

GOTO **AMSTERDAM 2024**

#GOTOams

goto;

Using Ktor 3.0

With all the shiny things

@garthgilmour
garth.gilmour@jetbrains.com

I'm Garth from JetBrains



November 23, 2023
16:00 UTC

Build Apps for iOS and Android with Shared Logic and Native UIs

Pamela Hill
Garth Gilmour






The screenshot shows a YouTube video player. The video title is "The Philosophy Of Domain Driven Design by Garth Gilmour - Belfast DDD Meetup - Feb 2019". The channel is "Intel Software". The video shows a man with glasses and a beard, wearing a dark t-shirt with a "BETEL" logo, speaking into a microphone. He is standing in front of a brick wall. The video player interface includes a play button, a progress bar, and various control icons at the bottom.

Where?
Anywhere there's a JVM?
As400 and Commodore64?

Compose Multiplatform

What?

Why?

[illegible]



Garth Gilmour
Head of Learning at Instill

Generics on the JVM: What you don't know will hurt you

@GarthGilmour

[illegible]

The screenshot shows a video player interface. The video content is a presentation slide for 'EstimateMate'. The slide features a large central display showing the number '5'. To the left of this display is a numeric keypad with buttons for digits 0-9, a decimal point, and a fraction template. Above the keypad are several tabs labeled 'Home', 'New', 'Template', 'Question Bank', 'Share', and 'Result'. Below the keypad are buttons for 'Previous', 'Next', and 'Done'. The top of the slide has the 'EstimateMate' logo and the text 'EstimateMate - Estimate Management Software'. The video player controls at the bottom show a progress bar at 10:52 / 14:20 and various playback icons.

10 Big Ideas

Lessons from Industry
for Academics

8:21 AM - 7th February 2022

[illegible]

Amey Crockett & Garth Gilmour @ Kotlin Dev Day Amsterdam 2022

1.1K subscribers

Subscribe

0:00 / 1:00

Next On Track

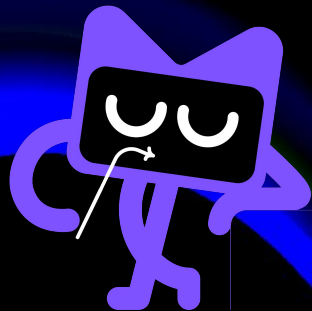
Amey Crockett & Garth Gilmour

INSTIL

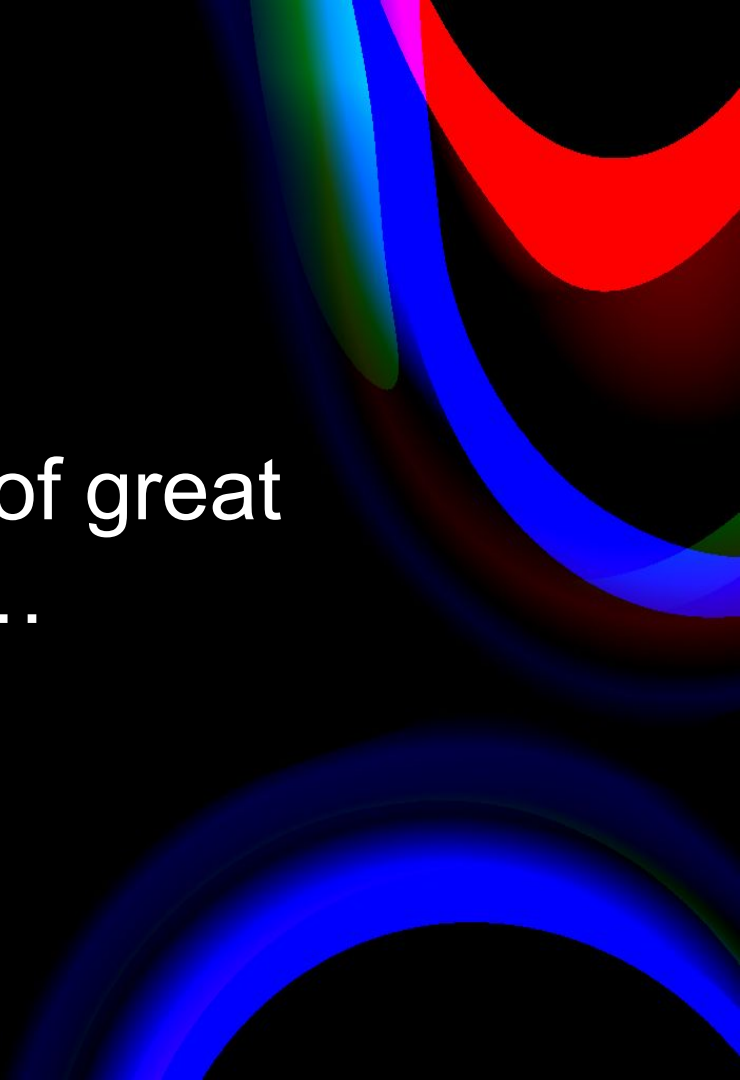
Using Kotlin to Create Kotlin to Teach Kotlin within Space



Ktor is a really awesome
framework for asynchronous
client and server applications



But there are a lot of great
frameworks...





http4k

http4s



spring®



helidon.io

VERT.X



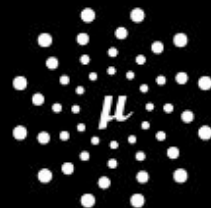
play



QUARKUS



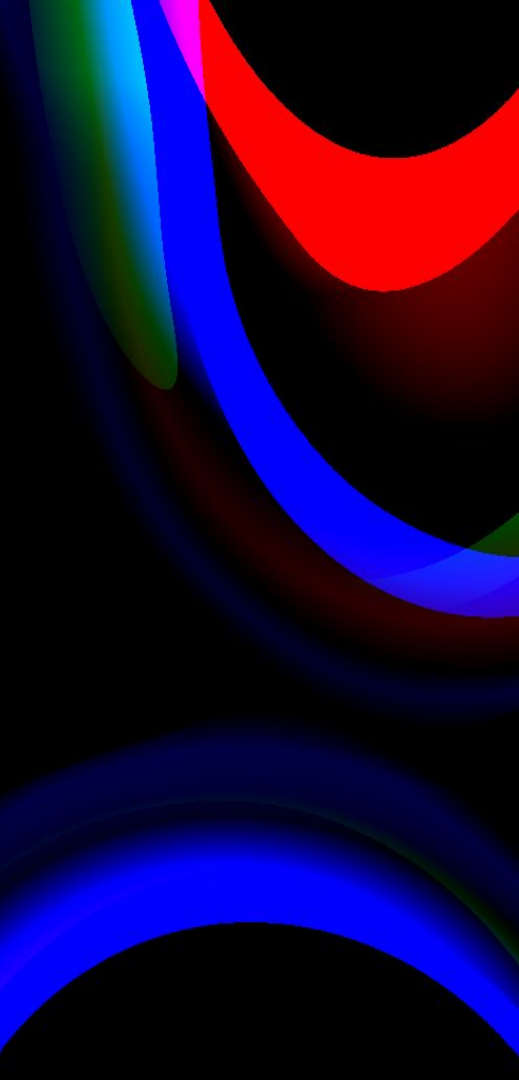
akka



MICRONAUT®



What makes Ktor different?

- 
- An abstract graphic in the top right corner consisting of several overlapping, curved, semi-transparent shapes in vibrant colors: red, blue, green, and yellow. The shapes are layered, creating a sense of depth and movement.
1. Our core values
 2. The shiny things

Our core values

Keep Ktor:

- Minimalist
- Flexible
- Simple
- Fast

Try to avoid

- Boilerplate and duplication
- Magic (e.g. Annotations)



Our core values

Ktors values are the antithesis of
Convention Over Configuration

- It never does more than you ask
- Placing a JAR on the classpath does not alter how the framework behaves

This doesn't mean COC is wrong

- But Ktor folks want to ensure they know everything that is going on
- If you don't install it then it's not there



Metaphor stolen from Christin Gorman
(see <https://vimeo.com/28885655>)

Shiny things

Let's take a case study and add:

- Testing
- DI / Koin
- GraphQL
- WebSockets
- Native execution
- OpenTelemetry
- Fine grained control



The sample projects

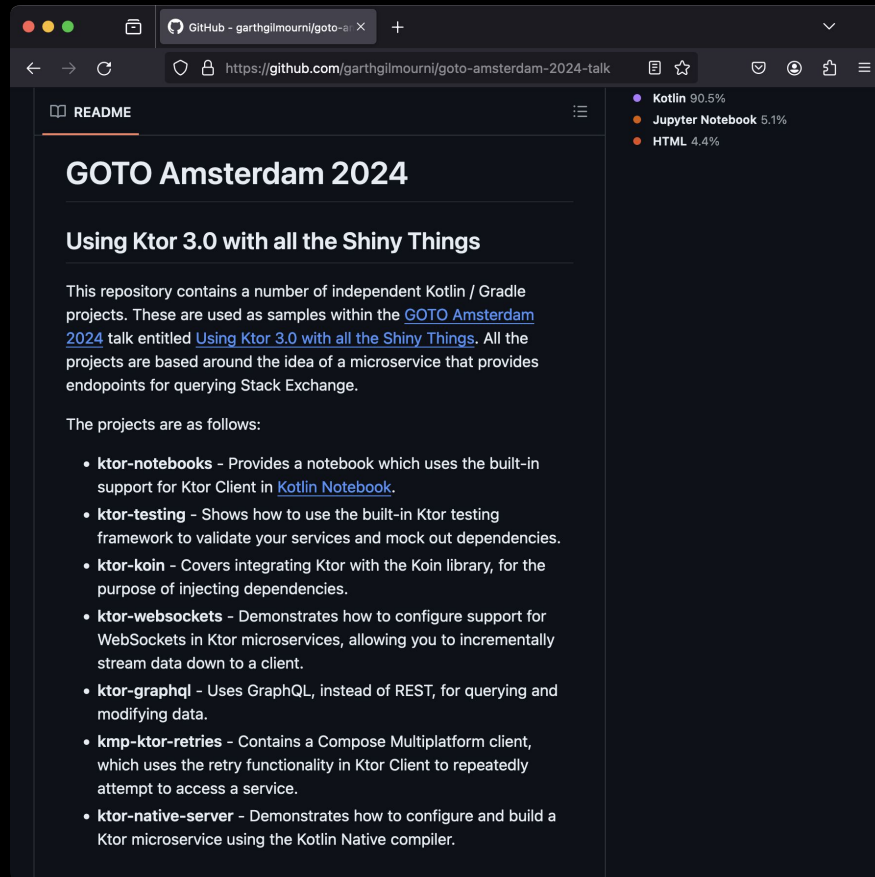
This is a lot of shiny!!!

- It kind of grew when I wasn't looking

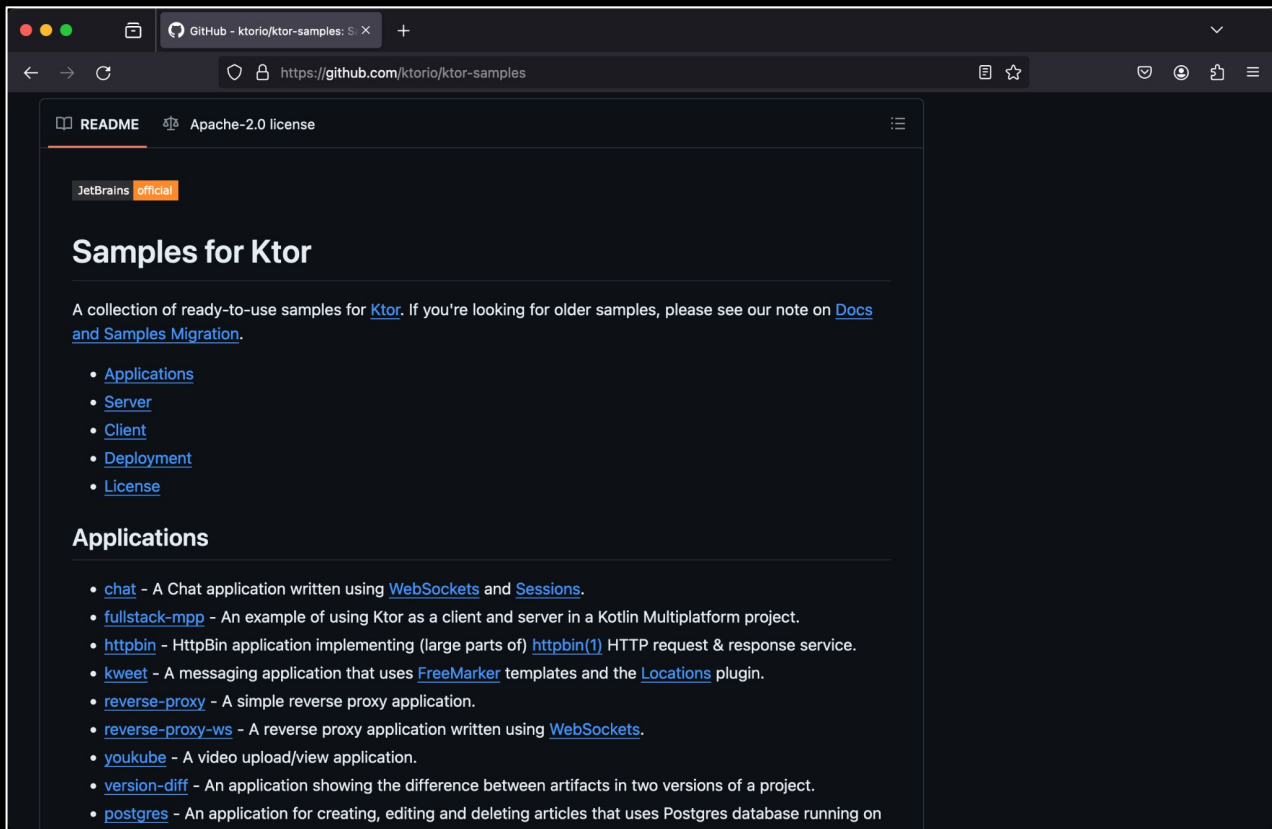
There are sample projects for everything

- Details of the repo on the next slide
- Please review in your copious free time
- The official samples give extra depth



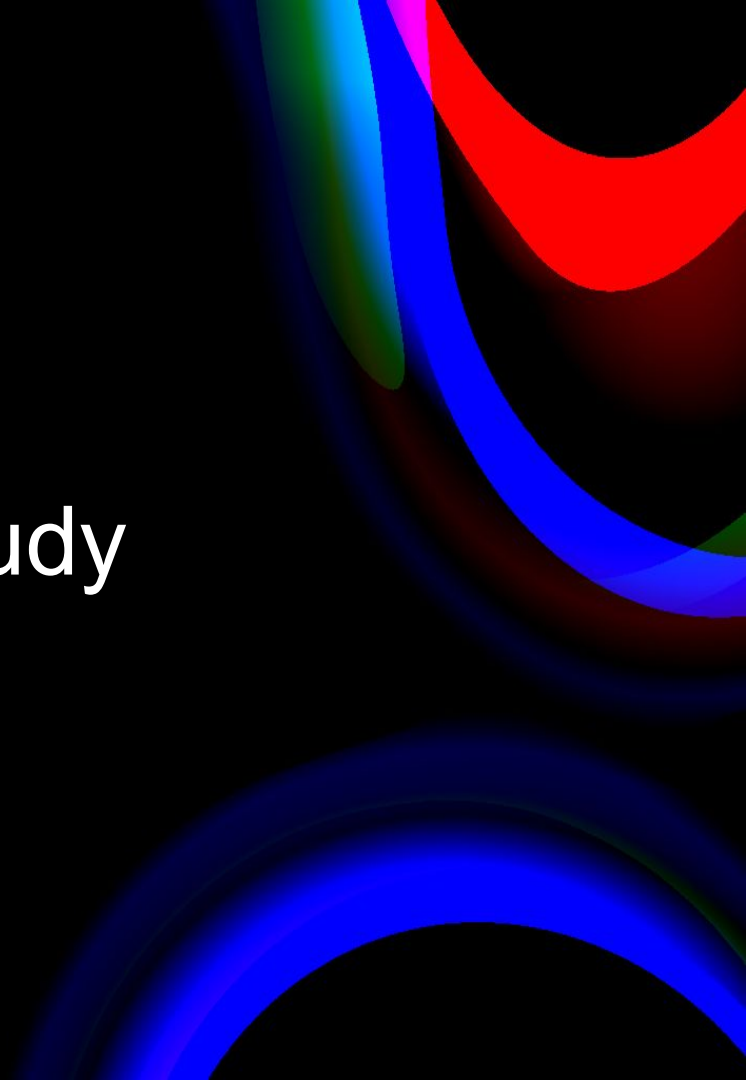


<https://github.com/garthgilmourni/goto-amsterdam-2024-talk>



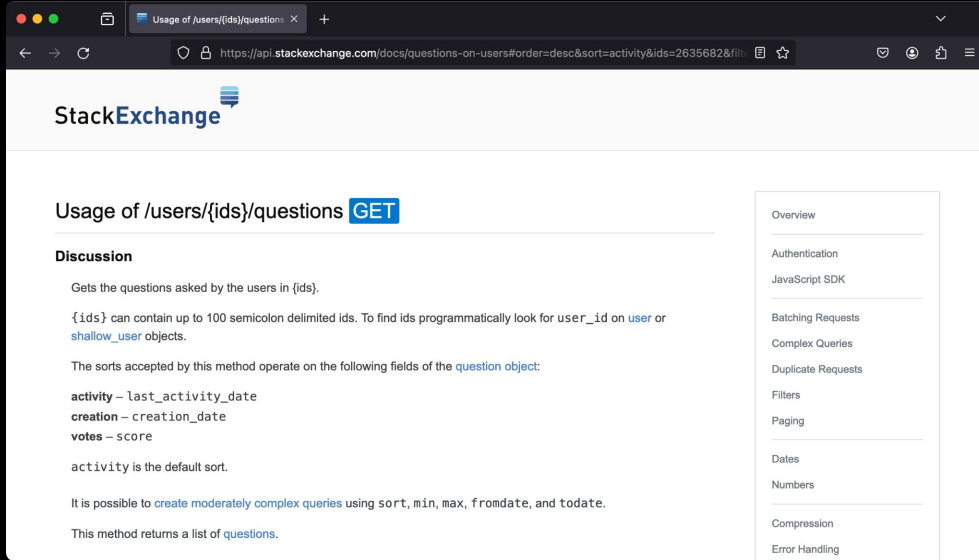
<https://github.com/ktorio/ktor-samples>

Our Case Study





Let's talk to Stack Overflow



StackExchange

Usage of /users/{ids}/questions GET

Discussion

Gets the questions asked by the users in {ids}.

