@ControllerAdvice

Spring Framework

DO NOT READ THIS POST if you don’t care about **Global Exception handling** in a Spring Framewok application, **otherwise you are welcomed!**

@mauricioperez

→

@mauricioperez

01

Exception Handling: You are the god

How are you going to be able to be the god on a Spring Framework application?

Well let’s define first that you become “god” when you know **how something is driven the way it is**, and where does something either good or bad **like an error** come from.

And... basically so can you do with **a global exception handler** if you know how to configure it the right way.

@mauricioperez

02

What is a global exception handler?

Simply put: an exception handler is nothing more than a class that you define with **@ControllerAdvice** annotation, thereby is stored as a bean in Spring's Context and it will be activated whenever any of your classes **throw an exception or something bad occurs to them**.

That's all, but I think **that I might deepen more on this**, so let's take a closer look.

See, an exception handler is seen easy **because of its little complexity of implementation** on a Spring Framework application.

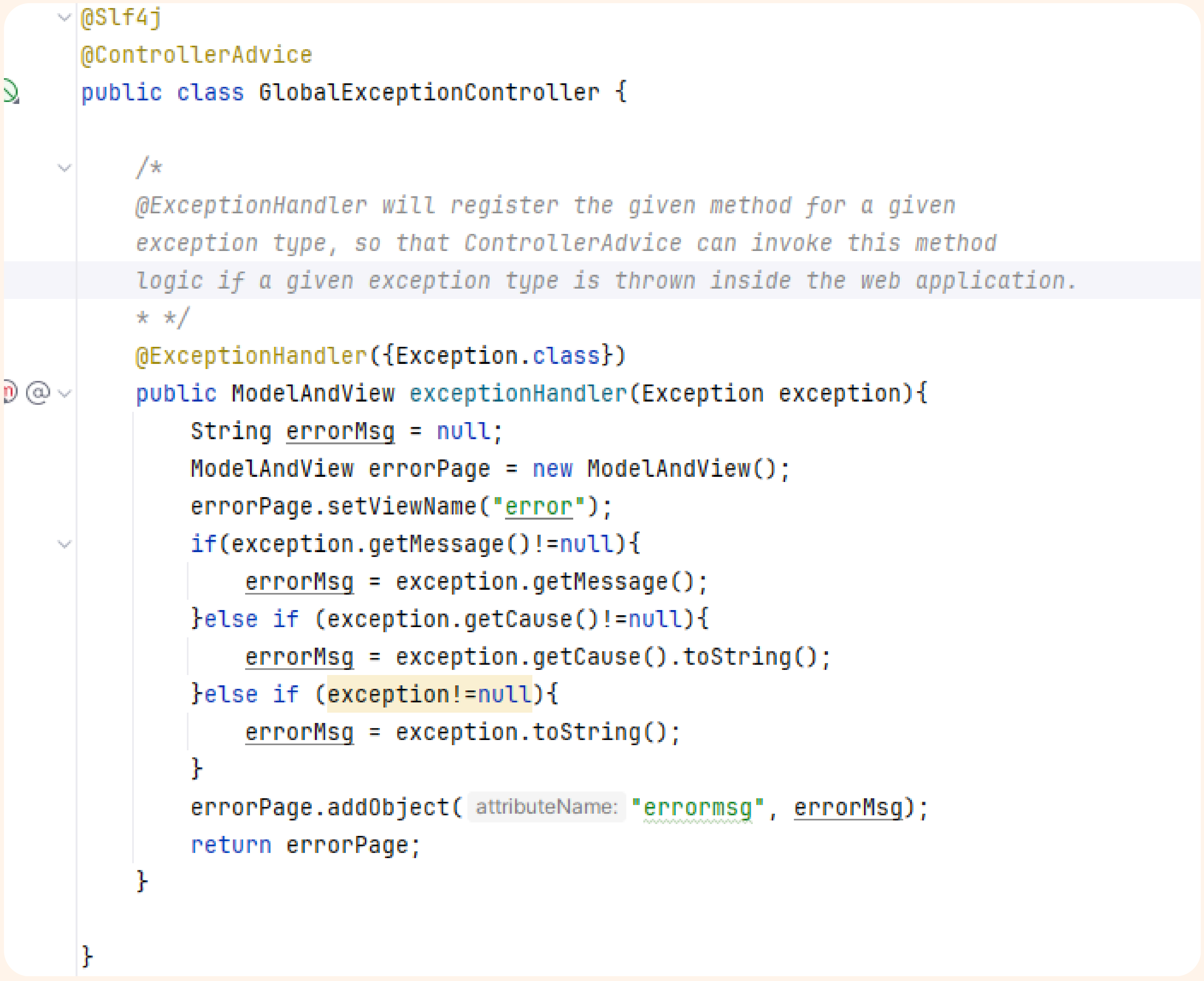
Let's see how it works ⟶

@mauricioperez

03

Example of a global exception handler

The following is a class that I usually implement to my Spring Framework projects:



```java
@Slf4j
@ControllerAdvice
public class GlobalExceptionController {

    /*
    @ExceptionHandler will register the given method for a given
    exception type, so that ControllerAdvice can invoke this method
    logic if a given exception type is thrown inside the web application.
    * */
    @ExceptionHandler({Exception.class})
    public ModelAndView exceptionHandler(Exception exception){
        String errorMsg = null;
        ModelAndView errorPage = new ModelAndView();
        errorPage.setViewName("error");
        if(exception.getMessage()!=null){
            errorMsg = exception.getMessage();
        }else if (exception.getCause()!=null){
            errorMsg = exception.getCause().toString();
        }else if (exception!=null){
