Rapid Web API development with Kotlin and Ktor

About me

Trayan Iliev

– CEO of **IPT – Intellectual Products & Technologies**
http://www.iproduct.org

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6+, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer: Spring, Java EE, Node.js, Express, GraphQL, SOA, REST, DDD & Reactive Microservices

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

Where to Find The Code and Materials?

https://github.com/iproduct/course-kotlin

Kotlin - A modern programming language with less legacy debts

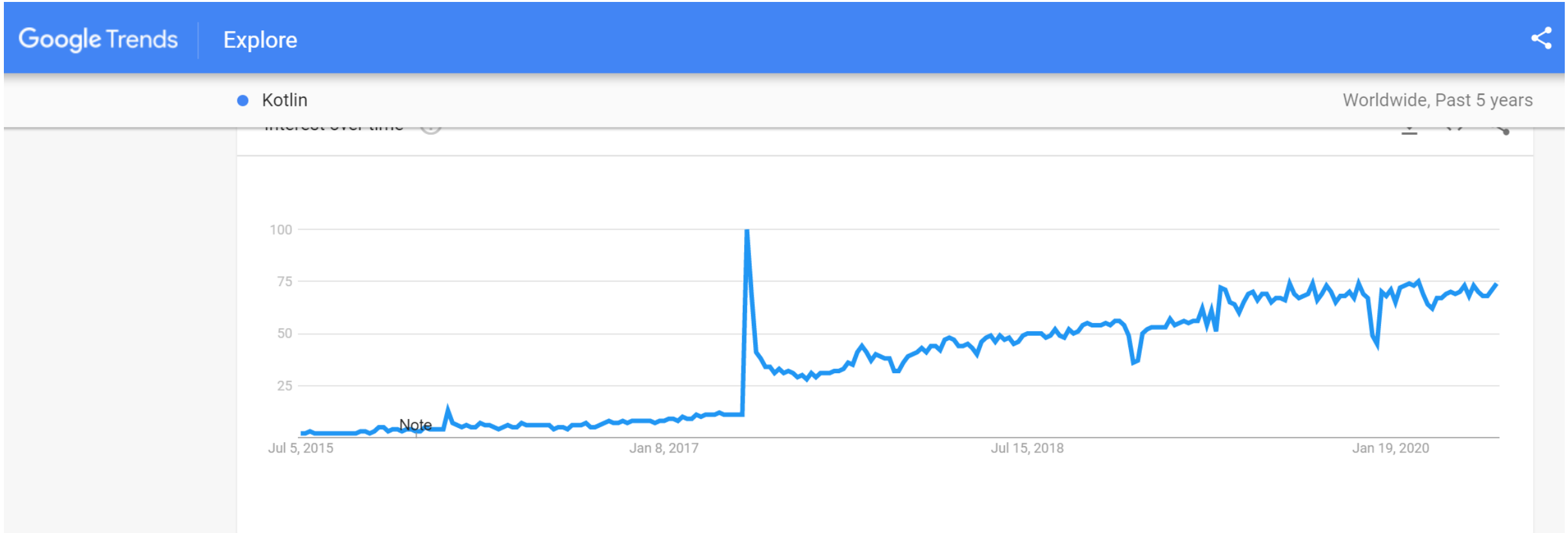
https://kotlinlang.org/

- Kotlin → since July 2011, JetBrains lead Dmitry Jemerov said that most languages did not have the features they were looking for, with the exception of Scala. However, he cited the slow compilation time of Scala as a deficiency.
- Kotlin docs admit how good Java is, but mention that the Java programming language has limitations and problems that are "either impossible or very hard to fix due to backward-compatibility issues."
- Kotlin is a statically typed JVM-targeted language "free of the legacy trouble and having the features so desperately wanted by the developers“, safer and more concise than Java, statically checking for pitfalls such as null pointer dereference, as well as simpler than Scala.

Kotlin Features

- statically typed,
- general-purpose programming language
- type inference - allows more concise syntax than Java
- fully interoperate with Java (the JVM version of its standard library depends on the Java Class Library)
- cross-platform - Kotlin mainly targets the JVM, but also compiles to JavaScript or native code (via LLVM).
- targeting mobile, native (including WebAssembly), web, server-side, data science and cross-platform application development
- open sourced under the Apache 2 license.