{ids} can contain up to 100 semicolon delimited ids. To find ids programmatically look for user_id on [user](#) or [shallow_user](#) objects.

The sorts accepted by this method operate on the following fields of the [question object](#):

- activity – last_activity_date
- creation – creation_date
- votes – score

activity is the default sort.

It is possible to [create moderately complex queries](#) using sort, min, max, fromdate, and todater.

This method returns a list of [questions](#).

- Overview
- Authentication
- JavaScript SDK
- Batching Requests
- Complex Queries
- Duplicate Requests
- Filters
- Paging
- Dates
- Numbers
- Compression
- Error Handling

https://api.stackexchange.com/2.3/users/{user_id}/questions?site={site_name}

<https://api.stackexchange.com/2.3/users/2635682/questions?site=stackoverflow>



```
▼ {  
  ▼ "items": [  
    ▶ { ... }, // 13 items  
    ▶ { ... }, // 14 items  
    ▶ { ... }, // 13 items  
    ▶ { ... }, // 13 items  
    ▶ { ... }, // 14 items  
    ▶ { ... }, // 13 items  
    ▶ { ... }, // 14 items  
    ▶ { ... }, // 14 items  
    ▶ { ... } // 12 items  
  ],  
  "has_more": false,  
  "quota_max": 300,  
  "quota_remaining": 208  
}
```

```
▼ {
  ▼ "items": [
    ▼ {
      ▼ "tags": [
        "typescript",
        "lambda"
      ],
      ▶ "owner": { ... }, // 8 items
      "is_answered": true,
      "view_count": 31042,
      "answer_count": 3,
      "score": 13,
      "last_activity_date": 1628001912,
      "creation_date": 1502622139,
      "last_edit_date": 1502884285,
      "question_id": 45659643,
      "content_license": "CC BY-SA 3.0",
      "link": "https://stackoverflow.com/questions/45659643/signature-of-lambda-return-types-in-typescript",
      "title": "Signature of lambda return types in TypeScript"
    },
  ],
}
```

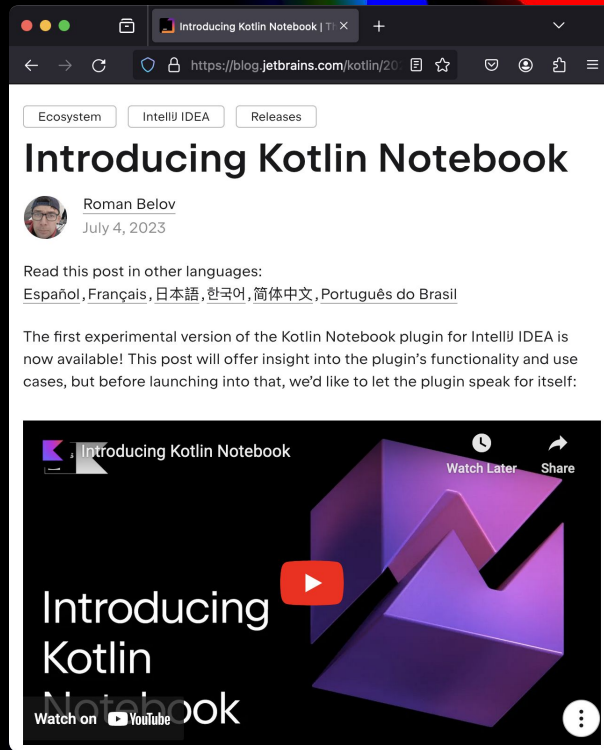
Prototyping with Kotlin Notebook

We could prototype our code in a notebook

- Kotlin Notebook was introduced at KC2023

Notebooks offer three Ktor specific benefits:

- `%use ktor-client`
- `NotebookHttpClient`
- `deserializeJson`



notebook-ktor-integration main

Current File

notebook-ktor-integration UsingKtorClient.ipynb <Injected JKTMeta file>

UsingKtorClient.ipynb

Managed: Kotlin Trusted

%use ktor-client

Executed at 2024.05.20 15:15:44 in 107ms

[26] import io.ktor.client.*

fun buildClient(rootURL: String) = http.config { this: HttpClientConfig<*> {

defaultRequest { this: DefaultRequest.DefaultRequestBuilder

url(rootURL)

}

}

Executed at 2024.05.20 15:15:44 in 65ms

[27] val rootURL = "https://api.stackexchange.com/2.3/users/"

val userID = "2635682"

val client = buildClient(rootURL)

val data = client

.get("\$userID/questions?site=stackoverflow")

.deserializeJson()

Executed at 2024.05.20 15:15:45 in 1s 633ms

[28] val titles = data.items.map { it.title }

titles.forEach(::println)

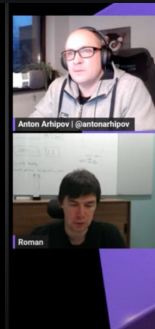
Executed at 2024.05.20 15:15:45 in 137ms

2:17 @ LF UTF-8 4 spaces main

Kotlin Notebook: Visual, Interactive, Fun | Kotlin Notebook Tutorial

Kotlin by JetBrains 68.6K subscribers

742 1 Share Download Clip Save



Data Analytics With Kotlin Notebooks

DAWebinar main Current File

year	month	location	date	temp	maxTemp	minTemp
2000	JANUARY	Amsterdam	2000-01-01T01:00	4.8	2.3	4.3
2000	JANUARY	Amsterdam	2000-01-01T02:00	5	1.9	4.3
2000	JANUARY	Amsterdam	2000-01-01T03:00	5.1	2.1	4.6
2000	JANUARY	Amsterdam	2000-01-01T04:00	5.5	2.5	4.8

```
[44] withYearAndMonth
      .groupBy { year }
      .aggregate { this: AggregateGroupedDStreamType3> H: AggregateGroupedDStreamType3>
        mean { temp } into "meanTemp"
        max { temp } into "maxTemp" } aggregate
      }
```

Executed at 2024-02-27 17:01:37 in 120ms

year	meanTemp	maxTemp	minTemp
2000	10.961	32.6	10.961
2001	10.588	31.3	10.588
2002	10.862	31.8	10.862
2003	10.535	32.8	10.535
2004	10.576	31.8	10.576

1:02:03 / 1:52:52 • Hierarchical Dataframe >

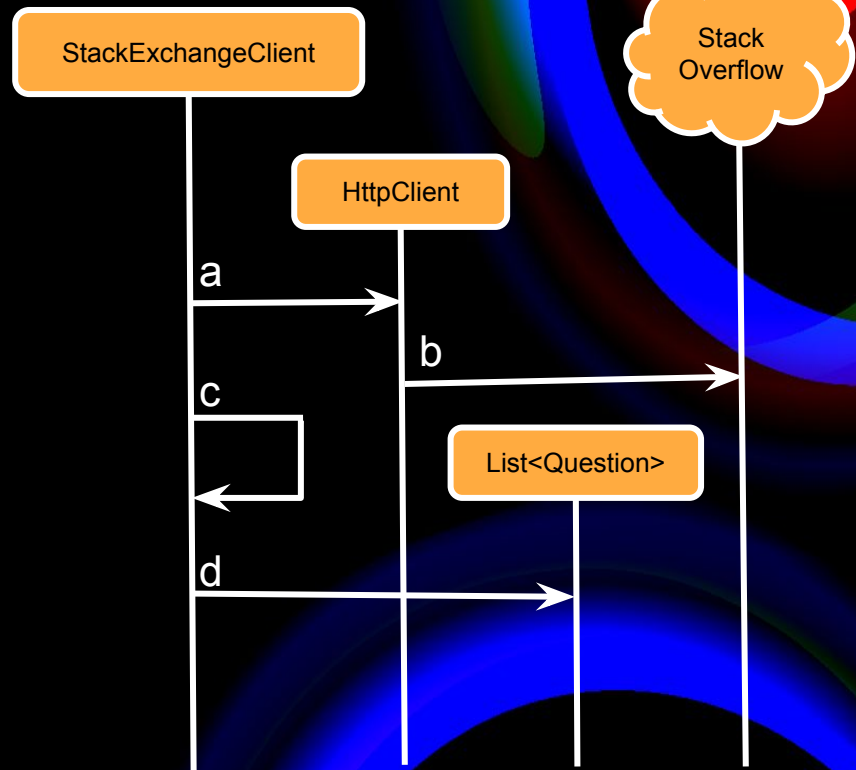
Data Analytics With Kotlin Notebooks

Kotlin by JetBrains 68.6K subscribers

188 1 Share Download Clip Save

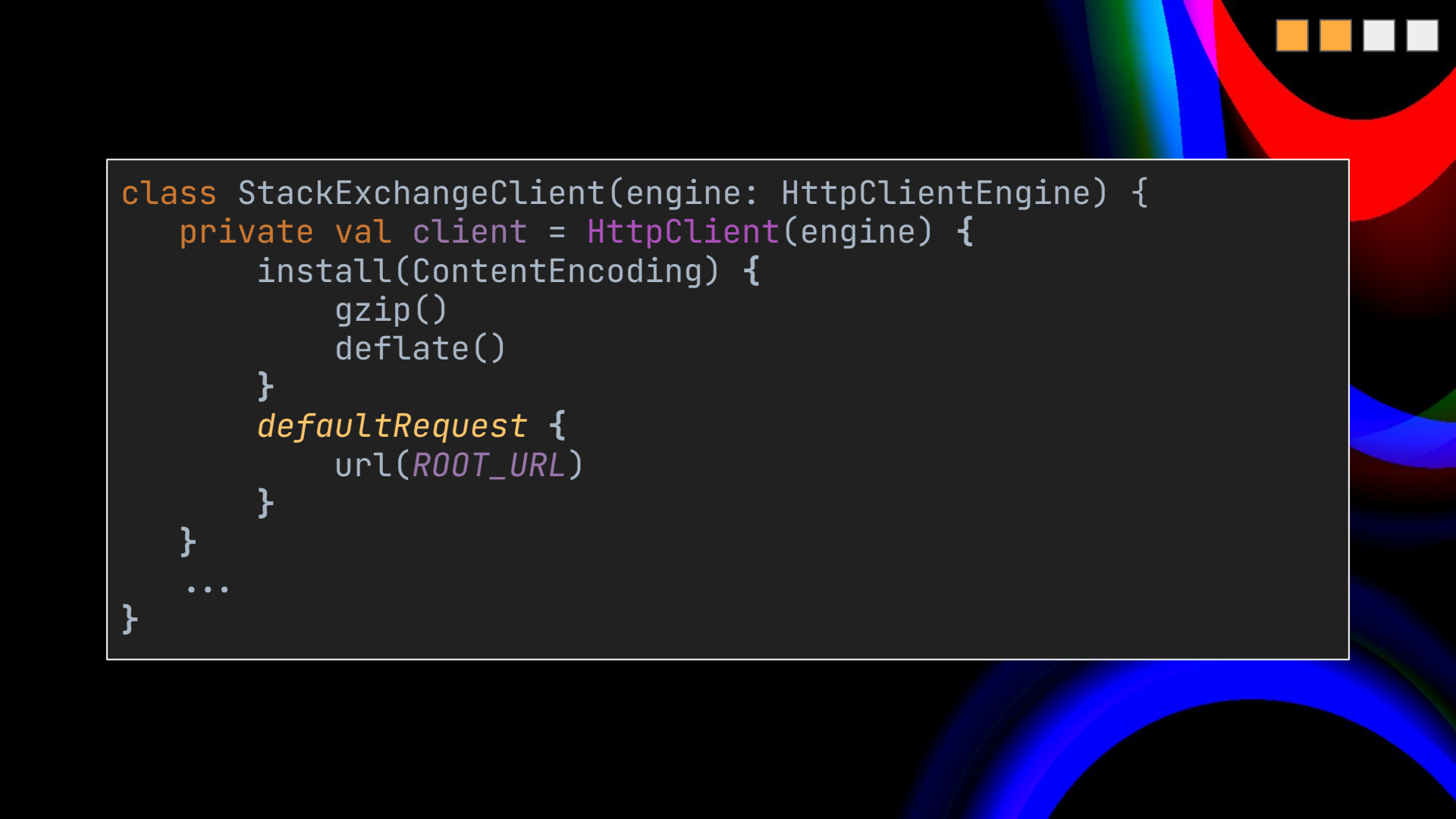
How do we build the service?

1. Create a Question type
2. Create a StackExchangeClient
 - a. Wrap the Ktor HttpClient
 - b. Fetch the data from Stack Overflow
 - c. Magpie out the ids and titles
 - d. Convert to a list of Question
3. Use this client in a route
4. Embed the route in a server





```
@Serializable  
class Question(var id: Long, var title: String)
```



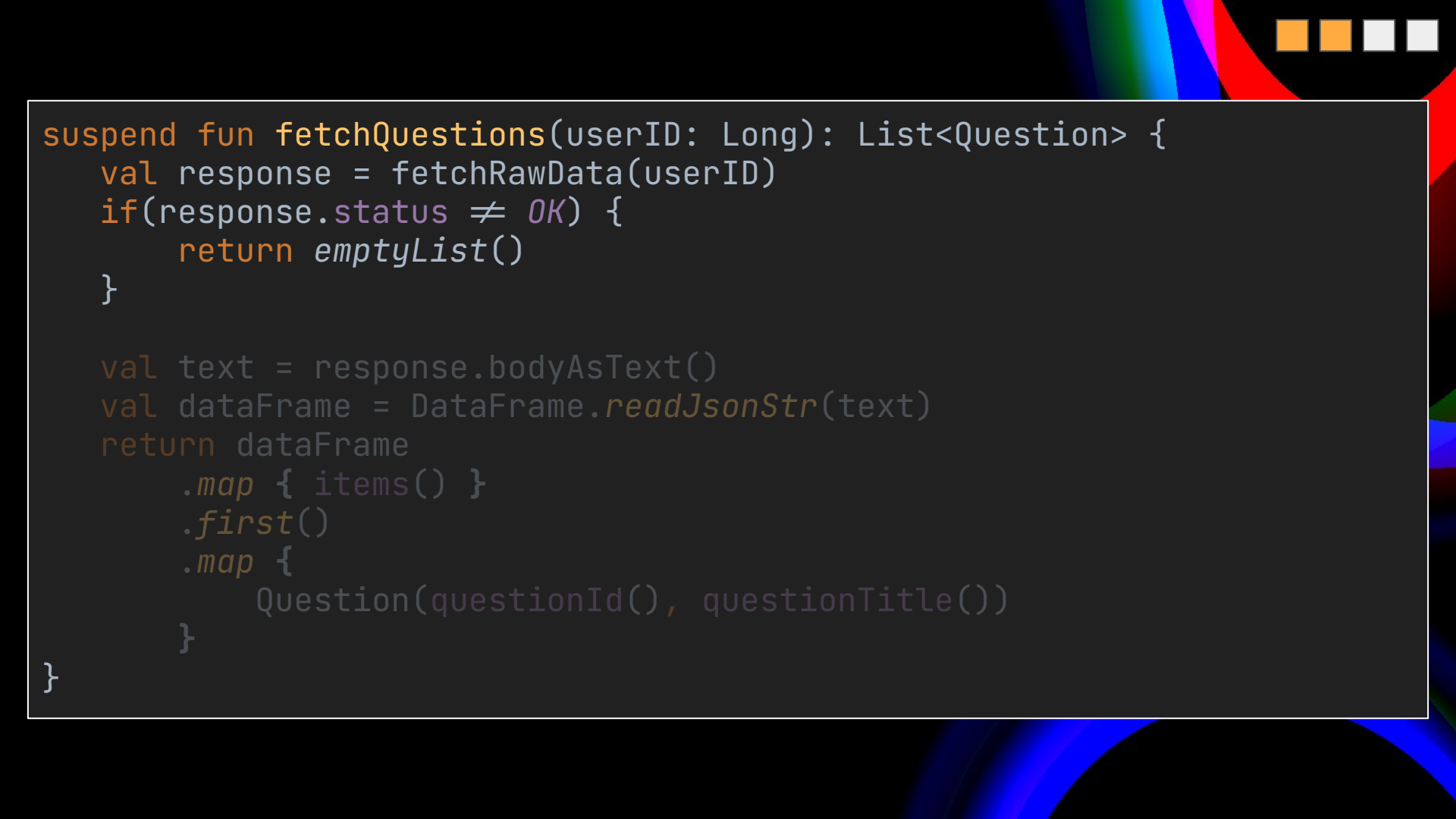
```
class StackExchangeClient(engine: HttpClientEngine) {  
    private val client = HttpClient(engine) {  
        install(ContentEncoding) {  
            gzip()  
            deflate()  
        }  
        defaultRequest {  
            url(ROOT_URL)  
        }  
    }  
    ...  
}
```

```
class StackExchangeClient(engine: HttpClientEngine) {  
    private val client = HttpClient(engine) {  
        install(ContentEncoding) {  
            gzip()  
            deflate()  
        }  
        defaultRequest {  
            url(ROOT_URL)  
        }  
    }  
    ...  
}
```

