Seminarvortrag

Sommersemester 2011, von Jan Martin

fhwedel
UNIVERSITY OF APPLIED SCIENCES

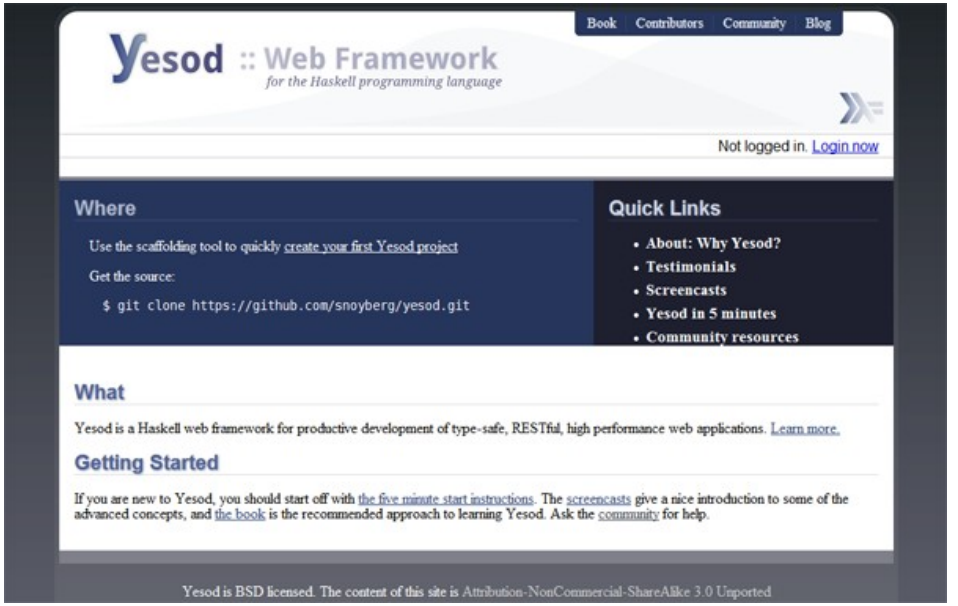
The Yesod Framework

Ein Webrahmenwerk in Haskell

Inhaltsverzeichnis

1. Einführung

- The Yesod Web
  http://www.yesodweb.com/
- Autor und Entwickler:
  Michael Snoyman
- Aktuelle Version: 0.8.2
- Yesod Web Framework Book
  - stellt die Dokumentation des Frameworks dar
  - befindet sich noch in der Entwicklung

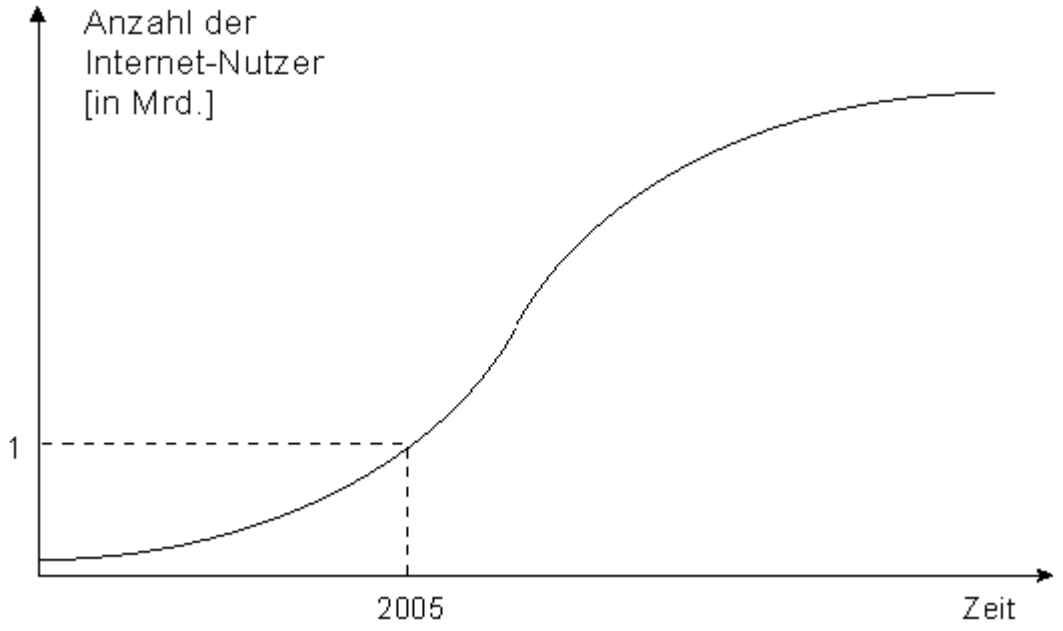


Yesod does not currently provide a prebuilt internationalization solution such as gettext. Instead, it provides some basic functionality for determining the user's language preference and leaves the rest to you. This chapter will give an example of how to use types to deal with i18n. We will additionally cover how to return responses in a character encoding besides UTF8. 0 comments

