VuePress at large. This chapter will cover concepts such as a basic understanding of Vue.js, what VuePress is, what it can do (and what it cannot do), as well as how to get things started in VuePress. VuePress installation, both at the global level and as a dependency, is also covered herein.

[chapter 3](#), *VuePress Development – Preliminary Steps*, consists of notes related to VuePress configuration values, both at the basic level and for the build pipeline. Other than that, asset handling and URL structure in VuePress is also included in this chapter.

[chapter 4](#), *Creating a Site in VuePress*, is where we get to actual site development! This chapter consists of a step-by-step tutorial about how to build a static site about coffee in VuePress. The code for the site (both in Markdown and rendered HTML) can be found over at this book's GitHub repo.

[chapter 5](#), *Using Markdown in VuePress*, covers Markdown usage in VuePress. In case you are not familiar with Markdown, this chapter also includes a brief introduction to its syntax and formatting, as well as how to use Markdown in VuePress.

[chapter 6](#), *Theme Development in VuePress*, deals with theme development in VuePress. Topics pertaining to default theme customization, navbar and sidebar settings, Git integration, as well as custom theme development are discussed in this chapter.

[chapter 7](#), *Doing More with VuePress*, starts with a discussion of internationalization in VuePress. It also teaches you how to localize your site in

VuePress, and then goes on to discuss the potential for growth, future roadmap, as well as other factors associated with VuePress.

[chapter 8](#), *Deploying VuePress to the Web*, covers VuePress deployment to remote servers on hosting providers such as Heroku, Netlify, and Surge.sh. A detailed step-by-step approach is followed in this chapter to help you easily deploy your VuePress sites.

To get the most out of this book

The following skills, tools, and utilities can prove useful and help you get the most out of this book:

- **Workstation:** Obviously, in order to actually learn VuePress development, you will need a computer, be it a laptop or desktop. Don't worry if your computer does not have a lot of memory; VuePress is not very heavy on resource usage.
- **Basic Coding Skills:** You will need to have a basic understanding of JavaScript and Vue. In the absence of that, you may still be able to get around VuePress, though it will take you some additional time. However, a working knowledge of Markdown will most certainly be required in order to build a legitimate website in VuePress.
- **Code Editor:** It is recommended to have a good code editor so as to better format your code and files. Visual Studio Code is what the writer generally uses, but you are free to opt for any free or premium editor of your choice.
- **Node.js and npm:** Node.js 8+ is needed to run VuePress. Installation of it is covered in the second chapter of this book.
- **Internet Connection and Web Browser:** Well, isn't this one obvious! You can't build and deploy a website sans a browser and internet.

Download the example code files

You can download the example code files for this book from your account at www.packt.com. If you purchased this book elsewhere, you can visit www.packt.com/support and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at www.packt.com.
2. Select the SUPPORT tab.
3. Click on Code Downloads & Errata.
4. Enter the name of the book in the Search box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at <https://github.com/PacktPublishing/VuePress-Quick-Start-Guide>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: http://www.packtpub.com/sites/default/files/downloads/9781789535754_ColorImages.pdf.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Mount the downloaded `WebStorm-10*.dmg` disk image file as another disk in your system."

A block of code is set as follows:

```
module
  .exports = {
    head: [
      [ ... ]
    ]
  }
```

Any command-line input or output is written as follows: **git add --all**
git commit -m "initial commit"
git push -u origin master

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Select System info from the Administration panel."



Warnings or important notes appear like this.



Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at

`customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packt.com/submit-errata, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit packt.com.

Introducing Static Site Generators and VuePress

In the early days of web development, all web pages were based on HTML and CSS. Everything was static and concepts such as content management systems were unheard of. Dynamic database-driven applications too came in rather late.

However, with the advent of content management systems, such as WordPress and Drupal, web development began to move at a faster pace. Static sites and pages were replaced by dynamic query-driven pages that fetched data from, and wrote data to, a database in the backend. This, of course, allowed for greater control and better structuring of content.

The advantages associated with such dynamic database-driven content management systems are plenty. For a start, you can present and store your content the way you like! You can have multiple user accounts, each with settings of its own, customize both the backend and frontend, and more.

Furthermore, for the end user, a database-driven dynamic website management program can provide easier access and a user-friendly interface.

With that said, what about static site generators? Of late, many new static site generators are rising in popularity and gaining momentum. Do they serve any specific purpose? More importantly, what exactly do we mean when we say *static site generator*?

In the course of this book, we will be learning about VuePress, an exciting and useful tool that can help you to generate amazingly fast and easy-to-operate web pages within minutes. We will explore its various features, abilities, limitations, and dependencies, as well as ways in which we can get the most out of it, such as by means of custom themes, extensions, configuration, and more!

VuePress can still be argued to be relatively new to the world of static site

generators, though it has been around for a short while now. For anyone familiar with **JavaScript (JS)**, Vue.js will definitely not be an unheard-of name. It is a JS framework that is gaining a loyal following with each passing day. Here is a brief overview of Vue.js.

Here are some of the topics that we will cover in this chapter:

- What is Vue.js?
- What are static site generators?
- Advantages and disadvantages of static site generators
- Some major static site generators and how they compare to VuePress
- What is VuePress?
- Why should you use VuePress?

What exactly is Vue.js?

Vue.js is a JS framework that lets you build user interfaces. Unlike many other gigantic JS frameworks, Vue can be used to create content-specific single page web apps. It can be integrated with other libraries and projects to suit your needs.

Discussing Vue.js in detail is obviously beyond the scope of this book. For what it is worth, Vue.js is a JS framework that focuses more on the *view* side of things—it is progressive, open source, and is backed by a loyal community. It can be adapted for rendering your web pages and data exactly the way you wish to!

As we can see, Vue.js is not a minnow in the world of JS frameworks. VuePress thus has a very solid foundation. But even if you are not familiar with VuePress, you will be able to master it as you progress through this book.

However, for folks who are still wondering why something like VuePress might be worth the trouble, and what the deal associated with static site generators such as VuePress is all about, we should first cover some basic information related to such site generators.

What are static site generators?

So, before going any further, let's first understand in brief the concept of static site generators. When we say *VuePress is a static site generator*, what do we mean?



Simply put, a static site generator does what its name suggests – it generates a set of web pages that are static in nature. Thus, once deployed, a static site generator can generate web pages on the fly and with great speed.

Most static site generators work by means of a very simplified file structure. You get a set of HTML files, coupled with some CSS stylesheets for better presentation of content, and that is all. Beyond that, we just have a set of JS files to go along with it. There are no complicated plugins, databases, or additional content management options.

To better understand the mode of operation of static site generators such as VuePress, we can compare them to a database-driven dynamic content management system, say, WordPress.

Comparing static site generators to dynamic ones – differences in methodology

How exactly does WordPress, or any other dynamic database-based CMS, tend to work?

When the user visits a given WordPress website, a request is sent by the browser to the server that the site is hosted on. Thereafter, WordPress analyzes the request, and on the basis of that, it prepares the data. This can mean that the given post is loaded along with its media elements, comments are displayed, metadata for the post is queried, and so on. To fetch such info, WordPress needs to query and pull data from its database server.

Thus, effectively, we are following a three-step approach here:

1. The user or client makes a request to the WordPress server
2. WordPress queries and pulls the required data from its database to complete the request
3. The requested information is presented to the user

Now, what if we were to remove the dependency on the database? What if we were to bypass all unwanted code and methods and simply churn out pages in their static form? Of course, we probably will not be able to showcase complicated operations, such as custom plugin functions, eCommerce integration, and so on. But we will entirely overcome the second step in the previous example, as follows:

1. The user makes a request to the server
2. The CMS presents the static page to the user as per the request

By doing so, we are, in a way, speeding up the entire website's performance. Furthermore, we are also reducing our server load as there are no repeated queries to or from the database.

This is the primary difference between a dynamic content management system and a static site generator. While proponents of either side can argue for or against the other, the basic difference lies in the manner of working.

Advantages of static site generators

Now that we have seen how a tool such as VuePress can be different from one such as WordPress, are there any benefits to using the static site generator workflow? Simply put, what benefits do static site generators such as VuePress have to offer?

To begin with, such static site generators consume far fewer resources in operation. You can run a blog or website built with a generator, such as VuePress, on a server with less memory than what you would require for a dynamic content management system-driven website. Furthermore, since there are no databases or custom data queries to deal with, the disk space and bandwidth consumption too is comparatively lesser.

In fact, it is common knowledge that you can even deploy and run a personal blog using a static site generator in just a few megabytes of space! All other things being constant (videos, images, and blog posts), the lack of databases does create a very light atmosphere here, that is much faster to operate.

Static site generators are almost always faster than dynamic content management systems. In fact, a static site generator, such as VuePress, once called, can run as a single page application. This means there will be no repeated queries or calls to load pages. Everything can load instantly! As you might already be aware, speed and page loading times matter a lot on the internet, as people tend to have shorter attention spans and less patience.

Plus, since Google and other search engines often consider page load times as a ranking factor, static site generators are search engine friendly by default. You would, of course, still need to work with metadata, sitemaps, and so on, but the basic worry of *my site is loading slow* will be dismissed. Generally, a single page application such as VuePress does not even need a separate caching mechanism.

It is a common notion that static websites tend to be more secure than dynamic ones. The reason is very obvious here—a static website does not have hundreds

of lines of PHP (or any other language) code, databases, and other content that can possibly be hacked. You do not have to worry about outdated WordPress or Drupal versions here.

Similarly, there is hardly any threat of a poorly coded or evil plugin or theme ruining your hard work. A static site generator stores most of its data in the form of HTML, CSS, and occasionally JS files. Barring a brute force attack to crack your password, there is very little scope for poor code causing any malware or security threat to your data. Code injections, malicious code hidden in server-side files, corrupted database entries, and so on, are all things of the past here.

For advanced users, static site generators offer greater room for customization and personalization. Don't like a given feature? Wish to tweak something to your liking? The code is all yours to customize, and since there are no absolute requirements of having a database or plugin at your disposal, you can build your website or blog exactly the way you want to! This is the primary reason why many power users tend to prefer static site generators, especially for their personal projects and websites.

On the other hand, it is natural that such a high level of customization may not be everyone's cup of tea. For folks that do not like to customize things much, a static site generator can appear to be a daunting task. But for developers, it's paradise!

Well, now that we have seen the advantages, it is safe to assume that static site generators such as VuePress are fairly capable tools. With the added advantages of speed and security, you can create your websites and pages to your heart's liking and not have to worry about dynamic database requirements, security plugins, code injections, and so on.

Disadvantages of static site generators

Everything has its good and bad aspects, and static site generators are no different.

The biggest drawback that such site generators tend to have is the steep learning curve. Often, you have to tweak your work environment a bit to install site generators. As we will see in the coming chapters, installation generally does go smoothly, but you need to ensure that all dependencies and variables are fully satisfied.

For the beginner-level users, such methodology is often too complicated to use. If you cannot install all the required frameworks and libraries, chances are very high that you may not be able to get the static site generator up and running.

While static site generators are ideal for creating blogs and basic websites, you cannot simply create every other website using them. For e-commerce, complicated database-driven content sites, and other content-aware projects, a static site generator is often not the best choice.

For what it is worth, most static site generators are rather lesser known when compared to WordPress or Drupal. This is especially true for entities such as VuePress. This means that migrating an existing blog or site from another CMS can take some time.

On the other hand, migrating a VuePress blog or site to another CMS can be even more time-consuming. You will almost definitely need to work with some code and tweak a few settings before you can initiate a successful migration. But migrating from WordPress to, say, Drupal or vice versa is generally a breeze due to the easy availability of various free and premium plugins for this task.

This shows that static site generators have their own pros and cons. But for what these generators are meant for—blogging and creation of simple static web projects—they can prove to be near perfect. Furthermore, since the target

audience here includes developers and enthusiasts who know what they are doing, and not end users that need a drag-and-drop solution, static site generators do not need to cater to everyone out there. This allows for a leaner and more concentrated development roadmap down the line.

But now that we have talked a good deal about static site generators, what are our options? Much like anything else, there is a good selection of generators that you can choose from. In this book, we will be focusing entirely on VuePress.

However, we must bear in mind that VuePress is a relatively new platform and it is still under active development. As such, in order to do proper justice to VuePress, we need to at least be aware of some other static site generators too. This will help us better comprehend the importance and usage of VuePress, as well as showcase just why VuePress is special. Instead of going into extreme details about various static site generators, we will simply be mentioning a few of the most popular ones.

Furthermore, while it is practically beyond the scope of this book to do a detailed comparison of multiple static site generators, we will still be stating how VuePress stands in comparison to certain other static site generators, so that developers or readers who are already familiar with a given static site generator (say, Jekyll) can better assess what to expect from VuePress.

Some major static site generators

Now it is time to take a look at some of the major static site generators.

We will be focusing mostly on ones that operate without an external database and are similar to VuePress, either in nature or in functionality.

Jekyll

The first name on our list has been around for quite a while. In fact, Jekyll is becoming something of a common name in the world of static site generators. It is a simple, easy-to-use tool that works right out of the box and serves one simple idea—the ability to convert your plain text files into blogs or websites that function as static sites.

Jekyll does not rely on databases and eliminates most of the unwanted features of dynamic content management systems. You can write your content in Markdown, Liquid, or others. Since everything is churned out in the form of a static page, there are considerable speed boosts too.

Compared to VuePress, Jekyll has one very distinct advantage: it is blog-aware and allows you to migrate your blog easily from other platforms. By blog-aware, we mean that categories, tags, and other blog-specific entities or metadata are well recognized by Jekyll. As we will learn in [Chapter 4, *Creating a Site in VuePress*](#), of this book, VuePress is not fully blog-aware by default, so you need to do some extra work to get the same results in VuePress as you would get out of the box in Jekyll.

That said, Jekyll does not rely on Vue.js and is, in fact, based predominantly on Ruby. For JS coders, tweaking Jekyll is sometimes a challenge.

The Jekyll home page can be found at <https://jekyllrb.com>.

Hugo

Hugo is yet another highly popular and fully open source static site generator. It comes with an Apache license and is based on Go. Thus, for coders working with or familiar with the Go programming language, Hugo is often the de facto choice in the field of static site generators.

But that is not the ultimate USP of Hugo. Unlike many other static site generators, Hugo can also play well with most dynamic APIs—custom content, taxonomies, menus, and more. In this regard, Hugo is more of a hybrid between a static site generator and a dynamic CMS. Furthermore, Hugo comes loaded with its own templates for SEO and website analytics. Also, Hugo has a wide range of custom shortcodes and a large collection of themes that you can choose from to get the most out of your website or blog.

Yet, for someone looking for extreme simplicity and not wanting to work with innumerable shortcodes or APIs, Hugo might seem to be overkill. It is more of a compromise between a beginner-friendly CMS and a developer-friendly site generator. The learning curve is not very difficult either.

The Hugo home page can be found at <https://gohugo.io>.

Gatsby

And now, finally, on to some JS-based site generators!

Gatsby is a static site generator based on JS that uses React for its template engine. The obvious part? Owing to the popularity of React, Gatsby naturally has a good number of loyal users in its community. The mode of working here is simple: Gatsby can pull data from a large number of sources and then process it to produce static web pages that can be hosted on a platform of your choice.

Thus, if you have data in the form of headless CMSs, databases, custom dynamic APIs, JSON, or even dynamic content management systems, such as WordPress or Drupal, you can use Gatsby to work with that data and then export it as static pages. In other words, Gatsby is the ideal tool if you wish to create device-aware progressive web applications and are working with React. Gatsby is like a JS framework that can be used to scale as per the needs of your project. However, in comparison to VuePress, Gatsby is limited in the sense that it relies on GraphQL and React.

The Gatsby home page can be found at <https://www.gatsbyjs.org>.

Hexo

Hexo is a static site generator that primarily prides itself on its ability to create blogs, documentation sites, and other projects wherein frequent updating of content is required. It has its own plugin system with which you can even use plugins from other projects with a similar lineage.

Hexo, too, is based on JS and is fairly close to VuePress in terms of genetics. However, it does not rely entirely on Vue.js for everything. In fact, the biggest and most obvious difference between Hexo and VuePress is that the former's theming architecture is string-based and not derived from Vue.

The Hexo home page can be found at <https://hexo.io>.

Nuxt.js

The last entry on our list, Nuxt.js, shares a lot in common with VuePress. Much like VuePress, Nuxt.js is also based on JS and uses Vue.js for its templating engine. In fact, Nuxt.js is a very strong toolkit in its own right and can pretty much accomplish a good deal of what VuePress claims to do.

So, if Nuxt is the answer to our questions, what is the need for VuePress? The rationale is simple—Nuxt.js is primarily meant for creating applications based on Vue.js, but for generation of static websites and blogs, VuePress is the way to go! As a result, VuePress is ideal for creating content-centric websites and entities, such as blogs, documentation sites, and so on. Nuxt.js generally deals with Vue.js-based applications and not content-centric sites.

The Nuxt.js home page can be found at <https://nuxtjs.org>.

Well, we have now discussed some major static site generators. So, what sets VuePress apart?

Why should you use VuePress?

Based on our brief discussion on other static site generators, it is obvious that VuePress can prove useful in cases where other static site generators may not suffice. For example, consider the following cases.

VuePress is a good pick if you are working with JS. More and more developers are nowadays turning toward JS as it is highly versatile, extendable, and can accomplish big projects with ease. Furthermore, if you are not keen on relying on React or AngularJS as the default framework, Vue.js is a natural choice in terms of prowess and features.

If you do not need to create application-only projects, but are in need of something that can be used to build content-centric websites, VuePress is a worthy option.

Since the theme engine of VuePress is based on Vue itself, it can prove to be a good learning tool if you are just getting started with the Vue.js JS framework.

Alright, so at this stage, we have learned what VuePress is, what a static site generator can and should do, and why should we use it. Now is the perfect time to prepare ourselves for what lies ahead—VuePress development, customization, deployment, and a lot more!

A primer to getting started with VuePress

In the coming chapters, we will be discussing concepts related to VuePress in depth. First, we will start with installation, and then move on to basic customization, tweaking, starting a blog or site, custom themes, and more.

But as you can see, installation is obviously the first step. VuePress does not have a gigantic pool of requirements in terms of disk space, bandwidth, or memory. In fact, you can even run it in a shared hosting environment, assuming your web hosting provider supports the required JS scripts (though on a practical note, currently there aren't many shared hosts out there that can claim to do so).

If you are familiar with JS development, you might already be aware of all you need and how to get it. There are very good chances that you might already have such an environment set up. To help you prepare for what is ahead, here is a basic rundown of everything that you need to run VuePress:



The biggest pre-requisite here is that you need Node.js version 8 or higher. It will come bundled with npm, so just make sure that your environment has the right version of Node.js.

Beyond that, the requirements are simple. For production sites, it is always a good idea to employ some server-side caching. Static site generators such as VuePress, as already pointed out, are quite speedy in their own right and you do not, therefore, need to have a custom-coded caching system.



A server-side caching mechanism can help further boost the performance of your sites.

That is all for now, basically. We will be saving the coding details and other inputs for the coming chapters.

Summary

In this chapter, we have discussed quite a few things. We learned about static site generators, what they are, why we should use them, and the advantages and disadvantages that they have to offer. While we cannot use a static site generator for every single scenario, there are various types of websites, blogs, and other projects wherein such generators can prove useful.

Now, in the world of static site generators, our options are plenty. However, VuePress is a good option for anyone looking to create content-reliant websites and render them as single page applications for faster page loading. VuePress relies on Vue.js—therefore, some knowledge with JS and its frameworks' methodology is essential to help you get the most out of VuePress.

As we progress through this book, we will be learning more about VuePress in detail. In the coming chapter, we will spend some time familiarizing ourselves with Vue.js and what it can do. This will help those readers who are not experts in working with Vue.js.

Thereafter, we will head on to learning about VuePress, starting with its installation, operating environment, setup, configuration, and so on. More importantly, we will also cover security steps so that our production sites are secure and free from risks. See you in the next chapter as we embark on our journey of learning VuePress development!

Getting Started with VuePress

In [Chapter 1](#), *Introducing Static Site Generators and VuePress*, we learned what static site generators are, as well as their possible advantages, and why we should use them. Further, we were also introduced to some major static site generators, as well as Vue.js.

In this chapter, it is time to turn all of our attention toward Vue.js and VuePress. Of course, if you are unfamiliar with the way VuePress works, you might be feeling daunted by it at the moment. But there is nothing to worry about—VuePress is rather easy to set up and even easier to master. As you will soon see, VuePress setup is certainly not rocket science at all.

However, in order to properly set up and install VuePress and make sure all the dependencies are met, we first need to familiarize ourselves with Vue.js for a while. This will help us better comprehend why VuePress requires a given system setup to work, and what should be done to get the most out of this simple and swift site generator.

Therefore, it is time to turn our attention toward Vue.js.

In this chapter, we will look at the following topics:

- Getting started with VuePress
- Basic Vue.js terminology
- Comparing Vue.js with other **JavaScript (JS)** frameworks, such as React or Angular
- Understanding the VuePress methodology
- VuePress installation and basic configuration

Getting started with Vue.js and VuePress

Before we head toward VuePress and its connection to Vue.js, let's have a look at what Vue.js is and how it can be useful to us.

What is Vue.js?

Simply put, Vue.js is an open source JS framework meant mainly for building user interfaces. It comes with an MIT license, which means you can extend its source code and build newer projects using it.

The biggest and most obvious goal of Vue.js is that it aims to be inherently adoptable. So, if you have projects that make use of other JS libraries or frameworks, you do not have to abandon them to use Vue.js—instead, you can choose to use it alongside the other JS libraries or frameworks that you might already be using.



Vue.js can easily power single page applications and also act as a web application framework.

So, what is the big deal behind Vue.js that makes it different from the rest of the pack? The primary motive or inspiration behind Vue.js is to simplify web development. Further, it is mostly non-opinionated, thereby providing for a uniform set of offerings for development.

As such, Vue.js is more geared toward **user interface (UI)** development. You can embed its core components into any existing project, or even make use of build tools and supporting libraries to set up single page applications.

All Vue templates are valid HTML that can be read and parsed by any modern web browser. This means anything you build using Vue can be rendered across all devices and browsers with ease. As you progress with VuePress, you will notice this feature of Vue is a big advantage that can help you avoid a lot of compatibility issues.

Now, how does Vue work? It's simple, actually. Each time you make an update to a JS object, the view is updated by Vue. Such *reactive* nature means that the output is rendered or re-rendered as and when changes are made, rather than compelling or forcing a complete refresh.

Some basic Vue.js terminology

We will bypass all detailed learning about Vue.js functioning here, simply because that is beyond the scope of our book. For a better understanding of Vue.js itself (and not VuePress), you can check out any of the awesome Packt books on the subject. For now, we will only be familiarizing ourselves with a few definitions so that when we use them in VuePress, they will be easier to recollect.

We have already learned that Vue templates follow an HTML-based syntax. Now, each template can be accompanied by *Components*. In simple words, Components are blocks of reusable code that are used to extend the functionality of existing HTML elements.

Similarly, Vue.js comes with its own ways of applying transition effects. These can be done in the form of CSS transitions or JS hooks. In advanced VuePress projects, you may find yourself integrating custom JS animation libraries for transition effects. This, of course, depends on the nature of your project and is not always required for a single page application or a blog.

Why not some other JS framework?

This section primarily pertains to folks who already have an existing background in JS and might be familiar with a JS framework or two. If you are a beginner with JS frameworks and Vue.js is the first framework you're becoming acquainted with, you can skip this portion.

However, if you already are aware of other frameworks, you might be wondering what it is that Vue.js can offer that other frameworks cannot?

Some of the most popular JS frameworks are Angular and React, and will be discussed in the following section.

Angular/AngularJS

Compared to Angular, Vue does not seem to be that popular. For all practical purposes, Angular is the bigger name of the two—it is backed by Google, is really easy to get started with, and is highly popular, which means there is no dearth of literature related to it.

However, Vue has a few benefits of its own. To begin with, Angular is highly opinionated about application structure—this is ideal for beginners, and also for developers who want to follow a specific pattern for app development. However, Vue offers greater flexibility and total control over your projects. This means you have more room for experimentation, as Vue is highly adaptable for a wide range of use cases.

Angular uses two-way data binding, whereas Vue relies on one-way data flow between its components. AngularJS (not Angular), in real-world scenarios, tends to become slower as the application grows in size and magnitude. This is because of multiple watchers between transactions. But neither Vue nor Angular suffer from this drawback.

It is interesting to note that in Vue, there is no one right way to structure your application. This means that you have greater flexibility and control. But, as a downside, it also implies that mobility from Vue to other frameworks (or vice versa) can sometimes be seriously hampered. Angular, on the other hand, has a good list of structuring rules that can enforce standards, but can also seem highly limiting for some users.

This is why the learning curve of AngularJS is very steep. There are various API standards that a developer must bear in mind. For Vue coders, such requirements are fewer in number.

React

React and Vue have a good deal in common. It will be easy to argue either for or against any of these frameworks—while React is meant for scaling and is a good framework by all standards, Vue has less of a steep learning curve for many users.

With that said, there is one area where Vue is currently behind React. React lets you write natively rendered apps for both iOS and Android. This means your application can seamlessly run across multiple platforms. Vue currently is attempting to do the same with Weex, a project that is currently being incubated by the Apache Foundation. See more at <https://weex.apache.org/>.



Weex is still in the testing phase and has not been as popularly adopted as React. This gives React a very clear edge over Vue.

Everything else

Ember, another reputed JS framework, differs from Vue in that in Ember, you need to manually declare dependencies for everything and also wrap it all in objects. In Vue, you can utilize plain JS objects too. Vue is significantly faster and better in terms of performance when compared to Ember, simply because Vue can automatically run batch updates.

Polymer, another JS framework backed by Google, differs from Vue in that the former requires basic polyfills to work, whereas Vue does not require any dependencies or polyfills (unless you are aiming for older web browsers).

Thus, we can see that Vue has several advantages and benefits of its own when compared to other JS frameworks. If you are a React or Angular user, you might already have assessed how Vue is going to be different from others. It is simple, allows for reusability of components, and consumes fewer resources.

How is Vue.js related to VuePress?

VuePress describes itself as *Vue-powered*, and that is exactly what the relationship is between the two. Vue.js powers VuePress, or, in other words, VuePress is based on Vue.js.

This means that all of the aforementioned features that you read about Vue.js can be made use of by VuePress users. Therefore, VuePress can be used to generate single page applications on the fly that render in web browsers very quickly and do not cause a strain on server resources. The fact that Vue.js is the backbone of VuePress means the following:

- VuePress is fast and swift to operate.
- It can load pages and **single page applications (SPAs)** very quickly.
- It does not require a vast amount of memory to operate.

Understanding what makes VuePress special

VuePress generates pre-rendered HTML in a static form once it is run and, after that, it can run as an SPA, so there are no repeated calls to the server for every simple query.

This is what makes VuePress special. Unlike other tools and platforms for website creation, VuePress is not resource-hungry. It is a simple site generator with minimal setup. As we will see later in this chapter, setting up VuePress does not require complex knowledge of rocket science or server management. You can do it within minutes, and even a basic learner of JS can get started with it.

Furthermore, if you are a Markdown user, VuePress can be of special use for you, as its entire project structure is based in Markdown. We will, as a matter of fact, cover Markdown extensions in VuePress in a later chapter of this book so as to help you get better acquainted with it.

All in all, VuePress is what you should use if you wish to make use of the advantages that Vue has to offer. By making judicious use of Vue components, as well as coupling it up with Markdown extensions and the ability for custom templates, VuePress can serve as the perfect, highly customizable, and versatile static site generator for a wide variety of users.

How does VuePress work?

