



# Moving away from null and exceptions: An alternative way of error handling

*Mario Fernandez*  
*Andrei Bechet*

**ThoughtWorks®**

# What to expect

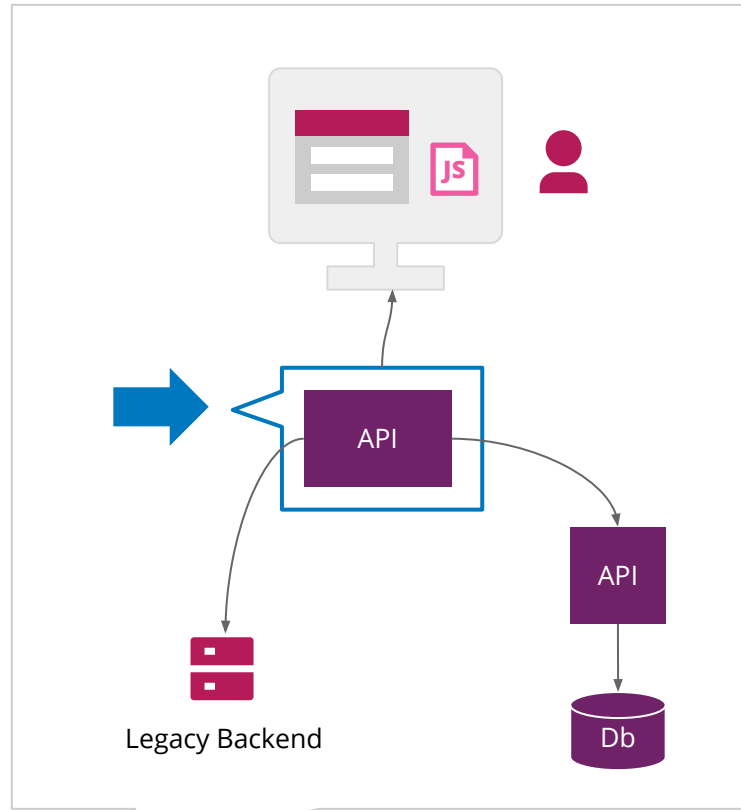
A background image showing a man with a beard and glasses, wearing a lanyard, smiling and interacting with a woman with long blonde hair. The image is overlaid with a semi-transparent purple filter.

# The two takeaways

Let's stop **using null**

Let's stop **abusing** exceptions

# Some Context



Big German Corp

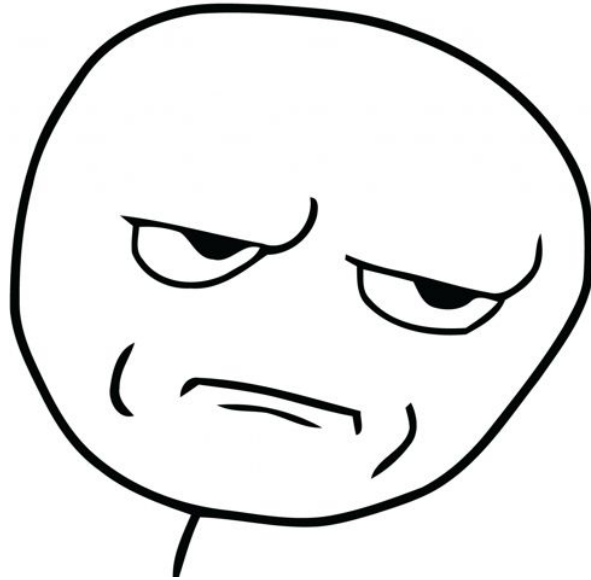






# Null

Did you have a **NullPointerException** lately  
in a production system?



```
public static boolean isAdmin(List<Scope> scopes) {  
    if(scopes == null) {  
        return false;  
    }  
  
    Scope adminScope = findAdminScope(scopes);  
  
    if(adminScope == null) {  
        return false;  
    }  
  
    return adminScope.isValid();  
}
```

