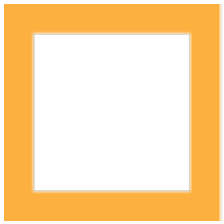
Full Stack Functional - Elm
Functional Programming

Jens Egholm Pedersen and Anders Kalhauge



cphbusiness

Spring 2018

Setup

Clone the `exercise-elm` repository

```
$ git clone https://github.com/
  cphbus-functional-programming/exercise-elm.git
```

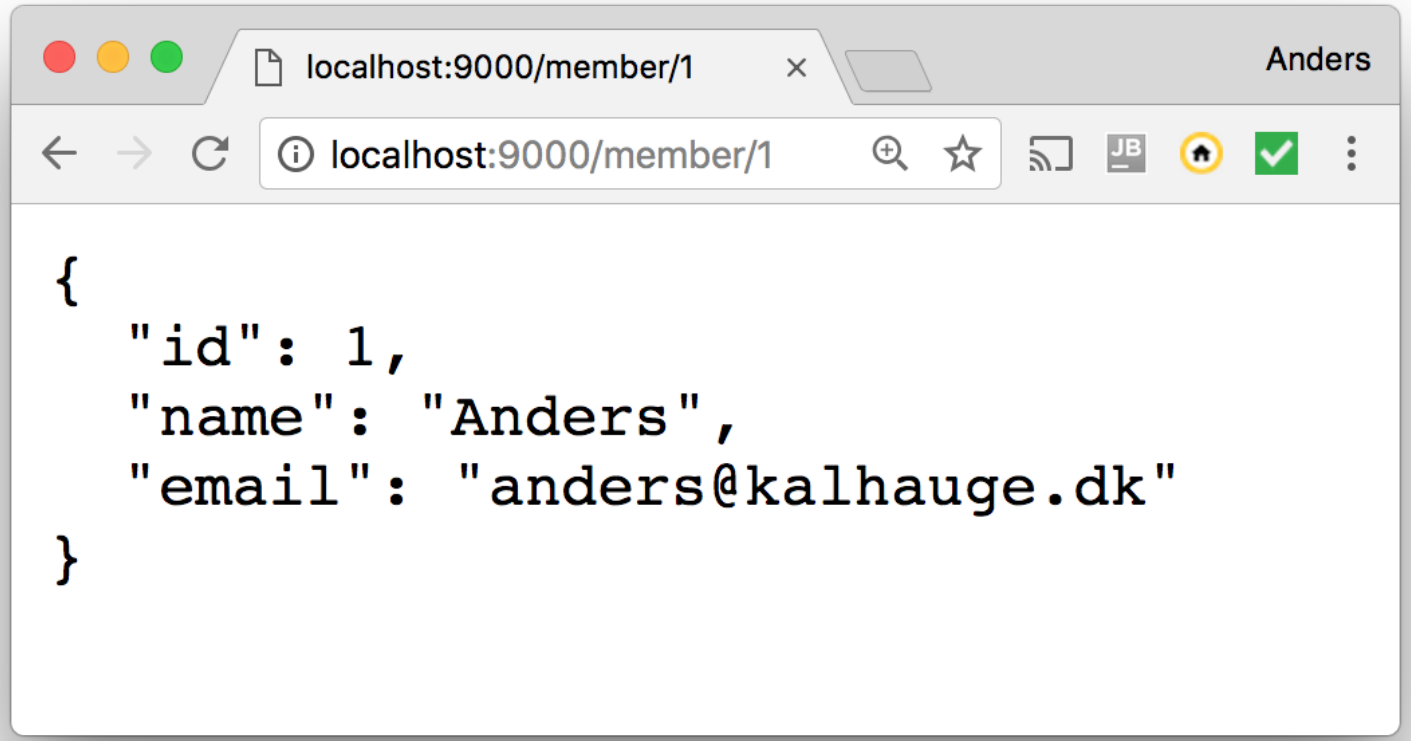
Run the member server. The server application is in the server folder:

```
$ cd exercise-elm
$ java -jar server/simple-rest-1.0.jar
```