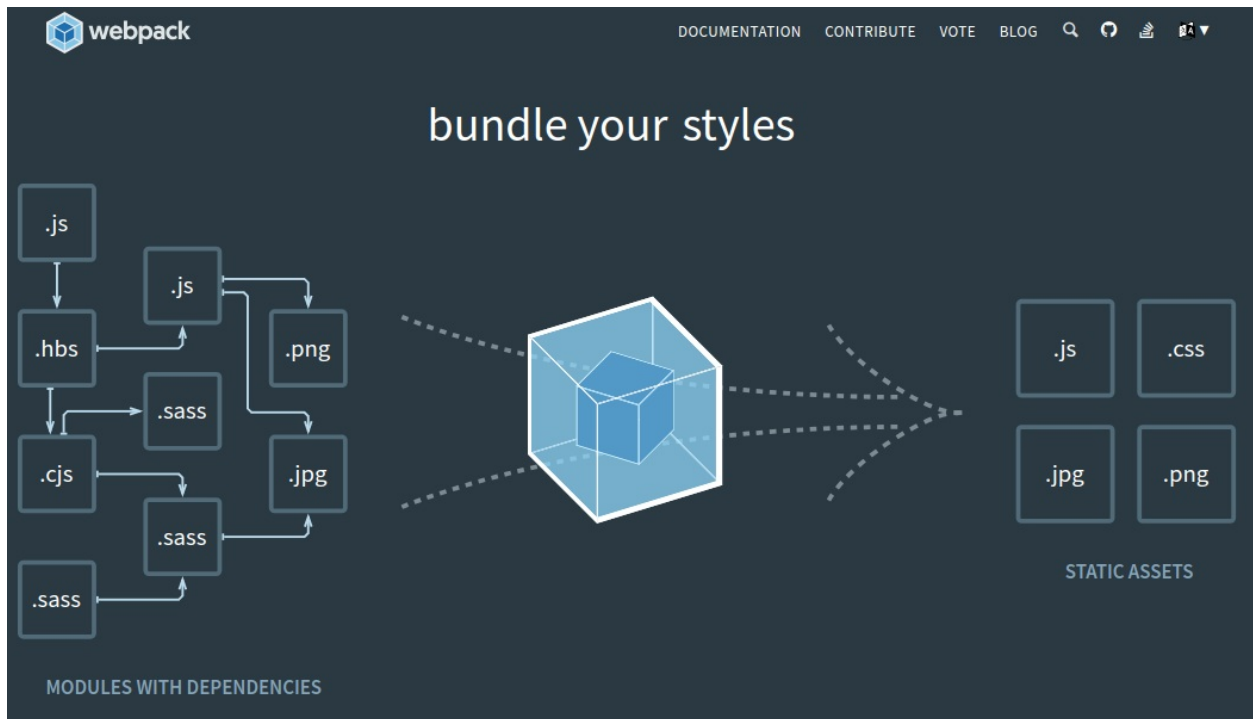
At the very basic level, every VuePress website has two major parts:

- A static site generator that has a Vue-based theming system
- A default VuePress theme that is ideally meant for documentation sites, but can easily be customized for other uses

Now, every page that VuePress generates is in static HTML and is fully search engine optimized. Upon page load, however, the static content is then turned into an SPA for faster performance.

Now, each SPA, as we can see, is powered by Vue. Vue makes use of the Vue router that seamlessly integrates with the Vue.js core and helps making HTML5-friendly SPAs an easy task. More information about vue-router is available at <https://github.com/vuejs/vue-router>.

To finally bundle the scripts and put together the code structure, VuePress relies on webpack. This is a popular tool that can be used to bundle scripts, code, images, and other media into one coherent unit for display in modern web browsers, and is shown here:



You can learn more about webpack at <https://webpack.js.org>.

Since VuePress makes use of Markdown, it relies on Markdown—that is, it compiles Markdown files into HTML. You can also embed dynamic content and make use of code inside your content as markdown—it can handle both. More info about it can be found at <https://markdown-it.github.io>.

Thus, we can see that VuePress makes use of standard tools and services that are all freely available and open source. In other words, you do not have to rely on complex proprietary software for learning VuePress and building your projects using it.

What can VuePress do?

The big question, however, is, what can be accomplished using VuePress?

VuePress comes with built-in Markdown extensions and the ability to leverage Vue inside Markdown. This means it is ideal for people looking to quickly generate websites and put content online. Furthermore, since its theming system is based on Vue, it is a good start for anyone wanting to develop websites using Vue.js. VuePress offers multiple language support and also comes with Google Analytics integration. The default VuePress theme is perfect for documentation websites and projects. It is fully responsive and mobile-friendly, has a native header-based search, and even navigation menus and sidebar. This means that VuePress is perfect for basic website development, web apps, and documentation projects.

However, on the downside, VuePress still does not have perfect blogging support. You can use it for blogging, but it does not come with out-of-the-box features for this. Similarly, while you can call and make use of any Vue.js extension that you feel like, VuePress does not have any plugins of its own.

This means that VuePress is a lightweight site generator meant for the average web development project, and not necessarily meant for bulkier and highly niche-focused projects.

What about the alternatives?

You might be wondering: why should we use VuePress, and not something else?

For a start, VuePress is an exciting new project that is based on Vue and can offer a great learning and development experience.

Unlike many other similar tools, VuePress focuses entirely on static websites and SPAs. For instance, while Nuxt is quite similar to VuePress, the former focuses entirely on building applications. VuePress, on the other hand, comes with features that make it a perfect tool for online documentation projects and other static sites.

Speaking of documentation, solutions such as Docsify and Docute are good picks too, as both are based on Vue and come with features that help you put together technical documentation easily. However, Docsify and Docute are both runtime-driven. It has become common knowledge by now that runtime-driven apps tend to fare poorly in terms of **search engine optimization (SEO)**. VuePress, on the other hand, does very well with SEO too.

Along similar lines, VuePress offers an edge over Hexo too. A lot of people use Hexo (even many Vue.js users tend to rely on Hexo). But Hexo's rendering of Markdown is not as versatile and flexible as that of VuePress, which is a clear advantage that the latter has over the former.

As we can see, VuePress, in spite of being relatively new and comparatively unheard of, has a good set of features that can make it a good choice for a lot of users.

Now that we have read and understood how VuePress works and exactly what it has to offer, we can turn our attention to actually setting things up with VuePress.

Getting started with VuePress

Now it is time to get things rolling. We have by now understood what Vue.js is and the major features it has to offer. Furthermore, we have also learned what VuePress does, how it is related to Vue.js, and what its major uses and features are. At this point, we are ready to get started with VuePress and head toward the creation of awesome web projects with it.

The first step, naturally, will be installation and setup of VuePress, so that it is live, running, and ready for use.

Installation

It must be noted that VuePress requires Node.js version 8 (or higher) to run, so you will have to have Node.js on your server in order to run VuePress.



VuePress requires Node.js 8 or higher on the server side.

There are two popular ways to install VuePress. You can choose to install it globally if VuePress is the only or primary tool that you will be using. Alternatively, if you are using VuePress within a bigger workflow, you may choose to install it as a dependency within an existing project.

Considering the fact that this is a quick-start guide and is meant to help you get familiar with VuePress quickly, using VuePress within a bigger and more complex existing project is technically beyond the scope of this book. Nonetheless, we will cover the commands for both types of installation to help you understand the key difference between the two.

Installing VuePress globally

There are two ways in which you can install VuePress:

- Using npm
- Using Yarn

npm is the default tool that comes with Node.js and most Node.js developers are highly familiar with its abilities. Yarn is also a highly popular dependency management system for JS developers. It works with a wide range of JS frameworks and libraries and is very fast, because it can cache every package that you download—this way, there is no need to re-download anything again.

If you have been actively following JS development, you may already be aware of both npm and Yarn. There are no blatant advantages of using one over the other here, as far as the global installation of VuePress is concerned—for the most part, it is a matter of personal preference. However, it is recommended to avoid using npm when installing VuePress within an existing project.

Here is how the global installation of VuePress is accomplished:

```
# installing vuepress globally
yarn global add vuepress
# creating our readme file
echo '# My VuePress Site' > README.md
# ready to develop
vuepress dev
# building it
vuepress build
```

This example has four simple steps with which you can install VuePress. Nothing else. Let us now analyze each of these steps in greater detail, starting with the first command:

```
| yarn global add vuepress
```

The preceding command will install VuePress globally. Note that it uses Yarn. For npm users, the syntax will look something like the following:

```
| npm install -g vuepress
```

The second command, as follows, will create a Markdown file for us to use:

```
| echo '# My VuePress Site' > README.md
```

The third command is to finally get VuePress started, as follows:

```
| vuepress dev
```

Lastly, the build command is entered as shown here:

```
| vuepress build
```

That's all. You have successfully installed VuePress globally on your server or localhost and can start building your next web project using it. Of course, some basic configuration might be required to further polish your site, but we can turn to this later on.

Installing VuePress within an existing project

It is a good idea to install VuePress as a dependency inside an existing project. Keeping in mind that VuePress is a relatively newer entity and requires decent knowledge of JS development, you may already be running various JS projects of your own. In such cases, having VuePress as a local dependency within your existing project means you can make use of its innumerable features and abilities for your project—say, to render a single page app for your project, or maintain content, and so on.

Installation of VuePress as a local dependency is rather simple. You just need to issue the relevant Yarn command, as follows:

```
| yarn add -D vuepress
```

Or, for npm users, the command will look like this:

```
| npm install -D vuepress
```

Remember that I mentioned previously to avoid using npm when installing VuePress within an existing project? This is because if your project has Webpack 3.x as a dependency already, npm will not be able to generate the correct dependency tree or structure for VuePress. It is worth noting that Webpack is highly popular and is used by several JS projects. As a result, for all practical purposes, when adding VuePress as a local dependency, you should consider using Yarn.



When adding VuePress as a local dependency with webpack 3.x required, you should consider using Yarn.

Following this, you need to create a docs directory as follows:

```
| mkdir docs
```

And then, as previously, we will create the Markdown file for our project:

```
| echo '# My VuePress Site' > README.md
```

That's all!

At this stage, you may wish to add some scripts to your `package.json` file. It is worth noting that the requirements for scripts will obviously depend on your project's purpose and workflow. Some examples are as follows:

```
| {  
  |   "scripts": {  
    |     "docs:dev": "vuepress dev docs",  
    |     "docs:build": "vuepress build docs"  
    |   }  
  | }
```

Now, to write scripts, use the following command:

```
| yarn docs:dev
```

Or, for npm users, the command will look like the following:

```
| npm run docs:dev
```

To generate static assets for your project, enter this command:

```
| yarn docs:build
```

For npm users, use the following command:

```
| npm run docs:build
```

The generated assets and built static files can be deployed to any server. This topic demands a separate discussion of its own and we will revisit it in later chapters as we progress with VuePress deployment.

For now, you should be able to do the following:

- Install VuePress using Yarn or npm
- Install VuePress either globally or as a dependency inside an existing package

Understanding VuePress configuration

So far, we have learned how to install VuePress. However, in order to do anything fruitful with VuePress (or any JS project, for that matter), it is important to first comprehend the configuration structure.

VuePress configuration is fairly simple and straightforward to understand. Upon installation, you will notice a `docs` directory in your VuePress project, which, in turn, will contain the `readme.md` Markdown file that you created during the installation process. Other than that, you might also notice a `package.json` file.

Now, you should create a `.vuepress` directory inside the `docs` directory. This `.vuepress` directory is where your `config.js` file will reside—this file, as any JS developer can tell you, should export a JS object.

For instance, to add a title and description to your VuePress site, your `config.js` file will export the following:

```
module.exports = {  
  title: 'My Fancy Title',  
  description: 'my fancy vuepress site'  
}
```

The preceding code will export a title and description for the final page that is rendered by your VuePress installation.

The following diagram shows how a default VuePress directory structure should look:

```
├─ docs
│   ├─ README.md
│   └─ .vuepress
│       └─ config.js
└─ package.json
```



It is noteworthy that you can use alternative configuration formats as well, rather than solely using `config.js`. For example, you can use `YAML` (which will then have a `config.yml` file) or `TOML` (which uses a `config.toml` file).

However, considering the fact that most VuePress users will have a fair amount of familiarity with `Vue.js`, and will be more comfortable using `config.js` as the default format, we will stick to it throughout the course of this book.

Other configurations

VuePress also allows you to edit theme configurations to personalize your website. VuePress comes with its own default theme that can be tweaked and customized to meet your needs. However, we will turn to theme configuration and tweaks only during [chapter 6, Theme Development in VuePress](#), which deals exclusively with custom theme development.

As we discussed earlier in this chapter, VuePress can be used to generate a single page web app. This means you can use custom hooks to add app-level configurations to your VuePress project. This is useful if you have previous experience with Vue.js and wish to import or make use of existing Vue.js plugins and components from an existing project.

In that case, you will need to add an additional `enhanceApp.js` file to your `.vuepress` directory. In that file, you can then specify custom hooks for adding plugins and registering components. The general syntax of the `.vuepress/enhanceApp.js` file looks as follows:

```
export default ({
  Vue,
  options,
  router,
  siteData
}) => {
  // enhancements come here
}
```

Note the following roles played by each of the elements in the preceding code:

- The `vue` mentions the version of Vue being used by your *VuePress* app
- The `options` specifies the options for the Vue instance
- The `router` specifies options for the router instance
- The `siteData` contains the metadata for your app



Note that app-level enhancements are not required if your app does not need them.

You can find a detailed list of all the configuration references for VuePress in the official documentation itself at <https://vuepress.vuejs.org/config/#basic-config>.

Summary

This brings us to the end of this chapter. We have so far covered all of the basic info related to Vue.js and VuePress. We have also learned what VuePress is, what it can do, what type of projects and applications or websites it can handle, and how it fares when compared to the others in its league. Similarly, we have also familiarized ourselves with some basic knowledge about Vue.js and how it is useful, what it can do, and more.

Beyond that, we have also covered how to install and set up VuePress. We learned how to install VuePress both globally, as well as in the form of a dependency in an existing project. We covered topics related to basic VuePress configuration files.

In the next chapter, we will be turning our attention toward VuePress development. However, instead of custom development at a large scale, we will first be focusing on asset handling and other custom calls. This will enable you to understand and better comprehend how VuePress works, and some coding tweaks you can use to get the most out of it. Thereafter, we will get started with the creation of a live site using VuePress, and then head to theme development.

VuePress Development – Preliminary Steps

In [chapter 2](#), *Getting Started with VuePress*, we covered VuePress installation and setup. At this point, you have now learned how to install VuePress, what its system requirements are, and how to set things up so as to get VuePress ready for actual work.

However, installation of VuePress is just the first step. In order to build a fully-functional project using it, and to get the most out of it, you need to learn how to do more with VuePress.

This chapter will help you accomplish just that.

In this chapter, you will learn about the following:

- VuePress development basics
- Basic config values for VuePress
- Browser config values for VuePress
- Build pipeline config values for VuePress
- Asset handling in VuePress
- Working with images, public files, and URLs

VuePress development – getting started

In this chapter, you will learn concepts such as asset handling and working with files and URLs in VuePress. Now, since VuePress is a static site generator, things tend to be slightly different from what you might find in, say, WordPress, and other database-driven content management systems.

With that said, considering the fact that in [Chapter 4, *Creating a Site in VuePress*](#), we will actually be setting up a live demo site using VuePress, it is vital that you have a good understanding of how files and URLs are treated in VuePress. The [Chapter 4, *Creating a Site in VuePress*](#), will help you plan the perfect structure for your website or blog.

VuePress configuration values

Before going any further, we need to spend some time to ensure we understand the nature of configuration that VuePress generally supports.

Basic config values for VuePress

First, we need to make ourselves familiar with the basic config values that are used in VuePress. This list is not complete, and you should refer to VuePress documentation for a complete list, which may span multiple pages.

For the sake of clarity and to avoid confusion, we will be discussing theme-related configuration in the chapter where custom themes for VuePress will be discussed. Similarly, we will return to discuss Markdown-related config values in the chapter pertaining to Markdown in VuePress.

Title

This value shows the title for your VuePress site. By default, it is prefixed to all the pages on your site and shown in the navigation bar as well, but you can change this feature:

```
| Type: string  
| Default: undefined
```

Description

This shows the site meta description. Thus, in the page HTML, it is rendered within a `<meta>` tag:

```
| Type: string  
| Default: undefined
```

Head

This includes the additional data that is rendered inside the `<head>` tag when the page is rendered in HTML. Generally, it takes a consolidated form of content enclosed within the `[` and `]` square brackets.

The basic syntax is as follows: module

```
.exports = {  
  head: [  
    [ ... ]  
  ]  
}
```

Naturally, the head config is of the array type and its general default form is `[ ]` (empty array).

ga

This config entity helps you to integrate your VuePress site with Google Analytics. If you are active in the field of web development, you might already be aware of vast resources that Google Analytics can provide for your site. So, you just need to pass your Google Analytics ID to this config value, and you're done.

The `ga` is of the array type, and the default value is undefined, as you need to pass your Google Analytics ID to it.

```
{ [path: string]: Object }
```

shouldPrefetch

Sometimes, some files tend to have a preloading resource hint attached to them. The `shouldPrefetch` function handles such limits and hints.

The `shouldPrefetch` is of the function type, and the default value it takes is of the format `( ) => true`.

Confused as to what limits and hints are we talking about? Well, Vue.js has something called `<link rel="preload">` resource hints. By default, nearly all assets that are to be loaded asynchronously are prefetched so as to speed up the page load times and performance. However, what if you are on a weaker server with limited resources, say, limited bandwidth? Prefetching everything will surely cost you a lot of bandwidth.

In such cases, you can use the `shouldPrefetch` function to control which files have the prefetch hints attached to them, and which ones do not. With this selective option, you can avoid consuming a lot of bandwidth on your server.

serviceWorker

This one is rather interesting. Before going any further, you need to ensure certain conditions are met in order to use the `serviceWorker` config entity:

- The `serviceWorker` can only be registered via HTTPS, so you need to ensure your website is using an SSL certificate
- The `serviceWorker` works only in production mode; that is, on live sites

Now, `serviceWorker` is a boolean type that is turned to false by default. But when you set it to true, VuePress will then automatically generate and register a service worker script. This can be used to help cache the content for offline use.

What is a service worker?



*In **JavaScript (JS)**, a service worker is a special type of script that is generated on the fly and run by the web browser in the background. Its running is conducted differently from the actual web page loading. The goal here is to perform some background functions that are very important, but do not need an actual web page or user configuration.*

Examples of cases where a service worker script is used nowadays are for push notifications in the browser, offline caching in the browser, background sync of data for easier recovery in case of connection failure, and so on.

When working with custom themes, as you shall see in a later chapter, the `serviceWorker` config value also emits the following events at times:

- `sw-ready`
- `sw-cached`
- `sw-updated`
- `sw-offline`
- `sw-error`

The `serviceWorker` option here will only handle the JS `serviceWorker` script. This means it will not do anything of its own accord to generate a single page web app; it depends on you to make use of the `serviceWorker` script to meet the needs of your project as required.

Once again, make sure you set `serviceWorker` to true only when you have SSL activated, as `serviceWorker` scripts do not work on non-HTTPS URLs.

dest

This config variable specifies the output directory for the final build process in VuePress. The default path here is `.vuepress/dist`, but you can change it by specifying any other path of the string type.

port

This is meant to specify the port value for connecting to your development server. Other than localhost, your web hosting provider should be able to tell you with greater accuracy which port you might wish to use here.

The default value is 8080 and you can only use values of the integer type, since port values are specified as integers only.

host

This term specifies the host address that you can use to connect to the server. Once again, your web hosting provider or server manager should have the accurate details related to this field.

It accepts values of the type string, and the default is set to '0.0.0.0' only.

Browser compatibility-related config values

VuePress currently has just one major browser compatibility-related config value that you should be aware of.

evergreen

This config value is of the `boolean` type, and is set to `false` by default.

However, you can set it to `true` if you want your VuePress site to target only evergreen web browsers. When set to `true`, it will disable support for ES5 transpilation and polyfills. This means your website will not work well with older versions of **Internet Explorer (IE)**.

The advantage? The build time will be significantly reduced and things will work faster. Plus, the final build size of your site will be much smaller.

What is an evergreen web browser?

An evergreen web browser is one that is automatically updated to future versions as and when they are released. Thus, the user does not need to download a new version and install it from scratch. For example, if you have IE 8 running and you wish to install IE 9, you need to download the installable file and then run it. On the other hand, modern versions of browsers, such as Chrome or Firefox, can update themselves as and when new updates are released, without you having to run an installer all over again.

This rolling model of web browser updates is referred to as an "evergreen web browser". Most of the newer web browsers are evergreen, whereas the older versions are not.



Build pipeline-related config values

Now, let us turn our attention to build pipeline-related config values. The build pipeline in static content management systems, such as VuePress and Jekyll, refers to the production line; that is, the flow of code from development to production.

At this point, your knowledge of JS and CSS will come in handy. Most of these build pipeline-related config values revolve around webpack. Since studying JS and CSS loaders in depth is beyond the scope of this quick start guide, we will just be referencing the relevant project's GitHub page so that you can learn more as and when required.

```
{ plugins: [require('autoprefixer')] }
```

SCSS

The `scss` provides options for the SCSS loader, that in turn is used to convert SASS to CSS. You can learn more about this loader at <https://github.com/webpack-contrib/sass-loader>.

This config entity accepts values of the object type and the default format is `{ }`.

stylus

The stylus config entity provides options to tweak the Stylus loader for `webpack`. By default, it accepts values of the object type and the syntax is as follows:

```
|{ preferPathResolver: 'webpack' }
```

You can learn more about stylus loader at <https://github.com/shama/stylus-loader>.

less

This term specifies options for the LESS loader that can be used to compile LESS to CSS. A detailed guide is available at <https://github.com/webpack-contrib/less-loader>.

It accepts values of the object type and the default format is { }.

sass

This provides options for the SASS loader that can compile SASS to CSS and load *.sass files. Additional details about this loader can be found at <https://github.com/webpack-contrib/sass-loader>.

It accepts values of the object type and the default format is as follows:

```
|{ indentedSyntax: true }
```

chainWebpack

The `chainWebpack` is a function with an undefined default value.

It can help you modify the internal `webpack` config by making use of Webpack Chain API by Mozilla. This particular API can generate and simplify the process for customization and configuration of webpack. You can learn more about it at GitHub at <https://github.com/mozilla-neutrino/webpack-chain>.

Here is some sample code showing the syntax to use with `chainWebpack` in VuePress: `module.exports = {
 chainWebpack: (config, isServer) => {
 // provide instance and config details here
 }
}`

configureWebpack

As the name suggests, this particular config entity lets you modify the internal configuration of `webpack`. It is, therefore, meant mostly for folks already familiar with `webpack` and its usage.

It makes use of `webpack-merge`, details of which can be found at <https://github.com/survivejs/webpack-merge>.

Generally, `webpack-merge` provides you with a merge function that simply concatenates arrays and merges objects. This results in the creation of new objects that you can work with.

In `VuePress`, `configureWebpack` has the default value of `undefined`. However, you should note that it can be either of the object or function type:

- When the value is an object type, the value is merged into the final configuration by making use of `webpack-merge`, as described previously.
- When the value is a function type, it receives the same configuration details as the first argument (the second argument generally being an `isServer` flag). Thereafter, you can choose to modify the configuration that was passed as an argument directly, or return its details as an object that can be merged.

When using `configureWebpack` as a function type, the syntax is as follows:

```
module.exports = {  
  configureWebpack: (configArgs, isServer) => {  
    if (!isServer) {  
      // modify or work with the configArgs here  
    }  
  }  
}
```

So, there you have it! You have now learned the basic details about all of the required config entities in `VuePress`. Once again, we have omitted the config terms related to theme configuration, as well as `Markdown`, which we shall turn to in subsequent chapters.

However, now you are aware of what terms to expect and where—for example, when working with an image asset that is to be used as the site favicon, you should call or reference it using the head config, and so on.

Now it is time to learn some theory about URLs and asset handling in VuePress. Thereafter, you will be ready to develop your first VuePress site.

What is asset handling?

When we say *asset handling*, what is it that we mean?

In the field of development, especially for the web, assets are anything that your project makes use of or works with. This can include, but is generally not limited to, the following:

- Images
- Video
- Other media files
- Documents, such as PDFs

Naturally, to work with such data files and assets, the CMS needs to implement specialized measures and protocols. In other words, you cannot treat a JPG file like a code file—it is not fully readable by the content management system, but the system must be aware that it is a media asset of the image type and should be treated as such.

In VuePress, the methodology is fairly simple. First, we must understand how URLs are configured in VuePress. Thereafter, we can turn our attention towards publicly viewable files, such as images and graphics.

URLs in VuePress

Generally, you would prefer to keep the live VuePress site at either of the two locations:

- At a root URL
- At a non-root URL

For non-root URLs, you need to specify the `base` option in your `config.js`—yes, the same `config.js` file that we discussed in the [Chapter 2, *Getting Started with VuePress*](#). It lies in the `.vuepress` directory.

Let us assume we have a project at [mysite.example.com](#) and we wish to deploy VuePress in a directory named `vuep`. Thus, our VuePress instance will run at `mysite.example.com/vuep/`.

In this case, the `base` value should be set to `/vuep/`.



Notice that `base` should begin and end with forward slash; that is, `/`.

Now, once you have specified the `base` option, it is automatically prepended to all asset URLs. This means any asset URL that you specify in the `.vuepress/config.js` file will have the `base` value prepended to it.

For now, the theoretical part is all you need to know. Live usage of this aspect will be conducted in the [Chapter 4, *Creating a Site in VuePress*](#), where we will set up a site using VuePress.

Handling public files in VuePress

There are various ways to access files in VuePress. By files, to reiterate, we are referring to media elements and other assets.

You can do so by doing the following:

- Referring to the files in your Markdown components
- Referring to the files in your theme components
- Addressing the files as public files

We will be covering the first two steps in the subsequent chapters.

Let us say that you have an image file that needs to be publicly viewable. By publicly viewable, we mean it should be visible to all visitors of your site, and not just admins—for instance, the site favicon.

All you need to do is put the said image file in the `.vuepress/public` directory. Thereafter, the file will be to the root of the generated directory and be publicly accessible.

You can then refer or specify to it as in the following example:

```
|mysite.example.com/vuep/favfile.png
```

Here, `favfile.png` is the file that you placed in the `/public/` directory.

Simple, isn't it?

Well, what if you ever wish to relocate your VuePress site? Say, change the `/vuep` location? You will have to probably change all the references to URLs made in this fashion.

To prevent that from happening, VuePress provides a fairly simple solution.

You can make use of the `$withBase` helper (built natively in VuePress, no extensions required) that will generate the correct path of its own accord and you will not have to worry about specifying absolute paths that may or may not be changed later on.

Here is an example of how to use it:

```
|
```

This code will make sure the reference sticks to the specified file, irrespective of the base URL at which you are using VuePress. You can make use of it to address your public files, and even employ it within theme components and Markdown files.

What about relative URLs?

In VuePress, much like any other content management system on the modern web, it is highly preferable and advisable to reference all asset elements via relative URLs. For example, `./sample.jpg` can be used in any of the file templates. In this case, the target image will first be processed with URL-loader and file-loader, and then copied on to the required locations for the final static building.

When working with `webpack` modules (and config entities, as mentioned earlier in this chapter in the section on build pipeline config), you can use the `~` prefix. This is pretty much the standard for many JS frameworks, and if you have had any type of experience with any of these JS frameworks, you might already be aware of it. For instance, `~some-directory/sample.jpg` is a good way to reference files with `webpack` aliases.

Earlier in this chapter, we talked about the `configureWebpack` option for configuration. You can easily configure your `webpack` aliases using this option in your `.vuepress/config.js` file.

Here is one such sample:

```
module.exports = {
  configureWebpack: {
    resolve: {
      alias: {
        '@alias': 'custom/path/to/directory/relative-url'
      }
    }
  }
}
```

In the preceding example, you are simply making use of `configureWebpack` to make sure the said alias refers to the custom path that you specified for the desired asset or element to be used in your project. When used in conjunction with other config tools, it can help you fetch assets with ease.

That's all there is for asset handling in VuePress. At this point, you are ready to go ahead with actual site development.

Summary

In this chapter, you have learned about most of the basic and advanced config entities in VuePress. You are aware of the basic terminology, as well as what value might be called where, and what value might be returned by which function.

This, obviously, will assume that you are familiar with basic JS objects, functions, and calls.

Furthermore, you are also aware of asset handling basics in VuePress, how URLs are treated relatively, how media elements might be referenced, and so on.

However, it might appear like information overload at this stage, simply because you have learned quite a good deal of theory, but so far have yet to compile it into production or development. But much like any other software or product that is built atop an existing language and offers a framework rich with configurable options, VuePress too needs to be treated as such.

In [Chapter 4](#), *Creating a Site in VuePress*, we are going to build a live VuePress website. You can use any code editor or IDE of your choice, as well as any platform. Note that we will focus on deployment in a later chapter, so it is advisable to set up a local site for the next chapter. You have already learned how to install VuePress in the previous chapter.

So, when working through the [Chapter 4](#), *Creating a Site in VuePress*, it will be a good idea to refer to this current chapter as and when required. This will help you understand which config options can be used where, and how to avoid some common pitfalls. Thus, as and when we encounter a given piece of configuration in the coming chapter when building a site in VuePress, we will be learning about it practically.

The bulk of the focus will be on working with media files such as images, because you cannot really build an actual live site without any images! Plus, we will also cover some real world concepts, such as integration with services like Google Analytics (notice that we have already learned the syntax for the config

reference in this chapter; see `ga` under the *Basic Config* section earlier in this chapter). This will help you build an actual VuePress site that really works and is not just a fancy sample.

So, with that said, make sure you have mastered and gained a fairly good idea of the terminology covered in this chapter. Following that, let's march onward to building a VuePress site!

Creating a Site in VuePress

In [Chapter 3](#), *VuePress Development – Preliminary Steps*, we learned the basics related to VuePress configuration and how to work with variables and functions. We learned about various values that we can tweak to personalize and set up our VuePress site.

Beyond that, in [Chapter 2](#), *Getting Started with VuePress*, we already installed VuePress on our system. Depending on the requirements that we may face, we can choose to install VuePress either globally, or as a dependency within an existing package.

