Extracted from:

Functional Web Development with Elixir, OTP, and Phoenix

Rethink the Modern Web App

The Pragmatic Bookshelf

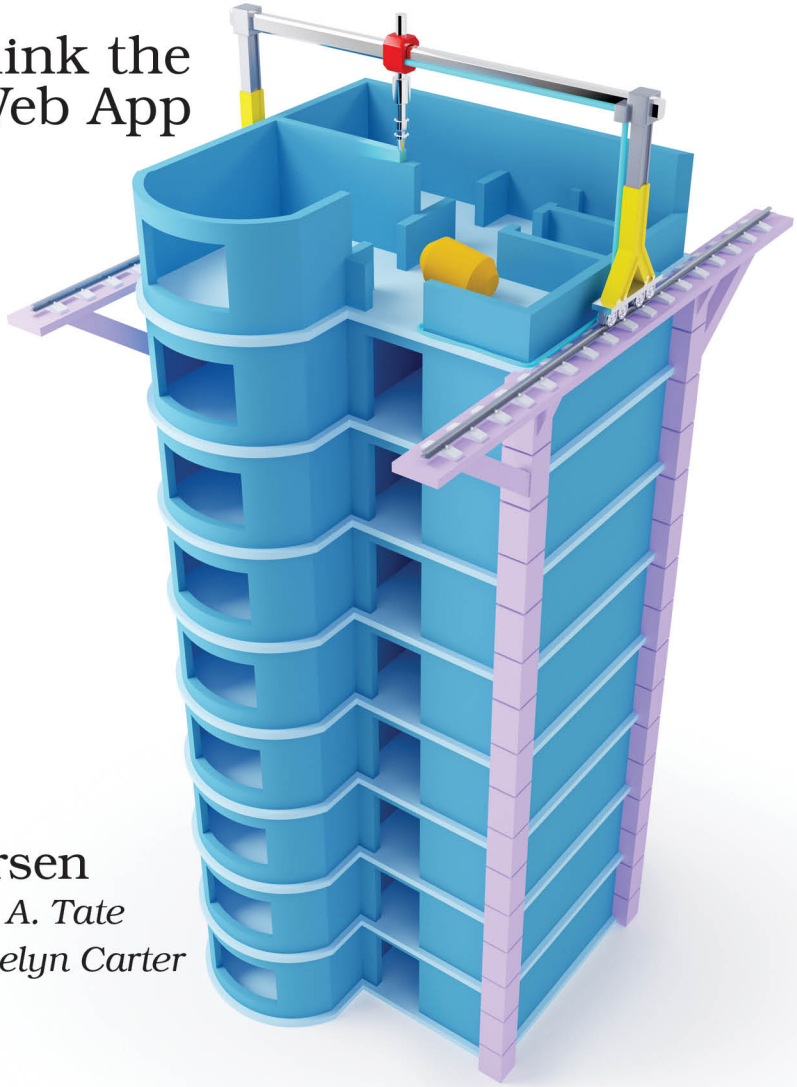
Raleigh, North Carolina

The Pragmatic Programmers

Your Elixir Source

Functional Web Development with Elixir, OTP, and Phoenix

Rethink the Modern Web App



Lance Halvorsen
Series editor: *Bruce A. Tate*
Development editor: *Jacquelyn Carter*

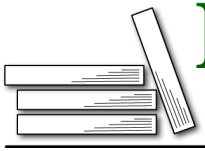
Functional Web Development with Elixir, OTP, and Phoenix

Rethink the Modern Web App

Lance Halvorsen

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Brian MacDonald
Supervising Editor: Jacquelyn Carter
Indexing: Potomac Indexing, LLC
Copy Editor: Liz Welch
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-243-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January 2018

Introduction

I've been building web applications for a long, long time, well before web frameworks existed. In those early days, a Perl script or two in an Apache cgi-bin directory and a couple of static HTML files were all you needed.

In the intervening years, web frameworks for both the front and back ends have become ubiquitous. This has been fantastic for the industry. The productivity gains are undeniable, and the consistency across applications allows us to onboard new developers more quickly.

But along with these gains have come some costs. The way we're commonly taught to use frameworks causes extremely tight coupling between the framework components and our business logic. Databases have gone from simply augmenting storage to dominating the way we model application domains.

Most people come to Elixir and Phoenix for the performance, and there's no question that the performance is fantastic. Personally, I see Elixir—especially the access it gives us to OTP—and Phoenix providing solid solutions to these long-standing problems without sacrificing any of the gains. I also see them opening up new possibilities—bringing back some of the benefits of statefulness that we've lost with the rise of HTTP.

