Servant, Shakespeare, Persistent

An overview, and "SQLi"

Cyrill Brunner

26th September 2022

What will I show you?

- Haskell, a functional language

What will I show you?

- Haskell, a functional language
- Servant, a type-safe API definition DSL

What will I show you?

- Haskell, a functional language
- Servant, a type-safe API definition DSL
- Shakespeare, a HTML/CSS/JavaScript templater

What will I show you?

- Haskell, a functional language
- Servant, a type-safe API definition DSL
- Shakespeare, a HTML/CSS/JavaScript templater
- Persistent, a backend-agnostic type-safe data serialization DSL

Haskell

```
primes = filterPrime [2..]
    where filterPrime (p:xs) =
            p : filterPrime [x | x <- xs, x `mod` p /= 0]
```

Haskell

```haskell
primes = filterPrime [2..]
    where filterPrime (p:xs) =
            p : filterPrime [x | x <- xs, x `mod` p /= 0]
```

```haskell
data Maybe a
  = Just a
  | Nothing
```

Haskell

```haskell
primes = filterPrime [2..]
    where filterPrime (p:xs) =
            p : filterPrime [x | x <- xs, x `mod` p /= 0]
```

```haskell
data Maybe a
  = Just a
  | Nothing
```

```haskell
data Expression a where
    ExpInt    :: Int
              -> Expression Int
    ExpBool   :: Bool
              -> Expression Bool
    ExpAdd    :: Expression Int
              -> Expression Int
              -> Expression Int
    ExpEquals :: Expression a
              -> Expression a
              -> Expression Bool
```

Servant I

```
type API =
    -- /api/product/ routes, can respond with JSON
    "api" :> ToServantApi (RestAPI "product" '[JSON] Product)
    :<|>
    -- GET /search?searchTerm=<term> returns HTML
    "search" :> QueryParam "searchTerm" Text
             :> Get '[HTML] Html
```

Servant II

```
-- ToServantApi (RestAPI "product" '[JSON] Product)
data RestAPI name encs res route = RestAPI
  {

  ,


  ,


  ,


  ,


  } deriving (Generic)
```

Servant II

```
-- ToServantApi (RestAPI "product" '[JSON] Product)
data RestAPI name encs res route = RestAPI
  { _getAll    :: route :- name :> Get encs [res]
    -- GET  /api/product       list of resources
  ,


  ,


  ,


  ,


  } deriving (Generic)
```

Servant II

```
-- ToServantApi (RestAPI "product" '[JSON] Product)
data RestAPI name encs res route = RestAPI
  { _getAll    :: route :- name :> Get encs [res]
    -- GET  /api/product       list of resources
  , _addNew    :: route :- name
               :> ReqBody encs res :> Verb 'POST 201 encs res
    -- POST /api/product       add new resource
  ,


  ,


  ,


  } deriving (Generic)
```

Servant II

```
-- ToServantApi (RestAPI "product" '[JSON] Product)
data RestAPI name encs res route = RestAPI
  { _getAll    :: route :- name :> Get encs [res]
    -- GET  /api/product        list of resources
  , _addNew    :: route :- name
               :> ReqBody encs res :> Verb 'POST 201 encs res
    -- POST /api/product        add new resource
  , _getSingle :: route :- name :> Capture "id" (Id res)
               :> Get encs res
    -- GET  /api/product/:id    find resource
  ,



  ,



  } deriving (Generic)
```

Servant II

```
-- ToServantApi (RestAPI "product" '[JSON] Product)
data RestAPI name encs res route = RestAPI
  { _getAll    :: route :- name :> Get encs [res]
    -- GET  /api/product        list of resources
  , _addNew    :: route :- name
               :> ReqBody encs res :> Verb 'POST 201 encs res
    -- POST /api/product        add new resource
  , _getSingle :: route :- name :> Capture "id" (Id res)
               :> Get encs res
    -- GET  /api/product/:id    find resource
  , _replace   :: route :- name :> Capture "id" (Id res)
               :> ReqBody encs res :> Put encs res
    -- PUT  /api/product/:id    replace resource
  ,


  } deriving (Generic)
```

Servant II

```
-- ToServantApi (RestAPI "product" '[JSON] Product)
data RestAPI name encs res route = RestAPI
  { _getAll    :: route :- name :> Get encs [res]
    -- GET  /api/product       list of resources
  , _addNew    :: route :- name
               :> ReqBody encs res :> Verb 'POST 201 encs res
    -- POST /api/product       add new resource
  , _getSingle :: route :- name :> Capture "id" (Id res)
               :> Get encs res
    -- GET  /api/product/:id   find resource
  , _replace   :: route :- name :> Capture "id" (Id res)
               :> ReqBody encs res :> Put encs res
    -- PUT  /api/product/:id   replace resource
  , _delete    :: route :- name :> Capture "id" (Id res)
               :> Verb 'DELETE 204 encs NoContent
    -- DELETE /api/product/:id delete resource
  } deriving (Generic)
```