This chapter, however, has not yet been written. Please check back later. 0 comments

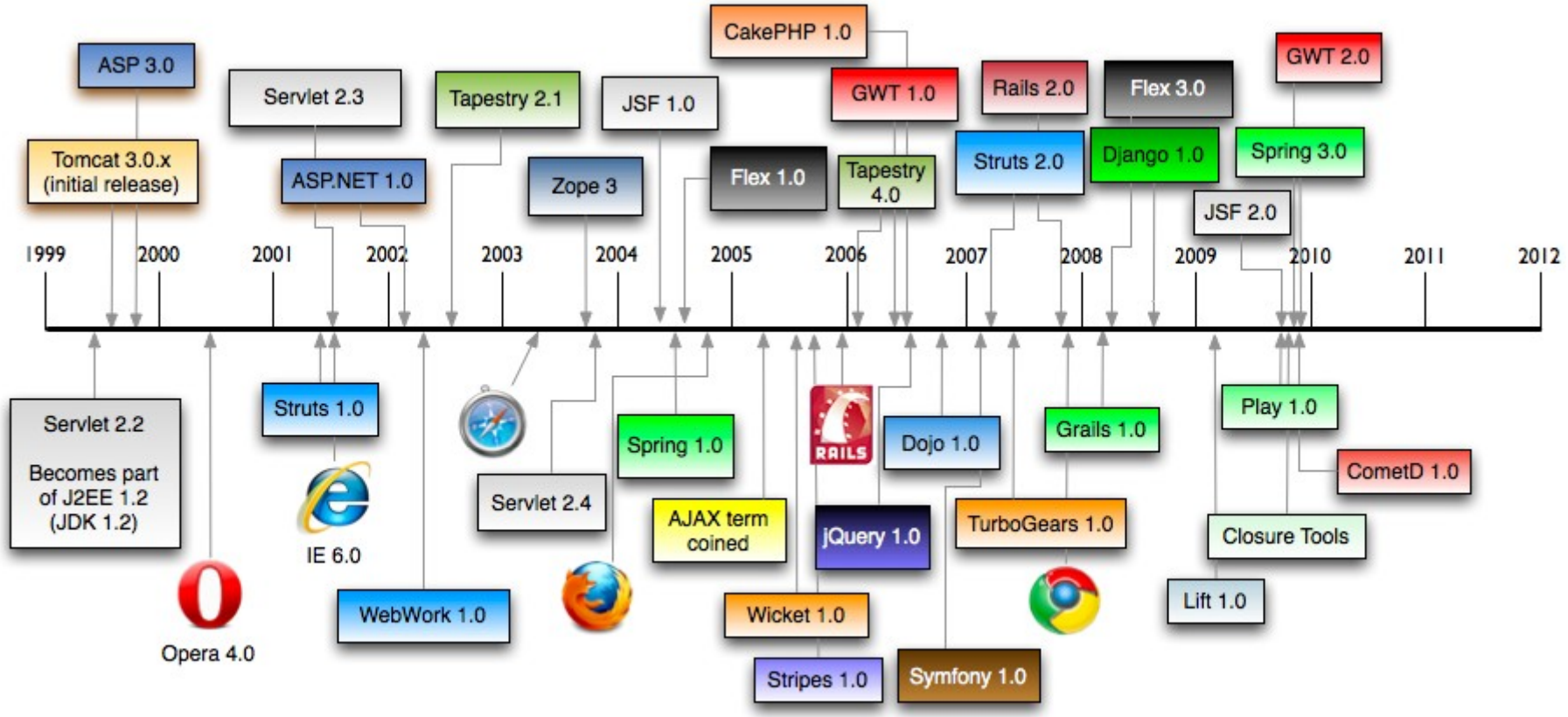
1. Einführung

- Rasanter Wachstum des Internets
- Anforderungen an Webanwendungen nehmen zu
- Webframeworks, um
  - Anforderungen gerecht zu werden
  - Entwicklungsprozess effizienter zu gestalten
  - nach Möglichkeit sehr schnell Ergebnisse zu erzielen
- Webframeworks als kritischer Erfolgsfaktor



