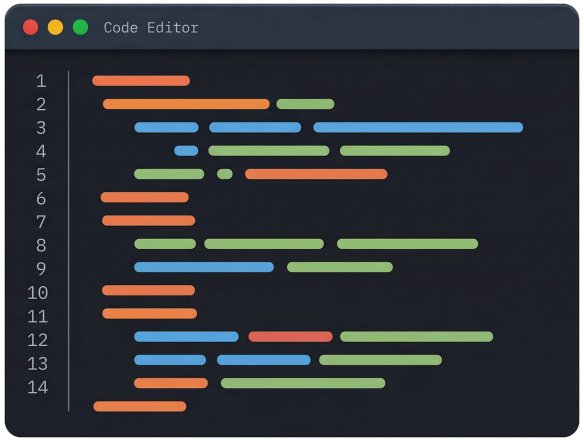
Idiomatic Kotlin applications with Spring Boot 4

Sébastien Deleuze
Spring Framework / AI Committer
Tanzu Division, Broadcom
https://seb.deleuze.fr

Does idiomatic code still matters in the age of coding agents?

Entropy, maintenance, performance, conciseness ...

Vibe coded without review



Idiomatic

Turn those best practices into agent skills

Specification

Copy page

The complete format specification for Agent Skills.

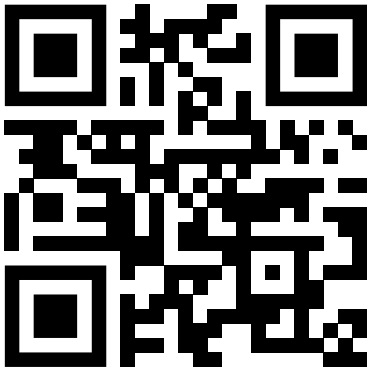
Directory structure

A skill is a directory containing, at minimum, a `SKILL.md` file:

```
skill-name/
├── SKILL.md          # Required: metadata + instructions
├── scripts/          # Optional: executable code
├── references/       # Optional: documentation
├── assets/           # Optional: templates, resources
└── ...               # Any additional files or directories
```

`SKILL.md` format

The `SKILL.md` file must contain YAML frontmatter followed by Markdown content.



Build system

Gradle Kotlin DSL

Project

- ○ Gradle - Groovy
- ● Gradle - Kotlin
- ○ Maven

Language

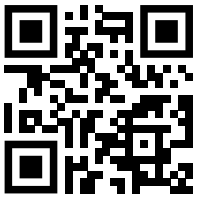
- ○ Java
- ● Kotlin
- ○ Groovy

build.gradle.kts

```
plugins {
  kotlin("jvm") version "2.3.20"
  kotlin("plugin.spring") version "2.3.20"
  id("org.springframework.boot") version "4.1.0"
  id("io.spring.dependency-management") version "1.1.7"
}

group = "com.example"
version = "0.0.1-SNAPSHOT"

java {
  toolchain {
    languageVersion = JavaLanguageVersion.of(25)
  }
}
...
```

Next generation declarative build tools are coming

Amper



Declarative Gradle

module.yaml

```yaml
product: jvm/app

settings:
  springBoot: enabled
  jvm:
    mainClass: com.example.MyApplicationKt


dependencies:
  - $spring.boot.starter.actuator
  - $spring.boot.starter.cache
```

build.gradle.dcl

```
kotlinJvmApplication {
    javaVersion = 25
    mainClass = "com.example.MyApplication.kt"

    dependencies {
        implementation("...:spring-boot-starter-actuator")
        implementation("...:spring-boot-starter-cache")
    }
}
```

Spring Boot 4

Spring Boot 4 brings many new features

Date	Title
September 9, 2025	Core Spring Resilience Features: `@ConcurrencyLimit`, `@Retryable`, and `RetryTemplate`
September 16, 2025	API Versioning in Spring
September 23, 2025	HTTP Service Client Enhancements
September 30, 2025	The state of HTTP clients in Spring
October 7, 2025	Introducing Jackson 3 support in Spring
October 14, 2025	Share Consumer - aka Kafka Queues in Spring Kafka
October 21, 2025	Multi-Factor Authentication in Spring Security
October 28, 2025	Modularizing Spring Boot
November 5, 2025	Spring gRPC - Next Steps
November 12, 2025	Null-safe applications with Spring Boot 4
November 18, 2025	OpenTelemetry with Spring Boot
November 25, 2025	Ahead of Time Repositories (Part 2)

