

# Lean backends using functional Kotlin

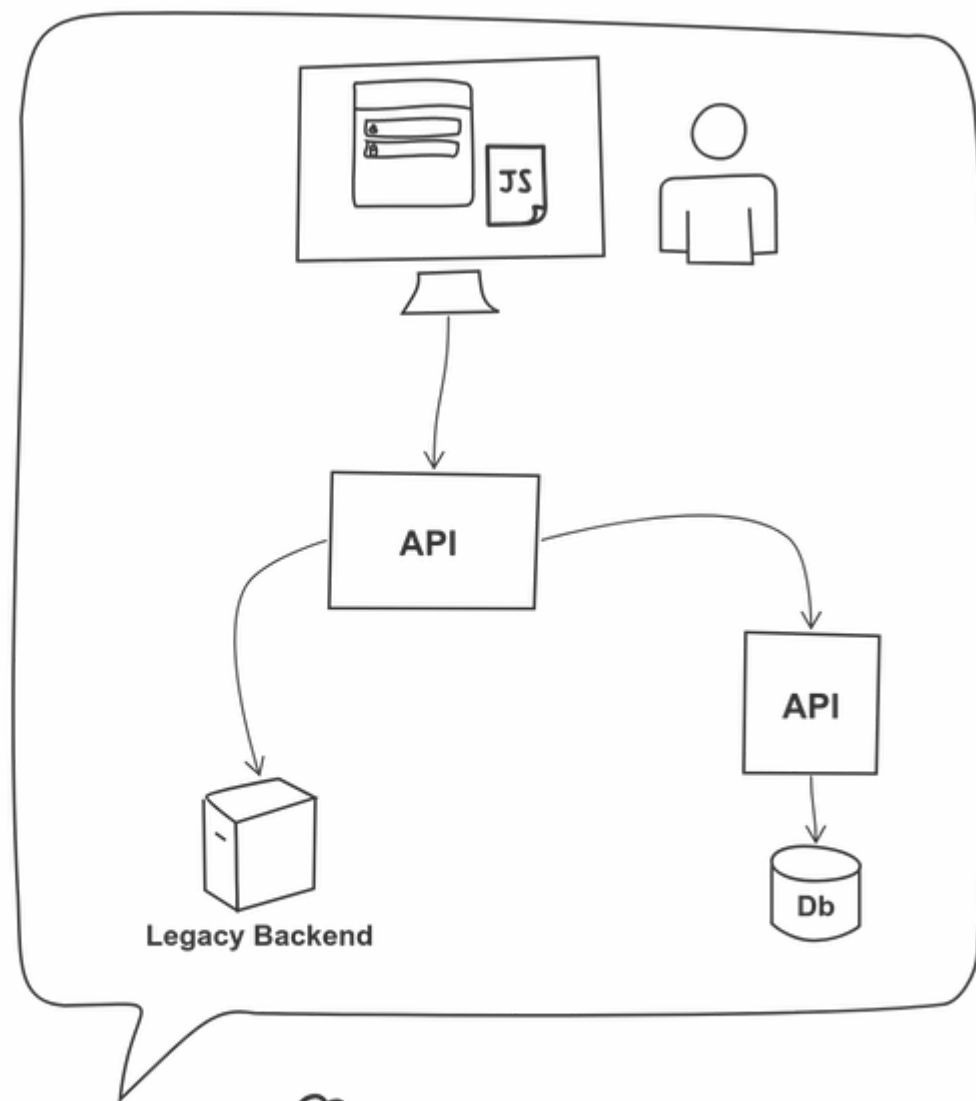
**What to expect from this talk**

**Mario Fernandez**

**Srishti Gaihre**

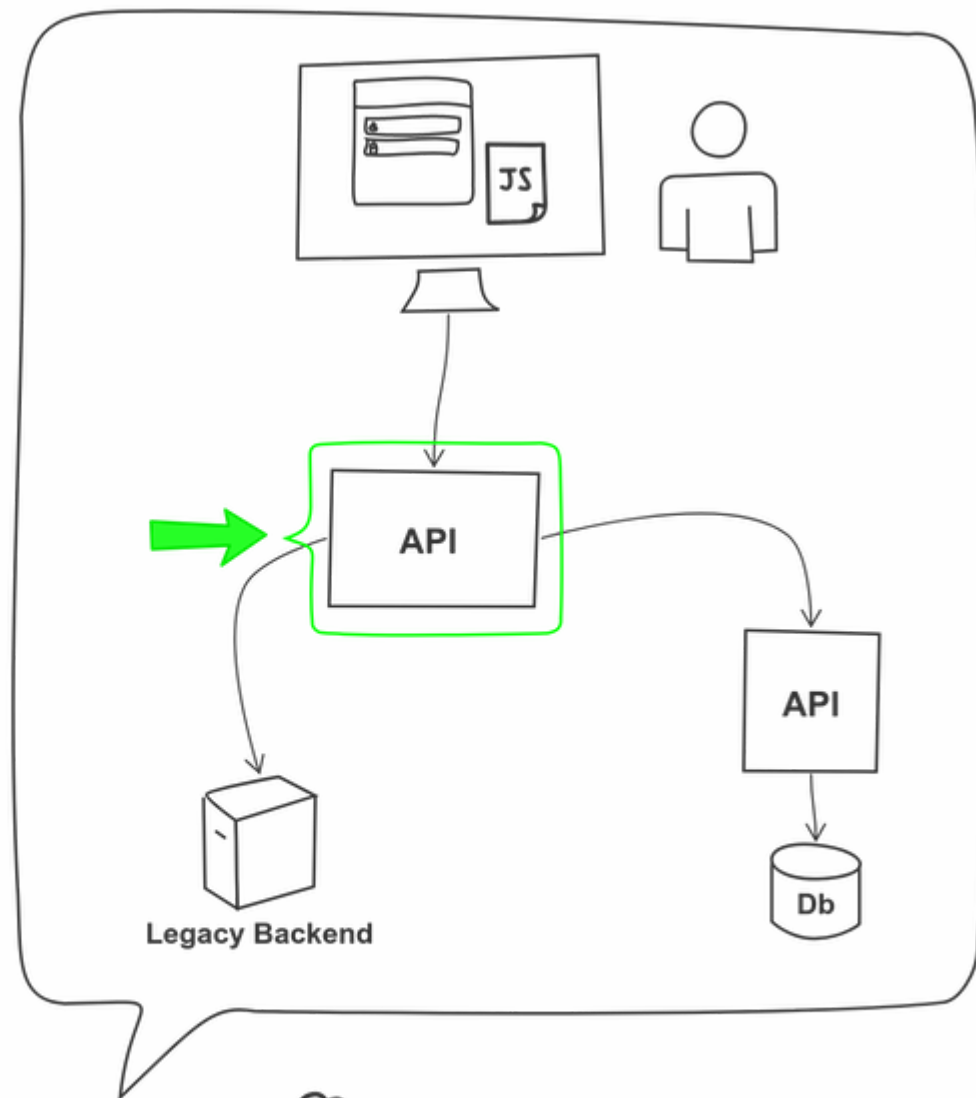
**ThoughtWorks**

**Let's start with some context**



Big German Corp





Big German Corp



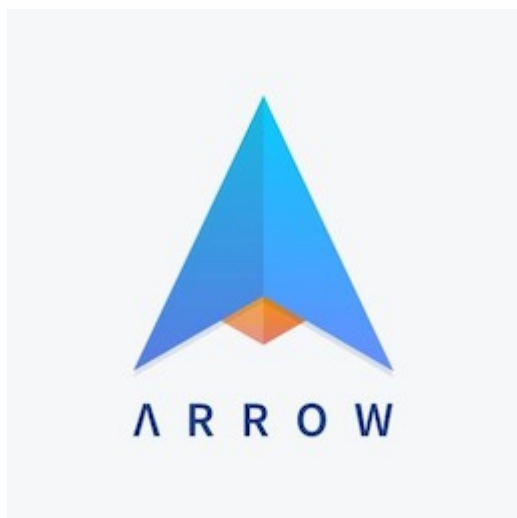


**Technologies that we will be mentioning**





**STRIKT**



**Spring  
Boot**

# Our pain points

**I don't know the state of my data**

**I don't know the state of my data**

**Half of my time I'm just dealing with *null* values**

**I don't know the state of my data**

**Half of my time I'm just dealing with *null* values**

**The other half I'm debugging the 500s thrown by our