Elm,

pour la construction d'une application web moderne

Xavier Van de Woestyne

#MIXIT18

Xavier Van de Woestyne

twitter.com/vdwxv github.com/xvw xvw.github.io

Elixir + Elm

«Useless software with useful languages»

Meetup Lillois !

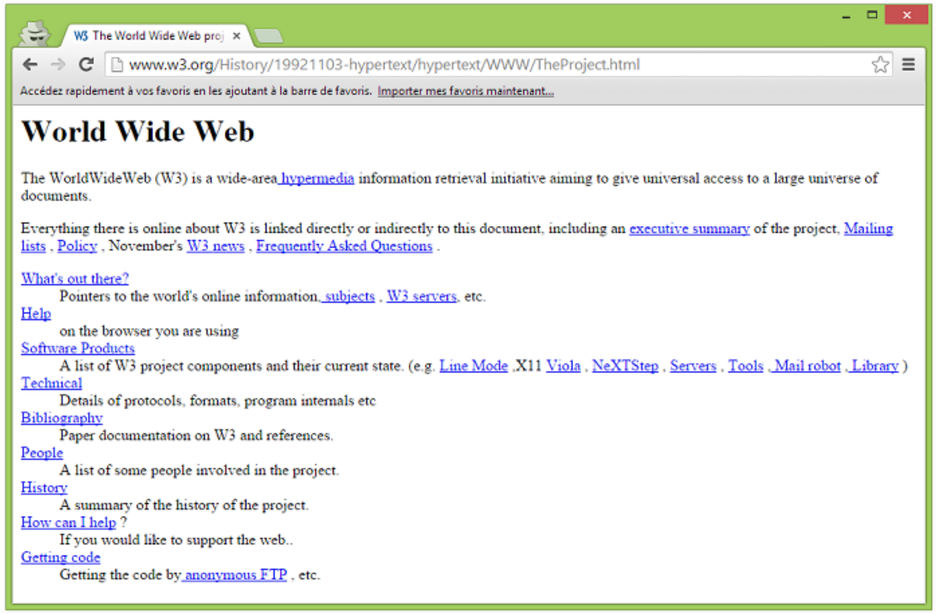
Nim
Java Elm
Ur/web Racket
OCaml
Erlang/Elixir Io
F# Haskell Go
Ruby Idris
PHP HaXe