The Road to GA

Kotlin 2 baseline and K2 compiler

2.2

The new baseline

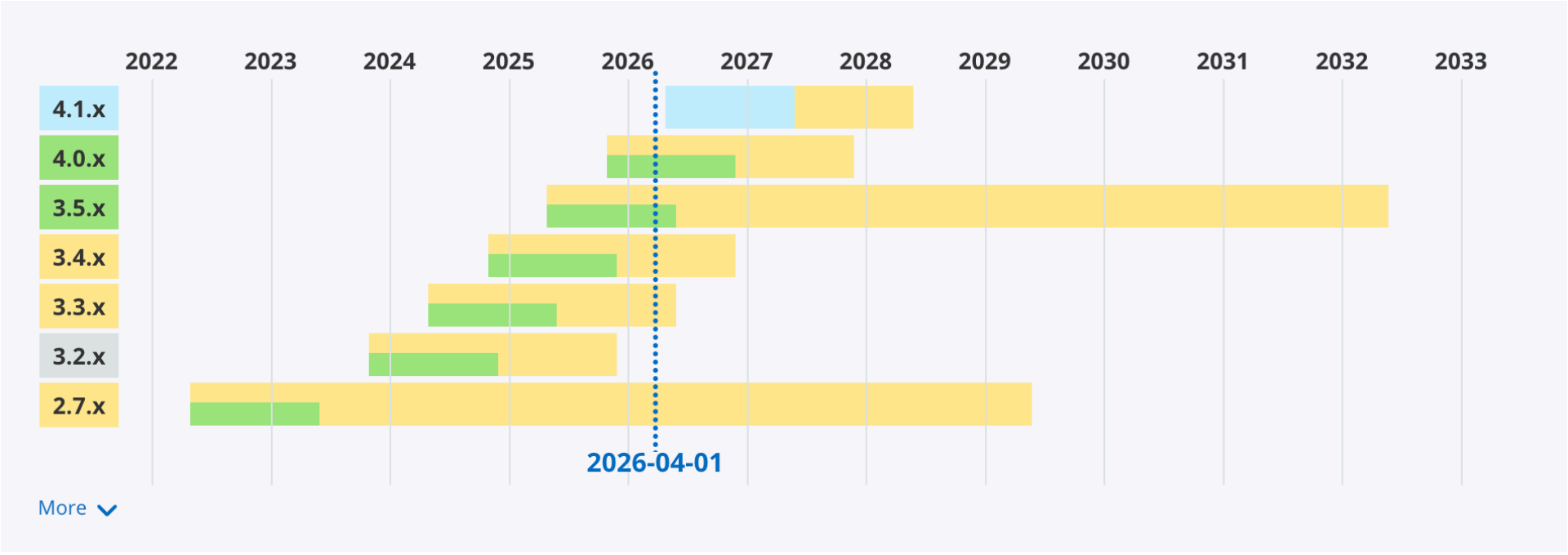
Benefits

- Faster compilation
- Better code analysis
- Java interoperability
- Multiplatform

Recommandation

Spring Boot	Kotlin
4.0	2.2
4.1	2.3
4.2	2.4

Upgrade to Spring Boot 4 for OSS support



OSS support

Enterprise support

Future release

Use the new annotation default targets

❌ Historical default annotation target

```kotlin
data class User(
    val username: String,
    @Email val email: String
)
```

```java
public final class User {

    private final String username;
    private final String email;

    public User(String username,
                @Email String email) {
        super();
        this.username = username;
        this.email = email;
    }

    // ...
}
```

⚠️ Specify an annotation use-site target

```kotlin
data class User(
    val username: String,
    @field:Email val email: String
)
```

```java
public final class User {

    private final String username;
    @Email
    private final String email;

    public User(String username,
                String email) {
        super();
        this.username = username;
        this.email = email;
    }

    // ...
}
```