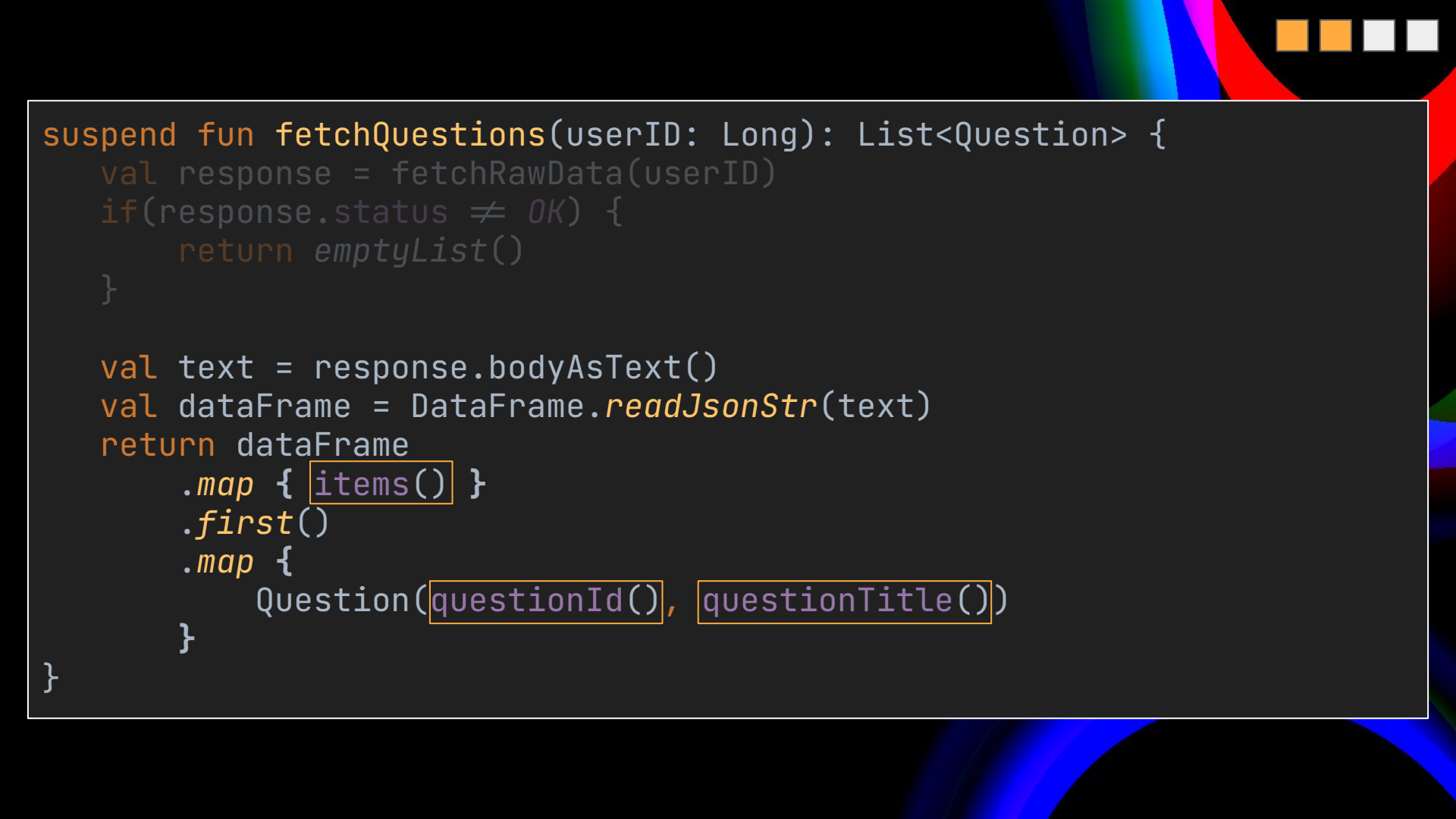




```
private suspend fun fetchRawData(userID: Long): HttpResponse {  
    val response = client.get {  
        accept(ContentType.parse("text/plain"))  
        url("$userID/questions?site=stackoverflow")  
    }  
    return response  
}
```



```
suspend fun fetchQuestions(userID: Long): List<Question> {  
    val response = fetchRawData(userID)  
    if(response.status != OK) {  
        return emptyList()  
    }  
  
    val text = response.bodyAsText()  
    val dataFrame = DataFrame.readJsonStr(text)  
    return dataFrame  
        .map { items() }  
        .first()  
        .map {  
            Question(questionId(), questionTitle())  
        }  
}
```



```
suspend fun fetchQuestions(userID: Long): List<Question> {  
    val response = fetchRawData(userID)  
    if(response.status != OK) {  
        return emptyList()  
    }  
  
    val text = response.bodyAsText()  
    val dataframe = DataFrame.readJsonStr(text)  
    return dataframe  
        .map { items() }  
        .first()  
        .map {  
            Question(questionId(), questionTitle())  
        }  
}
```



```
val items      = column<DataFrame<*>>("items")  
val questionId = column<Long>("question_id")  
val questionTitle = column<String>("title")
```

items

tags	answer_count	question_id	owner	view_count	title

ROMAN BELOV



**Kotlin for Data
Science, JetBrains**

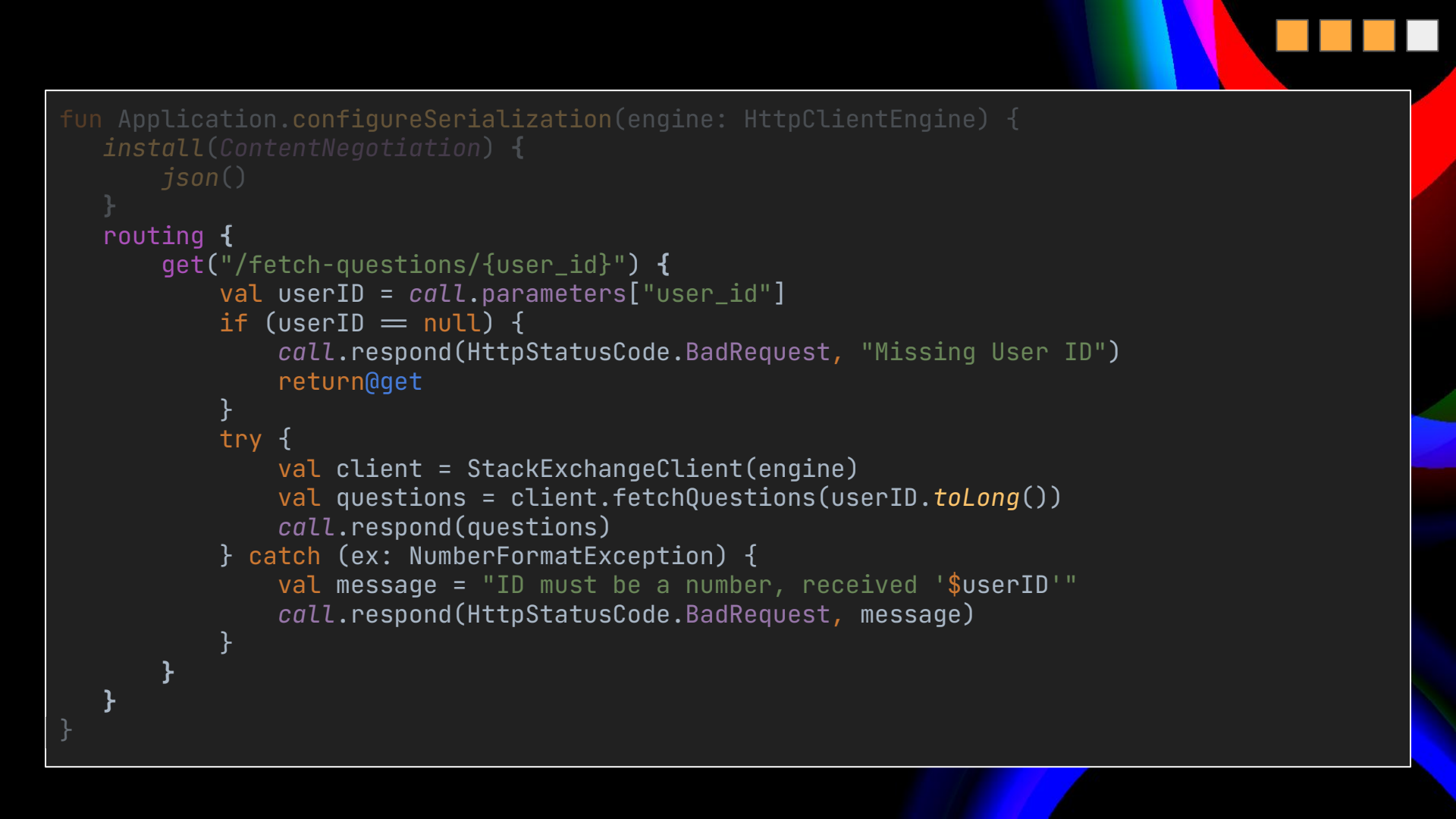
Present: Software Developer @ Kotlin for Data Science Previously: Marketing Manager @ Kotlin Software Developer @ dotMemory

**DataFrame: Kotlin's
Innovative Approach
to Data Structures**

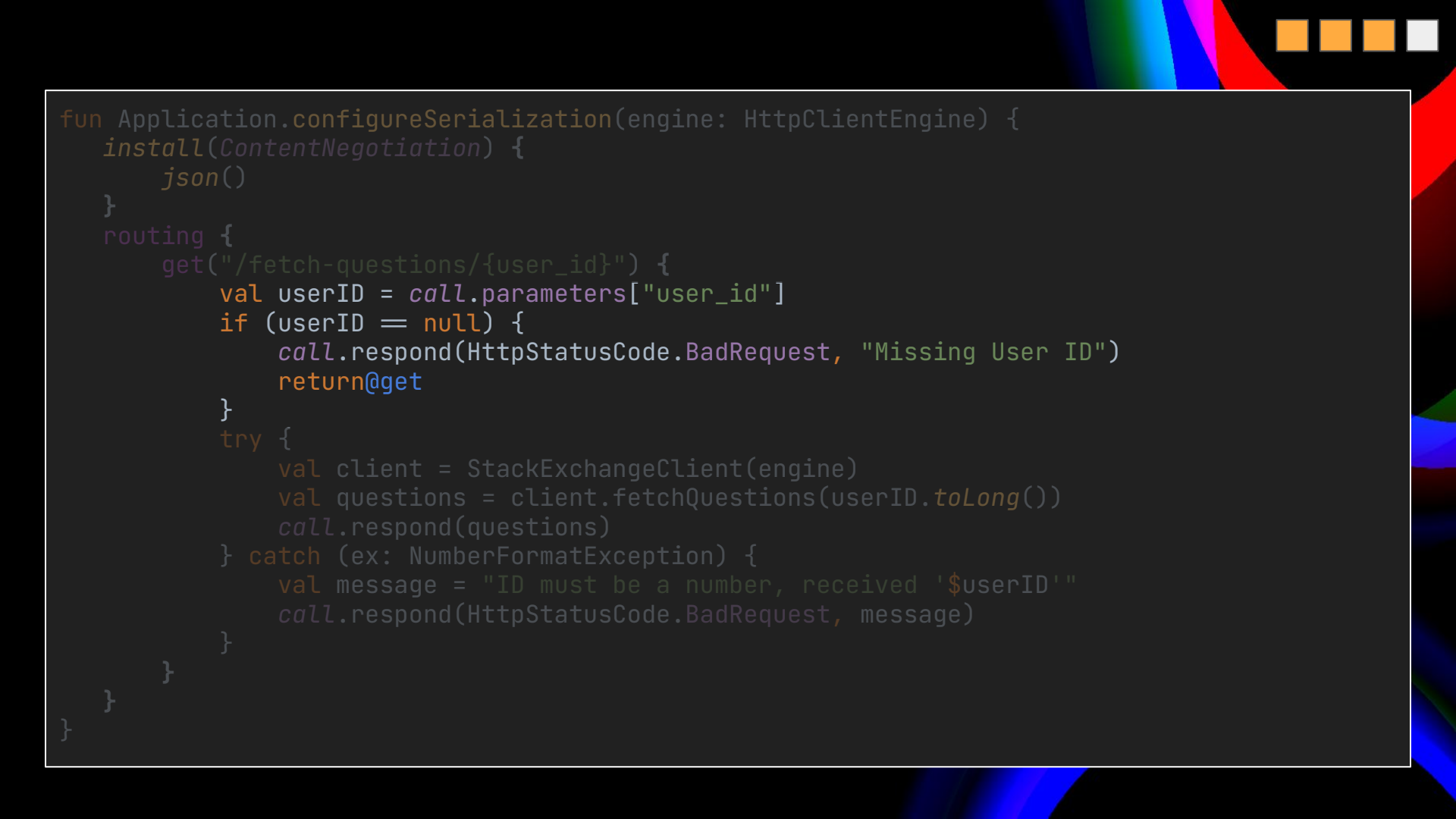
We'll talk about Dataframe — a library and a data structure that can help to read, write, generate, transform and organize data for displaying or plotting. It supports not only flat data structures but also hierarchical ones and thus can represent CSV, JSON or even subgraphs of objects in memory. While dataframes are usually mentioned in the context of data analytics, Kotlin Dataframe is also focused on general data cases and perfect for both professional and personal projects. We'll start with how it all works in Kotlin Notebook — an environment where you can write and execute fragments of code. It's a good fit for prototyping and trying out ideas. The IDE provides beautiful interactive tables that can display hierarchical data. Also, in the notebooks, the types of dataframe variables are updated after each fragment execution. It means that columns of the dataframe will appear in completion together with their content type. Kotlin Dataframe's new compiler plugin takes this idea even further and infers on-the-fly the types of columns while data wrangling. Come over, and we'll show it in action and share what Kotlin language features make it possible