Kotlin Popularity According to Google Trends



Source: Google Trends

Some Java Issues Addressed in Kotlin

- Null references are controlled by the type system.
- No raw types
- Arrays in Kotlin are invariant
- Kotlin has proper function types, as opposed to Java's SAM-conversions
- Use-site variance without wildcards
- Kotlin does not have checked exceptions

Source: https://kotlinlang.org/

What Java Has that Kotlin Does Not

- Checked exceptions
- Primitive types that are not classes
- Static members
- Non-private fields
- Wildcard-types
- Ternary-operator a ? b : c

Source: https://kotlinlang.org/

What Kotlin Has that Java Does Not - I

- Lambda expressions + Inline functions = performant custom control structures
- Extension functions
- Null-safety
- Smart casts
- String templates
- Properties
- Primary constructors

Source: https://kotlinlang.org/

What Kotlin Has that Java Does Not - II

- Primary constructors
- First-class delegation
- Type inference for variable and property types
- Singletons
- Declaration-site variance & Type projections
- Range expressions
- Operator overloading
- Companion objects

Source: https://kotlinlang.org/

What Kotlin Has that Java Does Not – III

- Data classes
- Separate interfaces for read-only and mutable collections
- Coroutines !!!

Source: https://kotlinlang.org/

Kotlin Is Consize – Reduces the Boilerplate

```
// Create a POJO with getters, setters, `equals()`, `hashCode()`, `toString()` and `copy()` in a single line:
data class Customer(val name: String, val email: String, val company: String)

// Want a singleton? Create an object:
object ThisIsASingleton {
    val companyName: String = "JetBrains"
}

fun main() {
    // Or filter a list using a lambda expression:
    val positiveNumbers = listOf(0, 1, -1, 2, -2, 3, -3).filter { it > 0 }
    println(positiveNumbers);

    val customerWithCompany = listOf(
        Customer("Trayan Iliev", "office@iproduct.org", "IPT"),
        Customer("John Smith", "john@nobrainer.com", "NoBrainer"),
        Customer("Silvia Popova", "silvia@acme.com", "ACME Corp")
    ).map({"${it.name} - ${it.company}"})  // template strings and single argument lambdas iterator
    println(customerWithCompany); // => [Trayan Iliev - IPT, John Smith - NoBrainer, Silvia Popova - ACME Corp]
}
```

Kotlin Is Safe – Avoids Null Pointer Exceptions

```
// Get rid of those pesky NullPointerExceptions
var output: String
output = null   // Compilation error

// Kotlin protects you from mistakenly operating on nullable types
val name: String? = null    // Nullable type
println(name.length())      // Compilation error
```

Kotlin Is Safe – Automatic Type Casting

```
data class Product(val name: String, val price: Double)
class Order(val number: Int, val products: List<Product>, val date: LocalDateTime = LocalDateTime.now() ){
    fun calculateTotal(): Double {
        return products
            .map { it.price }
            .reduce { acc, prod  ->  acc + prod }
        }
    }

// And if you check a type is right, the compiler will auto-cast it for you
fun calculateTotal(obj: Any): Double? {
    if (obj is Order)  return obj.calculateTotal()
    return 0.0
}

fun main() {
    val products = listOf(Product("Keyboard", 27.5),  Product("Mouse", 17.1))
    val order = Order(1, products)
    println(calculateTotal((order))); // => 44.6
}
```

Kotlin Is Interoperable – JVM, Android

```
import io.reactivex.rxjava3.core.Flowable
import io.reactivex.rxjava3.schedulers.Schedulers
import kotlinx.coroutines.delay
import kotlinx.coroutines.runBlocking

fun main() = runBlocking {
    Flowable
        .fromCallable {
            Thread.sleep(1000) //  imitate expensive computation
            "Done"
        }
        .subscribeOn(Schedulers.io())
        .observeOn(Schedulers.single())
        .subscribe(::println, Throwable::printStackTrace)
    delay(2000)
    println("Demo finished.")
}
```

Kotlin Is Interoperable – … and in The Browser

```
fun getCommonWorldString() = "common-world"
```

```
import io.ktor.samples.fullstack.common.*
import kotlin.browser.*

@Suppress("unused")
@JsName("helloWorld")
fun helloWorld(salutation: String) {
    val message = "$salutation from Kotlin.JS ${getCommonWorldString()}"
    document.getElementById("js-response")?.textContent = message
}

fun main() {
    document.addEventListener("DOMContentLoaded", {
        helloWorld("Hi!")
    })
}
```

Kotlin Is Interoperable – … Fullstack with Ktor as MPP

```
fun Application.main() {
    val currentDir = File(".").absoluteFile
    environment.log.info("Current directory: $currentDir")
    routing {
        get("/") {
            call.respondHtml {
                body {
                    +"Hello ${getCommonWorldString()} from Ktor"
                    div { id = "js-response“+ "Loading..." }
                    script(src = "/static/ktor-samples-fullstack-mpp-frontend.js") {}
}
            }
        }
        static("/static") { resource("ktor-samples-fullstack-mpp-frontend.js") }
}}

fun main(args: Array<String>) {
    embeddedServer(Netty, port = 8080) { main() }.start(wait = true)
}
```

