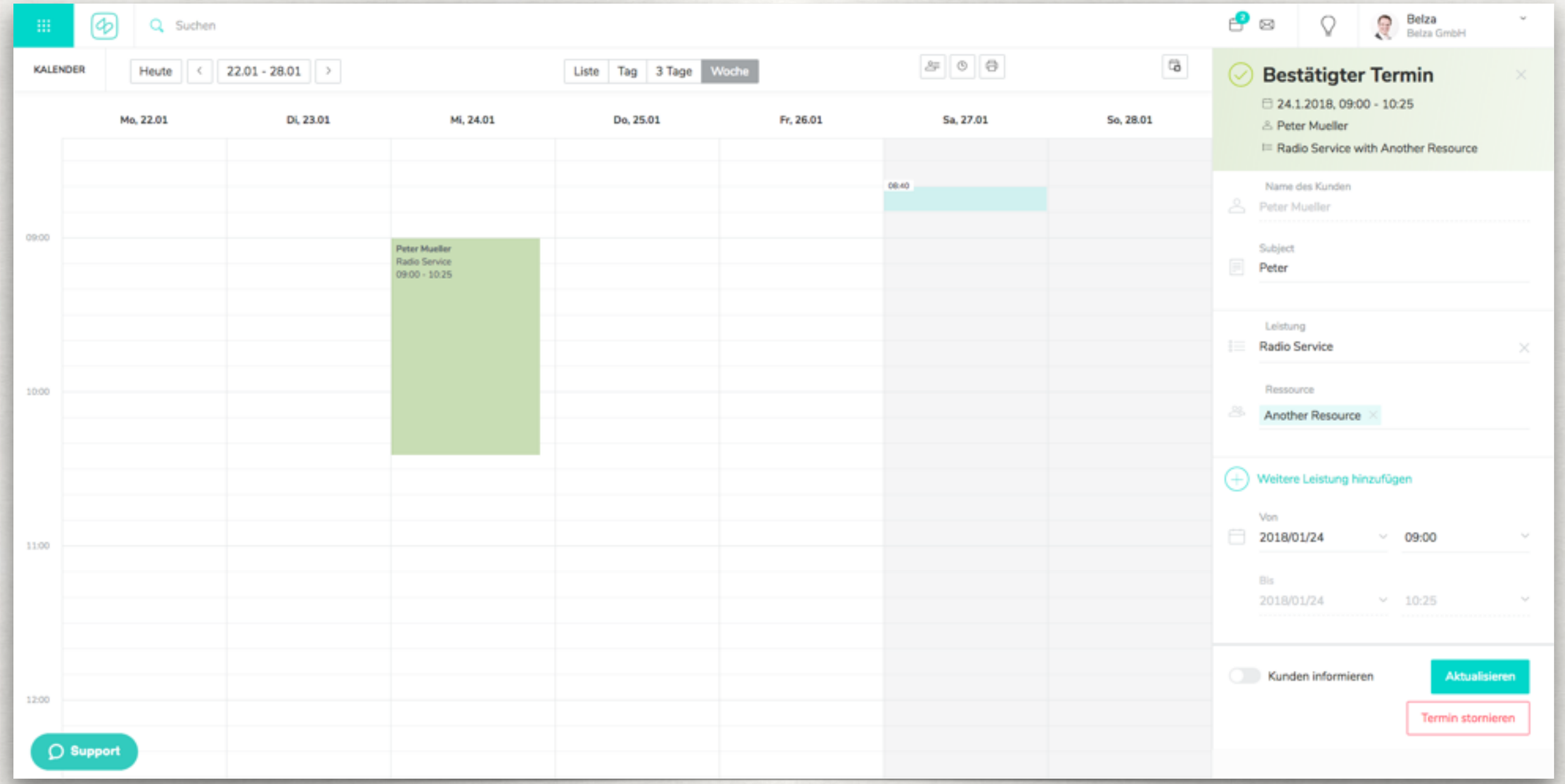
FROM REACT TO ELM

shore

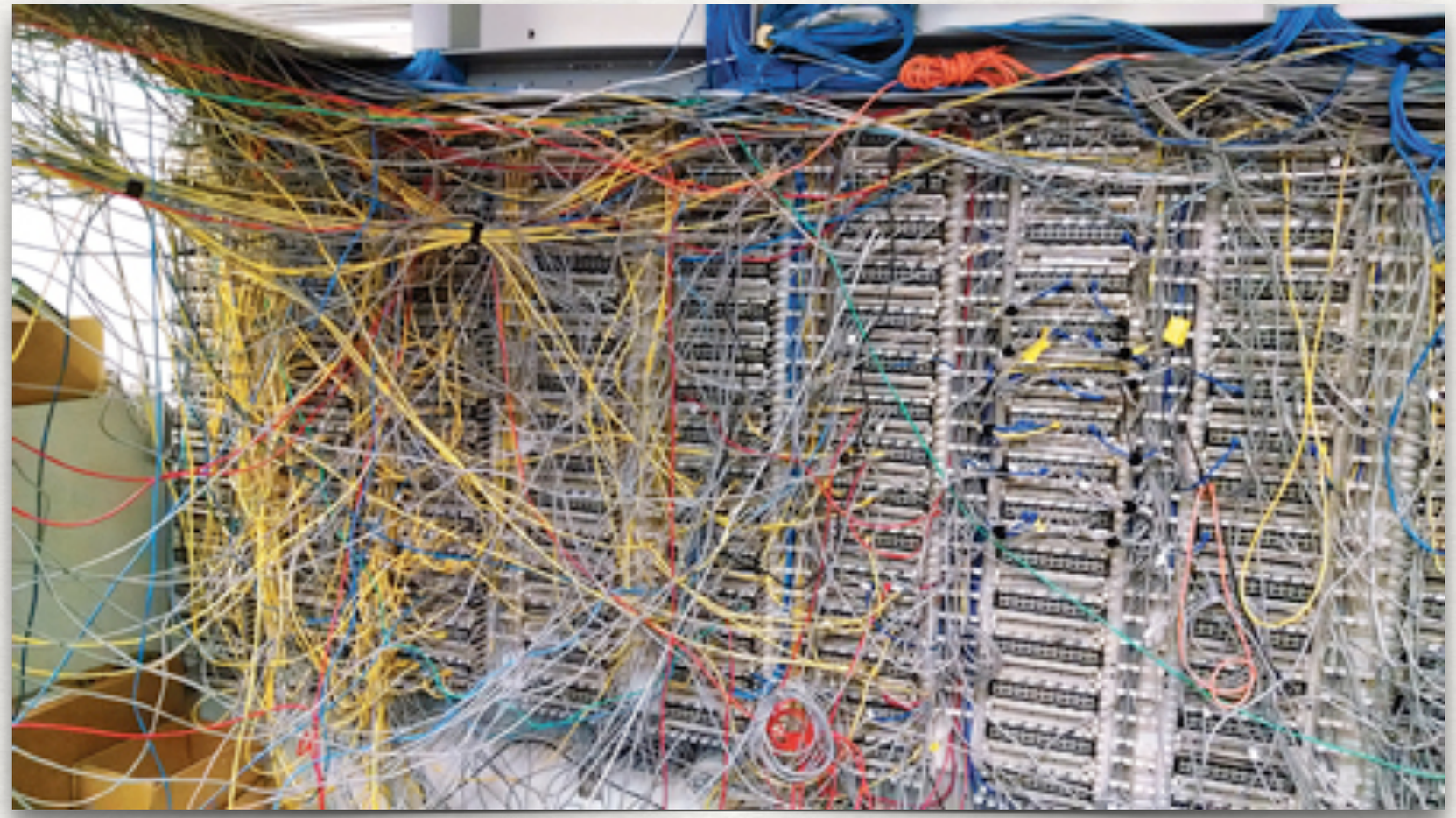
- Chris, Frontend Dev @ shore
- Our product helps SMBs digitize their business.