Irrespective of the method of installation, all that matters currently is that we should have a working installation of VuePress. Plus, now that we have mastered the capabilities of static site generators such as VuePress, and how to configure them, it is high time we started seeing things in action.

In this chapter, therefore, we will be turning our attention toward a practical implementation of VuePress. In other words, we will be building an actual website using VuePress. This will help us better understand the workflow in VuePress and also give us a more thorough understanding of its underlying concepts.

So, without wasting any further time, let us get things rolling!

In this chapter, you will look at the following:

- Building a site in VuePress
- How to create Markdown files and run the VuePress dev engine
- How to build and render static sites in VuePress
- An introduction to `.vuepress/config.js` files

Building a site in VuePress

It is worth noting that at this stage, you should have a running instance of VuePress. If you have not yet installed VuePress, refer to [chapter 2, *Getting Started with VuePress*](#), of this book.

Furthermore, you will need to work with basic Markdown files in order to input content. Most web developers are well aware of Markdown, and a majority of them use it on a regular basis. If, by chance, you do not know what Markdown is, or are not comfortable working with it, refer to [chapter 5, *Using Markdown in VuePress*](#), of this book. That said, we will only be touching the surface of Markdown elements in this chapter, such as paragraphs, headings, and links. So, even a basic awareness about Markdown is more than sufficient for now.

Lastly, while we will actually build a site in this chapter, we will not be deploying it to a live server on the web. For the deployment details, we have set aside a separate chapter of its own. If you wish to deploy your project files as soon as you have built the site, refer to [chapter 8, *Deploying VuePress to the Web*](#), of this book.

Now that we have got the introduction out of the way, it is time to start off with site development in VuePress!

What will we be building?

The big question is: what exactly are we looking to build?

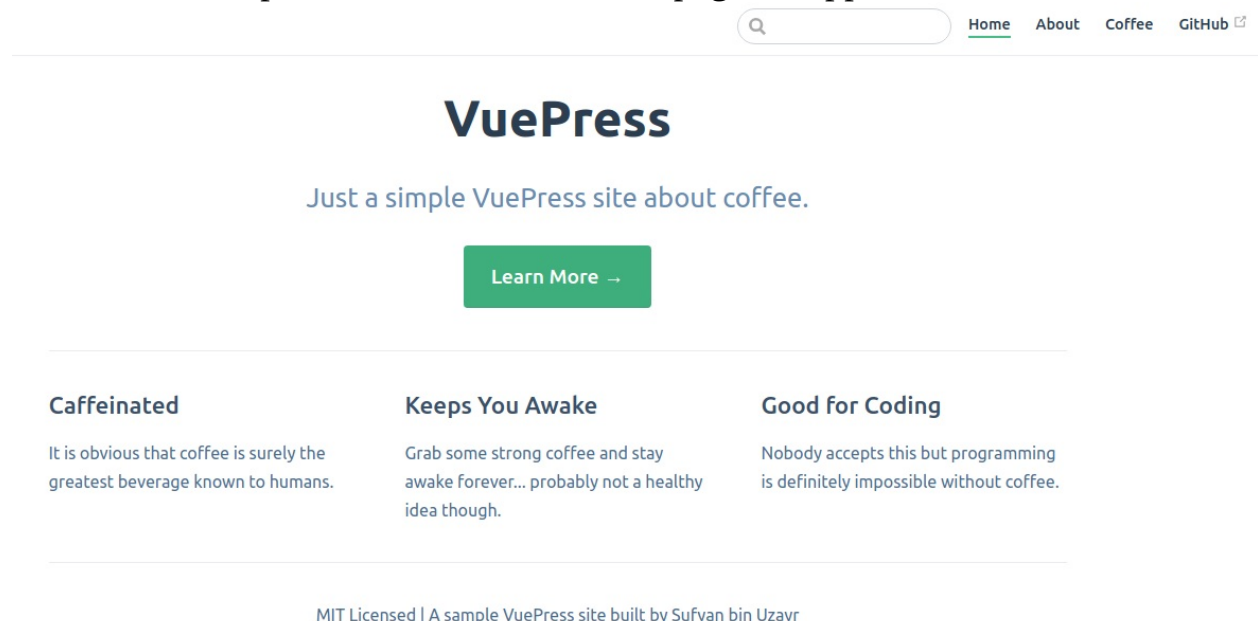
We will be creating a simple website using VuePress. For the sake of the example, let us assume that this given site is about coffee.

Here is how we intend to structure it:

- A home page, detailing what the site is about (and having a call-to-action button)
- An about page
- A sample page, about coffee itself

Furthermore, our site will also have a navigation menu with links to all our pages, as well as an external link. In essence, we are looking to build a multi-page static site in VuePress, with a customized home page view.

Here is an example of what the final home page is supposed to look like:



Creating a VuePress site

Now that we have understood what we are trying to do, let us get started with building our site.

The README.md file

In VuePress, the `README.md` file acts as our home page. Naturally, this is where everything that goes in the `index.html` file should go.

We can edit this file in Markdown to include everything that we require. Beyond that, we can make use of YAML Front Matter syntax to specify custom options for the home page, such as locale, metadata related to SEO, and custom layout.

After we have installed VuePress, we can create a `README.md` file straight away by using the following command:

```
| echo 'Hello World!' > README.md
```

This command will create a simple file in Markdown that contains the line Hello

Hello World!

World! Here is how the output will look:

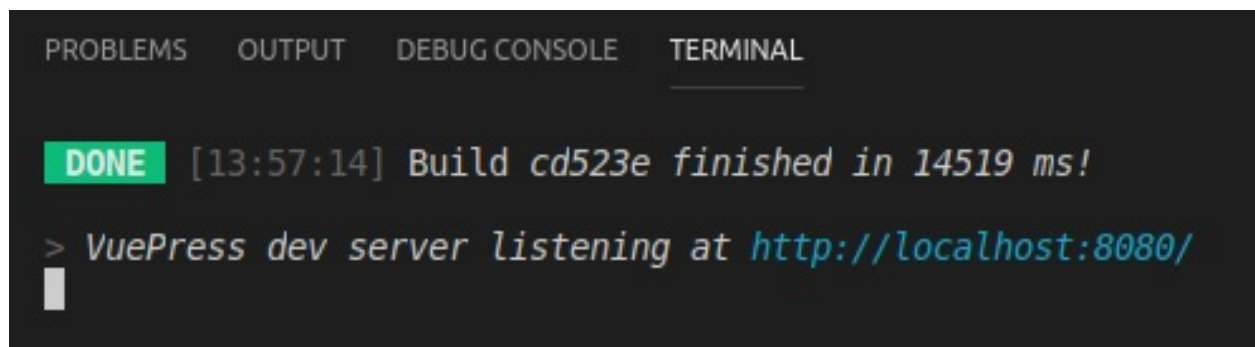
Obviously, this is not sufficient for our home page, is it? We will be building a custom home page shortly.

Starting the dev engine

At this point, it is a good idea to open a Terminal (or command line) and run the following command:

```
| vuepress dev
```

This will start the VuePress dev engine, which will appear like so:

A screenshot of a terminal window with a dark background. At the top, there are four tabs: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL', with 'TERMINAL' being the active tab. The terminal output shows a green 'DONE' box followed by the text '[13:57:14] Build cd523e finished in 14519 ms!'. Below this, a prompt character '>' is followed by the text 'VuePress dev server listening at http://localhost:8080/'. A white cursor is visible at the end of the line.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

DONE [13:57:14] Build cd523e finished in 14519 ms!

> VuePress dev server listening at http://localhost:8080/
|
```

Now we can open the provided URL, as shown in the above screenshot, in our web browser, and as we make changes to our file structure, the page will automatically be refreshed to showcase the live changes the minute they are saved. As long as the dev engine is running (that is, the Terminal is open), then we can view live changes as and when we save our content.

Be sure to run the `vuepress dev` command in the exact directory where you have installed VuePress. All files will be placed and created therein, including Markdown files, and any media elements, as well as any JS configuration files.

Building the home page

Remember the `README.md` file that we just created? Let us open it and edit it a little.



Any editor can be used to edit Markdown files. I personally just use the `xed` or `gedit` default text editors in Linux. You can also choose to use popular code editors for editing Markdown files.

We will be adding our front matter so as to generate a custom home page. We will be making use of YAML front matter. If you are not familiar with it, have no fear—we will be covering it in greater length in the next chapter itself, when we discuss Markdown.

Our home page should have a heading, followed by a call-to-action button to encourage our visitors to learn more about the product. Then, we will also add three separate sections to tell the world more about our offering. Lastly, we will add footer layout.

Now, let us build the home page.

It is really vital to know that the VuePress default theme ships with its own home page layout. All we have to do is specify in the front matter that we wish to make use of the home page layout. To do that, we just need to enter the following:

```
---  
home: true  
---
```

Then, we will add additional entries, such as our call-to-action button text and link, as follows:

```
---  
home: true  
navbar: true  
actionText: Learn More →  
actionLink: /about  
---
```

Now we can save our file to preview the changes. Once again, be reminded that we are making changes to the `README.md` file that we just created.

As we can see in the following screenshot, a call-to-action button has been added that links to an *about* page (not yet created, so clicking on the button will show a 404 error).



Next, we can add the remaining content to our front matter. The following is how the front matter should look now:

```
---
home: true
navbar: true
actionText: Learn More →
actionLink: /about
features:
  - title: Caffeinated
    details: It is obvious that coffee is surely the greatest beverage known to humans.
  - title: Keeps You Awake
    details: Grab some strong coffee and stay awake forever... probably not a healthy ic
  - title: Good for Coding
    details: Nobody accepts this but programming is definitely impossible without coffee
---
```

We have added a features section to our home page. We have added three features, each with its own title and body content.

Our home page, therefore, should become something like the following:

VuePress

Just a simple VuePress site about coffee.

[Learn More →](#)

Caffeinated

It is obvious that coffee is surely the greatest beverage known to humans.

Keeps You Awake

Grab some strong coffee and stay awake forever... probably not a healthy idea though.

Good for Coding

Nobody accepts this but programming is definitely impossible without coffee.

Next, all that is left to do is add the footer. We just need to specify a footer clause in our Front Matter, as follows:

```
---
| footer: MIT Licensed | A sample VuePress site built by Sufyan bin Uzayr
| ---
```

We have chosen the MIT License because VuePress also comes with this license. This, of course, depends entirely on you how you wish to license your content.

After adding the footer, our home page should look something like this:

VuePress

Just a simple VuePress site about coffee.

[Learn More →](#)

Caffeinated

It is obvious that coffee is surely the greatest beverage known to humans.

Keeps You Awake

Grab some strong coffee and stay awake forever... probably not a healthy idea though.

Good for Coding

Nobody accepts this but programming is definitely impossible without coffee.

MIT Licensed | A sample VuePress site built by Sufyan bin Uzayr

Adding metadata

Once we have built the home page, we can add some metadata to the front matter too, as follows: ---

meta:

- name: description

content: Just a simple VuePress site about coffee.

- name: keywords

content: vuepress coffee

lang: en-US

In the preceeding snippet, we have added a meta description for our site, and also specified some keywords. Plus, we have told the engine and crawlers that our site is using US English.

At this point, our `README.md` file should look something like the following:

```
---
meta:
  - name: description
    content: Just a simple VuePress site about coffee.
  - name: keywords
    content: vuepress coffee
lang: en-US
home: true
navbar: true
actionText: Learn More →
actionLink: /about
features:
  - title: Caffeinated
    details: It is obvious that coffee is surely the greatest beverage known to humans.
  - title: Keeps You Awake
    details: Grab some strong coffee and stay awake forever... probably not a healthy idea though.
  - title: Good for Coding
    details: Nobody accepts this but programming is definitely impossible without coffee.
footer: MIT Licensed | A sample VuePress site built by Sufyan bin Uzayr
---
```

It is noticeable that the entire file is composed solely of front matter. We will be learning about YAML front matter in detail in [chapter 5, Using Markdown in VuePress](#), as well as in [chapter 6, Theme Development in VuePress](#), when we discuss theme development.

For now, our home page is ready. We can optionally add any other content to this file in Markdown. Such content needs be added after the front matter, albeit it will be shown above the footer and below the home page hero content.

Adding other content pages

Now our site needs additional pages to showcase further bits of content. Considering the fact that VuePress is a static site generator, we only need to add content accordingly and VuePress does the rest.

Yes, it is that simple!:

1. We add our content as Markdown files
2. VuePress churns HTML files from those Markdown files, as long as the said files are in valid format

That is all. Because VuePress is still not a blogging or e-commerce engine, there are no custom taxonomies to be added, or likewise.

For our site, we will be adding two pages. The first one will be an *about* page, and the second one will be a page about coffee.

For our *about* page, we create an `about.md` file in the same directory as the `README.md` file. Then, we can populate our site with any content that we feel like. Here is the sample that we are using on our site:

```
# Hello World!
This is sample VuePress site about coffee.

## What is This About?
This site has been built as part of the VuePress Quick Start Guide book, authored by, yours truly,
Sufyan bin Uzayr.

In order to understand the site structure, you should look at Chapter Four of the book.

You can find additional information about this site's code at the GitHub page. The link is in the
navbar itself. :)

### More Info
VuePress is a static site generator powered by Vue.js

This site was built to showcase how to develop simple sites using VuePress. You can learn more
about VuePress [here](https://vuepress.vuejs.org/).

If you have any queries, the author can be found [on this page](http://sufyanism.com)
```

And this is what the output should show (be sure to run the `vuepress dev` command

again, once you have added this file):

Hello World!

This is sample VuePress site about coffee.

What is This About?

This site has been built as part of the VuePress Quick Start Guide book, authored by, yours truly, Sufyan bin Uzayr.

In order to understand the site structure, you should look at Chapter Four of the book.

You can find additional information about this site's code at the GitHub page. The link is in the navbar itself. 😊

More Info

VuePress is a static site generator powered by Vue.js

This site was built to showcase how to develop simple sites using VuePress. You can learn more about VuePress [here](#)🔗.

To view the output, simply navigate the web browser to `/about.html`, based on the localhost address that the dev engine shows.

It is noticeable that VuePress takes care of a couple of things all by itself:

- Emoji are shown based on the GitHub emoji service.
- More importantly, every H1, H2, H3, and other similar heading tags in Markdown are treated as anchor links in HTML. You can hover your cursor over any heading, and notice that there is an anchor link automatically generated for it.

Along similar lines, we can add the `coffee.md` file as well. In fact, we can add as many pages as we feel like. All we need to do is to add the Markdown files and inform VuePress about the correct location.

Adding the navigation menu

At this stage, we have added all the pages and content to our site. We have also built a home page, and we can browse and preview everything in the web browser.

But it is obvious that we are simply entering the URL for every page in order to preview them. Real-world users will not be able to guess the file name. Therefore, our site cannot function in the absence of a navigation menu.

It is logical for us to consider adding a navbar to our site. When we study theme development for VuePress in greater detail in [Chapter 6](#), *Theme Development in VuePress*, we will touch every single aspect of navbar customization (including permanently disabling the navbar). But, for now, our priority is to showcase a functional and useful navbar so that our users can navigate through the site with ease.

The good thing is that VuePress supports navbar functionality out of the box. With that said, unless we explicitly add items to it, it will not be shown.

In order to create a navigation menu, we need to use the following code:

```
module.exports = {
  themeConfig: {
    nav: [
      { text: 'Home', link: '/' },
      { text: 'About', link: '/about' },
      { text: 'Coffee', link: '/coffee' },
      { text: 'GitHub', link: 'https://github.com/packtpublishing/vuepress-quick-start-...' }
    ]
  }
}
```

The preceding code exports a navbar array for our theme configuration. Here, we are adding four links to the following pages:

- The home page, located at /
- The about page, located at /about
- The coffee page, located at /coffee
- An external link to our GitHub repository for this book

It is noteworthy that all external links need to be entered in full, with the HTTP/HTTPS protocol at the start of the URL, as seen in this screenshot:

```
JS config.js
1 // .vuepress/config.js
2 module.exports = {
3   themeConfig: {
4     nav: [
5       { text: 'Home', link: '/' },
6       { text: 'About', link: '/about' },
7       { text: 'Coffee', link: '/coffee' },
8       { text: 'GitHub', link: 'https://github.com/packtpublishing/vuepress-quick-start-guide' },
9     ]
10   }
11 }
```

We will need to add this code to our `config.js` file. This particular file is located in the `.vuepress` directory of our VuePress site (if, by chance, the file does not exist, feel free to create a blank file named `config.js` and then enter the preceding code therein). If we place this file outside of the `.vuepress` directory, things may not work as desired.



What is the `config.js` file?

In VuePress, the `config.js` file contains, as the name suggests, all the configuration-related code. This means any function or code that tweaks the VuePress configuration should be added to this file itself. In the absence of this file, VuePress will fall back to default configuration values.

Once we save the `config.js` entries, we can refresh the output and the navbar will be added, as seen at the top of the following screenshot:



Coffee is awesome.

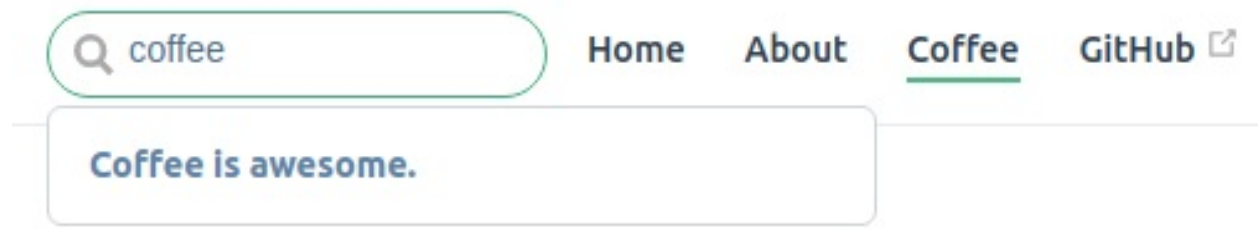
Coffee is a brewed drink prepared from roasted coffee beans, the seeds of berries from certain *Coffea* species. The genus *Coffea* is native to tropical Africa (specifically having its origin in Ethiopia and Sudan) and Madagascar, the Comoros, Mauritius, and Réunion in the Indian Ocean. Coffee plants are now cultivated in over 70 countries, primarily in the equatorial regions of the Americas, Southeast Asia, India, and Africa. The two most commonly grown are *C. arabica* and *C. robusta*. Once ripe, coffee berries are picked, processed, and dried. Dried coffee seeds (referred to as "beans") are roasted to varying degrees, depending on the desired flavor. Roasted beans are ground and then brewed with near-boiling water to produce the beverage known as coffee.

Coffee is darkly colored, bitter, slightly acidic and has a stimulating effect in humans, primarily due to its caffeine content. It is one of the most popular drinks in the world, and it can be prepared and presented in a variety of ways (e.g., espresso, French press, café latte). It is usually served hot, although iced coffee is a popular alternative. Clinical studies indicate that moderate coffee consumption is benign or mildly beneficial in healthy adults, with continuing research on whether long-term consumption lowers the risk of some diseases, although those long-term studies are of generally poor quality.

You may notice that a search bar has been added too. But wait, we didn't actually add the search bar anywhere, did we?

Yes, that's right—VuePress comes with its own search bar functionality that does not need to be tinkered with. It is placed right next to the navigation menu. We can, however, extend or replace the default search method with an external search indexing service.

The search bar shows results right as we type; there is no need to refresh the page. This is the beauty of JavaScript; several other web development languages would have shown the results only after forcing a full page reload.



VuePress native search, unlike that of WordPress and other software, does not search within the content body. Instead, it searches just the headings for the entered term. This is why, in the preceding screenshot, we are just shown the `coffee is awesome` page as the result, whereas the term `coffee` appears on the home page, as well as the *about* page too.

As a result, for a more powerful search experience, using an external service is often advisable. We will learn how to integrate this in [chapter 6](#), *Theme Development in VuePress*, of this book.

Final build

Now we have managed to build our complete site. We have added content as well as separate pages and even created a navigation menu.

So far, we used the `vuepress dev` command to run the dev engine and preview the site. Once we are sure we have built the site, we can safely (and literally) *build* it.

To do so, we need to enter the following command:

```
|vuepress build
```

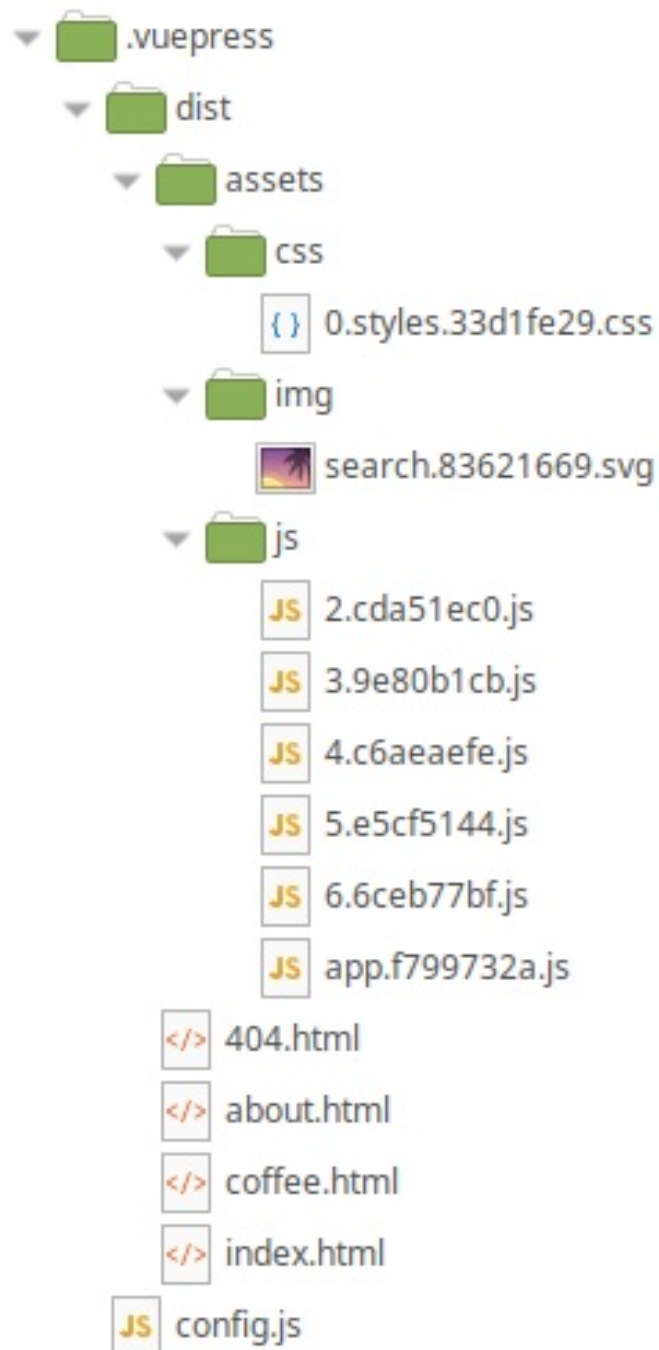
This command will extract our site metadata and compile the info. It will then render static HTML pages based on the Markdown content, as shown here:

```
root@sufyan-Aspire-3:/home/sufyan/mysite# vuepress build

WAIT Extracting site metadata...
[4:51:47 PM] Compiling Client
[4:51:47 PM] Compiling Server
(node:9757) DeprecationWarning: Tapable.plugin is deprecated.
[4:51:53 PM] Compiled Server in 6s
[4:51:57 PM] Compiled Client in 10s
WAIT Rendering static HTML...

DONE Success! Generated static files in .vuepress/dist.
```

The generated content is placed by default in the `dist` directory within the `.vuepress` directory. The general directory structure of our VuePress site, thus, is



as follows:

It is worth pointing out that VuePress generates HTML files based on the Markdown files that we have created. It further places all media elements, JS files, and CSS files in the assets sub-directory at the `.vuepress/dist` path. The `/img` directory consists of images, whereas the `/js` sub-directory consists of, as the name suggests, JS files.

Beyond that, VuePress also creates a `404` file that is displayed in case of *page not found* errors. We can customize this file to suit our needs, or leave it as it is.



Fun fact: The final build of this example site, all zipped together, is approx 68 KB. A similar site of three pages in a dynamic database-driven CMS, such as WordPress or Drupal, would have easily spanned 50+ MB.

That's all! We have successfully built a VuePress site! Easy, wasn't it?

Summary

In this chapter, we learned how to build a static site using VuePress. We learned about custom home page layout, adding a call-to-action button alongside other content, as well as additional pages.

Not just that—we even added a navigation menu to our site. Furthermore, we then learned about the role of `config.js` file. In the coming chapters, we will be learning additional customization-related tweaks. Most of such tweaks will be placed within the `config.js` file.

Lastly, we learned how to *build* a VuePress site using the `vuepress build` command. At this point, we have generated a static site using our own content.



You can find the raw code of this site, including all Markdown files, over at the book's GitHub repository, as a zipped file. You can also download the exported `build` files, including the static content, as another zipped file. The GitHub repository can be found at <https://github.com/packtpublicshing/vuepress-quick-start-guide>.

This particular site can now easily be deployed to any cloud hosting service of our choice. However, we shall turn toward deployment only in the final chapter of this book, wherein we will learn how to deploy our site to various hosting services, such as Heroku, Netlify, and Google Firebase!

For now, we will be digging deeper into Markdown so as to gain better command over content formatting and presentation. We will then be learning a bit about theme development to enable us to customize our website to be just the way we want it.

In the next chapter, we will turn our attention toward content editing in Markdown and how VuePress works with Markdown.

Using Markdown in VuePress

In [chapter 4](#), *Creating a Site in VuePress*, we learned how to set up a site or blog using VuePress. If you've been following the book so far, you should have an active VuePress installation on your local server or machine, with all the variables in place. Plus, you should also be aware of the various config values that you can tweak to get the most out of VuePress.

Beyond that, you also know how to work with pages, how to set up site structure and navigation, apply the default styles, and more.

In other words, you know how to get a VuePress site up and running, as well as how to populate it with content.

But speaking of content, how to manage and add content in VuePress? The easiest and preferred way to get things done in this case is with the use of Markdown.



Markdown is a simple and lightweight markup language that comes with its own formatting and syntax. Owing to its simplicity and ease of use, Markdown has risen in popularity ever since its birth.

In this chapter, we will learn not just what Markdown is, but how to use it in VuePress. We will learn about Markdown syntax, as well as Markdown extensions for VuePress, and Markdown's various config variables.

By the end of this chapter, you should be able to input content using Markdown not only in VuePress, but in any other content management system as well.

In this chapter, you will look at the following topics:

- What is Markdown?
- How is Markdown used?
- Markdown configuration in VuePress
- Markdown extensions for VuePress
- Formatting Markdown content in VuePress

Learning Markdown for usage in VuePress

It is highly advisable to gain some basic skills in Markdown formatting, as it is soon to become the de facto standard in the world of technical documentation, such as in README files for projects in VuePress, or even Vue.js.

What is Markdown?

If you have been active in the field of web development, you might already have come across Markdown, and may very well be using it in your projects. Naturally, owing to its popularity, Markdown does not need a hefty introduction.

With that said, it might be worth the effort to mention Markdown and its advantages in passing, so that even readers new to the topic may find it of some merit.

Simply put, Markdown is a content formatting solution that focuses on ease of use and readability. It was created in the year 2004 by John Gruber, in collaboration with Aaron Swartz. The goal here was to provide plain text content in a more presentable format and, should the need ever arise, optionally convert this into valid HTML.

Owing to its highly simplified and very presentable nature, Markdown gained a good following among the technically capable.



Markdown comes with a BSD-style license, and is currently available for various content management systems in the form of specialized plugins. Even WordPress has several Markdown plugins available.

Additionally, many websites and projects have implemented their own customised variants of Markdown. These users include SourceForge, GitHub, reddit, Stack Exchange, and many others.

An example

Notice that we talked about readability in the preceding section? Well, Markdown focuses heavily on being readable, even in its raw format.

Consider the following piece of simple HTML code:

```
<h1>VuePress Quick Start Guide</h1>
<h2>By: Sufyan bin Uzayr</h2>
<p>VuePress is a static site generator.</p>
<p>It is powered by Vue.js which is an amazing JS framework.</p>
<hr />
<p>Major features:</p>
<ul>
  <li>Easy to use</li>
  <li>Simple and powerful</li>
  <li>Supports Markdown</li>
</ul>
<p>Features again:</p>
<ol>
  <li>Easy to use</li>
  <li>Simple and powerful</li>
  <li>Supports Markdown</li>
</ol>
<p><a href="https://vuepress.vuejs.org/">VuePress Homepage</a></p>
<blockquote>
<p>This is a blockquote.</p>
</blockquote>
```

Now, compare the preceding example with the following code in Markdown:

```
# VuePress Quick Start Guide
## By: Sufyan bin Uzayr
VuePress is a static site generator.
It is powered by Vue.js which is an amazing JS framework.
* * *
Major features:
* Easy to use
* Simple and powerful
* Supports Markdown
Features again:
1. Easy to use
2. Simple and powerful
3. Supports Markdown
[VuePress Homepage](https://vuepress.vuejs.org/)
> This is a blockquote.
```

Both of the previous code samples will give the same output. This will appear as follows:

VuePress Quick Start Guide

By: Sufyan bin Uzayr

VuePress is a static site generator.

It is powered by Vue.js which is an amazing JS framework.

Major features:

- Easy to use
- Simple and powerful
- Supports Markdown

Features again:

1. Easy to use
2. Simple and powerful
3. Supports Markdown

[VuePress Homepage](#)

This is a blockquote.

But, as you can see, the Markdown code is rather clean to look at and easier to read than the HTML. It is not overloaded with cumbersome tags and other elements. Simple formatting aids mean that you can read the text even in its raw format (that is, without the output). This is where the simplicity and beauty of Markdown lies! It is highly readable, easy to grasp, and very easy to type.



You can read more about the features and settings of Markdown on the official website, available at <https://daringfireball.net/projects/markdown>.

Naturally, many static site content management systems, such as VuePress, tend to favor Markdown for these very reasons. As a result, it is a good idea to try to grasp some of the basic formatting aids in Markdown in order to get the most out of VuePress.

The markdown-it parser

Now that we have learned what Markdown is and what advantages it can offer us, it is time to get started on learning about Markdown usage and variations in VuePress. However, before we get to the VuePress environment, we must educate ourselves about a Markdown parser named markdown-it.



What does a Markdown parser do?

Its simplest task is to analyze and break down Markdown statements into syntactical components for easier processing.

markdown-it does its job fairly well and is rather easy to use. It comes with its own set of custom extensions, multiple plugins, and does not slow down your environment at all. Most of the Markdown extensions and config variables in VuePress work well with markdown-it, and some can be used only via markdown-it.

In order to use markdown-it, you first need to install it. It can be done via npm, as is the case with VuePress as well (you have learned about npm installation in [chapter 1, Introducing Static Site Generators and VuePress](#)):

```
| npm install markdown-it --save
```

Following this, you can load its plugins and make use of it as per your needs. To learn more about the markdown-it parser, you can check out its GitHub repository at <https://github.com/markdown-it/markdown-it>.

Alternatively, there is also a working demo available at <https://markdown-it.github.io/>, which can be seen in the following screenshot:

markdown-it demo

☐ html ☐ xhtmlOut ☐ breaks ☒ linkify ☒ typographer ☒ highlight language- ☐ CommonMark strict

clearpermalink

```
---
_Advertisement :)_

- __[pica](https://nodeca.github.io/pica/demo/)__ - high quality and
fast image
  resize in browser.
- __[babelfish](https://github.com/nodeca/babelfish/)__ - developer
friendly
  i18n with plurals support and easy syntax.