✅ Kotlin 2.2+ with -Xannotation-default-target=param-property

```kotlin
data class User(
    val username: String,
    @Email val email: String
)
```

```java
public final class User {

   private final String username;
   @Email
   private final String email;

   public User(String username,
               @Email String email) {
      super();
      this.username = username;
      this.email = email;
   }

   // ...
}
```

✅ Kotlin 2.4+ default annotation target

```kotlin
data class User(
    val username: String,
    @Email val email: String
)
```

```java
public final class User {

    private final String username;
    @Email
    private final String email;

    public User(String username,
                @Email String email) {
        super();
        this.username = username;
        this.email = email;
    }

    // ...
}
```

Take advantage of Spring null-safe APIs

Spring Boot 3 with Spring null-safety annotation

Projects with null-safe APIs

- Spring Framework
- Spring Data

Features

✅ Package default to non-null

✅ Method-level null-safety

✅ Field-level null-safety

❌ Checks during compiler’s type analysis

❌ OOTB Kotlin null-safety translation

❌ Proper specification / dependency

❌ Usable outside Spring

❌ Generic type nullability

Spring Boot 4 with JSpecify

Projects with null-safe APIs

- Spring Framework
- Spring Data
- Spring Boot
- Spring Security
- Spring AI
- Spring Batch
- Spring Kafka
- Spring Integration
- Spring GraphQL
- Spring Web Services
- Spring AMQP
- Spring Shell
- Spring HATEOAS
- Spring Modulith
- Spring Cloud (partial)
- Micrometer
- Reactor
- …

Features

✅ Package default to non-null

✅ Method-level null-safety

✅ Field-level null-safety

✅ Checks during compiler's type analysis

✅ OOTB Kotlin null-safety translation

✅ Proper specification / dependency

✅ Usable outside Spring

✅ Generic type nullability

Spring Boot 3

```
@SpringBootApplication
open class Boot3KotlinApplication

fun main(args: Array<String>) {
    val app = SpringApplication(Boot3KotlinApplication::class.java)
    app.
}
```

```
additionalProfiles (from getAdditionalProfiles())                    kotlin.collections.(Mutable)Set<String!>!
allSources (from getAllSources())                                    kotlin.collections.(Mutable)Set<Any!>!
applicationStartup (from getApplicationStartup()/setApplicationStartup())    ApplicationStartup!
classLoader (from getClassLoader())                                  ClassLoader!
environmentPrefix (from getEnvironmentPrefix()/setEnvironmentPrefix())    String!
initializers (from getInitializers())  kotlin.collections.(Mutable)Set<ApplicationContextInitializer<*>!>!
isKeepAlive (from isKeepAlive()/setKeepAlive())                      Boolean
listeners (from getListeners())                  kotlin.collections.(Mutable)Set<ApplicationListener<*>!>!
mainApplicationClass (from getMainApplicationClass()/setMainApplicationClass())    Class<*>!
resourceLoader (from getResourceLoader()/setResourceLoader())        ResourceLoader!
sources (from getSources()/setSources())                             kotlin.collections.(Mutable)Set<String!>!
webApplicationType (from getWebApplicationType()/setWebApplicationType())    WebApplicationType!

Press ↵ to insert, ⇥ to replace
```

Spring Boot 4

```
@SpringBootApplication
open class Boot4KotlinApplication

fun main(args: Array<String>) {
    val app = SpringApplication(Boot4KotlinApplication::class.java)
    app.
}
```

```
additionalProfiles (from getAdditionalProfiles())          kotlin.collections.(Mutable)Set<String>
allSources (from getAllSources())                          kotlin.collections.(Mutable)Set<Any>
applicationStartup (from getApplicationStartup()/setApplicationStartup())   ApplicationStartup
classLoader (from getClassLoader())                                         ClassLoader
environmentPrefix (from getEnvironmentPrefix())                             String?
initializers (from getInitializers())   kotlin.collections.(Mutable)Set<ApplicationContextInitializer<*>>
isKeepAlive (from isKeepAlive()/setKeepAlive())                             Boolean
listeners (from getListeners())                kotlin.collections.(Mutable)Set<ApplicationListener<*>>
mainApplicationClass (from getMainApplicationClass()/setMainApplicationClass())   Class<*>?
resourceLoader (from getResourceLoader())                                   ResourceLoader?
sources (from getSources()/setSources())                   kotlin.collections.(Mutable)Set<String>
webApplicationType (from getWebApplicationType())                           WebApplicationType?
Press ⏎ to insert, ⇥ to replace
```

