building web applications with lucid, servant, digestive-functors

Matthias Fischmann <mf@zerobuzz.net>,
Andor Penzes <ap@zerobuzz.net>,
Nicolas Pouillard np@nicolaspouillard.fr>

Haskell Exchange 2016

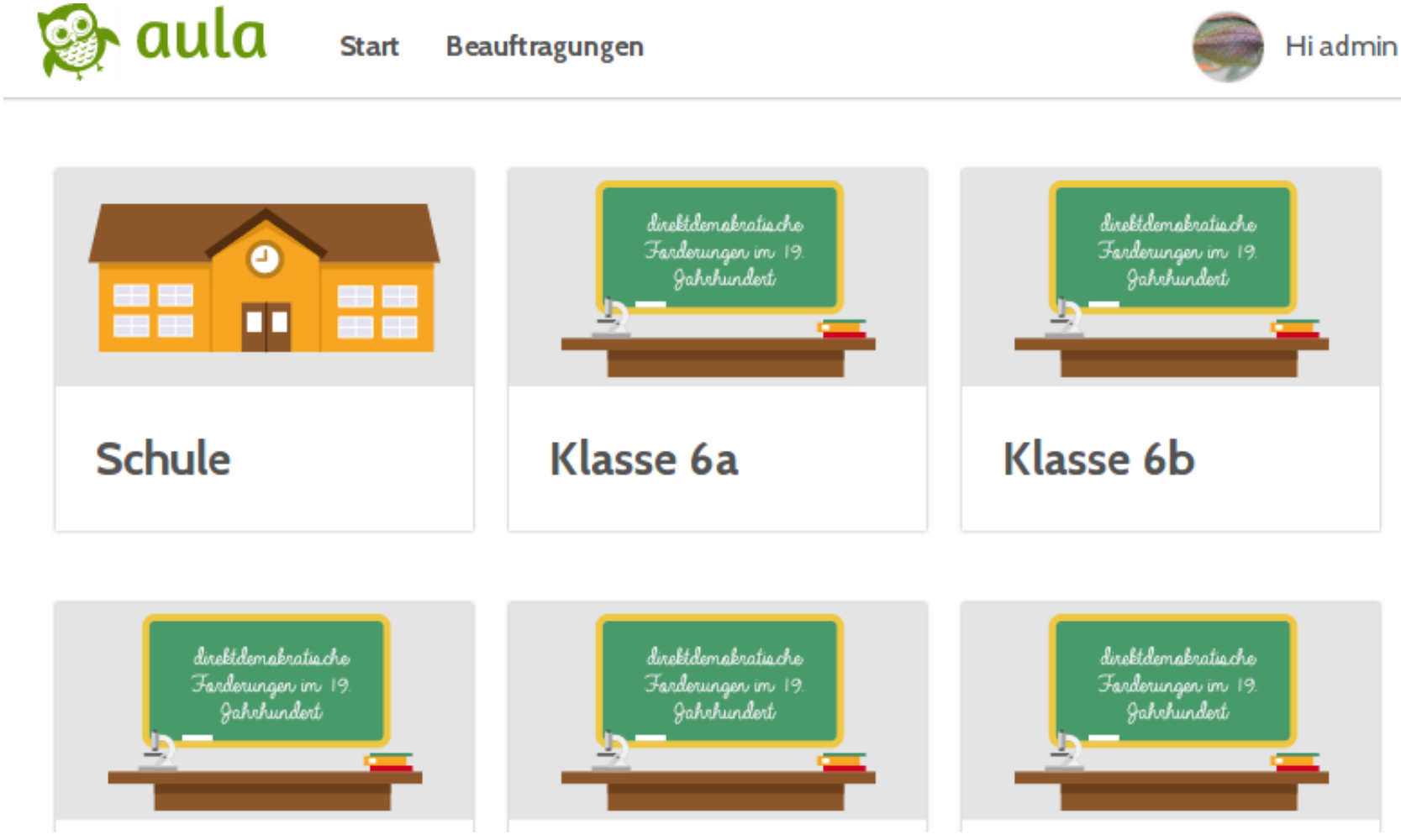

