Model View Controller meets Monad

“ ***It is all about composition*** ”

Goals

- show an end-to-end **functional** application
- leverage some well-consolidated functional library
- understand limitations (if any) and the improvement

Task

> Task represents a specification for a possibly lazy or asynchronous computation, which when executed will produce an A as a result, along with possible side-effects.

What does it refer you to?

```
trait Task[+A] {
    final def flatMap[B](f: A => Task[B]): Task[B] = ...
    final def map[B](f : A => B): Task[B] = ...
    //some interesting extesions
    def memoize: Task[A] = ...
}
object Task {
    def pure[A](a : A) : Task[A]
    def defer[A](a : Task[A]) : Task[A]
}
```

A Little taste

```
object App extends TaskApp {
    def
}
```

Observable

> a data type for modelling and processing asynchronous and reactive streaming of events with non-blocking back-pressure.

We use it to implement the Functional Reactive Programming

Books

1. **Scala with Cats Book**
2. **Category Theory for Programmers**
3. **Functional Reactive Programming**

References