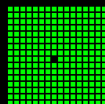


Making web apps with the MVU pattern

(Mr. Math, SPISE MISU ApS)

2022-11-07 @ Girls in IT (virtual edition)

Girls
in IT
Zealand

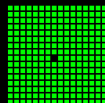


Overview

- About me (very shortly)
- MVU pattern
 - Background
 - The tool (Fable)
 - Demo (live coding)

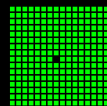
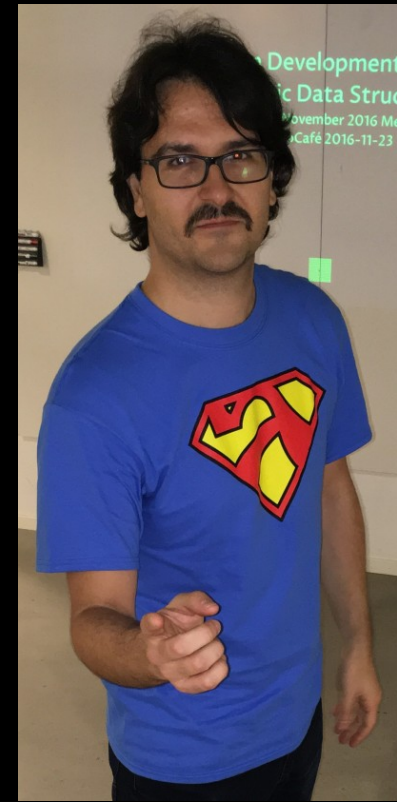
Note: Slides are released under the CC BY-SA license

- Creative Commons Attribution-ShareAlike (“copyleft”)



About me (very shortly)

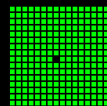
- **Mr. Ramón Soto Mathiesen** (*Spaniard* + *Dane*)
- MSc. Computer Science and minors in Mathematics (KU + Pisa)
- **CompSci** @ SPISE MISU ApS
 - Trying to solve EU GDPR with a scientific approach (technical means)
 - <https://uniprocess.org>
 - Permissive copyleft license (LGPL-3.0)
 - Mostly with **Haskell** and to a lesser extent **Elm** and **PureScript**
 - (Auto) sponsor R&D by doing freelance work
- Blog: <http://blog.stermon.com/> (slides under /talks/)
- PureScript / Elm / Haskell / TypeScript / F# / OCaml / Lisp / C++ / C# / JavaScript



Matching of expectations

- In this talk, I will try to showcase how we can make (react) web apps in a sound (mind) and safe (application) manner by using the MVU pattern, used by Elm and Fable (Elmish)
- There will be shown a few slides at the beginning and then some at the final, to recapitulate, but other than that, we are going to be “live-coding”, with the risks that this entails

Note: As we have about two hours for this session, please don't hesitate to interrupt whenever you have a question

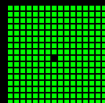


Background

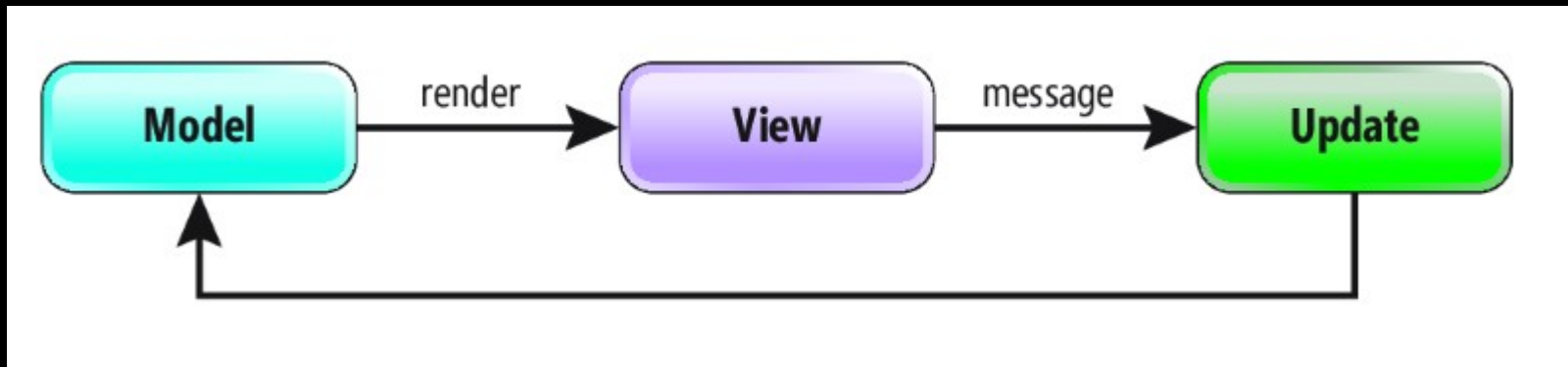


```
const MessagesDropdown = (props) => {
  const visibility = props.open ? 'visible' : '';
  const totalUnseenCount = props.threads.reduce((memo, t) => {
    return memo + (t.unseen ? 1 : 0);
  }, 0);
  let messageText = 'Messages';
  if (totalUnseenCount > 0) {
    messageText += ' (' + totalUnseenCount + ')';
  }
  return (
    <div
      className={'ui scrolling dropdown ' + visibility + ' item'}
      onClick={props.onDropdownClick}
    >
      {messageText} <i className='dropdown icon'></i>
      <div className={'menu transition ' + visibility}>
        <ThreadListCompact
          threads={props.threads}
        />
      </div>
    </div>
  );
};
```