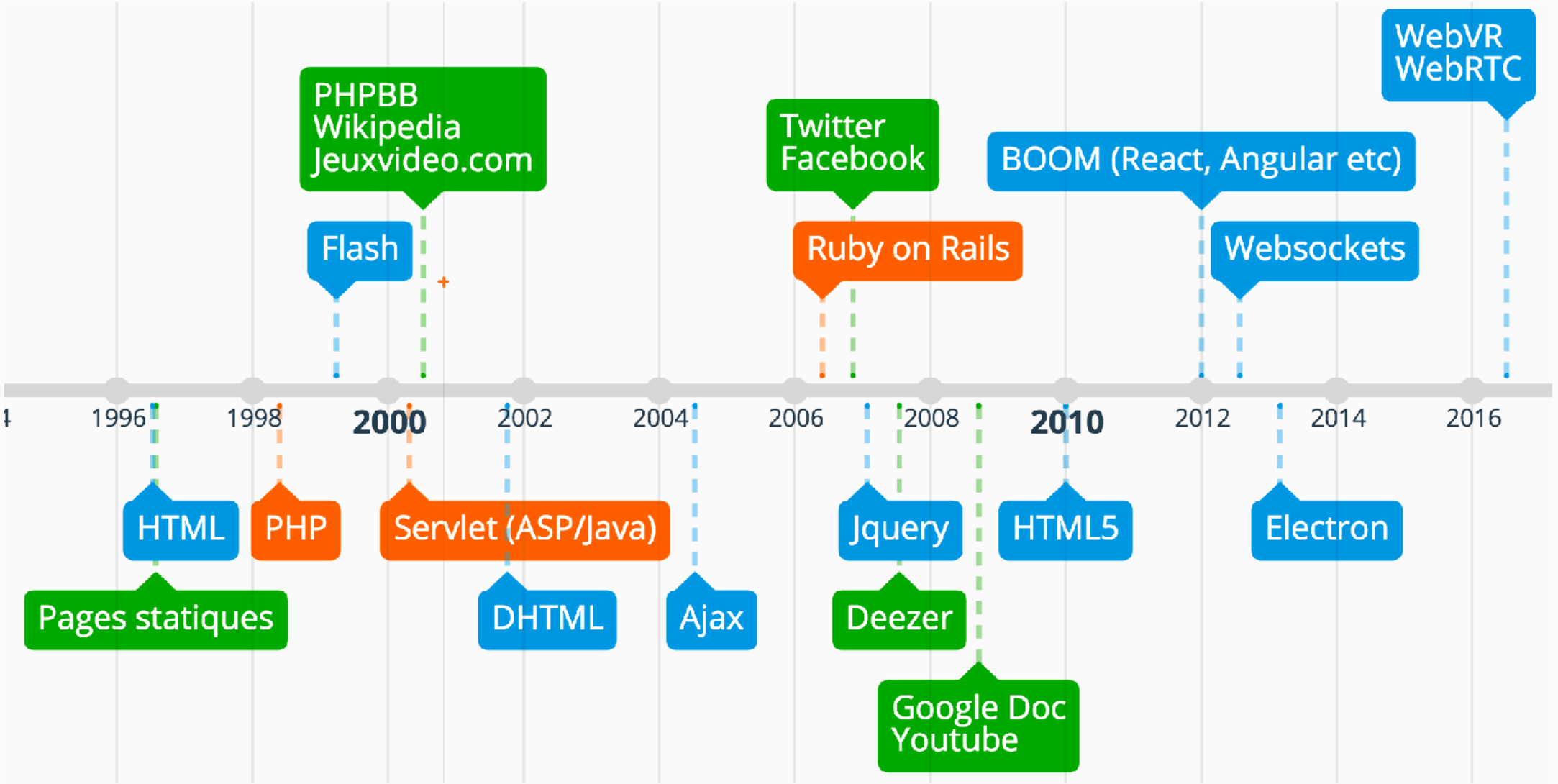
Objectifs !

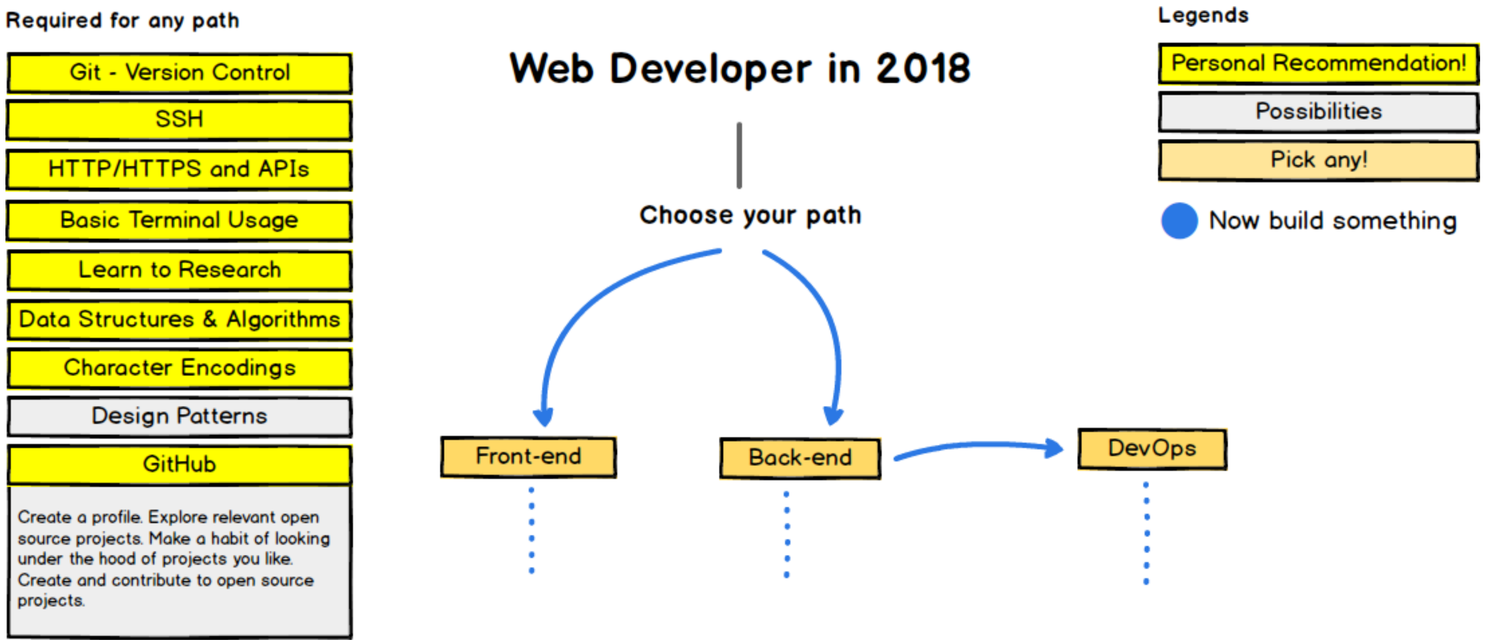
- Présenter un langage que j'aime bien ;
- détruire certains aprioris ;
- justifier certains choix de design ;
- être « *tout-terrain* ».

