Getting started with Fable and Fable Elmish

First steps in a wonderful world of F# for web development

Pavel Oborin
@POborin

A naive approach

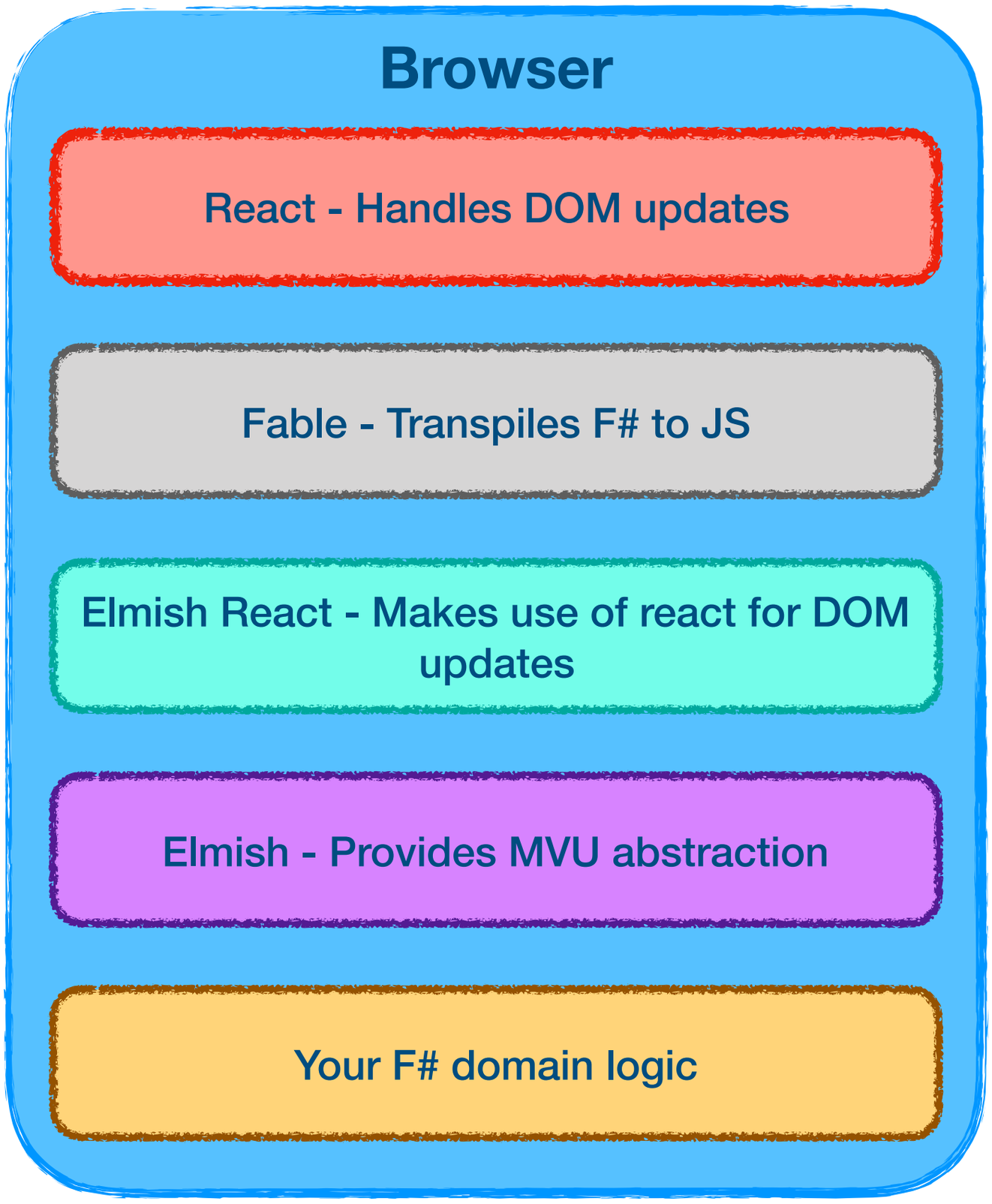
```
<html>
    <body>
    <button onclick="--counter; update();">-</button>
    <div id="counter"></div>
    <button onclick="++counter; update();">+</button>
    <script>
        var counter = 0;

        function update() {
        document.getElementById("counter").textContent = "" + counter;
        }

        update();
    </script>
    </body>
</html>
```

Can you spot some issues?

- **mutating the global variable *counter***
- **coupling business logic to the UI**
- **referencing the DOM element with its name**
- **embedded domain logic directly in the event**

Fable is an F#-to-JavaScript (JS) compiler powered by Babel, designed to produce readable and standard JS code.