application**

## Uncontrolled Data



## Dealing with Null



## Unhandled Exceptions





**Wait, when did this actually change?**

# Immutability



```
data class TokenAuthentication(  
    val id: Id,  
    val firstName: FirstName,  
    val lastName: LastName,  
    val scopes: List<Scope>,  
    val expiresAt: LocalDateTime  
)
```

```
val scopes: List<Scope> = buildScopes(token)
```

```
// List is immutable, it won't compile  
scopes.removeAt(1) ❌
```

```
// Creates a new list  
scopes.filter { it.isAdmin } ✅
```

**JSON**

```
@JsonIgnoreProperties(ignoreUnknown = true)
data class TokenAuthentication(
    val id: Id,
    @JsonAlias("name_first")
    val firstName: FirstName,
    @JsonAlias("name_last")
    val lastName: LastName,
    @JsonDeserialize(converter = ListSanitizer::class)
    val scopes: List<Scope>,
    val expiresAt: LocalDateTime
)
```

# **Dealing with change**

```
// Will create a new object
```

```
fun TokenAuthentication.clearScopes() = copy(scopes = emptyList())
```

# **A simple test**

```
expectThat(token) {  
    get { name }.isEqualTo("google-oauth2|3234123")  
    get { authorities.map { it.authority } }.contains("create:recipes")  
}
```

org.opentest4j.AssertionFailedError:

▼ Expect that Some(TokenAuthentication@52789c41):

▼ TokenAuthentication@52789c41:

Authenticated: true;

Authorities: profile, create:recipes:

▼ name:

x is equal to "google-oauth2|3234123" : found "google-oauth2|dude"



