HAppS

Haskell's High Availability Application Server

alex@alexjacobson.com

http://happs.org

What do you want?

- Fast functional prototype
- Easy deployment
- Ability to scale
- Efficiency/Performance
- Ability to change

LAMP lumps

- marshalling i/o to/from language objects
- marshalling objects to/from relational store
- Consistency between layers
- inferior programming languages ;-)
- Deployment complexity (memcache too!)
- Disk vs Memory complexity

Prevayler Inspiration

- keep data in memory
- command pattern for accessing it...
- but checkpointing/consistency really hard

Why Haskell for this

- referential transparency for checkpointing
- referential transparency for loose consistency (optimistic concurrency)
- laziness for state versions
- scrap your boilerplate programming
- type system keeps your code together

HAppS

- develop in Haskell
- operate on your data in memory with ACID
- scrap your boilerplate
- all in one executable
- replicate to scale

HAppS-Data

- auto-convert data to/from XML
- auto-convert data to/from name-value pairs
- defaultvalue
- migrations
- soon: validation
- todo?: auto-convert types to/from XSD

HAppS-IxSet

- collection type for relational operation
- instead of ad hoc Data.Map & Data.Set
- (@< (Published t)) (@= (Author author)) books
- efficient update/lookup and laziness
- eventually perhaps query/update-fusion

HAppS-Server

- Fast HTTP/Fast-CGI application serving
- Nice interface for exposing app via HTTP
- XSLT templating system
- Facebook library
- operations in state aware I/O monad.
- Soon: SMTP-Relay?

HAppS-State

- Command Pattern for Haskell
- ACID guarantees on in memory haskell data structures
- important: all transactional state is in memory!
- Multimaster for availability and query scaling
- Soon: Sharding for memory and update

HAppS-Store

- FlashMsgs (like RoR)
- HelpReqs

HAppS-Begin

- Example of app from which you can modify
- All in one executable
- Simple Deployment model
- Type-checker keeps you good.
- Automatic re-compile/restart on code changes

Searchpath

- import chasing across the internet
- like -i but with URLs!
- no more iterative manual package installs
- no more package version conflicts
- nothing is global unless you want it

How to Build An App

- sketch on paper
- figure out data types you produce/consume
- figure out state you want to maintain
- define side-effects associated with workflows
- define HTTP interface you want to expose
- layout pages

Show the code

- demo app
- show types
- show http
- show state

Demo Code Behavior

- get form to enter help request
- post help request and arrive on page with message acknowledging help requests
- see help requests so far

The Types

- data HelpReqForm = HelpReqForm
- data HelpReq = HelpReq FB.Uid Published HelpText Status--post the help
- newtype HelpText = HelpText String
- data Status = Open | Closed Published
- newtype HelpFeed = HelpFeed [HelpReq]

The HTTP

```
 dir "help" [ method () $ ok $ HelpReqForm ]

,dir "addHelp" [withData $ \helpReq ->
                  [method () $ do
                              addHelpReq helpReq
                              insFlashMsg uid "Help message received"
                              fbSeeOther "side-nav"
              ]]

,dir "helps" [method () $ do
              flashMsg <- extFlashMsg uid
              helpReqs <- getHelpReqs --haskell is lazy so we can take 1000 below
              (ok .
                 insEl (Attr "context" "helpfeed") . --insert xml attributes
                 insEl (Attr "flashMsg" flashMsg) .
                 HelpFeed .
                 take 1000) helpReqs  ]
```

State

```
addHelpReq helpReq = withHelpReqs $
                     do
                     seconds <- getTime >>= return . (div 1000)
                     modify $ insert $ gSet (Published seconds) helpReq

getHelpReqs:: (HasHelpReqs st, MonadReader st m) => m [HelpReq]
getHelpReqs = (return . byRevTime . helpReqs) =<< ask

commands = ['addHelpReq,'getHelpReqs]
```

THANK YOU