Le Web

un espace où tout change vite

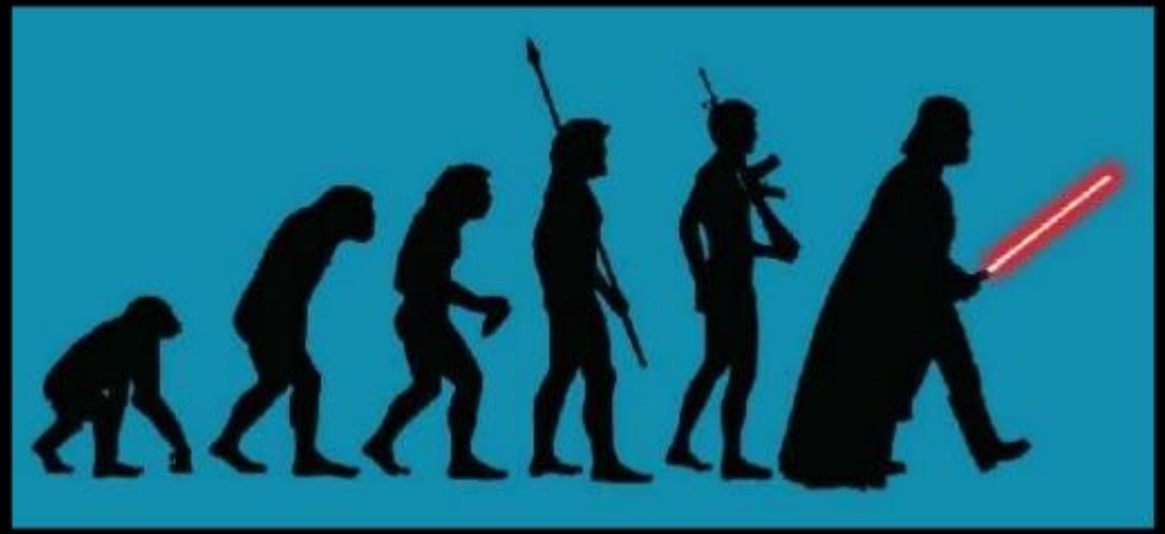


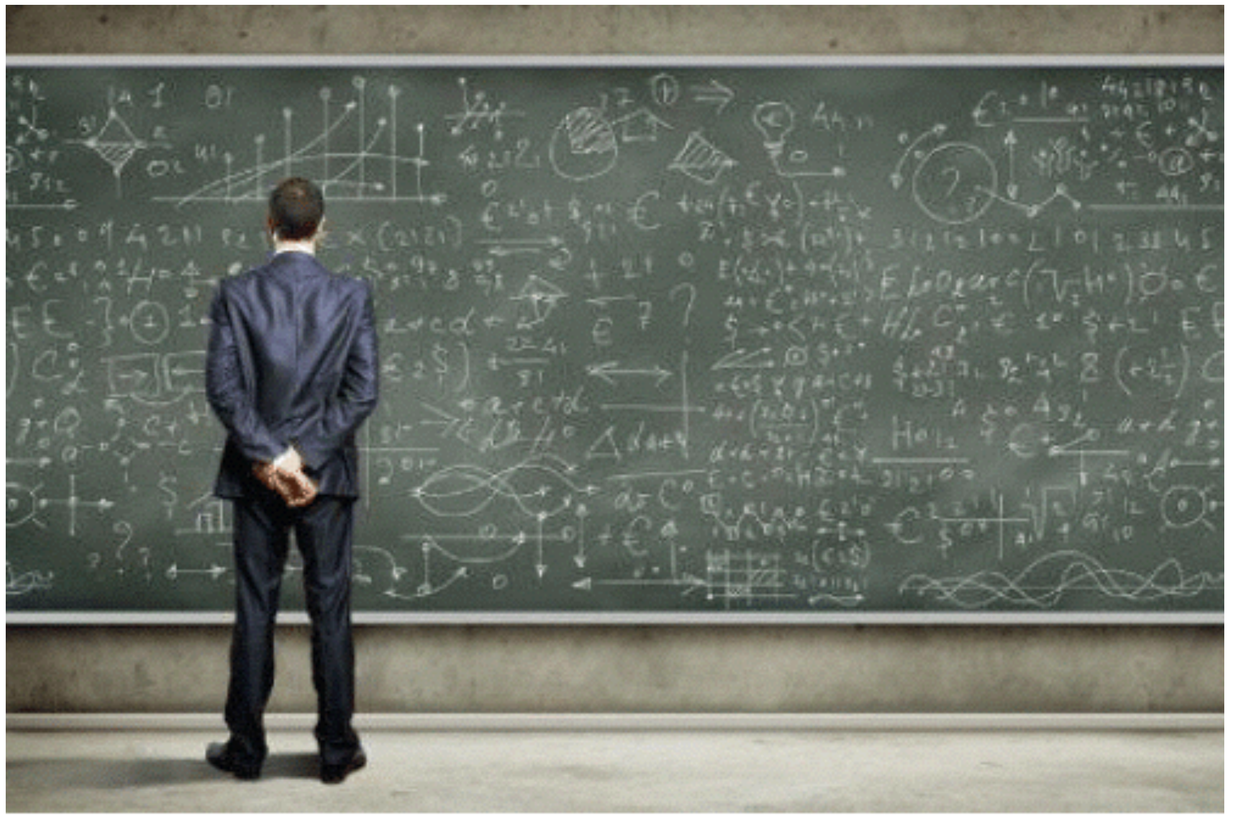

WebVR
WebRTC
PHPBB
Wikipedia
Jeuxvideo.com
Twitter
Facebook
BOOM (React, Angular etc)
Flash
Ruby on Rails
Websockets
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
HTML
PHP
Servlet (ASP/Java)
Jquery
HTML5
Electron
Pages statiques
DHTML
Ajax
Deezer
Google Doc
Youtube



