Yesod Web Framework

Michael Snoyman

QCon San Francisco 2011

What is Yesod

- Web framework
- Written in Haskell
  - Strongly typed
  - Pure/side-effect free
  - Fast
- Collection of libraries
- Full stack
  - Web server
  - Templating
  - ORM
  - Add-on libraries: everything from auth to gravatar
- Yesod (יסוד) means foundation in Hebrew

Brief history

- Started ~2.5 years ago by yours truly
- Went back to full-time web development
- Unhappy with existing options
    - Fan of static typing
    - Not a fan of Java
- Had used Haskell to save the day on a few projects at my previous job
- Decided to double-down on it
- Used it for a few contract jobs, great results

Used in the Real World®

- Through Suite Solutions:
  - Production Yesod site at Emerson (Social Knowledge Base)
  - Warp webserver powering Dell's context-sensitive help
  - Various Yesod libraries used at Cisco and LifeTech
- Three companies (that I know of) pushing Yesod-powered solutions to clients
- Suite Solutions sponsoring Yesod development
- Very active, friendly community, lots of them making sites too

Why Yesod?

- Evolutionary, not revolutionary
  - Follow standard practices (e.g., MVC)
  - Offer experimental options (e.g., MongoDB)
- Use compiler to avoid bugs
  - Type system fixes the "boundary issue"
  - Avoid things like XSS automatically
- Make it fast
  - High performance libraries under the surface
  - Simple, high-level API
- Encourage modularity (widgets, subsites, middleware)

Correctness

Type-safe URLs

- Datatype for all URLs in application
- All valid URLs can be expressed as a value
- Synchronized parse/render/dispatch functions
- Need four components to be aligned
  - Time for code generation (Template Haskell)
  - Want a simple syntax (QuasiQuotes)

Type-safe URLs: What you say

```
mkYesod "MyApp" [parseRoutes|
/ RootR GET
/blog/#BlogId BlogPostR GET
|]
```

Type-safe URLs: What you mean

```
data MyAppRoute = RootR | BlogPostR BlogId

renderMyAppRoute RootR = []
renderMyAppRoute (BlogPostR blogId) =
    ["blog", toSinglePiece blogId]

parseMyAppRoute [] = Just RootR
parseMyAppRoute ["blog", blogIdText] = do
    blogId <- fromSinglePiece blogIdText
    Just $ BlogPostR blogId
parseMyAppRoute _ = Nothing
```

Routing: Yesod vs Django

Django

```
urlpatterns = patterns('',
    (r'^articles/2003/$', 'news.views.special_case_2003'),
    (r'^articles/(\d{4})/$', 'news.views.year_archive'),
    (r'^articles/(\d{4})/(\d{2})/$', 'news.views.month_archive'),
    (r'^articles/(\d{4})/(\d{2})/(\d+)/$', 'news.views.article_detail'),
)
```

Yesod

```
/articles/2003 SpecialCase2003R
/articles/#Year YearArchiveR
/articles/#Year/#Month MonthArchiveR
/articles/#Year/#Month/#Day ArticleDetailR
```

Type-safe URLs: Why they matter

- Definition of paths in one place
- Automatic marshaling based on datatypes
- Change datatypes: compiler catches all errors

Example: try changing your URLs
/blog/5
/post/5
/post/2011/09/my-blog-post

Compile-time templates

- User-friendly syntax
- Syntax checked at compile time
- Use Haskell variables directly
    - No need for repetitious glue code
    - Types checked automatically
- Simple control structures for Hamlet
    - Basically logic-less...
    - Though you can get away with some logic
- Debug versions of CSS and JS
    - Quick development cycle

Hamlet (HTML)

```
!!!
<html>
    <head>
        <title>#{pageTitle} - My Site
        <link rel="stylesheet" href=@{StylesheetR}
    <body>
        <h1 .page-title>#{pageTitle}
        <p>Here is a list of your friends:
        $if null friends
            <p>Sorry, I lied, you don't have any friends.
        $else
            <ul>
              $forall friend <- friends
                <li>#{friendName friend} (#{show $ friendAge friend} years old)
        <footer>^{copyright}
```

Lucius (CSS)

```
section.blog {
    padding: 1em;
    border: 1px solid #000;
    h1 {
        color: #{headingColor};
    }
    background-image: url(@{MyBackgroundR});
}
```

Julius (Javascript)

```
$(function(){
    $("section.#{sectionClass}").hide();
    $("#mybutton").click(function(){
        document.location = "@{SomeRouteR}";
    });
    ^{addBling}
});
```

XSS Protection

- Html datatype
- ToHtml typeclass
- If you use textual type, entities escaped
- If you use an Html value, they aren't escaped
- Explicitly avoid escaping with preEscapedText
- OverloadedStrings extension makes it easy to type it in

XSS Protection: Example

```
name :: Text
name = "Michael <script>alert('XSS')</script>"

main :: IO ()
main = putStrLn $ renderHtml
    [shamlet|#{name}|]
```

Output:

```
Michael &lt;script&gt;alert(&#39;XSS&#39;)&lt;/script&gt;
```

Persistent

- Declare entity definitions once
- Automatically generate Haskell types, marshaling functions, and SQL schema
- Separate ID datatype for each table
- All marshaling and validity checking handled by library
- Automatic migrations
- Swap SQL and MongoDB easily.