Elmish is a set of simple abstractions for writing user interfaces in F# applications in a functional style following the **model-view-update** architecture made famous by Elm.



Model

```
type Model =
    { x : int }

type Msg =
    | Increment
    | Decrement

let init _ = ({ x = 0 }, Cmd.none)
```

Strongly typed

Ready for async

View

```
let view (model:Model) (dispatch:Msg->unit) : Fable.Import.React.ReactElement =
    div []
        [ button [ OnClick (fun _ -> dispatch Decrement) ]
            [ str "-" ]
        div []
            [ str (model.x.ToString()) ]
        button [ OnClick (fun _ -> dispatch Increment) ]
            [ str "+" ] ]
```

Let's dispatch a command

This is a Fable.React binding

Update

```
let update (msg:Msg) (model:Model) : Model * Cmd<Msg> =
    match msg with
    | Increment -> ({ x = model.Value + 1 }, Cmd.none)
    | Decrement -> ({ x = model.Value - 1 }, Cmd.none)
```

This match is exhaustive

Can match with conditions

Glue it together

Creates a message loop

Produces new commands

```
Program.mkProgram init update view
#if DEBUG
|> Program.withDebugger
#endif
|> Program.withReact "fable-elmish-counter"
|> Program.run
```

Where to start?

dotnet template

```
dotnet new fable-elmish-react -n myAwesomeApp -lang f#
cd myAwesomeApp
yarn install
npx webpack-dev-server
```

or just (command/ctrl) + shift + B in Ionide

REPL

F# HTML CSS

Live sample Code

```
open Fable.Core.JsInterop
open Fable.Helpers.React
open Fable.Helpers.React.Props
open Elmish
open Elmish.React

type Model =
    { x : int }

type Msg =
    | Increment
    | Decrement

let init () =
    ({ x = 0 }, Cmd.none)

let update msg model =
    match msg with
    | Increment ->
        ({ model with x = model.x + 1 }, Cmd.none)
    | Decrement ->
        ({ model with x = model.x - 1 }, Cmd.none)

let view model dispatch =
    div []
        [ button [ OnClick (fun _ -> dispatch Decrement) ] [ str "-" ]
          div [] [ str (model.x.ToString()) ]
          button [ OnClick (fun _ -> dispatch Increment) ] [ str "+" ] ]

open Elmish.React

// App
Program.mkProgram init update view
|> Program.withReact "elmish-app"
|> Program.run
```

