A short introduction to

a web app written in Yesod

Funky Foto

- An app to transform images online using custom Accelerate code
- Core technologies:
  - Yesod
  - hs-plugins
  - Accelerate

Type Safety

- Type safe URLS
- Type safe persistence layer
- type-checked HTML/CSS/Javascript

Type safe URLs

```
mkYesodData "Foundation" [$parseRoutes|

/effects                        ListEffectsR        GET
/effects/create                 CreateEffectR       POST PUT
/effects/#String/show           ShowEffectR         GET
/effects/#String/edit           EditEffectR         GET
/effects/#String/update         UpdateEffectR       POST PUT
/effects/#String/delete         DeleteEffectR       POST DELETE
/effects/#String/run            RunEffectR          GET
/effects/#String/result         ResultEffectR       POST

|]
```

Type-safe URLs

```
data Foundation = ListEffectsR
                | CreateEffectR String
                | ShowEffectR   String
                | ...
```

Hamlet

```
%ul.breadcrumb
  %li
    %a!href=@ListEffectsR@ Return to list &raquo;
%h1 Run Effect / $
  %strong $name$
```

```
%div.add-effect
  %h2 Upload a JPEG image
  %form!method=POST!action=@ResultEffectR name@ !enctype=multipart/form-data
    %input!type=file!name=file
    %button.get-funky GET FUNKY!
```

```
%div.controls
  %a!href=@ShowEffectR name@ Show
  \ | $
  %a!href=@EditEffectR name@ Edit
```

Type safe URLs created from data structure!

Handlers

This line for Foundation

```
/effects/#String/show                ShowEffectR                GET
```

Means that Yesod expects you to define:

```
getShowEffectR :: String -> Handler RepHtml
```

Handlers

```
getListEffectsR :: Handler RepHtmlJson
getListEffectsR  = do
  -- TODO: For now just return all effects. Pagination to come.
  compilesParam <- lookupGetParam "compiles"
  let effectFilter = maybe [] (\val -> if val == "yes" then [EffectCompilesEq True] e
  results <- runDB $ selectList effectFilter [EffectNameAsc] 1000 0
  let effects = map snd results
  (_, form, encType, csrfHtml) <- runFormPost $ createFormlet Nothing
  let newForm = $(widgetFile "effects/new")
      canCancel = False
      info = information ""
  let json = jsonList (map (jsonScalar . effectName) effects)
  defaultLayoutJson (addWidget $(widgetFile "effects/list")) json
```

Handlers

```haskell
getListEffectsR :: Handler RepHtmlJson
getListEffectsR  = do
  -- TODO: For now just return all effects. Pagination to come.
  compilesParam <- lookupGetParam "compiles"
  let effectFilter = maybe [] (\val -> if val == "yes" then [EffectCompilesEq True] e
  results <- runDB $ selectList effectFilter [EffectNameAsc] 1000 0
  let effects = map snd results
  (_, form, encType, csrfHtml) <- runFormPost $ createFormlet Nothing
  let newForm = $(widgetFile "effects/new")
      canCancel = False
      info = information ""
  let json = jsonList (map (jsonScalar . effectName) effects)
  defaultLayoutJson (addWidget $(widgetFile "effects/list")) json
```

Template Haskell here expects variables here

```
%h1 Effects / $
  %strong go ahead, try one.
%ul.effects
  $forall effects effect
    %li.effect
      %div
        %form!method=POST!action=@DeleteEffectR (effectName effect)@
          %a!href=@RunEffectR (effectName effect)@
            %img!src=@((PreviewImageR Thumb) (effectName
effect))@!width="180"!height="180"
          %h3 $effectName effect$
          %span.controls
            %a!href=@ShowEffectR (effectName effect)@ Show
            \ | $
            %a!href=@EditEffectR (effectName effect)@ Edit
            \ | $
            %a!href=@RunEffectR (effectName effect)@ Run
            \ | $
            %input.custom!type=submit!value=Delete!onclick="return confirm('Are you
sure?');"
^newForm^
```

```
$info$
%div.add-effect
  %form!enctype=$encType$!action=@CreateEffectR@!method=POST
    $csrfHtml$
    ^form^
    %div.special-button-container
      %button.add Add
$if canCancel
  %a.small!href=@ListEffectsR@ Cancel
```

Persistence

This:

```
share2 mkPersist (mkMigrate "migrateAll") [$persist|
Effect
    name        String  Eq Asc     -- an effect name. Unique.
    code        String  Update     -- the associated Accelerate code.
    compiles    Bool    Eq Update --
    UniqueEffect name
|]
```

Generates this:

```
data Effect = Effect { effectName :: String
                     , effectCode :: String
                     , effectCompiles :: Bool }
```

Persistence

Example use

Create

```
effectKey <- runDB $ insert (Effect name defaultEffectCode True)
```

Read

```
results <- runDB $ selectList effectFilter [EffectNameAsc] 1000 0
```

```
mbResult <- runDB $ do { getBy $ UniqueEffect name }
```

Update

```
runDB $ replace key (effect {effectCompiles = False})
```

Delete

```
runDB $ deleteBy $ UniqueEffect name
```

Persistence

- Don’t use SQL only persistence library
- Many backends
  - SQLite, MongoDB, etc
- At time of writing
  - no joins done in database

Interactive development!

- Just like Rails
  - Edit, Refresh cycle
- Uses

Other things

- MVars are great
- hs-plugins is a dream