Comparing Haskell Web Frameworks

Beerend Lauwers
Augusto Martins
Frank Wijmans

{B.Lauwers, A.PassalaquaMartins, F.S.Wijmans}@students.uu.nl

Utrecht University, The Netherlands

January 27, 2012

Outline

Motivation

Web applications are everywhere. Advantages for using the web approach for your application:

- No installing
- Thin client
- Updates without required user actions
- Easy integration
- Platform free; needs web browser

Problem

Inherent problems of the web paradigm.

- HTTP Protocol
  - Stateless nature
- User input
  - Untyped
  - Security problems
- Error handling
- Servers are single point of failure
  - Denial-of-service

Clean: iData

- Clean, developed in Nijmegen.
- It allows saving partial computations, where it is different from Haskell.
- iData (for *i*nteractive Data) elements
  - Conversion HTML forms.
  - These elements can be saved on server, in html form or in session.
- Generic programming
  - Mapping types to forms and user input to types.

Haskell solution

Haskell brings some unique language features to the table, compared to imperative languages.

- **Type safe** user input and database connections, eliminating security issues from the user.
- Usually, websites depend on other languages: JavaScript, HTML/XML, CSS and SQL.

Web frameworks

Many frameworks cover (some subset of) the features needed for web applications.

1. **Happstack**
2. **Snap**
3. **Yesod**
4. Haskell on a Horse
5. miku
6. Lemmachine
7. mohws
8. Salvia (Sebastiaan Visser, Utrecht University)
9. Scotty

Lets see how the top three differ, and how they score on the different features.

Feature analysis

A web framework ought to provide several functionalities. Let's walk through them from our common point of view, that of a functional programmer.

Path routing

URL's can be static or dynamic. (= query string element)
Ideally, when the user requests a path the server returns:

- The user-requested page
- An error page (i.e. no credentials or incorrect user action)

$\rightarrow$ Use Haskell's type safety to route a request and generate its appropriate response

- **Happstack**: Concept of *guards*
  - *dir* and *dirs* (*ServerMonad* / *MonadPlus*)
  - Extract part of URL (*FromReqUri* class)
  - Request method (*MatchMethod* class)
  - Arbitrary function (*guardRq*)
  - Not fully type–safe
- **Snap**: Sum of its parts
  - Each Snaplet generates routes
  - Routes can point to sub–Snaplets / directories
  - Snap core unifies all routes
  - Validity of routes must be confirmed by programmer
- **Yesod**: built–in type–safe routing
  - Split URL at forward slashes
  - Attempt to match with sitemap
  - Look up proper *Handler* function
  - "short–circuiting" behaviour: escape a computation (*MEither*)

Path routing: Yesod example

Sitemap:

```
/user/#String  UserR  GET  POST
/faq           FaqR
```

Sitemap datatype:

```
data Route = UserR String | FaqR
```

Handlers:

```
handleFaqR :: Handler RepHtml
getUserR, postUserR :: Text → Handler RepHtml
```

- **web–routes**: external library
  - Map URL types to Strings and back
  - Highly flexible
  - Define datatype (e.g. *SiteMap*) representing all valid routes
  - *RouteT* monad transformer wraps around web server monad
  - Guards against incorrect URL's: *RouteT SiteMap*
  - route :: SiteMap → RouteT SiteMap (ServerPartT IO) Response
- **web–routes–boomerang**: extends **web–routes**
  - Single grammar for parsing and printing
  - Greater control over URL appearance while retaining maintainability

Serving static files

- **Happstack**: Highly granular control
  - Manipulate *Response* types (*FilterMonad*)
  - Escape current computation and return different *Response* (*WebMonad*)
  - *serveDirectory* and *serveFile* + guards from previous section
  - Roll your own functions with *Happstack.Server.FileServe.BuildingBlocks*
- **Snap**: Utility module
  - *Snap.Util.FileServe*
  - Automatic / Customized generation of directory indices
  - Dynamic MIME–type handlers (e.g. pretty–printing source code)
- **Yesod**: Minimal functionality, leaves static file serving to server implementation

HTML generation

- **Happstack**: **BlazeHtml**
- **Snap**: **Heist** (originally in–house)
- **Yesod**: **Hamlet** (originally in–house)

- **BlazeHtml**: Fast, combinator–based, compiled
- **Heist**: XML templating engine, runtime
- **Hamlet**: Quasi–quotation, type–safe, compiled
- **HSP**: Embedded XML, compiled
- **HStringTemplate**: Based on Java's StringTemplate library, compiled / runtime

- **happstack–plugins**: Automatic type–checking, recompilation and reloading of modules on a running server