# **Ad-hoc Error Handling and it pollutes**

# **Null values sidestep the type system**

# **Runtime instead of compile time feedback**



# Nullable Types



Authorization: Bearer bGciOi...JIUzI1NiIs

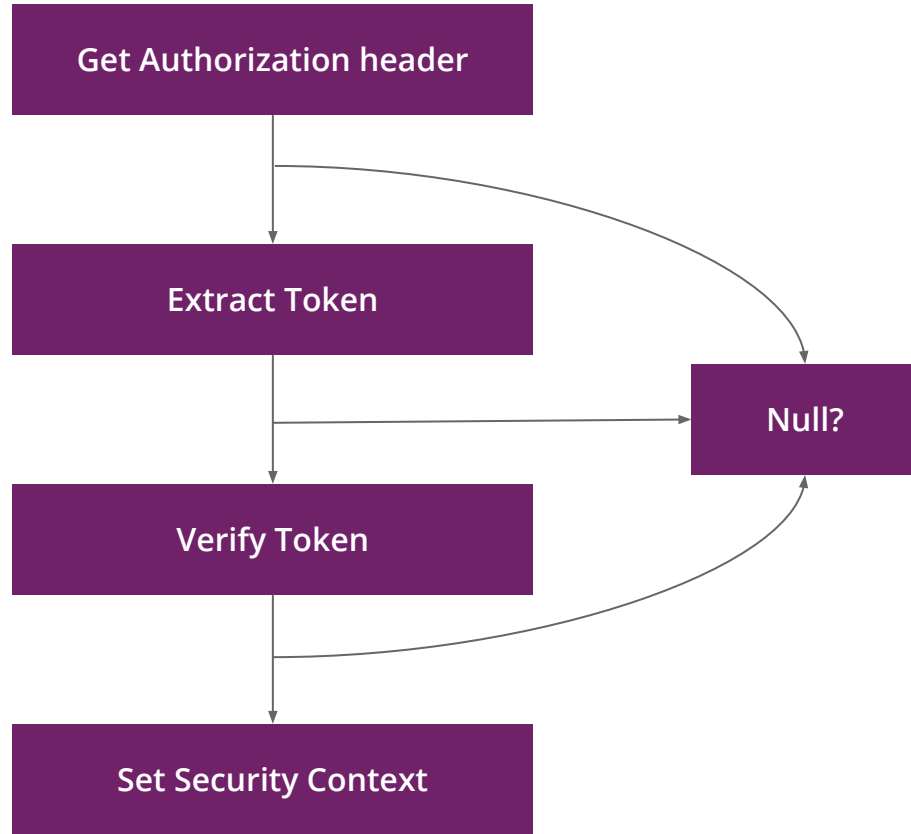
```
fun String.extractToken(): String? = if (startsWith("Bearer"))  
    split(" ").last()  
else  
    null
```

```
header.extractToken()
```

```
?.let { token -> doStuff(token) }
```



# What about more complex flows



```
request.getHeader(Headers.AUTHORIZATION)
    ?.let { header ->
        header.extractToken()
            ?.let { jwt ->
                verifier.verify(jwt)
                    ?.let { token ->
                        SecurityContextHolder.getContext().authentication = token
                    }
            }
        }
    }
```

# Data Types

*A digression about Functional Programming*

A **data type** is an abstraction that encapsulates one reusable coding pattern

Think of **containers** holding your data



Provide clear **semantics** and an **interface**  
to manipulate the data

# Examples

**Option**

**Either**

**Validated**

**IO**

[arrow-kt.io/docs/0.10/datatypes/intro/](https://arrow-kt.io/docs/0.10/datatypes/intro/)

A man with a beard and glasses, wearing a ThoughtWorks lanyard, is smiling and interacting with a woman with long blonde hair. The background is a blurred crowd of people at a conference. The word "Option" is overlaid in white text.

# Option

A value that might be absent



The diagram illustrates the structure of an `Option<T>` type. At the top, a purple rectangle labeled `Option<T>` has a small grey tab on its top edge. Below it are two light grey rectangular containers. The left container holds a purple rectangle labeled `Some<T>` and a purple oval labeled `T` underneath it. The right container holds a purple rectangle labeled `None`.