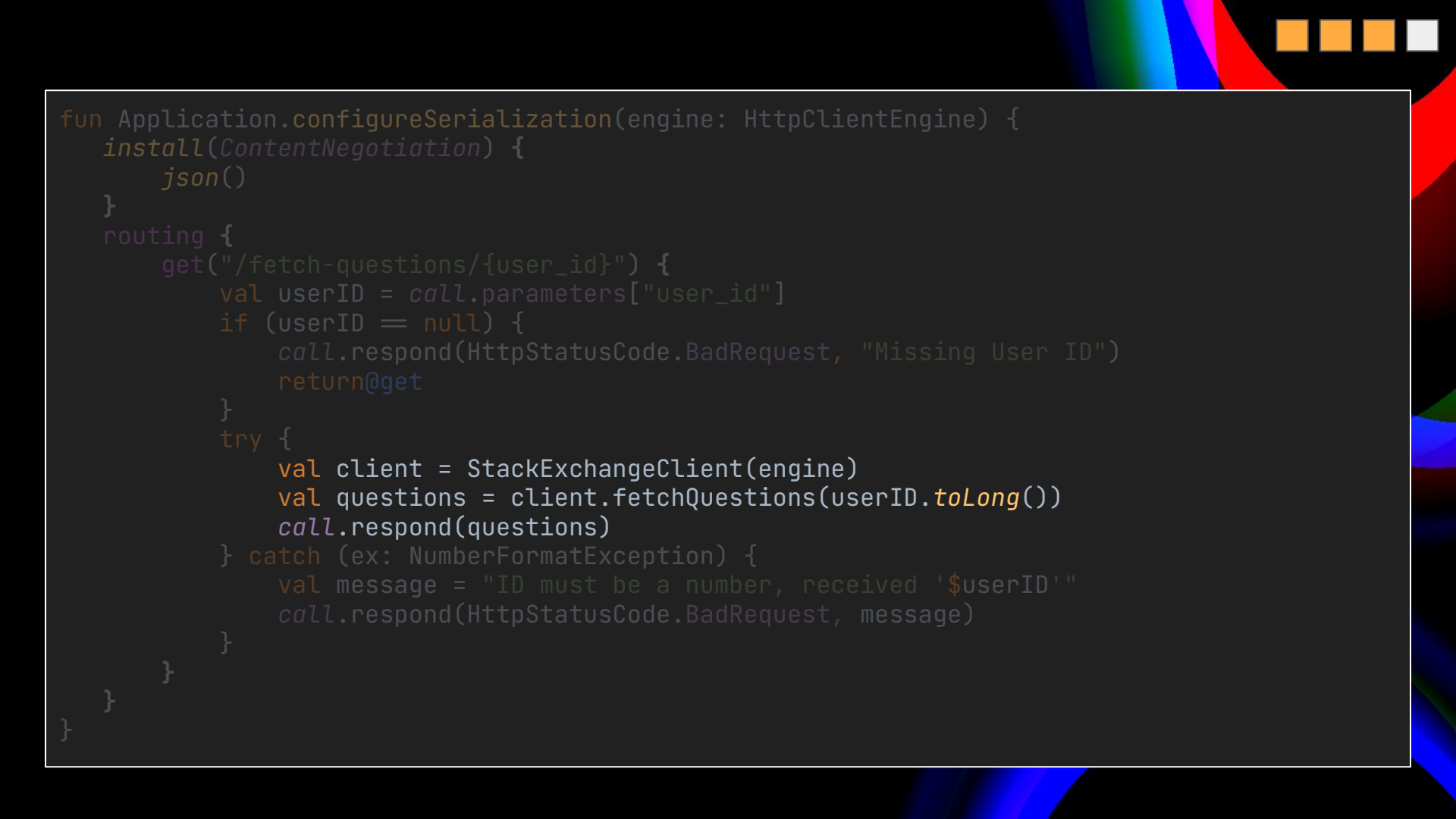
```
fun Application.configureSerialization(engine: HttpClientEngine) {
    install(ContentNegotiation) {
        json()
    }
    routing {
        get("/fetch-questions/{user_id}") {
            val userID = call.parameters["user_id"]
            if (userID == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing User ID")
                return@get
            }
            try {
                val client = StackExchangeClient(engine)
                val questions = client.fetchQuestions(userID.toLong())
                call.respond(questions)
            } catch (ex: NumberFormatException) {
                val message = "ID must be a number, received '$userID'"
                call.respond(HttpStatusCode.BadRequest, message)
            }
        }
    }
}
```



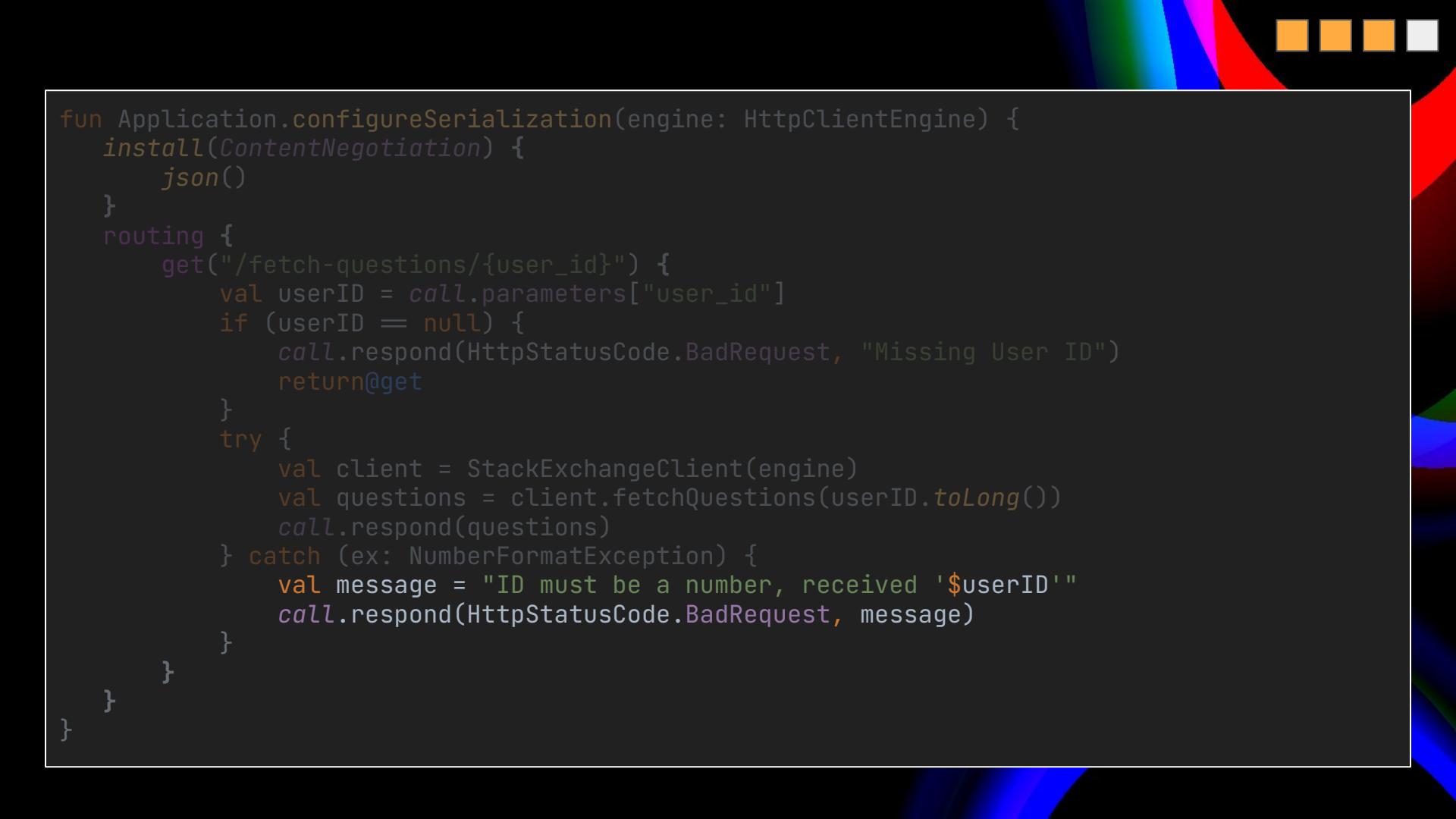
```
fun Application.configureSerialization(engine: HttpClientEngine) {
    install(ContentNegotiation) {
        json()
    }
    routing {
        get("/fetch-questions/{user_id}") {
            val userID = call.parameters["user_id"]
            if (userID == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing User ID")
                return@get
            }
            try {
                val client = StackExchangeClient(engine)
                val questions = client.fetchQuestions(userID.toLong())
                call.respond(questions)
            } catch (ex: NumberFormatException) {
                val message = "ID must be a number, received '$userID'"
                call.respond(HttpStatusCode.BadRequest, message)
            }
        }
    }
}
```



```
fun Application.configureSerialization(engine: HttpClientEngine) {
    install(ContentNegotiation) {
        json()
    }
    routing {
        get("/fetch-questions/{user_id}") {
            val userID = call.parameters["user_id"]
            if (userID == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing User ID")
                return@get
            }
            try {
                val client = StackExchangeClient(engine)
                val questions = client.fetchQuestions(userID.toLong())
                call.respond(questions)
            } catch (ex: NumberFormatException) {
                val message = "ID must be a number, received '$userID'"
                call.respond(HttpStatusCode.BadRequest, message)
            }
        }
    }
}
```



```
fun Application.configureSerialization(engine: HttpClientEngine) {
    install(ContentNegotiation) {
        json()
    }
    routing {
        get("/fetch-questions/{user_id}") {
            val userID = call.parameters["user_id"]
            if (userID == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing User ID")
                return@get
            }
            try {
                val client = StackExchangeClient(engine)
                val questions = client.fetchQuestions(userID.toLong())
                call.respond(questions)
            } catch (ex: NumberFormatException) {
                val message = "ID must be a number, received '$userID'"
                call.respond(HttpStatusCode.BadRequest, message)
            }
        }
    }
}
```



```
fun Application.configureSerialization(engine: HttpClientEngine) {
    install(ContentNegotiation) {
        json()
    }
    routing {
        get("/fetch-questions/{user_id}") {
            val userID = call.parameters["user_id"]
            if (userID == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing User ID")
                return@get
            }
            try {
                val client = StackExchangeClient(engine)
                val questions = client.fetchQuestions(userID.toLong())
                call.respond(questions)
            } catch (ex: NumberFormatException) {
                val message = "ID must be a number, received '$userID'"
                call.respond(HttpStatusCode.BadRequest, message)
            }
        }
    }
}
```



```
fun main() {  
    embeddedServer(  
        Netty,  
        port = 8080,  
        host = "0.0.0.0",  
        module = Application::module  
    ).start(wait = true)  
}
```



```
fun Application.module() {  
    configureSerialization(CIO.create())  
}
```

0.0.0.0:8080/fetch-questions/2635682

0.0.0.0:8080/fetch-questions/2635682 170%

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

▼ 0:

- id: 45659643
- title: "Signature of lambda return types in TypeScript"

▼ 1:

- id: 106402
- title: "Can Spring Parse and Inject Properties Files?"

▼ 2:

- id: 56583232
- ▼ title: "How to map multiple HTTP Verbs to the same path in HTTP4K"

▼ 3:

- id: 56170941
- ▼ title: "JAX-RS Client using Server Side Events prints the same items without completing"

▼ 4:

- id: 49631757
- ▼ title: "How to ensure Reactive Stream completes with Spring WebFlux and WebSockets"

▼ 5:

- id: 49590325

How did we build the service?

1. Created a Question type
2. Created a StackExchangeClient
 - a. Wrapped the Ktor HttpClient
 - b. Fetched the data from Stack Overflow
 - c. Magpied out the parts we want
 - d. Converted to an array of Question
3. Used this client in a route
4. Embedded the route in a server





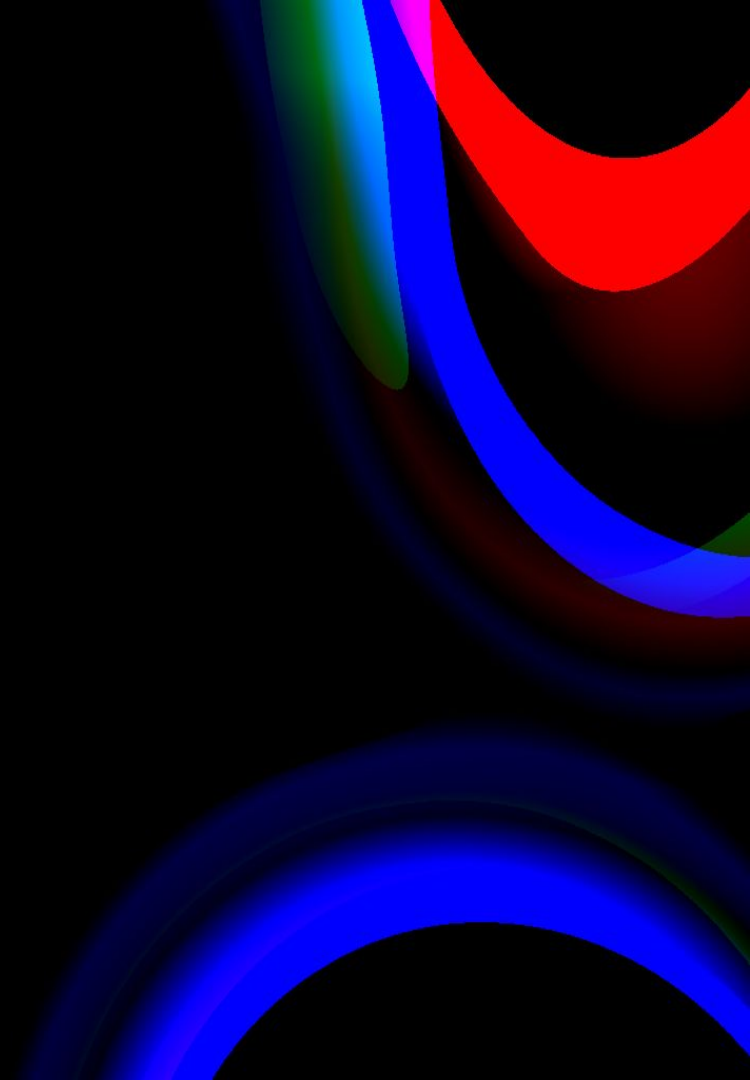
Testing

What are our test cases?

Valid ID returns a list of Question objects

Unknown ID returns an empty list

Invalid ID returns an error code



Creating tests

```
@Test  
fun sampleTest() = testApplication {
```

This code runs in a test environment

```
    application {
```

We setup the service for testing here

```
    }
```

We call endpoints and assert on output here

```
}
```

```
@Test
```

```
fun validUserIDProducesList() = testApplication {
```

← Create test environment

```
    application {
```

```
        configureSerialization(CIO.create())
```

← Inject HTTP client engine

```
    }
```

```
    val client = restClient()
```

```
    val response = client.get("/fetch-questions/2635682")
```

```
    assertEquals(HttpStatusCode.OK, response.status)
```

```
    val results = response.body<List<Question>>()
```

```
    assertEquals(9, results.size)
```

```
    val titles = results.map { it.title }
```

```
    assertContains(titles, "Signature ... in TypeScript")
```

```
    assertContains(titles, "How to ... path in HTTP4K")
```

```
}
```

```
@Test
fun validUserIDProducesList() = testApplication {

    application {
        configureSerialization(CIO.create())
    }

    val client = restClient()
    val response = client.get("/fetch-questions/2635682") ← Me!

    assertEquals(HttpStatusCode.OK, response.status)
    val results = response.body<List<Question>>()

    assertEquals(9, results.size)

    val titles = results.map { it.title }
    assertContains(titles, "Signature ... in TypeScript")
    assertContains(titles, "How to ... path in HTTP4K")
}
```

```
@Test
fun requiresNumericUserID() = testApplication {
    application {
        configureSerialization(CIO.create())
    }

    val client = restClient()
    val response = client.get("/fetch-questions/invalid")

    assertEquals(HttpStatusCode.BadRequest, response.status)
    assertEquals(
        "ID must be a number, received 'invalid'",
        response.bodyAsText()
    )
}
```

Incorrect



```
@Test
fun unknownUserIDProducesEmptyList() = testApplication {
    application {
        configureSerialization(CIO.create())
    }

    val client = restClient()
    val response = client.get("/fetch-questions/9191919191")

    assertEquals(HttpStatusCode.OK, response.status)

    val results = response.body<List<Question>>()
    assert(results.isEmpty())
}
```

WAT???



Creating unit tests

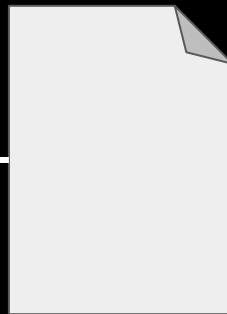
Our previous test still called out to Stack Overflow

It was an Integration Test, not a Unit Test

What happens when:

- User IDs are created or deleted?
- We don't have a network connection?
- Our test suite grows and grows?

```
private fun sampleData(): ByteReadChannel {  
    val path = Path("src/test/resources/sampleResponse.json")  
    return ByteReadChannel(path.readText())  
}
```



sampleResponse.json

```
@Test
fun validUserIDProducesList() = testApplication {

    val mockEngine = MockEngine { request →
        val pathMatches = request.url.fullPath.contains("123")
        val isGetRequest = request.method == HttpMethod.Get

        if (pathMatches && isGetRequest) {
            respond(sampleData(), OK, plainTextContent())
        } else {
            respond(emptyData(), HttpStatusCode.NotFound)
        }
    }

    application {
        configureSerialization(mockEngine)
    }
}
```



```
@Test
fun validUserIDProducesList() = testApplication {

    val mockEngine = MockEngine { request →
        val pathMatches = request.url.fullPath.contains("123")
        val isGetRequest = request.method == HttpMethod.Get

        if (pathMatches && isGetRequest) {
            respond(sampleData(), OK, plainTextContent())
        } else {
            respond(emptyData(), HttpStatusCode.NotFound)
        }
    }

    application {
        configureSerialization(mockEngine)
    }
}
```





```
val client = restClient()
val response = client.get("/fetch-questions/123")

assertEquals(OK, response.status)
val results = response.body<List<Question>>()

assertEquals(3, results.size)

val titles = results.map { it.title }
assertContains(titles, "What is Java")
assertContains(titles, "What is Kotlin")
assertContains(titles, "What is Haskell")
}
```

But wait! There's more...