Tool Support for Kotlin

Tool-friendly

Choose any Java IDE or build from the command line

IntelliJ IDEA

Bundled with both IntelliJ IDEA Community Edition and IntelliJ IDEA Ultimate

Download ↗ Instructions

Android Studio

Bundled with Android Studio

Download ↗ Instructions

Eclipse

Install the plugin from the Eclipse Marketplace

Instructions

Standalone Compiler

Use any editor and build from the command line

Instructions

Source: https://kotlinlang.org/

Who Uses Kotlin? – to Mention just a Few

Gradle is introducing Kotlin as a language for writing build scripts

Corda is an open-source distributed ledger platform, supported by major banks, and built entirely in Kotlin

Evernote recently integrated Kotlin into their Android client

Coursera Android app is partially written in Kotlin

Spring makes use of Kotlin's language features to offer more concise APIs

All new code in the Trello Android app is in Kotlin

Source: https://kotlinlang.org/

What is Ktor?

- **Ktor** is an OSS Apache 2 licensed framework for building **asynchronous servers and clients in connected systems** using the power of **Kotlin** programming language.
- Drawing inspiration from other frameworks, such as **Wasabi** and **Kara**, to leverage to the maximum extent the Kotlin language features such as **DSLs** and **coroutines**.
- It is **low-ceremony** in that it requires very **little code and configuration** to get a system up and running.
- Currently, the **Ktor client** works on **all platforms** Kotlin targets, that is, **JVM, JavaScript, and Native**. Right now, **Ktor server-side** is restricted to the **JVM**.