`Option<T>`

`Some<T>`

`T`

`None`

```
sealed class Option<out T> {  
    data class Some<out T>(val a: T): Option<T>()  
    object None: Option<Nothing>()  
}
```



# How to use it?



```
fun String.extractToken(): Option<String> = startsWith("Bearer ")  
    .maybe { split(" ").last() }
```

```
when (val token = header.extractToken()) {  
    is Option.None -> ResponseEntity.status(401).build()  
    is Option.Some -> ResponseEntity.ok(result.t)  
}
```



**A container is not just  
a holder of data**

```
interface Operations {  
    fun <A, B> Option<A>.map(f: (A) -> B): Option<B>  
    fun <A, B> Option<A>.flatMap(f: (A) -> Option<B>): Option<B>  
}
```

# Exceptions

*The hidden GOTO*

Did you get a **500** lately in a production system?

... pretty common to throw

```

ServletDispatchingHandler.java:36)\n\tat io.undertow.servlet.handlers.RedirectDirHandler.handleRequest(RedirectDirHandler.java:68)\n\tat io.undertow.servlet.handlers.security.SSLInformationAssociationHandler.handleRequest(SSLInformationAssociationHandler.java:132)\n\tat io.undertow.servlet.handlers.security.ServletAuthenticationCallHandler.handleRequest(ServletAuthenticationCallHandler.java:57)\n\tat io.undertow.server.handlers.PredicateHandler.handleRequest(PredicateHandler.java:43)\n\tat io.undertow.security.handlers.AbstractConfidentialityHandler.handleRequest(AbstractConfidentialityHandler.java:46)\n\tat io.undertow.servlet.handlers.security.ServletConfidentialityConstraintHandler.handleRequest(ServletConfidentialityConstraintHandler.java:64)\n\tat io.undertow.security.handlers.AuthenticationMechanismsHandler.handleRequest(AuthenticationMechanismsHandler.java:60)\n\tat io.undertow.servlet.handlers.security.CachedAuthenticatedSessionHandler.handleRequest(CachedAuthenticatedSessionHandler.java:77)\n\tat io.undertow.security.handlers.AbstractSecurityContextAssociationHandler.handleRequest(AbstractSecurityContextAssociationHandler.java:43)\n\tat io.undertow.server.handlers.PredicateHandler.handleRequest(PredicateHandler.java:43)\n\tat io.undertow.servlet.handlers.ServletInitialHandler.handleFirstRequest(ServletInitialHandler.java:269)\n\tat io.undertow.servlet.handlers.ServletInitialHandler.access$100(ServletInitialHandler.java:78)\n\tat io.undertow.servlet.handlers.ServletInitialHandler$2.call(ServletInitialHandler.java:133)\n\tat io.undertow.servlet.handlers.ServletInitialHandler$2.call(ServletInitialHandler.java:130)\n\tat io.undertow.core.ServletRequestContextThreadSetupAction$1.call(ServletRequestContextThreadSetupAction.java:48)\n\tat io.undertow.servlet.core.ContextClassLoaderSetupAction$1.call(ContextClassLoaderSetupAction.java:43)\n\tat io.undertow.servlet.handlers.ServletInitialHandler.dispatchRequest(ServletInitialHandler.java:249)\n\tat io.undertow.servlet.handlers.ServletInitialHandler.access$000(ServletInitialHandler.java:78)\n\tat io.undertow.servlet.handlers.ServletInitialHandler$1.handleRequest(ServletInitialHandler.java:99)\n\tat io.undertow.server.Connectors.executeRootHandler(Connectors.java:376)\n\tat io.undertow.server.HttpServerExchange$1.run(HttpServerExchange.java:830)\n\tat java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)\n\tat java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)\n\tat