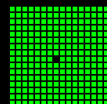
What's happening in the following JS-React snippet?



Background



The **Model-View-Update** Pattern
(MSDN Magazine Issues 2019 October)



The tool (Fable)

Simple and intuitive (readable)



A look at Microsoft Orleans through Erlang-tinted glasses

Some time ago, Microsoft announced Orleans, an implementation of the actor model in .Net which is designed for the cloud environment where instances are ephemeral.

We're not sure if this is a good fit, and whether or not it really promises around scalability, performance and reliability. Below is an account of my personal views having downloaded the SDK and looked through the samples and followed through Richard Anthony's Pluralsight course.

OR

As a long-time Erlang user, Orleans looks like a natural fit. However, I have some concerns regarding its design decisions.

For starters, it's not partition tolerant towards partitions to the data store used for its Site management. Should a partitioned or suffer an outage, it'll result in a full outage of your service. These are not traits of a masterless system that is desirable when you have strict uptime requirements.

When everything is working, Orleans guarantees that there is only one instance of a virtual actor running in when a node is lost the cluster's knowledge of nodes will diverge and during this time the single-activation will eventually consistent. However, you can provide stronger guarantees yourself (see Site Management section below).

Orleans uses at-least-once message delivery, which means it's possible for the same message to be sent to a receiving node in under load or simply fails to acknowledge the first message in a timely fashion. This again, is something you can mitigate yourself (see Message Delivery Guarantees section below).

Finally, its task scheduling mechanism appears to be identical to that of a naive event loop and exhibits all the fallacies of an

```
public void DoSomething(int x, int y)
{
    Foo(y,
        Bar(x,
            Zoo(Monkey())));
}
```

1. left-to-right

How we read English

2. top-to-bottom

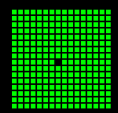
2. bottom-to-top

How we read code

1. right-to-left

```
1. left-to-right
let doSomething x y =
    monkey() |> zoo
    |> bar x
    |> foo y
2. top-to-bottom
```

- Forced indentation, just like Python, in combination with |> operator, makes it easy to read again & again



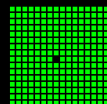
The tool (Fable)

Functions as first-class citizens



- Higher-order functions.
 - Passing functions as arguments is just like gluing functions together, composing them to bigger ones:

```
[0 .. 9] |> List.map (fun x -> x + x)
```



The tool (Fable)

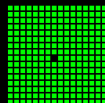
Strongly type-safe



- Computer says no:



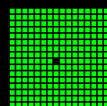
```
(* error FS0001: The type 'string' does not match the type 'int' *)  
let result = 42 + "42"
```



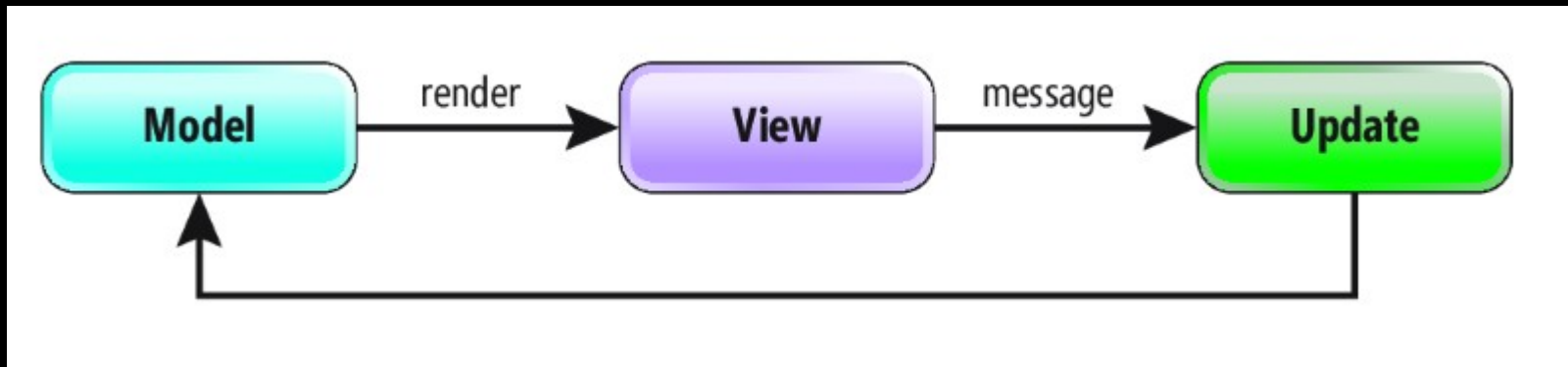
Demo (live coding)