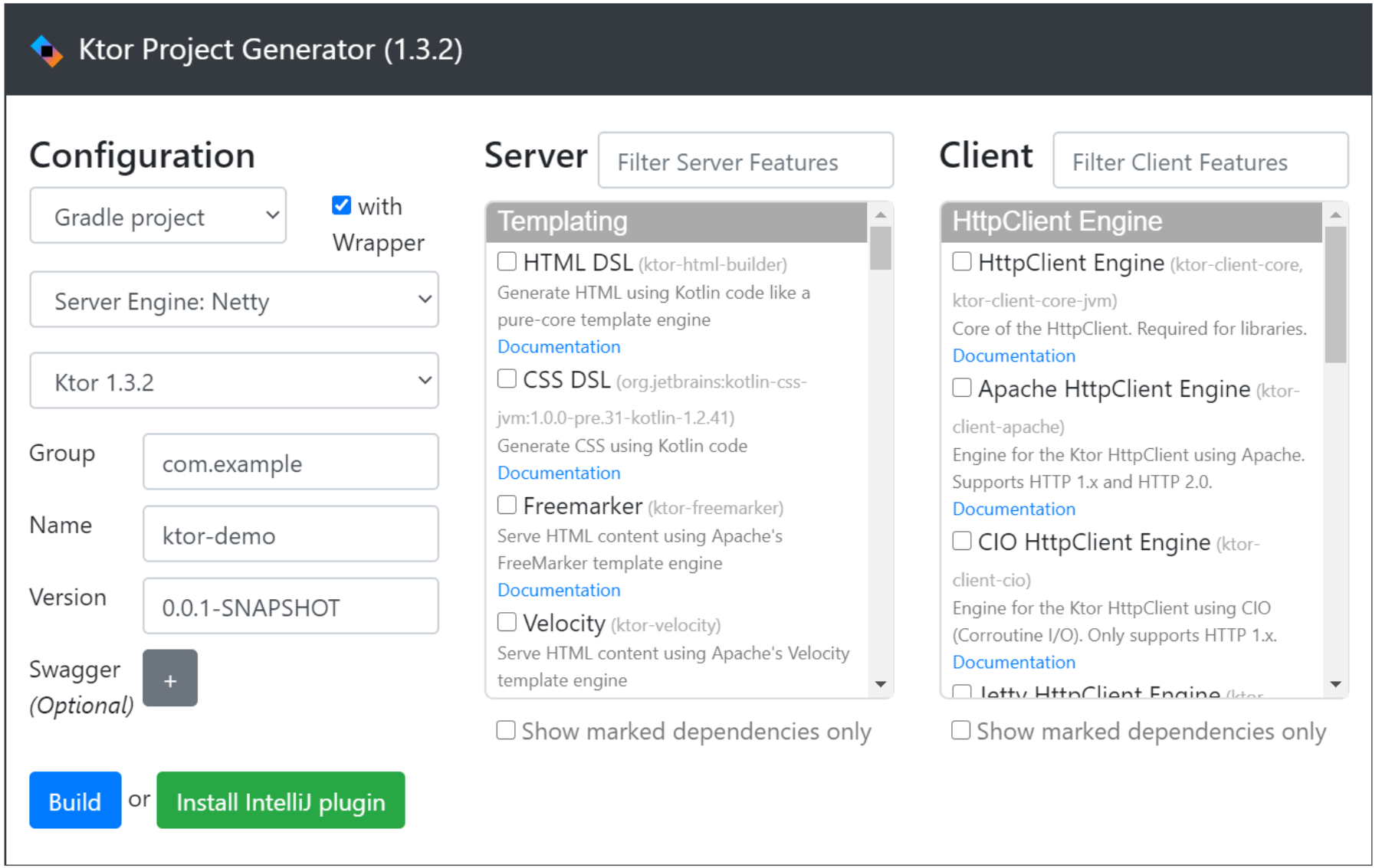
Source: https://kotlinlang.org/

Generate a Ktor project

Estimated reading time: 1 minute

Edit Page

NOTE: You can also use the Ktor IntelliJ plugin instead. This page can also be accessed at start.ktor.io.



Source: https://kotlinlang.org/

Simple Web Service with Ktor

```
fun main() {
    val server = embeddedServer(Netty, 8080) {
        routing {
            get("/hello") {
                call.respondText("<h2>Hello from Ktor and Kotlin!</h2>", ContentType.Text.Html)
            }
        }
    }
    server.start(true)
}
```

… And that's all :)

```
data class Product(val name: String, val price: Double, var id: Int)

object Repo: ConcurrentHashMap<Int, Product>() {
    private idCounter = AtomicInteger()
    fun addProduct(product: Product) {
        product.id = idCounter.incrementAndGet()
        put(product.id, product)
    }
}
fun main() {
    embeddedServer(Netty, 8080, watchPaths = listOf("build/classes"), module= Application::mymodule).start(true)
}
fun Application.mymodule() {
    install(DefaultHeaders)
    install(CORS) {  maxAgeInSeconds = Duration.ofDays(1).toSeconds() }
    install(Compression)
    install(CallLogging)
    install(ContentNegotiation) {
        gson {
            setDateFormat(DateFormat.LONG)
            setPrettyPrinting()
        }
    }
```

```
routing {
    get("/products") {
        call.respond(Repo.values)
    }
    get("/products/{id}") {
        try {
            val item = Repo.get(call.parameters["id"]?.toInt())
            if (item == null) {
                call.respond(
                    HttpStatusCode.NotFound,
                    """{"error":"Product not found with id = ${call.parameters["id"]}"}"""
                )
            } else {
                call.respond(item)
            }
        } catch(ex :NumberFormatException) {
            call.respond(HttpStatusCode.BadRequest,
                """{"error":"Invalid product id: ${call.parameters["id"]}"}""")
        }
    }
```

```
post("/products") {
        errorAware {
            val product: Product = call.receive<Product>(Product::class)
            println("Received Post Request: $product")
            Repo.addProduct(product)
            call.respond(HttpStatusCode.Created, product)
        }
    }
}

private suspend fun <R> PipelineContext<*, ApplicationCall>.errorAware(block: suspend () -> R): R? {
    return try {
        block()
    } catch (e: Exception) {
        call.respondText(
            """{"error":"$e"}""",
            ContentType.parse("application/json"),
            HttpStatusCode.InternalServerError
        )
        null
    }
}
```

Ktor Applications

- **Ktor Server Application** is a custom program **listening to one or more ports** using a **configured server engine**, **composed by modules** with the application logic, that install **features, like routing, sessions, compression**, etc. to handle **HTTP/S 1.x/2.x** and **WebSocket** requests.
- **ApplicationCall** – the context for handling routes, or directly intercepting the pipeline – provides access to two main properties **ApplicationRequest** and **ApplicationResponse** ,as well as **request parameters, attributes, authentication, session, typesafe locations**, and the **application** itself. Example:

```
intercept(ApplicationCallPipeline.Call) {
    if (call.request.uri == "/")
        call.respondHtml {
            body {
                    a(href = "/products") { + "Go to /products" }
} } }
```

Routing DSL Using Higher Order Functions

- **routing**, **get**, and **post** are all **higher-order functions** (functions that take other functions as parameters or return functions).
- Kotlin has a convention that **if the last parameter to a function is another function**, we can place this **outside of the brackets**
- **routing** is a **lambda with receiver** == higher-order function taking as parameter an **extension function** => anything enclosed within **routing** has access to members of the type **Routing**.
- **get** and **post** are functions of the **Routing** type => also **lambdas with receivers**, with own members, such as **call**.
- This combination of **conventions** and **functions** allows to create elegant DSLs, such as Ktor's **routing DSL**.