1. Einführung

- Geschichte der Webframeworks



Quelle: Matt Raible, Raible Designs
http://raibledesigns.com/rd/entry/my_future_of_web_frameworks

1. Einführung

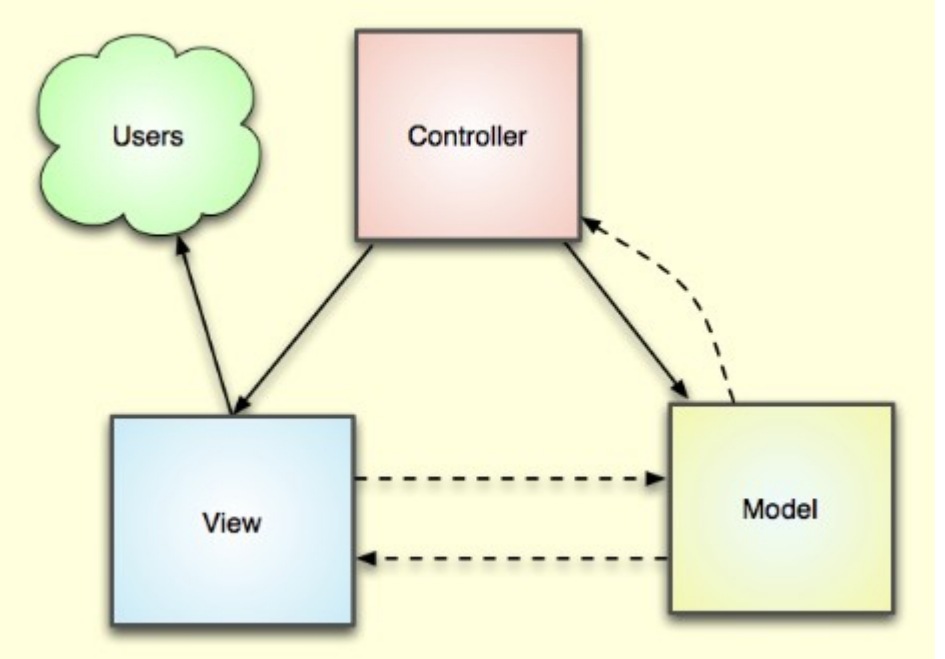
- Verbreitung serverseitiger Programmiersprachen

	2009 1 Oct	2010 1 Jan	2010 1 Apr	2010 1 Jul	2010 1 Oct	2011 1 Jan	2011 1 Apr	2011 4 Jun
PHP	71.1%	72.5%	73.9%	74.4%	74.9%	75.3%	75.9%	76.4%
ASP.NET	24.1%	24.4%	24.1%	24.0%	23.7%	23.4%	22.9%	22.5%
Java	3.2%	4.0%	4.0%	3.9%	3.8%	3.8%	3.8%	3.9%
ColdFusion			1.6%	1.5%	1.4%	1.3%	1.3%	1.3%
Perl			1.3%	1.2%	1.2%	1.1%	1.1%	1.1%
Ruby	0.5%	0.5%	0.5%	0.5%	0.5%	0.5%	0.6%	0.6%
Python	0.3%	0.3%	0.3%	0.3%	0.3%	0.3%	0.3%	0.2%

Quelle: W3Techs - World Wide Web Technology Surveys
http://w3techs.com/technologies/history_overview/programming_language/ms/q

2. Grundlagen

- Das Yesod-Framework
  - folgt dem MVC-Muster
  - bringt das starke Typsystem von Haskell überall dort ein, wo dies möglich ist
- Versprochene Eigenschaften resultierender Webanwendungen
  - sicher
  - RESTful
  - typsicher
  - schnell
  - skalierbar

