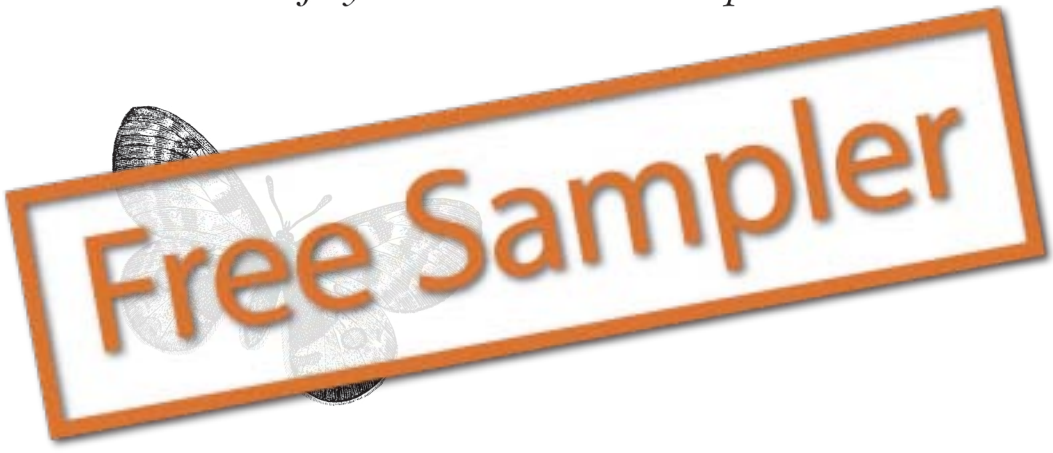
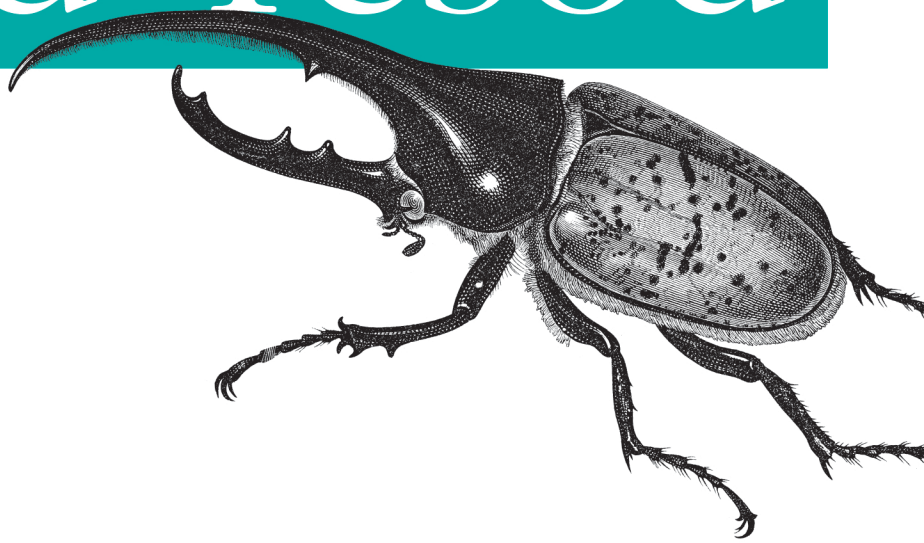
Safety-Driven Web Development



Developing Web Applications with

Haskell and Yesod



O'REILLY®

Michael Snoyman

Developing Web Applications with Haskell and Yesod
by Michael Snoyman

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Simon St. Laurent
Production Editor: Iris Febres
Proofreader: Iris Febres
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Revision History for the First Edition:
2012-04-20 First release
See *http://oreilly.com/catalog/errata.csp?isbn=9781449316976* for release details.

ISBN: 978-1-449-31697-6

[LSI]

1334948660

Table of Contents

Part III. Examples

Part IV. Appendices

CHAPTER 1
Introduction

Since web programming began, people have been trying to make the development process a more pleasant one. As a community, we have continually pushed new techniques to try and solve some of the lingering difficulties of security threats, the stateless nature of HTTP, the multiple languages (HTML, CSS, JavaScript) necessary to create a powerful web application, and more.