# Why Immutability?

**Easier to reason**

**Always in a valid state**

**Can be shared freely**

**Uncontrolled  
Data**



**Dealing  
with  
Null**



**Unhandled  
Exceptions**



**Immutability**

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

# The billion dollar mistake

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

# Nullable types



Authorization: Bearer bGciOi...JIUzI1NiIs

Authorization: Bearer bGciOi...JIUzI1NiIs

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

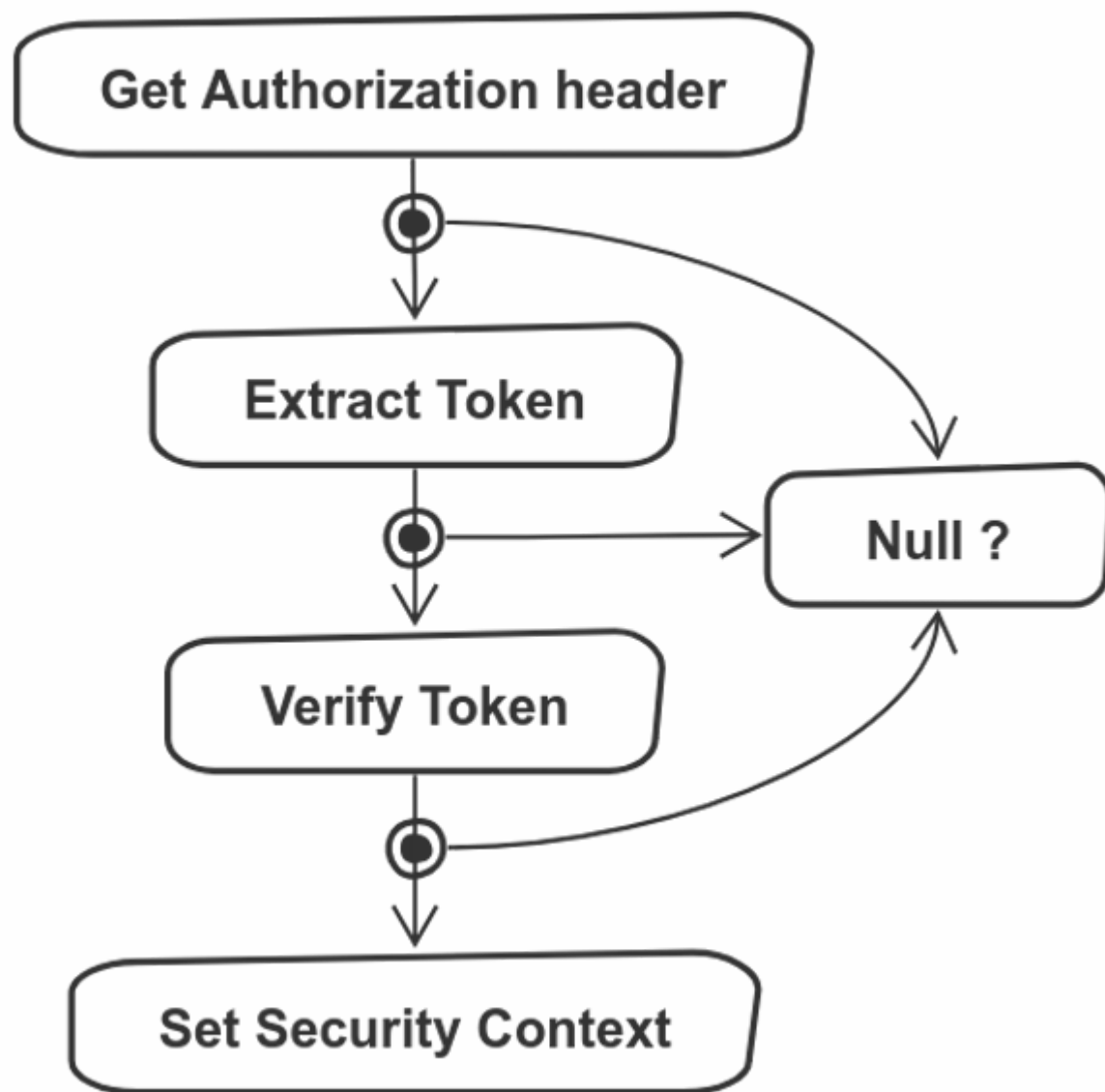
Authorization: Bearer bGci0i...JIUzI1NiIs

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

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



**It can get messy**



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

A small, pixelated image of the character Ryu from the Street Fighter II video game. He is shown in a dynamic pose, performing a Hadouken (fireball) attack. He is wearing his signature white gi with a black belt and red headband. A bright blue and white energy ball is visible coming from his hands.

```
1 function hell(win) {
2   // for listener purpose
3   return function() {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                      async.eachSeries(SCRIPTS, function(src, callback) {
14                        loadScript(win, BASE_URL+src, callback);
15                      });
16                    });
17                  });
18                });
19              });
20            });
21          });
22        });
23      });
24    });
25  };
26 }
```

# Option Datatype



# Datatype? 🤔

A digression about Functional Programming

# Datatype? 🤔

**A datatype is an abstraction that encapsulates one reusable coding pattern.**

A digression about Functional Programming



A diagram illustrating the structure of an `Option<T>` container. The container is represented by a large rounded rectangle. Inside, at the top, is a smaller rectangle labeled `Option<T>`. Below this, the container is split into two vertical sections. The left section contains a rounded rectangle labeled `Some<T>` and an oval labeled `T` below it. The right section contains a rounded rectangle labeled `None`.

