aula: political participation in schools

Matthias Fischmann <mf@zerobuzz.net>,
Andor Penzes <ap@zerobuzz.net>

HaL2016
Figure 1: idea lists
Figure 2: idea lists
pains explicabo high tempore from

Von SGKBNJB / 5 Quorum-Stimmen / 30 Verbesserungsvorschläge

5 von 47 Quorum-Stimmen

AUF DEN TISCH! ✓ DURCHFÜHRBAR ❌ NICHT DURCHFÜHRBAR

✓ STATEMENT ABGEBEN

eror annoying sapiente ever will do all quisquam officis great blanditiis aut foresee est

vero from last sunt holds aliqua so in eos are autem dignissimos quibusdam fugiat officia these except autem eu by chooses righteous so first extremely again you he similique quia

Diese Idee gehört zu keiner Kategorie

30 Verbesserungsvorschläge

NEUE VERBESSERUNGSVORSCHLAG

Figure 3: details of an idea
Figure 4: discussion of one idea
Figure 5: voting
Figure 7: user profile
Figure 8: delegations
the aula story

<table>
<thead>
<tr>
<th>concept</th>
<th>politik digital e.V.</th>
</tr>
</thead>
<tbody>
<tr>
<td>implementation</td>
<td>liquid democracy e.V.</td>
</tr>
<tr>
<td>funding</td>
<td>Bundeszentrale für politische Bildung</td>
</tr>
</tbody>
</table>

- implementation start in Feb’16
- production in Aug’16 (school year 2016/17)
- license: AGPL https://github.com/liqd/aula/
software: choices

building:
- ghc (7.10.2)
- cabal, stack
- docker (sometimes)

testing:
- hspec
- sensei, seito

libraries:
- HTTP request processing with servant
- multi-page app with lucid
- web forms with digestive-functors
- persistence with acid-state
usually servant is used to deliver JSON, but HTML works fine!

- define one page type for every end-point
- (newtype if needed)

For every page, define data \( P \) with

- \( \text{handler :: } \ldots \rightarrow m \ P \)
- \( \ldots : > \text{Get } P \) (or Post, or FormHandler) (servant route)
- \( \text{instance ToHtml } P \) (html rendering)
- [more stuff for HTML forms]
data PageOverviewOfSpaces =
   PageOverviewOfSpaces [IdeaSpace]

instance ToHtml PageOverviewOfSpaces where
    toHtml (PageOverviewOfSpaces spaces) =
    div' [class_ "container-main grid-view"] $
      ideaSpaceBox `mapM_` spaces

where
    ideaSpaceBox :: forall m. (Monad m) => IdeaSpace -> HtmlT m ()
    ideaSpaceBox ispace = div_ [class_ "col-1-3"] $ do
      div_ $ do
        a_ [href_ ...] $ do
          span_ [class_ "item-room-image"] $ mempty
          h2_ [class_ "item-room-title"] $ uilabel ispace
(blaze)

- faster
- not a monad (bind is not defined for performance reasons)
- slightly less nice syntax
type AulaMain =
  "space" ::> Get PageOverviewOfSpaces
  -- /space
  <|> "space" ::> Capture IdeaSpace
  :> "ideas" ::> Query ... 
  :> Get PageOverviewOfWildIdeas
  -- /space/7a/ideas?sort-by=age
...

aulaMain :: forall m. ActionM m => ServerT AulaMain m
aulaMain =
  (... :: m PageOverviewOfSpaces)
  <|> (\space query -> ... :: m PageOverviewOfWildIdeas)
...
data PageOverviewOfSpaces =
    PageOverviewOfSpaces [IdeaSpace]

instance ToHtml PageOverviewOfSpaces where
    toHtml (PageOverviewOfSpaces spaces) =
        ideaSpaceBox <$> spaces

where
    ideaSpaceBox :: forall m. (Monad m) => IdeaSpace -> HtmlT m ()
    ideaSpaceBox ispace = div_ $ do
        div_ . a_ [href_ ...] . span_ $ mempty
... 

ideaSpaceBox :: forall m. (Monad m) => IdeaSpace -> HtmlT m () 
ideaSpaceBox ispace = div_ $ do 
  let uri = "/space/" <> uriPart ispace <> "/ideas"
  div_ . a_ [href_ uri] . span_ $ mempty

- hard to hunt for broken URLs
- hard to track changes
module Frontend.Path

data Main =
  ListSpaces
  | Space IdeaSpace (Space r)
  ...

data Space =
  ...
  | ListIdeasInSpace (Maybe IdeasQuery)
  ...

listIdeas :: IdeaLocation -> Main
listIdeas loc =
  Main . Space spc . ListIdeasInSpace $ Nothing
module Frontend.Page

main :: Main -> String -> String
main ListSpaces root = root </> "space"
main (Space sid p) root = ...
ideaSpaceBox :: forall m. (Monad m) => IdeaSpace -> HtmlT m ()
ideaSpaceBox ispace = div_ $ do
  let uri = P.listIdeas (IdeaLocationSpace ispace)
  div_ . a_ [href_ uri] . span_ $ mempty