You will like those projects!

---

# h1 Heading 8-)
## h2 Heading
### h3 Heading
#### h4 Heading
##### h5 Heading
##### h6 Heading

## Horizontal Rules

```

htmlsourcedebug

Advertisement 😊

- [pica](#) - high quality and fast image resize in browser.
- [babelfish](#) - developer friendly i18n with plurals support and easy syntax.

You will like those projects!

Heading 😎

Heading

Heading

Heading

Heading

Heading

Horizontal Rules



VuePress uses markdown-it as the default Markdown parser or renderer. Naturally, you can make use not just of markdown-it, but also its custom extensions and plugins within VuePress.

In order to further customize markdown-it in VuePress, it might be a good idea to modify your `.vuepress/config.js` file to add the Markdown instance and modify it therein. We have already covered the contents and details of the `.vuepress/config.js` file in previous chapters. Note that you can refer to the markdown-it documentation for further details related to various plugins, and how to reference them for our usage.

Here is how your sample code should look when added to the `config.js` file:

```
module.exports = {
  markdown: {
    // the markdown-it-anchor
    anchor: { permalink: false },
    // the markdown-it-toc extension
    toc: { includeLevel: [1, 2] },
    config: md => {
      // this is where you add more markdown-it extensions
      md.use(require('markdown-it-name-of-extension'))
    }
  }
}
```

In the last part of this code, be sure to add the relevant name of the markdown-it extension that you wish to use. Note that we will be covering the markdown-it-anchor and markdown-it-toc extensions in this chapter itself to help you better understand how to reference them and what output to expect.

Now that you have learned what Markdown is, what the markdown-it parser is, and how VuePress uses it as the default renderer for Markdown, it is safe to turn to the Markdown configuration variables and values in VuePress. Following that, we will turn our attention to VuePress usage and handling of Markdown.

Markdown config reference

At this point, we will first focus on the various configuration-related entities for Markdown in VuePress. Note that these are VuePress-specific and may not hold true for other static site generators.

We have already covered basic config values in [chapter 3, *VuePress Development – Preliminary Steps*](#). Beyond that, we will be turning to theme-specific config values and their uses in the subsequent chapter that deals with custom themes.

For now, we will be focusing only on those values and variables that pertain to Markdown in the VuePress configuration. As you progress with VuePress in your projects, you can make use of these config reference entities to better manage your content in Markdown, and make sure that it is formatted and presented just the way you want it to be.

markdown.slugify

This is a very important function that you should, by all means, be aware of. In VuePress, `markdown.slugify` can be used to transform header text into slugs.

Thus, you can input the header text in Markdown, and then use this function to generate a slug (that is, a URL or permalink) on the basis of that header text. Naturally, this means it is used mostly for anchor links, tables of contents, and other related material.

Since `markdown.slugify` is of the function type, there is no value definition for it in terms of datatypes. Instead, it has a function definition. Let's take a look at its general syntax in closer detail to further understand its functioning:

```
// string.js slugify drops non ascii chars so we have to
// use a custom implementation here
const removeDiacritics = require('diacritics').remove
    // eslint-disable-next-line no-control-regex
    const rControl = /[ 000   01f]/g
    const rSpecial = /[\s~`!@#$%^&*()\_-+=\[ \{ } | \\ ; : " ' < > , . ? ] +/g
module.exports = function slugify (str) {
    return removeDiacritics(str)
    // Remove control characters
    .replace(rControl, ' ')
    // Replace special characters
    .replace(rSpecial, '-')
    // Remove continous separators
    .replace(/\/\{-2,\}/g, '-')
    // Remove prefixing and trailing separators
    .replace(/^\/+|\+$/g, '')
    // ensure it doesn't start with a number (#121)
    .replace(/^\(\\d)/, '_$1')
    // lowercase
    .toLowerCase()
}
```

You can also browse the same source code at <https://github.com/vuejs/vuepress/blob/master/lib/markdown/slugify.js>.

Now, did you notice what this function is doing? It takes the header text, removes its space characters so that they can be replaced by hyphens (-), then further removes any prefixes and trailing characters, and then, lastly, converts the header text to lowercase.

For instance, if our header text is `Header SAMPLE`, it will convert it to `header-sample` for the slug. Naturally, because the slug is part of the URL, it cannot have certain characters and spaces in between. `markdown.slugify` ensures that this is the case.

markdown.externalLinks

The `markdown.externalLinks` is used for adding external links to content (as is obvious from the name). By default, it pairs its values within the `<a>` tags and opens the external links in a new window.

The `markdown.externalLinks` is of the object type and its default syntax is as follows:
{ target: '_blank', rel: 'noopener noreferrer' }

If you wish to not open the external link in a new window (for SEO reasons, or something similar), you can eliminate the `target: '_blank'` part, much like you would in any other HTML document.

markdown.config

markdown.config is of the function type and its default value, therefore, is undefined.

It is used to modify the default configuration, and also add additional functionality or external plugins to the markdown-it Markdown parser that we discussed in the previous section.

The following is a sample demonstrating its use:

```
module.exports = {  
  markdown: {  
    config: md => {  
      md.set({ breaks: true })  
      md.use(require('markdown-it-something'))  
    }  
  }  
}
```

In the preceding example, markdown.config adds the requirement for an external entity named markdown-it-something, and can then be used to implement the added features of the same external entity.

markdown.toc

The `markdown.toc` provides options for adding a table of contents to our website. It is especially useful if you are planning a site that will require such a table of contents, for example, a knowledge base site or the chapter layout of an online novel or book.

It is of the object type and its default syntax is as follows:

```
|{ includeLevel: [2, 3] }
```

It is worth noting that `markdown.toc`, in reality, works with the `markdown-it-table-of-contents` plugin. This particular plugin provides a table of contents layout for the `markdown-it` plugin. You can learn more about it at <https://github.com/Oktavilla/markdown-it-table-of-contents>.

markdown.anchor

The `markdown.anchor` is of the object type and its general syntax is as follows:

```
|{ permalink: true, permalinkBefore: true, permalinkSymbol: '#' }
```

It provides options for adding header anchors to your content.



Note that this should not be confused with header slugs or IDs that are added by `markdown.slugify`, as discussed previously. Instead, `markdown.anchor` lets you add anchor links within your content.

It works in assonance with the `markdown-it-anchor` plugin that adds header anchor functionality to `markdown-it`. You can learn more about this plugin on its GitHub page at <https://github.com/valeriangalliat/markdown-it-anchor>.

markdown.lineNumbers

Whenever you add content such as code blocks to your VuePress site, you can optionally choose whether or not to show line numbers next to code blocks. For that purpose, you can make use of `markdown.lineNumbers` config entity.



The `markdown.lineNumbers` is of the `boolean` type, as it accepts a simple `true` or `false` value for showing (or not showing) line numbers.

For instance, when the `markdown.lineNumbers` config value is set to `true`, code inputs will have line numbers.

Here is how to do it:

```
module.exports = {  
  markdown: {  
    lineNumbers: true  
  }  
}
```

With the config value set as such, any code inputs will have line numbers appended to them. For example, if we were to show the same code in the page as preformatted content, here is how it will appear:

```
1 //Testing line numbers  
2 module.exports = {  
3   markdown: {  
4     lineNumbers: true  
5   }  
6 }
```

Now that we have covered almost all the relevant Markdown config terms, it is time to turn our attention toward Markdown extensions as used in VuePress.

Markdown extensions in VuePress

When we speak of Markdown extensions in VuePress, we are referring to those Markdown extensions that are available for immediate use within VuePress. You have seen how Markdown formatting can be used in everyday usage of VuePress in [chapter 4](#), *Creating a Site in VuePress*.

Header anchors

All headers in VuePress automatically get anchor links applied to them for easier navigation. This is mainly and primarily to facilitate navigation across site content, as VuePress is primarily a documentation management tool and not really a business site solution. Navigating by means of headers is the easiest way to manage content in this case.

If you wish to configure or modify the anchor settings, you can make use of the `markdown.anchor` config option, as discussed in a previous section of this chapter.

Links

In VuePress, links can either be internal or external.

As the name suggests, internal links point to local content, whereas external links point to content of other websites and projects.

Let's discuss each one separately in the following section.

External links

All external and outbound links to third-party websites automatically get the `target="_blank" rel="noopener noreferrer"` tag attached to them. This means that all external links tend to open in new windows (or browser tabs). You can modify this setting using the `markdown.externalLinks` config option, as discussed in the previous section of the chapter.

Internal links

At this point, it is worth remembering that VuePress generates **single page applications (SPAs)** for easier and faster navigation across site content. Now, in order to make SPA navigation possible, all local or inbound links need to be converted to router-links.

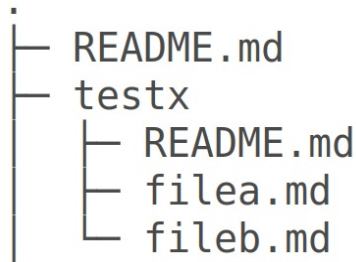
VuePress accomplishes this by converting all internal links ending in Markdown or HTML formats into `<router-link>`.

We learned in the previous chapter that every sub-directory in VuePress, unless it is not publicly accessible, must have a `README.md` file that will, in turn, be converted into an `index.html` file by VuePress before being served in the browser.

Naturally, when adding an internal link to your content, you need to specify the proper file extension within the file path, otherwise you will get a 404 error. Thus, the following considerations must be borne in mind when adding internal or local links in VuePress:

- The `.html` or `.md`—the proper file extension must be appended.
- When writing the relative path to a file or resource, the trailing slash `/` is mandatory. In the absence of the trailing slash, VuePress fails to follow the relative path and will give a 404 error. Thus, `/mypath` is incorrect, whereas `/mypath/` is correct.
- All path specifications are case-sensitive in nature.

Now, to better comprehend these requirements, let's take an example. Consider the following directory structure:



The following table shows how the linking syntax for this structure should work:

RELATIVE PATH	MARKDOWN	EXPLANATION
/	[Home]	The README file located at root folder
/testx/	[testx]	The README file located inside testx sub-directory.
/testx/filea.md	[testx - filea]	The filea.md file inside testx sub-dir; notice the .md extension.
/testx/fileb.html	[testx - fileb]	The fileb.md file inside testx sub-dir; notice the .html extension, it still points to the .md file.

In this example, we can clearly see that the .md extension is automatically converted to parsed HTML by VuePress when rendering the page.

Emoji

Considering the fact that emoji have become rather popular, there should be a way to properly format and include them in your content, should you so desire. WordPress has even integrated its own custom WordPress emoji that you can use in your content, or, alternatively, remove outright and rely instead on browser emoji.

However, unlike WordPress, VuePress focuses more on being lightweight and swift. Therefore, shipping with custom emoji sets in its core is of little sense in VuePress.

However, you can easily make use of the markdown-it-emoji plugin that comes loaded with over 1,000 emoji for you to choose from. Here is a sample:

```
"upside_down_face": "🙄",
"wink": "😉",
"relieved": "😌",
"heart_eyes": "😍",
"kissing_heart": "😘",
"kissing": "😗",
"kissing_smiling_eyes": "😙",
"kissing_closed_eyes": "😚",
"yum": "😋",
"stuck_out_tongue_winking_eye": "😜",
"stuck_out_tongue_closed_eyes": "😝",
"stuck_out_tongue": "😛",
"money_mouth_face": "😏",
"hugs": "🤗",
"nerd_face": "😄",
"sunglasses": "😎",
"clown_face": "🤡",
```

You can choose to go with the full version that offers all the GitHub-support emoji, or the light version that only offers the Unicode emoji. This particular plugin also supports character-based emoticons, such as :-) and others. It works with the markdown-it parser and can be installed via npm as follows: **npm install markdown-it-emoji --save**

Of course, our primary focus here is how to use markdown-it-emoji in VuePress. To learn more about the plugin itself, you can find its GitHub page at <https://github.com/markdown-it/markdown-it-emoji>.

In VuePress, you simply need to specify the emoji you require by typing its name inside two colons. For instance, to add a skull, an alien, a screaming face, and a sunglasses emoji on each successive line, try the following code: :skull:
:alien:
:scream:
:sunglasses:

And the final output in browser will be the following:



Of course, the final actual appearance of the emoji may also vary depending on your operating system and browser version. At times, certain web browsers tend to replace the default web emoji with their own variants.

Adding tables

There are two major types of tables that VuePress supports in Markdown right out of the box.

The first one, obviously, is a table of contents. This is not a pure table, in the sense there are hardly any rows or columns, but is called a table nonetheless, so let's keep it that way. The syntax here is simple, as follows:

```
| [[toc]]
```

This will automatically generate a table of contents on the basis of the headers in your content. You can customize it further by using the `markdown.toc` config option, as described previously in this chapter.

The second type of table in VuePress resembles the table layout on GitHub. In fact, it is called a GitHub style table.

Let's assume we are creating a simple table outlining each country's capital, tourist attraction, popular sport, and currency. You can enter the formatting and details in Markdown as follows:

Country	Capital	Attraction	Currency	Fav Sport	
-----	-----	-----	-----	-----	
United States of America	Washington DC	The Statue of Liberty	US Dollar	Baseball	
Turkey	Ankara	Hagia Sofia	Turkish Lira	Football	
India	New Delhi	Taj Mahal	Indian Rupee	Cricket	

And the output will be as follows:

Country	Capital	Attraction	Currency	Fav Sport
United States of America	Washington DC	The Statue of Liberty	US Dollar	Baseball
Turkey	Ankara	Hagia Sofia	Turkish Lira	Football
India	New Delhi	Taj Mahal	Indian Rupee	Cricket

That's all that there is to tables in VuePress. The tables are automatically formatted, and alternate rows are highlighted, as well, to give the table a more presentable appearance.

Front matter

The concept of front matter by means of YAML data serialization was first popularized by Jekyll, another very popular and versatile static site generator that we discussed briefly in [Chapter 1, *Introducing Static Site Generators and VuePress*](#).

Put simply, front matter contents are placed at the start of the file in between triple dashes. Inside that content, you can specify custom and predefined variables to provide further info about your project. Such variables will then be available to you for use throughout all pages, as well as custom components.

Discussing all of the variables associated with front matter is far beyond the scope of this quick start guide. However, you can check out more info on this topic on the website of the Jekyll project at <https://jekyllrb.com/docs/frontmatter/>.

Now, let's return to VuePress.

In VuePress, YAML front matter is supported out of the box without the use of any custom imports or additional extensions. Thus, whatever you specify between the triple-dashed lines will be available throughout the entire page.

Let's understand it with the help of an example. Consider the following front matter code:

```
---
title: My Fancy VuePress Site
lang: en-US
meta:
  - name: site description
    content: hey, this is vuepress!
  - name: keywords
    content: vuepress blogging vuejs
---
```

Now, in the preceding example, the title and lang variables are specified for the entire page anyway. You can, then, add custom meta tags, such as site description and SEO-related keywords, that will in turn be applicable to the entire page.

Just in case YAML is not your preferred type of scripting solution, you can alternatively opt for other options. TOML is supported too—to use it, you just need to specify that you prefer TOML, as follows:

```
---toml
title: My Fancy VuePress Site
lang: en-US
meta:
  - name: site description
    content: hey, this is vuepress!
  - name: keywords
content: vuepress blogging vuejs
---
```

However, more often than not, many JavaScript coders would prefer JSON. In this case, you need to follow the standard JSON format for front matter; that is, using curly braces and quotes, as shown in the following example:

```
---
{
  "title": "My Fancy VuePress Site",
  "lang": "en-US",
  "meta":
    - "name": "site description",
      "content": "hey, this is vuepress!",
    - "name": "keywords",
      "content": "vuepress" "blogging" "vuejs"
}
---
```

Front matter specification is fairly common in the world of static site generators, and anyone who has had experience of working with one such generator is almost certain to already be aware of the role of front matter.

Custom formatting options

There are some other lesser-used formatting options in Markdown that VuePress natively supports. These will be discussed in the following section.

Highlighting in code

You can optionally choose to highlight lines in code so as to make the output more presentable. Here is some sample code:

```
| export default {  
|   data () {  
|     return {  
|       msg: 'Highlight me!'  
|     }  
|   }  
| }
```

And the output is as follows:

```
1 export default {  
2   data () {  
3     return {  
4       msg: 'Highlight me!'  
5     }  
6   }  
7 }
```

Customized containers

Sometimes, you may wish to put emphasis on certain pieces of your content, such as warnings, notices, and tips. You can easily highlight them in VuePress so that the user's attention is drawn straight to the content at hand.

This means that you first need to specify the given content as a tip, warning, or likewise. Thereafter, VuePress will generate the required notice or tip with a given colored container.

The final actual appearance may vary, depending on your own configuration, but this presentation is fairly comparable to GitHub-flavored Markdown, and if you have some experience of formatting `README` and other files in Markdown in GitHub, you may have already come across such customized content containers.

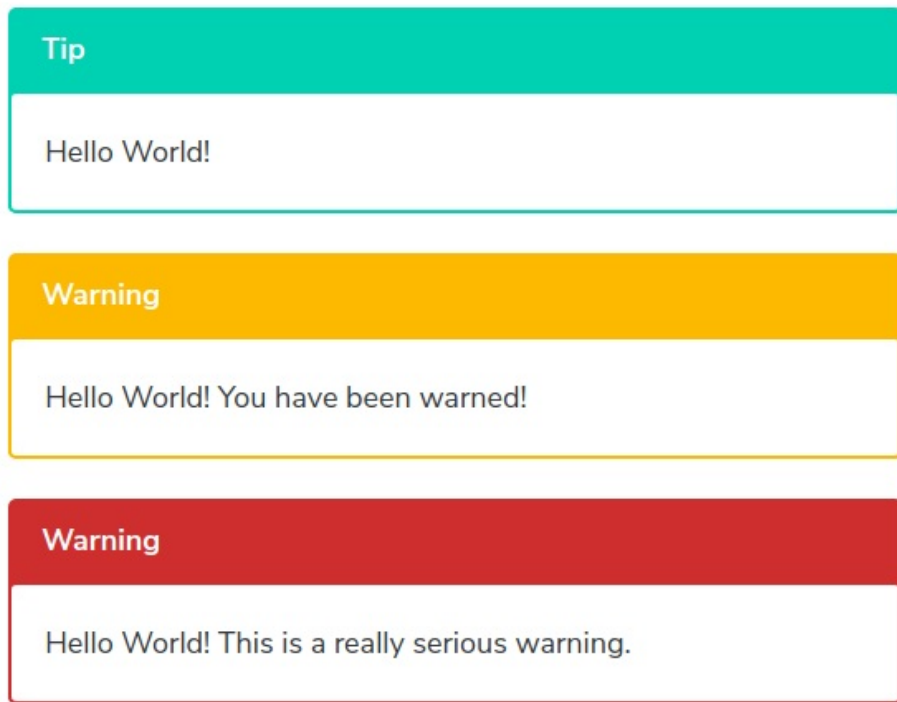
Here is how you can do it:

```
 ::: tip
Hello World!
 :::

 ::: warning
Hello World! You have been warned!
 :::

 ::: danger
Hello World! This is a really serious warning.
 :::
```

And the output appears as follows:



You can, optionally, tweak the title of each container too, as shown here:

```
::: tip HOWDY
Hello World!
:::

::: warning STOP
Hello World! You have been warned!
:::

::: danger NOOO
Hello World! This is a really serious warning.
:::
```

And once again, the output will show the following:

HOWDY

Hello World!

STOP

Hello World! You have been warned!

NOOO

Hello World! This is a really serious warning.

Importing code snippets

If you have existing files containing code, you can import code snippets from them into your VuePress project. The syntax to do this is as follows: <<<
`@/filepath`

However, you need to bear in mind that you cannot use a path alias in webpack, as all imports of code snippets must generally be executed before webpack is compiled.



Remember that importing of code snippets is an experimental feature and is still in beta. It may at times not work as expected.

The code snippets import function supports line highlighting, as discussed earlier in this chapter. In this case, the syntax will become the following: <<<
`@/filepath{highlightLines}`

The default value for `@` will be `process.cwd()`.

Summary

This brings us to the end of this chapter.

In this chapter, we learned what Markdown is and what makes it so popular, such as its simplicity and readability, as well as ease of use. Beyond that, we were introduced to markdown-it parser, a very common and popular entity for rendering Markdown in static site generation tools such as VuePress.

We have also covered VuePress config values for Markdown. Furthermore, we learned about the several Markdown extensions that we have available for use in VuePress. This, of course, is not an exhaustive list. Considering the fact that VuePress is under active development, you can expect newer values and extensions to be added eventually.

With that said, having finished this chapter, you should by now have a good enough understanding of Markdown usage in VuePress and should be able to add `.md` files, format them in the way you want your content to be presented, and save them so that VuePress can render them in HTML and put together an SPA.

Speaking of SPAs, we now need to cover one highly important aspect of web development—templating and theming!

In the previous chapter, we learned about the default VuePress theme and what it can do for us, the layout it offers, and how we can use it to present our content. However, what about custom themes? In the world of today, it is often preferred, and sometimes even required, for websites to have their own individual look and feel. A custom theme can help you accomplish just that and make your website stand out from the crowd!

Thus, in the next chapter, we will be turning our attention toward custom themes in VuePress and covering the relevant topics around this.

Theme Development in VuePress

In the previous chapter, we learned about formatting our content with the help of Markdown. Prior to that, we had already set up a basic VuePress-powered website. Furthermore, we are also aware of most of the configuration settings that VuePress provides.

However, one key aspect of web development is still missing.

Yes, we are talking about theme development. As you might be aware, a good theme or template is an intangible part of any website or blog.

VuePress comes with its own default theme. In [Chapter 4, *Creating a Site in VuePress*](#), when setting up your first VuePress website, you were introduced to the default theme and how to use it to present your content.

In this chapter, however, we will be revisiting everything related to theme development and configuration in the world of VuePress. This means we will not only be covering config entities and variables, but also be revisiting the default VuePress theme and how it works, what specific parts of its code can be used for, and so on.

But beyond that, we will also turn our attention to customization of themes in VuePress. We will talk at length about custom theme configuration toward the latter part of this chapter.

In this chapter, you will learn about the following:

- Theme development for VuePress
- Config entities and values for theme development
- Working with default theme config in VuePress
- Custom theme development in VuePress
- Using CSS customization in VuePress
- Syncing with Git repository

Theme development in VuePress

So now, without wasting any more time, let's get things rolling!

Working with themes in VuePress

VuePress, as a static site generator, focuses on simplicity and agility. This means, unlike WordPress, you will not find a plethora of themes and plugins to use. While it can be a deterrent for some people, as an easy-to-install theme is a real time-saver in some cases, this functional and developer-focused approach is highly useful for many others.

It helps you code your theme the way you want it to be. Plus, you can also work with just the default theme and quickly put together a site live within minutes. In [chapter 4, *Creating a Site in VuePress*](#), we saw that setting up the default home page in VuePress is as easy as editing a simple Markdown file!

What and how?

The current chapter's layout is fairly straightforward, simply because we are dealing with just one particular concept of VuePress—themes. With that said, it is worth remembering that theme development in itself is a fairly broad topic. As VuePress grows in size and popularity, and newer versions come out over the course of time, you may need to brush up and revise your theme development skills in order to ensure that you adhere to the latest coding standards put forth by VuePress and Vue.js.

At this point, we will be discussing, or rather dissecting, the default theme structure for VuePress. This means we will cover home page, navigation, sidebar and all other related entities. You may have learned the basics in [Chapter 4, *Creating a Site in VuePress*](#), but for now, we will be more interested in understanding how and why code works here. You have already seen the practical application of the default theme in [Chapter 4, *Creating a Site in VuePress*](#).

After this, we will turn our attention toward customization. A custom theme in VuePress requires that several changes be made to the default structuring, as we shall soon see.

But before going any further, let's spend a moment trying to cover some of the basic theme configuration values in VuePress. Unlike the Markdown config entities or the basic config entities that we covered in previous chapters, theme config is not that elaborate.

Config entities for theming

At this point, let's first learn about some basic config entities related to theme development in VuePress. We have already learned about other config entities in previous chapters.

theme

The theme config value is used to specify a custom theme. This means that if you are using a custom theme in your project, you need to invoke the theme config to specify it.

It is of the string type and is undefined by default, because VuePress uses its own default theme unless otherwise specified.

themeConfig

As the name suggests, `themeConfig` provides configuration options for the current theme.

It is of the object type and is, by default, written as `{ }`.

Note that `themeConfig` contents and options will vary, depending on the way that you have structured your theme. You will be using the `themeConfig` entity often as you progress with theme customization.

Here is one handy example. Remember the Service Worker that we learned about in [chapter 3, VuePress Development – Preliminary Steps](#)? That was a site-level Service Worker. But what about a theme-level one?

When you use `themeConfig.serviceWorker` you can customize it to work for the theme-level stuff only. In this manner, you can customize several elements of VuePress development to suit the needs of your theme with the help of `themeConfig`.

How is this useful?

Well, imagine the following. Your visitors have your VuePress site opened in multiple tabs and are browsing your content. Now, what if you update your content at that very time? In normal cases, your visitors will not see the new content until all the tabs are closed and then opened – in other words, a hard browser-side refresh or reload of the opened page.

When you configure the Service Worker to work with `themeConfig`, you can also use it for some fun things. One such thing is a UI for popups.

Yes, everyone knows what popups are and how annoying they can be. But we are not talking about an advertisement-like spammy popup. Instead, the `themeConfig.serviceWorker` element will add an `updatePopup` option that will inform the

users when new content is available.

In other words, when you update your site and the tab is opened by the user, a popup will appear informing the user about new content and a button to refresh, so that new content is displayed straightaway, without having to close the clients.



This option is very helpful when you have a site that you update often, such as documentation for a technical project, and so on.

The basic syntax for the `themeConfig.serviceWorker.updatePopup` option is as follows (a Boolean value):