The server starts listening on port 9000

Check Server

Start a browser and write `http://localhost:9000/member/1` in the address:



Installing Elm packages

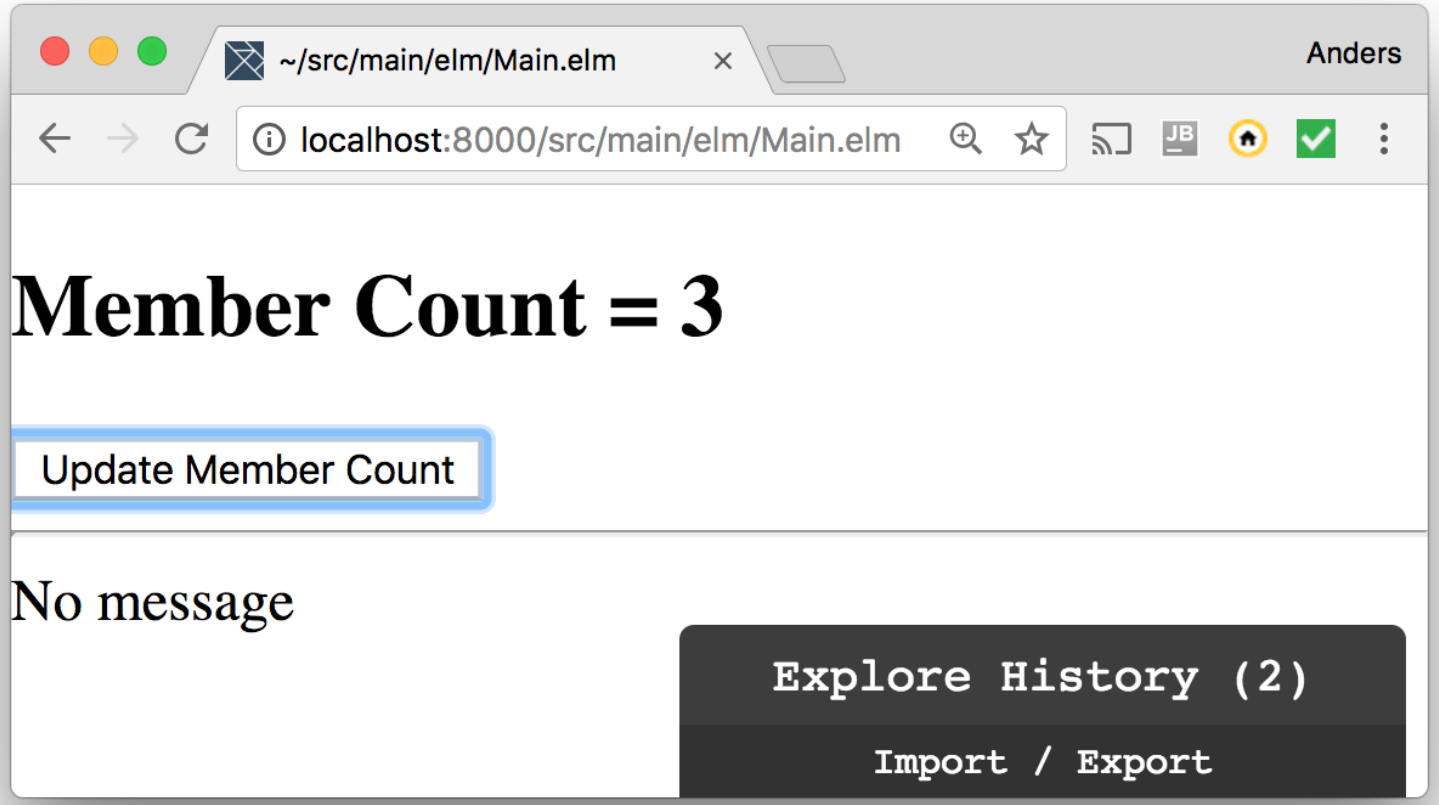
In a new terminal write:

```
$ elm-package install
```

and answer `Y` to the question. Start `elm-reactor`:

```
$ elm-reactor
```

In the browser address field write `http://localhost:8000/src/main/elm/Main.elm` and press the button:



Start coding

Open your favourite editor, ie:

```
$ atom .
```

Open the `Main.elm` file. It is in the `src/main/elm/` folder.