Persistent: Declare entities

```
mkPersist [persist|
Person
    name String
    age Int Maybe
BlogPost
    title String
    authorId PersonId
|]
```

Persistent: CRUD

```
runMigration migrateAll

johnId <- insert $ Person "John Doe" $ Just 35
janeId <- insert $ Person "Jane Doe" Nothing

insert $ BlogPost "My fr1st p0st" johnId
insert $ BlogPost "One more for good measure" johnId

oneJohnPost <- selectList [BlogPostAuthorId ==. johnId] [LimitTo 1]
liftIO $ print (oneJohnPost :: [(BlogPostId, BlogPost)])

john <- get johnId
liftIO $ print (john :: Maybe Person)

delete janeId
deleteWhere [BlogPostAuthorId ==. johnId]
```

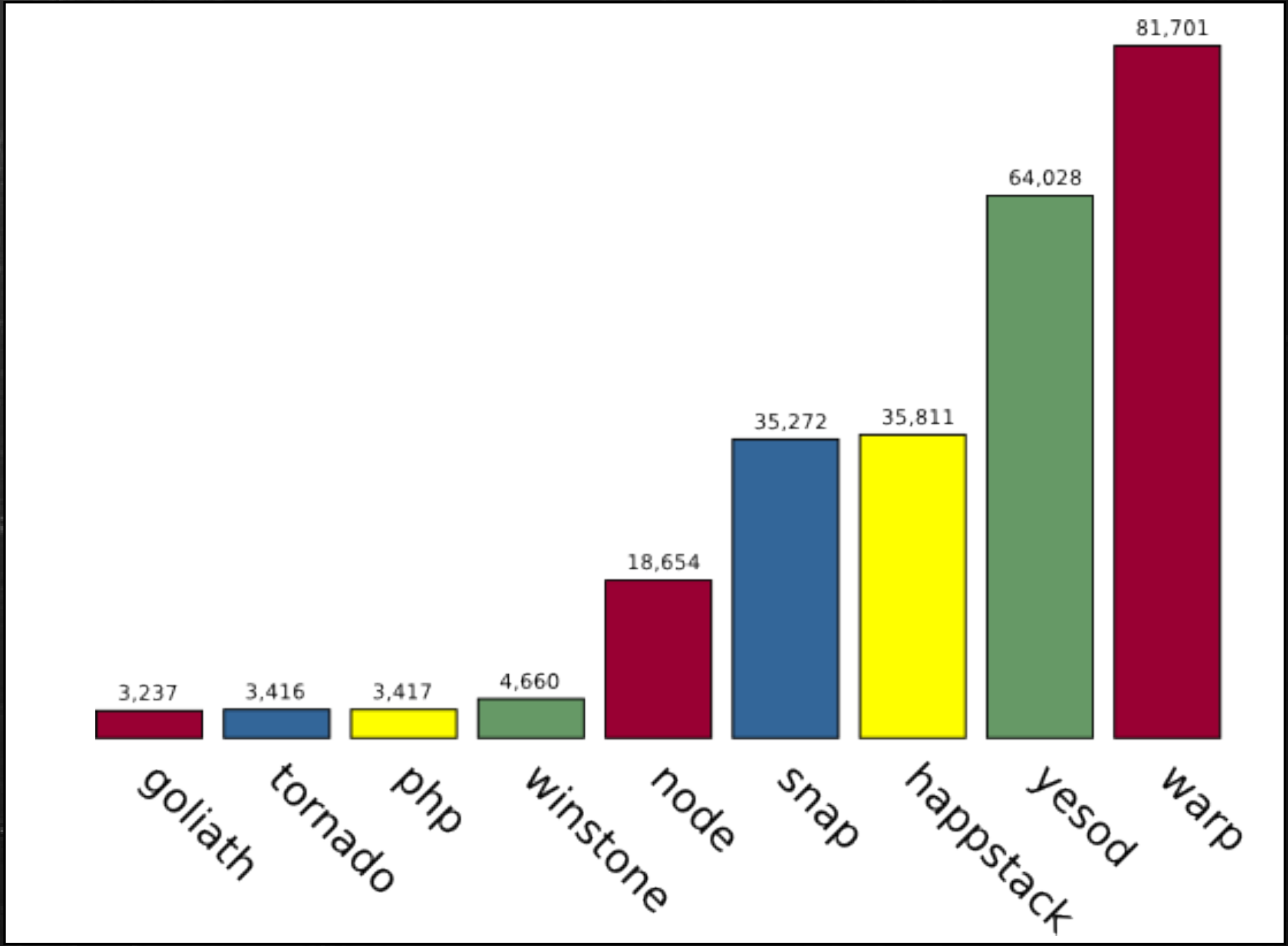
Performance

Web Application Interface (WAI)