            errorMsg = exception.toString();
        }
        errorPage.addObject( attributeName: "errormsg", errorMsg);
        return errorPage;
    }

}
```

Important

@mauricioperez

Spring AOP is the **cornerstone for every backend & highly scalable system** you make with Spring Framework.

I uploaded a video on Spring AOP on my Youtube channel called: "***The Real Code Show***" (link in the description).

If you want to learn more deeply about Spring Framework and Java, please visit my channel: **youtube.com/@TheRealCodeShow**

Subscribe, and I will be glad to help you! I upload videos every week.

(latest video)

THE REAL CODE SHOW

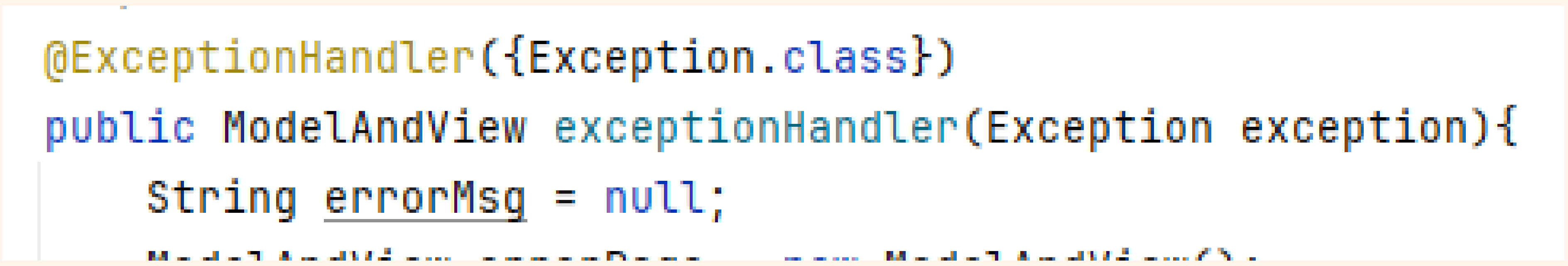
@mauricioperez

04 Composition of a Global Exception Controller

So as you just saw, a global exception handler is nothing more **than a class that behaves like a controller**, because the annotation that uses it at the top of it (@ControllerAdvice) **is a derivated-annotation from @Controller**.

Now, this class can have into it methods, and also methods **need to be annotated with a particular annotation too**: **@ExceptionHandler**

But **it asks for parameters**, look at the following method of my exception controller:



```
@ExceptionHandler({Exception.class})
public ModelAndView exceptionHandler(Exception exception){
    String errorMsg = null;
```

@mauricioperez

05 Methods of a Global Exception Handler

So you saw that the methods into a class that is for intercepting errors globally giving covering to all application **need parameters**.

And basically by these parameters is what you are telling to every method: “Look Spring what I need you to do is **to execute this behaviour that I am tellling you in this method** and you will be executing this whenever an **error occurs based on the parameter I am giving you** on @ExceptionHandler annotation”.

So basically you are going to be intercepting those methods which throw a given exception **that you told in the parameter** of @ExceptionHandler annotation

That’s why it needs parameters. ⟶

@mauricioperez

06

Why would you need an Exception Handler?

