Next generation of FRONT END architectures
HELLO!

I am Luca Mezzalira

Solutions Architect @ DAZN
Google Developer Expert on Web technologies
Community Manager of London JavaScript Community
AGENDA

▸ Front end architectures overview

▸ Communicating sequential processes

▸ Reactive programming

▸ MVI
1. ARCHITECTURES PAST & PRESENT
Architectures Timeline

80s  MVC
90s  MVP
2005 MVVM
2009 DCI
2013 FLUX
2015 MVI
What are we trying to solve with event driven architectures?
OBJECTS COMMUNICATION

- Events
- Signals
- Pub/Sub system
- Actions + Dispatcher
- ...

Events
Signals
Pub/Sub system
Actions + Dispatcher
...
2. COMMUNICATING SEQUENTIAL PROCESSES
CSP

- Created in 1978
- CSP-js porting from Clojure (core.async) and GO
- Composition over Inheritance
- Easier to test
- Loose coupled
- Well encapsulated
- Clean and reusable code
TRANSFORM: producing some value from another

REDUCER: combining values of a data structure to produce a new one
CSP APIs

- set a buffer inside a channel
- put and take value from a channel
- close a channel
- close a channel after N ms
- put the array values inside a channel
- return a channel with the values inside an array
- split, merge or pipe channels
- pub/sub system (observer pattern)
- broadcast values to multiple channels simultaneously
- map, filter, remove data from a channel
- avoid duplicated data inside a channel
- decorate data inside a channel
2. 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.
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
COLD OBSERVABLES

Cold observables start running upon subscription: the observable sequence only starts pushing values to the observers when subscribe is called.

Values are also not shared among subscribers

Usually unicast
HOT OBSERVABLES

When an observer subscribes to a hot observable sequence, it will get all values in the stream that are emitted after it subscribes.

The hot observable sequence is shared among all subscribers, and each subscriber is pushed the next value in the sequence.

Usually multicast
4. MODEL VIEW INTENT
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
Shall we use Reactive Programming in any project then?
THANKS EVERYONE

Q&A

@lucamezzalira
www.lucamezzalira.com
www.londonjs.uk
mezzalab@gmail.com