```
module.exports = {  
  themeConfig: {  
    serviceWorker: {  
      updatePopup: true //now popup is set to true and will show up  
      // default display for frontend is:  
      // updatePopup: {  
      //   message: "New content is available.",  
      //   buttonText: "Refresh"  
      // }  
    }  
  }  
}
```



Note that the popup feature in VuePress, as of now, is still in beta.

As you can see, `themeConfig` can be used for varied purposes to add more bells and whistles to your VuePress installation.

For now, that is all. These are the only two configuration values that we need to bear in mind. Thus, it is safe to turn our attention toward the default VuePress theme in order to gain better a functional understanding of this.

Default theme config in VuePress

This particular section pertains entirely and only to the default VuePress theme. If you are using a custom theme, most or all of these settings may not apply.

At this stage, it is highly advisable to turn to the dummy site that you created in [chapter 4](#), *Creating a Site in VuePress*. We covered concepts such as the home page, navigation, sidebar, a search function, and so on. We will now gain a better understanding of these, and also learning about other features that the default VuePress theme provides out of the box (such as integration with GitHub repo).

Home page

We are aware that the default VuePress theme comes with its own home page layout. In order to use this, all we have to do is set the `home:` value to `true` in the root directory's `README.md` file. The said file will then be parsed as `index.html` by VuePress, and the default home page will be shown.

We can add metadata to the home page directly via the `README.md` file. In the previous chapter, we have learned how to write the front matter in YAML format. Thus, it is time to analyze the code for the home page. Copying the same home page front matter from [chapter 4, *Creating a Site in VuePress*](#), gives us the following: ---

meta:

- name: description

content: Just a simple VuePress site about coffee.

- name: keywords

content: vuepress coffee

lang: en-US

home: true

navbar: true

actionText: Learn More →

actionLink: /about

features:

- title: Caffeinated

details: It is obvious that coffee is surely the greatest beverage known to humans.

- title: Keeps You Awake

details: Grab some strong coffee and stay awake forever... probably not a healthy idea though.

- title: Good for Coding

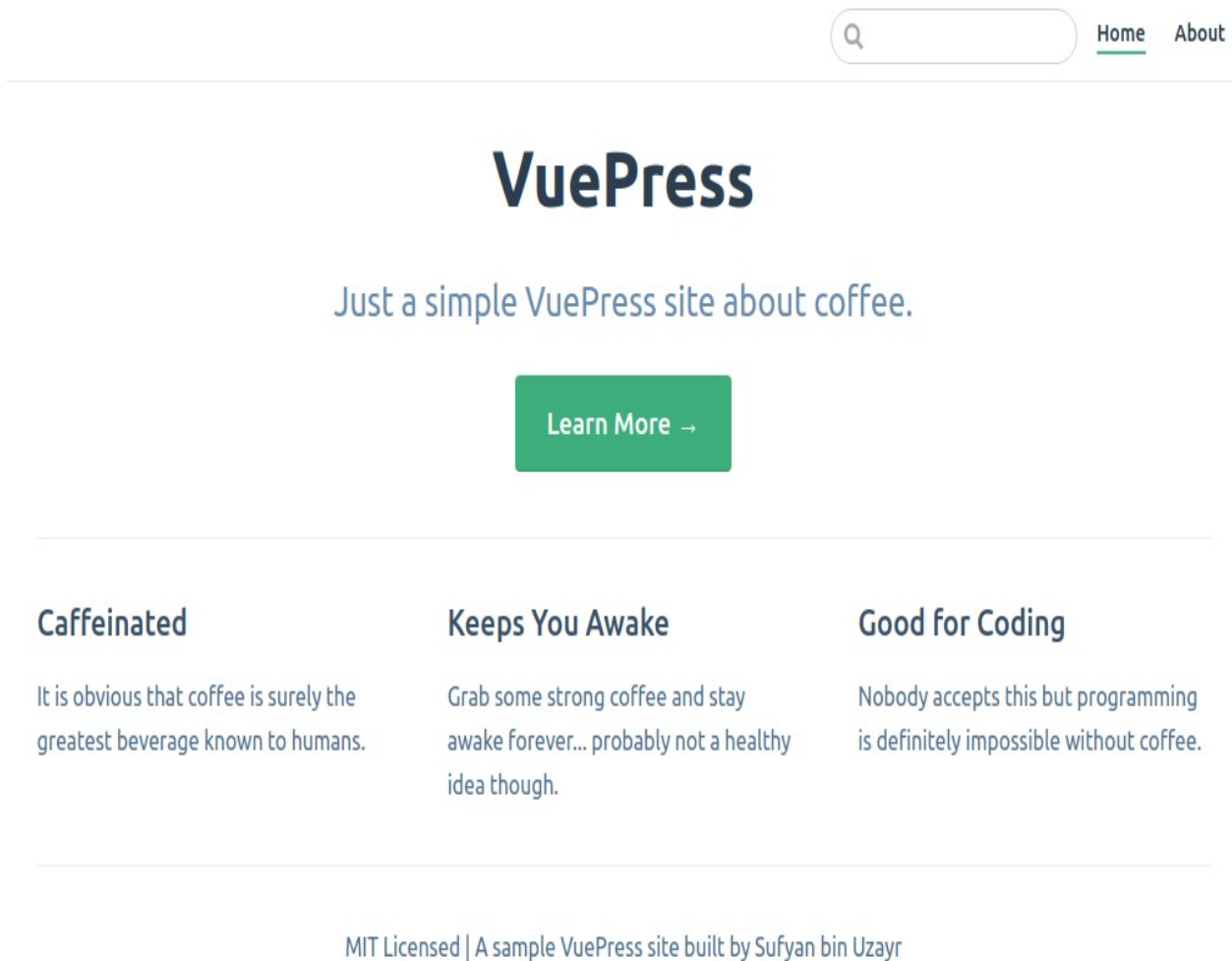
details: Nobody accepts this but programming is definitely impossible without coffee.

footer: MIT Licensed | A sample VuePress site built by Sufyan bin Uzayr

In the preceding code, we first specify the metadata about our site. Then, we add

the link and anchor text details for the call-to-action button. Lastly, we enter additional info to be displayed.

As we saw in [Chapter 4, Creating a Site in VuePress](#), here is how the end result looks:



Anything that is placed after the front matter is also parsed as simplified Markdown, and it will be shown below the front matter on the home page.

But what if you do not wish to use the default home page? Sure, you can create a custom theme, but why should you build a new theme just for adding a new page?

In this case, the better option would be to customize the home page layout. We can do this by means of custom layouts.

Custom layouts for pages

We know that every Markdown `.md` file is rendered in HTML in the web browser by VuePress. In general, VuePress parses the contents of the Markdown file within `<div class="page">` containers in HTML.

This means that our default layout—comprising of the main content, nav bar and other links—is included within the container.

Thus, the task here is simple. If we have to add any custom layout or component, we just need to ensure it is included within our default container.

For example, say we have specified a custom `layout` named `MyLayout`. Now, in the YAML front matter, we just need to add the following line: `layout: MyLayout`

With this, the new layout will be rendered for the given page.

In this manner, you can specify custom layouts for as many pages as you want.

Nav bar

The navigation bar in VuePress consists of a generic set of items, such as the navigation menu, the search bar, a link to your GitHub repo (if any), and so on.

Nav bar links

Remember the `themeConfig` variable that we discussed previously? You can use it to add any amount of links to your navbar.

Let's analyze the following code:

```
// Goes to .vuepress/config.js
module.exports = {
  themeConfig: {
    nav: [
      { text: 'Home', link: '/' },
      { text: 'About', link: '/about' },
      { text: 'Coffee', link: '/coffee' },
      { text: 'External', link: 'http://sufyanism.com' },
    ]
  }
}
```

In the preceding example, we are creating a simple navigation link menu that offers four links:

- Home page
- About page
- A page about coffee
- An external link (GitHub)

Fairly simple, isn't it? You can keep adding links here as per your needs. Here is how it looks in the browser:

[Home](#) [About](#) [Coffee](#) [GitHub](#) 

And what about dropdown menus? All you need to do is provide the said links as arrays, not independent links. For instance, consider the following code:

```
module.exports = {
  themeConfig: {
    nav: [
      {
        text: 'Linkz',
        items: [
```

```

{ text: 'Home', link: '/' },
{ text: 'About', link: '/about/' }
]
}
]
}
}

```

With this, the links will be shown as a drop-down menu.

Furthermore, you can also have highly complicated menus, with sub-groups and nested items under drop-down entries. For a VuePress website, such a type of menu structure is highly unlikely to be used, and potentially not so useful.

Nevertheless, here is the default syntax for such a menu structure, which you can modify as per your requirements: `module.exports = {`

```

themeConfig: {
  nav: [
    {
      text: 'Label',
      items: [
        { text: 'First', items: [/* */] },
        { text: 'Second', items: [/* */] }
      ]
    }
  ]
}
}

```

Search bar

You can choose to use either the native search or Algolia DocSearch in VuePress. The default search bar location is right next to the navigation menu links, as we saw in [Chapter 4](#), *Creating a Site in VuePress*:



Native search box

VuePress comes with its own native search box that you do not need to tinker with. However, if you want, you can use the `themeConfig` value to entirely disable the search box, or limit the number of suggestions that are shown for the search term.

Here is how to disable the native search box:

```
module.exports = {  
  themeConfig: {  
    search: false,  
  }  
}
```

Or, to limit the number of suggestions, try the following code:

```
module.exports = {  
  themeConfig: {  
    searchMaxSuggestions: 5  
  }  
}
```

One big drawback of native search in VuePress is that it builds its index from the `h1`, `h2`, and `h3` tags only. This means it will not scan through the actual body content for the search term.

Naturally, such search functionality may not suffice for everyone, as often, the correct keyword is found in the content and not the headings.

To overcome this, we can integrate an external search mechanism into our theme.

Using Algolia Search

If you are not familiar with Algolia DocSearch, it might be useful to visit their website and learn more at <https://community.algolia.com/docsearch/>.

Basically, Algolia DocSearch is an enhanced version of the documentation search engine. It crawls through your content, indexes it, and then returns better search results that span and crawl the entire content, not just headings. A screenshot of their website is shown here:



The best search experience for docs, integrates in minutes, for free

Providing search for 900+ docs, and counting.

JOIN THE PROGRAM



Algolia DocSearch is fully compatible with various environments, including Bootstrap, React, and of course Vue.js Plus, it is a context-aware search mechanism that learns as you type. As a result, Algolia DocSearch can offer an array of features that may otherwise be lacking in native search methodology, such as:

- Autocomplete suggestions
- Customized analytics to view detailed search stats

Algolia DocSearch comes with an MIT License and is open source. With that said, in order to make use of it on your VuePress website, you will need to register for an account with Algolia.



Detailed documentation for integrating Algolia DocSearch with your site is available at <https://github.com/algolia/docsearch#docsearch-options>.

Once you have signed up for an account with Algolia, you can submit your site for indexing and crawling. Thereafter, you just need to inform VuePress about the Algolia DocSearch engine. To do that, you will once again use the very handy `themeConfig` option, as follows:

```
module.exports = {  
  themeConfig: {  
    algolia: {  
      apiKey: '<API>',  
      indexName: '<INDEX>'  
    }  
  }  
}
```

Note that you will need to specify your own API key and index name, as provided by the Algolia DocSearch when site indexing is complete. In the absence of an API key or index name, or in case of incorrect values for either of them, the search functionality will not work.

How to disable the navbar?

Sometimes, you may need to or wish to disable the navbar entirely. In that case, you can just specify this via the `themeConfig` option, as follows: // Goes to

```
.vuepress/config.js
module.exports = {
  themeConfig: {
    navbar: false
  }
}
```

The navbar display value is now set to false, and it will not therefore show up on the frontend.

However, the preceding option disables navbar globally. To disable it just for a specific page, you should preferably edit the front matter for the said page, as follows: ---

```
navbar: false
---
```

Previous and next links

By default, VuePress will add prev and next navigation links at the end of each page on your site. These links are automatically inferred on the basis of headers in your sidebar, and calculated from the active page.

However, you can override their appearance and selection as well. In the front matter of the active page, you just need to specify your choices. For example, for a custom selection of pages, you would add the following:

```
---
prev: ./another-page
next: ./yet-another-page
---
```

Optionally, you can also disable both of them:

```
---
prev: false
next: false
---
```

Or, just disable one of them with the following code:

```
---
prev: ./some-fancy-page
next: false
---
```

Or:

```
---
prev: false
next: ./my-awesome-page
---
```

And so on.

That covers the part about navigation per se. Now, it is time to turn our attention toward the sidebar.

The sidebar

VuePress provides your site with a native sidebar that is built on the basis of your page structure and the header links therein.

To enable the sidebar for your site, you need to configure it using an array of links via the `themeConfig.sidebar` option. Note that the following code goes into your `config.js` file: // goes to `.vuepress/config.js`

```
module.exports = {  
  themeConfig: {  
    sidebar: [  
      '/',  
      '/page-1',  
      ['/page-2', 'optional link text']  
    ]  
  }  
}
```

In this sample, the links go to the `README.md` files by default. It means any link ending in `/` forward slash will go to the relevant `README.md` file.

Notice the optional link text? If you specify something here, then it will be shown as the link text in the sidebar. Alternatively, if you specify a title in the front matter for the page, then it will be shown in the sidebar as link text. Lastly, if no title is specified in either location, VuePress will automatically pick the link text from the first header of the concerned page.

Modifying header links in the sidebar

You can play with header links in the sidebar of VuePress. A number of examples will be discussed here.

Displaying the header links of every page

By default, the sidebar will only show the header links for the active page. You can change it to show the header links of all pages straightaway. All you need to do is to set the concerned value to `true`, as follows:

```
module.exports = {  
  themeConfig: {  
    displayAllHeaders: true // Default is false  
  }  
}
```

Nested header links

The sidebar, as you are aware, will automatically display the links for headers in the current active page. Thus, as you browse through the pages, the relevant header links will be shown for easier navigation.

Such headers are nested under the active page, with a default depth of 1 (which means all `H2` tags are shown as headers, whereby the page title itself is `H1`).

You can modify this behavior to show deeper nesting of header links using the `themeConfig.sidebarDepth` option. For example, if the depth of nesting is changed to 0, all header links are hidden and only `H1` values (page titles) are shown. Similarly, changing the depth to 2 will imply that all `H3` headings nested under `H2` header links are also shown, and so on.

You can specify the relevant depth in the front matter. For instance, to show `H1` page titles, `H2` header links as well as `H3` and `H4` links, you will specify the depth to be 3, as follows: ---

```
sidebarDepth: 3
```

```
---
```

Active header links

As the user scrolls down the page, the sidebar's active links are automatically updated. This is the default behavior for virtually all static site generators that tend to work as single page apps. In this manner, there is no need to refresh or reload the full page at all.

However, you can disable this behavior should you need to, with the help of `themeConfig.activeHeaderLinks` option, as shown here:

```
module.exports = {  
  themeConfig: {  
    activeHeaderLinks: false, // Default is true  
  }  
}
```

Note that this will mean that the active header link is not highlighted during browsing, and this may impact the overall UX of the site. However, it can also boost page speed slightly, as the corresponding scripts will no longer be loaded.

Organizing the sidebar

You can choose to organize or divide your sidebar into multiple groups. This is accomplished with the help of objects.

Sidebar groups in VuePress are collapsible by default. However, you can specify the collapsible option to be false as well. The syntax for creating sidebar groups is as follows: // goes in .vuepress/config.js

```
module.exports = {  
  themeConfig: {  
    sidebar: [  
      {  
        title: 'A Group',  
        children: [  
          '/',  
        ],  
      },  
      {  
        title: 'B Group',  
        children: [ /* ... */ ]  
      }  
    ]  
  }  
}
```

This syntax is self-explanatory. We are just creating groups of links with the given titles, and then adding pages and links as children to each.

Using more than one sidebar

You can, indeed, have multiple sidebars for different sections of your content.

However, for this to properly work, you first need to organize your content accordingly. A good idea is to organize your pages into directories and sub-directories.

For example, consider the following directory structure wherein the pages are organized as per directories:

Now, to create multiple sidebars, we need to add the following to our `config.js` file:

```
// goes in .vuepress/config.js
module.exports = {
  themeConfig: {
    sidebar: {
      '/testx/': [
        '', /* /testx/readme.md */
        'one', /* /testx/one.md */
      ],

      '/testxy/': [
        '', /* /testxy/readme.md */
        'two' /* /testxy/two.md */
      ],

      // fallback
      '/': [
        '', /* readme.md at / */
        'about' /* /about.md */
      ]
    }
  }
}
```

The preceding configuration will declare sidebars for each section. Note that it is highly recommended to declare the fallback option only at the end, because VuePress reads config for sidebars from top to bottom; that is, chronologically in order of declaration.

```
---<br/> sidebar: auto<br/> ---
```

```
// goes in .vuepress/config.js<br/> module.exports = {<br/> themeConfig:<br/> {<br/> sidebar: 'auto'<br/> }<br/> }
```

How to disable the sidebar?

If you wish to disable the sidebar on a particular page, you can specify this in the front matter. This will give a full-width appearance to the page, and is done as follows:

```
---  
sidebar: false  
---
```

For GitHub users

Considering the facts that VuePress is actively maintained on its own GitHub repository, and that a good number of Vue.js users tend to prefer GitHub, it is natural for VuePress to come loaded with some native support for GitHub.

Syncing to a GitHub repository

You can easily add edit links and updates to and from your GitHub repository by simply specifying the repo name.

However, if you are not a GitHub user and rely on some other service, such as BitBucket or GitLab, you still have nothing to worry about. In this case, you can simply provide the full URL to your repository in the `config.js` file (as we will see in the upcoming syntax), and you will find that VuePress can pull the relevant info therefrom.

Thus, to sum it up:

- GitHub users only need to tell VuePress which repository to use
- GitLab and BitBucket (or other Git-based platform users) need to specify the full URL to their repository

To enable this particular feature, the following piece of code is all you need. Pay specific attention to the comments in this code, as you will need to change it to your required values in order to make things work. Plus, depending on how you have organized your repository, you will probably not be needing most of this code (for instance, if your docs are not in a different repository, you can safely omit that line from the following code):

```
// goes in .vuepress/config.js
module.exports = {
  themeConfig: {
    // GitHub by default, provide full URLs for others.
    repo: 'repo-address-here',
    // Customize the header label that is shown in menu
    repoLabel: 'Contribute!',

    // Further options related to "Edit on Git" link

    // if docs are maintained in a separate repo:
    docsRepo: 'docs-repo-here-full-address',
    // if docs are in a sub-directory of main repo:
    docsDir: 'docs-directory-here',
    // if you do not want to use the Master branch on Git:
    docsBranch: 'master-or-branch-name',
    // do you want people to edit your docs? Boolean value
  }
}
```

```
editLinks: true,  
  // modify the "edit this page" link that is shown  
  editLinkText: 'Help me edit this page!'  
}  
}
```

In the preceding code snippet, you will need to specify the address to your repository, as well as the the documentation repository (or whatever literature you are hosting on VuePress), if it's different. Then, you can tweak the appearance of the link and choose whether or not you want people to edit or contribute to your repository.

 editLink: false

Timestamps from GitHub

When working with GitHub repositories, it is natural for developers to showcase or display a *last updated on* timestamp, so as to keep track of progress and also to inform users that the said project is actually under active development.

This timestamp comes from the Git commit and is shown when the first commit is made for a given page. Following that, it is updated as and when a new commit is made.

[Edit this page on GitHub](#) 

Last Updated: 7/28/2018, 8:11:24 PM

In VuePress, it is turned off by default. But you can choose to turn it on and show it as a Last Updated display.

You can use the `themeConfig.lastUpdated` option that will pull the timestamp of each page's last Git commit, and display it at the bottom of the page.

The syntax is as follows. Note that the timestamp is a string value:

```
module.exports = {  
  themeConfig: {  
    lastUpdated: 'Last Updated', // timestamp at bottom of page  
  }  
}
```

That's all that there is when working with GitHub repositories for your VuePress site.

Now, since we are talking theme development, we need to turn toward something that makes the web pages appear the way they are.

Yes, we are talking about Cascading Style Sheets, or CSS.

CSS in VuePress theme development

At this point, we are going to look at which custom CSS overrides and changes we can play with in VuePress.



Note that you will need a working knowledge of CSS in order to execute any of the tweaks suggested in this section.

Custom classes for pages

More often than not, you may need to add custom CSS for a specific page on your VuePress site. In such cases, what you need is to add a custom class that is unique to the given page.

By now, you may already be aware that page-specific content needs to be declared with the YAML front matter for that particular page.

Thus, we will first need to add the unique page class to the theme container `div` in the front matter:

```
---
pageClass: custom-page-class-name
---
```

Thereafter, we can safely write custom CSS for that particular page only, as follows:

```
/* this will NOT go to .vuepress/config.js */

.theme-container.custom-page-class-name {
  /* page-specific custom CSS comes here */
}
```

But the big question is: where do we add the theme containers and write our custom CSS? Surely, not in the Markdown files.

CSS override file structure

The first step is to create an `override.styl` file in your `.vuepress/` directory. The general idea here is to add sitewide constant overrides here, such as `color`, `textcolor`, and so on.

For example, to make the entire text color black, here is what you can add:

```
|$textColor = #000000
```

You can write in normal CSS syntax. But if you so desire, using Stylus is a good idea too.



Stylus is a CSS preprocessor that is famous for making things easier to read and write when working with CSS. It supports multiple iterations, nested operators, and also does away with the need for colons, braces, and other syntactical decorations. Learn more about Stylus at <http://stylus-lang.com/>.

However, the `override.styl` file is only for Stylus constants. What about your own extra styles in CSS?

For that, you need to add another `style.styl` file in `.vuepress/` directory. In this file, you can write any CSS style that you wish to. Once again, you can use Stylus or normal CSS syntax.

If you are not very fond of Stylus, bear in mind that you can use other CSS preprocessors too, such as LESS or SASS. Remember the build pipeline config values that you learned in [chapter 3, VuePress Development – Preliminary Steps](#)? You just need to use the correct one for the relevant loaders, depending on the CSS preprocessor of your choice.



At any point, you can turn to the list of config values (build pipeline) and refer to these; this is why this book covered it well in advance in [chapter 3, VuePress Development – Preliminary Steps](#).

Backwards compatibility

This info is not relevant anymore for newer versions of VuePress. However, if you are using an older version of VuePress, you might even be able to use just the `override.styl` file and not the second `style.styl` file. Consider the following example: `// .vuepress/override.styl`
`$textColor = black // stylus constants override.`

`#my-style {} // extra CSS styles in the same file.`

While this works in practice for older versions, even then, it is double the processing. This is because Stylus requires that all Stylus constants override be compiled first, and all user-specified extra CSS be compiled later.

If you were to write your styles in the `override.styl` file, it would mean that the same file would be duplicated multiple times after being imported once. To avoid this, starting with version 0.12.0 and higher, VuePress has split CSS processing into `override.styl` and `style.styl` files, as described previously. This is the correct way of adding custom CSS styles.



Learn more about how older VuePress versions worked with the same file in this GitHub issue, available at <https://github.com/vuejs/vuepress/issues/637>.

Ejecting the theme

Sometimes, creating a custom theme from scratch is not required. The reasons for this are plenty:

- It takes a lot of time, especially on a platform like VuePress, which is still in its infancy
- Coding a custom theme requires extensive knowledge of JavaScript and can potentially defeat the very purpose of using a static site generator, where the goal is to save time and effort
- Maintaining a custom theme is not always easy in terms of bug fixes, regular updates, and so on

With that said, you may not be fond of using the default theme either, as it does not help your site stand out from the crowd. In such cases, you can choose to use custom CSS styles and other such measures, as described previously, to give your site a unique look even when using the default theme.

But what if you wish to make grand changes to your default theme? Once you update VuePress, any changes that you make to the source code of the default theme will be lost.

In these cases, you can make a copy of the default theme, and then edit the copy to make changes as if it were a custom theme. This is called ejecting the theme.

To eject your theme, you need to pass the following command:

```
|vuepress eject [target-directory-here]
```

This command will pull the default theme source code and copy it into the `.vuepress/theme` sub-directory. This particular directory will act as new home for your custom theme, and you can make changes therein.

Note that once you eject your theme, you will be responsible for its maintenance and bug fixes. You can still update VuePress as and when required, but you will

not get bug fixes or feature updates related to the default theme anymore.



Make sure you eject your VuePress theme only when you know what you're doing and you have a sound knowledge of Vue.js at large. This, of course, is beyond the scope of this quick start guide.

But once you have, indeed, ejected your theme, it is time to turn toward custom theme development!

Custom theme development in VuePress

For creating custom VuePress themes, Vue single file components are used. This means you will need functional knowledge of Vue.js in order to create a VuePress theme. Perchance you find yourself lacking in terms of time or skills for this, it is advisable to customize the default VuePress theme instead, as discussed earlier in this chapter.

With that said, the first step to create a custom VuePress theme is to create a `/theme/` directory in your VuePress root folder.

Then, in that `.vuepress/theme/` directory, create a `layout.vue` file.

Thereafter, you can proceed with custom theme creation pretty much as you would for a custom Vue.js application. The layout options and settings are entirely up to you.

You can also choose to use a custom VuePress theme from an npm dependency. In this case, you will need to make use of the theme config option in your `config.js` file. Simply add the following code to your `.vuepress/config.js` file:

```
module.exports = {  
  theme: 'your-theme-name'  
}
```

Note that you will need to declare and publish the theme on npm beforehand in order to use it as a dependency.

Customizing the default theme

The default VuePress theme that we discussed earlier can be customized to make a custom theme. However, in this case, you will need to eject the default theme, as mentioned previously in this chapter.

Following that, you can start tweaking the custom theme yourself—note that any future updates or bug fixes for the default theme will not be automatically copied over once you eject the theme. You can, however, update VuePress as usual.

Working with metadata

Every VuePress theme needs metadata to display site details properly, and as we have already learned, this is best done using the layout component.

At this stage, you should familiarize yourself with two Vue.js properties, namely, `$page` and `$site`. Both of these are injected into every component.

Thus, each time the layout component is invoked, the `$site` and `$page` properties are invoked too. It is, therefore, a logical idea to place your site's metadata to be exposed as `this.$site` and `this.$page`.

Let's look at `$site` first. Here is a sample value:

```
{
  "title": "Site-Title",
  "description": "Site description comes here",
  "base": "/",
  "pages": [
    {
      "path": "/",
      "title": "Title",
      "frontmatter": { }
    },
    ...
  ]
}
```

The preceding syntax shows very clearly that `$site` holds value pertaining to sitewide metadata.

And for `$page`, we have the following syntax:

```
{
  "path": "/current/path/to/page",
  "title": "Page Title",
  "headers": [/* ... */],
  "frontmatter": { }
}
```

As can be seen, `$page` holds page-specific metadata.

For `$site` values, most of the details are copied from the `.vuepress/config.js` file. But the `pages` value contains an array that comprises of metadata objects for each page. These metadata objects are generally lifted from the front matter of the concerned pages, or inferred, such as in the case of the page title, which can be specified in the front matter or taken from the header tags.

The `$page` values can often be used to construct customized behavior for themes. You can order the pages as per your requirements, too. This is fairly similar to customizing the order of a Vue.js application.

Other possible enhancements

The following are some other potential pointers that you can make use of when customizing your VuePress theme.



Note that you will need working knowledge of Vue.js workflow in order to implement most of these enhancements.

Application-level enhancements

In the root directory of your theme, you can create an `enhanceApp.js` file to work with application-level enhancements.

This particular file will export a hook function. Now, in this function, you should receive an object value that will contain application-specific details or data.

You can then very easily make use of the hook to register custom plugins, add extensions, register global components, customize other features, and so on.

The general syntax for this hook function is as follows:

```
export default ({
  Vue, // the current Vue.js version
  options, // specify the options for Vue instance
  router, // the router for our app
  siteData // site metadata
}) => {
  // enter custom enhancements here
}
```

Sound confusing? All of this is useful only if you are an experienced Vue.js developer and are looking to build something customized with VuePress. If you just want to get a simple site up and running, you can safely ignore these details.

These are all the details that you would need to get started with custom VuePress theme development. Once again, a theme in VuePress requires decent knowledge of Vue.js, and it is better to experiment a little before entering into production-level development.

Summary

This marks the end of this particular chapter related to VuePress themes.

We have covered quite a lot of ground in this chapter. We now have a very good understanding of what the default VuePress theme is, how it can be tweaked, how we can change the appearance of the header links, the navbar, the sidebar, and other things.

Furthermore, for cases when the aim is to build a documentation site based on a GitHub project, we have also learned how to integrate the site with our GitHub repository. Beyond that, we learned how to add CSS styles to our theme to change the appearance of our site.

Not only that, we now know how to eject the VuePress default theme and start creating a custom theme of our own, though, of course, this will require Vue.js application knowledge too.

Currently, you have an active site on localhost that you built in [chapter 4](#), *Creating a Site in VuePress*. It is a good idea to experiment with theme changes and tweaks on that very site. You have all the syntax and code samples in this chapter, but for the sake of clarity, I have refrained from making them use case-particular. Thus, you can modify these agnostic code samples to suit your production and website-specific needs.

Combined with our progress so far, you should now be able to install VuePress, tweak the `config.js` file to modify theme and other items, as well as create content in Markdown files and upload these.

All that is left is to get the website live for the world to see!

But before that, we need to cover another step of learning—localization and internationalization. Therein, we will talk about support for multiple languages in VuePress and how and when we can make use of this. The next chapter will deal with this, and following that, we will turn our attention toward getting our site live on the internet.

Doing More with VuePress!

In the previous chapters, we learned how to set things up in VuePress and build a site using it. We are also aware of how to format our content in Markdown, and the ways in which we can customize the VuePress theme.

By and large, assuming you have followed the contents of the previous chapter, you should now be able to have a good deal of command over VuePress.

But with all that said and done, one key aspect of web development still remains. Nowadays, more and more people are turning toward the internet, and English is no longer the only language spoken on the web. So, naturally, this particular key aspect deals with internationalization of your website.

In VuePress, internationalization is not rocket science. In fact, it is fairly easy to add support for different locales and languages to your website content—so much so that it can be done within minutes.

Of course, you would still need to enter the correct content in the languages that you wish to add. If you are thinking of automatic translation, then you're mistaken—internationalization does not imply that content is translated of its own accord.

So, how should you go about adding new languages to VuePress? This chapter will explain it all.

In this chapter, you will learn the following:

- Internationalization basics
- Internationalization for VuePress
- Analyzing the future growth of VuePress as a piece of software

VuePress: internationalization and doing more with it

But beyond that, what else should you know about VuePress? Well, bear in mind that it is under active development and is not as old as many other content management systems out there.

Thus, we will also be devoting a good part of this chapter later on to discuss what the future holds for VuePress, and, now that you have learned how to use it, what you should consider using it for and when.

For now, let's start with internationalization.

What is internationalization?

Before going any further, we first need to spend a moment to understand what is meant by internationalization.

At the very onset, it should be emphatically repeated that internationalization does not actually cover the machine-based translation of content.

Instead, it refers to the translation-enabling of content.

Internationalization, in itself, refers to the process by which support is added to software and products so that they can be easily be adapted and translated later into other languages. As a result, internationalization involves proper planning and strategy, such as the following:

- Comprehending the manner in which different languages are spoken and written
- Understanding the cultural aspects of multiple languages, as well as their differences

Both of these points are very important. For example, several languages, such as Arabic, are written from right to left, and not the other way round. Similarly, several other languages, such as Tamil and Hindi, have very complex scripts of their own that may seem daunting to an English speaker. Further, many languages, such as Japanese and Chinese, often contain characters that need special symbols and keyboard layouts.

Therefore, it is safe to say that internationalization is the manner and process by which content can be made ready for translation to local languages of the users. This very process of actually adapting content to suit the needs of various local languages and regions is called localization.



***Internationalization** is also abbreviated as **i18n**, or **i-18-n**, which implies **i-eighteen-n**; that is, the eighteen letters from I to N.*

In terms of software, often the biggest concern is that it should properly be ensured that characters of languages that require a single piece of code (such as English) are easily convertible into corresponding characters of languages that require more than one piece of character code (say, Chinese).

Once this condition, and certain other minor ones, are met, we can safely say that the given piece of software supports internationalization or is localization-ready.

Internationalization in VuePress

VuePress comes with support for internationalization by default. This means you can easily add locale settings for the languages of your choice, and then localize your site's content in the required languages. Thereafter, VuePress will automatically take care of mundane tasks, such as proper content structuring, menu switcher for other languages, and so on.

In VuePress, internationalization itself is implemented at two levels:

- Config at the site level
- Config at the theme level

We will be covering both. For the sake of explanation, we will be using any second language in our code, such as French. However, in order to actually localize your site in a given language, you will need to enter details related to the concerned language properly.

i18n config in VuePress

Let's first start with the theme level i18n config.

i18n config for the default theme

In the default VuePress theme, there is native support for internationalization. This is made possible with the help of the `themeConfig.locales` option.

Each locale that you add should, ideally, have its own nav and sidebar options. Sidebar and navbar options have been discussed in [Chapter 6, Theme Development in VuePress](#), already. Furthermore, you should also specify site-specific metadata and other fields for each locale separately.

Let's look at the following piece of code:

```
module.exports = {
  locales: { /* ... */ },
  themeConfig: {
    locales: {
      '/': {
        // text for the language dropdown in menu
        selectText: 'Languages',
        // label for this language in the language dropdown
        label: 'English',
        // text for the edit-on-github link
        editLinkText: 'Edit this page on GitHub',
        // configuring Service Worker popup UI (optional)
        serviceWorker: {
          updatePopup: {
            message: "New content is available.",
            buttonText: "Refresh"
          }
        },
        // algolia docsearch options (optional)
        algolia: {},
        nav: [
          { text: 'Nested', link: '/nested/' }
        ],
        sidebar: {
          '/': [/* ... */],
          '/nested/': [/* ... */]
        }
      },
      '/fr/': {
        // text for the language dropdown in menu
        selectText: 'Languages-text-in-french',
        // label for this language in the language dropdown
        label: 'French',
        // text for the edit-on-github link
        editLinkText: 'text-in-french',
        // configuring Service Worker popup UI (optional)
        serviceWorker: {
          updatePopup: {
            message: "text-in-french",
```

```

        buttonText: "Refresh-text-in-french"
      }
    },
    // algolia docsearch options (optional)
    algolia: {},
    nav: [
      { text: 'Nested', link: '/nested/' }
    ],
    sidebar: {
      '/': [/* ... */],
      '/nested/': [/* ... */]
    }
  }
}

'/es/': {
  // text for the language dropdown in menu
  selectText: 'Languages-text-in-spanish',
  // label for this language in the language dropdown
  label: 'Spanish',
  // text for the edit-on-github link
  editLinkText: 'text-in-spanish',
  // configuring Service Worker popup UI (optional)
  serviceWorker: {
    updatePopup: {
      message: "text-in-spanish",
      buttonText: "Refresh-text-in-spanish"
    }
  },
  // algolia docsearch options (optional)
  algolia: {},
  nav: [
    { text: 'Nested', link: '/nested/' }
  ],
  sidebar: {
    '/': [/* ... */],
    '/nested/': [/* ... */]
  }
}
}
}

```

What does the preceding code do?

Well, it sets up our theme for multiple locales—English, French, and Spanish.

It adds a drop-down in the menu for language selection. Then, it adds corresponding text for entries for GitHub, as well as Algolia Search and Service Worker Popup UI in different languages.

As a result, when the user is on the English version of our site, they will see Refresh as the button text. But when the user is on the French (located at `/fr` URL) version, they will see the Refresh word in French, and when the user is on the Spanish version of our site (located at `/es` URL), they will see Refresh word on the same button, but this time in Spanish, as specified by us.

Note that in the preceding code, you need to enter the actual corresponding values for French and Spanish locales. Refresh-text-in-spanish implies this is where you should specify the corresponding text, otherwise VuePress will actually (literally) show the phrase Refresh-text-in-spanish on the frontend.

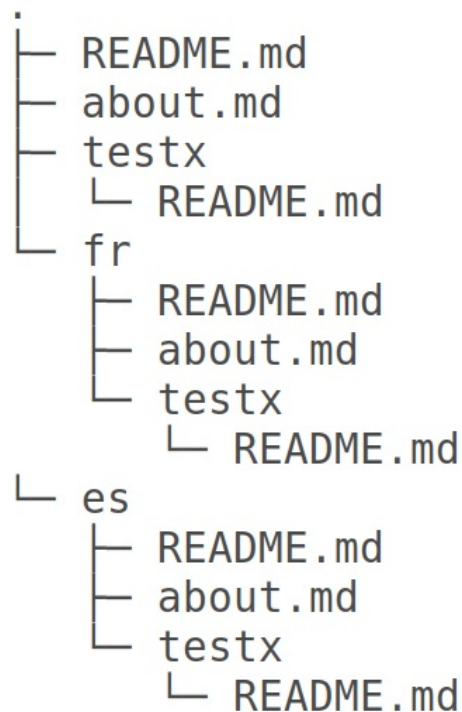
You can repeat this process for as many languages as you like.

i18n config at the site level

Now we've seen how to add multiple language support at the theme level in VuePress. However, before actually doing this, we need to first implement this at the site level.

Therefore, before adding support for a given locale in the theme, we first need to specify the locale option in our `config.js` file.

Consider the following site structure:



Notice that we have create three identical copies of the same structure with same filenames. The `/` copy is, obviously, for the English site. But the `/fr/` and `/es/` copies are for the French and Spanish versions, respectively. The `fr/about.md` file contains the same content as the `/about.md` file, albeit in a different language.

Now, once we have structured our site the way we want to, we can then specify

the locales option in our `config.js` file, as follows:

```
//goes in .vuepress/config.js
module.exports = {
  locales: {
    // The default locale can use '/' as its path
    '/': {
      lang: 'en-US', // this will be set as the lang attribute in HTML
      title: 'Good Morning',
      description: 'English VuePress'
    },
    '/fr/': {
      lang: 'fr-FR',
      title: 'Bon Jour',
      description: 'French VuePress'
    },
    '/es/': {
      lang: 'es-ES',
      title: 'Buenos Dias',
      description: 'Spanish VuePress'
    }
  }
}
```

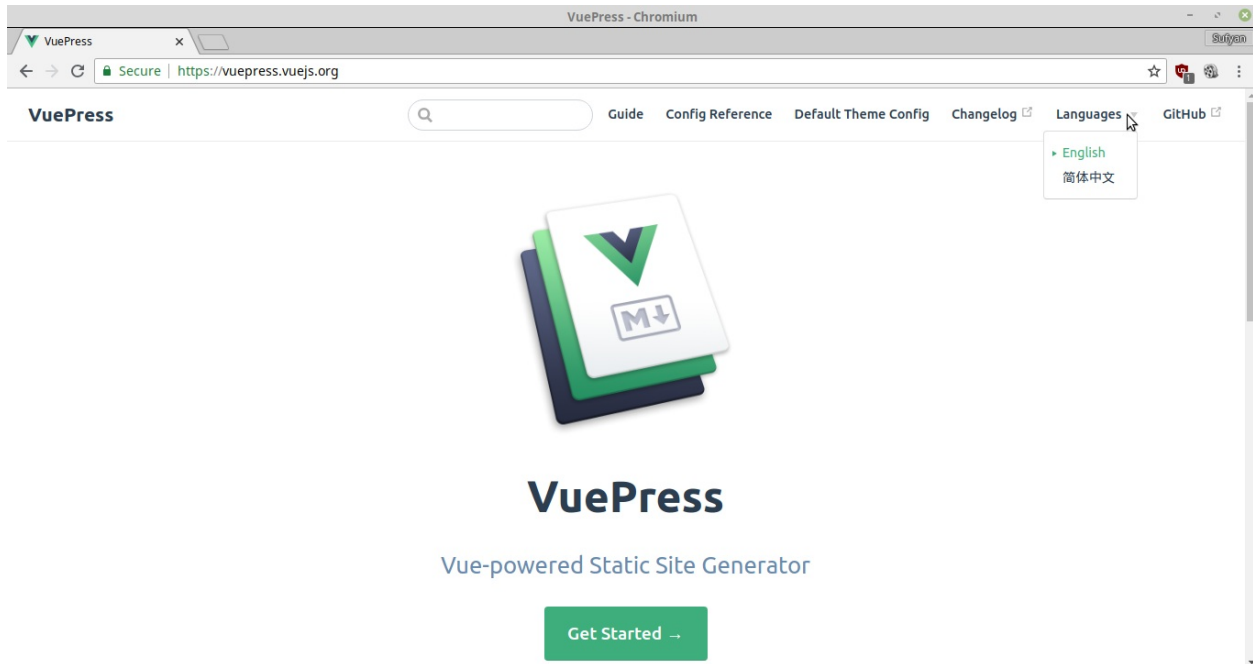
As you can see, we specify the title and description attributes for each locale separately, thereby enabling VuePress to add support for multiple locales and switch to the correct version of title, description, and other info, as per the language/locale selection.



If, by chance, a given locale does not have its own title or description values, VuePress will automatically fallback to the root values (in our case, English).

Once we are done implementing the preceding things on the site level, it is safe to implement the theme-level changes, as discussed earlier. Following that, VuePress will automatically add a language switcher entry to the menu at the navbar level.

A live example of this is the official VuePress site itself, which has been localized into English and Chinese languages. The language switcher is present in the menu, as are all the relevant tags in the required places.



That covers the extent of the i18n discussion for us, as far as VuePress is concerned. Once you are certain you have added the right directory structure and also made the necessary arrangements for the theme-specific locale changes, you can safely take your site live.

Speaking of this, we will soon be turning to deployment in the next chapter. However, at this stage, it is wiser to quickly discuss the remaining tidbits concerning VuePress at large.

Analyzing VuePress

Now that we have covered almost everything related to VuePress development, let's focus on some practical aspects pertaining to this impressive static site generator.

The big question is: where does this knowledge of VuePress take us?

Or to put it in different words: what can one expect from VuePress in near future, and just what does it feel like to maintain a VuePress site in the long run?

What's next for VuePress?

At this stage, this question is almost expected. Now that you have been introduced to a brand new static site generator tool, what lies ahead?

VuePress is definitely a solution that might gain a good deal of popularity with time. This is primarily because some of the features that it has to offer have long been desired by the developers' community.

For instance, the fact that it comes with internationalization support right out of the box is a big plus. You can create websites and projects on multiple locales with ease.

Another really impressive point about VuePress is that it comes with a fully functional default theme of its own. Not many static site generators can boast of this fact! Even more so, the default theme, as we have seen in the previous chapter, is not a mere starter or skeleton theme. It is a full-fledged piece of code with frontend display and amazing scope for customization.

Beyond that, VuePress supports Markdown editing, and owing to the sheer popularity of Markdown, this too is highly likely to work in favour of VuePress. In fact, even popular tools, such as WordPress, have added plugins for Markdown editing and formatting in recent times. Ghost, a common blogging platform based on Node.js, also supports Markdown natively.

Now that the topic of blogging has come up, it is worth mentioning that VuePress is unfortunately not fully ready for blogging yet. It is, indeed, possible to create a blog using VuePress, but such a blog will be highly lacking in terms of features and functionality. For example, there is hardly any support for tagging or categorization of blog posts, and so on.



VuePress is still under active development. This means there are really good chances that blogging support might be added in near future, pretty much like Jekyll.

However, as of now, VuePress is more of a documentation generator software, and less of a blogging platform. It can create amazing static websites that can be

served as single page apps on the fly. This very fact is the USP of VuePress!

Over the course of time, as VuePress gains more momentum, it'll also rise in popularity and usage. With Vue.js ever on the rise in the world of JavaScript frameworks, VuePress has a solid base and foundation of its own. It will not be surprising should VuePress continue to rise in popularity and start being counted among some of the top static site generators out there.

Working with VuePress in the long run

Now that you have learned how to set up and maintain a website with VuePress, should you consider using this tool for your projects in the long run?

There are very good chances you may already be using various other static site generators or similar such services. In that case, will turning toward VuePress be of extra benefit for you?

VuePress is definitely something you should seriously look at in the long run if you are looking to create documentation sites for your projects. Apparently, VuePress seems to be custom-made for the task of creating and maintaining documentation. It has native support for internationalization, multiple header link structure, a simple navigation mechanism, and the ability to integrate with GitHub or GitLab for easier pushing of updates.

Furthermore, when you think of other such projects that are often used for the purpose of documentation, you will notice that VuePress has a very clear advantage over the others. Take up Nuxt.js for instance—while it can create impressive and stellar documentation sites, it is also meant for app-level development.

VuePress, on the other hand, is not the *jack of all trades and master of none*. You can build single page apps using it, but you can hardly pull an event or e-commerce registration system entirely based on VuePress. For documentation? Perfect!

Perhaps the fact that VuePress plays very well and integrates perfectly with Markdown is another factor to consider. If you are someone who prefers writing in Markdown (just like myself), VuePress might just be the static site generation CMS that you are looking for!

On the other hand, the only caveat at the moment is that the stripped-down VuePress software does not have superb blogging support. Often, it is not

uncommon for developers to build their documentation sites along the lines of a blog, with comments, queries, tags, and categories of content. VuePress does none of that as of now. Of course, it might add such features in near future, but that is more of a prediction and less of a certainty.

When all is said and done, it is safe to assume that you can and should use VuePress if any of the following apply:

- You are looking for a simple static site generator
- You prefer working with Vue.js
- You need something that has great internationalization abilities, comes with a superb default theme, and loves Markdown
- Your primary purpose is to put together a streamlined series of content and Git integration might be of help for you

On the other side of the spectrum, you may not find VuePress to be worth your efforts if either of these apply to you:

- You need support for advanced features, such as blogging, magazine-style publishing, WYSIWYG editing, and so on
- You do not prefer Markdown, nor do you feel the need to have a streamlined organization of content modeled like a documentation project

Summary

So there you have it!

In this chapter, we have covered all that there is related to internationalization in VuePress. With the help of the right locale selection, you can easily localize your VuePress website into the languages of your choice.

Further, we have also discussed in brief what the future might hold for VuePress. Make no mistake about it, this particular product is being steadily developed and new features, as well as bug fixes, are being provided rather regularly. Of course, it may or may not fit within your workflow, but if you need something that operates on Markdown and is easy to use, VuePress is definitely worthy of a shot!

What lies ahead?

In the final and ultimate chapter of this book, we will be turning our attention toward VuePress deployment to the cloud so that we can take our website live. In [chapter 4](#), *Creating a Site in VuePress*, we already put together a small VuePress site, and in the subsequent chapters, we learned about content editing with the help of Markdown and theme customization in VuePress.

Naturally, it makes sense to wind up this quick start guide by learning how to deploy our VuePress site. Thus, in the following chapter, we will learn about deployment of VuePress websites to services such as Heroku, Netlify, and many others!

Deploying VuePress to the Web

In the previous chapters of this book, we have learned a considerable deal about VuePress. We started with static site generators and where VuePress fits in, then we moved onto installation, setup, and configuration of variables, as well as actual usage and content editing in Markdown.

Beyond that, we also paid special attention to localization, internationalization and theme customization in VuePress.

Now, at this stage, it is likely that you already have a site ready in VuePress. You have probably also configured and modified it the way you want to.

This means the only thing left for us to do is to take our site live on the net! To do that, we need to deploy our website so that the world can see it.

Deployment of web applications and websites is not a big deal, and if you have ample web development experience, you might as well be deploying projects on a regular basis. However, JavaScript applications, such as VuePress, cannot simply be pushed live on any server. Often, shared hosting providers do not just support JavaScript out of the box.

With that said, in this chapter, we will be turning our attention toward VuePress deployment. We will cover topics such as where to deploy our site, how to do it, and more.

In this chapter, you will learn about the following:

- VuePress deployment advisory
- Deploying VuePress to various cloud/remote hosting providers
- Using custom domains on VuePress sites

Deploying VuePress – an introduction

Currently, we have a VuePress site that we built in [chapter 4](#), *Creating a Site in VuePress*. Furthermore, we also managed to tweak the appearance and edit content in Markdown in the chapters following that one.

Thus, what we currently have is a website hosted on localhost; that is, our own device. It can be the laptop or the computer that you were developing on. Alternatively, if you were developing straight on a remote server, you probably already have the site deployed.

As a result, this chapter will assume that the website we built is on a local storage device and proceed accordingly. We will learn about deployment to a varied number of services, including the following:

- GitHub Pages
- GitLab Pages
- Google Firebase
- Heroku
- Netlify
- Surge.sh

But before going any further, it is worth noting that our focus will be only on those services that are either totally free to use or offer a free plan. You can, of course, choose to use a premium plan if you already have one or are planning to get one. But there is very little sense in signing up for a paid-only service purely for the sake of deploying a static site.

Naturally, this implies that VuePress deployment will require you to register with the service that you are planning to use. In some cases, you may already have an account—for example, if you are a GitHub user, you already have a Pages account too; all you need to do is to set things up to use it.

This brings us to an important question. Why can't we just randomly use any cloud hosting service to deploy our VuePress site?

Prerequisites for VuePress deployment

As we are already aware, VuePress has very few system requirements of its own. It does not use a database to fetch and write data to or from the server, and therefore, MySQL or MS-SQL is rarely needed.

Considering the fact that VuePress operates as a single page web application, it is rather swift and low on memory usage. You can run it on even the most basic of servers with limited memory, unlike other content management systems that require a good deal of memory to work with.

However, VuePress does have a few requirements in order to run properly:

- It is obvious that VuePress is powered by Vue.js. As a result, your server should have support for JavaScript and Vue.js and have all the dependencies and variables installed.
- VuePress requires Node.js 8 or higher to run.

Can we use shared hosting to run VuePress?

It is very common nowadays to run websites and projects, especially if they are not heavy traffic sites, on shared hosting environment. Such hosting plans can cost as little as \$3 per month and come loaded with all the bells and whistles.

In fact, a good number of WordPress websites tend to run smoothly on shared hosting plans, and some can easily handle over 25,000 visitors per day, if optimized properly. Naturally, it might be a temptation to consider running VuePress on a shared hosting platform. After all, if WordPress and Drupal can run on them, why not VuePress (which consumes far fewer resources than both WordPress and Drupal)?

Well, in textbook terms, yes, VuePress can be deployed to a shared server.

In practical terms, unfortunately, such deployment is not possible. This is because shared hosting tends to have its own set of preconfigured variables and language support installed. You cannot install new dependencies and other languages or frameworks on the server simply because you do not have root access to the server.

Most shared hosting providers offer PHP and MySQL by default, as well as Python and other scripting languages. WordPress, Drupal, and various other CMSs require PHP to work. But JavaScript frameworks, such as Vue.js and Node.js, are rarely found in a shared hosting environment.

This means that, while you can theoretically run VuePress on a shared server, you cannot practically hope to do so. Chances are very slim for the majority of shared hosting providers supporting Vue.js site generators on their servers any time soon.

So does that mean you need to buy a VPS or dedicated server of your own? Of

course not!

There are amazing services, both free and premium, that can help us deploy our VuePress site.

Getting started with VuePress deployment

Before getting started with the transfer process, make sure you have your project's directory structure in order.

If you are using VuePress as a local dependency inside an existing project (refer to the first chapter of this book to understand the difference between local dependency and global installation), you should run the following npm commands before beginning deployment:

```
{
  "scripts": {
    "docs:build": "vuepress build docs"
  }
}
```

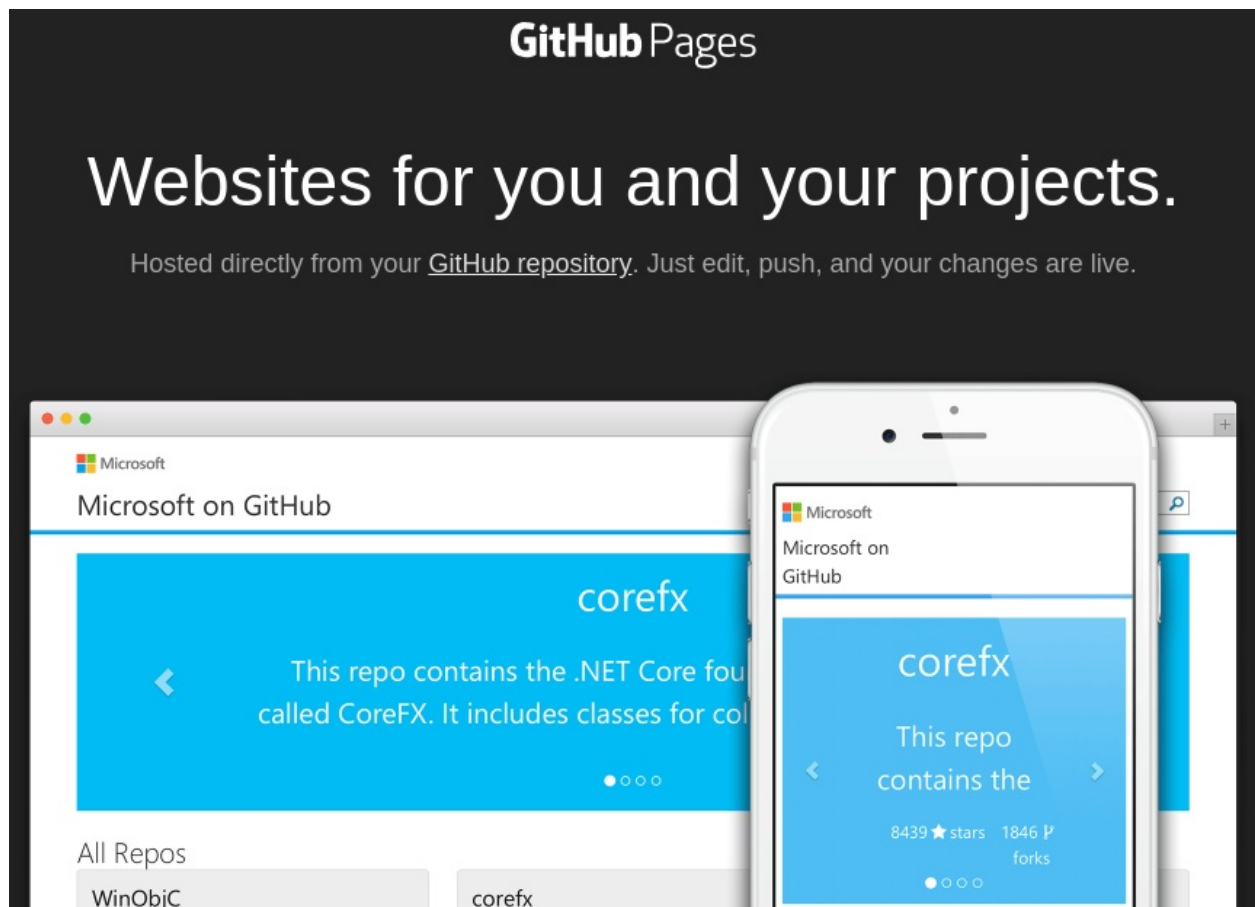
By default, the build output location used by VuePress is the dist directory within the `.vuepress` directory. You can opt to change it if you want, but for this chapter, we will be sticking with the default location only. If you have changed the default build output location, be sure to edit your deployment commands accordingly. Otherwise, deployment may fail.

Deploying VuePress to GitHub Pages

We will now learn how to deploy an existing VuePress site to GitHub Pages, a very popular service among the development community for hosting static sites.

What is GitHub Pages?

GitHub Pages is a popular service that lets you deploy and host websites for your projects. You can deploy directly from your GitHub repository as well. This means each time you make changes to your data, you just need to push them to the GitHub repository and the Pages site will be updated accordingly.



GitHub Pages is a rather well known name when it comes to hosting static sites. In [chapter 1, *Introducing Static Site Generators and VuePress*](#), you were briefly introduced to the blogging tool Jekyll, which also happens to be a static site generator just like VuePress. GitHub Pages works perfectly well in Jekyll and is, in fact, the most popular platform nowadays for hosting Jekyll-powered blogs.

There is no separate pricing structure for GitHub Pages, although you do need to sign up for one of the plans offered by GitHub. There is a free plan on offer, and

for higher end users, there are paid plans with additional features.

For the most part, a good number of newer developers tend to rely entirely on the free plan of GitHub.

Usage

Using GitHub Pages for your VuePress site's deployment is fairly simple.

First, you need to specify the correct base in your `config.js` file. By default, VuePress reads the base as `/`, so if you are deploying on the root domain, you can choose not to specify a base.



Here, by root domain, we mean something like `yourusername.github.io`.

But if you are deploying to a sub-directory, the base needs to be specified. For example, if you are deploying to `yourusername.github.io/vuep`, then the base will be `/vuep/`.



When working with GitHub Pages, your repository name is also the deployment address in general.

The next step is to create a deployment script. In your VuePress project, create a `deploy.sh` file with the following syntax (be sure to replace the path with that of your repository): `#!/usr/bin/env sh`

```
# aborting on errors
```

```
set -e
```

```
# building
```

```
npm run docs:build
```

```
# navigating to the build output directory
```

```
cd .vuepress/dist
```

```
# if you are deploying to a custom domain
```

```
# be sure to add proper CNAME records to your domain
```

```
echo 'www.myfancydomain.com' > CNAME-record-here
```

```
git init
```

```
git add -A  
git commit -m 'deploy'
```

```
# if deploying to https://yourusername.github.io  
git push -f git@github.com:yourusername/yourusername.github.io.git  
master
```

```
# or if deploying to https://username.github.io/reponame  
git push -f git@github.com:username/repo.git master:gh-pages
```

Note that of the last two commands of the preceding syntax, you must use only one, depending on your deployment path selection.

That is all. The minute you run the preceding script, deployment is done. Bear in mind, you must specify the paths and details as per your project structure.

Alternatively, if you are using VuePress as the sole site for your Pages account, and/or you wish to create an organization site, you can also deploy directly via the repository.

For that to happen, the first step is to create a new repository with your GitHub username (or organization name), along the lines of `username.github.io`, as shown here:

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 **sufyanism** ▾

Repository name

/ ✓

Great repository names are short and memorable. Need inspiration? How about [turbo-umbrella](#).

Description (optional)

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾



The next step is to clone the new repository using the following command:

```
| git clone https://github.com/username/username.github.io
```

With the preceding command, be sure to change the username field to that of your own username.

And then, all that is left to do is add, commit, and push the changes live, as follows: **git add --all**
git commit -m "initial commit"
git push -u origin master

Using custom domains

GitHub Pages allows its users to add custom domain names to their project sites, irrespective of the plan that they are on.

In order to map a domain name to your site, the first step is obviously to register the said domain name.



There are various domain name registrars out there, such as [GoDaddy](#), [Register.com](#), [Gandi.net](#), and more. I personally use [Namecheap](#) and [Name.com](#) for all of my domains, but this is more a matter of preference and less of a recommendation.

Once you have a domain name registered, you will need to modify its DNS records to point to the hosting platform in question, such as GitHub Pages. Generally, this is accomplished by means of CNAME records, albeit some platforms tend to work with A records too.



Depending on the interface and settings provided by your domain name registrar, the procedure for adding or modifying DNS records may vary. Be sure to contact your registrar's support team in case of doubts! Incorrect configurations can result in domain name non-resolution.

For GitHub Pages, you simply add the domain name to your repository settings (not the Pages settings). Thereafter, you just need to enter the CNAME records and wait for DNS propagation.



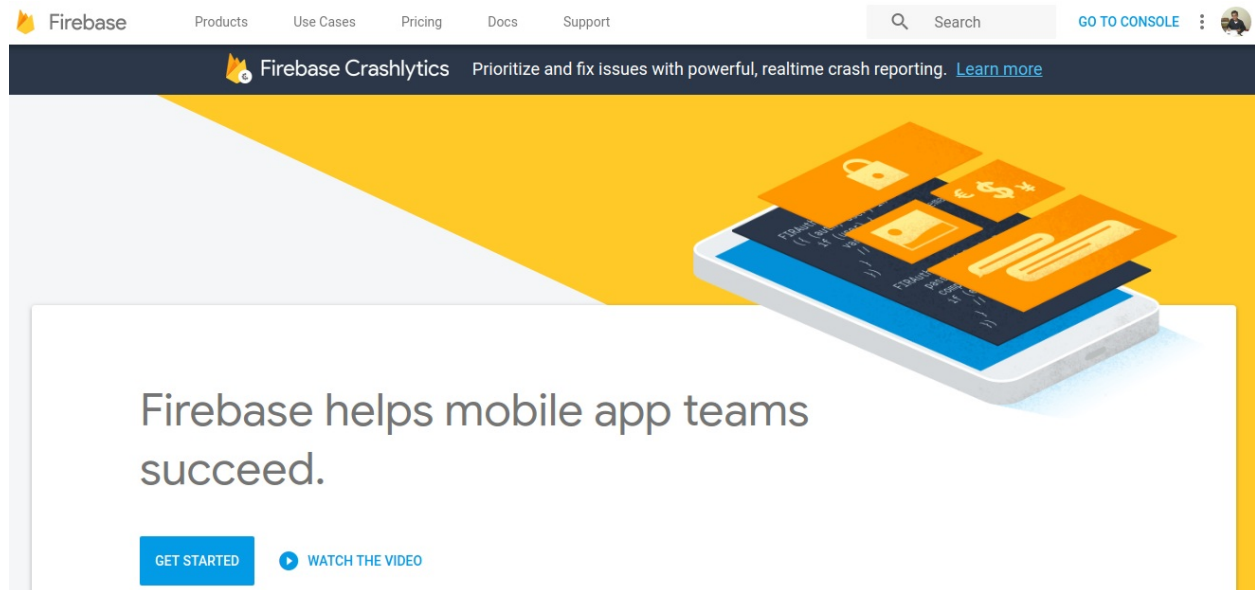
You can find the detailed custom domain name mapping guide at <https://help.github.com/articles/using-a-custom-domain-with-github-pages/>.

Deploying VuePress to Google Firebase

VuePress can be deployed to Firebase with the help of Firebase CLI tools. Firebase is a well known, but comparatively more complicated, option for hosting sites and projects in the cloud.

What is Google Firebase?

If you are not yet familiar with Firebase, the first thing to notice is that it is backed by Google. This means your project gets enterprise-level reliability, amazing cloud infrastructure, and various other features that Google has to offer.



Firebase lets you host your applications and projects in the cloud using Google infrastructure. But it does not just stop at that—there are various amazing features that come bundled with every Firebase plan (including the free one):

- A/B Testing
- Google Analytics
- Crashlytics
- Cloud Messaging (FCM)
- Remote Configuration
- Performance Monitoring
- Cloud Testing
- Custom Domains
- Custom SSL Certificates
- And More!

In fact, a simple VuePress site does not even need most of these features. With

that said, the fact that Google Firebase offers such an amazing set of features for free is a really amazing deal!

The free plan of Firebase offers you 1 GB of storage with no database access. This is more than sufficient for the majority of VuePress websites. However, the sad part is that there is only 10 GB of monthly bandwidth in the free plan. For a busy website, 10 GB may be less than required in terms of bandwidth.

You can opt for the paid plans, too, which start at \$25 per month, or choose to pay extra for bandwidth on a pay-as-you-go basis. This means that if you overstep the free plan, you only pay for the extra usage.

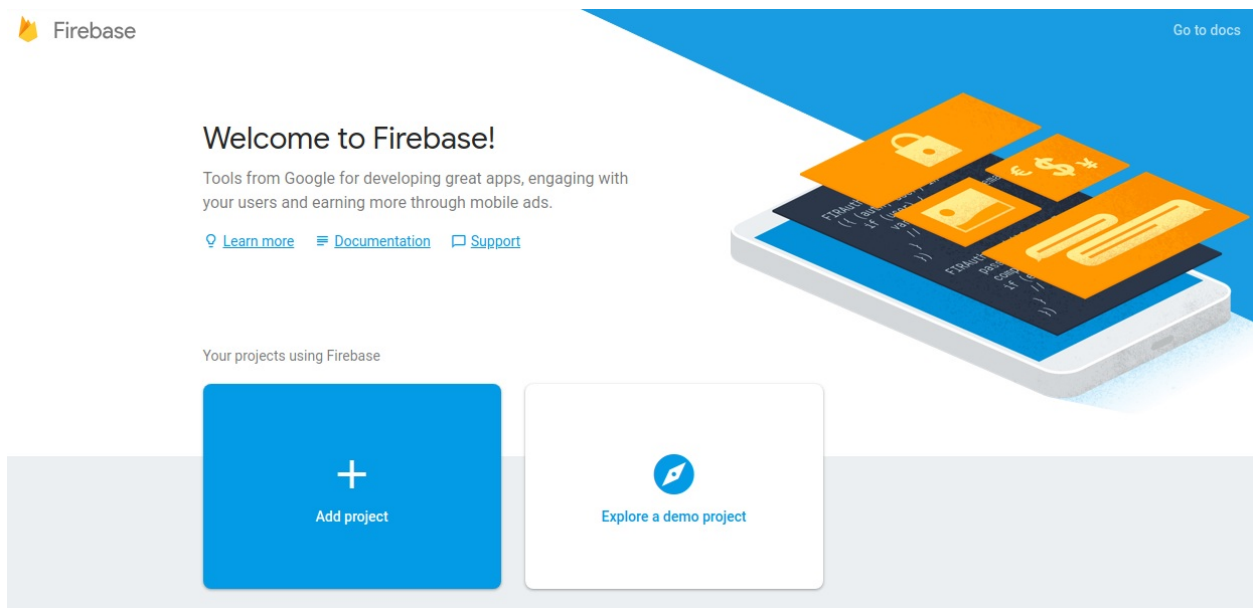


If you are worried about the pricing structure of Google Firebase, you can make use of their fee calculator to assess how much you are likely to pay per month. Find the calculator at <https://firebase.google.com/pricing/>.

Usage

In order to make use of Firebase, we will need to sign up for an account, be it free or premium.

Once we have signed up for Firebase, we will be given a Firebase ID. Make sure you keep it safe (you can find the key in the Firebase Console).



Adding a project to Firebase Console is simple—all we need to do is specify the project name, as shown here (by default, the project is added on the free plan):

Add a project

×

Project name

VuePress

Project ID ?

vuepress-sufyan

Locations ?

United States (Analytics)
us-central (Cloud Firestore) 

☒ Use the default settings for sharing Google Analytics for Firebase data

✓ Share your Analytics data with Google to improve Google Products and Services

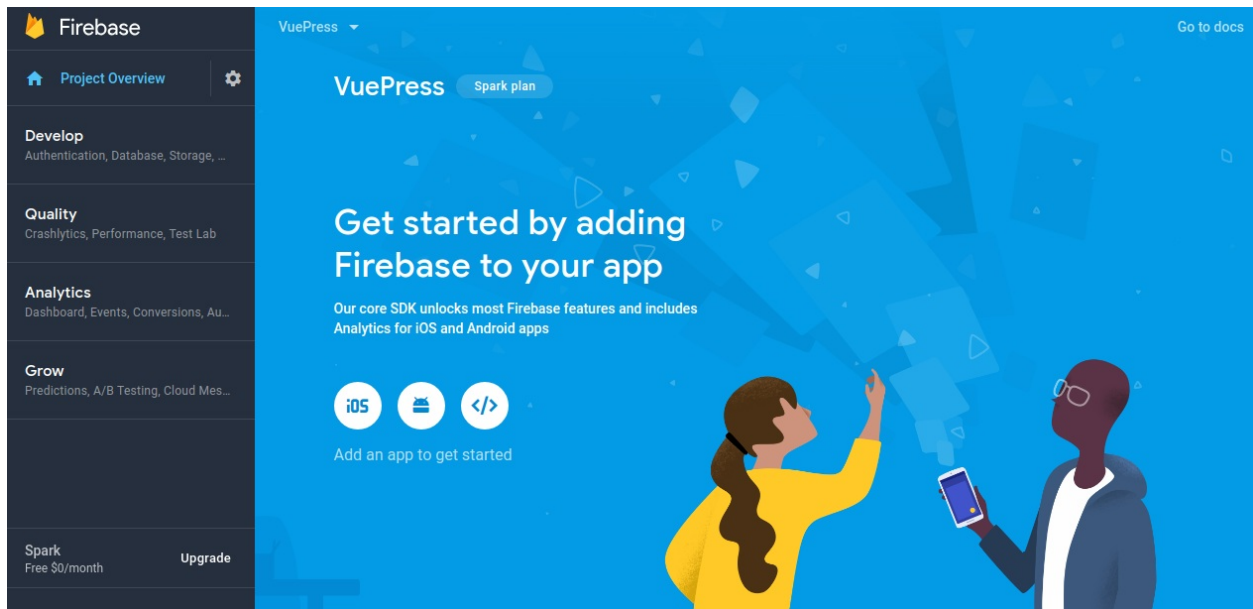
✓ Share your Analytics data with Google to enable technical support

✓ Share your Analytics data with Google to enable Benchmarking

✓ Share your Analytics data with Google Account Specialists

☒ I accept the [controller-controller terms](#). This is required when sharing Analytics data to improve Google Products and Services. [Learn more](#)

Once we have created a project, Firebase will automatically take us to the project's console page. Feel free to experiment a bit; since VuePress is still not connected to Google Firebase, there is no fear of breaking anything at this stage. The console page looks like the following:



Next, it is time to prepare our VuePress instance for deployment. We already have Node and npm on our system, but we need Firebase Tools in order to run the Firebase commands on our system.



Firebase Tools is a set of command line tools and commands that can be used to deploy code to Firebase projects, add and delete users and other data from Firebase, and read and write data to and from our Firebase database, if needed. More details can be found at <https://www.npmjs.com/package/firebase-tools>.

To install Firebase Tools, we need to run this command:

```
| npm install -g firebase-tools
```

Once installed, we can use the Firebase command in our CLI.

At this point, we need to create two files in the root folder of our VuePress site.

First, we will create the `firebase.json` file with the following syntax (add the path to your specific dist directory): {

```
"hosting": {  
  "public": "path-to-dist-directory",  
  "ignore": [ ]  
}
```

The second file we need to create is `.firebaserc` (notice that it is a dot file and will be hidden by default on Linux/UNIX systems' file viewers). The syntax for this file is as follows (add your Firebase ID): {

```
"projects": {  
  "default": "Firebase-ID-comes-here"  
}
```

Then, we just need to build the project using the following command:

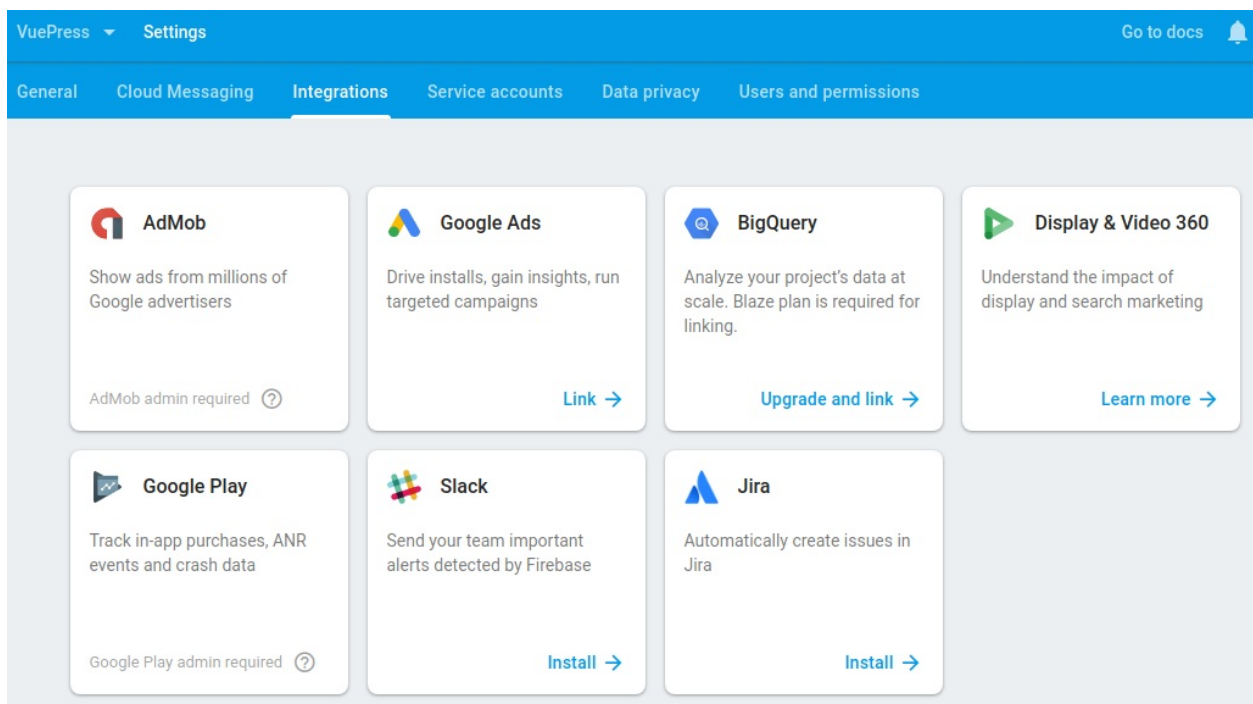
```
| npm run docs:build
```

And finally, to deploy our site, we use this command:

```
| firebase deploy
```

That's all! Our site will be deployed on the Firebase servers.

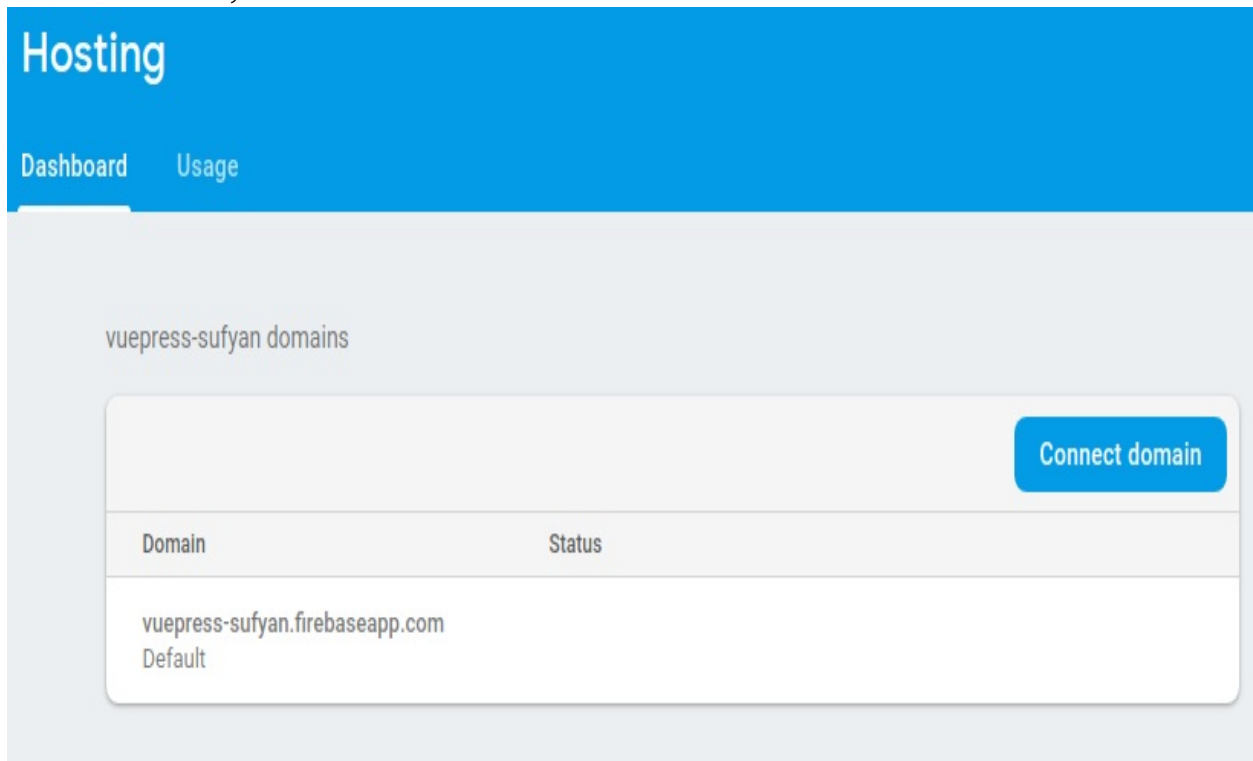
We can then return to the Firebase console for additional tasks, such as custom domain mapping, A/B testing, Crashlytics, or even other integrations, such as AdSense, Slack, and more.



Using custom domains

At this point, our site will be located at a free sub-domain, along the lines of `yoursite.firebaseio.com`.

To map a custom domain, in the Firebase Console, we need to click the Connect domain button, as shown in this screenshot:



We will then specify our domain name, and verify its ownership. The easiest way to do so is by means of a CNAME record, as shown here:

Connect domain

1 Add domain ——— 2 Verify ownership ——— 3 Go live

Enter the exact domain name that you want people to see when they visit your site. It can be a domain (yourdomain.com) or a subdomain (app.yourdomain.com)

Domain

Cancel

Continue



Keep in mind that domain name propagation can take a few hours, even in case of CNAME records.

Depending on the domain registrar that we registered our domain name with, the process for adding a CNAME record might vary. It is, therefore, advisable to check with the registrar's support team for more advice.



Learn more about connecting custom domains to Firebase at <https://firebase.google.com/docs/hosting/custom-domain>.

Deploying VuePress to Surge

VuePress can be deployed to the Surge.sh static site hosting service. This particular service is not as feature-rich as, say, Google Firebase, but it has proven to be highly reliable and is really fast in terms of performance.

What is Surge?

Surge is a service that lets you publish your site to the web right from your command line. It is more geared toward frontend developers, but there is no hard restriction on who can use it.

Unlike Firebase, which is often tailored to suit the needs of Android or other mobile app developers, Surge is meant to cater to the needs of HTML and JavaScript coders, especially web developers.

A screenshot of the Surge website banner. It features a light green background. In the top left corner is the Surge logo, a green cartoon character with a mustache, next to the word "surge" in a bold, lowercase font. In the top right corner are links for "Pricing", "Tour", "Help", "Blog", and "@surge_sh". The main heading is "Static web publishing" in a large, bold, dark font, followed by "for Front-End Developers" in a slightly smaller, bold, dark font. Below this is a line of text: "Simple, single-command web publishing. Publish HTML, CSS, and JS for free, without leaving the command line." At the bottom, there are three large numbers: "2,911,768", "31.25 TB", and "359,436", each followed by a label: "deployments", "published", and "projects" respectively.

 **surge** [Pricing](#) [Tour](#) [Help](#) [Blog](#) [@surge_sh](#)

Static web publishing *for* Front-End Developers

Simple, single-command web publishing. Publish
HTML, CSS, and JS for free, without leaving the
command line.

2,911,768	31.25 TB	359,436
deployments	published	projects

Surge comes with out-of-the-box plugins for both Gulp and Grunt. It also integrates with GitHub and supports a variety of static site generators, including the likes of Jekyll and, of course, VuePress!

In terms of pricing, Surge too has both a free and paid plan offering. The paid plan costs \$30 per month and offers unlimited everything, in addition to custom SSL, custom redirect rules, and password protection.

The free plan, on the other hand, lets us add a custom domain with basic SSL, but does not impose any limits on publishing. This means we can make any number of changes to our site and not have to worry about bandwidth limits or other such problems. This is what makes Surge especially helpful for frontend developers!

Usage

Before going any further, we will need to install Surge via npm. Simply run the following command:

```
| npm install -g surge
```

This should install Surge on our system and we can now use the Surge command. More details about the command can be found at <https://www.npmjs.com/package/surge>.

Thereafter, we just need to build our docs, as follows:

```
| npm run docs:build
```

And finally, it's time for deployment (note that you might need to pass the correct path to your dist directory if, by chance, you have changed the values or path):

```
| surge .vuepress/dist
```

That's all! Surge is among one of the easiest static site hosting services to use!



If we add a file with the name `404.html` to our root directory, Surge is smart enough to use that file as the custom 404 page to be displayed in case of page not found errors.

Using custom domains

If you have a custom domain that you wish to use, the best thing to do is to follow the official Surge documentation, which contains step-by-step guidelines for over a dozen domain registrars.

This documentation can be found at <https://surge.sh/help/adding-a-custom-domain>.

It is, however, noteworthy that if we are not using the default surge.sh sub-domain, we need to pass the custom domain name along with the deployment command. Thus, the previous command will be changed to the following: **surge .vuepress/dist mycustomdomain.com**

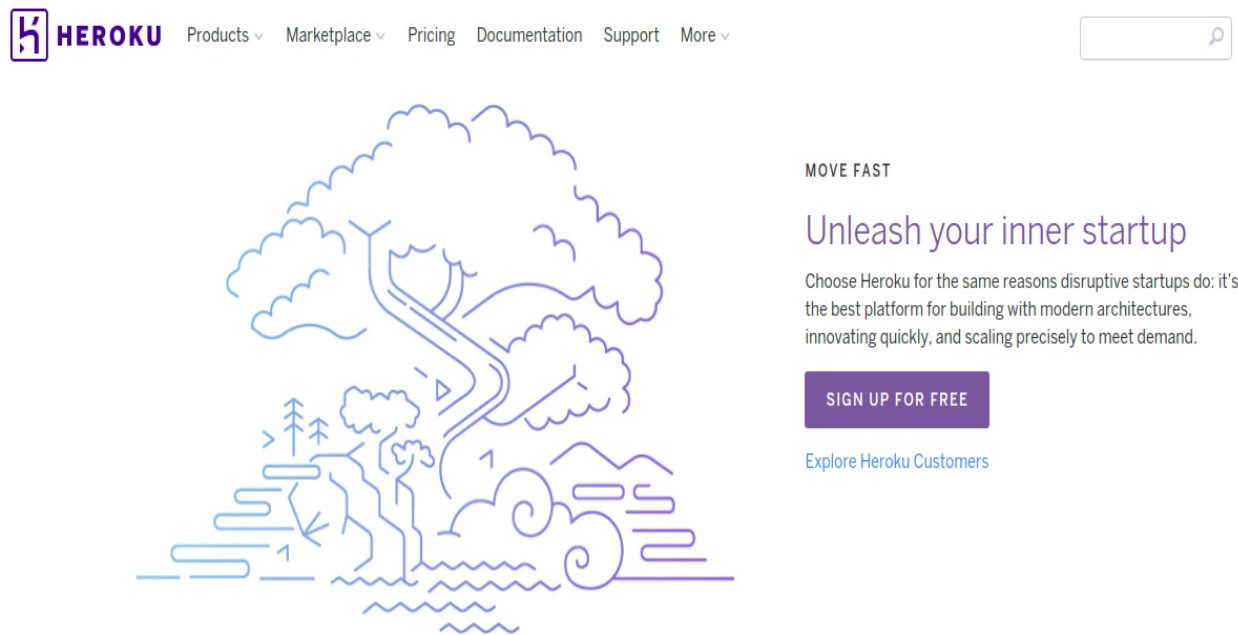
Deploying VuePress to Heroku

Using Heroku for a VuePress website often seems like overkill. This is because Heroku offers a gigantic array of features that are generally not required for a simple VuePress site.

However, if you already have a Heroku account (say, for another development projects of yours), it might make sense to add your VuePress site therein, rather than wasting time, effort, and money on signing up for another service.

What is Heroku?

Heroku is a robust and powerful cloud hosting platform that offers support for a very wide range of languages, scripts, technologies, and so on. You can run PHP, JavaScript, Ruby, UNIX, Python, Perl, and many other apps on Heroku with ease.



Heroku is also capable of supporting multiple database management systems, such as MySQL and even PostgreSQL. This means that Heroku has a wide user base that spans multiple methodologies and coding paradigms.

Naturally, in the middle of all this, using Heroku for a static site might seem unnecessary. But with that said, if your goal is to use VuePress within the larger framework of your workflow, say, an existing React or Vue.js app, then Heroku can offer a seamless platform for the entire project and also integrate with various management tools.

In terms of pricing, Heroku offers a very diverse pricing model that starts at \$7 and goes all the way up to enterprise pricing, reaching \$500 per month. It might

be more logical to present a bird's eye view of the pricing table itself:



Free

Ideal for experimenting with cloud applications in a limited sandbox.

CORE PLATFORM FEATURES

SLEEPS AFTER 30 MINS OF INACTIVITY

USES AN ACCOUNT-BASED POOL OF FREE DYNO HOURS

CUSTOM DOMAINS

512 MB RAM | 1 web/1 worker

Free



Hobby

Perfect for small scale personal projects and hobby apps.

CORE PLATFORM FEATURES

NEVER SLEEPS

FREE SSL & AUTOMATED CERTIFICATE MANAGEMENT FOR CUSTOM DOMAINS

APPLICATION METRICS

MULTIPLE WORKERS FOR MORE POWERFUL APPS

512 MB RAM | 10 Process Types

\$7 per dyno/month
prorated to the second

PROFESSIONAL

 Standard 1X 2X Enhanced visibility, performance, and availability for powering your professional applications. ALL HOBBY FEATURES + SIMPLE HORIZONTAL SCALABILITY THRESHOLD ALERTS PREBOOT LANGUAGE RUNTIME METRICS 512MB OR 1GB RAM	 Performance M L Superior performance when it's most critical for your super scale, high traffic apps. ALL STANDARD FEATURES + MIX WITH STANDARD 1X, 2X DYNOS DEDICATED AUTOSCALING 2.5GB OR 14GB RAM
∞ Process Types ✓ \$25 - \$500 per dyno/month prorated to the second	

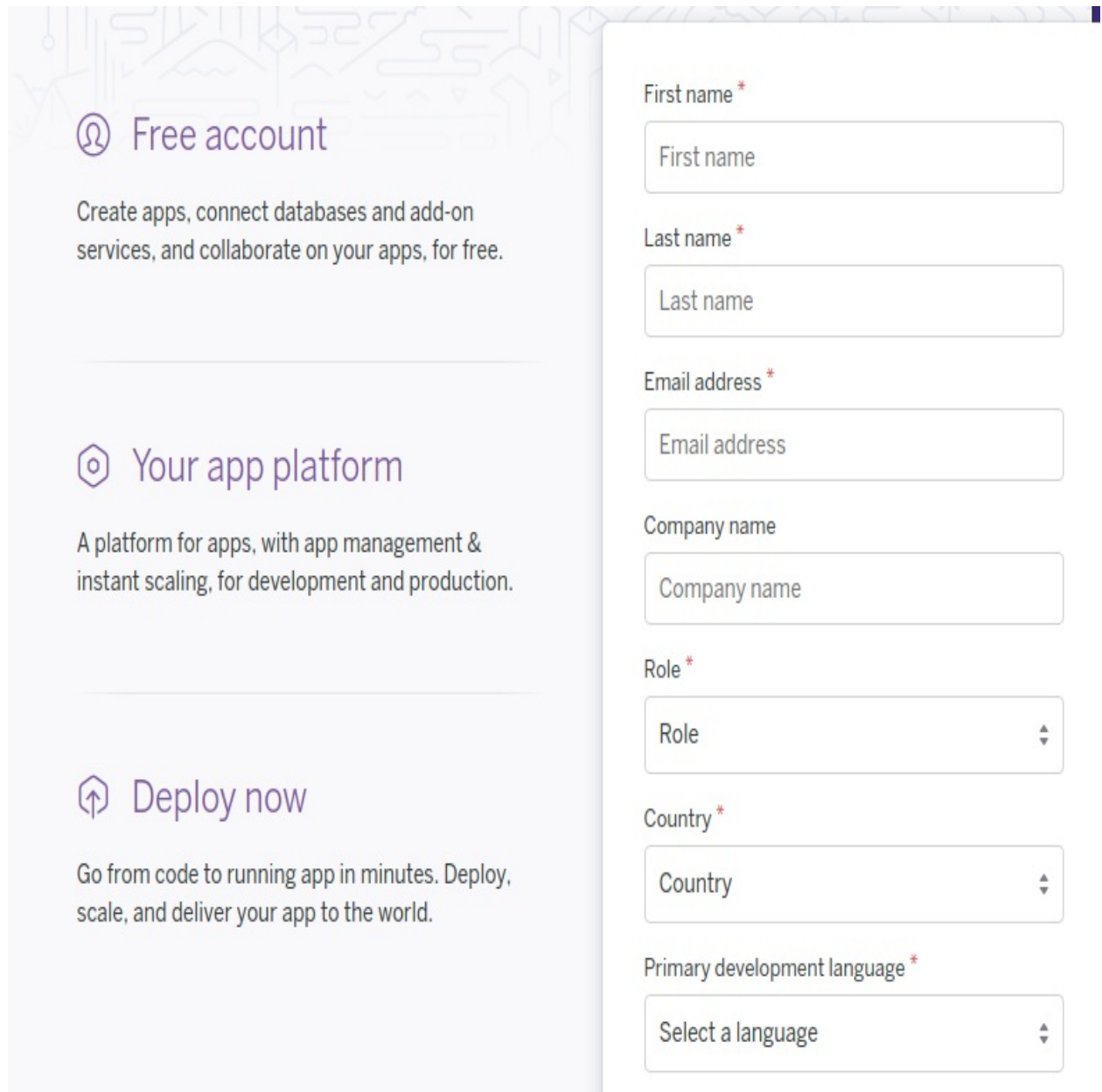
The free plan, as can be seen, lets us have one worker connection, and also allows us to map our custom domain.



You can calculate the projected cost of running a Heroku instance for your project at <https://www.heroku.com/pricing>.

Usage

In order to deploy our VuePress site to Heroku, we first need to sign up for a Heroku account. We can sign up for a free account at <https://signup.heroku.com/>. The sign-up screen will look something like this:

The image shows a screenshot of the Heroku sign-up page. On the left, there are three sections: 'Free account' with a '@' icon, 'Your app platform' with a hexagon icon, and 'Deploy now' with an upward arrow icon. Each section has a brief description. On the right, there is a form with several fields: 'First name *', 'Last name *', 'Email address *', 'Company name', 'Role *', 'Country *', and 'Primary development language *'. Each field has a corresponding input box or dropdown menu. The form is set against a light purple background with a subtle pattern of icons.

Note that there is a requirement to select a primary development language during

the signup process. If there is no other language that you mainly work in, Node.js is what you need to select for VuePress deployment. It does not matter per se; you can select any language and still be able to deploy apps in other languages with ease.

Once we have successfully created an account, we will be taken to the Heroku Dashboard. We can now register a new app here:

Create New App

App name

vuepress-sufyan

vuepress-sufyan is available

Choose a region

Europe

Add this app to a pipeline

Pipelines form a continuous deployment workflow and enable additional features. [Learn more.](#)

Choose a pipeline

Create app

There are two major methods to deploy a VuePress project to Heroku:

- Using the Heroku CLI to rely on Heroku Git
- Connecting to GitHub itself

Connecting to GitHub means we are deploying a branch of our repository to Heroku, such that changes are automatically pushed to the live site, as shown here:

Deployment method



Heroku Git
Use Heroku CLI



GitHub
Connect to GitHub



Container Registry
Use Heroku CLI

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

View your code diffs on GitHub

Connect your app to a GitHub repository to see commit diffs in the activity log.

Deploy changes with GitHub

Connecting to a repository will allow you to deploy a branch to your app.

Automatic deploys from GitHub

Select a branch to deploy automatically whenever it is pushed to.

Create review apps in pipelines

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. [Learn more.](#)

Connect to GitHub

However, let's discuss the mainstream route of using the CLI, as it can be more challenging at times.

Before going any further, the Heroku CLI needs to be installed. With npm, it is as easy as issuing the following command:

```
| npm install -g heroku
```

However, things are slightly different here. Heroku CLI relies on a cutting-edge version of Node, and if our system does not keep pace with it, things can potentially break. To avoid this, if Heroku CLI is installed via npm, automatic updates are not offered. We will need to manually update the CLI as and when possible or required.

Naturally, this may add too much additional work. The easier route is to use the official installers. Heroku provides us with installers for Windows, Mac, and Ubuntu platforms. For other Linux versions, the general command for installation is the following (to be run as superuser):

```
| sudo snap install --classic heroku
```



More details about the Heroku CLI installation process, as well as links to download the installers for Windows and macOS, can be found at <https://devcenter.heroku.com/articles/heroku-cli>.

Heroku CLI requires Git to function. Make sure your system has Git installed before installing Heroku CLI.

Once we have created an account and set up the Heroku CLI, it is time to log in to Heroku using our password and username credentials. All we have to do is pass the following command, and then enter our login details when prompted:

```
| heroku login
```

Thereafter, we need to create a `static.json` file in the root of our VuePress project. The file should be populated with the following (the path to the `dist` directory needs to be specified in full):

```
| {  
|   "root": "path-to-vuepress-dist-directory"  
| }
```



The `static.json` file is very important and remote deployment will not work in its absence!

Once that is done, we need to set things up with the Heroku Git. To add the Git commit, use the following code:

```
| # version change  
| cd project-path  
  
| git init  
  
| heroku git:remote -a app-name-here  
| git add  
| git commit -m "Ready to deploy."
```

To create the new app, enter the following command:

```
| # creates a new app  
| heroku apps:create vuepress-app-name
```

To set the `buildpack`, use this command:

```
| # buildpack for static sites  
| heroku buildpacks:set https://github.com/heroku/heroku-buildpack-static.git
```



The Heroku `buildpack` is a custom set of tools and commands meant mainly for static sites.

Learn more about its usage over at its GitHub page at <https://github.com/heroku/heroku-buildpack-static>.

Lastly, we just need to deploy our site and push the changes, as follows:

```
| git push heroku master
```

And we are done! Metrics and usage statistics about the site can be viewed in the Heroku Dashboard itself.

Using custom domains

We can add multiple custom domains to our VuePress site in the Heroku Dashboard. By default, our site will use the `herokuapp.com` sub-domain only.

However, in order to add a custom domain, verification of your account is necessary. This can be done by adding credit card details. While there will be no billing or charges on the card, such details are mandatory in order to prevent misuse of the Heroku system.

Domain Your app can be found at <https://vuepress-sufyan.herokuapp.com/>

SSL Upgrade to paid dynos to configure Heroku SSL

Add domain

Custom domains will appear here

Custom domains allow you to access your app via one or more non-Heroku domain names (for example, `www.yourcustomdomain.com`)

Once the custom domain is added, we can modify CNAME records over at the domain registrar to point the domain to our VuePress site.

Deploying VuePress to Netlify

Netlify has become famous among web developers due to its automated mechanism of working. You can simply push changes from GitHub, GitLab, and others, and Netlify can handle the rest.

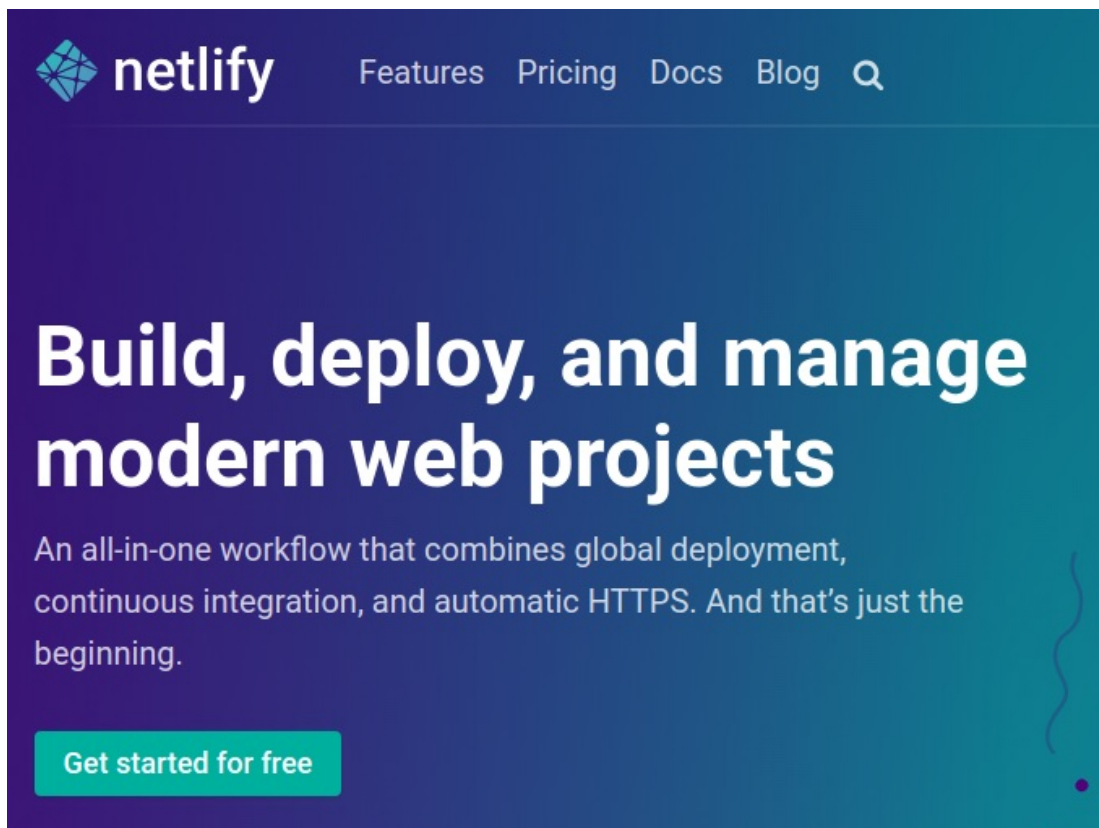
In fact, there is very little chance of something going wrong when working with Netlify. It is often the recommended solution for folks looking for a cloud hosting service that simply works within a few clicks.

Needless to say, Netlify works very well with VuePress too.

What is Netlify?

Netlify defines itself as an *all in one platform for automating modern web projects*. Put simply, it lets you deploy your web projects on a global infrastructure with features such as automatic SSL, custom domains, multiple **content delivery networks (CDN)**, and so on.

You can choose to deploy directly via Git, and then push changes as and when required:



Netlify has premium plans starting from \$45 per month upward. The free plan, however, lets us deploy our personal projects with ease and also allows us to use custom domains. The downside is that there is a one-user limit on the free plan—this rules out any team collaboration. The pricing options are shown here:

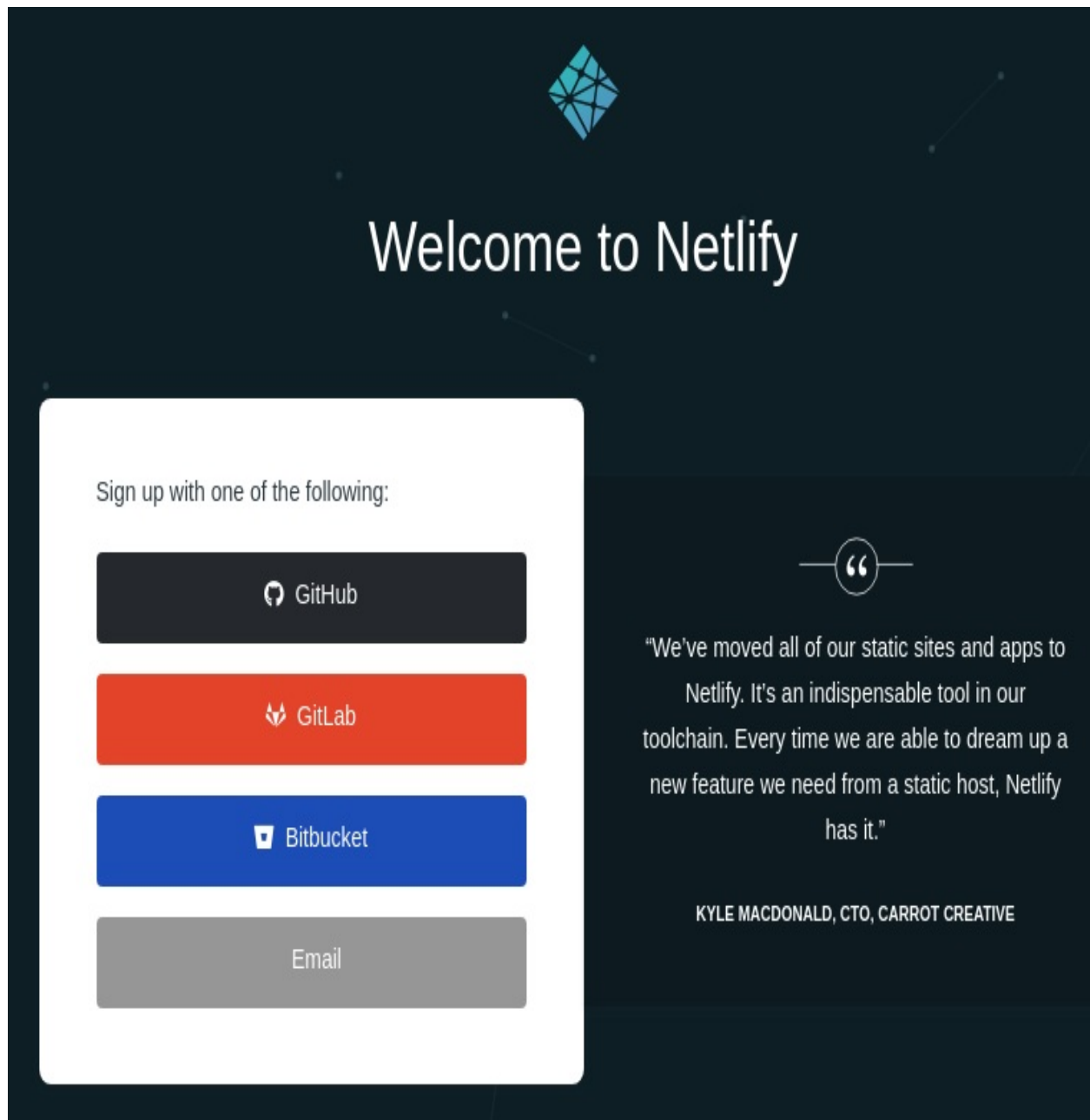
Free \$0 /month Great for personal websites and projects 1 user limit - Continuous deployment - Custom domains & HTTPS Get started for free	Team Pro \$45 /month For small companies and web teams 5 users included - Owner/collaborator roles - Password protection Get started	Team Business \$290 /month For agencies and larger organizations 10 users included - Multiple roles - Role-based access control Get started	Enterprise Custom For enterprises with demanding requirements - Advanced Enterprise CDN - More Global POPs - Dedicated Build Environment Learn More
--	---	--	--

Also, enterprise-level CDN is also not available on the free plan. But for what it's worth, the free plan should suffice for VuePress deployment.

Usage

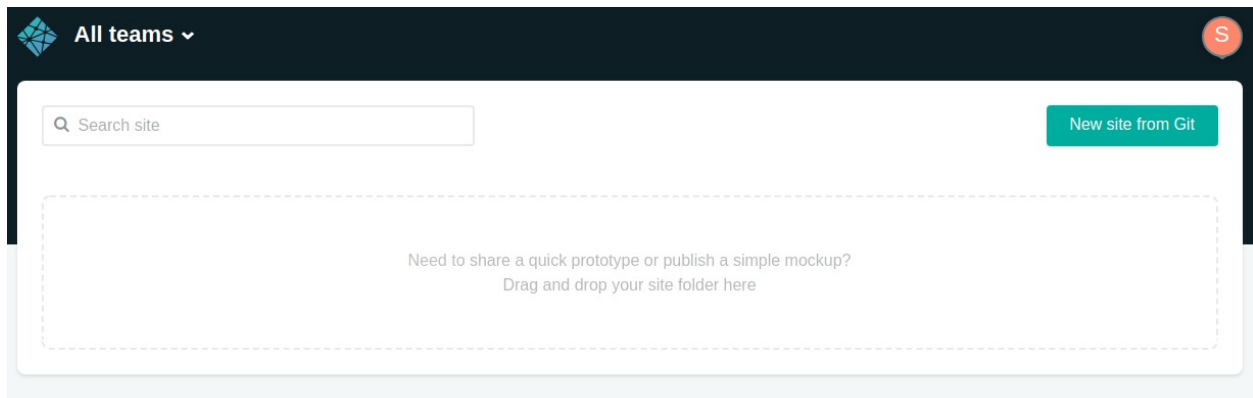
In order to deploy our site to Netlify, we first need to sign up for a plan. Thereafter, we will be able to add details about our project therein.

The good thing is that Netlify offers multiple sign-up options. We can choose to sign up using GitHub, GitLab, or BitBucket, and then push our repositories easily, as shown here:



Alternatively, we can rely on the email sign-up mechanism.

The Netlify interface is amazing in that it is very empty, so there is no possibility of a newbie user getting confused:



All we need to do is click the New Site From Git button!

Then, we will need to connect to the Git service of our choice, as shown here:

Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider

2. Pick a repository

3. Build options, and deploy!

Continuous Deployment

Choose the Git provider where your site's source code is hosted. When you push to Git, we run your build tool of choice on our servers and deploy the result.

 GitHub

 GitLab

 Bitbucket

☐

Limit GitHub access to public repositories. [What's this?](#)

Thereafter, we just need to pick the repository where our project is located, and then specify the options as follows:

- For the build command, we will enter `npm run docs:build`
- And for the `publish` directory, we must specify the full path to our `dist` directory, such as `.vuepress/dist`

That's all!

Using custom domains

Netlify allows us to add a custom domain name to our site even in the free plan. By default, the site is first live at the `sitename.netlify.com` sub-domain.

After the site has been added, we can specify the domain name we want to add, and then add CNAME records, as per the interface provided by our domain name registrar.



It is worth noting that Netlify also offers DNS management services for domains, albeit on the paid plans.

As can be seen, deploying a site to Netlify is rather easy. The only catch is that working from localhost for deployment can be rather tricky. But if you are a GitHub, GitLab, or BitBucket user, Netlify is a very simple and reliable solution for deployment.

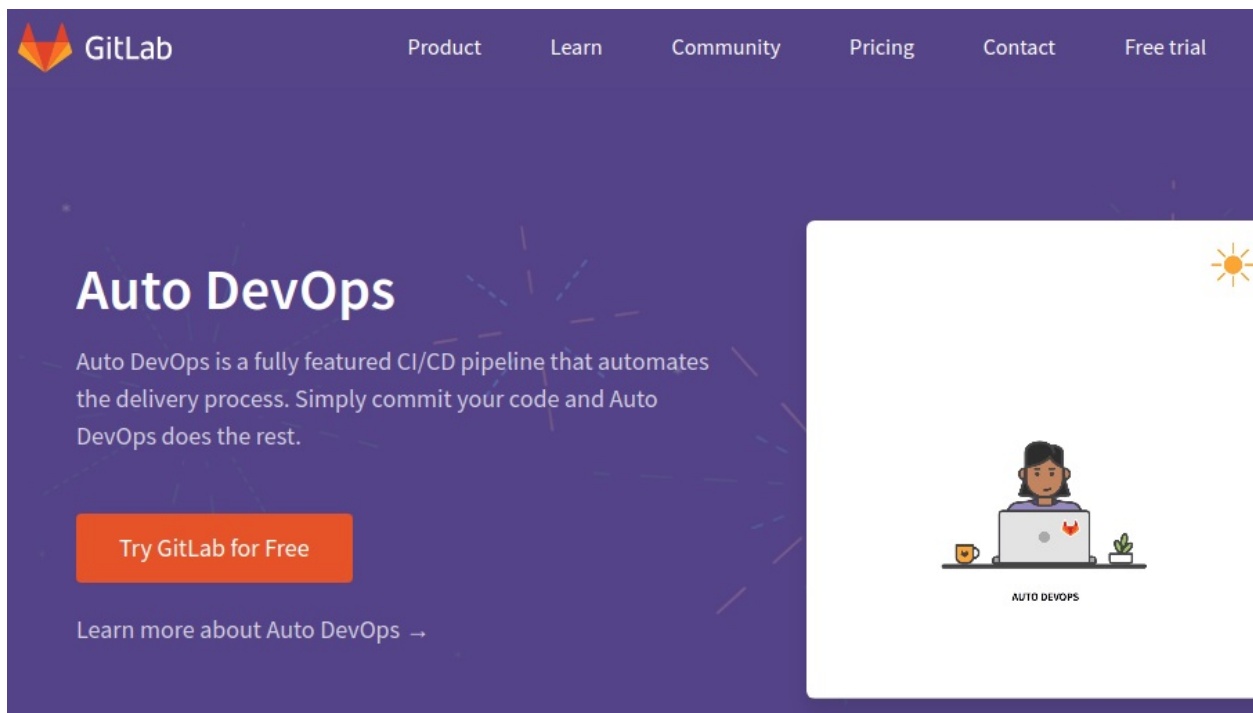
Deploying VuePress to GitLab Pages

Pretty much like GitHub Pages, GitLab Pages too is a similar breed of offering and provides more or less the same level of features.

Furthermore, much like GitHub Pages, GitLab Pages is also heavily focused on static sites and their deployment. It works out of the box with services and software such as Jekyll, Hugo, Harp, Hexo, Pelican, and obviously VuePress. We learned about such static site generators in [chapter 1](#), *Introducing Static Site Generators and VuePress*, of this book.

What is GitLab Pages?

GitLab Pages offers reliable and highly versatile hosting solutions for projects and repositories based on GitLab. While GitLab is fairly similar to GitHub in the sense that both of these services are based on Git, the former has a self-hosted offering too that has risen in popularity of late. There are many developers who prefer GitLab for this very reason.



GitLab Pages allows us to connect custom domains, as well as SSL certificates. We can choose to run GitLab on our own servers, or over at [GitLab.com](https://gitlab.com)—irrespective of our choice, GitLab Pages can still be used to deploy sites to the cloud.

In terms of pricing, both the self-managed and hosted variants of GitLab come with a free plan. The good thing is that, unlike GitHub, we can even have private projects in the free plan! The paid plans are higher end usage-specific; if your end goal is just to set up a VuePress site, the free plan is more than sufficient.



GitLab Pages examples

Example websites hosted by GitLab Pages

Filter by name...		Last created ▾
🔖 	jekyll Example Jekyll site using GitLab Pages: https://pages.gitlab.io/jekyll	★ 141 🔄 1 week ago
🔖 	plain-html Example plain HTML site using GitLab Pages: https://pages.gitlab.io/plain-html	★ 94 🔄 1 day ago
🔖 	middleman Example Middleman site using GitLab Pages: https://pages.gitlab.io/middleman	★ 12 🔄 1 day ago
🔖 	jekyll-branched Example Jekyll site deployed in a branch using GitLab Pages: https://pages.gitlab.io/jekyll-branc...	★ 6 🔄 1 month ago
🔖 	hugo Example Hugo site using GitLab Pages: https://pages.gitlab.io/hugo	★ 137 🔄 1 day ago

It is noteworthy that there is no separate pricing structure for GitLab Pages. The pricing, be it free or paid, is for GitLab itself. GitLab Pages comes as a free add-on with all plans.

Usage

In order to deploy our VuePress site to GitLab Pages, we first need to ensure that the correct base is specified in our `config.js` file. Otherwise, the deployment will just not work.

By default, VuePress reads the base value as `/`. This means that if the site is to be deployed at `username.gitlab.io`, then we can omit the base value in `config.js`.

However, if we are to deploy the site elsewhere, say, `username.gitlab.io/mysite`, then we need to specify the base as `/mysite/`, and so on.

Furthermore, it is very important that the `dest` value in the `config.js` file is set to `public`. Otherwise, GitLab Pages may not be able to read the project data.

Once we have entered the required values in the `config.js` file, we need to create a new file named `.gitlab-ci.yml` in the root of our VuePress project. We will then populate the said file with the following contents, as suggested by VuePress developers themselves: `image: node:9.11.1`

```
pages:
  cache:
  paths:
  - node_modules/

script:
  - npm install
  - npm run docs:build
artifacts:
  paths:
  - public
only:
  - master
```

This file simply runs the required scripts to build and deploy our site as and when we make changes to it.

Using custom domains

By default, each site on GitLab Pages is available on a project page address, such as `myusername.gitlab.io/myproject`.

We can change it to `myusername.gitlab.io` (that is, removing the `/myproject` sub-path) very easily. All we need to do is enter the project's name in the settings as `myusername.gitlab.io`—this works with virtually all of the static site generators hosted on GitLab Pages.

If, by chance, we need to add a custom domain name to our site, this is doable as well. To do this, we need to specify the domain name in the site settings, and then add the relevant CNAME records to our domain itself.



GitLab Pages offers a detailed guide to custom domain mapping at <https://docs.gitlab.com/ce/user/project/pages/introduction.html#add-a-custom-domain-to-your-pages-website>.

Summary

Well, this brings us to the end of this quick start guide for VuePress. In this chapter, we covered VuePress deployment for various cloud hosting services.

Links to remote or cloud platforms

For easier reference, here are the links to the cloud and remote hosting providers that we have talked about in this chapter:

- **GitHub Pages:** <https://pages.github.com/>
- **GitLab Pages:** <https://about.gitlab.com/features/pages/>
- **Google Firebase:** <https://firebase.google.com/>
- **Heroku:** <http://heroku.com/home/>
- **Netlify:** <https://www.netlify.com/>
- **Surge.sh:** <http://surge.sh/>

Concluding words

During the course of this book, we have learned about and acquired a good deal of knowledge related to VuePress, including the following:

- VuePress installation on localhost or machine
- VuePress configuration
- VuePress usage and building
- VuePress theme customization
- Working with content in Markdown
- VuePress internationalization
- And finally, in this chapter, we learned how to deploy a site from a localhost or storage device to a remote or cloud platform

As you progress with VuePress development and deployment, you will notice that it is very easy to use, and a rather robust piece of software. Furthermore, VuePress is still under active development and is a relatively new tool. This means that, as time goes by, VuePress is bound to grow in both features and popularity.

In fact, VuePress came up with nine new releases (both major and minor combined) during the period while I was writing this book!

At this point, it is highly advisable and recommended to learn Vue.js too, especially if you are looking to build projects in VuePress and do more with them. Having a JavaScript framework at your command is always a good idea in today's times, keeping in mind the rising popularity of JavaScript.

Readers can browse this book's repository over at GitHub for code samples. This can be found at <https://github.com/packtpublishing/vuepress-quick-start-guide>.

Plus, the latest VuePress code can be found at its own GitHub repository, over at

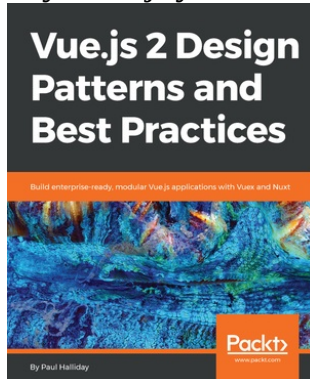
<https://github.com/vuejs/vuepress>.

That said, I hope you've had a great time reading this book, and learning about VuePress in particular.

Happy coding!

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

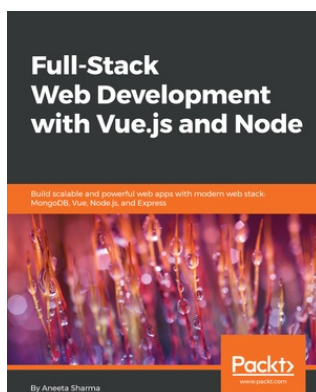


Vue.js 2 Design Patterns and Best Practices

Paul Halliday

ISBN: 978-1-78883-979-2

- Understand the theory and patterns of Vue.js
- Build scalable and modular Vue.js applications
- Take advantage of Vuex for reactive state management
- Create single page applications with vue-router
- Use Nuxt for FAST server-side rendered Vue applications
- Convert your application to a Progressive Web App (PWA) and add ServiceWorkers and offline support
- Build your app with Vue.js by following best practices and explore the common anti-patterns to avoid



Full-Stack Web Development with Vue.js and Node

Aneeta Sharma

ISBN: 978-1-78883-114-7

- Build an application with Express.js
- Create schemas using Mongoose
- Develop a single-page application using Vue.js and Express.js
- Create RESTful APIs using Express.js
- Add test cases to improve the reliability of the application
- Learn how to deploy apps on Heroku using GitHub
- Add authorization using passports

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!