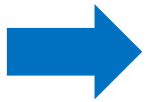
Features

- A **feature** is a **singleton** (usually a **companion object**) that you can **install and configure for a pipeline**.
- Ktor includes some **standard features**, but you **can add your own** or other features from the community.
- You can install features in **any pipeline**, like the **application** itself, or specific **routes**.
- Features are **injected into the request and response pipeline**. Usually, an application would have a series of features such as **DefaultHeaders** which add headers to every outgoing response, **Routing** which allows us to define routes to handle requests, etc.

Installing Features

Using *install*:

```
fun Application.main() {
    install(DefaultHeaders)
    install(CallLogging)
    install(Routing) {
        get("/") {
            call.respondText("Hello, World!")
        }
    }
}
```



Using routing DSL:

```
fun Application.main() {
    install(DefaultHeaders)
    install(CallLogging)
    routing {
        get("/") {
            call.respondText("Hello, World!")
        }
    }
}
```

Standard Features I

- Authenticating Clients
- Enable Automatic HEAD Responses
- Basic and Form authentication
- Controlling cache headers
- CallId
- Log the client requests
- Client/Server Sessions
- Enable HTTP Compression Facilities

Standard Features - II

- Easy '304 Not Modified' Responses
- Content conversion based on Content-Type and Accept headers
- Cookie/Header Sessions
- Enable Cross-Origin Resource Sharing (CORS)
- Data Conversion
- Send Headers Automatically
- Digest authentication
- DoubleReceive for request body

Standard Features - III

- XForwardedHeaderSupport (Reverse Proxy Support)
- Using Freemarker Templates
- JSON support using Gson
- Enable HTTP Strict Transport Security
- Emit HTML with a DSL
- Redirect HTTP requests to HTTPS
- JSON support using Jackson
- JWT and JWK authentication
- LDAP authentication

Standard Features - IV

- Type-safe Routing
- Metrics with Micrometer metrics
- Metrics with Dropwizard metrics
- Using Mustache Templates
- OAuth authentication
- Streaming Movies and Other Content
- Using Pebble Templates
- Structured Handling of HTTP Requests
- JSON support using kotlinx.serialization

Standard Features - V

- Handle Conversations with Sessions
- Add an URL for shutting down the server
- Serving Static Content
- Handle Exceptions and Customize Status Pages
- Session Storages
- Templates, Using Thymeleaf Templates, Using Velocity Templates
- Session Transformers
- Webjars support
- WebSockets

Ktor Application Modules - I

- **Ktor module** is a user-defined **function receiving the Application class** that is in charge of **configuring the server pipeline**, **install features**, **registering routes**, **handling requests**, etc. (e.g. com.example.ApplicationKt.module)

- Example:

```
fun main(args: Array<String>): Unit = io.ktor.server.netty.EngineMain.main(args)


fun Application.module(testing: Boolean = false) {
    install(DefaultHeaders)
    install(CallLogging)
    routing {
        get("/") {
            call.respondText("Hello, Ktor World!")
        }
    }
}
```

Ktor Application Modules - II

- You have to specify the modules to load when the server starts in the **application.conf** file (in HOCON - Human-Optimized Config Object Notation):

```
ktor {
    deployment {
        port = 8080
        port = ${?PORT}
        watch = [ "build/classes" ]
    }
    application {
        modules = [ com.example.ApplicationKt.module ]
    }
}
```

- Specifying the **watch** path allows **auto reloading** (same as **watchPaths** parameter of the **embeddedServer**) + **gradlew -t installDist**

Ktor Server Lifecycle – Starting Ktor

- Entry points:
  - With a plain main by calling **embeddedServer**
  - Running a **EngineMain** main function and using a HOCON **application.conf** configuration file
  - As a **Servlet** within a web server
  - As part of a test using **withTestApplication** from the **ktor-server-test-host** artifact
- There are multiple **ApplicationEngine**s, like: **Netty**, **Jetty**, **CIO** or **Tomcat**.
- **embeddedServer** - when you run your own **main** method and call it, you provide a specific **ApplicationEngineFactory**.
- **EngineMain**:
  - CIO: io.ktor.server.cio.EngineMain.main
  - Jetty: io.ktor.server.jetty.EngineMain.main
  - Netty: io.ktor.server.netty.EngineMain.main
  - Tomcat: io.ktor.server.tomcat.EngineMain.main