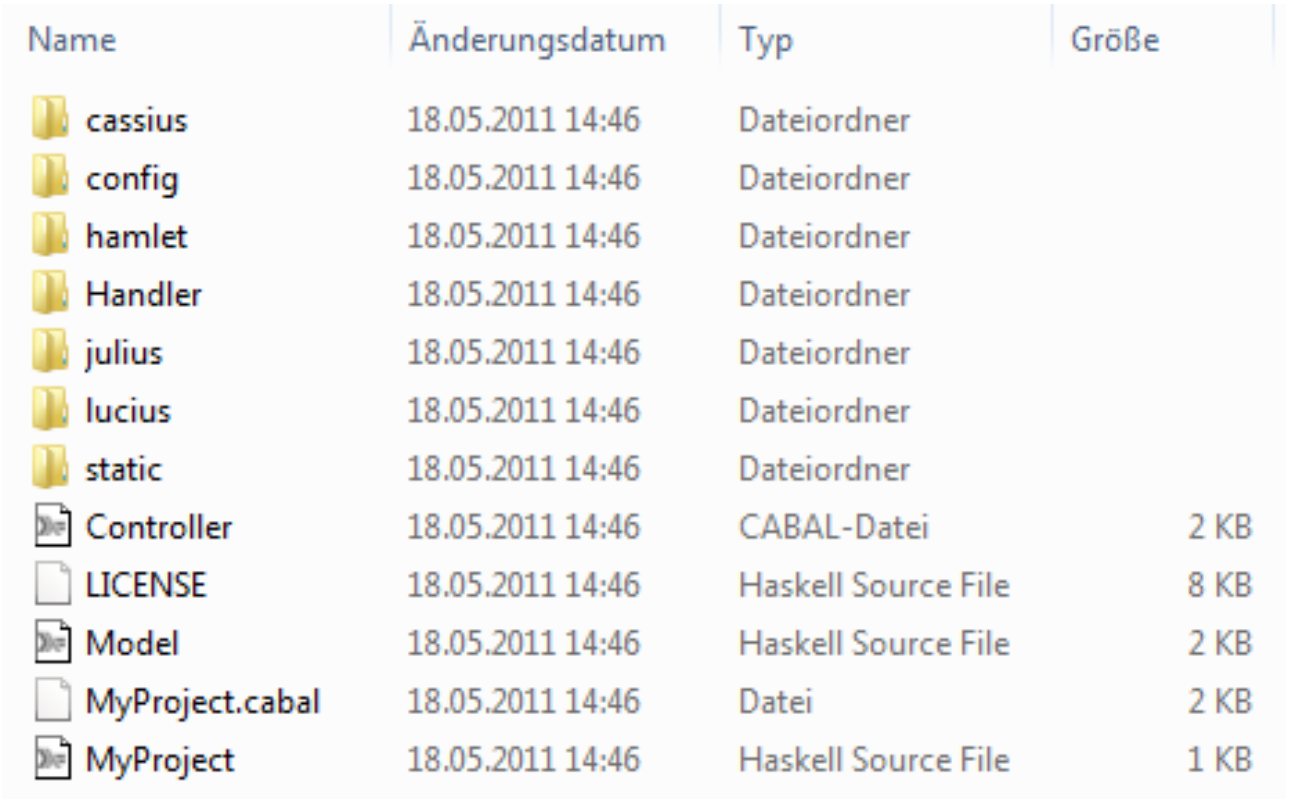


2. Grundlagen

- Installation
  - The Haskell Platform einrichten
  - Liste inkludierter Pakete mit `cabal update` aktualisieren
  - Yesod über `cabal install yesod` installieren
  - `yesod init` startet einen Assistenten zur Einrichtung eines neuen Projekts
  - zusätzliche Support-Bibliotheken können über den Befehl `cabal install --only-dependencies` installiert werden
  - `yesod devel` startet die Webapplikation

2. Grundlagen

- Installation



Name	Änderungsdatum	Typ	Größe
cassius	18.05.2011 14:46	Dateiordner	
config	18.05.2011 14:46	Dateiordner	
hamlet	18.05.2011 14:46	Dateiordner	
Handler	18.05.2011 14:46	Dateiordner	
julius	18.05.2011 14:46	Dateiordner	
lucius	18.05.2011 14:46	Dateiordner	
static	18.05.2011 14:46	Dateiordner	
Controller	18.05.2011 14:46	CABAL-Datei	2 KB
LICENSE	18.05.2011 14:46	Haskell Source File	8 KB
Model	18.05.2011 14:46	Haskell Source File	2 KB
MyProject.cabal	18.05.2011 14:46	Datei	2 KB
MyProject	18.05.2011 14:46	Haskell Source File	1 KB