Problems

-
1
+

Console

Iframe loaded

https://bit.ly/2VmnD71

Html to Elmish

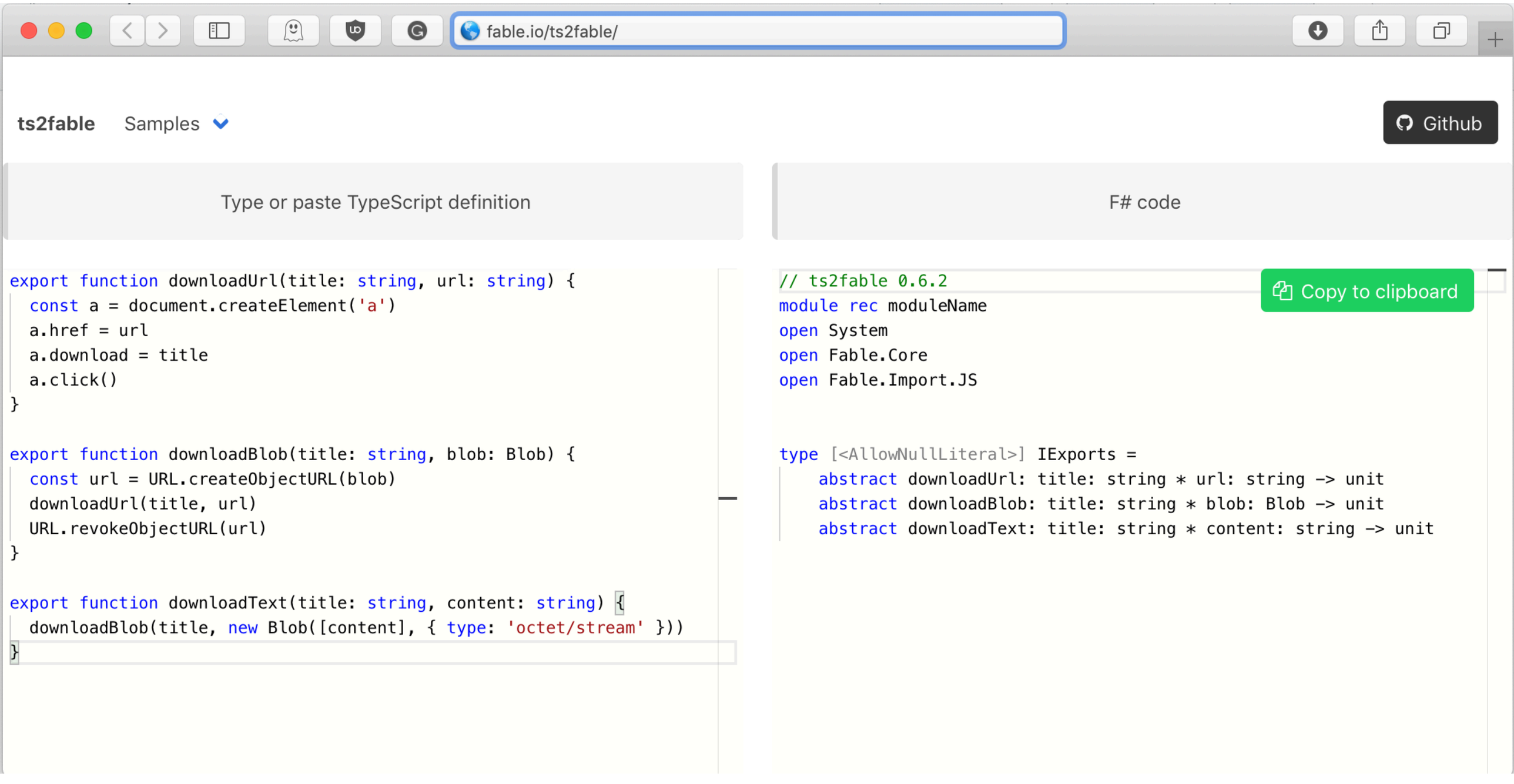
When things are going sour…

- Fable compiler cannot compile some frameworks to JS (FSharp.Data, System.IO etc)

- Porting native JS libraries can be super simple or extremely difficult

- Documentation is fragmented

ts2fable


fable.io/ts2fable/
ts2fable
Samples
Github
Type or paste TypeScript definition
F# code
export function downloadUrl(title: string, url: string) {
const a = document.createElement('a')
a.href = url
a.download = title
a.click()
}
export function downloadBlob(title: string, blob: Blob) {
const url = URL.createObjectURL(blob)
downloadUrl(title, url)
URL.revokeObjectURL(url)
}
export function downloadText(title: string, content: string) {
downloadBlob(title, new Blob([content], { type: 'octet/stream' }))
}
// ts2fable 0.6.2
Copy to clipboard
module rec moduleName
open System
open Fable.Core
open Fable.Import.JS
type [<AllowNullLiteral>] IExports =
abstract downloadUrl: title: string * url: string -> unit
abstract downloadBlob: title: string * blob: Blob -> unit
abstract downloadText: title: string * content: string -> unit

Fabulous bonus

```
type Model =
  { x : int }

type Msg =
  | Increment
  | Decrement

let init () = ({ x = 0 }, Cmd.none)
```

```
let update msg model =
  match msg with
  | Increment -> ({ x = model.x + 1 }, Cmd.none)
  | Decrement -> ({ x = model.x - 1 }, Cmd.none)

let view (model: Model) dispatch =
  View.ContentPage(
    content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Cent
      children = [
        View.Label(text = sprintf "%d" model.x, horizontalOptions = LayoutOptions.C
        View.Button(text = "Increment", command = (fun () -> dispatch Increment), h
        View.Button(text = "Decrement", command = (fun () -> dispatch Decrement), h
      ]))

let program = Program.mkProgram init update view
```

“Yesterday I did anon record support in FCS. Today it got integrated to VF# master. @ChetHusk integrated and made a new FCS release, @k_cieslak made a new Ionide release (I think), @alfonsogcnunez made a new Fable release. What an amazing community!”

–Don Syme

- Safe Stack - https://safe-stack.github.io/
- Fable - https://fable.io
- WTF# Elmish - https://wtfsharp.net/wtf-is-elmish
- Fulma - https://github.com/Fulma/Fulma
- F# Interop with Javascript in Fable - https://bit.ly/2zsW9Fh

- Tips for working with Elmish - https://bit.ly/2IFQou1
- Sample project - https://github.com/poborin/fable-elmish-counter
- Sample online REPL - https://bit.ly/2VmnD71

Thank you!