ELM AT SHORE

ELM AT SHORE

- Many bugs
- Often changing requirements -> patched up code

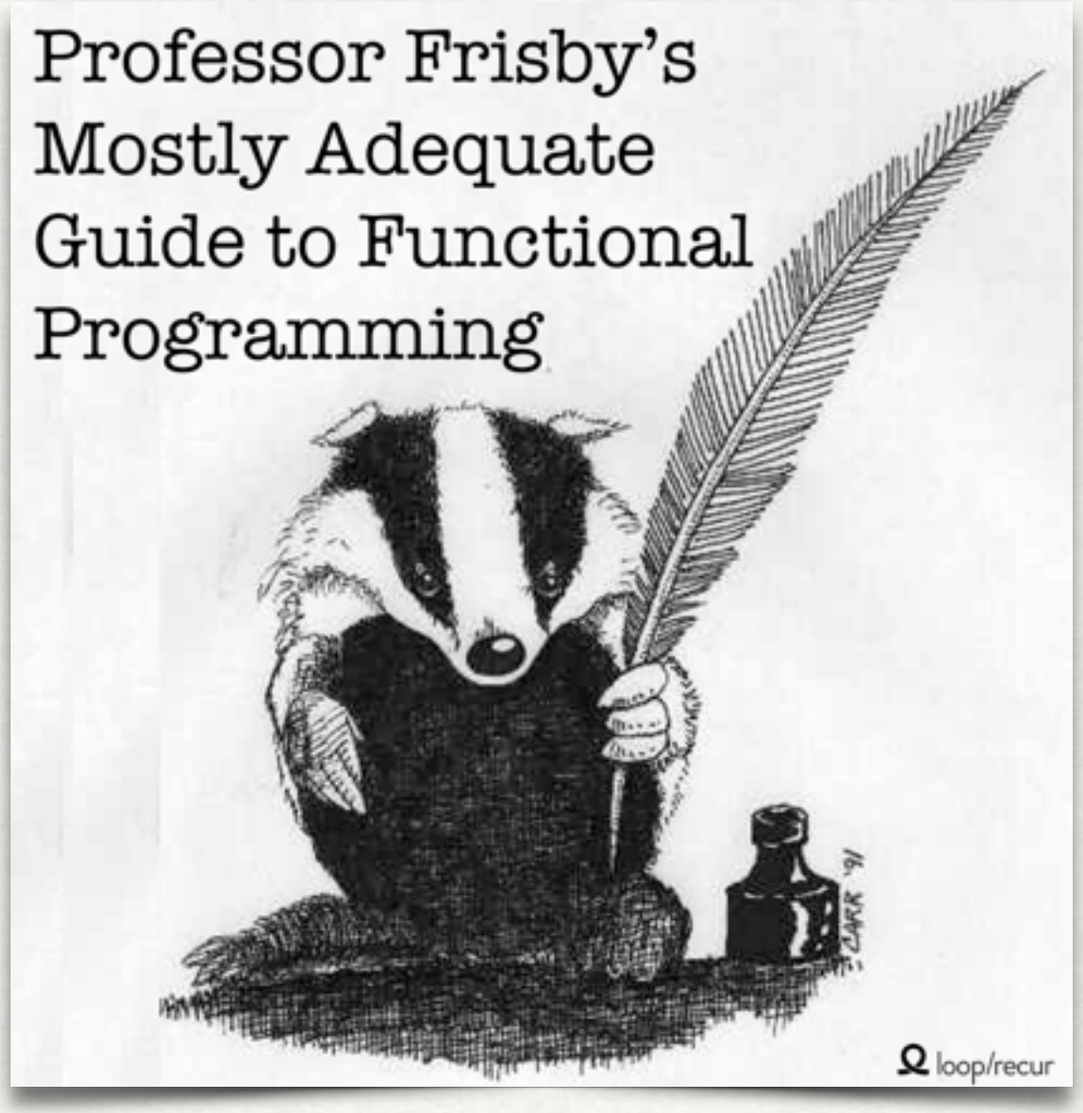


“Elm has no runtime errors”