fhwedel
UNIVERSITY OF APPLIED SCIENCES

- Einführendes Beispiel

```
1  {-# LANGUAGE QuasiQuotes, TypeFamilies, OverloadedStrings #-}
2  {-# LANGUAGE MultiParamTypeClasses, TemplateHaskell #-}
3  import Yesod
4
5  data Links = Links
6
7  mkYesod "Links" [parseRoutes|
8  / HomeR GET
9  /page1 Page1R GET
10 /page2 Page2R GET
11 |]
12
13 instance Yesod Links where
14     approot _ = ""
15
16 getHomeR  = defaultLayout [hamlet|<a href="@{Page1R}">Go to page 1!|]
17 getPage1R = defaultLayout [hamlet|<a href="@{Page2R}">Go to page 2!|]
18 getPage2R = defaultLayout [hamlet|<a href="@{HomeR}">Go home!|]
19
20 main = warpDebug 3000 Links
```

3. Konzepte bei Yesod

3.1. Templates

- Templatesysteme verarbeiten Zeichenketten, indem enthaltene Platzhalter mit Inhalten gefüllt werden
- Implementierung der View-Schicht
- Yesod verwendet dafür das Hamlet-Paket
- Unterschiedliche Templatesprachen
  - Hamlet für HTML
  - Cassius und Lucius für CSS
  - Julius für JavaScript

3.1. Templates

- Ausschnitt eines Hamlet-Templates

```
1 <div #an-id style=width:500px;>
2     <a href=@{MyDest}>I'm a link!
3     <ul .a-class>
4         $forall p <- people
5             <li>#{familyName p}, #{firstName p}
```

- Ziel ist die redundanzfreie Beschreibung des Markups
  - Struktur durch Einrückung
  - Anführungszeichen bei Attributwerten optional
  - Verkürzte Schreibweisen für ID- und Klassenangaben

3.1. Templates

- Interpolationen zur Füllung der Platzhalter mit Daten
- unterschiedliche Interpolationsvarianten
  - Variableninterpolation (#), Einfügen von Text, automatisches Escapen
  - URL-Interpolationen (@), URL-Kennungen im Rahmen der Beschreibung des Routingsystems festgelegt, String-Konvertierung durch Renderfunktion
  - Template-Interpolation (^), Einbindung weiterer Templates zwecks Modularisierung

```
1 <div #an-id style=width:500px;>
2     <a href=@{MyDest}>I'm a link!
3     <ul .a-class>
4         $forall p <- people
5             <li>#{familyName p}, #{firstName p}
```

3.1. Templates

- Cassius
  - zur Integration von Cascading Stylesheets
  - redundanzfreie Beschreibung durch Signifikanz der Einrückung

```
a
    color: darkgreen
h1
    border-bottom: 1px solid grey
```

- Lucius
  - Möglichkeit der Schachtelung von Stylesheet-Beschreibungen

```
foo, bar {
    baz, bin {
        color: red;
    }
}
```

=>

```
foo baz, foo bin, bar baz, bar bin {
    color: red;
}
```

3.1. Templates

- Anwendung von Templates

```
1     let name = "Michael" :: String
2     let nameId = "name" :: String
3
4     addHamlet [hamlet|
5 <h1>Hello World!
6 <p>
7     Welcome to my system. Your name is #
8     <span ##{nameId}>#{reverse $ take 2 $ reverse name}
9     . Enjoy your stay!
10 <p
11    <a href=@{BlogR "einstein" "relativity"}>general relativity
12 |]
```

- beispielhafte Routenkonfiguration

```
1 mkYesod "RouteExample" [parseRoutes|
2 / HomeR GET
3 /blog/#Author/#Title BlogR GET
4 |]
```

3.2. Widgets

- HTML, CSS und JavaScript bilden die drei Technologien im Frontend
- Kombinierung oftmals unhandlich
- Beschreibung an unterschiedlichen Stellen, es ergeben sich zwei Probleme
  - Einbinden globaler JavaScript- oder CSS-Beschreibungen nach Auszeichnung des `<head>`-Kontextes nicht mehr möglich
  - Aufteilung einer Webseite in einzelne Frontendmodule unpraktikabel, da sie einerseits global einsetzbar und andererseits mit anderen Modulen kombinierbar sein müssen
  - vgl. Integration von Frontendplugins, beispielsweise unter Nutzung von jQuery

