MVI
an architecture for reactive programming
HELLO!

I am Luca Mezzalira

Solutions Architect @ DAZN
Google Developer Expert on Web Technologies
Community Manager of London JavaScript User Group
AGENDA

- Reactive programming
- Model - View - Intent
- Cycle.js
1. REACTIVE PROGRAMMING
PROGRAMMING PARADIGMS

IMPERATIVE
It is a programming paradigm that uses statements that change a program's state. In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform. Imperative programming focuses on describing how a program operates.

FUNCTIONAL
It is a declarative programming paradigm, which means programming is done with expressions. In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result f(x) each time.

REACTIVE
It is a programming paradigm oriented around data flows and the propagation of change. This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow.
PASSIVE

A → B

REACTIVE

A → B
Imperative vs Reactive
OBSERVER PATTERN

The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems.
STREAMS

A stream is a sequence of ongoing events ordered in time. It can emit three different things: a value (of some type), an error, or a "completed" signal.

EVERYTHING CAN BE A STREAM
2. MODEL VIEW INTENT
**Input**: data events from the Model.

**Output**: a Virtual DOM rendering of the model, and raw user input events (such as clicks, keyboard typing, accelerometer events, etc).
INPUT: raw user input events from the View.
OUTPUT: model-friendly user intention events.
MODEL

Input: user interaction events from the Intent.
Output: data events.
MVI RULES

▸ A module shouldn’t control any other module (controller in MVC)
▸ The only shared part between modules are observables
▸ Intent is a component with only one responsibility: It should interpret what the user is trying to do in terms of model updates, and export these "user intentions" as events
3. Cycle.js
Cycle.js
Core concepts

- Pure Functions
- Streams
- Components
- Drivers
Drivers are functions that listen to Observable sinks (their input), perform imperative side effects, and may return Observable sources (their output).

http://cycle.js.org/drivers.html
Cycle Diversity

- Refactored in Typescript
- It accepts different type of reactive libraries (xstream, rxjs 4 and 5, most)
- Integration with Snabbdom for DOM rendering by default

https://github.com/cyclejs/cyclejs/releases/tag/v7.0.0
Snabbdom-jsx
https://github.com/yelouafi/snabbdom-jsx

React
https://github.com/pH200/cycle-react

React-Native
https://github.com/cyclejs/cycle-react-native
Cycle.js in action
Shall we use Reactive Programming in any project?