`Option<T>`

`Some<T>`

`T`

`None`



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

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

```
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>  
}
```

A digression about Functional Programming

**[hackernoon.com/kotlin-functors-applicatives-and-monads-in-pictures-part-1-3-c47a1b1ce251](https://hackernoon.com/kotlin-functors-applicatives-and-monads-in-pictures-part-1-3-c47a1b1ce251)**

A digression about Functional Programming

**Let's try our previous example with *Option***

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

**Not much of an improvement 🙄**

**Non-nested syntax thanks to arrow**

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



@Test

```
fun `verify does not work with a invalid jwt token`() {  
    expectThat(  
        RemoteVerifier(keySet).verify(jwt)  
    ).isEmpty()  
}
```

# Why Option?

**Explicit about what can be null**

**Avoid if-null litter**

**Compile time checks**

**Save a billion dollars 🧐**

**Uncontrolled  
Data**



**Dealing  
with  
Null**



**Unhandled  
Exceptions**



**Immutability**



**Option**

com.auth0.jwt.exceptions.JWTDecodeException:

The string '{"typ":"JWT","alg":"RS256"}' is not a valid token.

at com.auth0.jwt.impl.JWTParser.convertFromJSON(JWTParser.java:52)

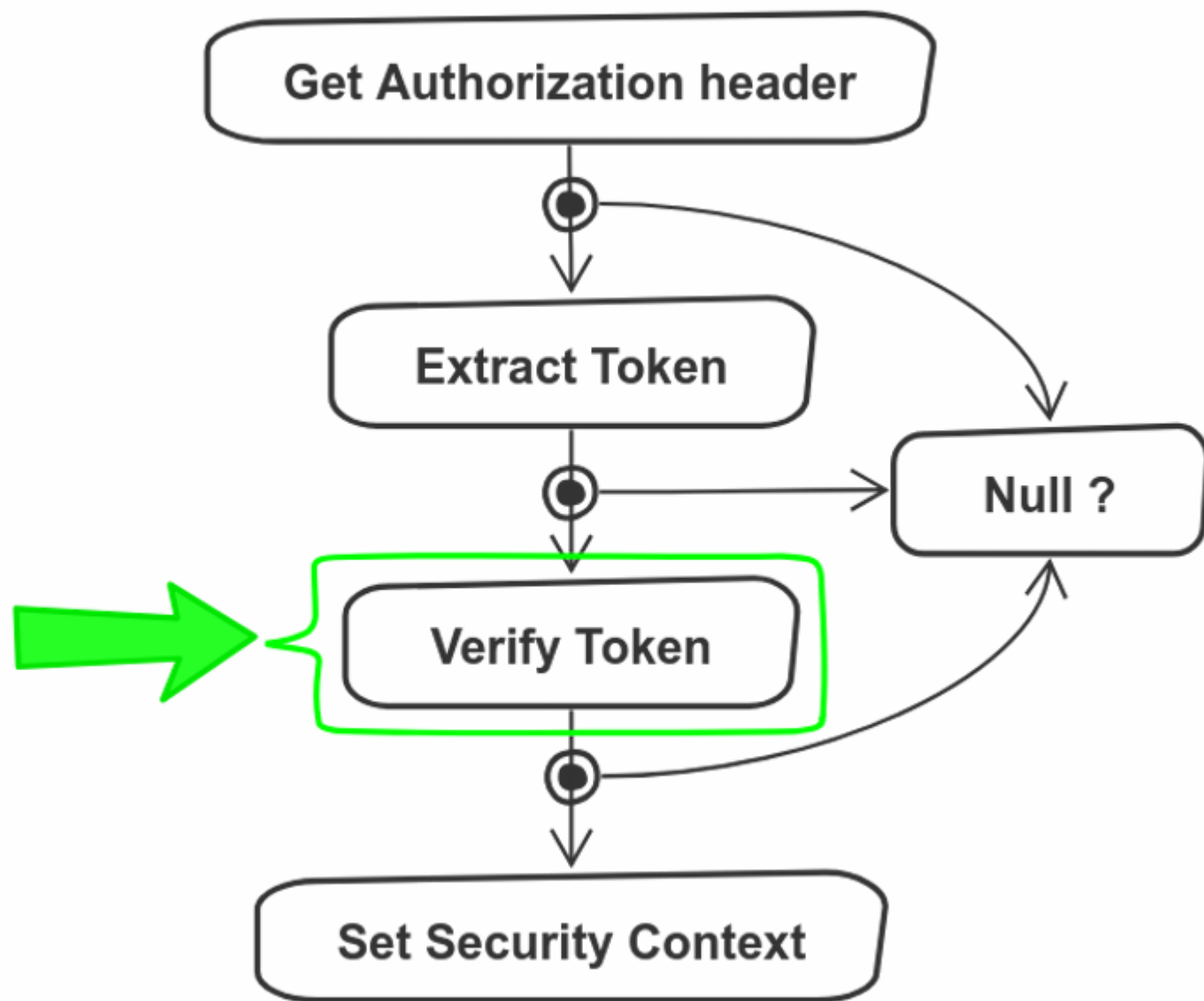
at com.auth0.jwt.impl.JWTParser.parseHeader(JWTParser.java:33)