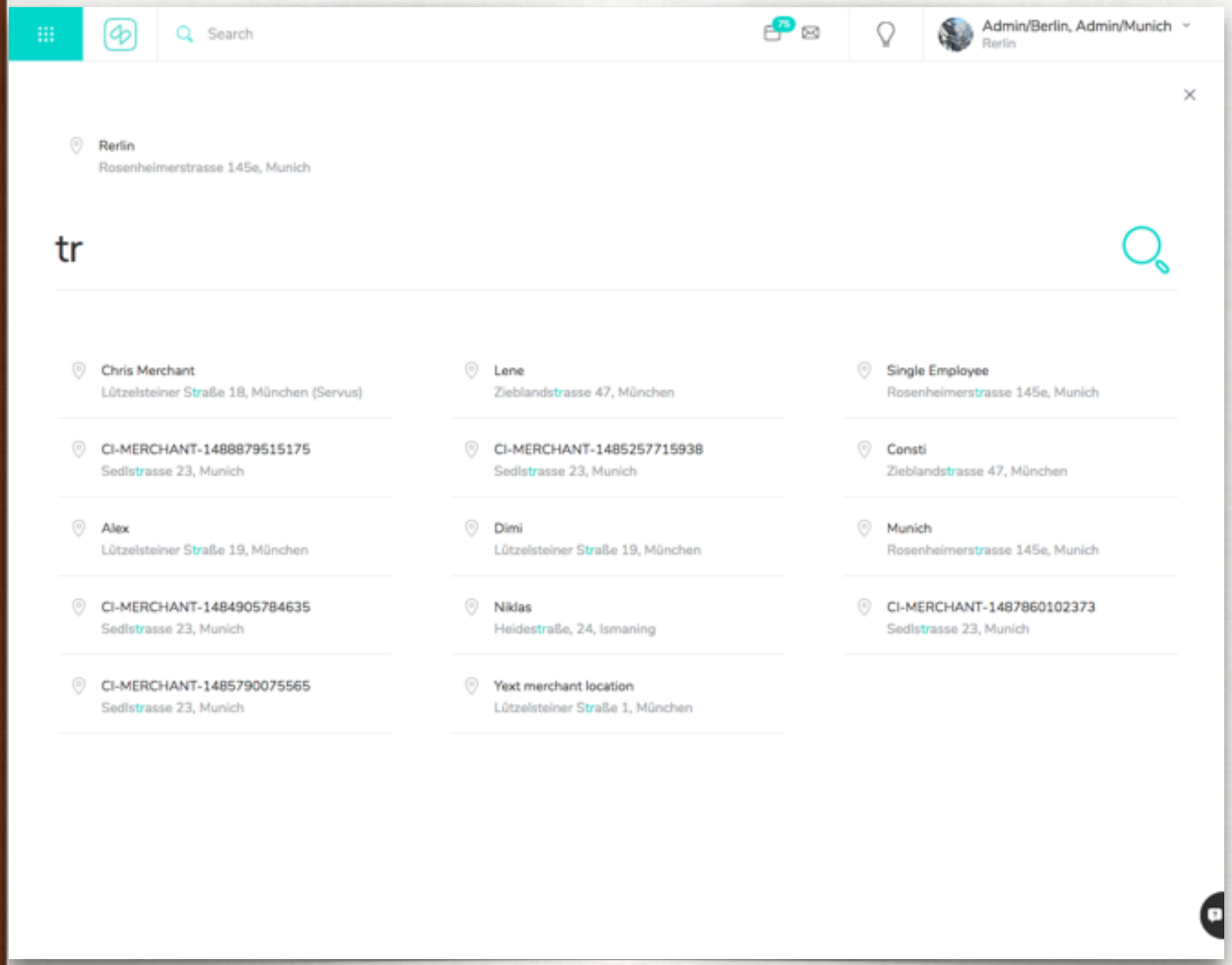
- Talk at reactive conf in Bratislava
- WTF ??? :-) Can’t be true

TRYING IT

- 5 months went by until I tried it out
- I was amazed
- “How are not more people doing this”
- Makes many problems impossible that you are dealing with in JS every day.