You can mock out calls from the HttpClient in tests

- See <https://ktor.io/docs/testing.html#external-services>

This functionality can be combined with tools like Kotest

IntelliJ IDEA will generate tests for you

The AI Assistant can help write the code

```

12 fun Application.configureSerialization(engine: HttpClientEngine) {
13     install(ContentNegotiation) { this: ContentNegotiationConfig
14         json()
15     }
16     routing { this: Routing
17         get("/fetch-questions/{user_id}") { this: PipelineContext<Unit, Applic
18
19         Introduces a test for the selected Ktor routes.
20
21         Introduce local variable
22         Add explicit 'this'
23         Add names to call arguments
24         Create test for Ktor module
25         Create test for Ktor route
26         Extract Route to a Separate Method
27         Introduce import alias
28         OpenAPI documentation for current module is missing
29         Do not show implicit receiver and parameter hints
30
31         @ AI Actions...
32         Press F1 to toggle preview
33
34         sCode.BadRequest, message: "Missin
35
36         changeClient(engine)
37         val questions = client.fetchQuestions(userID.toLong())
38
39         call.respond(questions)
40     } catch (ex: NumberFormatException) {
41         val message = "ID must be a number, received '$userID'"
42         call.respond(HttpStatusCode.BadRequest, message)
43     }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }

```



Koin

Koin / Ktor integration

Our last example highlighted an architectural problem

We passed the `HttpClientEngine` through multiple layers

In this case it's only an annoyance, but will not scale

Good news is we can use Dependency Injection

```
fun Application.module() {  
    ...  
    configureSerialization(CIO.create())  
    ...  
}
```



```
fun Application.configureSerialization(engine: HttpClientEngine) {  
    ...  
    val client = StackExchangeClient(engine)  
    ...  
}
```

```
class StackExchangeClient(engine: HttpClientEngine) {  
    ...  
    private val client = HttpClient(engine)  
    ...  
}
```

Dependency Injection in Ktor | X

+

https://insert-koin.io/docs/reference/koin-ktor/ktor/

🔍 Search

Koin

Setup

Documentation

Tutorials

Cheat Sheets

Chat on Slack

🔍 Search

Versions

Setup

Core & Tests

Android

Compose

Annotations

Multiplatform

Ktor

Dependency Injection in Ktor

Ktor & Koin Isolated Context

🏠 > Ktor > Dependency Injection in Ktor

Dependency Injection in Ktor

The `koin-ktor` module is dedicated to bring dependency injection for Ktor.

Install Koin Plugin

To start a Koin container in Ktor, just install the `Koin` plugin like follow:

```
fun Application.main() {  
    // Install Koin  
    install(Koin) {  
        slf4jLogger()  
        modules(helloAppModule)  
    }  
}
```

Inject in Ktor

Koin `inject()` and `get()` functions are available from `Application`, `Route`, `Routing` classes:

Install Koin Plugin

Inject in Ktor

Resolve from Ktor Request Scope (since 3.5.0)

Run Koin from an external Ktor Module

Ktor Events

Koin / Ktor integration

Koin provides us with:

- A Ktor plugin to bootstrap the container
- A DSL for declaring dependencies
- A way to inject them as needed

```
fun Application.module() {  
    install(Koin) {  
        modules(module {  
            single { CIO.create() }  
        })  
    }  
    configureSerialization()  
}
```



```
class StackExchangeClient : KoinComponent {  
    private val engine by inject<HttpClientEngine>()  
    private val client = HttpClient(engine)  
    ...  
}
```

```
val testModule = module {
    single { CIO.create() }
}

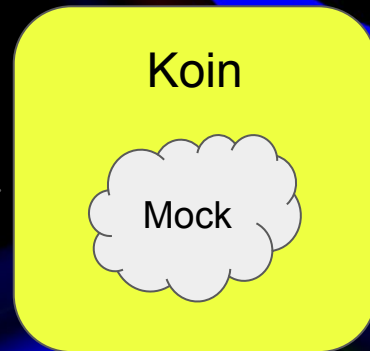
class IntegrationTest {

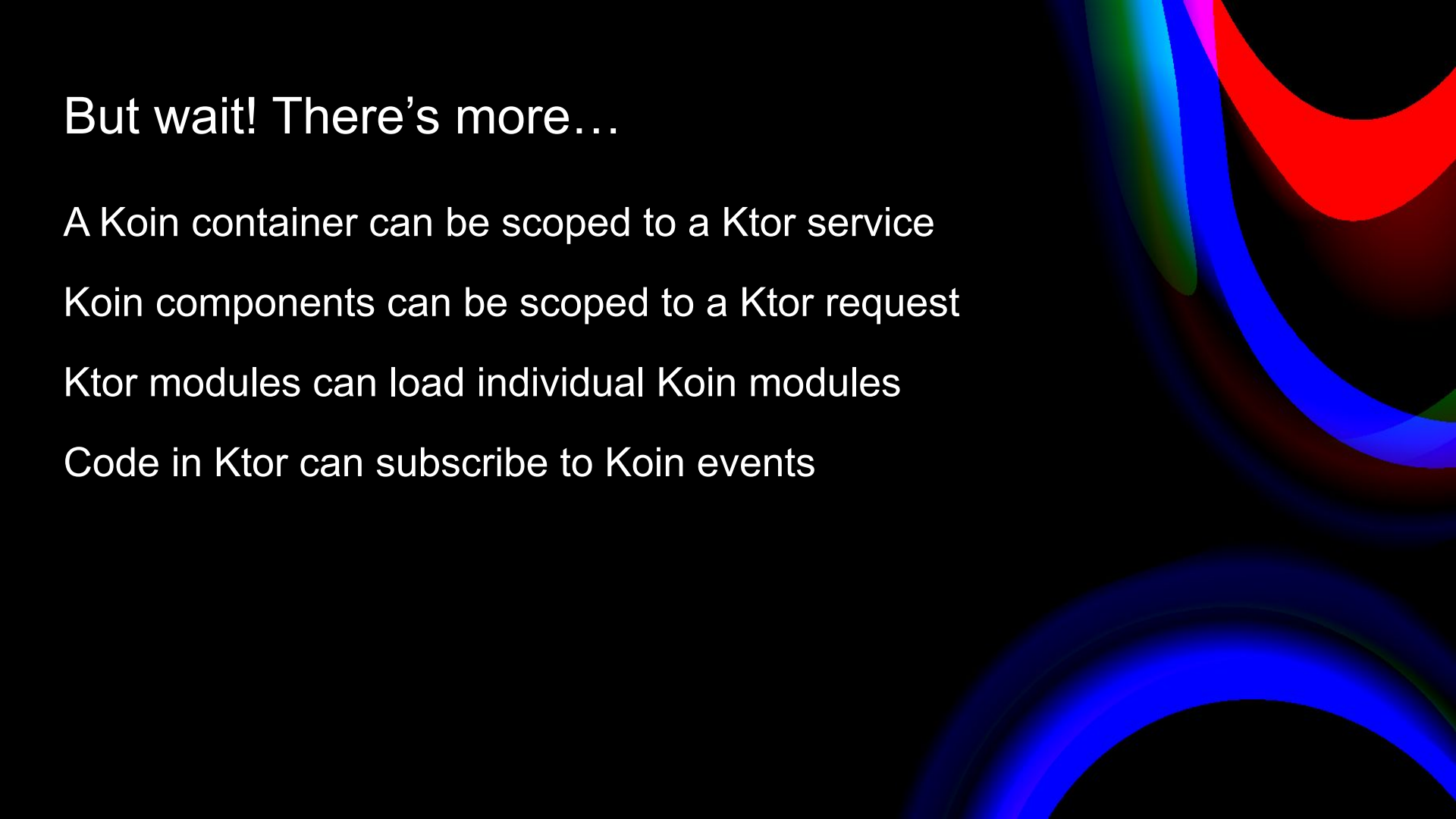
    @After
    fun shutdown() {
        stopKoin()
    }

    @Test
    fun validUserIDProducesList() = testApplication {
        application {
            install(Koin) {
                modules(testModule)
            }
            configureSerialization()
        }
        ...
    }
}
```



```
class UnitTest {  
    @AfterTest  
    fun shutdown() {  
        stopKoin()  
    }  
  
    @Test  
    fun sampleTest() = testApplication {  
        val mockEngine = MockEngine {  
            respond(emptyData(), HttpStatusCode.BadRequest)  
        }  
  
        application {  
            install(Koin) {  
                modules(org.koin.dsl.module {  
                    single<HttpClientEngine> { mockEngine }  
                })  
            }  
            configureSerialization()  
        }  
        ...  
    }  
}
```





But wait! There's more...

A Koin container can be scoped to a Ktor service

Koin components can be scoped to a Ktor request

Ktor modules can load individual Koin modules

Code in Ktor can subscribe to Koin events

But wait! There's even more...



Ktor 2024 Roadmap DI Update

Since the Ktor Roadmap for 2024 was published there have been a lot of questions regarding Dependency Injection (DI). Specifically how Ktor will...

 Garth Gilmour
March 18, 2024



The Ktor Roadmap for 2024

In 2024, we aim to expand the Ktor ecosystem while keeping the framework lightweight and flexible. One of our objectives is to help Ktor developers...

 Garth Gilmour
March 12, 2024

 5



GraphQL

GraphQL

0.0.0.0:8080/graphql

170%

☆

🔒

🔍

🔗

☰

📄

< Docs

🔍 ⌘ K

🕒

Query

🔧 Fields

hello: String!

questions(userID: String!): [Question!]!

🔄

⌘

⚙️

1 ▾ {

2 ▾ questions(userID: "2635682") {

3 title

4 }

5 }

▶

🔧

🗑️

📄

Variables

Headers

▾

1

+ GraphQL

▾ {

▾ "data": {

▾ "questions": [

▾ {

"title":

"Signature of lambda

return types in

TypeScript"

},

▾ {

"title": "Can

Spring Parse and Inject

Properties Files?"

},

▾ {

"title": "How to

map multiple HTTP Verbs to

the same path in HTTP4K"

}

]

}

}

```
fun Application.configureGraphQL() { ← Replaces REST
    install(GraphQL) {
        schema {
            packages = listOf("demos.kotlin.conf.model")
            queries = listOf(
                HelloGraphQLQuery(),
                StackOverflowQuery()
            )
        }
    }
    install(Routing) {
        graphQLPostRoute()
        graphiQLRoute()
        graphQLSDLRoute()
    }
}
```

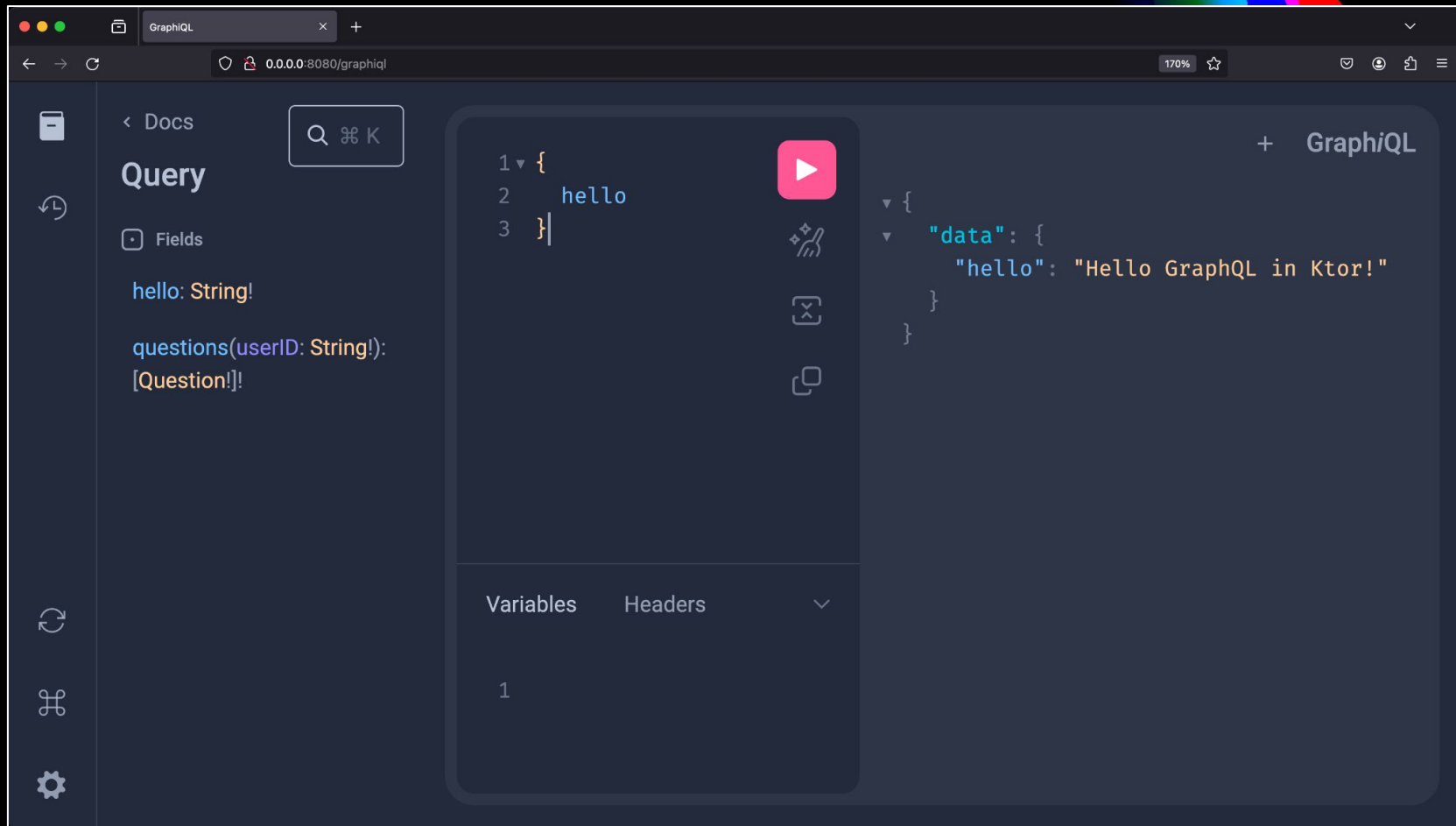
```
fun Application.configureGraphQL() {  
    install(GraphQL) {  
        schema {  
            packages = listOf("demos.goto.ams.model")  
            queries = listOf(  
                HelloGraphQLQuery(),  
                StackOverflowQuery()  
            )  
        }  
    }  
    install(Routing) {  
        graphQLPostRoute()  
        graphiQLRoute()  
        graphQLSDLRoute()  
    }  
}
```

Contains types
↓
Queries to run ←

```
class HelloGraphQLQuery : Query {  
    fun hello(): String = "Hello GraphQL in Ktor!"  
}
```

```
class StackOverflowQuery : Query {  
    private val client = StackExchangeClient()  
  
    suspend fun questions(userID: String) =  
        client.fetchQuestions(userID.toInt())  
}
```

```
fun Application.configureGraphQL() {  
    install(GraphQL) {  
        schema {  
            packages = listOf("demos.goto.ams.model")  
            queries = listOf(  
                HelloGraphQLQuery(),  
                StackOverflowQuery()  
            )  
        }  
    }  
    install(Routing) {  
        graphQLPostRoute() ← Accept queries via POST  
        graphiQLRoute() ← Include a GraphQL UI  
        graphQLSDLRoute() ← Generate a schema  
    }  
}
```



GraphQL

0.0.0.0:8080/graphql

170%

☆

🔒

🔍

🔗

☰

📄

< Docs

🔍 ⌘ K

🕒

Query

🔧 Fields

hello: String!

questions(userID: String!): [Question!]!

🔄

🔗

⚙️

1 ▾ {

2 ▾ questions(userID: "2635682") {

3 title

4 }

5 }

▶

🔧

🗑️

📄

Variables

Headers

▾

1

+ GraphQL

▾ {

▾ "data": {

▾ "questions": [

▾ {

"title":

"Signature of lambda

return types in

TypeScript"

},

▾ {

"title": "Can

Spring Parse and Inject

Properties Files?"