java.base/java.lang.Thread.run(Thread.java:834)\nCaused by: org.springframework.http.converter.HttpMessageNotReadableException: JSON parse error: Unrecognized field \"businessPartnerId\" (com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException: Unrecognized field \"businessPartnerId\" (com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException), not marked as ignorable; nested exception is com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException: Unrecognized field \"businessPartnerId\" (com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException), not marked as ignorable (0 known properties in \"com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException\", not marked as ignorable (0 known properties in \"com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException\")))\n\tat [Source: (PushbackInputStream); line: 1, column: 863] (through reference chain: com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException[\"businessPartnerId\"])\n\tat org.springframework.http.converter.json.AbstractJackson2HttpMessageConverter.readJavaType(AbstractJackson2HttpMessageConverter.java:245)\n\tat org.springframework.http.converter.json.AbstractJackson2HttpMessageConverter.read(AbstractJackson2HttpMessageConverter.java:227)\n\tat org.springframework.web.client.HttpMessageConverterExtractor.extractData(HttpMessageConverterExtractor.java:104)\n\tat ... 105 common frames omitted\nCaused by: com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException: Unrecognized field \"businessPartnerId\" (com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException), not marked as ignorable (0 known properties in \"com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException\", not marked as ignorable (0 known properties in \"com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException\"))\n\tat [Source: (PushbackInputStream); line: 1, column: 863] (through reference chain: com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException[\"businessPartnerId\"])\n\tat com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException.from(UnrecognizedPropertyException.java:61)\n\tat com.fasterxml.jackson.databind.DeserializationContext.handleUnknownProperty(DeserializationContext.java:840)\n\tat com.fasterxml.jackson.databind.deser.std.StdDeserializer.handleUnknownProperty(StdDeserializer.java:1206)\n\tat com.fasterxml.jackson.databind.deser.BeanDeserializerBase.handleUnknownProperty(BeanDeserializerBase.java:1592)\n\tat com.fasterxml.jackson.databind.deser.BeanDeserializerBase.handleUnknownProperties(BeanDeserializerBase.java:1542)\n\tat com.fasterxml.jackson.databind.deser.BeanDeserializer._deserializeUsingPropertyBased(BeanDeserializer.java:438)\n\tat com.fasterxml.jackson.databind.deser.BeanDeserializerBase.deserializeFromObjectUsingNonDefault(BeanDeserializerBase.java:1287)\n\tat com.fasterxml.jackson.databind.deser.BeanDeserializer.deserializeFromObject(BeanDeserializer.java:326)\n\tat com.fasterxml.jackson.databind.deser.BeanDeserializer.deserialize(BeanDeserializer.java:159)\n\tat com.fasterxml.jackson.databind.ObjectMapper._readMapAndClose(ObjectMapper.java:4202)\n\tat com.fasterxml.jackson.databind.ObjectMapper.readValue(ObjectMapper.java:32