source : https://github.com/kamranahmedse/developer-roadmap

EVOLUTION OF JS



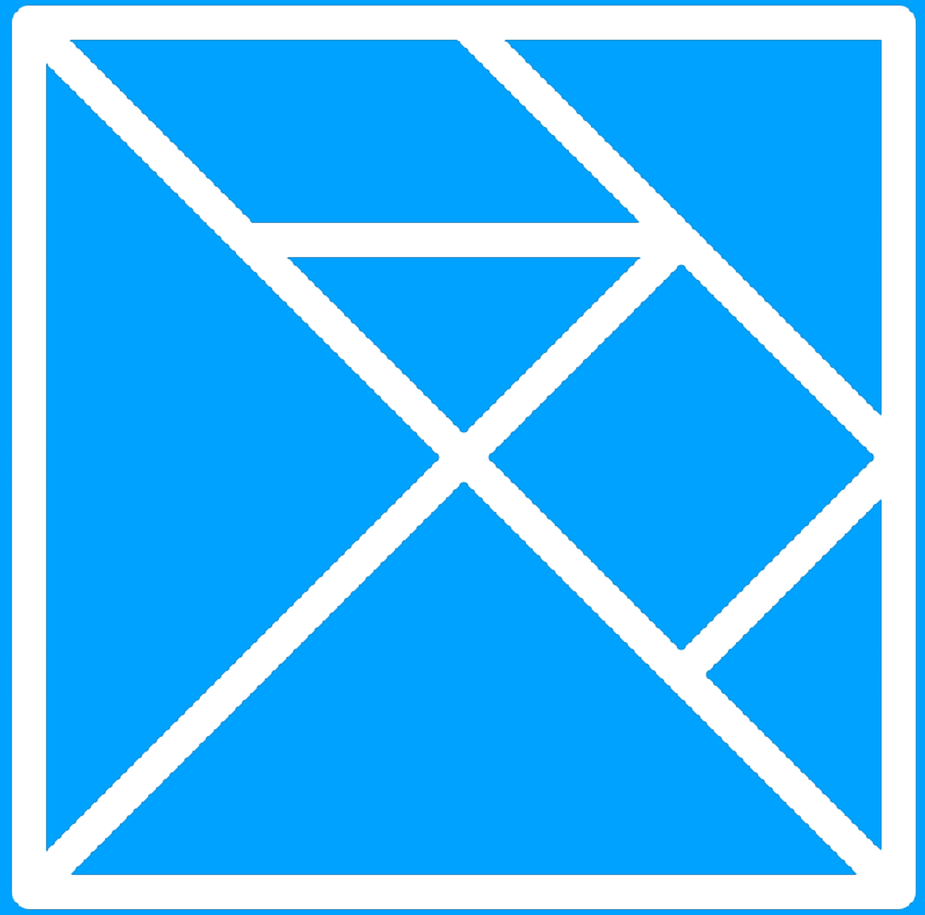


Marc was almost ready to implement his "Hello World" React app.

API first

Tierless

Elm



- Créé en 2012 par Evan Czaplicki ;
- destiné à construire des interfaces graphiques ;
- concurrentML à JavaScript ;
- fonctionnel strict et statiquement typé ;
- développé comme un **langage ML** (à la Miranda).

Evan Czaplicki
@czaplic

Abonné

@vdwxv better question: why did Haskell move away from the ML syntax? They expected list operations to be more common than types. Misjudged.

À l'origine en anglais

RETWEETS	J'AIME
15	24

10:58 - 28 juil. 2016

6 15 24

- Peu de constructions (mais expressive) ;
- un système de module pauvre ;
- (presque) pas de Typeclasses ;
- arrive avec beaucoup d’outils ;
- expose son propre framework.

Impose de bonnes pratiques

- Compilation (et analyse statique) ;
- versionnement sémantique imposé (et utilisé) ;
- syntaxe à la ML (:troll:) ;
- formatage unifié du code.

Un système de type expressif

```
int x = 99 ;
```

Sommes

```
type Direction
    = North
    | East
    | West
    | South

type Maybe a
    = Just a
    | Nothing
```