at com.auth0.jwt.JWTDecoder.<init>(JWTDecoder.java:37)

at com.auth0.jwt.JWT.decode(JWT.java:21)

at com.auth0.jwt.JWTVerifier.verify(JWTVerifier.java:352)

**Verifying our token**

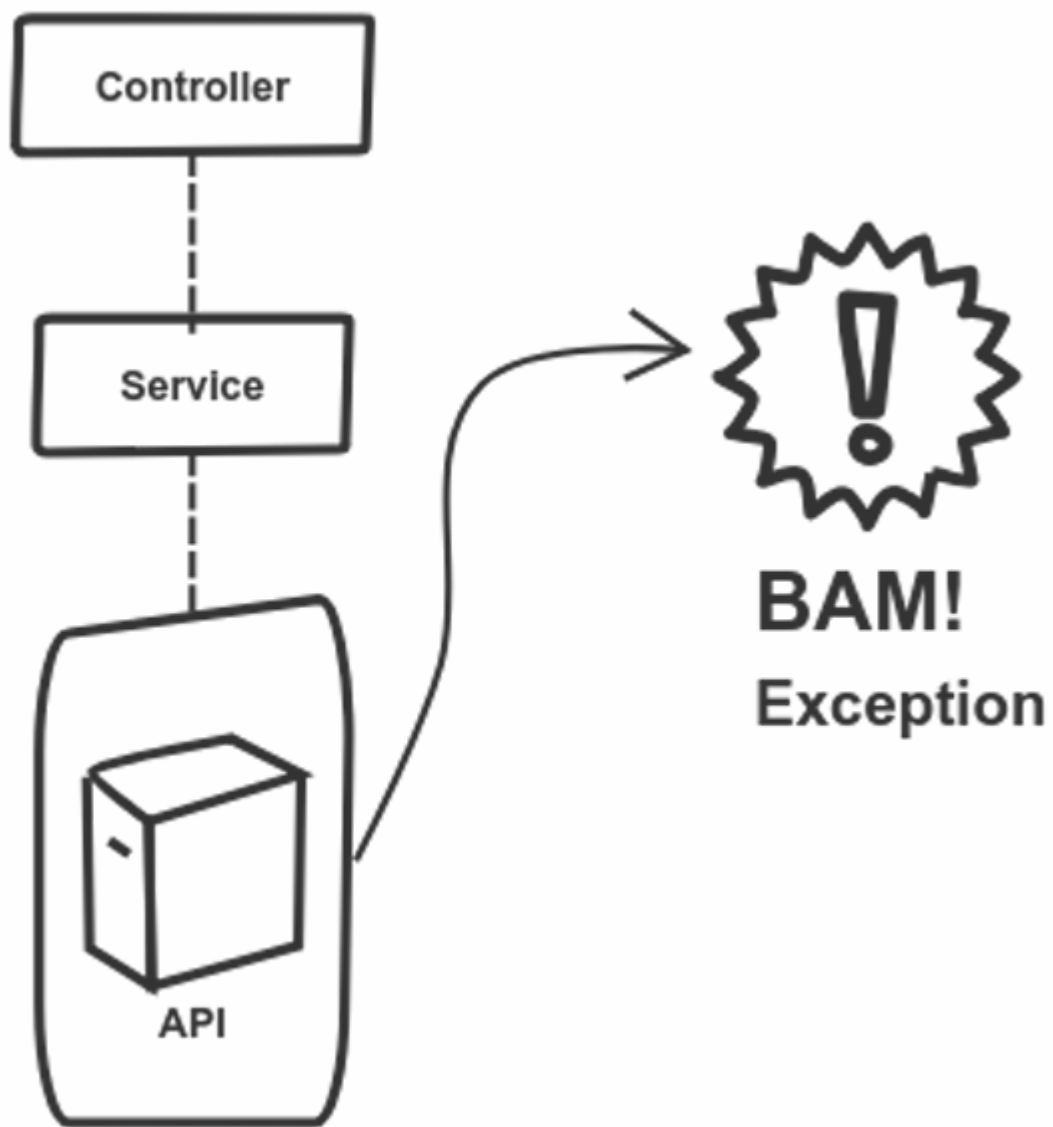


```
interface Verifier {  
    fun verify(token: String): TokenAuthentication  
}
```