Servant III

```
productsRestApi ::
  Server ("api" :> ToServantApi (RestAPI "product" '[JSON]
  ↪  Product))
productsRestApi :: RestAPI "product" '[JSON] Product AsServer
productsRestApi = toServantApi (RestAPI {..})
  where _getAll    ::                          Handler [Product]
        _addNew    ::              Product -> Handler Product
        _getSingle :: ProductId ->             Handler Product
        _replace   :: ProductId -> Product -> Handler Product
        _delete    :: ProductId ->             Handler NoContent
        ...

searchApi :: Server ("search" :> QueryParam "searchTerm" Text
                              :> Get '[HTML] Html)
searchApi :: Maybe Text -> Handler Html
searchApi filterTerm = do
  ...

  pure htmlAnswer
```

Servant IV

```
searchApi :: Maybe Text -> ClientM Html
searchApi = client ("search" :> QueryParam "searchTerm" Text :>
↪  Get '[HTML] Html)
```

Servant V

- `servant`

Servant V

- `servant`
- `servant-server`

Servant V

- `servant`
- `servant-server`
- `servant-auth`

Servant V

- `servant`
- `servant-server`
- `servant-auth`
- `servant-client`

Servant V

- `servant`
- `servant-server`
- `servant-auth`
- `servant-client`
- `servant-js` and `servant-foreign`

Servant V

- `servant`
- `servant-server`
- `servant-auth`
- `servant-client`
- `servant-js` and `servant-foreign`
- `servant-blaze` vs. `servant-lucid`

Servant V

- `servant`
- `servant-server`
- `servant-auth`
- `servant-client`
- `servant-js` and `servant-foreign`
- `servant-blaze` vs. `servant-lucid`
- `servant-docs`

Servant V

- `servant`
- `servant-server`
- `servant-auth`
- `servant-client`
- `servant-js` and `servant-foreign`
- `servant-blaze` vs. `servant-lucid`
- `servant-docs`
- `servant-swagger` and `servant-swagger-ui`

Servant V

- `servant`
- `servant-server`
- `servant-auth`
- `servant-client`
- `servant-js` and `servant-foreign`
- `servant-blaze` vs. `servant-lucid`
- `servant-docs`
- `servant-swagger` and `servant-swagger-ui`
- `servant-streaming` and `servant-conduit`

Shakespeare I

```
pure $ unlines
  [ "<!DOCTYPE html>"
  , "<html><head><title>Search Interface</title></head><body>"
  , "<script>"
  , "function addItem(productId, name, description) {"
  , "  const list = document.getElementById('list');"
  , "  if(list instanceof HTMLDivElement) {"
  , "    const parent = list.parentElement;"
  , "    list.remove();"
  , "    parent.innerHTML +="
  , "      `<ul><li>"
  , "        (${productId}) - ${name}: ${description}</ul>`;"
  , "  } else {"
  , "    list.innerHTML +="
  , "      `<li> (${productId}) - ${name}: ${description}`;"
  , "  }"
  , "}"
  , ...
  ]
```

Shakespeare II

```
pure [shamlet|
    $doctype 5
    <html>
        <head>
            <title>Search Interface
            <script>
            #{preEscapedToMarkup script}
        <body>
            ...
|]
```

Shakespeare III

```
script :: Text
script = toStrict $ renderJavascriptUrl undefined [julius|
  function addItem(productId, name, description) {
    const list = document.getElementById('list');
    if(list instanceof HTMLDivElement) {
      const parent = list.parentElement;
      list.remove();
      parent.innerHTML +=
        `<ul><li> (${productId}) - ${name}:
        ↪  ${description}</ul>`;
    } else {
      list.innerHTML +=
        `<li> (${productId}) - ${name}: ${description}`;
    }
  }
  ...
|]
```

Persistent I

```
share [mkPersist sqlSettings, mkMigrate "migrateAll"]
↪  [persistLowerCase|
Product
  name        Text
  description Text

  deriving Show
  deriving Eq

User
  username Text

InShoppingCart
  user    UserId
  product ProductId
  qty     Int
|]
```

Persistent II

```
allMigrations = do
  Product.migrateAll
  OtherModule.migrateAll

main = do
  runMigration allMigrations

  sqlPool <- createPoolConfig (MySQLConfig ...)

  let application =
          hoistServer (...) api
              & serve
              & ...
              & cacheMiddleware cacheConfig
              & gzipMiddleware
              & loggingMiddleware

  run 8080 application
```

Persistent III

```
searchApi filterTerm = do
  products :: [Entity Product] <- runSqlPool (
    selectList
      (maybe []
            (\term -> [ProductName `like` term]
                  ||. [ProductDescription `like` term])
          filterTerm
      )
      []
    ) pool
  ...

f `like` a =
  Filter f (FilterValue a) (BackendSpecificFilter "LIKE")
```

Persistent IV

```
dbRestAPI = toServant RestAPI { .. }
  where
    _getAll' = runDB (DB.selectList [] [])
    _addNew' res = do
        idx <- runDB (DB.insert res)
        pure $ Entity idx res
    _getSingle' idx =
        Entity idx
          <$> whenNothingM (runDB (DB.get idx))
                           (throwError err404)
    _replace' idx res = do
        runDB (DB.repsert idx res)
        pure $ Entity idx res
    _delete' key = do
        runDB (DB.delete key)
        pure NoContent
```