Extracted from:

Web Development with Clojure

Build Bulletproof Web Apps with Less Code

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Web Development with Clojure

Build Bulletproof Web Apps with Less Code

Dmitri Sotnikov

edited by Michael Swaine

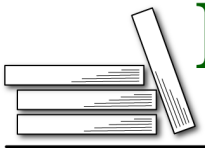
Web Development with Clojure

Build Bulletproof Web Apps with Less Code

Dmitri Sotnikov

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Michael Swaine (editor)
Potomac Indexing, LLC (indexer)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-64-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January 2014

Introduction

This book's cover has a bonsai tree on it. I chose it to represent elegance and simplicity, as these qualities make Clojure such an attractive language. A good software project is like a bonsai. You have to meticulously craft it to take the shape you want, and the tool you use should make it a pleasant experience. I hope to convince you here that Clojure is that tool.

What You Need

This book is aimed at readers of all levels. While having some basic proficiency with functional programming will be helpful, it's by no means required to follow the material in this book. If you're not a Clojure user already, this book is a good starting point, as it focuses on applying the language to solve concrete problems. This means we'll focus on a small number of language features needed to build common web applications.

Why Clojure?

Clojure is a small language that has simplicity and correctness as its primary goals. Being a functional language, it emphasizes immutability and declarative programming. As you'll see in this book, these features make it easy and idiomatic to write clean and correct code.

There are many languages to choose from and as many opinions on what makes any one of them a good language. Some languages are simple but verbose. You've probably heard people say that verbosity really doesn't matter, the argument being that when two languages are Turing complete, anything that can be written in one language can also be written in the other with a bit of extra code.

I think that's missing the point, however. The real question is not whether something can be expressed in principle. It's how well the language maps to the problem being solved. One language will let you think in terms of your problem domain while another will force you to translate the problem to its constructs.

The latter is often tedious and rarely enjoyable. You end up writing a lot of boilerplate code and constantly repeating yourself. There's a certain amount of irony involved in having to write repetitive code.

Other languages aren't verbose and they provide many different tools for solving problems. Unfortunately, having many tools does not directly translate into higher productivity.

The more features there are, the more things you have to keep in your head to work with the language effectively. With many languages I find myself constantly expending mental overhead thinking about all the different features and how they interact with one another.

What matters to me in a language is whether I can use it without thinking about it. When a language is lacking in expressiveness I'm acutely aware that I'm writing code that I shouldn't be. On the other hand, when a language has too many features I often feel overwhelmed or I get distracted playing with them.

To make an analogy with mathematics, having a general formula that you can derive others from is better than having to memorize a whole bunch of formulas for specific problems.

This is where Clojure comes in. It allows us to easily derive a solution to a particular problem from a small set of general patterns. All you need to become productive is to learn a few simple concepts and a bit of syntax. These concepts can then be combined in a myriad ways to solve all kinds of problems.

Why Make Web Apps in Clojure?

Clojure boasts tens of thousands of users; it's used in a wide range of settings, including banks and hospitals. Clojure is likely the most popular Lisp dialect today for starting new development. Despite being a young language, it has proven itself in serious production systems and the feedback from users has been overwhelmingly positive.

As web development is one of the major domains for using Clojure, several popular libraries and frameworks have sprouted in this area. The Clojure web stack is based on the Ring and Compojure libraries.[1,2] Ring is the base HTTP library, while Compojure provides routing on top of it. In the following chapters you'll become familiar with the web stack and how to use it effectively to build your web applications.

1. https://github.com/ring-clojure/ring
2. https://github.com/weavejester/compojure

There are many platforms for doing web development, so why should you choose Clojure over other options?

Well, consider those options. Many popular platforms force you to make trade-offs. Some platforms lack performance, others require a lot of boilerplate, and others lack the infrastructure necessary for real-world applications.

Clojure addresses the questions of performance and infrastructure by being a hosted language. The Java Virtual Machine is a mature and highly performant environment with great tooling and deployment options. Clojure brings expressive power akin to that of Ruby and Python to this excellent platform. When working with Clojure you won't have to worry about being limited by your runtime when your application grows.

The most common way to handle the boilerplate in web applications is by using a framework. There are many frameworks, such as Ruby on Rails, Django, and Spring. The frameworks provide canned functionality needed for building a modern site.

The benefits the frameworks offer also come with inherent costs. Since many operations are done implicitly, you have to memorize what effects any action might have. This opaqueness makes your code more difficult to reason about. When you need to do something that is at odds with the framework's design it can quickly become awkward and difficult. You might have to dive deep into the internals of the particular framework and create hacks around the expected behaviors.

So instead of using frameworks, Clojure makes a number of powerful libraries available, and we can put these libraries together in a way that makes sense for our particular project. As you'll see, we manage to avoid having to write boilerplate while retaining the code clarity we desire. As you read on I think you'll agree that this model has clear advantages over the framework-based approach.

My goal is to give you both a solid understanding of the Clojure web stack and the expertise to quickly and easily build web applications using it. The following chapters will guide you all the way from setting up your development environment to having a complete real-world application. I will show what's available, then guide you in structuring your application using the current best practices.