**That signature is not quite telling the truth**



```
/**
 * 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);
```



**Exceptions make the flow implicit**

**Exceptions force you to be aware of the internal  
implementation**

[kotlinlang.org/docs/reference/exceptions.html](https://kotlinlang.org/docs/reference/exceptions.html)

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

# Either DataType



# **Option and Either are quite similar**

A digression about Functional Programming

**Either<E, T>**

Right<T>

T

Left<E>

E





```
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>  
}
```

A digression about Functional Programming

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

**Isolating the problematic code**

```
private fun JWTVerifier.unsafeVerify(token: String) = try {  
    verifyByCallingExternalApi(token).right()  
} catch (e: JWTVerificationException) {  
    e.left()  
}
```

# **Operating with Either**

```
override fun verify(token: String)
    : Either<JWTVerificationException, TokenAuthentication> {
    val key = key(keySet)
    val algorithm = algorithm(key)
    val verifier = verifier(algorithm, leeway)
    return verifier
        .unsafeVerify(token)
        .map { it.asToken() }
}
```

```
Either.fx<JWTVerificationException, TokenAuthentication> {  
    // Either<Throwable, ResponseEntity<UnprocessedResponse>>  
    val response = unsafeRequest()  
    val (body) = response  
        .mapLeft { JWTVerificationException(it) }  
    body.map()  
}
```

```
@GetMapping("{id}")
```

```
fun recipe(@PathVariable id: Int): ResponseEntity<RecipeDetails> {  
    return when (val result = repository.find(id)) {  
        is Either.Left -> ResponseEntity.status(result.a).build()  
        is Either.Right -> ResponseEntity.ok(result.b)  
    }  
}
```



@Test

```
fun `verify works if the expiration is not taken into account`() {  
    val hundredYears = 3600L * 24 * 365 * 100  
    val verifier = RemoteVerifier(keySet, hundredYears)  
  
    expectThat(verifier.verify jwt)).isRight().and {  
        get { name }  
            .isEqualTo("google-oauth2|111460419457288935787")  
        get { authorities.map { it.authority } }  
            .contains("create:recipes")  
    }  
}
```