JavaScript and CSS Generation

- **Happstack**: **JMacro** (JavaScript)
- **Snap**: None
- **Yesod**: *Cassius* (CSS), *Lucius* (CSS) and *Julius* (JavaScript) quasi-quoters
  - *Cassius*: Insertion of variables and URL's, whitespace mark–up
  - *Lucius*: Insertion of variables and URL's, standard CSS mark–up, CSS nesting
  - *Julius*: Insertion of variables, URL's, and templates
- **JMacro**: external library
  - Haskell values and –techniques
    - Lambda expressions: fun addTwo x → \y → x + y;
    - Haskell–style function application: var multThenAdd = add 2 (mult 2 3);
  - Automatic variable scoping: prevent overlap with variables of the same name
  - Antiquotation: use Haskell code directly within JavaScript code

Parsing request data and form generation

- **Happstack**
  - **Parsing request data**: Built–in
    - Extract from query string, GET and POST, cookies (*HasRqData* monad)
    - *look*, *lookCookie*, *lookRead*
    - Custom parsing and error messages with *checkRq*
    - Optional parameters: *optional* function from *Control.Applicative*
  - **Form generation**: **digestive–functors** library
- **Snap**
  - **Parsing request data**: Built–in
    - *rqCookies*, *rqQueryString*, *rqParams*
    - Modify parameters: *rqModifyParams*, *rqSetParam*
  - **Form generation**: **digestive–functors** library
- **Yesod**: Both parsing and form generation in **yesod–form**
  - Type–safe form generation
  - Form parsing into Haskell datatypes
  - JavaScript validation code generation
  - Applicative way of combining forms similar to **digestive–functors**

- **digestive–functors**: external library
  - Inspired by **Formlets** library
  - Combine simple form elements into more complex forms
  - *FormState* newtype: *State* monad that provides unique identifiers to each $<$ *input* $>$ tag

```
data Info = Person String String Int deriving (Show)
infoForm = User <$> inputText Nothing <*> inputText Nothing <*>
    inputTextRead "Could not parse value" (Just 25)

data Group = Persons Info Info Info deriving (Show)
groupForm = Persons <$> infoForm <*> infoForm <*> infoForm
```

Sessions and state handling

- **Happstack**: **happstack–auth** and **happstack–extra**
  - Both external libraries
  - Both simple session management
  - **happstack–extra**: Session ID saved in cookie, rest in–memory on server
- **Snap**: **snap-auth** and **mysnapsession**
  - **snap-auth**
    - Uses Yesod's **clientsession** as a base
    - Adds ability to save extra data in session cookie
  - **mysnapsession**: external library
    - Simple session management, supports in–memory sessions
    - Continuation–based programming model for multiple–request stateful interactions
- **Yesod**: **clientsession** (originally in–house)
  - Session data is stored in cookie
  - Message system
  - Integrates with **yesod–auth**'s ultimate destination design

Persistence

- **Happstack**: **acid–state** (originally in–house)
  - In–Memory ACID data store, non–relational
  - Can store arbitrary Haskell values
  - Write to file functions along with parameters
  - Restoring state = re–running logs
  - Uses **SafeCopy** for data migration and extension
  - Type–safe querying and updating
- **Snap**: Snaplets for **HDBC**, **mongoDB** and **acid–state**
- **Yesod**: **persistent** (originally in–house)
  - Define datatype, derive database-explainable version using TH
  - Can store arbitrary Haskell values
  - Databases supported out of the box: MongoDB, PostgreSQL, SQLite
  - Support for data migration and extension, dry runs
  - Generated primary key datatype enforces type–safe relational queries
  - Type–safe insert, update, delete queries written in Haskell:
  - persons ← selectList [Age ==. 22] [Asc Age]
  - Uniqueness constraints, default values, nullability, foreign keys

Persistence: acid–state example

Create datatype:

```
data CounterState = CounterState { count :: Integer } deriving (Eq,
    Ord, Read, Show, Data, Typeable)
$(deriveSafeCopy 0 'base ''CounterState)
```

Update function:

```
incCountBy :: Integer → Update CounterState Integer
incCountBy n =
 do c@CounterState{..}  ← get -- uses RecordWildCards extension
 let newCount = count + n
 put $ c { count = newCount }
 return newCount
```

Query function:

```
peekCount :: Query CounterState Integer
peekCount = count <$> ask
```

Ensure ACIDity:

```
$(makeAcidic ''CounterState ['incCountBy, 'peekCount])
```

makeAcidic generates the following:

```
data PeekCount  = PeekCount
data IncCountBy = IncCountBy Integer
instance IsAcidic CounterState where
 acidEvents = [ UpdateEvent (\(IncCountBy newState) → incCountBy
     newState), QueryEvent (\PeekCount → peekCount) ]
instance Method PeekCount where
 type MethodResult PeekCount = Integer
 type MethodState PeekCount = CounterState
instance QueryEvent PeekCount
instance Method IncCountBy where
 type MethodResult IncCountBy = ()
 type MethodState IncCountBy = CounterState
instance UpdateEvent IncCountBy
-- And more
```