Testing Ktor APIs using TestApplicationEngine - I

```
class ServerTest {


    @Test fun `login should succeed with token`() = withServer {
        val req = handleRequest {
            method = HttpMethod.Post
            uri = "/login"
            addHeader("Content-Type", "application/json")
            setBody(
                Gson().toJson(UserPasswordCredential("user", "pass"))
            )
        }

        req.requestHandled shouldBe true
        req.response.status() shouldEqual HttpStatusCode.OK
        req.response.content.shouldNotBeNullOrBlank().length shouldBeGreaterThan 6
    }
```

Testing Ktor APIs using TestApplicationEngine - II

```
@Test fun `request without token should fail`() = withServer {
    val req = handleRequest {
        uri = "/secret"
    }
    req.requestHandled shouldBe true
    req.response.status() shouldEqual HttpStatusCode.Unauthorized
}

@Test fun `request with token should pass`() = withServer {
    val req = handleRequest {
        uri = "/secret"
        addJwtHeader()
    }
    req.requestHandled shouldBe true
    req.response.let {
        it.status() shouldEqual HttpStatusCode.OK
        it.content.shouldNotBeNullOrBlank()
    }
}
```

Testing Ktor APIs using TestApplicationEngine - III

```
    @Test fun `optional route should work with token`() = withServer {
        val req = handleRequest {
            uri = "/optional"
            addJwtHeader()
        }
        req.let {
            it.requestHandled.shouldBeTrue()
            it.response.status() shouldEqual HttpStatusCode.OK
            it.response.content.shouldNotBeNullOrBlank() shouldBeEqualTo "authenticated!"
        }
    }

    private fun TestApplicationRequest.addJwtHeader() = addHeader("Authorization", "Bearer ${getToken()}")

    private fun getToken() = JwtConfig.makeToken(testUser)

    private fun withServer(block: TestApplicationEngine.() -> Unit) {
        withTestApplication({ module() }, block)
    }
}
```

Ktor Server Lifecycle - Monitoring Events

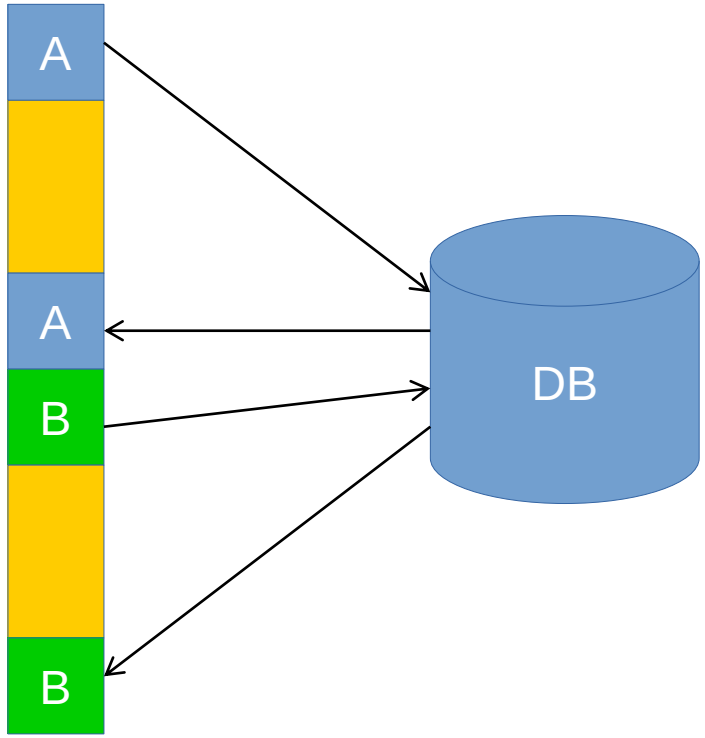
- Ktor defined events:
  - val ApplicationStarting = EventDefinition<Application>()
  - val ApplicationStarted = EventDefinition<Application>()
  - val ApplicationStopPreparing = EventDefinition<ApplicationEnvironment>()
  - val ApplicationStopping = EventDefinition<Application>()
  - val ApplicationStopped = EventDefinition<Application>()

- Example:

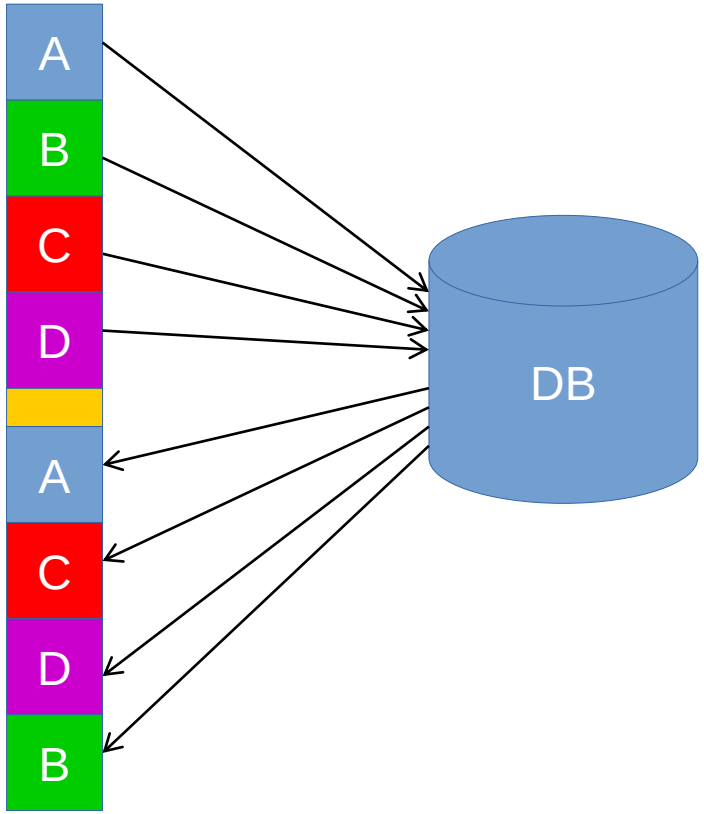
```
val starting: (Application) -> Unit = { LOG.info("Application starting: $it") }
environment.monitor.subscribe(ApplicationStarting, starting) // subscribe
environment.monitor.unsubscribe(ApplicationStarting, starting) // unsubscribe
```

Synchronous vs. Asynchronous IO

Synchronous



Asynchronous



42

Suspendable Functions (Coroutines) and async-await in Kotlin

```
fun main() {
    val time = measureTimeMillis {
        runBlocking {
            val one = async { doSomethingUsefulOne() }
            val two = async { doSomethingUsefulTwo() }
            println("The answer is ${one.await() + two.await()}") // => 42
        }
    }
    println("Completed in $time ms") // => Completed in 1022 ms
}

private suspend fun doSomethingUsefulOne(): Int {
    delay(1000L) // pretend we are doing something useful here
    return 24
}

private suspend fun doSomethingUsefulTwo(): Int {
    delay(1000L) // pretend we are doing other useful thing here also
    return 18
}
```

Structured Concurrency with coroutineScope

```
fun main() = runBlocking {
    val time = measureTimeMillis {
        println(compute())
    }
    println("Completed in $time ms")
}

suspend fun compute() = coroutineScope {
    val one = async { doSomethingUsefulOne() }
    val two = async { doSomethingUsefulTwo() }
    "The answer is ${one.await() + two.await()}"
}
private suspend fun doSomethingUsefulOne(): Int {
    delay(1000L) // pretend we are doing something useful here
    return 24
}
private suspend fun doSomethingUsefulTwo(): Int {
    delay(1000L) // pretend we are doing other useful thing here also
    return 18
}
```

Structured Concurrency using Async Functions (Deferred)

```
fun main() {
    val time = measureTimeMillis {
        val one = somethingUsefulOneAsync() // we can initiate async actions outside of a coroutine
        val two = somethingUsefulTwoAsync()
        // but waiting for a result must involve either suspending or blocking.
        // here we use `runBlocking { ... }` to block the main thread while waiting for the result
        runBlocking {
            println("The answer is ${one.await() + two.await()}")
        }
    }
    println("Completed in $time ms")
}
fun somethingUsefulOneAsync() = GlobalScope.async {  // The result type is Deferred<Int>
    delay(1000L) // pretend we are doing something useful here
    24
}
fun somethingUsefulTwoAsync() = GlobalScope.async {  // The result type is Deferred<Int>
    delay(1000L) // pretend we are doing something useful here, too
    18
}
```

Structured Concurrency: Parent Task Gets Canceled on Child Failure

```
fun main() = runBlocking<Unit> {
    try {
        failedConcurrentSum()
    } catch(e: ArithmeticException) {
        println("Computation failed with ArithmeticException")
    }
}
suspend fun failedConcurrentSum(): Int = coroutineScope {
    val one = async<Int> {
        try {
            delay(Long.MAX_VALUE) // Emulates expensive computation
            42
        } finally {
            println("First child was cancelled")
        }
    }
    val two = async<Int> {
        println("Second child throws an exception")
        throw ArithmeticException()
    }
    one.await() + two.await()
}
```

