Introduction to full stack web development with Go and Vue.js

Enrico Bassetti, Emanuele Panizzi

Contents

Preface

Who is this book for?

The ideal reader is a student enrolled in a Computer Science university program: someone with a basic knowledge of at least one programming language and willing to learn how to build a full-stack web application.

This book is also suitable for developers who want to develop real-world web applications professionally using state-of-the-art technologies.

Requirements

To fully understand and follow the examples in this book, you should have some basic knowledge of how to code in at least one programming language and use your operating system's command line for launching tools and managing files.

We suggest you test the examples in this book in a Debian GNU/Linux real or virtual machine. We tested these examples on GNU/Linux; they might work in other operating systems, provided that tools are installed and configured correctly (however, some commands might be different, and you need to translate them). If unsure, you can download the Debian ISO freely from https://www.debian.org and VirtualBox from https://www.virtualbox.org/, and then use VirtualBox to create a Debian virtual machine.

If you want to deepen your knowledge of the GNU/Linux command line, we suggest "*The Linux Command Line*" book from *William E. Shotts, Jr.* here: https://sourceforge.net/projects/linuxcommand/files/TLCL/19.01/TLCL-19.01.pdf/download.

Typographic conventions

Examples that involve running programs on the command line will use the dollar sign $ to denote the shell prompt as a standard user. Since the dollar sign indicates your shell prompt, you should not type it in.

Lines without the dollar sign represent the output for the command. For example, in this block of text:

```
$ git --version
git version 2.30.2
```

The text `git --version` is the command you need to run, and `git version 2.30.2` represents the command's output.

Note that, in UNIX philosophy, commands that run successfully should have no output (if not explicitly requested using a flag). Many of the commands that we will use have this behavior.

Note for e-book readers

Some e-book readers have a small screen, and some images/tables may overflow out of the screen. To scroll laterally and see the rest of the image or table, consult the manual of your e-book reader.

About the Authors

Emanuele Panizzi is an Associate Professor in Computer Science at Sapienza University of Rome, Italy. He leads a research team focusing on human-computer interaction, app design, gamification, and context-aware mobile interaction. In the two areas of smart parking and earthquake detection, his current study uses AI to recognize users' behaviour and context. Designing mobile user interfaces with implicit interaction and crowdsensing applications is the experimental component of this study. Panizzi supervised several large software projects for Sapienza University and for companies and start-tups. He teaches HCI and software architecture. He has served as a consultant for major national and international corporations.

Enrico Bassetti is a post-doctoral researcher at Sapienza University of Rome, Italy. His research focused on network security, fully distributed sensing systems, and Internet-of-Things devices. In 2019, he earned his Master's in Cybersecurity from Sapienza University and defended his Ph.D. in 2023. In addition to his research experience, Bassetti has ten years of company experience as a system architect consultant, DevOps advocate, system/network administrator, and trainer on security and Linux.

If you have any feedback about any aspect of this book, email us at panizzi@di.uniroma1.it and bassetti@di.uniroma1.it.

1 Introduction

1.1 Types of Applications in Web Development

In the world of web development, applications are the heart and soul of the digital landscape. These computer programs enable users to accomplish a wide array of tasks, from managing their finances to playing games or simply staying connected with friends and family. Applications come in various shapes and sizes, tailored to suit different platforms and user needs.

1.1.1 Desktop and Mobile Applications

Desktop applications, also known as native desktop applications, are designed to run directly on a desktop computer's operating system. To use a desktop app, users typically need to install it on their computer. These applications can vary widely in their complexity, from lightweight utilities to sophisticated software suites.

Mobile applications are tailored for smartphones, tablets, or even wearable devices like smartwatches. Like desktop apps, mobile apps also need to be installed on the user's device. They can be further categorized into two types:

- *Native Apps*: Native apps are explicitly developed for a specific mobile operating system, such as Swift for iOS, Java or Kotlin for Android, or Objective-C for older iOS versions. This approach provides developers with the maximum control and access to device-specific features and capabilities.
- *Hybrid Apps*: Hybrid apps, on the other hand, are created using web technologies like HTML, CSS, and JavaScript and then cross-compiled to run on multiple operating systems. Tools like Apache Cordova or frameworks like Ionic allow developers to build hybrid apps that can be deployed on both iOS and Android platforms.

1.1.2 Web Applications

Web applications, often referred to as web apps, are distinct from their desktop and mobile counterparts. They run within a web browser, which means users don't need to install them on their devices.

Instead, web apps can be accessed simply by navigating to a specific URL. These applications are ubiquitous on the internet, offering a wide range of services, from email clients to project management tools.

A specific subset of web apps is known as *Single Page Applications* or SPAs. SPAs interact with users by dynamically updating the current web page with new data from the web server, without requiring a full page reload. This approach results in a smoother, more seamless user experience, as only the necessary content is updated, making the application feel more like a traditional desktop or mobile app.

Progressive Web Apps, or PWAs, are a modern evolution of web applications. They offer the best of both worlds, combining the accessibility of web apps with the capabilities of native apps. Users can install PWAs by adding them to their device's home screen, providing easy access like a native app. Additionally, PWAs can work offline or in low-quality network conditions, making them highly resilient and user-friendly.

1.1.3 Web APIs

Web API stands for Web Application Programming Interface. It is a set of rules and protocols that allows one software application (desktop, mobile or web application, or even servers) to request and exchange data or functionality with a server over the Internet. Web APIs are the building blocks that enable different web services, servers, and clients to communicate effectively.

At their core, web APIs define a collection of endpoints or URLs that developers can use to send requests and receive responses in a structured format, typically in JSON (JavaScript Object Notation) or XML (eXtensible Markup Language). These endpoints are like doors into a web service, each offering a specific function or data resource.