Querying and updating:

```
-- P.S.: type EventState ev = MethodState ev
--       type EventResult ev = MethodResult ev
update' :: (UpdateEvent event, MonadIO m) ⇒ AcidState (EventState
    event) → event → m (EventResult event)
query'  :: (QueryEvent event , MonadIO m) ⇒ AcidState (EventState
    event) → event → m (EventResult event)
```

Example update:

```
c <- update' acid (IncCountBy 1)
```

Persistence: persistent example

Generate DB–explainable code:

```
mkPersist sqlSettings [persist|
Person
  name String
  age Int
|]
```

PersistEntity
PersistField PersistValue

Example query:

```
persons ← selectList [Age ==. 22] [Asc Age, LimitTo 10]
```

Type–safe foreign keys:

```
Car
  year Int
  owner PersonId
```

Authentication

- **Happstack**: **happstack-authenticate** (based upon **authenticate**)
  - **authenticate** features
  - Multiple authentication methods per account
  - Multiple personalities per user account
  - themeable **BlazeHtml**–based templates
- **Snap**: **snap–auth**
  - Basic user/password authentication functionality
- **Yesod**: **yesod–auth** (based upon **authenticate**, originally in–house)
  - **authenticate** features
  - E–mail, Google Mail (OpenID), HashDB
- **authenticate**: external library
  - BrowserId, Facebook Connect, Kerberos, OAuth, OpenID and rpxnow

Project management

- **Happstack**: No tools
- **Snap**: Some commands
  - Bare–bones skeleton
  - Simple, fully–working "Hello World!" example
- **Yesod**: Scaffolding tool
  - Asks user some questions
  - Generates cabal package with skeleton Yesod project

Happstack

Advantages:

- Flexible and complete server
- Using alternative packages is possible
- “Crash course” helps beginners
- Framework with most features

Happstack

Advantages:

- Flexible and complete server
- Using alternative packages is possible
- “Crash course” helps beginners
- Framework with most features

Disadvantages:

- Not the fastest server
- Might be too extensive for simple applications
- Lacks a project management tool

Snap

Advantages:

- Snaplets, which are easy to use
- Standalone libraries are easily integrated
- Well documented
- Good for smaller applications or applications with different module structure
- Scaffolding tool for management

Snap

Advantages:

- Snaplets, which are easy to use
- Standalone libraries are easily integrated
- Well documented
- Good for smaller applications or applications with different module structure
- Scaffolding tool for management

Disadvantages:

- Advanced features require external libraries
- Only a mid–level framework

Yesod

Advantages:

- Type safety in maximum form
- Easy to start (scaffolding, documentation in books and blogs)
- Packages are consistent; all from same developer
- Warp server, said to be the fastest
- Using Haskell's advanced language features extensively

Yesod

Advantages:

- Type safety in maximum form
- Easy to start (scaffolding, documentation in books and blogs)
- Packages are consistent; all from same developer
- Warp server, said to be the fastest
- Using Haskell's advanced language features extensively

Disadvantages:

- Alternative packages difficult to swap
- Comprehensive feature set can be expensive for small applications
- Abuse of language features

Conclusion and future work

All three frameworks contain different development philosophies:

1. Happstack allows the programmer to choose among a very diverse set of tools while providing a good feature set out of the box.

Conclusion and future work

All three frameworks contain different development philosophies:

1. Happstack allows the programmer to choose among a very diverse set of tools while providing a good feature set out of the box.
2. Snap provides a minimal system and does not impose any limitations, but also requires the programmer to resort to external libraries for most of the required functionality.

Conclusion and future work

All three frameworks contain different development philosophies:

1. Happstack allows the programmer to choose among a very diverse set of tools while providing a good feature set out of the box.
2. Snap provides a minimal system and does not impose any limitations, but also requires the programmer to resort to external libraries for most of the required functionality.
3. Yesod provides most of the usual functionality and tries to do so consistently, but this also makes it more difficult to divert from the given programming style.

Conclusion and future work

All three frameworks contain different development philosophies:

1. Happstack allows the programmer to choose among a very diverse set of tools while providing a good feature set out of the box.
2. Snap provides a minimal system and does not impose any limitations, but also requires the programmer to resort to external libraries for most of the required functionality.
3. Yesod provides most of the usual functionality and tries to do so consistently, but this also makes it more difficult to divert from the given programming style.

In general, though, most of the functionality implemented is framework–independent.

- Benchmark individual libraries per functionality to isolate and compare the cost of language features

Conclusion and future work (cont.)

- Many libraries available for HTML generation but few libraries for CSS and JavaScript generation
  - Happstack provides only a *JMacro* library wrapper, and Snap does not provide any CSS or JavaScript generation Snaplets.
- Snap and Happstack could also support type–safe URL routing like Yesod's or *web – routes*'
- Happstack could make good use of a project scaffolding and management tool (such as the ones found in both Yesod and Snap)
- More Snaplets available through Hackage
- Better control over static file sharing in the *warp* web server

Questions

Thank you!

Questions?