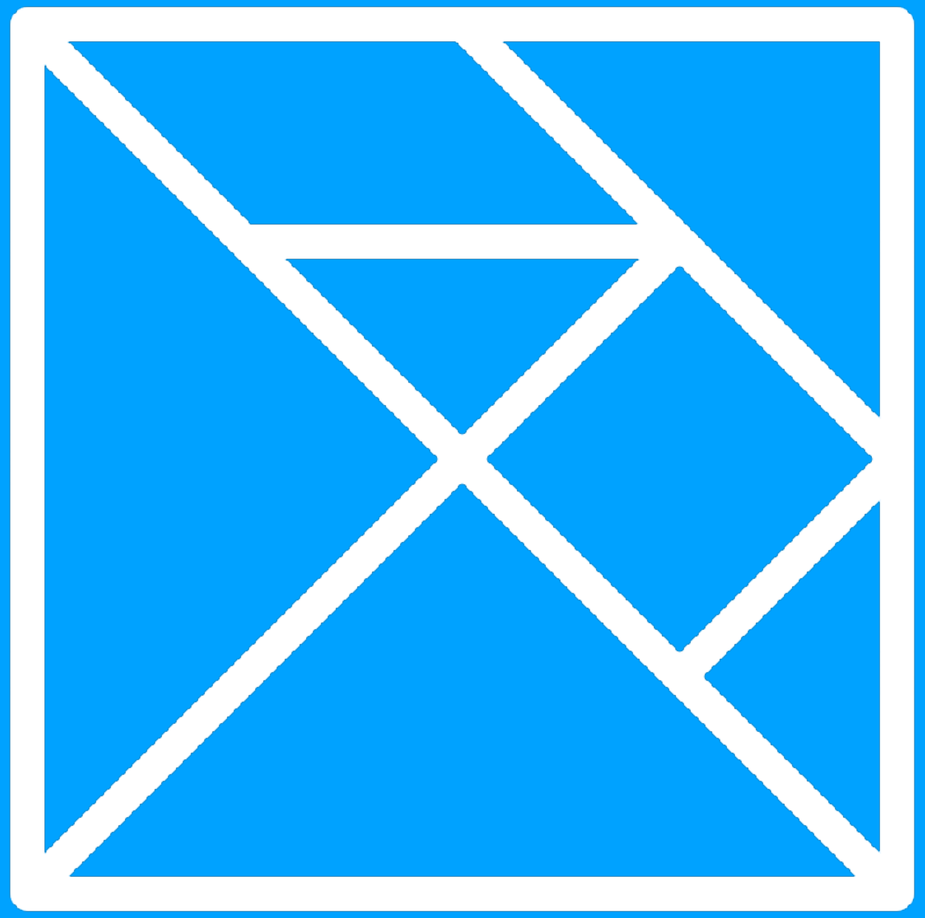
Produits

```
type alias Point =
   (Int, Int)

type alias Point2 =
   { x : Int
   , y : Int
   }
```

Alias

```
type alias Firstname =
    String
```

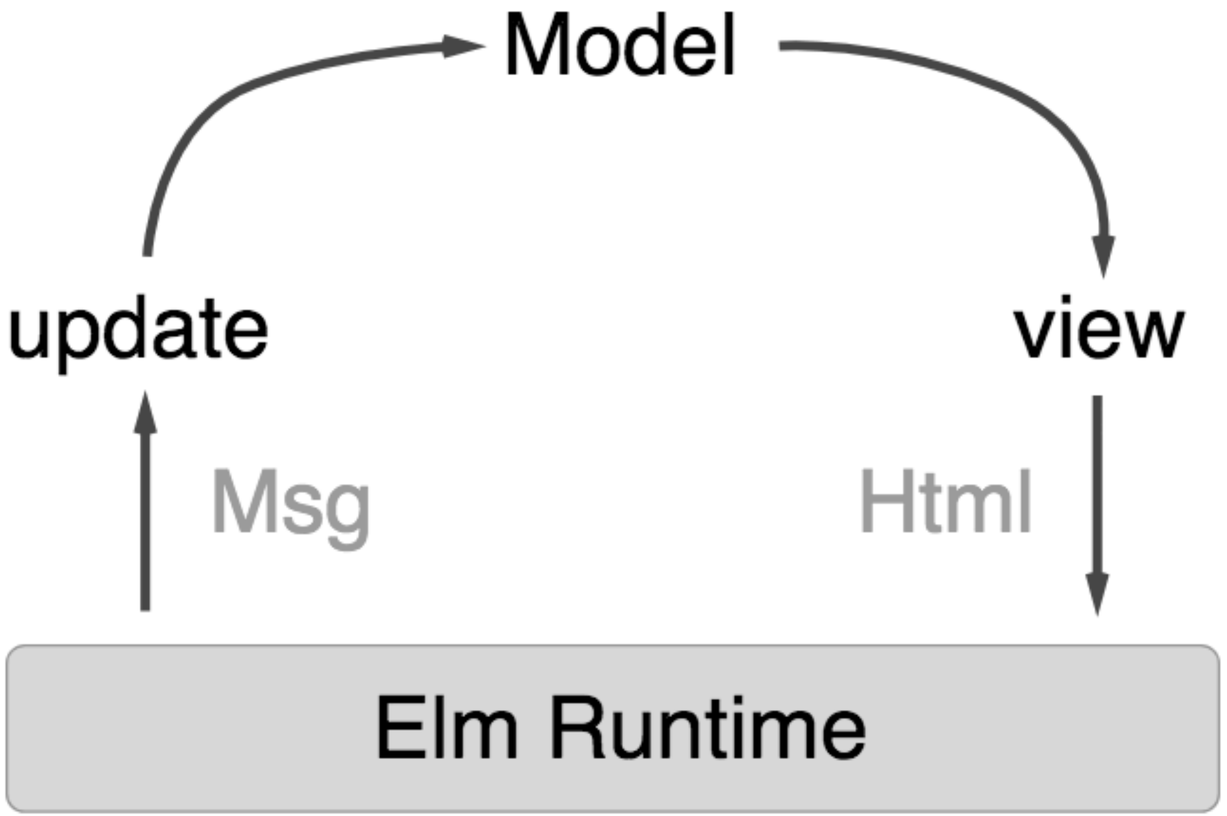
```
Detected errors in 1 module.
================================== ERRORS ==================================


-- MISSING PATTERNS ------------------------------------------------ test5.elm

This `case` does not have branches for all possibilities.

41|>  case msg of
42|>    More ->
43|>      (model, Cmd.none)

You need to account for the following values:

    Main.FetchSucceed _
    Main.FetchFail _

Add branches to cover each of these patterns!

If you are seeing this error for the first time, check out these hints:
<https://github.com/elm-lang/elm-compiler/blob/0.17.1/hints/missing-patterns.md>
The recommendations about wildcard patterns and `Debug.crash` are important!
```