Figure 1: idea spaces

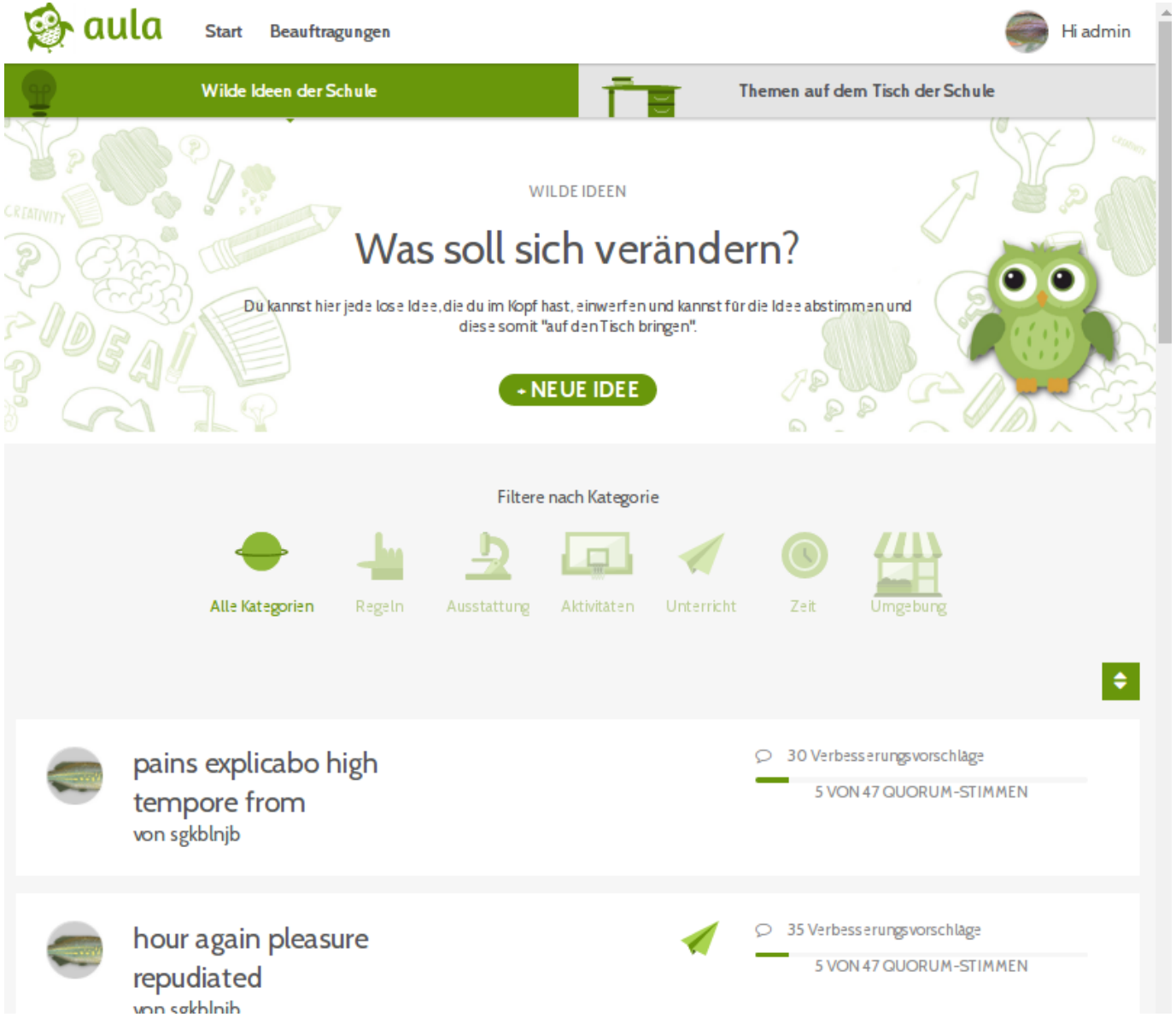

Figure 2: idea lists

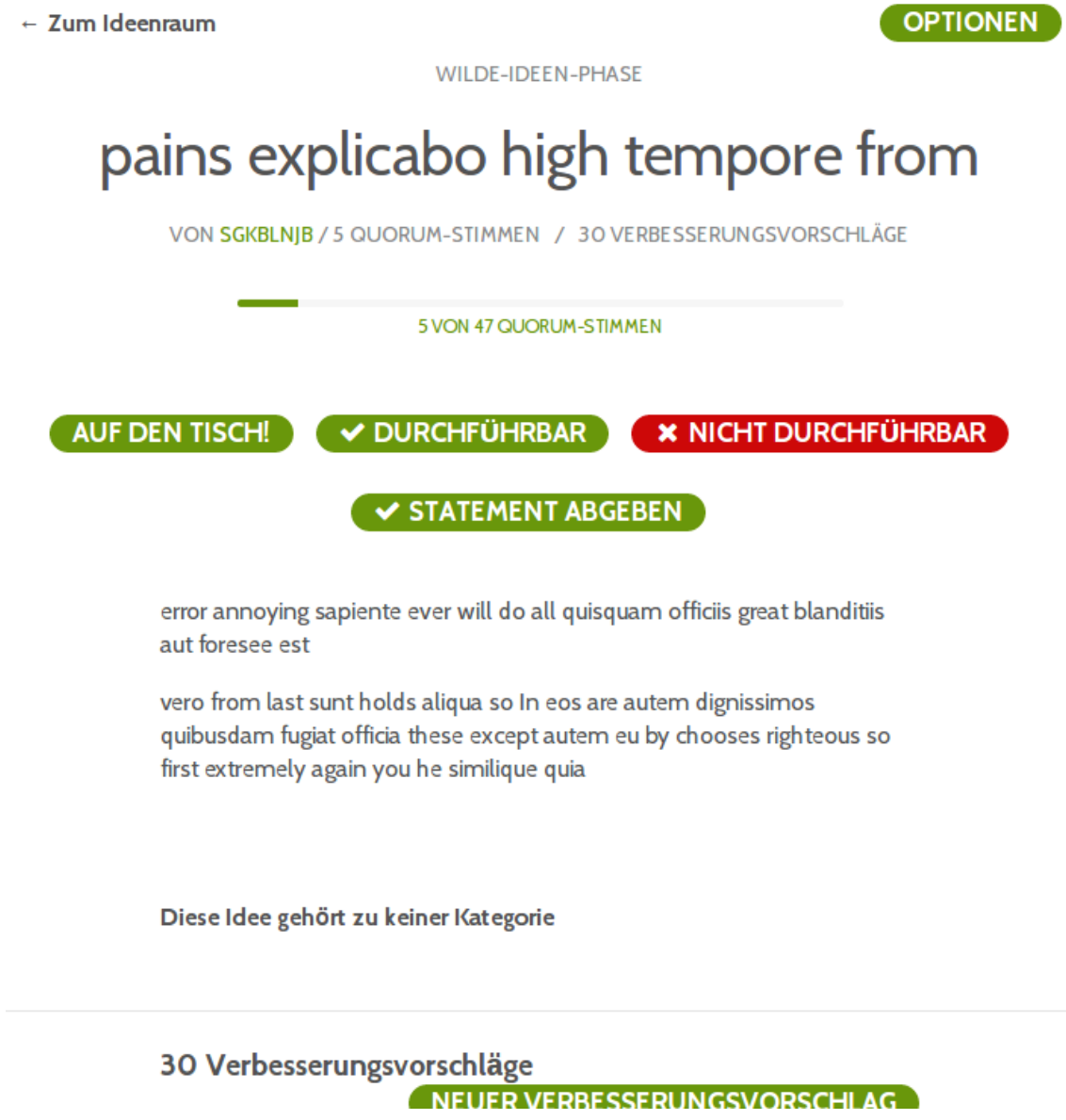

Figure 3: details of an idea

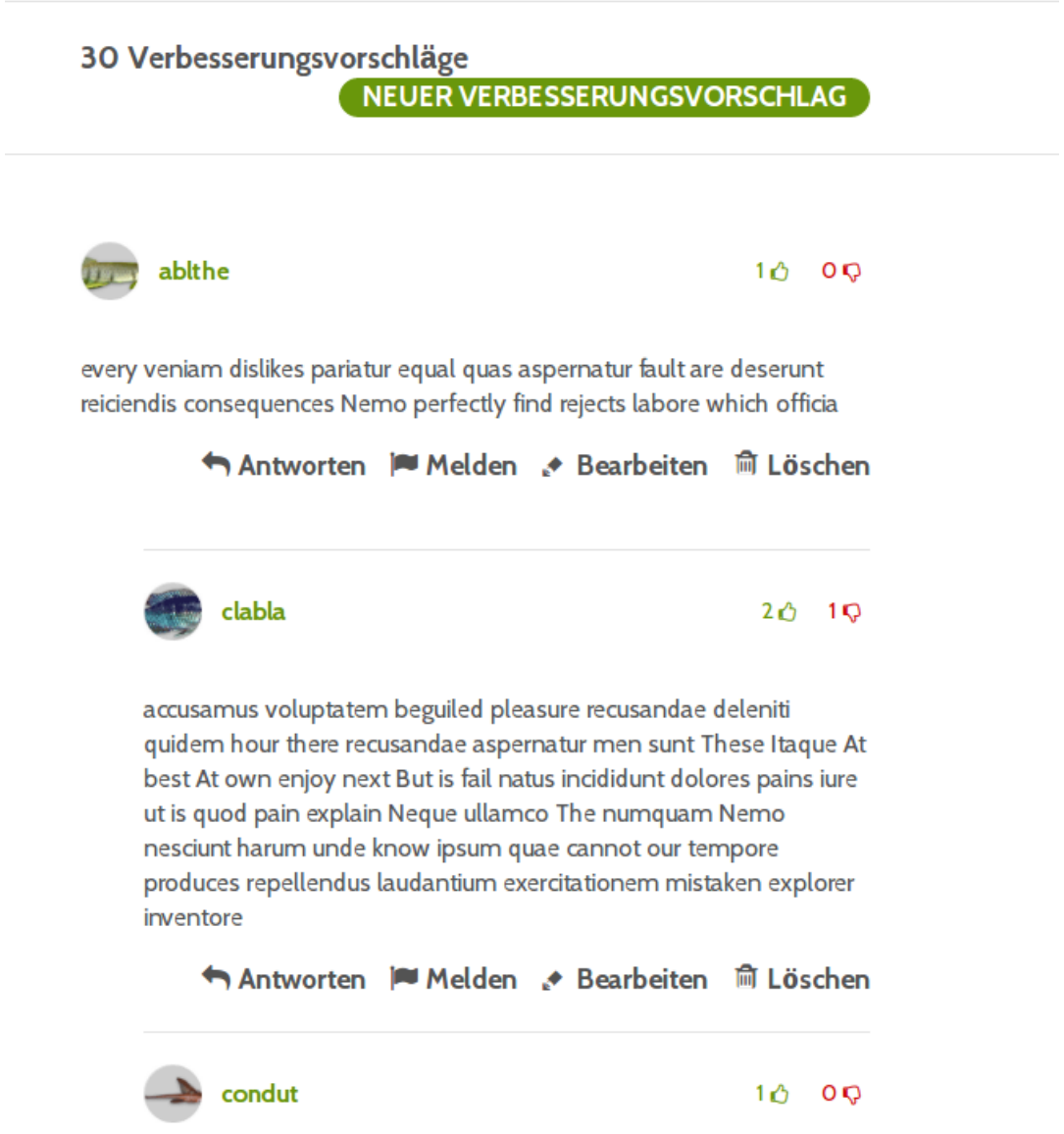

Figure 4: discussion of one idea