1.2 Full stack web development

Full stack web development refers to the process of creating web applications that encompass both the front-end and back-end aspects. A full stack developer is proficient in working with all the technologies and layers involved in building a web application, including the client-side, server-side, and the underlying database.

The typical full stack application can be described by these layers:

- the "**Front-End**" (also known as "client-side"): is what users directly interact with through their web browsers; it includes the visual elements, layout, and interactive features that users see

and engage with. Front-end technologies commonly include HTML, CSS, and JavaScript, along with various front-end libraries and frameworks like Vue.js, React, or Angular.

- the "**Back-End**" (also known as "server-side"): is responsible for handling business logic, data processing, and communication with databases and external services. It manages requests from the client-side, processes them, and sends back the necessary data or responses. Back-end development involves working with programming languages like PHP, Go, Python, Node.js, Ruby, or Java, as well as frameworks such as Express, Django, or Spring.
- the **database layer** of a full stack application stores and manages the application's data. It could be a relational database like MySQL or PostgreSQL, a NoSQL database like MongoDB or Redis, flat files, or any combination of those.

In this book, we will go through these layers, and we will present some of these technologies, namely OpenAPI for API definition, Go for backend development, Vue.js for frontend development. We won't cover the database part,as there are multiple valid books on the topic.

Additionally, we will see how a full stack application can be deployed, and we will present the "containers" as deployment strategy.

1.3 Structure of a web application

There are several different approaches when dealing with web application. Regardless of programming languages and frameworks that are involved, you may have *server-side* or *client-side* rendered application.

In *server-side* rendering, the frontend code (JavaScript, HTML, CSS) is rendered on the server side. The frontend browser receives web pages already formatted and it draw them on the screen. Using this approach, the HTML code (and its generator) are embedded in the backend project, and there is no need for serialization and de-serialization of data. Responses may also contains partial web pages: their content replaces a portion of an already-shown web page. Requests may be synchronous or asynchronous (i.e., in background).

In *client-side* rendering, the backend and the frontend are two different applications. Data and message exchanged between the two applications are serialized and encoded in HTTP messages. The frontend is in change of preparing the HTML and CSS based on data and messages exchanged with the backend. Differently from the *server-side* rendering, the backend does not produce HTML: data are serialized to the frontend, which is expected to fully manipulate them. In this type of rendering, JavaScript is heavly used to request data from the backend and build UI elements. Requests are almost always asynchronous.

2 Version Control with Git

To introduce you to *version control*, let us tell you a story (based on `notFinal.doc` PHDcomics). Let's say that you are at the end of your final year at the university, and you are writing your final thesis. You write the document, and when you think that it's ready, you name it `final.doc`, and you send it to your supervisor/professor for revision and approval.

The professor reads your thesis, adds comments, and marks paragraphs of the text that you should rewrite. And they send back the document, named `final.comments.doc`. You eagerly start modifying your thesis, and you send it as `final_rev2.doc`. And then your professor sends back a `final_rev2.comments2.doc`. Now you send back `final_rev2.comments2.corrections1.doc`, and they send back… you get the point.

At the end of this back and forth, you end up with many files with inconsistent naming at best, hoping that the last created file is also the latest and greatest version.

This problem is named *versioning*, and it is common, especially when multiple people work on the same project, or the project lasts for years.

2.1 Version control systems

Version control systems (VCS) start from a set of needs every team or solo developer face after a few years. Projects grow in complexity and size; several changes are made by different collaborators/developers, and tracking and evaluating those changes is necessary. Sometimes, projects may have multiple versions being developed or tested concurrently (consider, for example, trying various optimizations to the same algorithm). And when the project is finally months/years old, there should be a way to track down when a specific feature or bug was introduced in the code.

Version Control Systems are now essential tools for modern software development and collaborative projects (even in low-code or no-code projects). They provide:

- Consistent labeling of revisions: multiple versions of a file are labeled using a standardized way (enforced by the tool);
- Tracking of changes: each difference between two revisions is tracked, and it can be recalled in the future (for example, if you need to explore an older version of the code);

- Metadata (date, authors, etc.) to revisions, to help, e.g., tracking down when and who made a particular change;
- Consistent **point-in-time revisions**: each revision is saved explicitly, and it may contain multiple files;
- Branches: revisions may diverge, and the same project may have multiple parallel "versions", clearly labeled;
- Merges: different branches may be "merged" by merging their changes;
- Synchronization between multiple users and computers.

Various applications provide these features, such as CVS, Subversion, Mercurial. We will look at Git, a modern solution that quickly became industry standard a few years ago. Note that most (if not all) concepts are shared between different version control systems.

2.2 Git

Git, a distributed version control system, was created in 2005 by Linus Torvalds to manage the source code of the Linux Kernel. It utilizes directed acyclic graphs (DAG) and data structures similar to Merkle trees. Thanks to its distributed nature and robust capabilities, Git has become the industry standard for version control. It enables developers to collaborate seamlessly, manage complex projects, and maintain a comprehensive historical record of code modifications.

We will introduce Git incrementally in the following sections.

2.2.1 Repository

A *repository* is a set of *commits*, *branches*, and *tags*, usually for the same project. We will define all these terms in the following sections; for now, think of the repository as a "bucket" which contains all Git-related items for a given project.

A project (for example, a mobile app) may be divided into multiple repositories for organizational reasons (for example, because some parts have a different life cycle). Nevertheless, for the sake of simplicity, in this book we assume that a repository contains a project and that a project is in one repository.

2.2.2 Working copy

The *working copy*, or *working directory*, is a local copy of the project. It is the set of files tracked by the version control system, and it's usually the directory that contains your project. It includes subdirectories as well. Each time you add, modify or delete a file, you need to tell Git about that.