Yesod attempts to ease the web development process by playing to the strengths of the Haskell programming language. Haskell's strong compile-time guarantees of correctness not only encompass types; referential transparency ensures that we don't have any unintended side effects. Pattern matching on algebraic data types can help guarantee we've accounted for every possible case. By building upon Haskell, entire classes of bugs disappear.

Unfortunately, using Haskell isn't enough. The Web, by its very nature, is *not* type safe. Even the simplest case of distinguishing between an integer and string is impossible: all data on the Web is transferred as raw bytes, evading our best efforts at type safety. Every app writer is left with the task of validating all input. I call this problem *the boundary issue*: as much as your application is type safe on the inside, every boundary with the outside world still needs to be sanitized.

Type Safety

This is where Yesod comes in. By using high-level declarative techniques, you can specify the exact input types you are expecting. And the process works the other way as well: using a process of type-safe URLs, you can make sure that the data you send out is also guaranteed to be well formed.

The boundary issue is not just a problem when dealing with the client: the same problem exists when persisting and loading data. Once again, Yesod saves you on the boundary by performing the marshaling of data for you. You can specify your entities in a high-level definition and remain blissfully ignorant of the details.

Concise

We all know that there is a lot of boilerplate coding involved in web applications. Wherever possible, Yesod tries to use Haskell's features to save your fingers the work:

- The forms library reduces the amount of code used for common cases by leveraging the Applicative type class.
- Routes are declared in a very terse format, without sacrificing type safety.
- Serializing your data to and from a database is handled automatically via code generation.

In Yesod, we have two kinds of code generation. To get your project started, we provide a scaffolding tool to set up your file and folder structure. However, most code generation is done at compile time via meta programming. This means your generated code will never get stale, as a simple library upgrade will bring all your generated code up-to-date.

But for those who like to stay in control, and know exactly what their code is doing, you can always run closer to the compiler and write all your code yourself.

Performance

Haskell's main compiler, the GHC, has amazing performance characteristics, and is improving all the time. This choice of language by itself gives Yesod a large performance advantage over other offerings. But that's not enough: we need an architecture designed for performance.

Our approach to templates is one example: by allowing HTML, CSS, and JavaScript to be analyzed at compile time, Yesod both avoids costly disk I/O at runtime and can optimize the rendering of this code. But the architectural decisions go deeper: we use advanced techniques such as conduits and builders in the underlying libraries to make sure our code runs in constant memory, without exhausting precious file handles and other resources. By offering high-level abstractions, you can get highly compressed and properly cached CSS and JavaScript.

Yesod's flagship web server, Warp, is the fastest Haskell web server around. When these two pieces of technology are combined, it produces one of the fastest web application deployment solutions available.

Modular

Yesod has spawned the creation of dozens of packages, most of which are usable in a context outside of Yesod itself. One of the goals of the project is to contribute back to the community as much as possible; as such, even if you are not planning on using

Yesod in your next project, a large portion of this book may still be relevant for your needs.

Of course, these libraries have all been designed to integrate well together. Using the Yesod Framework should give you a strong feeling of consistency throughout the >various APIs.

A Solid Foundation

I remember once seeing a PHP framework advertising support for UTF-8. This struck me as surprising: you mean having UTF-8 support isn't automatic? In the Haskell world, issues like character encoding are already well addressed and fully supported. In fact, we usually have the opposite problem: there are a number of packages providing powerful and well-designed support for the problem. The Haskell community is constantly pushing the boundaries finding the cleanest, most efficient solutions for each challenge.

The downside of such a powerful ecosystem is the complexity of choice. By using Yesod, you will already have most of the tools chosen for you, and you can be guaranteed they work together. Of course, you always have the option of pulling in your own solution.

As a real-life example, Yesod and Hamlet (the default templating language) use `blaze-builder` for textual content generation. This choice was made because blaze provides the fastest interface for generating UTF-8 data. Anyone who wants to use one of the other great libraries out there, such as `text`, should have no problem dropping it in.

Introduction to Haskell

Haskell is a powerful, fast, type-safe, functional programming language. This book takes as an assumption that you are already familiar with most of the basics of Haskell. There are two wonderful books for learning Haskell, both of which are available for reading online:

- Learn You a Haskell for Great Good!
- Real World Haskell

Yesod relies on a few features in Haskell that most introductory tutorials do not cover. Though you will rarely need to understand how these work, it's always best to start off with a good appreciation for what your tools are doing. These are covered in the next chapter.