Package are null-marked

Package org.springframework.boot

```
@NullMarked
package org.springframework.boot
```

Core Spring Boot classes.

See Also:

`SpringApplication`

All Classes and Interfaces | Interfaces | Classes | Enum Classes | Exception Classes | Annotation Interfaces

Class	Description
AotInitializerNotFoundException	Exception thrown when the AOT initializer couldn't be found.
ApplicationArguments	Provides access to the arguments that were used to run a `SpringApplication`.
ApplicationContextFactory	Strategy interface for creating the `ConfigurableApplicationContext` used by a `SpringApplication`.
ApplicationRunner	Interface used to indicate that a bean should *run* when it is contained within a `SpringApplication`.

By default, type usages are non-null

refresh

```
protected void refresh(ConfigurableApplicationContext applicationContext)
```

Refresh the underlying ApplicationContext.

Parameters:

`applicationContext` - the application context to refresh

getApplicationStartup

```
public ApplicationStartup getApplicationStartup()
```

Returns the ApplicationStartup used for collecting startup metrics.

Returns:

the application startup

Since:

2.4.0

Nullable type usages are specified explicitly

getWebApplicationType

```
public @Nullable WebApplicationType getWebApplicationType()
```

Returns the type of web application that is being run.

Returns:

the type of web application

Since:

2.0.0

setEnvironment

```
public void setEnvironment(@Nullable ConfigurableEnvironment environment)
```

Sets the underlying environment that should be used with the created application context.

Parameters:

`environment` - the environment

Turn your Java APIs to Kotlin ones with JSpecify

Make your Java libraries null-safe

1 Read JSpecify user guide

2 Add `org.jspecify:jspecify:1.0.0` dependency

3 Annotate packages with `@NullMarked`

4 Annotate with `@Nullable` when relevant:
- fields
- method parameters
- method return values

5 Introduce build-time checks with NullAway

WebMvc or WebFlux?

WebMvc is recommended for most use cases

Provides support for Coroutines and `Flow<T>`
Use Java 25 and set `spring.threads.virtual.enabled=true`

Programmatic or annotation-based configuration?

In Kotlin, both are idiomatic!

I tend to prefer programmatic for microservices
and annotations for bigger applications
but you don’t have to chose and can combine them!

With annotations, autowire constructor parameters

```
@RestController
class UserController(private val repository: UserRepository) {

    @GetMapping(“/users")
    fun users() = repository.findAll()
}
```

Combine @Bean and router DSL

```
@Configuration(proxyBeanMethods = false)
class UserConfiguration {

    @Bean
    fun myRouter(userRepository: UserRepository) = router {
        GET("/users") {
            ok().body(userRepository.findAll())
        }
    }
}
```

Typical programmatic configuration application class

```
// Don't use @SpringBootApplication if you don't need component scanning
@EnableAutoConfiguration
@SpringBootConfiguration(proxyBeanMethods = false)
@Import(Beans::class)
class BackendApplication

fun main(args: Array<String>) {
    runApplication<BackendApplication>(*args)
}
```

Typical programmatic configuration beans class

```
class Beans : BeanRegistrarDsl({

    registerBean {
        val builder = bean<RestClient.Builder>()
            .baseUrl(env.getRequiredProperty("image.generator.url"))
            .build()
        PetManagement(bean<PetRepository>(),
                      bean<PetImageRepository>(),
                      bean<VisitRepository>(), builder)
    }

    registerBean { globalRouter() }
    registerBean { vetRouter(bean<VetRepository>(),
                             bean<SpecialtyRepository>()) }
})
```

Typical programmatic router

```
fun vetRouter(vetRepository: VetRepository,
              specialtyRepository: SpecialtyRepository) = router {

    GET("/vets.html") {
        ok().contentType(MediaType.TEXT_HTML)
            .body(renderVets(vetRepository.findAll()
                .map { it.toDto(specialtyRepository) }))
    }
    GET("/vets") {
        ok().contentType(MediaType.APPLICATION_JSON)
            .bodyWithType(vetRepository.findAll()
                .map { it.toDto(specialtyRepository) })
    }
}
```

Create your own configuration DSL!