`Html.program` tells the Elm runtime which functions to call when the program initializes, when the model should be viewed and updated, and which function to set subscriptions.
The functions are referenced in a record:

```
main =
  Html.program
    { init = init
    , view = view
    , update = update
    , subscriptions = subscriptions
    }
```

Change the name of the `init` function to `start`

Exercise 2 - A member record

Create a record type that can hold the data from the Restful server. It is easyer to work with the record type if it's fields have the same names and types as the JSON object (see slide #4)

Give it the type alias `Member`

Create a decoder from a JSON value to a `Member`, use this signature:

```
import Json.Decode as Decode
...
decodeMember : Decode.Decoder Member
```

Hint: You should use `Decode.map3` because `Member` has three fields.

Create an encoder from a `Member` to a JSON value, use the signature:

```
import Json.Encode as Encode
...
encodeMember : Member -> Encode.Value
```

Hint: `Encode.object` takes a list of pairs [1] with a field name and a `Encode.Value`. Use `Encode.int` and `Encode.string` to get values from integers and strings.

[1] two-tupples

Exercise 5 - The request message

Create messages for `GetMember` and `MemberReceived`, let you inspire by the messages for `GetMemberCount` and `MemberCountReceived`, just remember that we receive a `Member` this time, and not an `Int`.

Add the messages to the `update` function, let them return: `(model, Cmd.none)` for now.

Exercise 6 - Add a button

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Add a button to the view that gets a member.

Add a field `member` in the model to hold a member. Make sure it is initialized with a dummy member[2].

[2] We can make the type `Maybe` later

Create a function, that can start getting a member from the server. It should have this signature:

```
getMember : Cmd Msg
```

See `getMemberCount`, just remember to use the new decoder for `Member`s. Let the member id be hardcoded to `1`, the url should be `(url "1")`.

Exercise 9 - Update update

Update the cases for `GetMember` and `MemberReceived (Ok member)` in the `update` function.

Exercise 10 - View the result

Update the `view` function so that the member just received is shown in a form.

Exercise 11 - Activating the form

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Add `onInput` events to the fields in the form. Also add messages needed to update the `member` in the `Model`.

Add nesceary cases to the `update` function.

Hint: see `http://elm-lang.org/examples/form`

Exercise 12 - The post messages

Create messages for `PostMember` and `MemberPosted`. When a member is posted a string is returned with the value `"OK"`.

Add the messages to the `update` function, let them return: `(model, Cmd.none)` for now.

Exercise 13 - Create the body

The `Http.post` function have the following signature:

```
post : String -> Body -> Decoder a -> Request a
```

Create[3] a function that creates a `Body` from a `Member`:

```
memberJsonBody : Member -> Http.Body
memberJsonBody member =
  Http.jsonBody <| encodeMember member
```

`Explain` the function to your pair mate.

[3] Copy'n'paste

Create a function, that posts the member in the form to the server. It should have this signature:

```
postMember : Member -> Cmd Msg
```

`Http.post` have the following signature:

```
post : String -> Body -> Decoder a -> Request a
```

Where the first argument is the url to our service, the body is the value of our newly implemented function with the member, the decoder is the on decoding the result, here a string.

Exercise 15 - Update update

Update the cases for `PostMember` and `MemberPosted (Ok message)` in the `update` function.

Ensure that `getMemberCount` is called on happy return.

What next?

- Ability to show a list of members.
- Ability to choose a single member.
- Ability to delete members[4].

[4] not supportet in the server yet