- Low-level interface between web apps and servers
- Used by multiple frameworks, pioneered by Yesod
- Some apps use WAI directly without a framework
- Multiple backends, mostly Warp
- Built for performance an generality

Warp benchmarks

Benchmarks are *old*, haven't had a chance to update yet.



blaze-builder

- Think of StringBuilder from Java
- Efficiently fill up memory buffers
- Buffer filling action
  - Avoids extra buffer copies
- Keep a difference list of them
  - Diff list == O(1) append
  - Still a persistent data structure == cheap parallelism
- Optimal buffer size = minimal system calls
- Used through entire stack
  - Templates
  - Server

blaze-builder: Example

- Web server generates:
  - Status line
  - 4 response headers
- Application generates:
  - 3 response headers
  - HTML interspersed with 7 variables
- Result: (1 + 4 + 3 + 7 + 8 == 23) Builders
- Concatenated together
- They all copy to a single memory buffer
- Entire response == 1 system call

Enumerator

- Abstraction over data streams
- Complicated at first, simplifies many common activities
- Deterministic resource handling
- Easily combine different enumerator libraries
    - http-enumerator
    - persistent
    - xml-enumerator
    - warp
    - zlib-enum

Multi-threaded runtime

- Async programming is efficient, but difficult
- So pretend it's a sync API, and use async inside
- Light-weight threads
- Uses whatever system call the current OS supports
    - kqueue
    - epoll
- Persistent data structures == simple concurrency
- Warp uses no locks, timeout code uses a single lock-free shared memory access (atomicModifyIORef)

Haskell is fast

- GHC is industrial strength compiler
- Lots of development, lots of optimizations
- Actively developed constantly
- Exciting new routes like LLVM backend
- Performance comparable to Java
- Check out the programming language shootout

Modularity

Widgets

- Package up HTML, CSS and Javascript together
- Reuse same widget all over the place
- No need to remember to include CSS/JS separately
- Affect both <body> and <head> simultaneously
- Can perform database queries as well
- Example: recent posts component on multiple pages

Widgets: Example

```
existingLinks :: Widget
existingLinks = do
    links <- lift $ runDB
            $ selectList [] [LimitTo 5, Desc LinkAdded]
    toWidget [hamlet|
<ul .links>
    $forall (linkid, link) <- links
        <li>
            <a href=#{linkUrl link}>
                #{linkTitle link}
|]
    toWidget [lucius|.links { list-style: none } |]
    toWidgetHead [hamlet|
        <meta name=keywords content=links>|]
```

defaultLayout

- Define your site template
- Automatically used by special pages
    - Error responses (e.g., 404)
    - Subsites (e.g., login page)
- Defined in terms of widgets
- Growl, breadcrumbs

defaultLayout: Example

```
getAboutR :: Handler RepHtml
getAboutR = defaultLayout [whamlet|
<p>This is a simple application, pay it no heed.
^{existingLinks}
|]
```

Subsites

- Multiple routes, config data, all grouped together
- Due to defaultLayout, fits in with rest of site
- Used for:
    - Static files
    - Authentication
    - Admin site (work-in-progress)

Middleware

- At WAI level: can be used outside of Yesod
- Yesod turns on some middlewares by default
    - GZIP
    - JSON-P
    - Autohead

Other Goodness

Designer friendly

- Designers **like** Hamlet
  - "HTML done right"
- Routing file easy to understand
- More local error messages
- Immediate feedback (on compile) for bad HTML
  - But I'll admit, error messages aren't great

Users of Yesod

- Refugees from Rails/Django/PHP who already love Haskell
- Haskell programmers new to web development
- Web developers interested in trying out a functional language
- Even some people with neither Haskell nor web experience

Type-safe URLs: Fringe benefits

- Authorization
  - Single function for whole site
  - Pattern match to make sure we cover all cases
- Breadcrumbs
  - Define title and parent page for each route
  - Easily move around entire pieces of the site
  - Ties in nicely with defaultLayout (covered later)

GUI apps

- Writing cross-platform applications can be a pain
- Just make it a web app!
- TKYProf does just that
- wai-handler-launch and wai-handler-webkit
- Already in use in the real world

Everything else

- clientsession
- websocket/eventsource support
- devel server
- Scaffolded site
- Deploy to Heroku
- Third-party packages (goodies)
- First framework (?) with BrowserID support
- Only framework (?) with first-class MongoDB support

Questions?

More info at:

www.yesodweb.com