```



At first Exceptions may sound like  
a good idea

... but they easily lead to bad decisions

A man with a beard and glasses, wearing a ThoughtWorks lanyard, is smiling and clapping his hands. He is wearing a dark jacket over a t-shirt. The background is a blurred crowd of people at a conference or event. The word "Why" is overlaid in large white text.

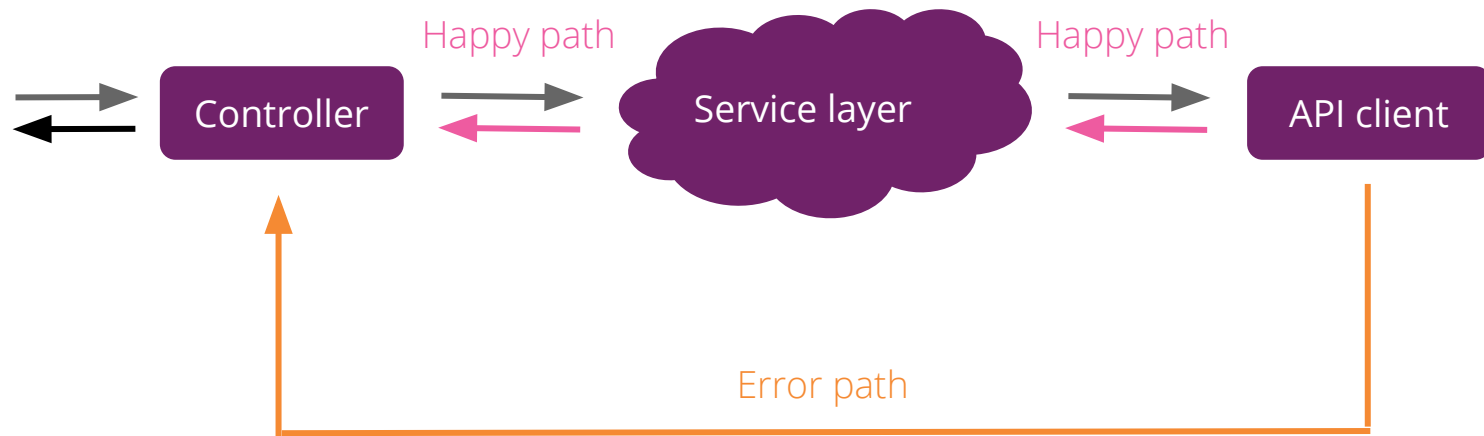
# Why

Easy to **ignore** or **miss**

Exceptions often end up being used  
as **flow control**

A photograph of a man with a beard and glasses, wearing a ThoughtWorks lanyard, smiling and interacting with a woman at a conference. The image is overlaid with a semi-transparent purple filter.

# Breaks encapsulation



```
interface Verifier {  
    /**  
     * @param jwt a jwt token  
     * @return authentication credentials  
     */  
    fun verify(jwt: String): TokenAuthentication  
}
```



```
/**
 * Perform the verification against the given Token
 *
 * @param token to verify.
 * @return a verified and decoded JWT.
 * @throws AlgorithmMismatchException
 * @throws SignatureVerificationException
 * @throws TokenExpiredException
 * @throws InvalidClaimException
 */
public DecodedJWT verifyByCallingExternalApi(String token);
```

A man with a beard and glasses, wearing a ThoughtWorks lanyard, is smiling and interacting with a woman at a conference. The background is a blurred crowd of people, and the entire image has a purple tint.

# A typical way of handling this

```
@ExceptionHandler(JWTVerificationException::class)
fun handleException(exception: JWTVerificationException):
    ResponseEntity<ErrorMessage> {
    return ResponseEntity
        .status(HttpStatus.BAD_GATEWAY)
        .body(ErrorMessage.fromException(exception))
}
```

A background image of a man with a beard and glasses, wearing a ThoughtWorks lanyard, smiling and interacting with a woman at a conference. The image is overlaid with a semi-transparent purple filter.

# What is an Exception?

A cluster going down 🔥  
is an exception

**Not really an exception**

As the context grows it becomes harder  
to **test** and **reason about**

# Either



Two different values depending on the result  
of the computation



The diagram illustrates the structure of the `Either<T>` type. At the top, a blue box labeled `Either<T>` is connected by a vertical dotted line to two separate teal boxes below. The left teal box is labeled `Left` and `Exception`, while the right teal box is labeled `Right` and `T`. A small purple rectangle is positioned above the `Either<T>` box.

`Either<T>`

**Left**  
Exception

**Right**  
T

```
sealed class Either<out L, out R> {  
    data class Left<out L, out R>(val a: L) : Either<L, R>()  
    data class Right<out L, out R>(val b: R) : Either<L, R>()  
}
```



# How to use it?

```
interface Verifier {  
    fun verify(token: String): Either<TokenEx, TokenAuth>  
}
```

```
fun Verifier.unsafeVerify(jwt: String): Either<TokenEx, TokenAuth> = try {  
    verifyByCallingExternalApi(jwt).right()  
} catch (e: JWTVerificationException) {  
    e.left()  
}
```

A background image of a man with a beard and glasses, wearing a ThoughtWorks lanyard, smiling and interacting with a woman at a conference. The image is overlaid with a semi-transparent purple filter.