3.2. Widgets

- Aufbau von Widgets
  - Titel der Webseite
  - externe Stylesheets
  - externe JavaScript-Dateien
  - CSS-Deklarationen innerhalb des Dokuments
  - JavaScript-Code innerhalb des Dokuments
  - beliebiger Inhalt innerhalb des `<head>`-Kontexts
  - beliebiger Inhalt innerhalb des `<body>`-Kontexts

3.2. Widgets

- Beispiel

```
1 getHomeR = defaultLayout $ do
2     setTitle "I was built by widgets"
3     addHtml [hamlet|<a>Hello Widgets!|]
4     addCassius [cassius|
5 a:hover
6     text-decoration: underline
7 |]
```

- Monaden-Semantik
- über `addWidget` können weitere Widgets eingebunden werden
- Fähigkeit, aus kleineren Widgets größere zusammenzusetzen

3.3. Routing & Handlers

- Routing
  - Definition gültiger Routen über den Front Controller
  - kein Routing anhand des Dateinamens
  - keine regulären Ausdrücke
  - Zwei-Wege-Verbindung zwischen Front Controller und Handler
  - Nutzung von Quasi-Quotations zur handlichen Konfiguration

```
1 mkYesod "RouteExample" [parseRoutes|
2 /home HomeR GET
3 |]
```

3.3. Routing & Handlers

- Drei unterschiedliche Angabemöglichkeiten von Routen
  - statisch
  - dynamisch, einzel
  - dynamisch, multi

```
1 mkYesod "RouteExample" [parseRoutes|
2 /home HomeR GET
3 /book/#Author/#Title BookR GET
4 /uploads/*Files FileR GET
5 |]
```

  - auch eigene Haskelltypen sind möglich
  - Konvertierungsfunktionen von und zu Text müssen dazu angegeben werden, ungültige Argumente sind mit dem Wert `Nothing` zu kennzeichnen

3.3. Routing & Handlers

- GHandler-Monade
  - bildet den Kontext während der Bearbeitung in Handlern
  - Reader zum Auslesen von Informationen aus der Anfrage
  - Writer zum Hinzufügen von Headerinformationen zur Antwort
  - Zustandskomponente verwaltet Sitzungsvariablen
  - Fehlerkomponente zum Auslösen von Fehlerantworten
  - durch Instanzierung von `monadIO` sind IO-Aktionen über `liftIO` durchführbar

```
data GHandler sub master a
```

3.3. Routing & Handlers

- GHandler-Monade

```
data GHandler sub master a
```

  - ohne Subsite wird stattdessen die Mastersite angegeben
  - Resultat eines Controllers ist üblicherweise `RepHtml` , kann aber für Klartext beispielsweise auch `RepPlain` sein
  - Beispiel für den Resultattyp eines Handlers, welcher im Rahmen einer Webanwendung „MyApp“ HTML zurückgeben soll

```
GHandler MyApp MyApp RepHtml
```

  - mit der Anwendung eines typischen Aliastyps

```
type Handler = GHandler MyApp MyApp
```

  - ergibt sich für den entsprechenden Handler der zweiten zuvor definierten Route die Signatur

```
getBookR :: Author -> Title -> Handler RepHtml
```

3.4. Formulare

- WWW ist nicht typisiert
- alle übertragenen Daten sind einfache Bytes
- Differenzierung von Typen wie Texten, Zahlen oder Booleschen Werten zunächst unmöglich
- Validierung von Nutzerdaten erforderlich
- Konvertierung in repräsentierende Haskelldatentypen
- Generierung von HTML und JavaScript

3.4. Formulare

- Anwendungsbeispiel
  - Implementierung eines Formulars zur Eingabe
    - einer Mindestanzahl,
    - einer Höchstanzahl,
    - und der Bezeichnungen für Singular
    - und Plural von Objekten
  - Anschließende Anzeige einer Nachricht darüber, wieviele Objekte gewählt wurden
  - Objektanzahl wird zufällig bestimmt und liegt zwischen den vorgegebenen Werten
  - Beispiel:
    Die Eingabe (1, 7, “Gurke“, “Gurken“) könnte die Ausgabe „You got 5 Gurken“ hervorrufen