FIRST PROJECT



- Worked out great
- Felt awesome
- Multiple refactoring very easy
- Forces good architecture
- 1/2 half bundle size

MORE PROJECTS

- Initially a coworker and I started
- Other people got interested => they need to learn
- Integrating with more projects, legacy API Client we used ports

```
update : Msg -> Model -> ( Model, Cmd Msg )
update msg model =
    case msg of
        FeedbackLoaded (Ok { feedbacks, totalPages }) ->
            ( { model
                | feedbacks = model.feedbacks ++ feedbacks
                , feedbackTotalPages = totalPages
              }
            , Cmd.none
            )

        FeedbackLoaded (Err error) ->
            ( model, Cmd.none )

        RowClick id ->
            ( model, sendToJs (Ports.RowClick id) )
```

```
let app = Elm.Main.embed(node, [translations, companyBaseUrl]);

const handlePortMsg = ({ action, payload }) => {
  switch (action) {
    case 'RowClick':
      onAction({
        type: 'NAVIGATE_TO',
        payload: { url: `#/booking/appointments/${payload}` },
        meta: { proxy: true },
      });
      break;
    default:
      console.error(`Received unknown action type '${action}' from Elm.`);
  }
};

app.ports.msgForJs.subscribe(handlePortMsg);
```

IMPRESSIONS

- Puts your mind at ease
- Painless and confident refactoring
- Easy for beginners
- Great tools, no fighting with details
  - format
  - safe package manager
  - webpack integration

MAIN COMPLAINTS

"Not backed by Facebook"

"I wanna use NPM"

"FP is just a phase"

"I need to learn this"

"Coding React is faster"

"It won't let me do …"

REAL WORLD PROBLEMS & SOLUTIONS

- Translations: I18Next
- Localization: Web Components
- Everything else: 5-10% Ports
- Unit test (fewer than with JS)
- Elm CSS (validity guarantee)

elm packages ChristophP / elm-i18next / 2.0.1

Elm i18next - Load and use JSON translations files at runtime

Functions for working with dynamically loaded translations in Elm. PRs and suggestions welcome.

Simple Example

```
elm package install ChristophP/elm-i18next
```

Then use the module in your app like this.

```
import Http
import Html exposing (Html)
import I18Next exposing
      ( t
      , tr
      , Translations
      , Delims(..)
      , initialTranslations
      , fetchTranslations
      )
```

LOOKING BACK AFTER 9 MONTHS

- Still no runtime errors from Elm code
- 60 % of FE code is in Elm
- Much more maintainable code (even BE interested)
- Require less up front knowledge (guarantees)
- More interest in FP, Haskell, Elixir etc.
- Started “Elm |> Munich Meetup”

THE BROADER PICTURE

- FP is awesome, plenty of other compile-to-JS solution
  - PureScript
  - GHCjs
  - ClojureScript
  - ReasonML
- It’s about expressing the problem as easy as possible. Separating data and logic helps growth and composition
- Build stuff fast vs. stay flexible

DIFFERENCE

TAKE AWAYS

- Much Less Bugs
- Mostly Happier Devs
- Hiring Plus
- Code grows with the requirements
- Outlook on Elm 0.19
  - easier for SPAs (REALLY small bundlesizes, SSR, easy upgrade)

- Chris, Frontend Dev @ Shore
- Github: ChristophP
- elmlang.slack.com: christophp

Elm |> Munich

Next Wednesday, 24. January

BACKUP

REACT TO ELM

Elm	React/Redux
Model	State
update	Reducer
Msg	Actions
Cmd	Sagas/Redux Loop/Redux Promises
view	render

PAIN POINTS

- Translations -> I18next
- Localization
- Dates and local time
- Learning curve
- Slow release cycle sometimes frustrating
- Strictness but for a good reason

TESTING

```
all : Test
all =
    describe "GitHub Response Decoder"
        [ test "it results in an Err for invalid JSON" <|
            \() ->
                let
                    json =
                        """{ "pizza": [] }"""

                    isErrorResult result =
                        False
                in
                    json
                        |> decodeString responseDecoder
                        |> isErrorResult
                        |> Expect.true "Expected decoding an invalid response to return an Err."
        , test "it successfully decodes a valid response" <|
            \() ->
                """{ "items": [
                    /* TODO: put JSON here! */
                ] }"""
                    |> decodeString responseDecoder
                    |> Expect.equal
                        (Ok
                            [ { id = 5, name = "foo", stars = 42 }
                            , { id = 3, name = "bar", stars = 77 }
                            ]
                        )
        ]
```