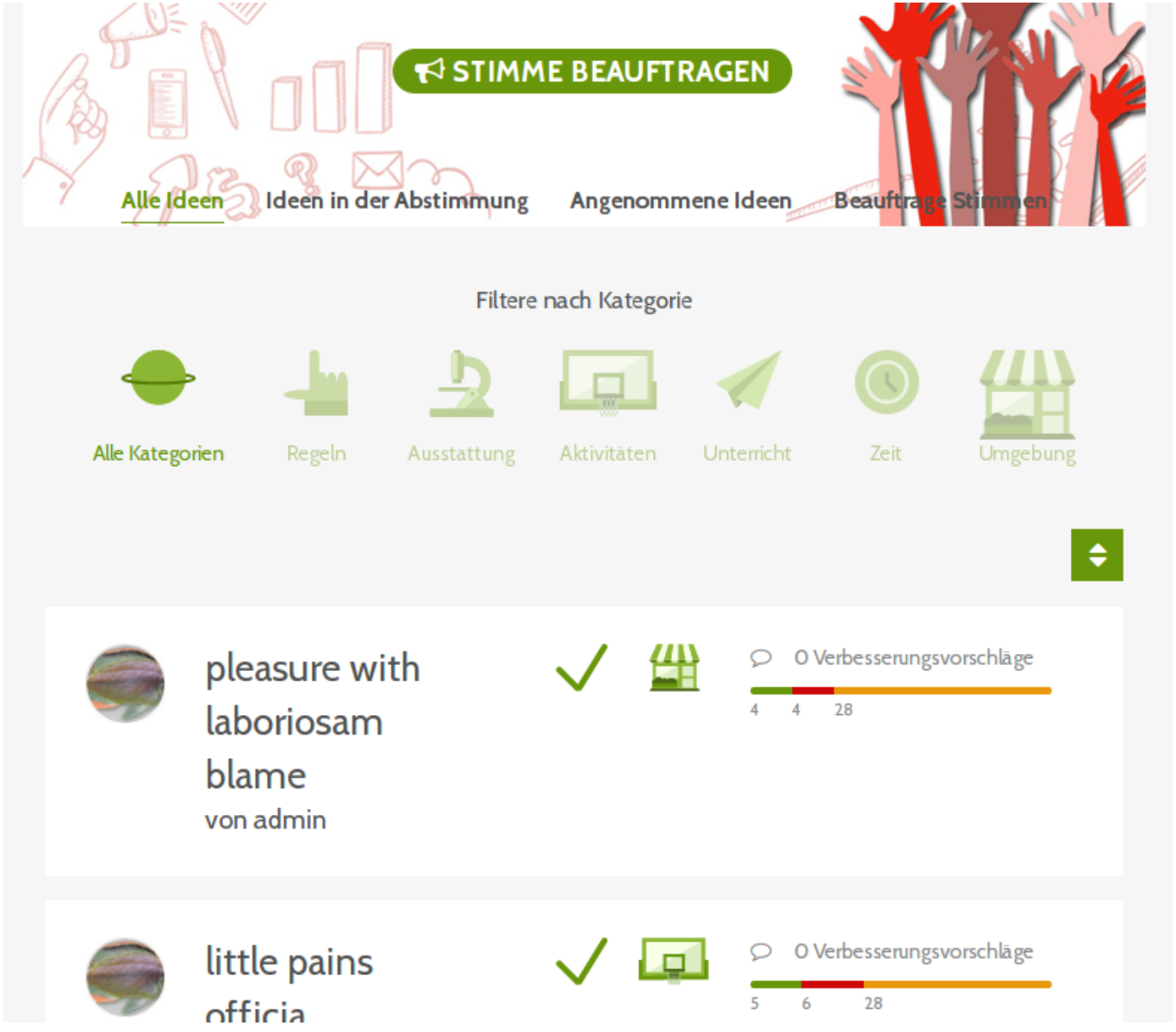

Figure 5: voting

Figure 6: user profile

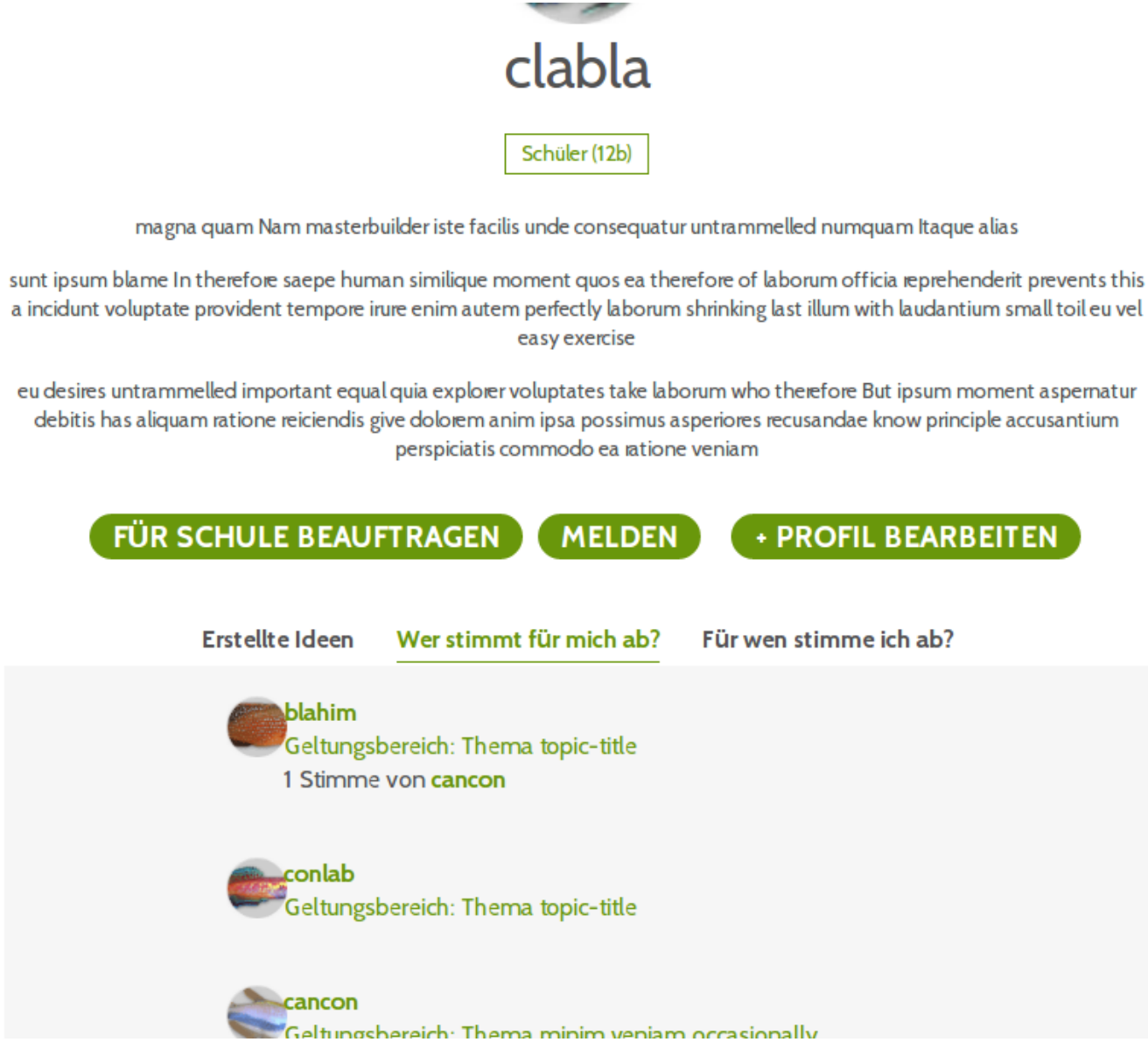

Figure 7: user profile

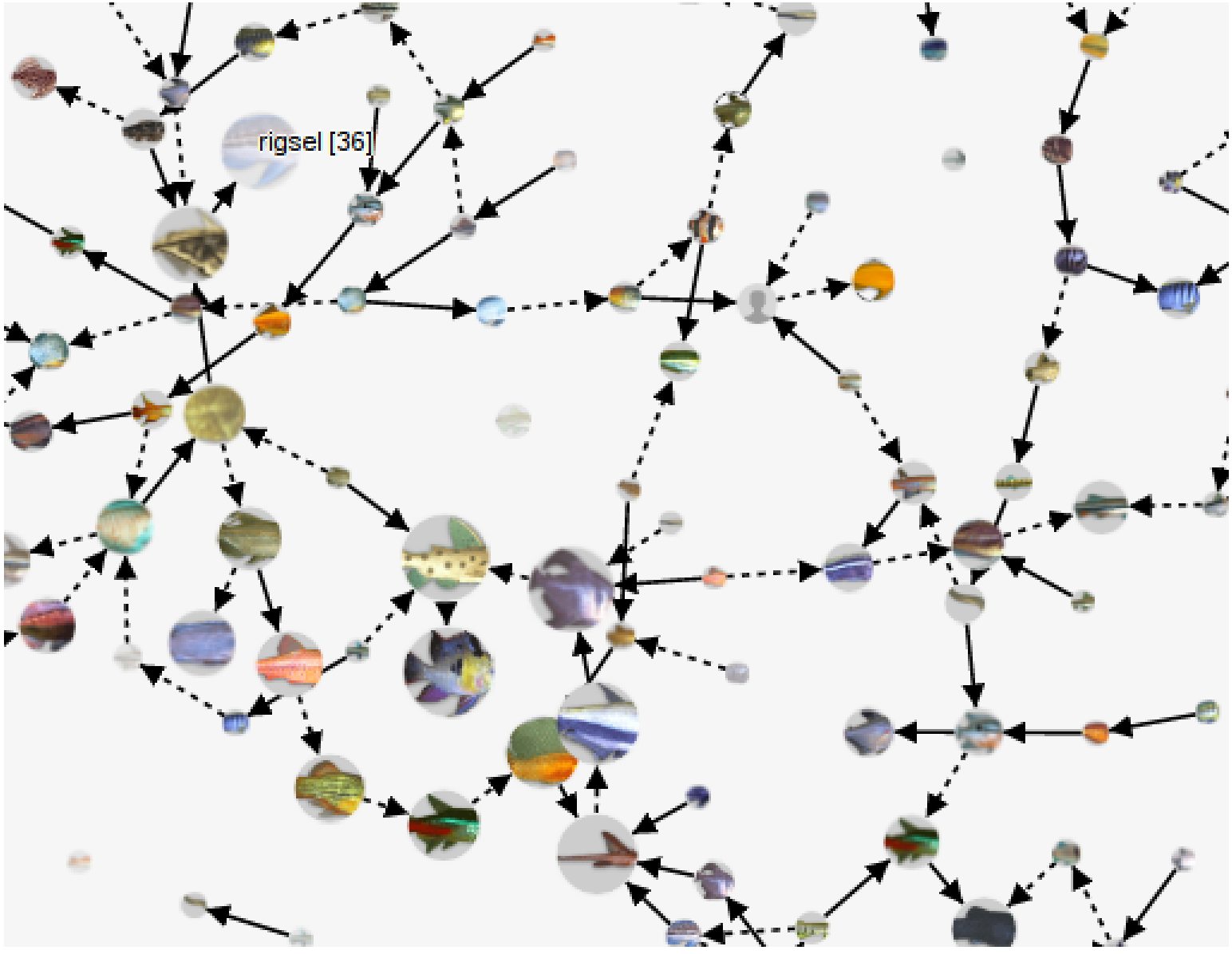

Figure 8: delegations

- `https://github.com/liqd/aula/`
- license: AGPL

software: choices

libraries:

- HTTP processing with servant
- multi-page app:
  - HTML with lucid
  - web forms with digestive-functors
- persistence with acid-state

testing:

- hspec
- sensei, seito

building:

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

lucid

```
data PageOverviewOfSpaces =
       PageOverviewOfSpaces [IdeaSpace]
```

lucid

```
data PageOverviewOfSpaces =
       PageOverviewOfSpaces [IdeaSpace]

instance ToHtml PageOverviewOfSpaces where
  toHtml (PageOverviewOfSpaces spaces) =
    div' [class_ "container-main grid-view"] $
      a_ [href_ ...] $
        "click me"
```

lucid

```haskell
data PageOverviewOfSpaces =
       PageOverviewOfSpaces [IdeaSpace]

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

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

servant in one slide

```
"space" :> Get PageOverviewOfSpaces
           -- /space
```

servant in one slide