3.4. Formulare

- Datentypdefinition der Formulardaten

```
1 data Params = Params
2     { minNumber :: Int
3     , maxNumber :: Int
4     , singleWord :: Text
5     , pluralWord :: Text
6     }
```

  - Plain-Old-Haskell-Datentyp
  - `Text` zur Typunterscheidung textueller Inhalte gegenüber allgemein übertragenen Daten im Format `String`

3.4. Formulare

- Beschreibung der Formularfelder

```
1 paramsFormlet :: Maybe Params -> Form s m Params
2 -- Same as: paramsFormlet :: Formlet s m Params
3 paramsFormlet mparams = fieldsToTable $ Params
4     <$> intField "Minimum number" (fmap minNumber mparams)
5     <*> intField "Maximum number" (fmap maxNumber mparams)
6     <*> stringField "Single word" (fmap singleWord mparams)
7     <*> stringField "Plural word" (fmap pluralWord mparams)
```

- Aufbau anhand von Funktoren mit der Applicative-Typklasse
- `intField` erzeugt z. B. ein Formularfeld für ganzzahlige Eingaben
- `fieldsToTable` generiert aus der Collection an Formularfeldern eine HTML-Repräsentation unter Nutzung von Tabellen

3.4. Formulare

- Handling

```
getRootR :: Handler RepHtml
getRootR = do
    (res, form, enctype) <- runFormGet $ paramsFormlet Nothing
    output <-
        case res of
            FormMissing -> return "Please fill out the form to get a result."
            FormFailure _ -> return "Please correct the errors below."
            FormSuccess (Params min max single plural) -> do
                number <- liftIO $ randomRIO (min, max)
                let word = if number == 1 then single else plural
                return $ T.concat ["You got ", T.pack $ show number, " ",  word]
    defaultLayout [hamlet|
<p>#{output}
<form enctype="#{enctype}">
    <table>
        ^{form}
        <tr>
            <td colspan="2">
                <input type="submit" value="Randomize!">
|]
```

3.5. Sessions

- Sessions können erst auf der Anwendungsschicht implementiert werden
- Sessionverwaltung sollte einfach und sicher sein
- ClientSession-Paket liefert Implementierung
- grundsätzlich zwei Konzepte
  - Verschlüsselung der übertragenen Sitzungsdaten
  - Beifügen einer Signatur zur Prevention von Hijacking-Attacken

3.5. Sessions

- Auslagerung der Datenhaltung an den Client
  - Sitzungsdaten befinden sich in einem Cookie
  - werden bei jeder Anfrage erneut versendet
- Drei Funktionen zur Steuerung der Funktionsweise von Sessions
  - `encryptKey` liefert den Schlüssel, welcher zur Verschlüsselung verwendet wird
  - `clientSessionDuration` bestimmt die Gültigkeitsdauer einer Sitzung
  - `sessionIpAddress` bestimmt, ob zur Steigerung der Sicherheit auch die IP-Adresse des Nutzers gespeichert werden soll

3.5. Sessions

- Verwaltung von Sitzungsdaten
  - Schreiben mit `setSession`
  - Lesen mit `lookupSession`
  - Löschen mit `deleteSession`
- Messages
- Ultimate Destinations
  - `setUltDest :: Route master -> GHandler sub master ()`
    definiert die übergebene typsichere URL als Ultimate Destination
  - `setUltDest' :: GHandler sub master ()`
    definiert die aktuell verarbeitete Route als Ultimate Destination
  - `setUltDestString :: String -> GHandler sub master ()`
    definiert die übergebene URL in Form einer Zeichenkette als Ultimate Destination

3.6. Persistenz

- Paket Persistent realisiert Anbindung an ein Speichersystem
- Abstraktion vom tatsächlich eingesetzten Datenbankbackend
- zahlreiche Bemühungen, Haskell-Werte on-the-fly in Datenbanken zu sichern
- Typsicherheit oberstes Gesetz
- Interaktion mit externem System
- Konzept der Migrations

3.6. Persistenz

- Beispiel
  - in einer Datenbank sollen Personendaten gespeichert werden

```
1 CREATE TABLE Person (
2     id PRIMARY KEY,
3     name VARCHAR NOT NULL,
4     age INTEGER
5 )
```