Libraries can provide high-level configuration DSLs

```
abstract class CustomDsl(private val init: CustomDsl.() -> Unit) : BeanRegistrar {

    private lateinit var registry: BeanRegistry
    private lateinit var env: Environment

    override fun register(registry: BeanRegistry, env: Environment) {
        this.registry = registry
        this.env = env
        init()
    }

    fun userTypes(vararg types: String) {
        for (type in types) {
            registry.registerBean(type, UserRepository::class.java)
        }
    }
}
```

Easy to use, flexible and powerful on application side

```
@SpringBootApplication
@Import(MyCustomDsl::class)
class MyApplication {
}

class MyCustomDsl : CustomDsl({
    userTypes("type1", "type2")
})
```

Logging

Simple Kotlin extensions for SLF4J

```
inline fun <reified T> LoggerFactory.loggerForType(): Logger =
    LoggerFactory.getLogger(T::class.java)

inline fun logger(): Logger =
    LoggerFactory.getLogger(MethodHandles.lookup().lookupClass())

inline fun Logger.debug(message: () -> String) {
    if (isDebugEnabled) {
        debug(message())
    }
}

// ...
```

Logger with automatic class detection

```
@RestController
class UserController {

    companion object {
        val logger = logger()
    }

    @GetMapping("/")
    fun users(): List<User> {
        logger.info("...")
        // ...
    }
}
```

Logger with explicit type

```
val logger = loggerForType<UserController>()

@RestController
class UserController {

    @GetMapping("/")
    fun users(): List<User> {
        logger.info("...")
        // ...
    }
}
```