# Why Either?

**Makes flow explicit**

**Interface tells the whole truth**

**Compile time checks**

**Uncontrolled  
Data**



**Dealing  
with  
Null**



**Unhandled  
Exceptions**



**Immutability**



**Option**



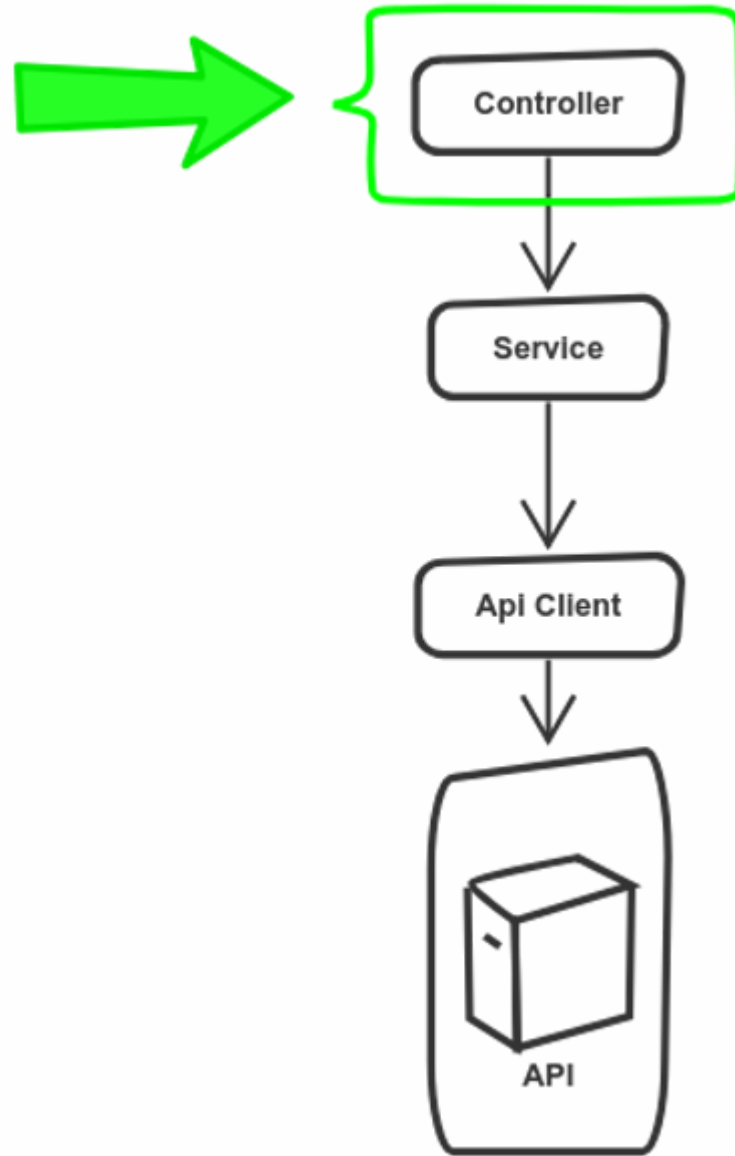
**Either**

**What comes next?**

**Purely functional code**

A high-angle, wide-view photograph of Earth from space. The image captures the curvature of the planet, with a thin blue line of the atmosphere separating the dark, featureless blackness of space from the bright, cloud-covered surface of the Earth. The clouds are predominantly white and wispy, scattered across the visible land and ocean. The perspective is from a high altitude, looking down at the planet's surface.

# Edge of the World



**10**



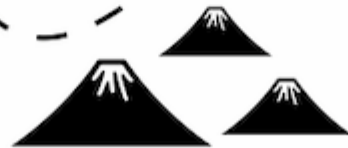
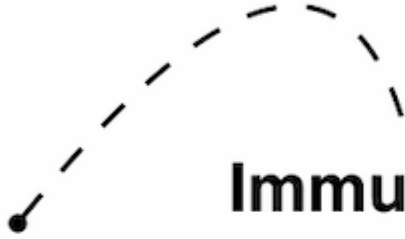
```
I0<Either<JWTVerificationException, TokenAuthentication>>
```

**We are hitting the limit of what's convenient to do  
with Kotlin and Arrow here**

**Wrap Up**



**Immutability**



**Option**



**Either**





**Backend**

***At your own pace!***

# JOIN OUR COMMUNITY

*26 years experience*

*42 offices in 13 countries*

*Thought leader in agile software development and continuous delivery*

*6000+ thoughtworkers worldwide*

*300+ thoughtworkers in Germany*

[de-recruiting@thoughtworks.com](mailto:de-recruiting@thoughtworks.com)