Throughout this book we'll be building a web application together that shows what Elixir, OTP, and Phoenix can really do when used well together. But most of all, you'll have fun and learn things you can use to make the code for your day job or side projects more beautiful, easier to maintain, and a joy to work on.

Who This Book Is For

On a practical level, this book is for people who have some familiarity with Elixir and Phoenix, and who want to take that knowledge further. But there's a wider list for whom the ideas in this book will resonate.

For people who view OTP with a little trepidation, or for those who haven't quite mastered OTP Behaviours, this book will give you the confidence to use OTP in any application.

For people who have felt the sting of tight coupling between business logic and web frameworks, this book will show you a way out of that pain forever.

For people who feel constrained by traditional web development, you will learn new techniques and new ways to structure apps that will spark your imagination.

For people who are wondering what all the fuss is about with Elixir and Phoenix, you'll get a great taste of what makes people so excited. You just might become a convert!

Who This Book Is Not For

Readers looking for an introduction to Elixir or Phoenix would do well to begin with other resources.

We won't cover the basics of Elixir. I'll assume you know them before you begin.

If you need to get up to speed first, don't worry—we'll be here when you're ready. In the meantime, Dave Thomas's book, *Programming Elixir 1.3 [Tho16]*, is a great place to start.

The same is true for Phoenix. We will take a close look at channels and Presence, but you won't learn the rest of Phoenix here.

You should be able to follow along in this book without that information, but if you want to fill in the gaps, *Programming Phoenix [TV16]* by Chris McCord, Bruce Tate, and José Valim is the book to reach for.

About This Book

Throughout this book, we'll be building a game in distinct layers—from the bare essentials of the business logic to a web front end with stateful Phoenix channels.

The book is divided up into three sequential parts that parallel those layers. The first part lays the foundation, and each of the next two parts builds a new layer that depends on the one that came before.

If you're planning on implementing the game as you read, which is a great idea, you'll need to follow through the parts in order for the code to work.

If you're the sort of reader who likes to skip around, though, all is not lost. You can read the first few sections of any chapter—up until where we start to really implement the code—in any order, and they will still hold value.

Before we get to work in earnest, you'll read an overview of the whole book in Chapter 1, *Mapping Our Route*, on page ?.

Now let's see what each part has to offer.

Define the Functional Core in Elixir

We'll begin with only the most basic elements of Elixir—data structures, functions, and modules.

In Chapter 2, *Model Data and Behavior*, on page ?, we'll use data structures to model our domain entities. We'll define functions that work with these data structures to establish the business logic of the game. We'll also define modules to organize these functions and keep the code legible and easy to maintain.

In Chapter 3, *Manage State with a State Machine*, on page ?, we'll build a purely functional finite state machine to manage the game over time and enforce the rules. We'll proceed the same way we did in Chapter 2, with a data structure, multiple clauses of a single function, and a module to hold them all.

Add OTP for Concurrency and Fault Tolerance

This is where we'll introduce OTP to provide concurrency, parallelism, and fault tolerance.

In Chapter 4, *Wrap It Up in a GenServer*, on page ? we'll build a `GenServer` module to contain the business logic and state machine we built in Part 1. You'll learn how to spawn a new, long-lived process from this `GenServer` for each pair of players. That process will hold the state for their game as well as provide an interface to interact with it.

In Chapter 5, *Process Supervision for Recovery*, on page ? we'll explore how to make our game resilient to failures large and small. We'll build a supervisor to watch over each game process and restart it if it crashes. You'll also see how to restore a game process's state after a crash and even after the whole BEAM, Erlang's virtual machine, crashes or restarts.

Add a Web Interface with Phoenix

With all the work we've done in the previous two parts, we'll finally be ready to build a web interface with Phoenix.

In Chapter 6, *Generate a New Web Interface with Phoenix*, on page ? we'll create a new Phoenix project. You'll learn how OTP applications let us seamlessly integrate our work from the first two parts into this new Phoenix project as a dependency. Then we'll explore how we can call into that earlier work directly from different Phoenix components.

In Chapter 7, *Create Persistent Connections with Phoenix Channels*, on page ? we'll focus on the stateful, persistent connections that Phoenix provides called channels. You'll learn how to use JavaScript functions in a browser to communicate directly over a channel with a specific game process on the server. We'll also explore how to use Phoenix Presence to keep track of which players are actually playing an individual game.

Online Resources

The code we'll develop is available at the Pragmatic Programmers site for this book. There's also a community forum and errata-submission form for you to ask questions, report any problems with the text, or make suggestions for future versions.[1]

1. https://pragprog.com/book/lhelph/functional-web-development-with-elixir-otp-and-phoenix