Potential related upcoming official dependency

Testing

Test classes

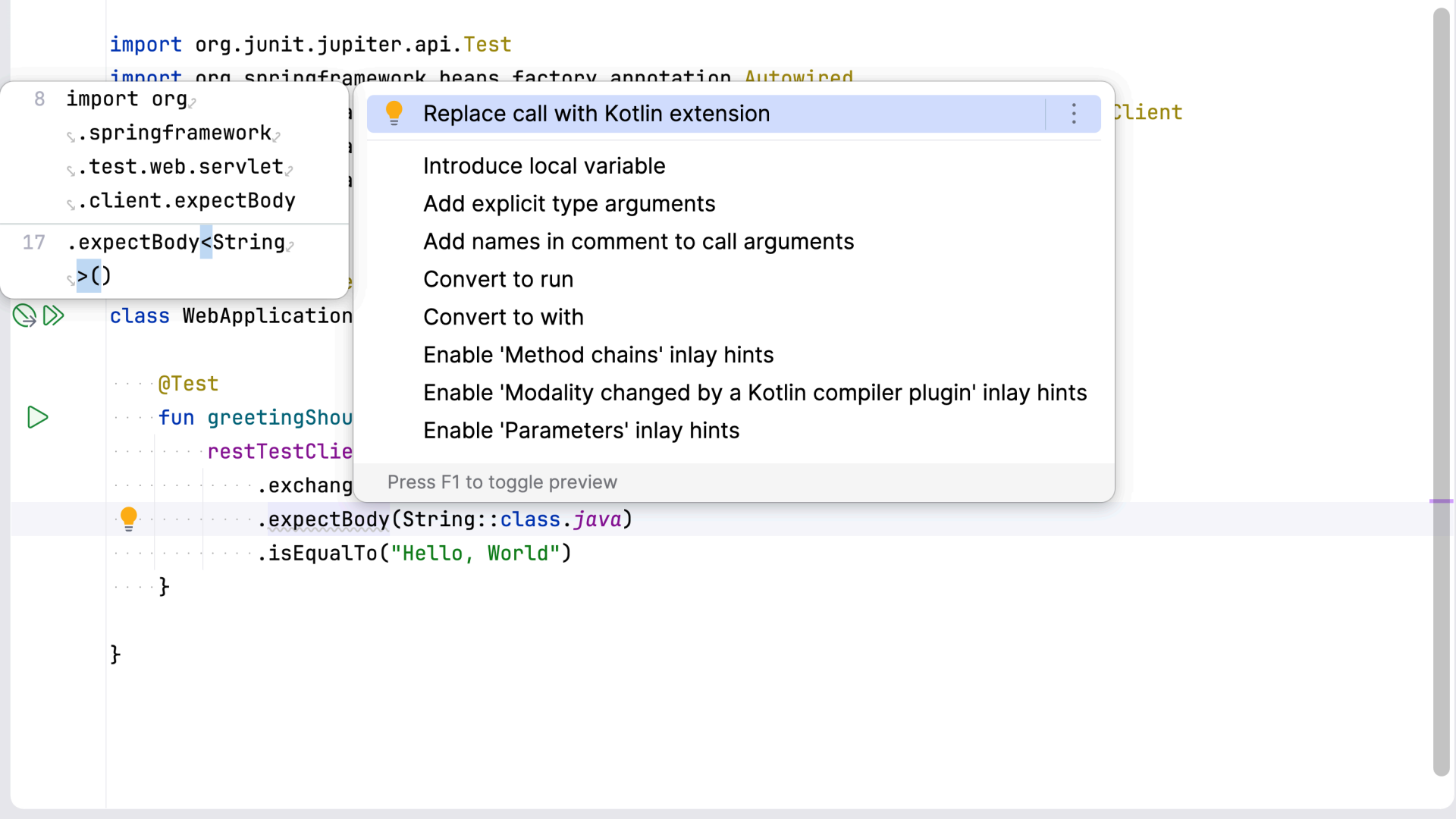
```
@SpringBootTest
@AutoConfigureRestTestClient
class WebApplicationTest(@Autowired private val restTestClient: RestTestClient) {

    @Test
    fun greetingShouldReturnDefaultMessage() {
        restTestClient.get().uri("/")
            .exchange()
            .expectBody<String>()
            .isEqualTo("Hello, World")
    }

}
```

Autowiring of constructors

```
@SpringBootTest
@AutoConfigureRestTestClient
class WebApplicationTest(@Autowired private val restTestClient: RestTestClient) {

    @Test
    fun greetingShouldReturnDefaultMessage() {
        restTestClient.get().uri("/")
            .exchange()
            .expectBody<String>()
            .isEqualTo("Hello, World")
    }

}
```

Use Kotlin extension when specifying types

```
@SpringBootTest
@AutoConfigureRestTestClient
class WebApplicationTest(@Autowired private val restTestClient: RestTestClient) {

    @Test
    fun greetingShouldReturnDefaultMessage() {
        restTestClient.get().uri("/")
            .exchange()
            .expectBody<String>()
            .isEqualTo("Hello, World")
    }

}
```


import org.junit.jupiter.api.Test
8 import org
.springframework
.test.web.servlet
.client.expectBody
17 .expectBody<String
>()
Replace call with Kotlin extension
Introduce local variable
Add explicit type arguments
Add names in comment to call arguments
Convert to run
Convert to with
Enable 'Method chains' inlay hints
Enable 'Modality changed by a Kotlin compiler plugin' inlay hints
Enable 'Parameters' inlay hints
Press F1 to toggle preview
class WebApplication
@Test
fun greetingShou
restTestClie
.exchang
.expectBody(String::class.java)
.isEqualTo("Hello, World")
}
}

https://docs.spring.io/spring-framework/docs/current/kdoc-api/

spring-beans
spring-context
spring-core
spring-jdbc
spring-messaging
spring-r2dbc
spring-test
org.springframework.test.web.reactive.server
org.springframework.test.web.servlet
org.springframework.test.web.servlet.client
expectBody()

expectBody

```
inline fun <B : Any> RestTestClient.ResponseSpec.expectBody(): RestTestClient.BodySpec<B, *>
```

Extension for ResponseSpec.expectBody providing an `expectBody<Foo>()` variant leveraging Kotlin reified type parameters. This extension is not subject to type erasure and retains actual generic type arguments.