Because of these things:

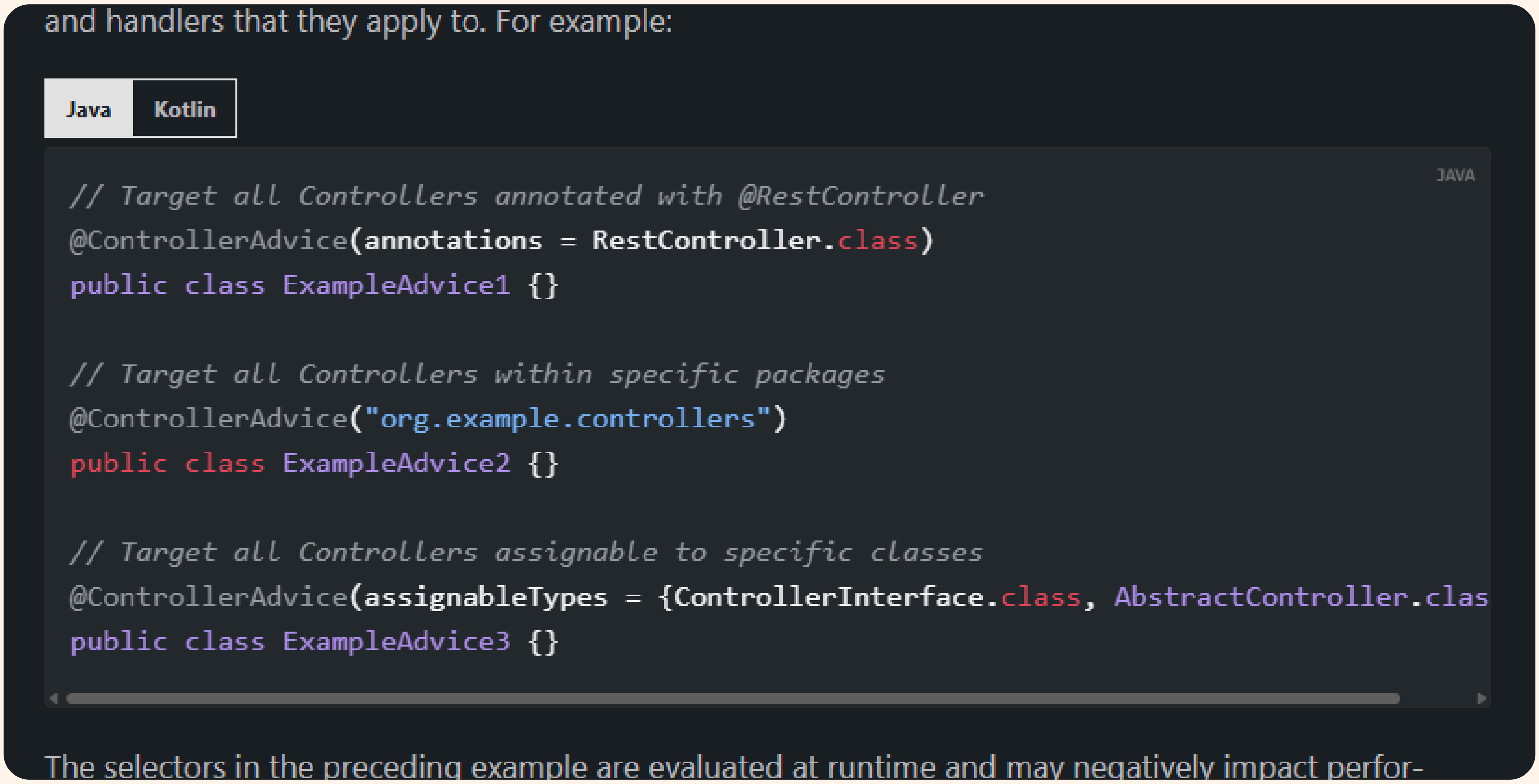
1. You avoid unexpected behaviour from your application due to an exception that was thrown by **some of the millions of methods you may have on your application**
2. **To trace the proceedence of some error** thrown and you quickly act against it to mitigate it
3. **To have an ordered workflow** whenever something gets bad or the user might crack your application and so you have a global perspective to handle something bad.
4. To catch errors dinamically and **see the perspective globally**.
5. To separate cross-cutting concerns: **you avoid the try-catch syntax**.

@mauricioperez

05

Examples from Spring Docs

And here you have another examples from the Spring Framework documentation:



and handlers that they apply to. For example:

```java
// Target all Controllers annotated with @RestController
@ControllerAdvice(annotations = RestController.class)
public class ExampleAdvice1 {}

// Target all Controllers within specific packages
@ControllerAdvice("org.example.controllers")
public class ExampleAdvice2 {}

// Target all Controllers assignable to specific classes
@ControllerAdvice(assignableTypes = {ControllerInterface.class, AbstractController.clas
public class ExampleAdvice3 {}
```

The selectors in the preceding example are evaluated at runtime and may negatively impact perfor-

@mauricioperez

05 Read documentation

If you want to learn more about this cool thing on Spring Framework, I implore you to read the Spring oficial documentation, and obviously follow my Linkedin Profile for more.

Here you have the Spring Documentation for Exception Handler Controllers:

https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-advice.html

Thank you!

@mauricioperez

If you liked it, don't hesitate to react to this post and share it!