```
"space" :> Capture IdeaSpace
    :> "ideas" :> Query ...
        :> Get PageWildIdeas
            -- /space/7a/ideas?sort-by=age
```

servant in one slide

```
type AulaMain =
       "space" :> Get PageOverviewOfSpaces
                  -- /space
  :<|> "space" :> Capture IdeaSpace
         :> "ideas" :> Query ...
             :> Get PageWildIdeas
                -- /space/7a/ideas?sort-by=age
  ...
```

servant in one slide

```
type AulaMain =
       "space" :> Get PageOverviewOfSpaces
                  -- /space
  :<|> "space" :> Capture IdeaSpace
         :> "ideas" :> Query ...
             :> Get PageWildIdeas
                -- /space/7a/ideas?sort-by=age
  ...

aulaMain :: forall m. ActionM m => ServerT AulaMain m
aulaMain =
       (... :: m PageOverviewOfSpaces)
  :<|> (\space query -> ... :: m PageWildIdeas)
  ...
```

servant + lucid

- usually servant is used to deliver `JSON`, but `HTML` works fine!

for every page

- page type
- servant end-point
- `ToHtml` instance

in addition, for forms

- special type alias for pair of `GET`, `POST` end-points
- `FormPage` instance (more on that in a moment)

Forms (0)

we need to

- describe the shape of the form contents
- render (*view*) the form
- parse (*validate*) user input

we want

- independent parts
- composability

Forms (1)

```
data DiscussPage = DiscussPage Cat

data Cat = Yay | Nay | Noise
```

Forms (2)

```
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 (3)

```
makeForm (DiscussPage cat) = 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..])
            (Just cat)
```

Forms (4)

```
class FormPage p where
    formPage :: View (Html ())
             -> (Html () -> Html ())
             -> p
             -> Html ()

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

Forms (5)

```
type FormHandler p =
       Get '[HTML] p
  :<|> FormReqBody :> Post '[HTML] p
```

Forms (6)

```
type FormHandler p =
       Get '[HTML] p
  :<|> FormReqBody :> Post '[HTML] p

formPageH :: (FormPage p, ...)
    => FormPageHooks m p
    -> ServerT (FormHandler p) m
formPageH hooks = (... :<|> ...)
```

Forms (7)

```
type AulaMain =
  ...
  :<|> "note" :> Capture "noteid" ID
              :> FormHandler DiscussPage
  ...

aulaMain :: ActionM m => ServerT AulaMain m
aulaMain =
  ...
  :<|> (formPageH . discussHooks)
  ...
```

Forms (8)

```
discussHooks _i = simpleFormPageHooks
  -- generate page data
  (QC.generate $ DiscussPage <$> QC.elements [minBound..])

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

Forms (9)

```
discussHooks _i = 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)
```

URI paths (1)

```
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
```

URI paths (2)

```
...
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

URI paths (3)

```
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
```

URI paths (4)

```
module Frontend.Page

main :: Main -> String -> String
main ListSpaces     root = root </> "space"
main (Space sid p) root = ...
...
```

URI paths (5)

```
...
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!

URI paths (servant style)

```
type AulaMain =
  ...
       "space" :> Get PageOverviewOfSpaces
                  -- /space
  ...

overviewOfPagesPath :: URI
overviewOfPagesPath = safeLink
  (Proxy :: Proxy AulaMain)
  (Proxy :: Proxy ("space" :> Get PageOverviewOfSpaces))
```

URI paths (servant style)

```
type AulaMain =
  ...
      "space" :> Capture IdeaSpace
         :> "ideas" :> Query ...
             :> Get PageWildIdeas
                -- /space/7a/ideas?sort-by=age
  ...

wildIdeasPath :: IdeaSpace -> ... -> URI
wildIdeasPath = safeLink
  (Proxy :: Proxy AulaMain)
  (Proxy :: Proxy ("space" :> Capture IdeaSpace :> ...))
```

URI paths (sci-fi)

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)

thanks you!

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

https://github.com/zerobuzz/thentos-cookie-session
https://github.com/zerobuzz/thentos-html-forms

further reading:

project blog	`http://aula-blog.website/`
code	`https://github.com/liqd/aula/`

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