},

▾ {

"title": "How to

map multiple HTTP Verbs to

the same path in HTTP4K"

}

]

}

}

Supercharge your GraphQL with Ktor and GraalVM

Derek Kuc

@derek_kuc
KotlinConf'23
Amsterdam

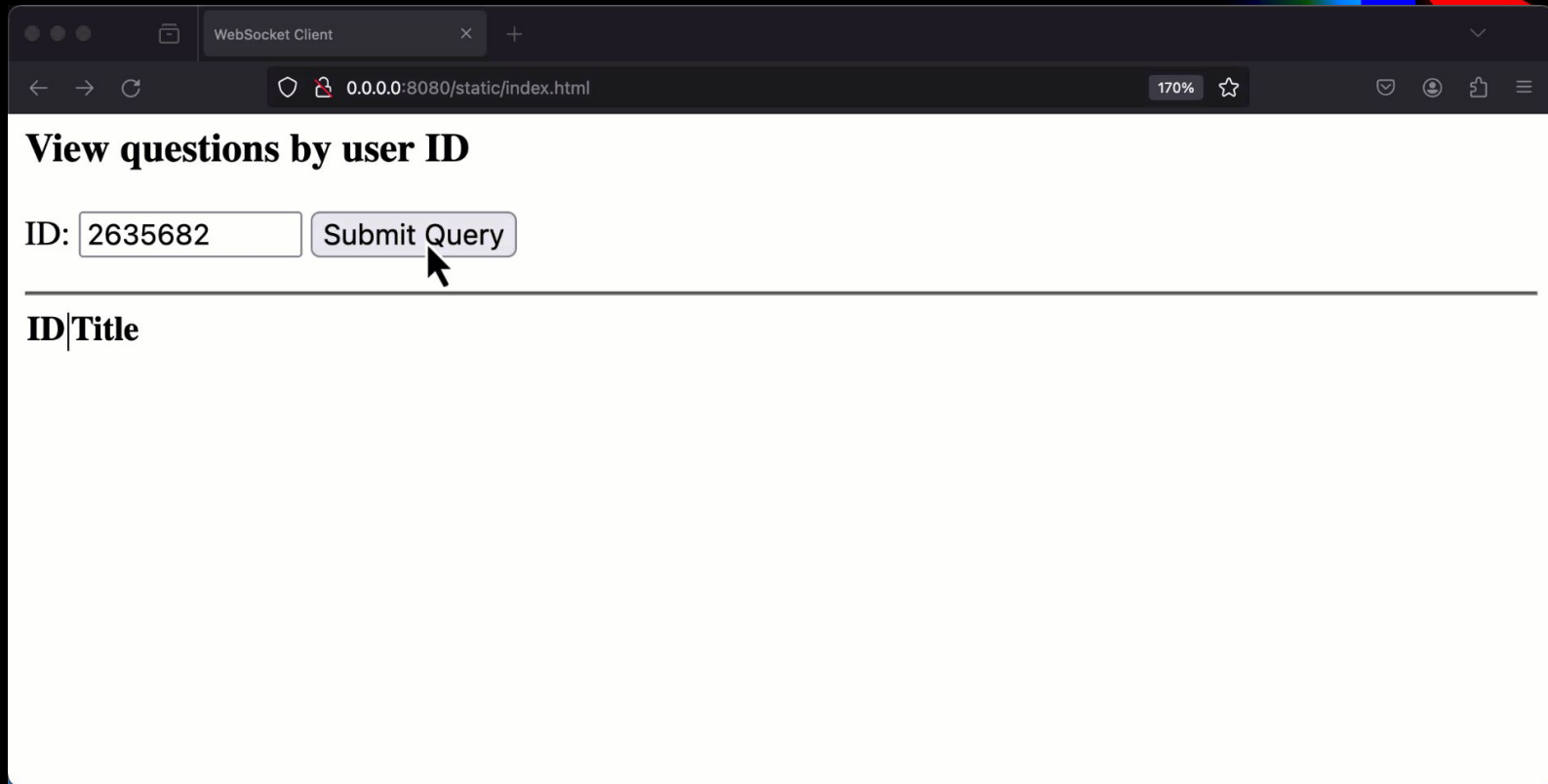


0:23 / 44:17 • Introduction >





WebSockets



Ktor and WebSockets

WebSockets are a scary topic

- You may have been bitten by them

Ktor support is much better than most

- This is partially a matter of timing
- WebSockets predate Ktor, so the designers kept them in mind

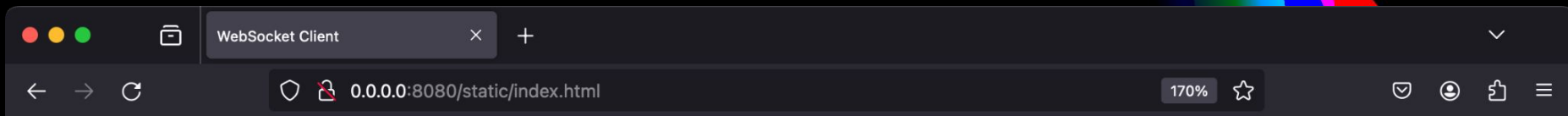


```
fun Application.configureWebSockets() {  
    install(WebSockets) {  
        contentConverter = KotlinxWebsocketSerializationConverter(Json)  
        pingPeriod = Duration.ofSeconds(15)  
        timeout = Duration.ofSeconds(15)  
        maxFrameSize = Long.MAX_VALUE  
        masking = false  
    }  
    routing {  
        val client = StackExchangeClient()  
        websocket("/questions") {  
            val user = receiveDeserialized<User>()  
            val questions = client.fetchQuestions(user.id.toLong())  
            for (question in questions) {  
                sendSerialized(question)  
                delay(1000)  
            }  
        }  
    }  
}
```



No Magic!

```
fun Application.configureWebSockets() {  
    install(WebSockets) {  
        contentConverter = KotlinxWebsocketSerializationConverter(Json)  
        pingPeriod = Duration.ofSeconds(15)  
        timeout = Duration.ofSeconds(15)  
        maxFrameSize = Long.MAX_VALUE  
        masking = false  
    }  
    routing {  
        val client = StackExchangeClient()  
        websocket("/questions") {  
            val user = receiveDeserialized<User>() ←  
            val questions = client.fetchQuestions(user.id.toLong())  
            for (question in questions) {  
                sendSerialized(question) ←  
                delay(1000)  
            }  
        }  
    }  
}
```



View questions by user ID

ID:

ID	Title
45659643	Signature of lambda return types in TypeScript
106402	Can Spring Parse and Inject Properties Files?
56583232	How to map multiple HTTP Verbs to the same path in HTTP4K
56170941	JAX-RS Client using Server Side Events prints the same items without completing
49631757	How to ensure Reactive Stream completes with Spring WebFlux and WebSockets
49590325	Cannot use <code>'subscribe'</code> or <code>'subscribeWith'</code> with <code>'ReactorNettyWebSocketClient'</code> in Kotlin
45081057	How to alter test object properties in KotlinTest via interceptTestCase
22808485	Purpose of third argument to <code>'reduce'</code> function in Java 8 functional programming
150445	Can a Sequence Diagram realistically capture your logic in the same depth as code?

```
<body onload="setupSocket()">
  <h3>View questions by user ID</h3>
  <form onsubmit="return sendUserIDToServer()">
    <label for="userID">ID: </label>
    <input type="text" id="userID" name="userID"
      size="10" value="2635682">
    <input type="submit">
  </form>
  <hr/>
  <table rules="all">
    <thead>
      <tr>
        <th>ID</th><th>Title</th>
      </tr>
    </thead>
    <tbody id="questionsTableBody">
    </tbody>
  </table>
</body>
```

```
function setupSocket() {  
    serverURL = 'ws://0.0.0.0:8080/questions';  
    socket = new WebSocket(serverURL);  
    socket.onmessage = readAndDisplayQuestion;  
}  
  
function readAndDisplayQuestion(event) {  
    let question = JSON.parse(event.data);  
    addQuestionToTable(question);  
}
```

```
function sendUserIDToServer() {
    let userID = buildUserIDFromForm();
    sendUserIDViaSocket(userID);
    //prevent form submission
    return false;
}

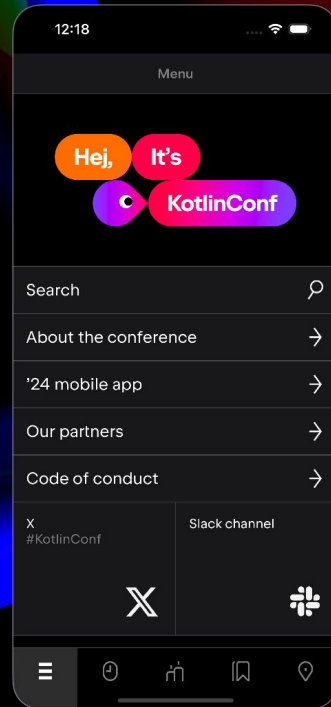
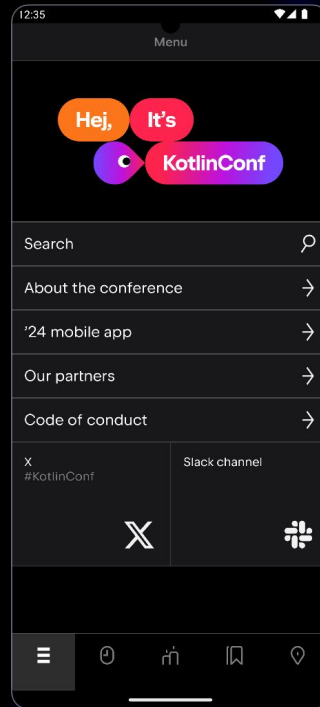
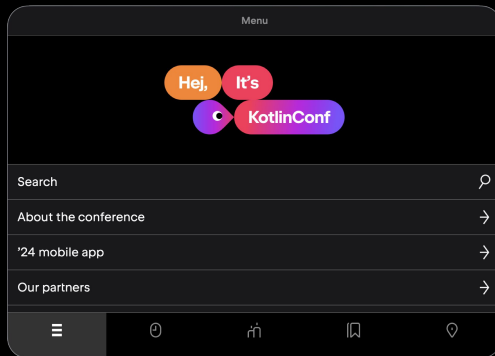
function buildUserIDFromForm() {
    return {
        id: getFormValue("userID"),
    }
}

function sendUserIDViaSocket(data) {
    socket.send(JSON.stringify(data));
}
```

But wait! There's more...

Ktor + WebSockets is very powerful

Especially when combined with WASM





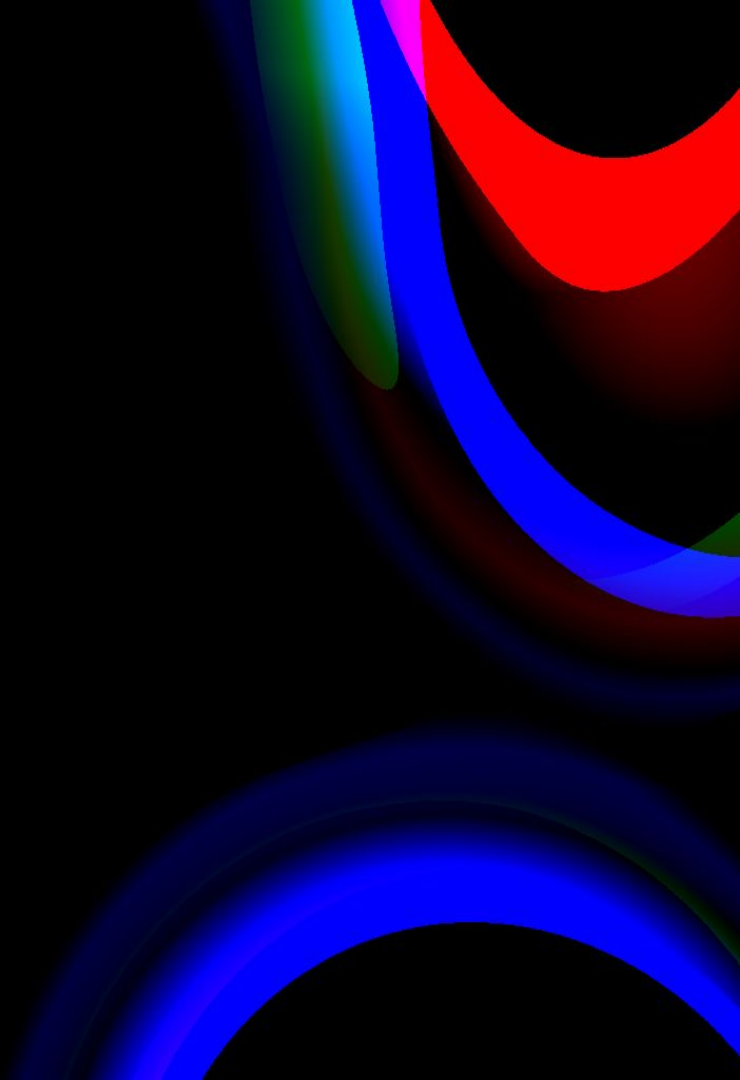
KMP

Ktor and KMP

Ktor Client is fully multiplatform

Ktor Server supports Kotlin Native

- You can build services as executables





November 21, 2023
17:00 UTC

The State of Kotlin Multiplatform

Svetlana Isakova
Márton Braun



November 23, 2023
16:00 UTC

Build Apps for iOS and Android with Shared Logic and Native UIs

Pamela Hill
Garth Gilmour



November 28, 2023
16:00 UTC

Build Apps for iOS, Android, and Desktop in 100% Kotlin with Compose Multiplatform

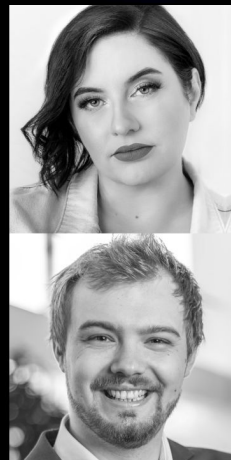
Sebastian Aigner
Márton Braun



November 30, 2023
16:00 UTC

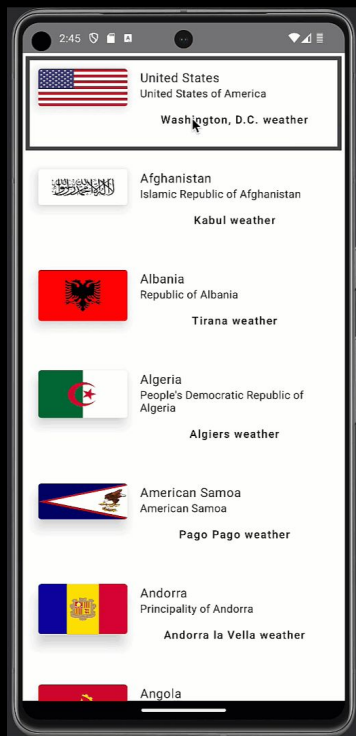
iOS Development with Kotlin Multiplatform: Tips and Tricks

Pamela Hill
Tadeas Kriz

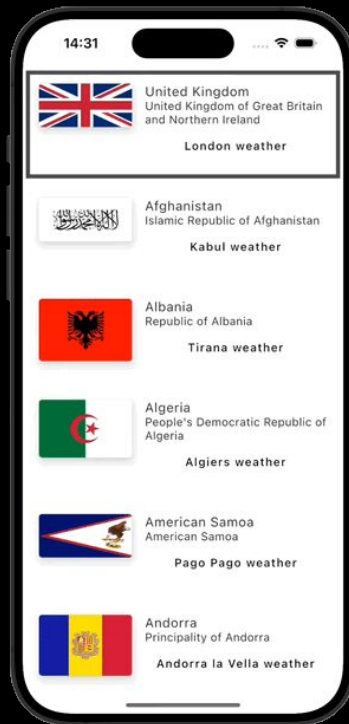




Android + CM

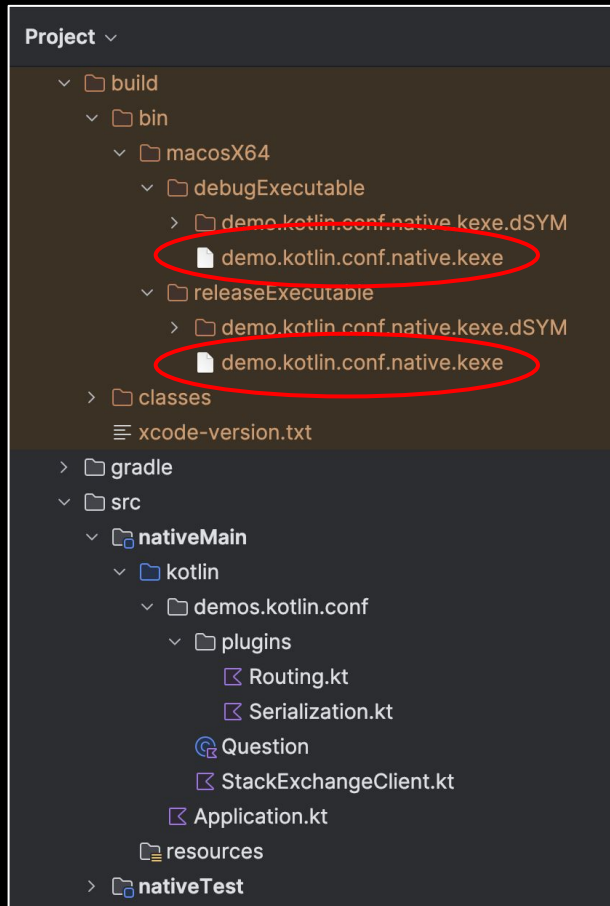


iOS + CM



Desktop + CM





```
kotlin {  
    macOSX64 {  
        binaries {  
            executable()  
        }  
    }  
  
    sourceSets {  
        macOSMain {  
            dependencies {  
                //Replaces CIO in the Ktor Client for TLS  
                implementation("io.ktor:ktor-client-curl:$ktor_version")  
            }  
        }  
        macOSTest {  
            dependencies { ... }  
        }  
    }  
}
```

Use Curl for TLS



```
fun Application.module() {  
    configureSerialization(io.ktor.client.engine.curl.Curl.create())  
    configureRouting()  
}
```

```
suspend fun fetchQuestions(userID: Long): List<Question> {
```

```
    ...
```

```
    val text = response.bodyAsText()
```

```
    val root = Json.parseToJsonElement(text)
```

```
    val questions = root
```

```
        .jsonObject["items"]
```

```
        ?.jsonArray
```

```
        ?.map {
```

```
            it.jsonObject.run {
```

```
                Question(
```

```
                    get("question_id").toString().toLong(),
```

```
                    get("title").toString()
```

```
                )
```

```
            }
```

```
        }
```

```
    return questions ?: emptyList()
```

```
}
```



Dataframe not available, so use
low-level kotlinx-serialization

```
08:28:27: Executing 'macosX64Binaries'...
```

```
> Task :checkKotlinGradlePluginConfigurationErrors
> Task :compileKotlinMacosX64
> Task :xcodeVersion
> Task :linkDebugExecutableMacosX64
> Task :linkReleaseExecutableMacosX64
> Task :macosX64Binaries
```

```
BUILD SUCCESSFUL in 1m 21s
```

```
5 actionable tasks: 5 executed
```

```
08:29:49: Execution finished 'macosX64Binaries'.
```



Terminal Local x + v

```
> ls
```

```
demo.kotlin.conf.native.kexe
```

```
demo.kotlin.conf.native.kexe.dSYM
```

```
> ./demo.kotlin.conf.native.kexe
```

```
[INFO] (ktor.application): Application started in 0.003 seconds.
```

```
[INFO] (ktor.application): Responding at http://0.0.0.0:8080
```



0.0.0.0:8080/fetch-questions/2635682

← → ↺

0.0.0.0:8080/fetch-questions/2635682

170% ☆

🔍 🧑 📄 ☰

JSON Raw Data Headers

Save Copy Collapse All Expand All

🔍 Filter JSON

▼ 0:

id: 45659643

title: "Signature of lambda return types in TypeScript"

▼ 1:

id: 106402

title: "Can Spring Parse and Inject Properties Files?"

▼ 2:

id: 56583232

▼ title: "How to map multiple HTTP Verbs to the same path in HTTP4K"

▼ 3:

id: 56170941

▼ title: "JAX-RS Client using Server Side Events prints the same items without completing"

▼ 4:

id: 49631757

▼ title: "How to ensure Reactive Stream completes with Spring WebFlux and WebSockets"

▼ 5:



OpenTelemetry

OpenTelemetry

Observability is a vital component of DevOps

- Now DevSecOps, or maybe DevFinSecOps 🤔😂

OpenTelemetry offers a universal solution

- The spec applies across platforms and industries
- The API and SDK exist for all major languages

They provide a plugin to integrate with Ktor

- You can instrument your code on client and server

What is OpenTelemetry? | Open X

+

← → ↺

https://opentelemetry.io/docs/what-is-opentelemetry/

📄 ☆

🔖 🔄 🏠 ☰

OpenTelemetry

Docs Ecosystem Status Community Blog 🔍 Search this site...

Docs

What is OpenTelemetry?

▶ Getting Started

▶ Concepts

▶ Demo

▶ Language APIs & SDKs

▶ Collector

▶ Kubernetes

▶ FaaS

▶ Migration

▶ Specs

▶ Security

▶ Contributing

Docs / What is OpenTelemetry?

What is OpenTelemetry?

A short explanation of what OpenTelemetry is and isn't.

OpenTelemetry is an [Observability](#) framework and toolkit designed to create and manage telemetry data such as [traces](#), [metrics](#), and [logs](#). Crucially, OpenTelemetry is vendor- and tool-agnostic, meaning that it can be used with a broad variety of Observability backends, including open source tools like [Jaeger](#) and [Prometheus](#), as well as commercial offerings.

OpenTelemetry is not an observability backend like Jaeger, Prometheus, or other commercial vendors. OpenTelemetry is focused on the generation, collection, management, and export of telemetry. A major goal of OpenTelemetry is that you can easily instrument your applications or systems, no matter their language, infrastructure, or runtime environment. Crucially, the storage and visualization of telemetry is intentionally left to other tools.

What is observability?

[Observability](#) is the ability to understand the internal state of a system by examining its outputs. In the context of software, this means being able to understand the internal state of a system by examining its telemetry data, which includes traces, metrics, and logs.

To make a system observable, it must be [instrumented](#). That is, the code must emit [traces](#), [metrics](#), or [logs](#). The instrumented data must then be sent to an observability backend.

📄 View page source

🔗 Edit this page

🔗 Create documentation issue

What is observability?

Why OpenTelemetry?

Main OpenTelemetry components

Extensibility

History

What next?

```
fun Application.configureOpenTelemetry() {  
    install(KtorServerTracing) {  
        val openTelemetry = getOpenTelemetry()  
        setOpenTelemetry(openTelemetry)  
  
        attributeExtractor {  
            onStart {  
                val userId = request  
                    .call  
                    .request  
                    .path()  
                    .substringAfterLast("/")  
                attributes.put("user_id", userId)  
            }  
        }  
    }  
}
```

JAEGER UI

Search

Compare

System Architecture

Monitor

Q Lookup by Trace ID...

About Jaeger

Search

Upload

Service (2)
kotlin-conf-opentelemetry

Operation (2)
all

Tags ?
http.status_code=200 error=true

Lookback
Last Hour

Max Duration
e.g. 1.2s, 100m...

Min Duration
e.g. 1.2s, 100m...

Limit Results
20

Find Traces

2 Traces

Sort: Most Recent Download Results Deep Dependency Graph

Compare traces by selecting result items

☐	kotlin-conf-opentelemetry: GET 5dfbca7	2.56ms
1 Span	kotlin-conf-opentelemetry (1)	Today 4:01:07 pm 3 minutes ago
☐	kotlin-conf-opentelemetry: GET /fetch-questions/{user_id} d4d4c48	520.41ms
1 Span	kotlin-conf-opentelemetry (1)	Today 4:01:06 pm 3 minutes ago

Jaeger UI

localhost:16686/trace/d4d4c4836048ea64c2c78f63f9da07eb 120%

JAEGER UI Search Compare System Architecture Monitor Q Lookup by Trace ID... About Jaeger

← kotlin-conf-opentelemetry: GET / fetch-questions/{user_id} d4d4c48 Find... ? ? ? ? ? Trace Timeline Archive Trace

Trace Start May 20 2024, 16:01:06.784 Duration 520.41ms Services 1 Depth 1 Total Spans 1

0µs 130.1ms 260.21ms 390.31ms 520.41ms

Service & Operation > > > 0µs 130.1ms 260.21ms 390.31ms 520.41ms

kotlin-conf-opentelemetry GET /fetch-questio...

GET /fetch-questions/{user_id} Service: kotlin-conf-opentelemetry 520.41ms | Start Time: 0µs

Tags

- http.method GET
- http.response_content_length 891
- http.route /fetch-questions/{user_id}
- http.scheme http
- http.status_code 200
- http.target /fetch-questions/2635682
- internal.span.format otlp

Jaeger UI

localhost:16686/trace/d4d4c4836048ea64c2c78f63f9da07eb

120%

🔍

🔒

👤

📄

☰

JAEGER UI

Search

Compare

System Architecture

Monitor

🔍 Lookup by Trace ID...

About Jaeger

⬅️

▼

kotlin-conf-opentelemetry: GET / fetch-questions/{user_id} d4d4c48

Find...

?

⚙️

⬆️

⬇️

✖️

🔗

Trace Timeline

📄 Archive Trace

Trace Start May 20 2024, 16:01:06.784

Duration 520.41ms

Services 1

Depth 1

Total Spans 1

0μs

130.1ms

260.21ms

390.31ms

520.41ms

Service & Operation

▼

>

⌵

>>

||

0μs

130.1ms

260.21ms

390.31ms

520.41ms

net.host.name0.0.0.0

net.host.port8080

net.protocol.namehttp

net.protocol.version1.1

span.kindserver

user_agent.originalMozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36

user_id2635682

> Process:

otel.library.name = io.opentelemetry.ktor-2.0

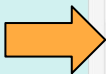
otel.library.version = 1.32.0-alpha

telemetry.sdk.language = java

telemetry.sdk.name = o...

SpanID: 5dfdfcf34d23335

🔗





Retries

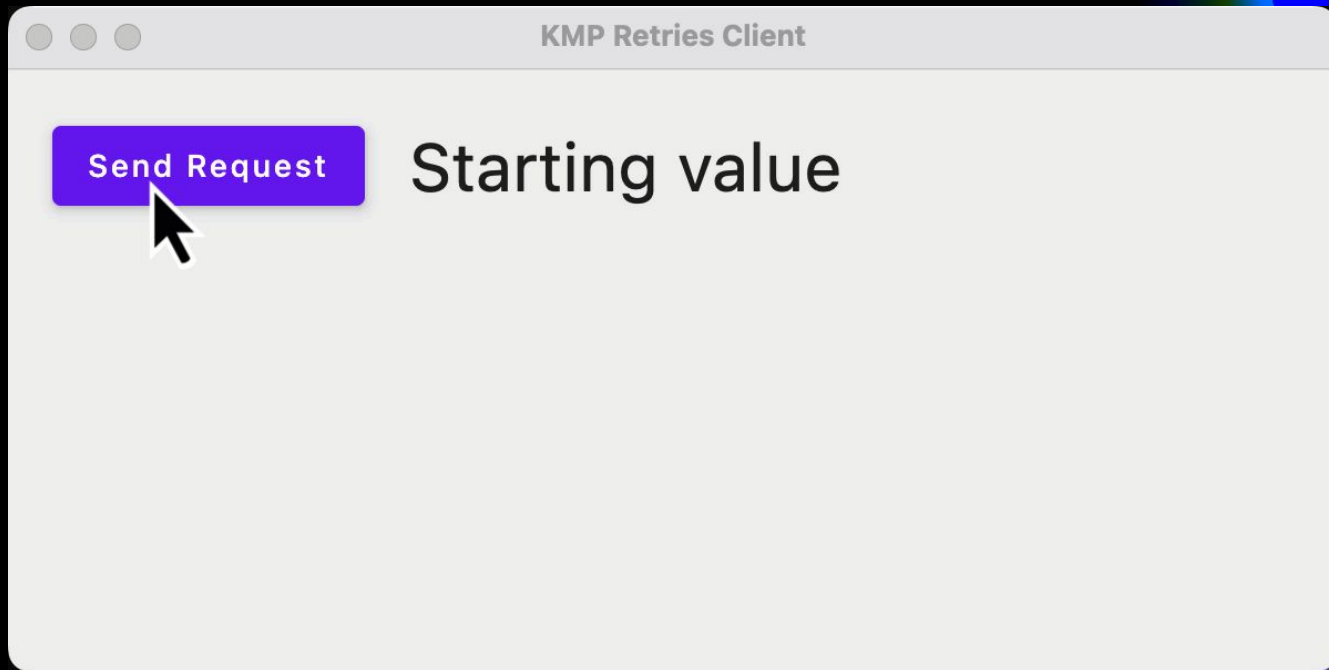
Retries

Part of the “No Magic” philosophy is fine grained control

It is very easy to customize any aspect of Ktor

Let's look retrying client requests



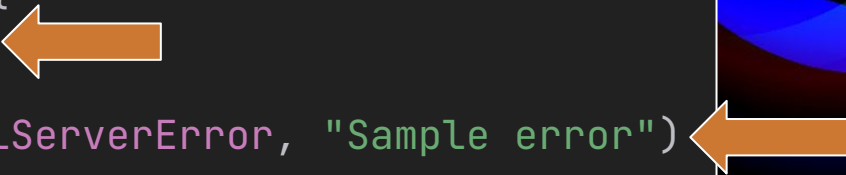


Retries (The Server Side)



```
var callCount = 1

fun Application.configureSerialization() {
    install(ContentNegotiation) {
        json()
    }
    routing {
        get("/fetch-questions/{user_id}") {
            if (callCount % 4 == 0) {
                processRequest()
            } else {
                call.respond(InternalServerError, "Sample error")
            }
            callCount++
        }
    }
}
```



Retries (The Client Side)



```
fun buildClient(logMessages: MutableList<String>): HttpClient {  
    val client = HttpClient(CIO) {  
        install(ContentNegotiation) {  
            json()  
        }  
        install(HttpRequestRetry) {  
            retryOnServerError(maxRetries = 5) ←  
            delayMillis { retry → ←  
                retry * 2000L  
            }  
        }  
    }  
    client.plugin(HttpSend).intercept { request →  
        val call = execute(request)  
        val status = call.response.status.value  
        val time = currentTime()  
  
        logMessages.add("Call at $time which returned $status")  
        call  
    }  
    return client  
}
```

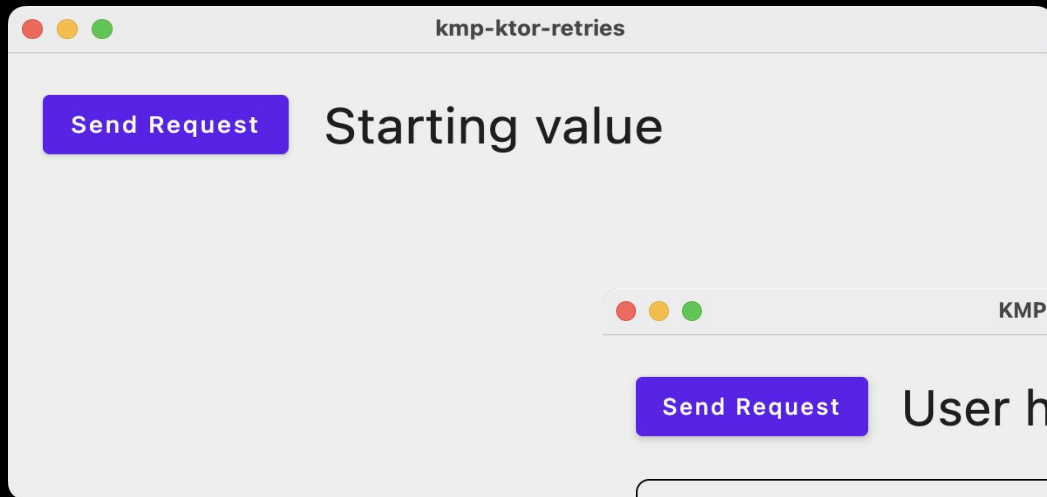
```
fun buildClient(logMessages: MutableList<String>): HttpClient {  
    val client = HttpClient(CIO) {  
        install(ContentNegotiation) {  
            json()  
        }  
        install(HttpRequestRetry) {  
            retryOnServerError(maxRetries = 5)  
            delayMillis { retry →  
                retry * 2000L  
            }  
        }  
    }  
  
    client.plugin(HttpSend).intercept { request →  
        val call = execute(request)  
        val status = call.response.status.value  
        val time = currentTime()  
  
        logMessages.add("Call at $time which returned $status")  
        call  
    }  
    return client  
}
```



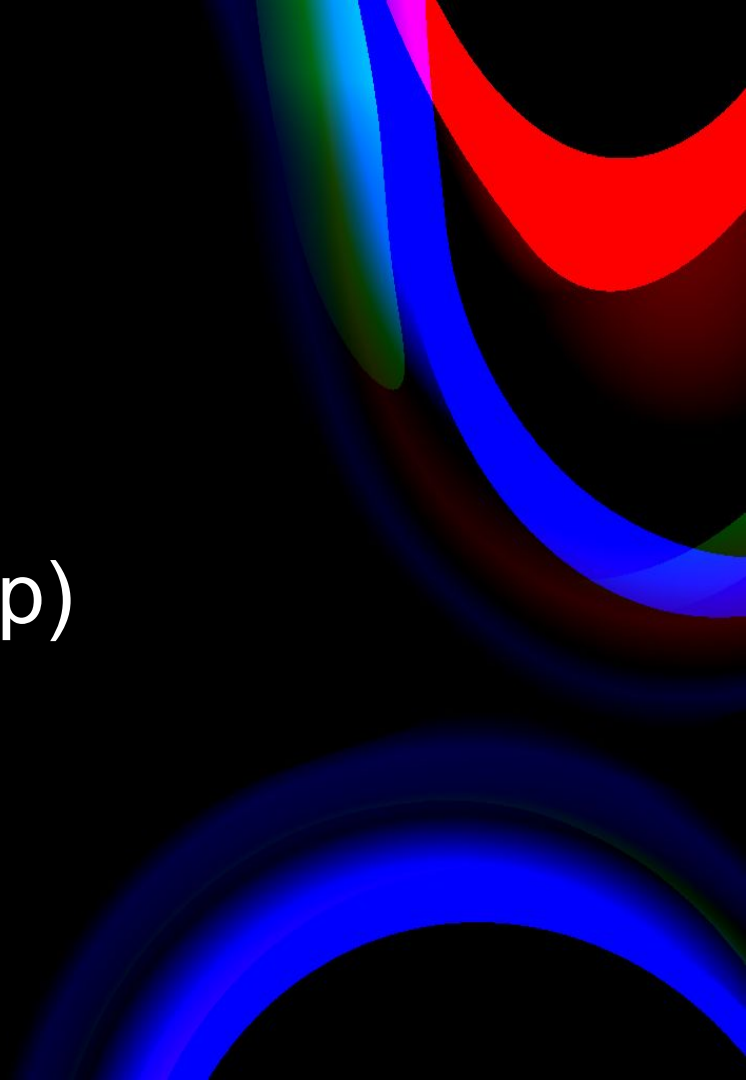
```
suspend fun fetchContent(client: HttpClient): String {  
    val response = client.get("../fetch-questions/2635682")  
    val questions: List<Question> = response.body()  
    return "User has asked ${questions.size} questions"  
}
```

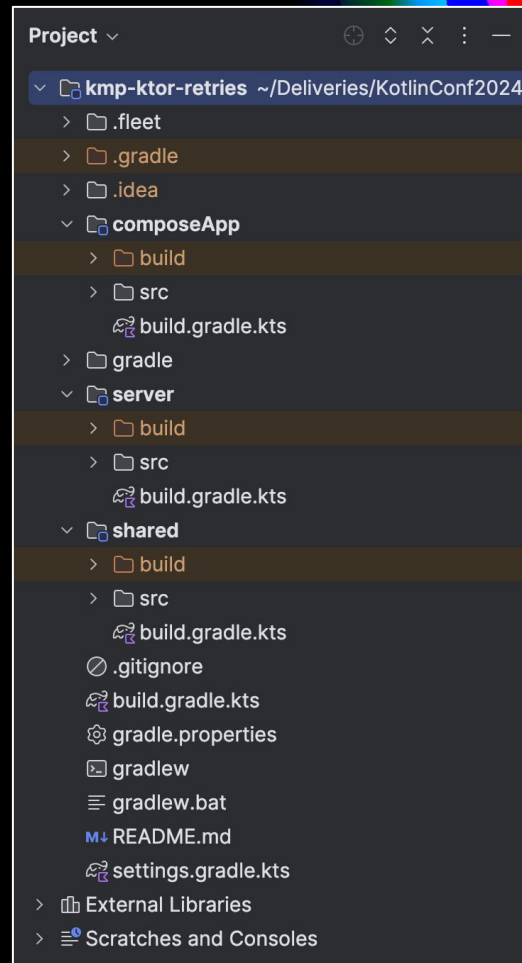
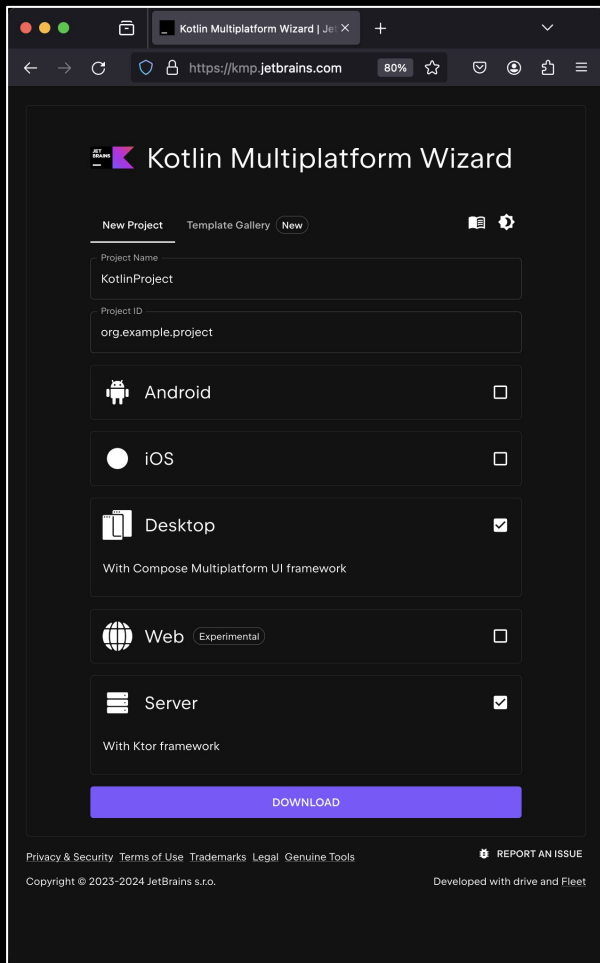
```
val logMessages = remember { mutableListof<String>() }  
  
val client = remember { buildClient(logMessages) }
```

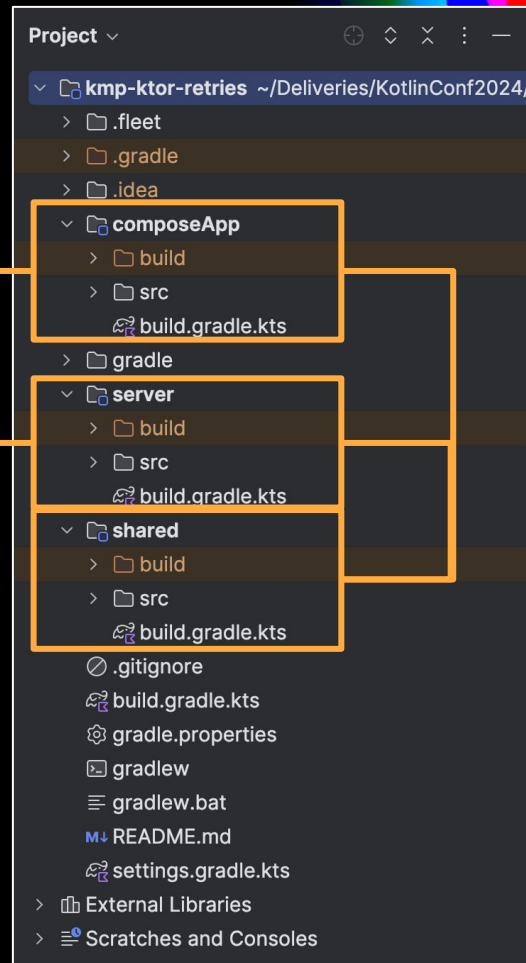
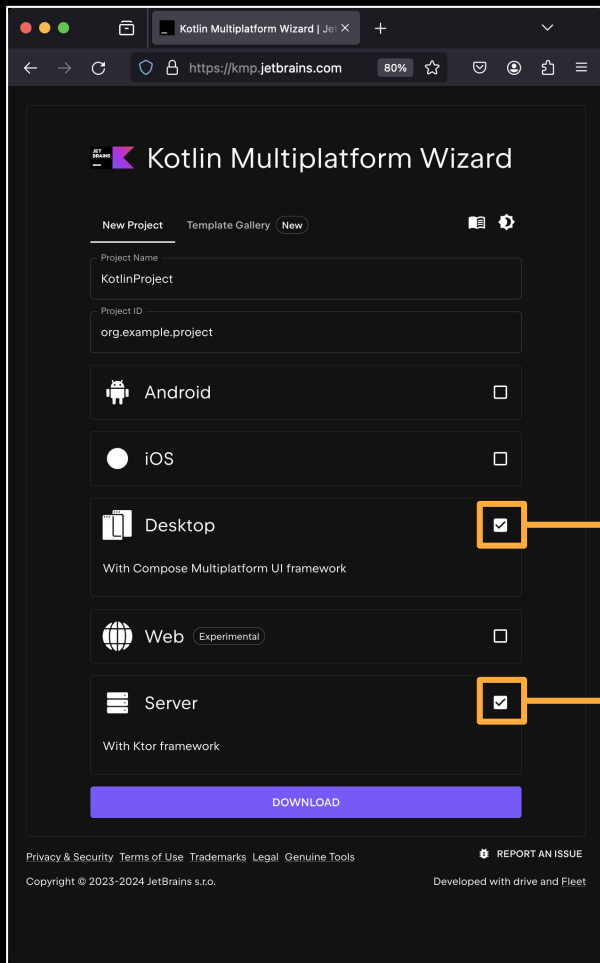
```
Row(verticalAlignment = Alignment.CenterVertically) {  
    Button(onClick = {  
        scope.launch {  
            content.value = fetchContent(client) ←  
        }  
    }) {  
        Text("Send Request")  
    }  
    Text(  
        modifier = Modifier.padding(start = 20.dp),  
        style = TextStyle(fontSize = 30.sp),  
        text = content.value  
    )  
}
```



Retries (Project Setup)







3.0 Features



Additional Ktor 3.0 features

Kotlinx-io replaces bespoke networking

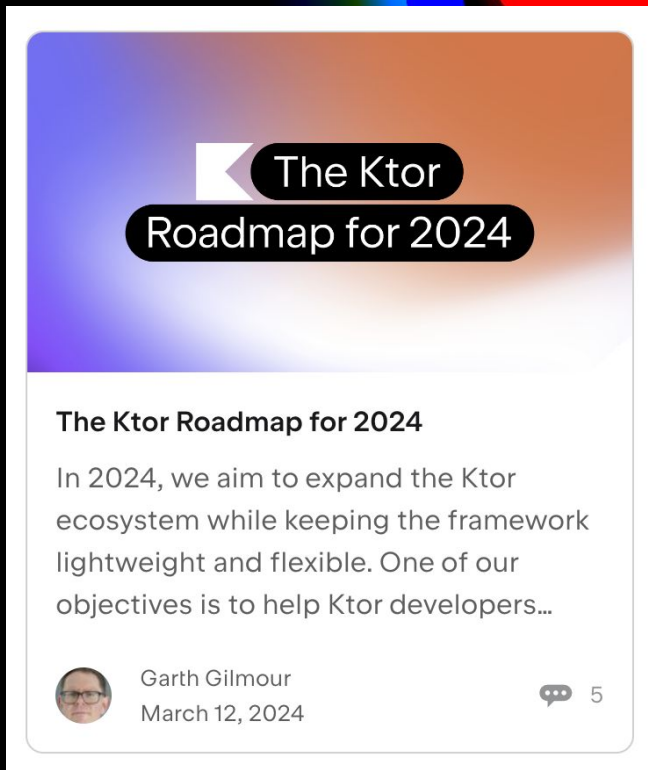
- KMP library for data sources & sinks
- Enhances Ktor integration with KMP

gRPC integrated into Ktor client and server

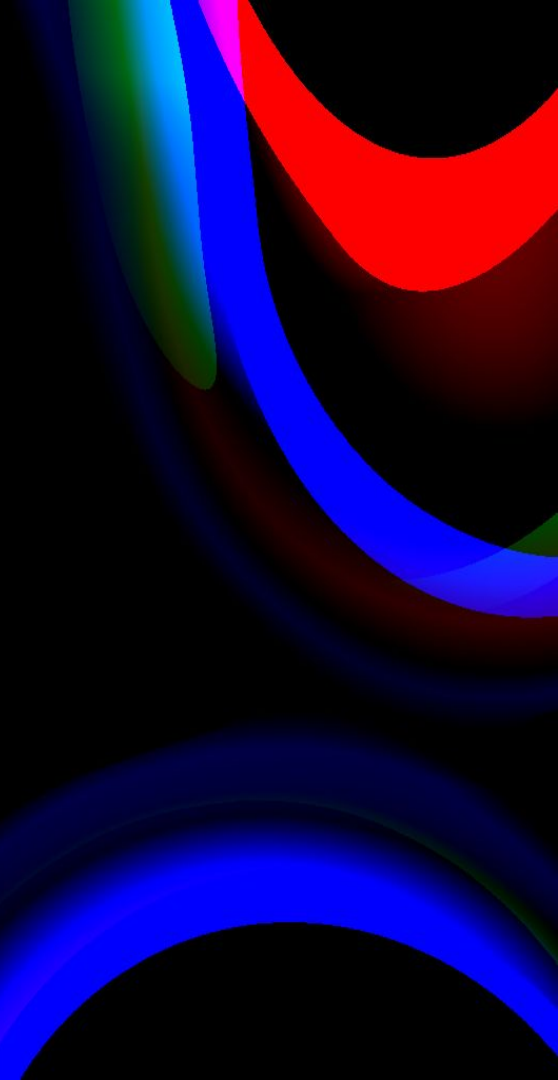
- Via an idiomatic implementation

A plugin for managed transactions

- To simplify DB access in simple cases, when a transaction and request share the same scope



Conclusions

An abstract graphic in the top right corner of the slide. It features several overlapping, curved, ribbon-like shapes in vibrant colors: red, blue, green, and yellow. The lines are thick and have a slight gradient, giving them a three-dimensional appearance. They curve and loop, creating a dynamic and modern look against the black background.

What have we seen?

Lots of shiny things:

- Testing
- DI / Koin
- GraphQL
- WebSockets
- Native execution
- OpenTelemetry
- Fine grained control



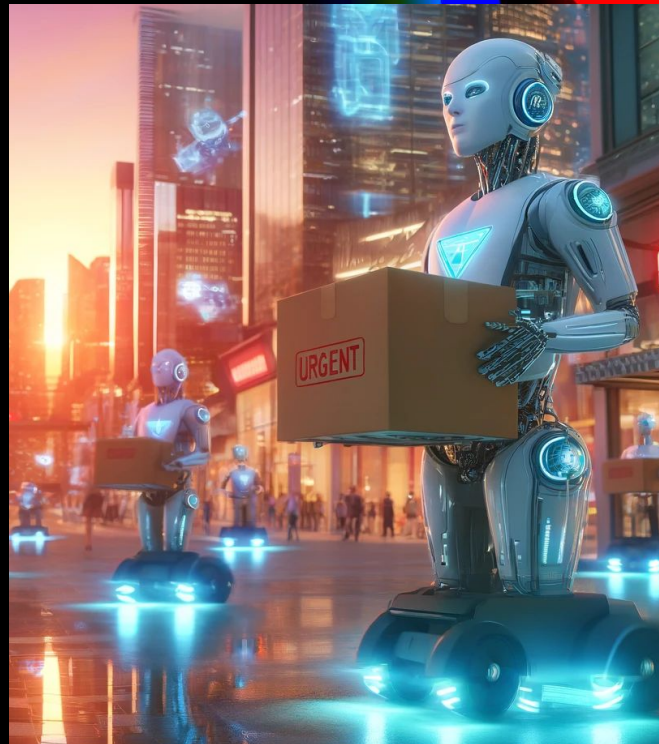
What have we left out?

Database access

- Exposed
- JOOQ
- JPA

Asynchrony everywhere

- Using Coroutines



CHANTAL LONCLE



**Software Developer,
JetBrains**

As a Software Developer in the Exposed team at JetBrains, Chantal has been working on maintaining and enhancing the database access library since early 2023.

Exploring the Exposed Library: A Kotlin Solution to Database Access

Exposed is a database access library built with Kotlin that serves as an abstraction layer providing multiple database connectivity options. In addition to reducing the boilerplate code necessary to interface with a database, the library ensures type-safe SQL and reliable interactions as a seamless extension of a Kotlin developer's code.

This talk ultimately seeks to provide Kotlin developers with a basic understanding of the current state of the Exposed library and its core functionality. We'll introduce the necessary configuration for a database connection, followed by an insight into the concept of transactions and their managers. We'll then examine schema definition, which will lead into an exploration of the different approaches for defining and manipulating relational data: the domain-specific language approach, the data access object pattern, and using plain SQL.

Lastly, to give perspective on the extended capabilities of Exposed, we'll lightly touch on topics geared towards enabling a tailored developer experience, such as: user-defined column types and functions; custom extensions for controlled SQL query building; and statement lifecycle interceptors.

NIKITA KOVAL



Researcher at JetBrains

Nikita is a researcher in the Kotlin team at JetBrains and works on a PhD in the field of concurrency at the same time. His primary research interests are concurrent data structures and algorithms, their verification, and practically applicable code analysis. Here you can find all the recent information about me and my projects.

<https://nikitakoval.org>

Redesigning Communication and Synchronization in Kotlin Coroutines

Have you ever wondered how channels, mutexes, and other synchronization primitives in coroutines work under the hood? In recent years, we have significantly redesigned them; much of the work was also published at the top-tier PPOPP and PLDI academic conferences. In this talk, I will present the high-level design of these core constructs, discussing their performance and implementation trade-offs.

Thanks to our users and contributors

We have seen 3rd party plugins

- For Koin, GraphQL, and OpenTelemetry

This is the future of Ktor!

- We want to build a vibrant ecosystem where all users can become contributors
- The new Ktor Plugin Registry will help

On behalf of the team thank you so much

- For patronage, feedback and contributions



The Ktor Plugin Registry Has Launched

← → ↺ 🔒 https://blog.jetbrains.com/kotlin/2024/04/05/ktor-plugin-registry/ ☆ 📄 📧 📧 📧

All News Releases Multiplatform Ecosystem

Ktor

The Ktor Plugin Registry Has Launched!



Garth Gilmour
April 5, 2024

Read this post in other languages:
[한국어](#), [简体中文](#)

Ktor uses plugins to give you fine-grained control over the functionality included in your application. Until now, third-party plugins were managed differently from those provided by the Ktor team. This was a barrier to growth for the Ktor ecosystem.

Earlier this month, in the [Ktor 2024 product roadmap](#), we announced our Ktor Plugin Registry. This registry is now available. It enables the submission of community-based plugins to the Ktor team, categorizes plugins for easy searching, and provides users with essential documentation.

Introduction

At the moment you can build a Ktor project via [the online project generator](#), picking the best mix of plugins for the task at hand. As shown below you can view

GitHub - ktorio/ktor-plugin-regi

← → ↺ 🔒 https://github.com/ktorio/ktor-plugin-regi/ ☆ 📄 📧 📧 📧

README Code of conduct Apache-2.0 license

Ktor Plugin Registry

This project contains references to all [Ktor](#) plugins available in the web-based [Ktor project generator](#) and the [IDEA plugin](#).

Adding a plugin

To add a new plugin, follow these easy steps:

- Publish your project to Maven.**
 - Use the gradle [maven publish plugin](#) to publish to [Maven Central](#) or some other public Maven repository.
 - We read from repositories listed in [Repositories.kt](#).
 - If you'd like to include another Maven repository for your plugin, include an update to `Repositories.kt` in your pull request in step 2. It should have public read access.
- Fork and clone this repository.**
- Run `./gradlew --console=plain -q createPlugin`
 - This will prompt you with a couple questions about the new plugin.
 - After it is completed, you should have some new files in this

What did we learn?

Ktor is a great choice

- Within a crowded marketplace

Ktor plays the long game

- Sustained progress over fad features

Ktor continues to grow

- In terms of features, community and resources

The team is committed to the community

- Please let us know how to help!



GOTO
Guide



Remember to
rate this session

THANK YOU!



#GOTOams