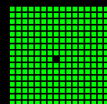
- Make a basic “Hello, World!” web-app
- Transition to a “Counter” web-app
- Transition to a “Random Doggo image” web-app
 - <https://random.dog/woof.json>
- Transition to a “Voting areas in CPH” web-app
 - <https://www.opendata.dk/city-of-copenhagen/...>



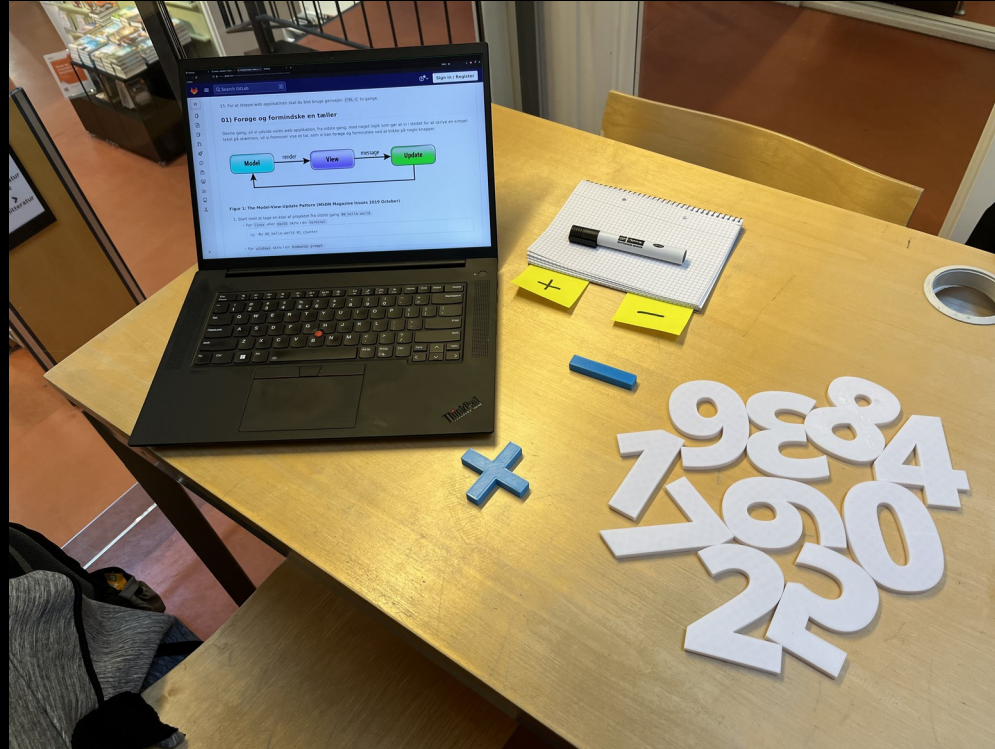
Demo (live coding)



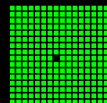
The **Model-View-Update** Pattern
(MSDN Magazine Issues 2019 October)



Summary

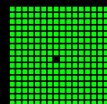


Explaining MVU-pattern IRL (3D print + notebook)



Summary

- Once the MVU-pattern is understood, it's just re-applying the same approach again-and-again
- When having to deal with 3rd party systems, the update function changes a bit as it now has to deal with external effects
- Trivially going from JSON payload to sound code (with types)
- Want to give it a try?
 - Try Fable:
 - <https://fable.io/repl/>
 - Try Elm:
 - <https://elm-lang.org/try>
 - Coding Pirates material:
 - <https://gitlab.com/codingpirates/web-app-workshop>
Note: Material in Danish, please use Google Translator



Q&A

Any questions?