- Automatic testing: “every path has a handler”
- Changes in URI paths only have one location
- Harder in html template languages!
Is there a function that computes paths from page types?

uriPath :: <routing table>
    -> <page type>
    -> <variable path segments and URI query ...>
    -> String

(would require dependent types)
we have started off with digestive-functors and explored how this fits in with our approach.

the code i am showing you now is from an upcoming general-purpose package (watch out for news in the aula README).

if it doesn’t compile, revert to aula!
instance FormPage DiscussPage where

formPage v form (DiscussPage _) =
  html_ . body_ . div_ $ do
  h1_ "please enter and categorise a note"
  form $ do
    label_ $ do
      span_ "your note"
      DF.inputText "note" v
    label_ $ do
      span_ "category"
      DF.inputSelect "category" v
    footer_ $ do
      DF.inputSubmit "send!"

...
Forms (2)

```
makeForm (DiscussPage someCat) = DiscussPayload
  <$> ("note" .: validateNote)
  <*> ("category" .: catChoice)

where
  validateNote :: Monad m => Form (Html ()) m ST.Text
  validateNote = DF.text Nothing

  catChoice :: Monad m => Form (Html ()) m Cat
  catChoice = DF.choice
    (((\c -> (c, toHtml c)) <$> [minBound..]) <$> [minBound..])
    (Just someCat)
...
class FormPage p where
    formPage :: (Monad m, html ~ HtmlT m ())
                => View html
                -> (html -> html)
                -> p
                -> html
                -> html

makeForm :: Monad m
          => p
          -> Form (Html ()) m (FormPagePayload p)
discussHooks = simpleFormPageHooks

-- generate page data
(QC.generate $ DiscussPage <$> QC.elements [minBound..])

-- process payload
(\payload -> putStrLn $ "result: " <> show payload)

-- optional arguments
& formRequireCsrf .~ False
& formLogMsg .~ (putStrLn . ("log entry: " <>)). show)
formPageH :: forall m p uimsg err hooks handler.
  ( FormPage p
  , CsrfStore m
  , CleanupTempFiles m
  , MonadServantErr err m
  , hooks ~ FormPageHooks m p {- get post -} uimsg
  , handler ~ FormHandler p {- get post -}
  )
  => hooks -> ServerT handler m
formPageH hooks = getH :<|> postH
type FormHandler p =
    Get '[HTML] p
  <|> FormReqBody :> Post '[HTML] p
type AulaMain =
  ...
  <|> "note" :> Capture "noteid" ID :> "settings"
    :> FormHandler DiscussPage
  ...

aulaMain :: ActionM m => ServerT AulaMain m
aulaMain =
  ...
  <|> (\i -> formPageH (userSettingsHooks i))
persistence (1)

Many options:

- **postgresql-simple:**
  - do it like everybody else
  - sql commands are strings
  - query results are relations with very simple types

- **acid-state:**
  - store all application data in an MVar
  - queries are calls to `readMVar`
  - update commands must be serializable (changelog + snapshots)
  - reputation for stability and scalability issues (but that’s compared to postgresql!)

- ... (lots!)
we picked acid-state.
type AMap a = Map (IdOf a) a

type Ideas = AMap Idea

type Users = AMap User

data AulaData = AulaData
  { _dbSpaceSet :: Set IdeaSpace
  , _dbIdeaMap :: Ideas
  , _dbUserMap :: Users
  , _dbTopicMap :: Topics
  ...
}
persistence (4)

type Query a = forall m. MonadReader AulaData m => m a

findInById :: Getter AulaData (AMap a) -> IdOf a
    -> Query (Maybe a)
findInById l i = view (l . at i)

findUser :: AUID User
    -> Query (Maybe User)
findUser = findInById dbUserMap

handler = do
    ...
    user <- maybe404 =<< query (findUser uid)
    ...

handling hierarchies of data is different.

-- can't do this:

```haskell
data Store = Store (Map ID User, Map ID Doc)

data User = User { myDocs :: [Document], ... }

data Doc = Doc { creator :: User, ... }
```
where do you break up your reference graph into a tree?

- make everything that is separately addressable?
  - makes construction of page types more work.
- keep discussion threads nested in the discussed ideas?
  - then addressing comments gets harder
questions? opinions?

further reading:

<table>
<thead>
<tr>
<th>project blog</th>
<th><a href="http://aula-blog.website/">http://aula-blog.website/</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td><a href="https://github.com/liqd/aula/">https://github.com/liqd/aula/</a></td>
</tr>
</tbody>
</table>

(The production systems are only accessible from inside the participating schools.)

general-purpose libraries (will be released later this year):

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="https://github.com/zerobuzz/thentos-prelude">https://github.com/zerobuzz/thentos-prelude</a></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/zerobuzz/thentos-cookie-session">https://github.com/zerobuzz/thentos-cookie-session</a></td>
</tr>
<tr>
<td></td>
<td><a href="https://github.com/zerobuzz/thentos-html-forms">https://github.com/zerobuzz/thentos-html-forms</a></td>
</tr>
</tbody>
</table>