# An evolving computation

```
interface Operations {  
    fun <T, A, B> Either<T, A>.map(f: (A) -> B): Either<T, B>  
    fun <T, A, B> Either<T, A>.flatMap(f: (A) -> Either<T, B>):  
        Either<T, B>  
}
```



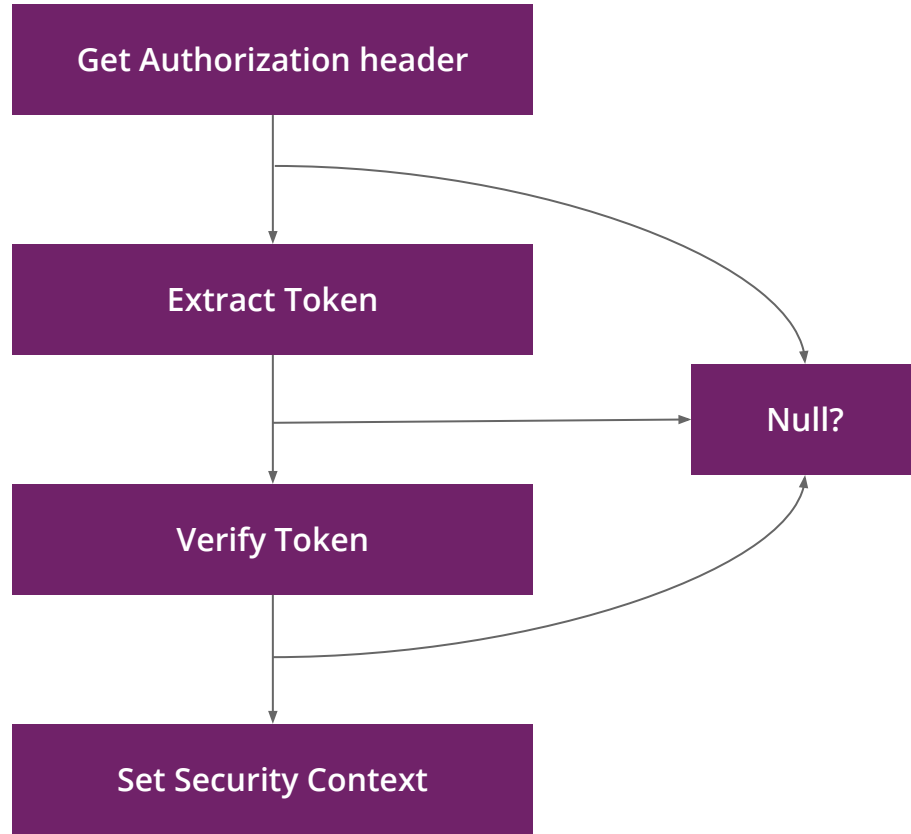
```
verifier
```

```
  .unsafeVerify.jwt)
```

```
  .map { it.asToken() }
```



# What about more complex flows



```
request.getHeader(Headers.AUTHORIZATION)
  .toEither()
  .flatMap { header ->
    header.extractToken()
      .flatMap { jwt ->
        verifier
          .verify(jwt)
            .map { token ->
              SecurityContextHolder.getContext().authentication = token
            }
        }
      }
  }
}
```



# Non-Nested Syntax

Similar to **async/await**



```
Either.fx {  
    val (header) = request.getHeader(Headers.AUTHORIZATION).toEither()  
    val (jwt) = header.extractToken()  
    val (token) = verifier.verify(jwt)  
    SecurityContextHolder.getContext().authentication = token  
}
```

[thoughtworks.com/insights/blog/either-data-type-alternative-throwing-exceptions](https://thoughtworks.com/insights/blog/either-data-type-alternative-throwing-exceptions)



# Summary

Null and exceptions can lead to flaky,  
hard to understand code

Let's stop **using null**

Let's stop **abusing** exceptions

# Continue the conversation on Slack

***XConfEurope2020***

[xconfeurope2020.slack.com](https://xconfeurope2020.slack.com)

[#talk4-alternative-to-null-and-exceptions](#)

***#XConfOnline***