  - zugehörige Datentypdefinition in Haskell

```
1 data Person = Person
2     { personName :: String
3     , personAge :: Int
4     }
```

3.6. Persistenz

- Einfache praktische Anwendungen
  - Beschreibung der vorigen Datenbanktabelle in Yesod

```
1 mkPersist [persist|
2 Person
3     name String
4     age Int
5 |]
```

  - Einfügen und Auslesen eines Datensatzes

```
1 main = withSqliteConn ":memory:" $ runSqlConn $ do
2     runMigration $ migrate (undefined :: Person)
3     personId <- insert $ Person "Max Mustermann" 32
4     person <- get personId
5     liftIO $ print person
```

  - Konzept der Migrations/Umstellungen zur Anpassung des Datenbankschemas gemäß Konfiguration
  - Löschen von Attributen aus Sicherheitsgründen nur über gesonderte Funktion `runMigrationUnsafe` möglich

3.6. Persistenz

- Attribute
  - Beschreibung zusätzlicher Attribute für die Felder einer Entität
  - Beispiele
    - `Maybe` -Attribut für optionale Felder
    - `default` zur Angabe von Standardwerten
  - Assoziierte Typen
    - `Eq, Ne, Lt, Le, Gt, Ge, In`
    - Vergleichsoperatoren als wohldefiniert beschreiben

```
1 mkPersist [persist|
2 Person
3     name String Eq
4     age Int Lt
5 |]
```

  - automatische Einführung eines Filters

```
Filter Person = PersonNameEq String | PersonAgeLt Int
```

3.6. Persistenz

- Attribute

```
1 main = withSqliteConn ":memory:" $ runSqlConn $ do
2     runMigration migrateAll
3     persons <- selectList
4         [ PersonNameEq "Tom"
5         , PersonAgeLt 18
6         ]
7         [] 0 0
8     liftIO $ print persons
```

- Relationen

```
1 mkPersist [persist|
2 Person
3     name String
4 Hobby
5     name String
6 PersonHobby
7     person PersonId
8     hobby HobbyId
9     UniquePersonHobby person hobby
10 |]
```

4. Vergleich zu anderen Webframeworks

- Andere Webframeworks in Haskell
  - Happstack
    - optional zu verwendenes Speichersystem ohne SQL
    - viele unterschiedliche Template-Engines einsetzbar
  - Snap
    - nur für UNIX-Systeme
    - gut dokumentiert, aber noch am Anfang der Entwicklung
  - weitere Webframeworks in Haskell
    - Haskell on a Horse
    - Lemmachine
    - Salvia

4. Vergleich zu anderen Webframeworks

- Vergleich zu dynamischen Sprachen
  - Ruby on Rails
    - „Don't Repeat Yourself“
    - „Convention Over Configuration“
    - dynamische Typisierung von Ruby
  - Zend
    - zahlreiche Möglichkeiten
    - dynamische Typisierung von PHP

4. Vergleich zu anderen Webframeworks

- Vergleich zu dynamischen Sprachen
  - Vorzüge der Mainstream-Lösungen
    - weite Verbeitung
    - hohe Unterstützung
    - viele Alternativen zur Lösung von Standardaufgaben
  - Vorzüge des Yesod-Frameworks
    - starke Typisierung durch Haskell
    - konsequente Typsicherheit im Zusammenspiel mit externen Systemen
    - Fehlerreduzierung
    - komfortable Konfiguration durch Quasi Quotations
    - elegantes Zusammenführen der einzelnen Frontendtechnologien anhand von Widgets

5. Fazit

- Zusammenfassung
  - Yesod richtet sich an Entwickler,
    - die für die Webentwicklung Haskell einsetzen möchten
    - denen die Ansprüche der gängigen Webframeworks nicht genügen
  - Yesod ist ein MVC-Webframework
  - Installation unkompliziert
  - Typsicherheit in allen Ebenen

5. Fazit

- weitere Aufgabenbereiche
  - Authentifizierung bzw. Autorisierung
  - Versenden von E-Mails
  - Testen der Webapplikation
  - Multilingualität
  - Deployment
- Zukunftsaussichten
  - Weiterentwicklung in vollem Gange
  - Gesamtpaket angebotener Möglichkeiten überzeugt
  - Marktstellung ungünstig, Nischenprodukt

Vielen Dank für die Aufmerksamkeit!