La Elm-architecture

Un langage ML qui ressemble à Haskell ?

Monades
Signaux
Arrows
Joinades


Model
update
view
Msg
Html
Elm Runtime

re-render

dirty nodes

DOM virtuel

DOM incrémental

```
main : Platform.Program Flags Model Message
main =
    Html.program
        { init : (Model, Cmd Message)
        , update : Message -> Model -> (Model, Cmd Message)
        , view : Model -> Html Message
        , subscriptions = Model -> Sub Message
        }
```

Hello World de Elm : un compteur

1ère étape : le model

```
type alias Model = Int
```

2nd étape : l'ensemble de nos actions

```
type Message
    = Increment
    | Decrement
```

Ensuite, l'initialisation et l'update

```
init : (Model, Cmd Message)
init =
    (0, Cmd.none)

update : Message -> Model -> (Model, Cmd Message)
update message model =
    case message of
        Increment ->
            (model + 1, Cmd.none)
        Decrement ->
            (model - 1, Cmd.none)
```

La vue

```
view : Model -> Html Message
view model =
    div []
      [ button [ onClick Decrement ] [ text « - »]
      , text (toString model)
      , button [ onClick Increment ] [ text « + »]
      ]
```

Le programme

```
main : Platform.Program Flags Model Message
main =
    Html.program
        { init = init
        , update = update
        , view = view
        , subscriptions = (\_ -> Sub.none)
        }
```

Sur la morphologie des messages et des modèles

FRP : Commandes et Souscriptions

- `Cmd message` : requête un effet
- `Sub message` : remonte le résultat d'un effet sous forme de message

En dehors du monde «*typesafe* »

- Flags : permet de « passer des arguments à un programme Elm »
- Ports : mécanisme de FFI articulé autour de la Elm architecture

```
port requestTree : File.Path -> Cmd msg
port retreiveTree : (File.Tree -> msg) -> Sub msg
```

```
import elm from '../../src/Main.elm'
const flags = {..}
const container = document.getElementById('app');
const elmApp = elm.Main.embed(container, flags);
elmApp.ports.requestTree.subscribe((pwd) => {
   const tree = onRécupèreLeTreeEnJavaScript
   elmApp.ports.retreiveTree.send(tree)
});
```

Monadic parsers

JSON.parse

Et les composants ?

Les plus

Les moins

Fin, merci :)