Author

Sebastien Deleuze
Arjen Poutsma
Stephane Nicoll

Since

7.0

Kotlin Serialization first class support

Kotlin Serialization support

Spring Boot 3

❌ No dedicated starter / Module

⚠️ Library classpath activation

⚠️ Overlap with Jackson

Spring Boot 4

✅ Activation with spring-boot-kotlinx-serialization-json

✅ Dedicated starter spring-boot-starter-kotlinx-serialization-json

✅ Only `@Serializable`

✅ No overlap with Jackson

✅ Open polymorphism support

Data access for SQL databases

Idiomatic JPA ≠ idiomatic Kotlin

```kotlin
@Entity
open class Person {

    @Id
    @GeneratedValue
    var id: Long? = null

    var name: String? = null
}
```

Guidelines validated by Spring and Kotlin teams

How to avoid common pitfalls
with JPA and Kotlin

Using Spring Data JPA
with Kotlin

Use Spring Data JDBC for Kotlin idiomatic code

```kotlin
@Table("persons")
data class Person(
    val name: String,
    @Id val id: Long? = null
)
```

Context propagation with Coroutines

Automatic context propagation with Coroutines

➕ Add `io.micrometer:context-propagation` dependency

⚙️ Configure `spring.reactor.context-propagation=auto`

✅ Observability and tracing now work in suspending functions!

💡 See `PropagationContextElement` for more details

Spring AI with Kotlin

https://kotlinlang.org/docs/spring-ai-guide.html

Create a Kotlin app that answers questions with Spring AI — tutorial

Edit page 16 March 2026

In this tutorial, you'll learn how to build a Kotlin app that connects to an LLM via Spring AI ↗, stores documents in a vector database, and answers questions using context from those documents.

You will use the following tools during this tutorial:

- Spring Boot ↗ as the base to configure and run the web application.
- Spring AI ↗ to interact with the LLM and perform context-based retrieval.

Create a Kotlin app that answers questions with Spring AI — tutorial
Before you start
Create the project
Update the project configuration
Create a controller to load and search documents
Implement an AI chat endpoint

Spring AI roadmap

2.0 (May 2026)

- ⭐ Spring Boot 4 baseline
- ⭐ Idiomatic Kotlin null-safe API
- ⭐ Focus on quality and consistency

2.1 (November 2026)

- ⭐ New spring-ai-agent module
- ⭐ Durable workflow orchestrations
- ⭐ Session API
- ⭐ Kotlin DSLs

3.0 (May 2027)

- ⭐ Virtual Thread friendly Unified Streaming API
- ⭐ Adapter to Kotlin Flow

Faster startup with AOT cache (Project Leyden)

Faster startup with AOT cache

```
java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(25)
    }
}

tasks.withType<BootBuildImage>() {
    environment = mapOf("BP_JVM_AOTCACHE_ENABLED" to "true")
}
```

Faster startup with AOT cache

```
./gradlew bootBuildImage
...
    [creator]     AOTCache creation is complete: application.aot 62455808 bytes
    [creator]     Removed temporary AOT configuration file application.aot.config
    [creator]         Writing env.launch/BPL_JVM_AOTCACHE_ENABLED.default
...
Successfully built image 'docker.io/library/my-kotlin-application:1.0.0-SNAPSHOT'

# With AOT Cache
Started MyApplicationKt in 0.297 seconds (process running for 0.392)

# Without AOT Cache
Started MyApplicationKt in 0.807 seconds (process running for 0.963)
```

Thank you

https://seb.deleuze.fr


vmware Tanzu
145
Sébastien Deleuze
Master Engineer
Close the Null-Void: Eliminate all undefined attacks.
Graalchemy: next three cards cost 0 mana to execute instantly.
Kotlanic Manœúvre: +10 damage idiomatic strike.
POWER 101
72