Channels

```
fun main() = runBlocking {

    val channel = Channel<Int>()
    launch {
        for (x in 1..5) {
            delay(1000)
            channel.send(x * x)
        }
       channel.close() // we're done sending
    }

    // here we print received values using `for` loop (until the channel is closed)
    for (y in channel) println(y)
    println("Done!")
}

// => 1, 4, 9, 16, 25, Done!
```

Pipelines

```
fun main() = runBlocking {
    fun CoroutineScope.produceNumbers(limit: Int) = produce<Int> {
        var x = 1
        while (x <= limit) send(x++) // infinite stream of integers starting from 1
    }
    fun CoroutineScope.square(numbers: ReceiveChannel<Int>): ReceiveChannel<Int> = produce {
        for (x in numbers) send(x * x)
    }


    // here we print received values using `for` loop (until the channel is closed)
    for (y in square(produceNumbers(5))) println(y)
    println("Done!")
}

// => 1, 4, 9, 16, 25, Done!
```

Ktor Server Lifecycle - Pipelines

- *Ktor defines pipelines for **asynchronous extensible computations**.*
- *All the pipelines have an associated **subject type**, **context type**, and a **list of phases** with **interceptors** associated to them. As well as, **attributes** that act as a small typed object container.*
- ***Phases** are ordered and can be defined to be executed, after or before another phase, or at the end.*
- *Each pipeline has an **ordered list of phase contexts** for that **instance**, which contain a set of interceptors for each phase:*

Pipeline: Phase1(Interceptor1, Interceptor2) => Phase2(Interceptor3, Interc4)

- *Each interceptor for a specific phase does not depend on other interceptors on the same phase, but on interceptors from previous phases.*
- *All registered interceptors are executed **in the order defined by the phases**.*

ApplicationCallPipeline

- *The server part of Ktor defines an **ApplicationCallPipeline** without a subject and with **ApplicationCall** as context.*
- *The **Application** instance is an **ApplicationCallPipeline**.*
- *When server handles a **HTTP request**, it will execute the **Application pipeline**.*
- *The context class **ApplicationCall** contains the application, the request, the response, and the attributes and parameters.*
- *In the end, the **application modules**, will end **registering interceptors for specific phases** for the **Application pipeline**, to process the **request** and emitting a **response**.*
- *The ApplicationCallPipeline defines the following built-in phases for its pipeline:*

ApplicationCallPipeline Built-in Phases

- Setup: phase used for preparing the call and its attributes for processing (like the CallId feature)
- Monitoring: phase used for tracing calls: useful for logging, metrics, error handling and so on (like the CallLogging feature)
- Features: most features should intercept this phase (like the Authentication).
- Call: features and interceptors used to complete the call, like the Routing
- Fallback: features that process unhandled calls in a normal way and resolve them somehow, like the StatusPages feature

Ktor Pipelines in Depth

```
class PipelinePhase(val name: String)
class Pipeline <TSubject : Any, TContext : Any> {
    constructor(vararg phases: PipelinePhase)

    val attributes: Attributes

    fun addPhase(phase: PipelinePhase)
    fun insertPhaseAfter(reference: PipelinePhase, phase: PipelinePhase)
    fun insertPhaseBefore(reference: PipelinePhase, phase: PipelinePhase)

    fun intercept(phase: PipelinePhase, block: suspend PipelineContext.(TSubject) -> Unit)

    fun merge(from: Pipeline)

    suspend fun execute(context: TContext, subject: TSubject): TSubject
}
```

Adding Custom Phases and Interceptors

```
val phase1 = PipelinePhase("MyPhase1")
val phase2 = PipelinePhase("MyPhase2")
pipeline.insertPhaseAfter(ApplicationCallPipeline.Features, phase1)
pipeline.insertPhaseAfter(phase1, phase2)
```

- Then you can intercept phases, so your interceptors will be called in that phase in the order they are registered:

```
pipeline.intercept(phase1) { println("Phase1[A]") }
pipeline.intercept(phase2) { println("Phase2[A]") }
pipeline.intercept(phase2) { println("Phase2[B]") }
pipeline.intercept(phase1) { println("Phase1[B]") }

pipeline.execute(context, subject)

// => Phase1[A] Phase1[B] Phase2[A] Phase2[B]
```

Interceptors and the PipelineContext

```
class PipelineContext<TSubject : Any, out TContext : Any>() {
    val context: TContext
    val subject: TSubject

    fun finish()
    suspend fun proceedWith(subject: TSubject): TSubject
    suspend fun proceed(): TSubject
}
```

The subject is accessible from the context as a property with its own name, and it is propagated between interceptors. You can change the instance (for example for immutable subjects) using the method:

PipelineContext.proceedWith(subject)

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD