Hands-On Kotlin Web Development with Ktor

Leonid Stashevskii

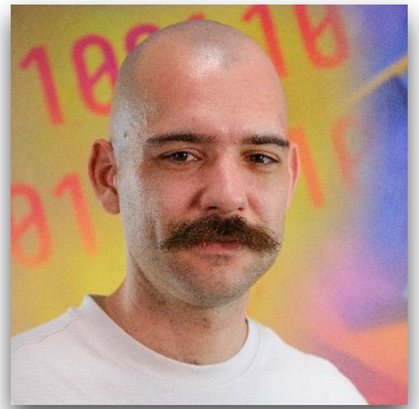
Ktor developer

@_ _ _e5l

Anton Arhipov

Developer Advocate

@antonarhipov



Simon Vergauwen

Developer Advocate

@vergauwen_simon

Hands-On Kotlin Web Development with Ktor

Part 1. Introduction, CRUD

Part 2. Persistence, Exposed

Part 3. Authentication

Part 4. LLM chat, LangChain4j

Rules of the game: hands-on

For each part, we will:

1. Provide you with a starter project

2. Give you some time to add functionality

3. Provide a sample solution and walk you through the code

Setup

IntelliJ IDEA with Kotlin (and Ktor) plugins

~~Docker~~

HTTP client

Database client ?

Setup https://github.com/nomisRev/ktor-workshop-2025

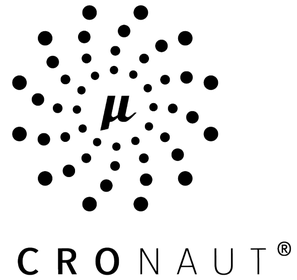
```
% git branch
* main
  branch01
  branch02
  branch03
  branch04
  branch05
  branch06
  branch06-jdbc
  branch07
  branch07-jdbc
  branch08
  branch08-jdbc
  branch09
  branch09-jdbc
  branch10
  branch11
```

Setup https://github.com/nomisRev/ktor-workshop-2025

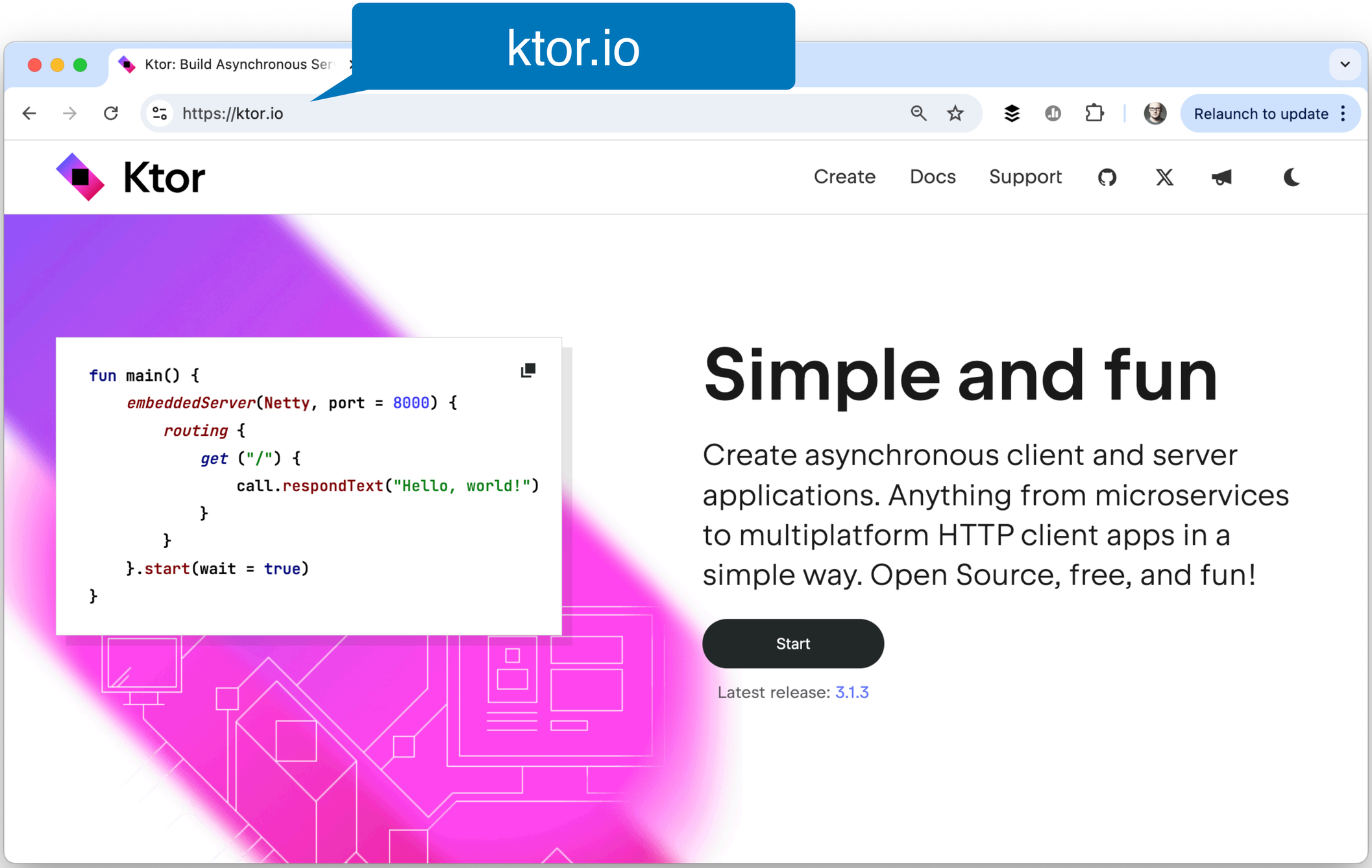
```
% git branch
* main
  branch01 ..... Initial state, generated application
  branch02 ..... First tests - empty
  branch03 ..... First tests - implementation
  branch04 ..... CRUD implementation (fake repository)
  branch05 ..... Adding structure and DI
  branch06
  branch06-jdbc ... Database access with Exposed - basics
  branch07
  branch07-jdbc ... Database access with Exposed - adding relations
  branch08
  branch08-jdbc ... Database access with Exposed - adding entities
  branch09
  branch09-jdbc ... Authentication with OAuth2, sessions, and JWT
  branch10 ..... WebSockets, SSE, KotlinX Serialization
  branch11 ..... Final project
```

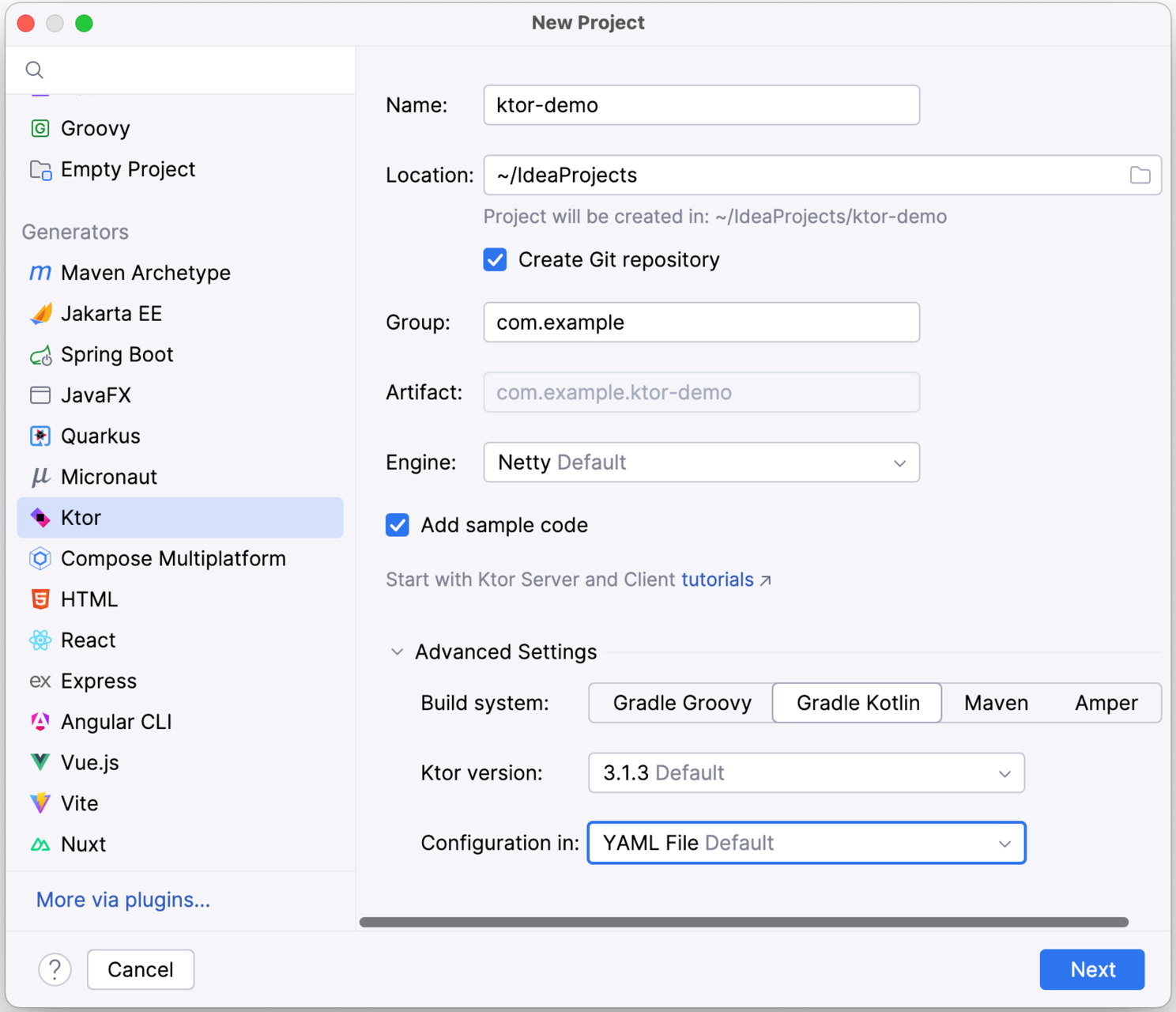
Part 1. Introduction to Ktor

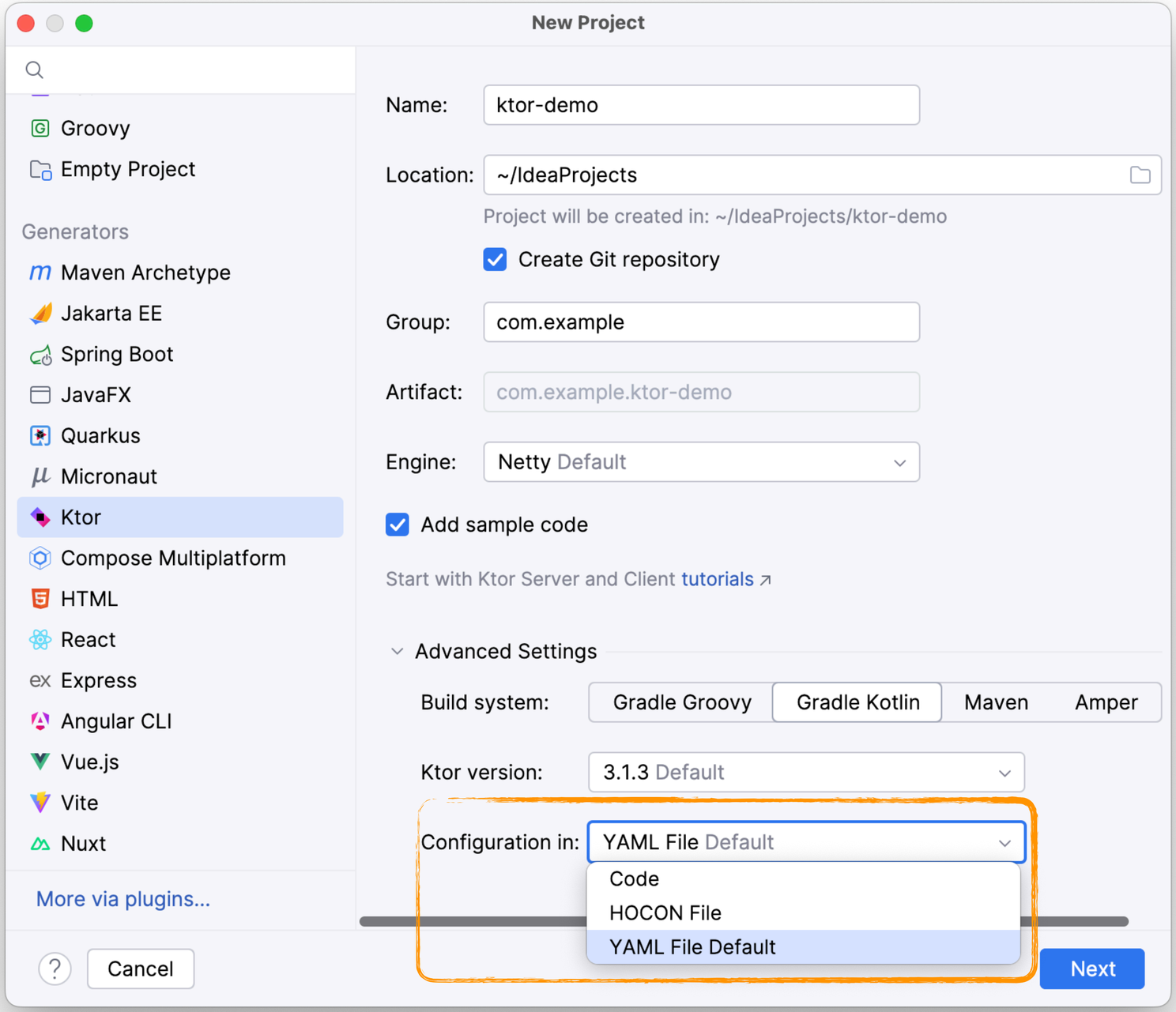
What web framework should I use with Kotlin?

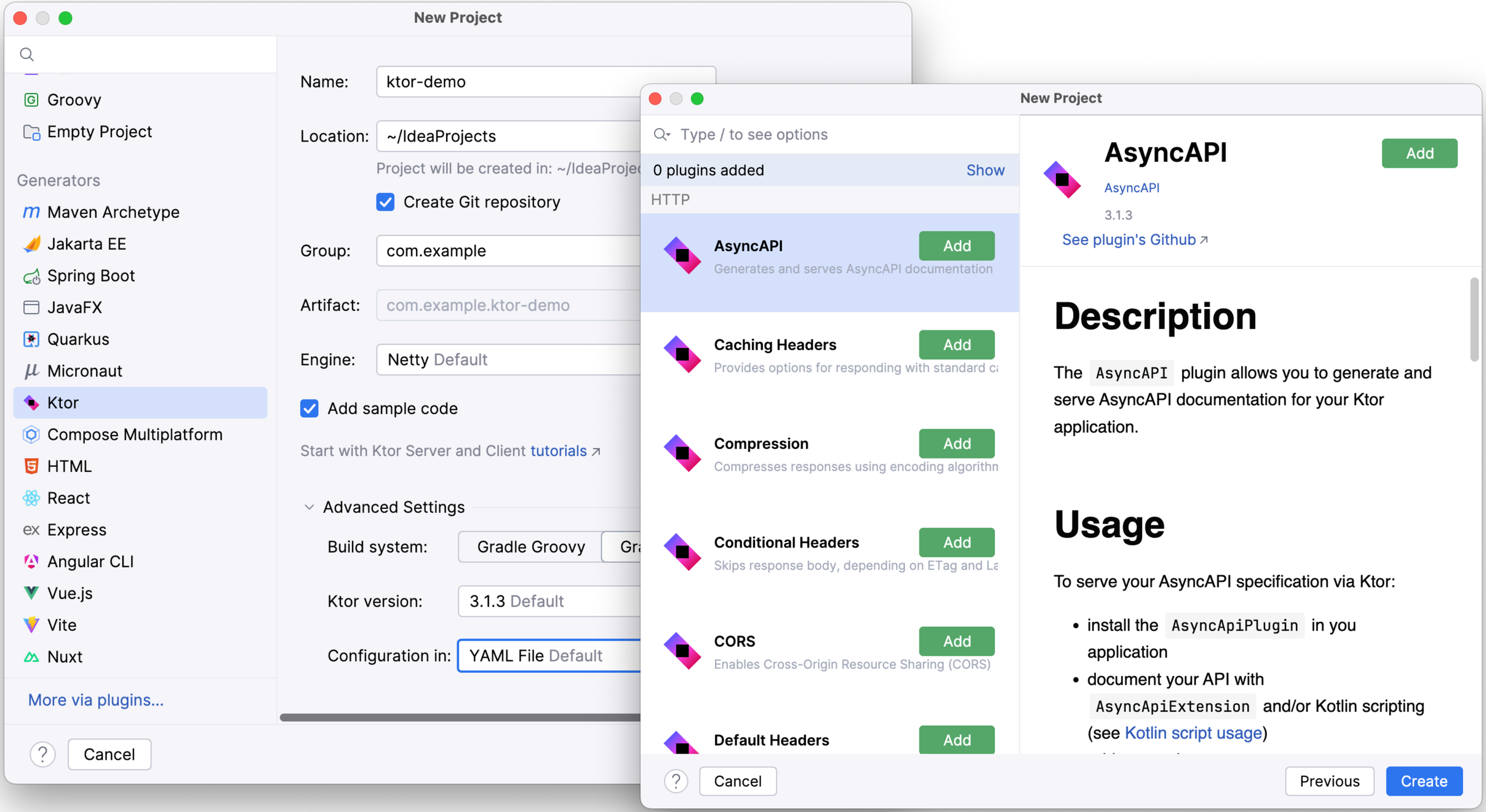


** There is more, of course!*

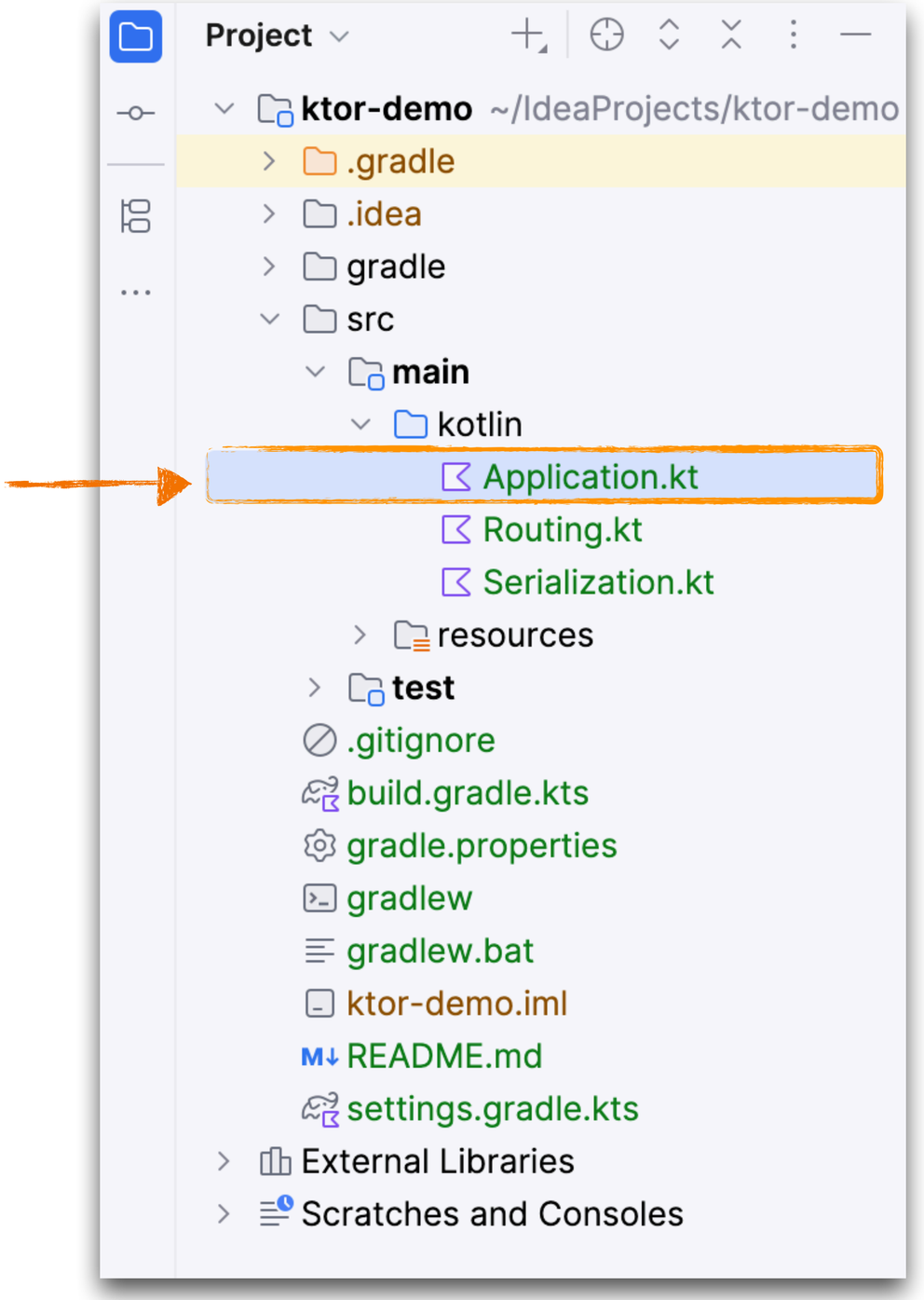

ktor.io
Ktor: Build Asynchronous Ser
https://ktor.io
Relaunch to update
Ktor
Create
Docs
Support
fun main() {
embeddedServer(Netty, port = 8000) {
routing {
get ("/") {
call.respondText("Hello, world!")
}
}
}.start(wait = true)
}
Simple and fun
Create asynchronous client and server applications. Anything from microservices to multiplatform HTTP client apps in a simple way. Open Source, free, and fun!
Start
Latest release: 3.1.3


New Project
Groovy
Empty Project
Generators
Maven Archetype
Jakarta EE
Spring Boot
JavaFX
Quarkus
Micronaut
Ktor
Compose Multiplatform
HTML
React
Express
Angular CLI
Vue.js
Vite
Nuxt
More via plugins...
Name:
ktor-demo
Location:
~/IdeaProjects
Project will be created in: ~/IdeaProjects/ktor-demo
Create Git repository
Group:
com.example
Artifact:
com.example.ktor-demo
Engine:
Netty Default
Add sample code
Start with Ktor Server and Client tutorials
Advanced Settings
Build system:
Gradle Groovy
Gradle Kotlin
Maven
Amper
Ktor version:
3.1.3 Default
Configuration in:
YAML File Default
Cancel
Next


New Project
Groovy
Empty Project
Generators
Maven Archetype
Jakarta EE
Spring Boot
JavaFX
Quarkus
Micronaut
Ktor
Compose Multiplatform
HTML
React
Express
Angular CLI
Vue.js
Vite
Nuxt
More via plugins...
Name: ktor-demo
Location: ~/IdeaProjects
Project will be created in: ~/IdeaProjects/ktor-demo
Create Git repository
Group: com.example
Artifact: com.example.ktor-demo
Engine: Netty Default
Add sample code
Start with Ktor Server and Client tutorials
Advanced Settings
Build system: Gradle Groovy Gradle Kotlin Maven Amper
Ktor version: 3.1.3 Default
Configuration in: YAML File Default
Code
HOCON File
YAML File Default
Cancel
Next

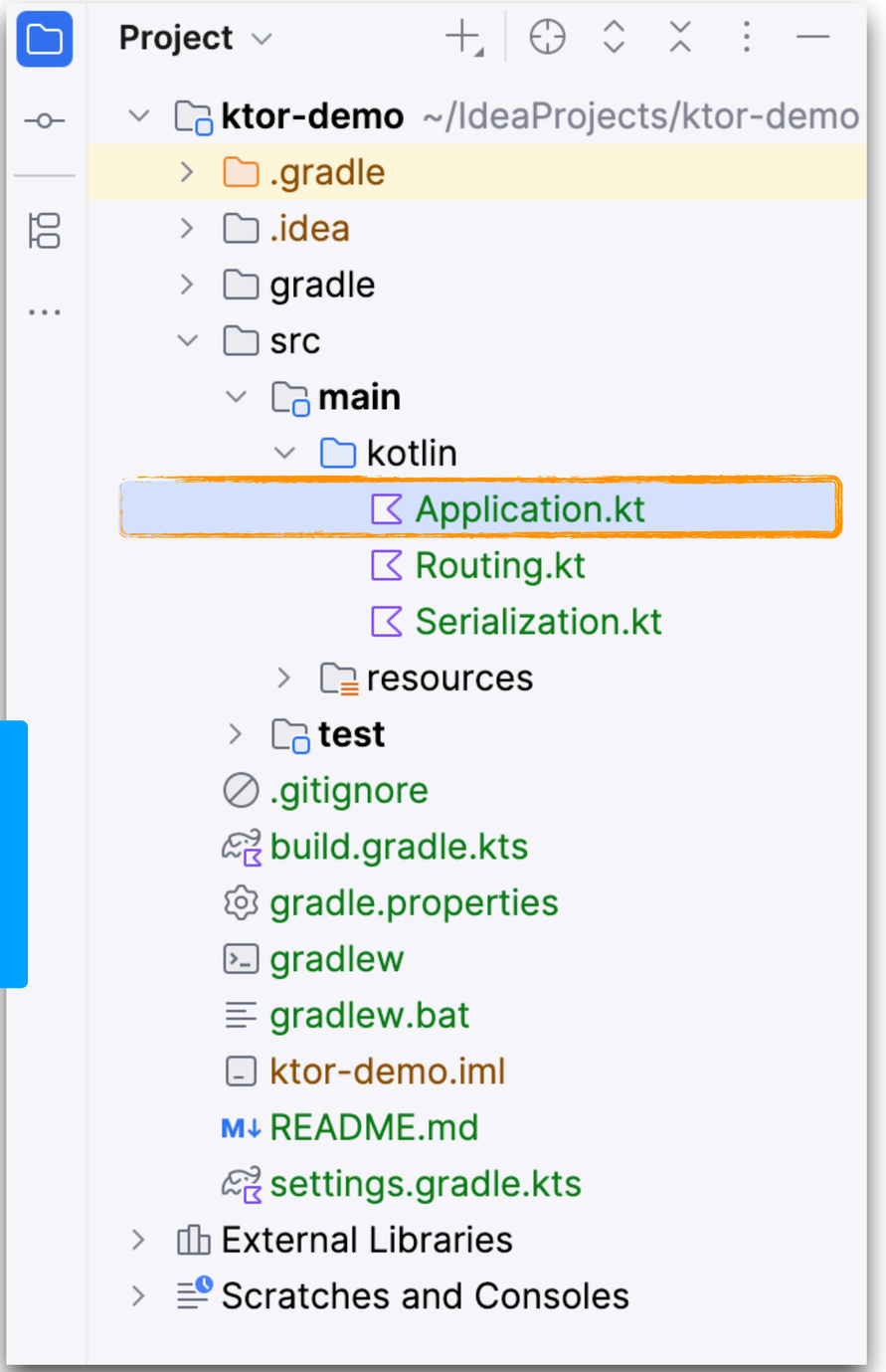

New Project
Groovy
Empty Project
Generators
Maven Archetype
Jakarta EE
Spring Boot
JavaFX
Quarkus
Micronaut
Ktor
Compose Multiplatform
HTML
React
Express
Angular CLI
Vue.js
Vite
Nuxt
More via plugins...
Name: ktor-demo
Location: ~/IdeaProjects
Project will be created in: ~/IdeaProje
Create Git repository
Group: com.example
Artifact: com.example.ktor-demo
Engine: Netty Default
Add sample code
Start with Ktor Server and Client tutorials
Advanced Settings
Build system: Gradle Groovy
Ktor version: 3.1.3 Default
Configuration in: YAML File Default
Cancel
New Project
Type / to see options
0 plugins added
Show
HTTP
AsyncAPI
Generates and serves AsyncAPI documentation
Add
Caching Headers
Provides options for responding with standard c
Compression
Compresses responses using encoding algorithn
Conditional Headers
Skips response body, depending on ETag and La
CORS
Enables Cross-Origin Resource Sharing (CORS)
Default Headers
AsyncAPI
AsyncAPI
3.1.3
See plugin's Github
Description
The AsyncAPI plugin allows you to generate and serve AsyncAPI documentation for your Ktor application.
Usage
To serve your AsyncAPI specification via Ktor:
install the AsyncApiPlugin in you application
document your API with AsyncApiExtension and/or Kotlin scripting (see Kotlin script usage)
Cancel
Previous
Create

```kotlin
package com.example

import io.ktor.server.application.*

fun main(args: Array<String>) {
    io.ktor.server.netty.EngineMain.main(args)
}

fun Application.module() {
    configureSerialization()
    configureRouting()
}
```

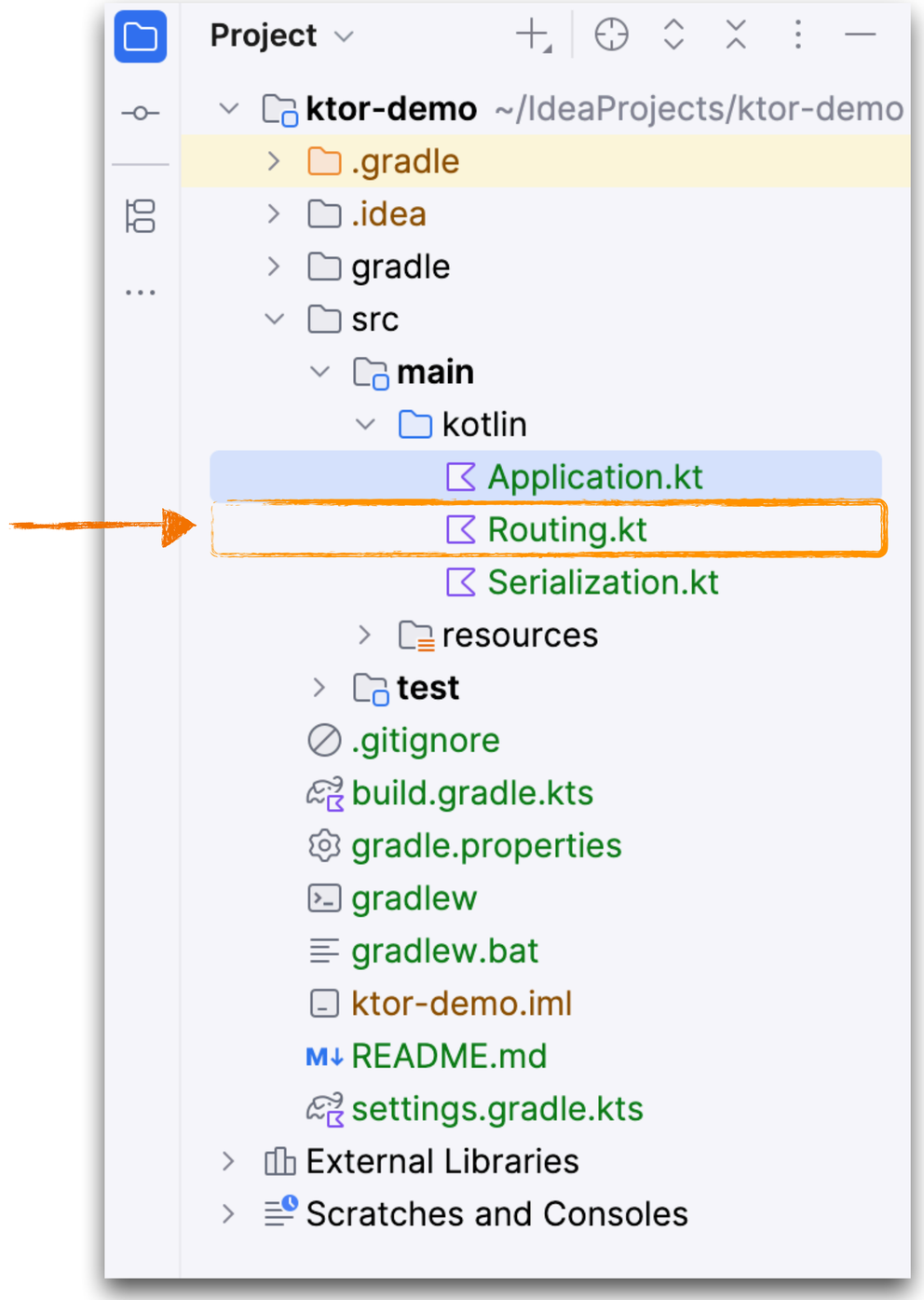


```kotlin
package com.example

import io.ktor.server.application.*

fun main(args: Array<String>) {
    io.ktor.server.netty.EngineMain.main(args)
}

fun Applicatio
    configureS
    configureR
}
```

Start an embedded server with Netty engine on port 8080

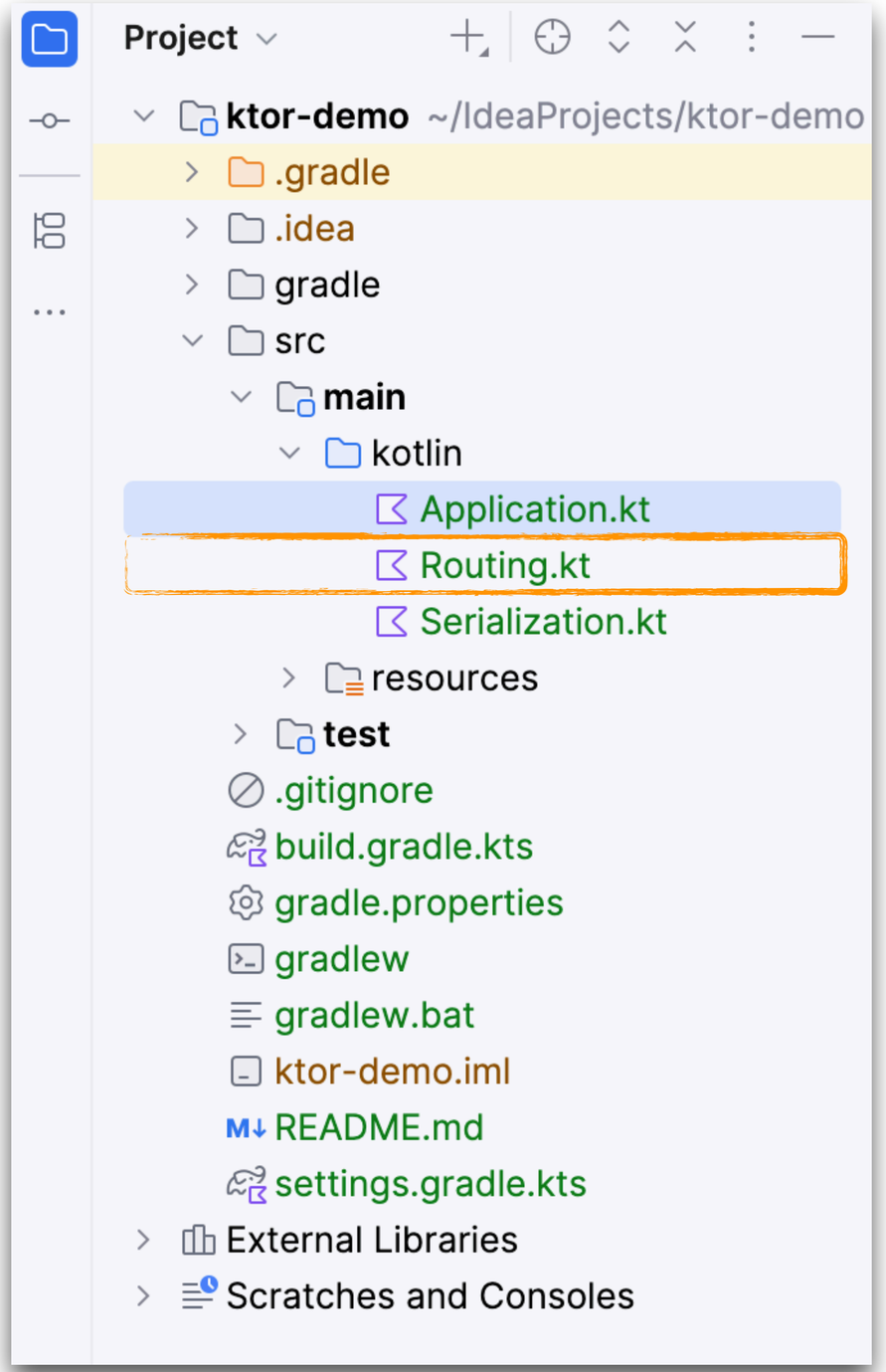


```kotlin
package com.example

import io.ktor.server.application.*
import io.ktor.server.response.*
import io.ktor.server.routing.*

fun Application.configureRouting() {
    routing {
        get("/") {
            call.respondText("Hello World!")
        }
    }
}
```



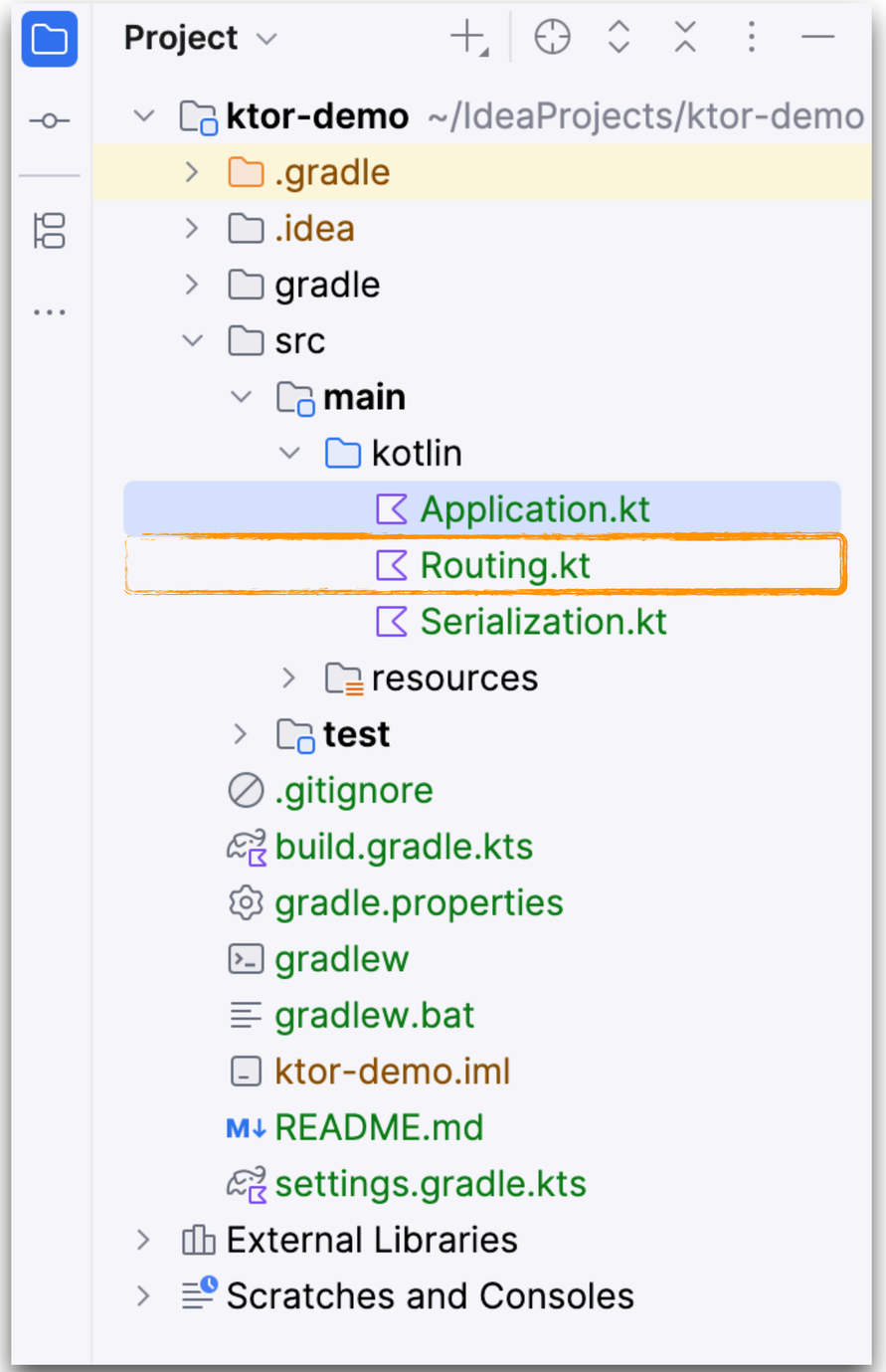
```kotlin
package com.example

import
import
import

fun Application.configureRouting() {
    routing {
        get("/") {
            call.respondText("Hello World!")
        }
    }
}
```

Adding routes



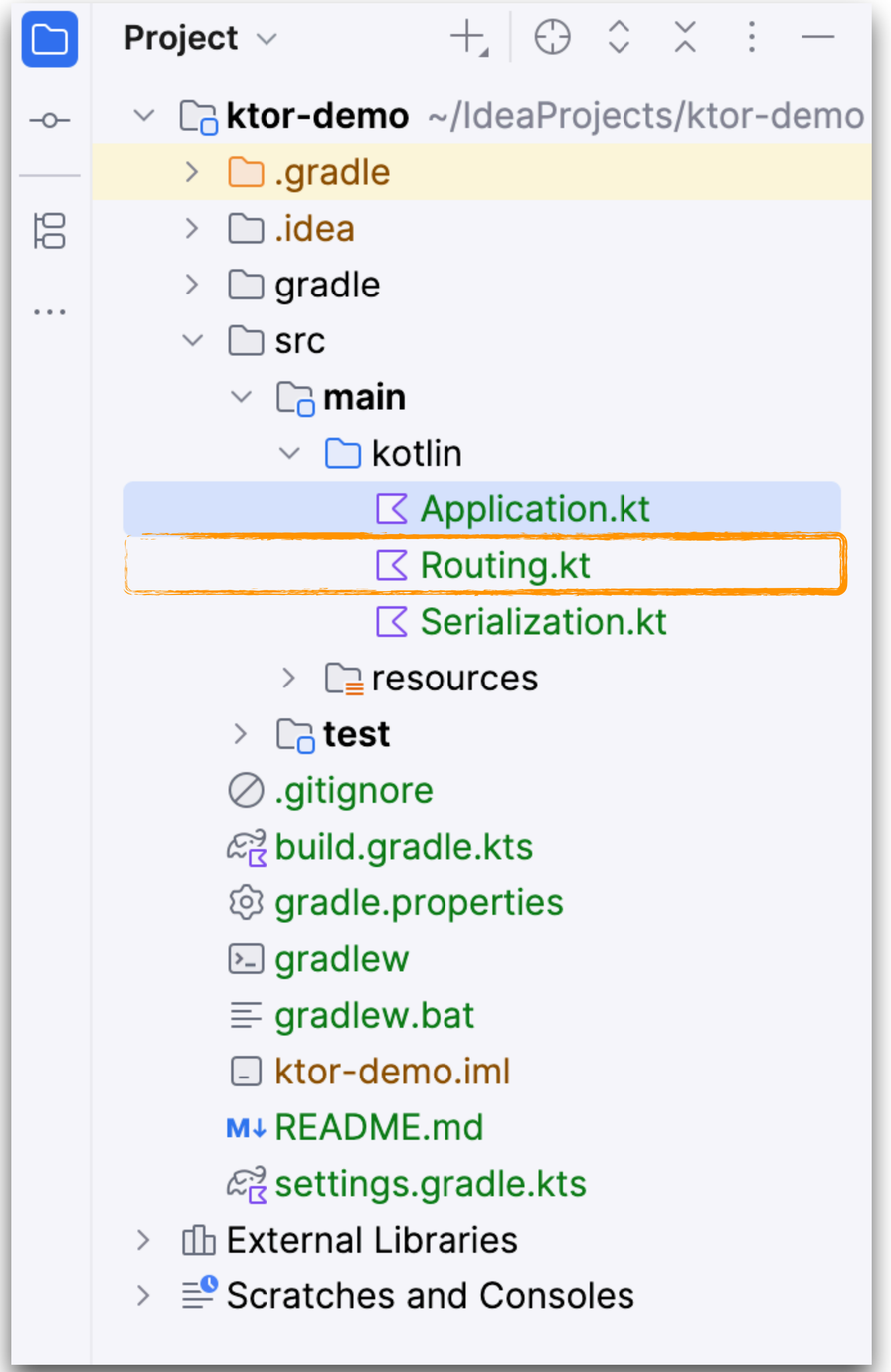
```kotlin
package com.example

import io.ktor.server.application.*
import io.ktor.server.response.*
import io.ktor.server.routing.*

fun Application.configureRouting() {
    routing {
        get("/") {
            call.respondText("Hello World!")
        }
    }
}
```

Request mapping

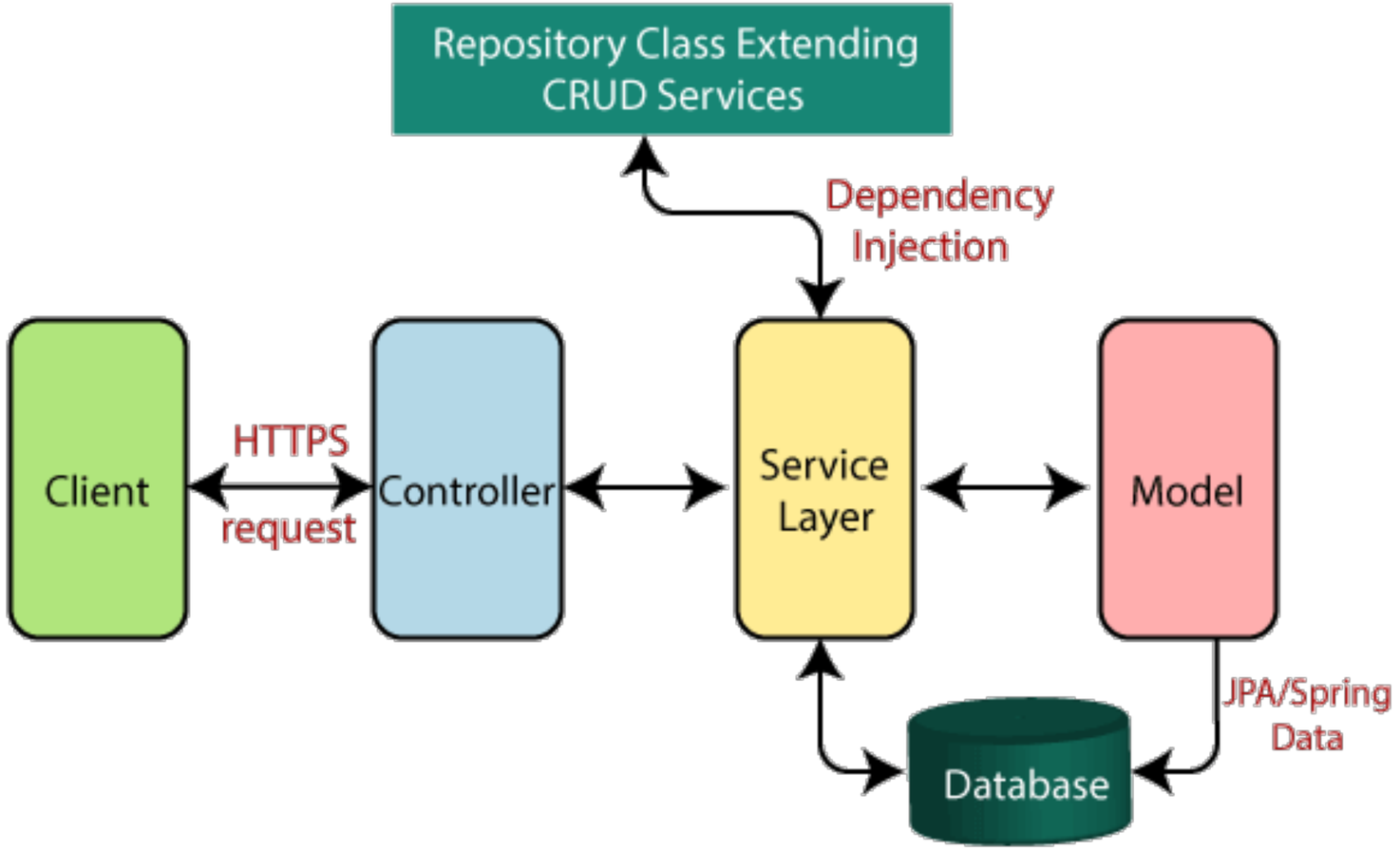


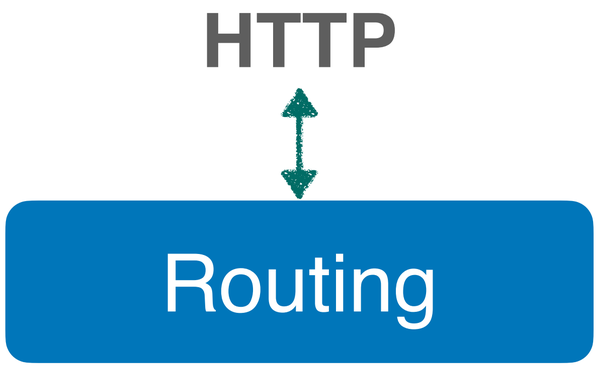
```kotlin
package com.example

import io.ktor.server.application.*
import io.ktor.server.response.*
import io.ktor.server.routing.*

fun Application.configureRouting() {
    routing {
        get("/") {
            call.respondText("Hello World!")
        }
    }
}
```

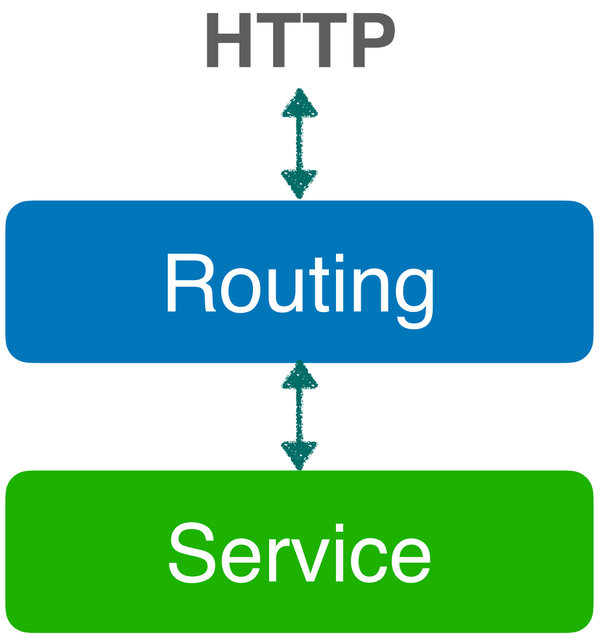
Write response

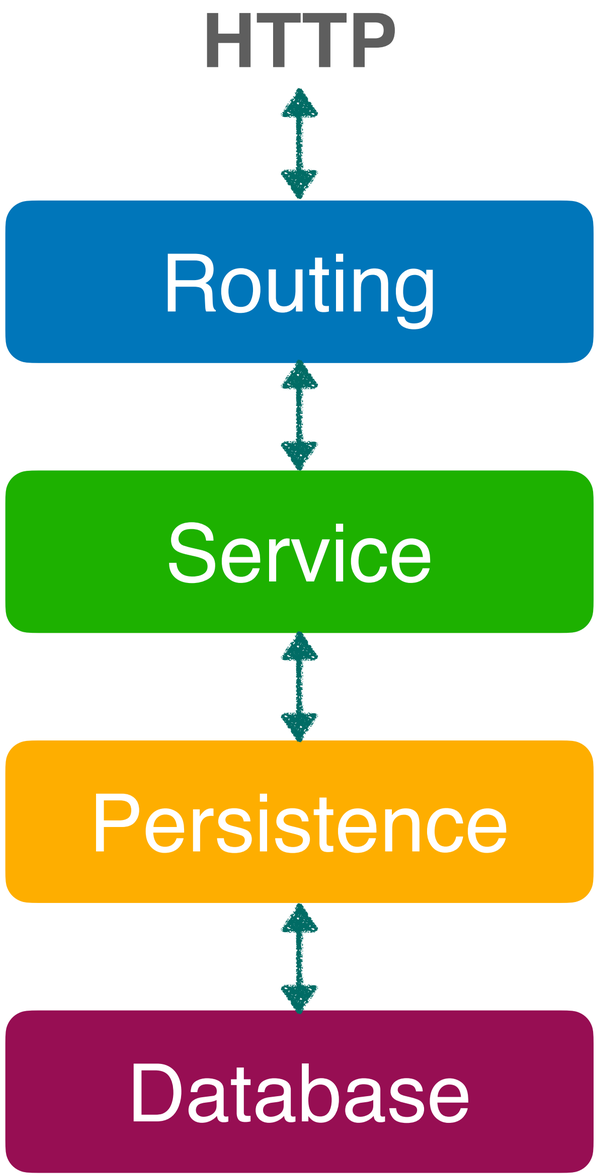


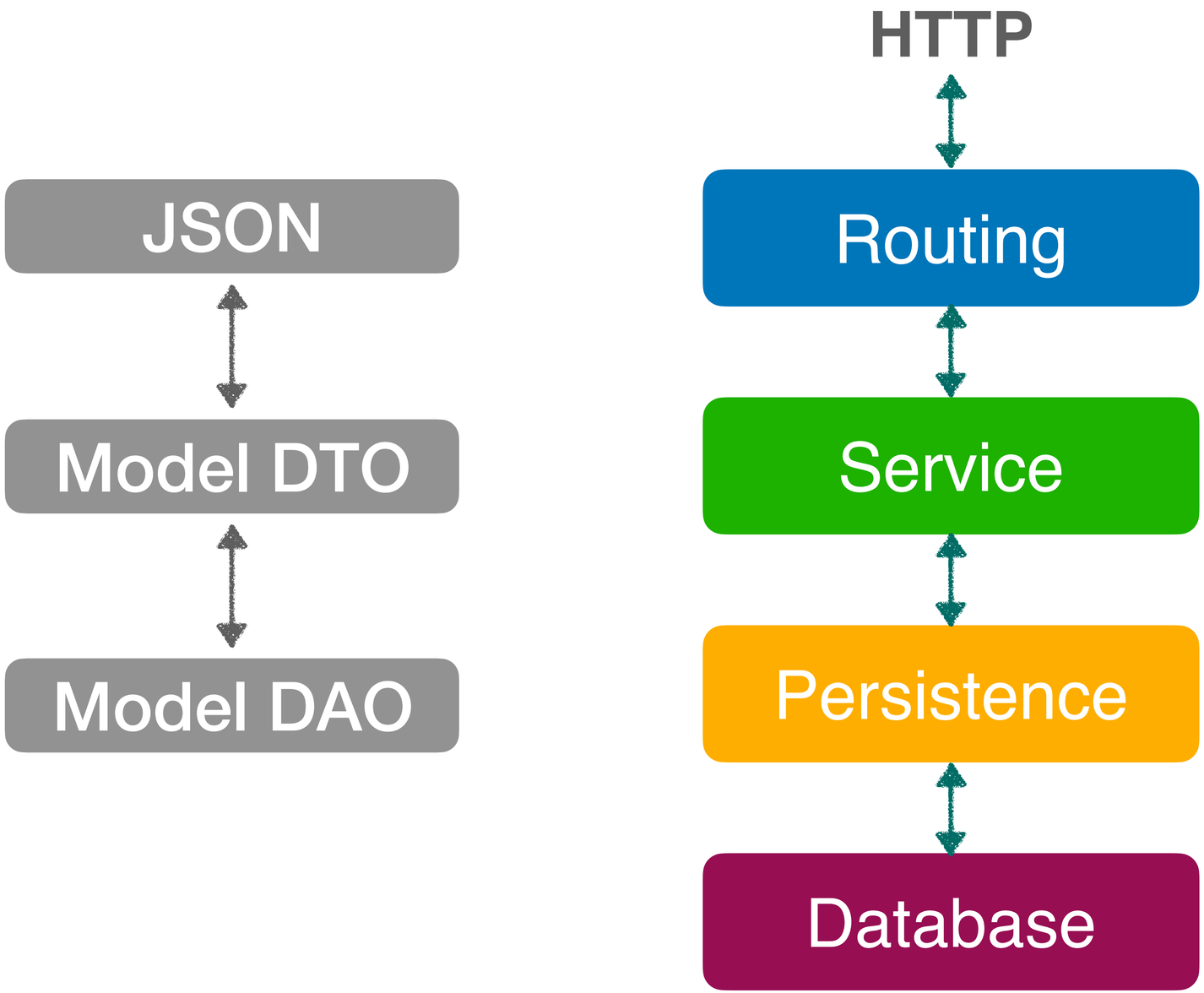
Spring Boot flow architecture

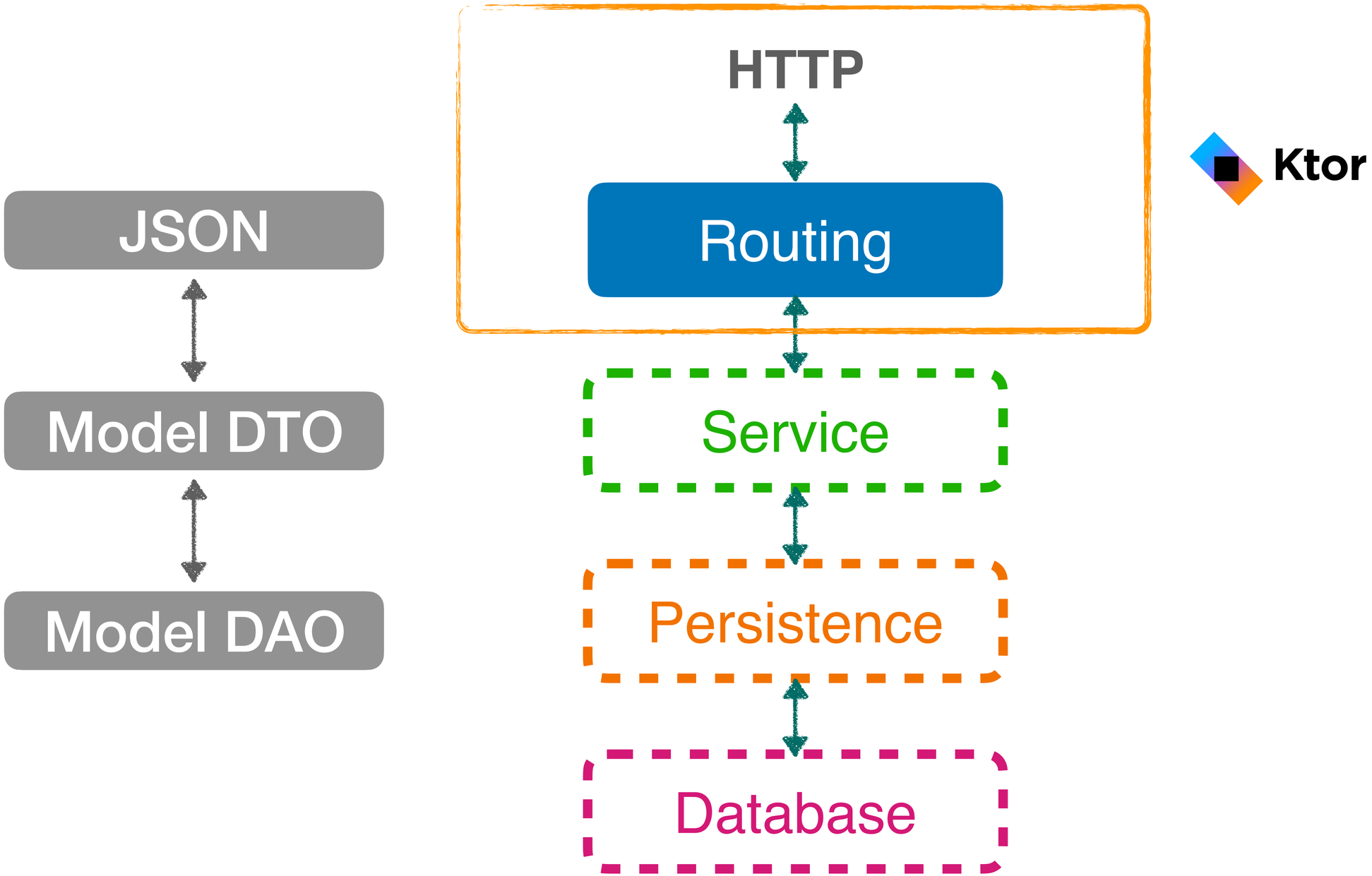


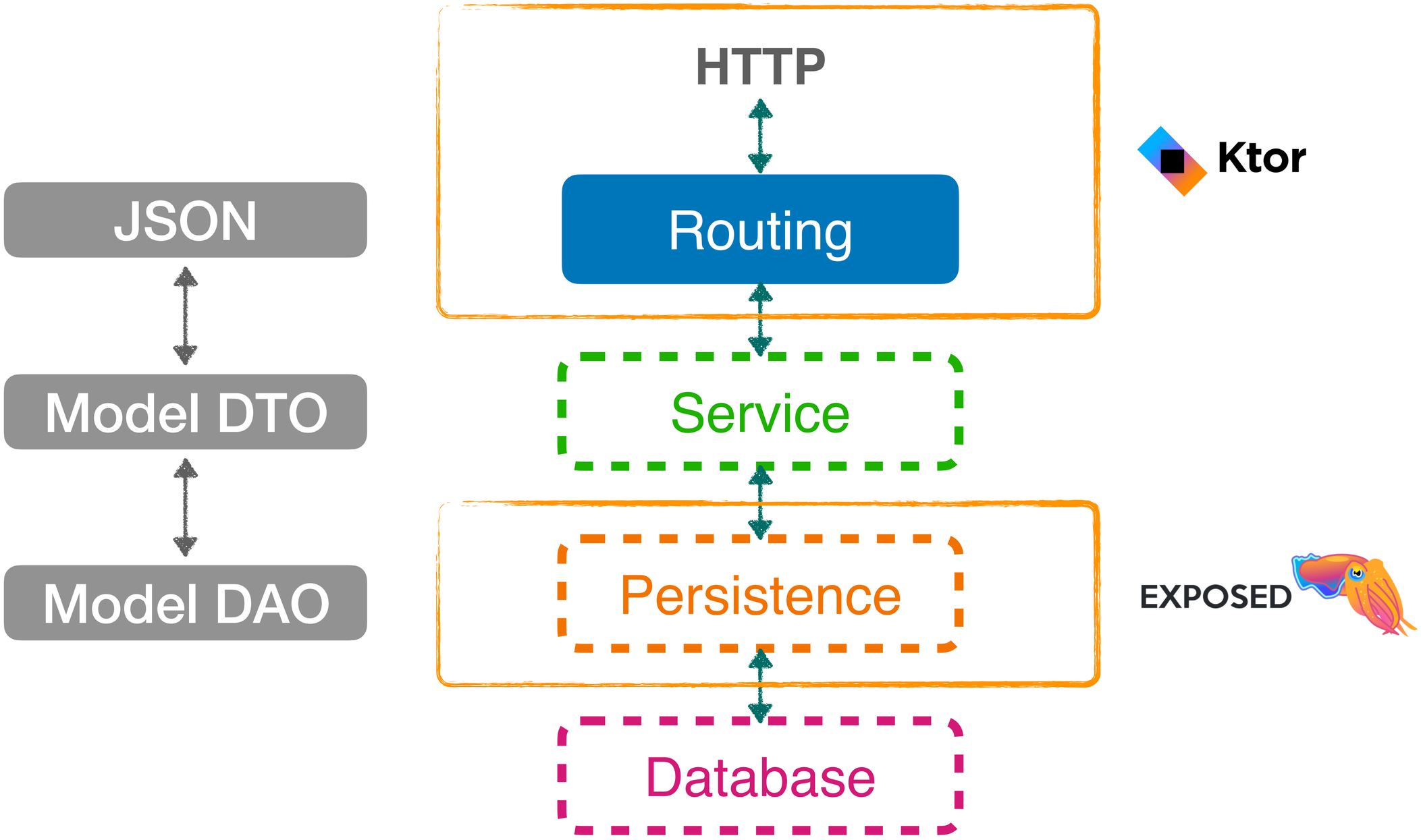

HTTP
Routing

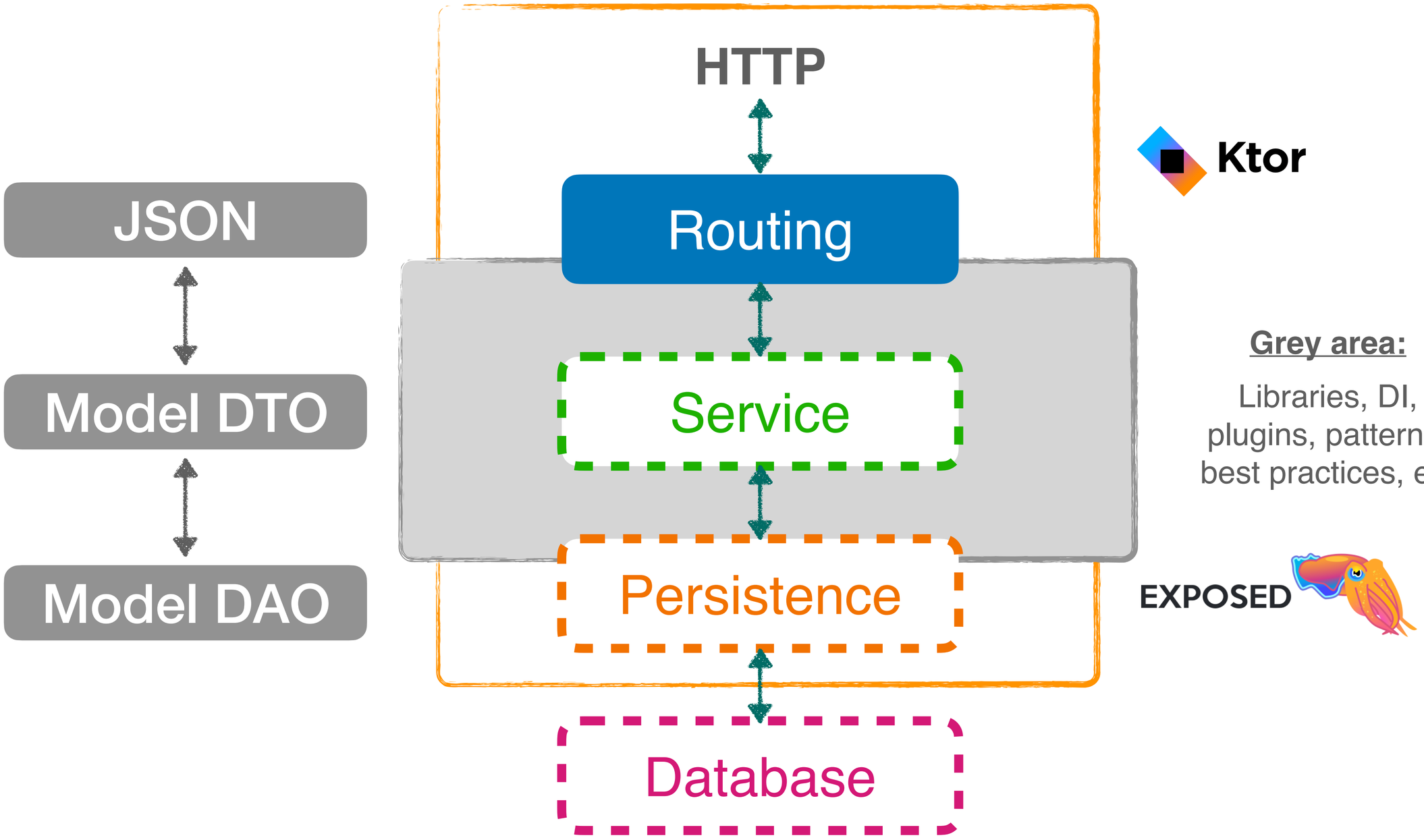

HTTP
Routing
Service

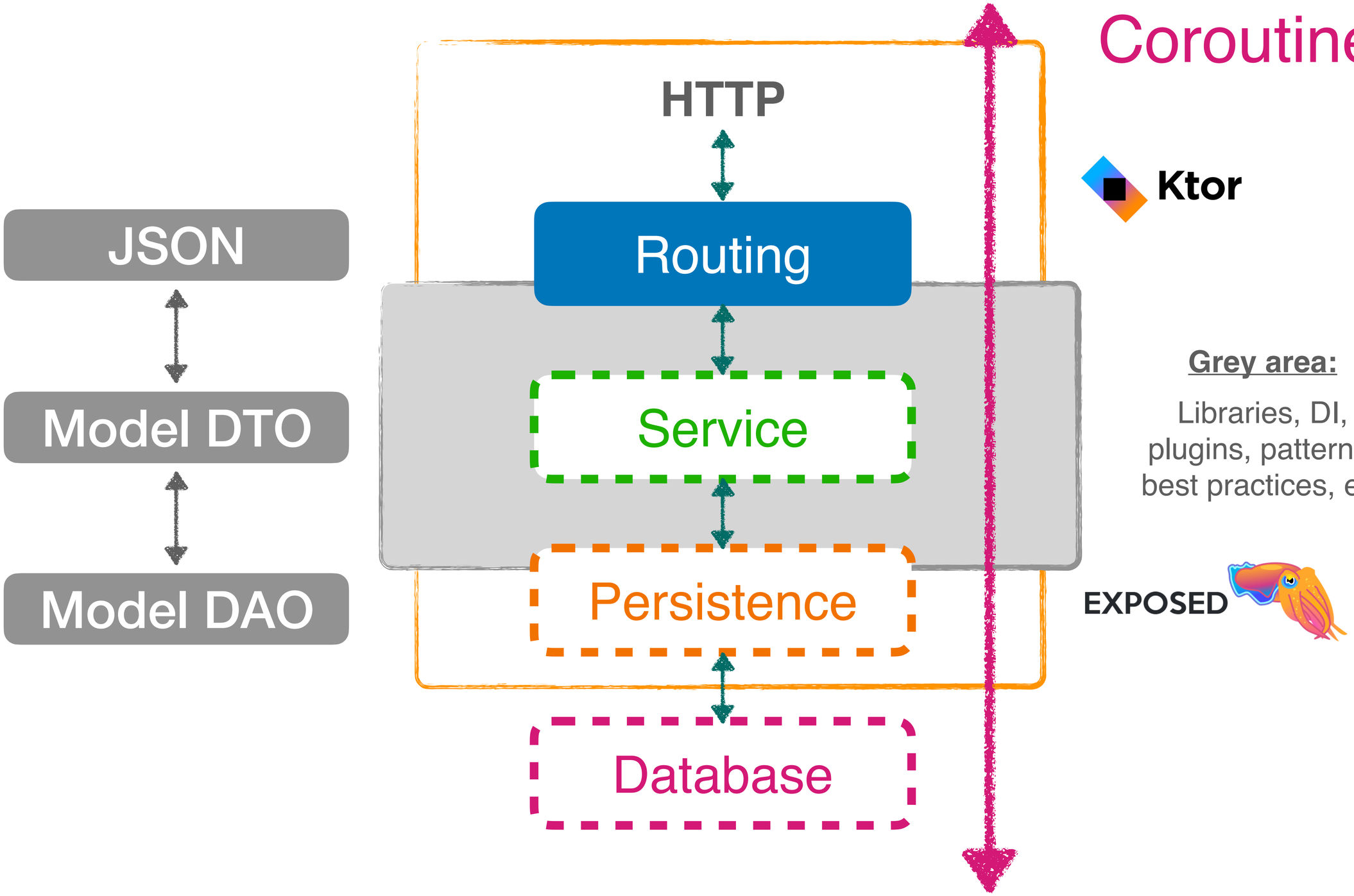

HTTP
Routing
Service
Persistence
Database


HTTP
Routing
Service
Persistence
Database
JSON
Model DTO
Model DAO


HTTP
Routing
Ktor
JSON
Model DTO
Model DAO
Service
Persistence
Database


HTTP
Routing
Ktor
JSON
Model DTO
Model DAO
Service
Persistence
EXPOSED
Database


HTTP
JSON
Routing
Model DTO
Service
Model DAO
Persistence
Database
Ktor
Grey area:
Libraries, DI,
plugins, patterns,
best practices, etc
EXPOSED


Coroutines
HTTP
Routing
Service
Persistence
Database
JSON
Model DTO
Model DAO
Ktor
Grey area:
Libraries, DI,
plugins, patterns,
best practices, etc
EXPOSED

Ktor's approach - explicitness

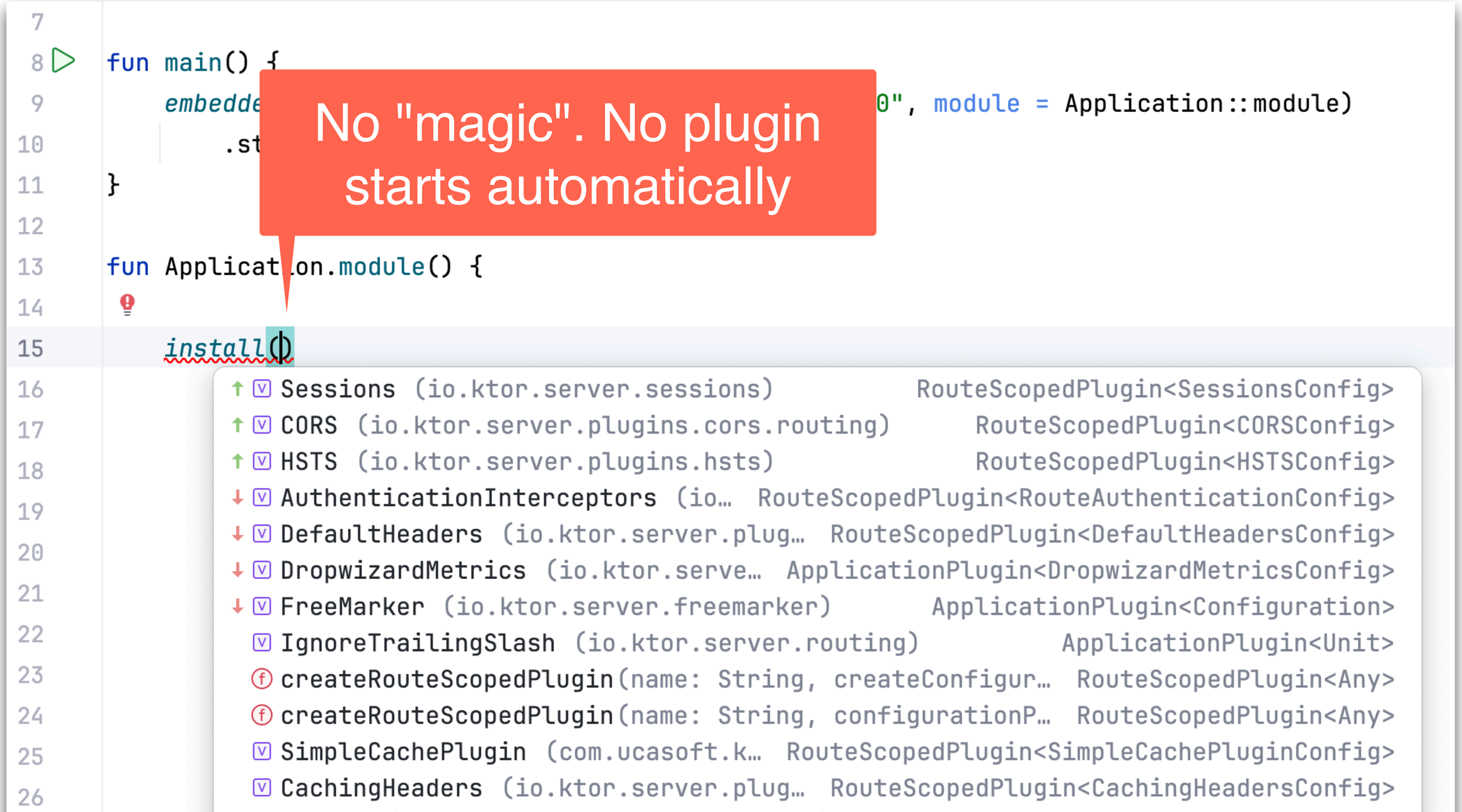
Ktor's approach - explicitness

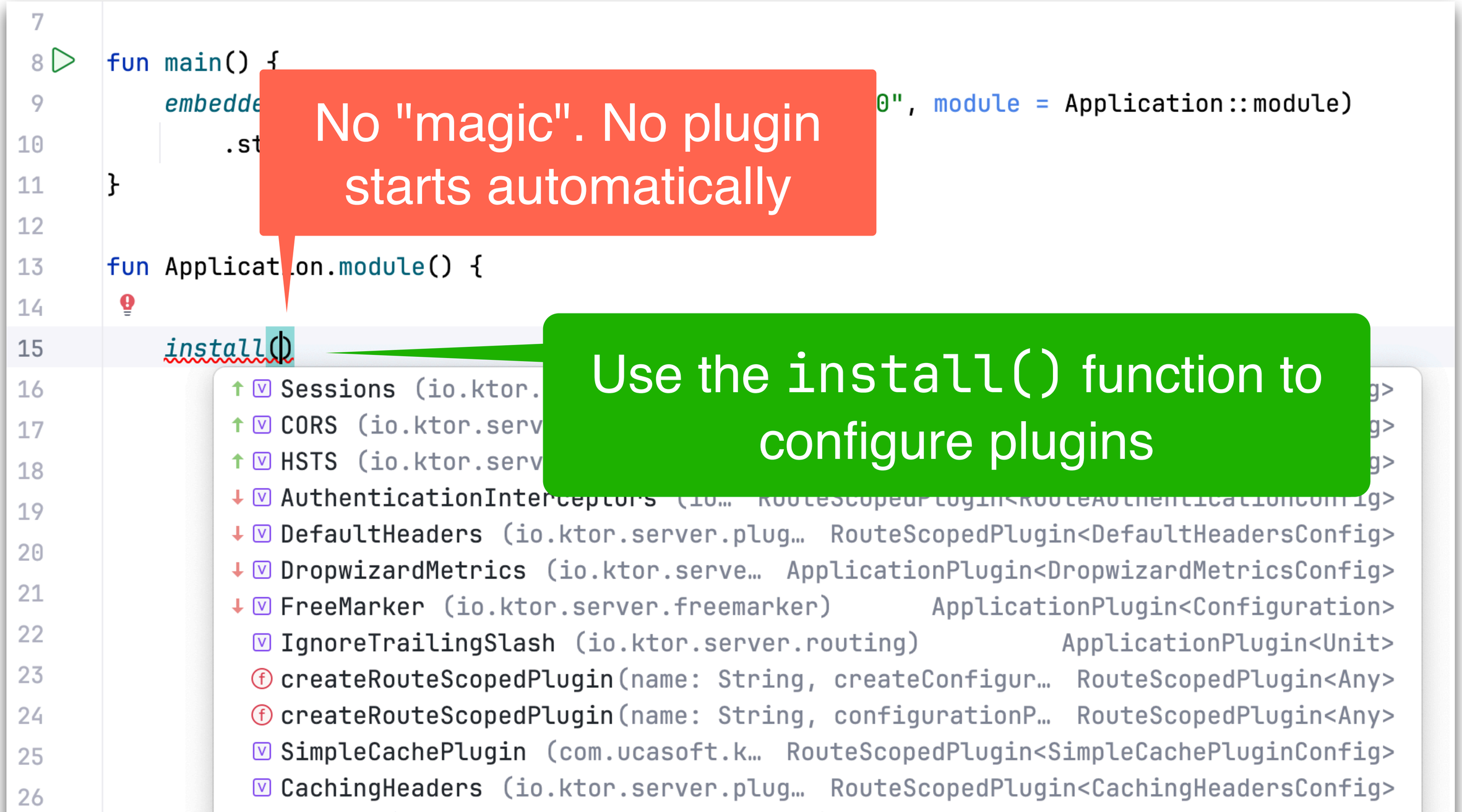
```
23  repositories { this: RepositoryHandler
24      mavenCentral()
25  }
26
27  dependencies { this: DependencyHandlerScope
28      implementation("io.ktor:ktor-server-core-jvm")
29      implementation("io.ktor:ktor-server-auth-jvm")
30      implementation("io.ktor:ktor-server-auth-jwt-j
31      implementation("io.ktor:ktor-server-resources"
32      implementation("io.ktor:ktor-server-webjars-jvm")
33      implementation("io.ktor:ktor-server-host-common-jvm")
34      implementation("io.ktor:ktor-server-status-pages-jvm")
35      implementation("io.ktor:ktor-server-caching-headers-jvm")
36      implementation("io.ktor:ktor-server-compression-jvm")
37      implementation("io.ktor:ktor-server-cors-jvm")
38      implementation("io.ktor:ktor-server-default-headers-jvm")
```

Fine-grained approach to dependencies

"Dependency per plugin"

Compatible with JPMS


No "magic". No plugin starts automatically
fun main() {
fun Application.module() {
install()
Sessions (io.ktor.server.sessions) RouteScopedPlugin<SessionsConfig>
CORS (io.ktor.server.plugins.cors.routing) RouteScopedPlugin<CORSConfig>
HSTS (io.ktor.server.plugins.hsts) RouteScopedPlugin<HSTSConfig>
AuthenticationInterceptors (io… RouteScopedPlugin<RouteAuthenticationConfig>
DefaultHeaders (io.ktor.server.plug… RouteScopedPlugin<DefaultHeadersConfig>
DropwizardMetrics (io.ktor.serve… ApplicationPlugin<DropwizardMetricsConfig>
FreeMarker (io.ktor.server.freemarker) ApplicationPlugin<Configuration>
IgnoreTrailingSlash (io.ktor.server.routing) ApplicationPlugin<Unit>
createRouteScopedPlugin(name: String, createConfigur… RouteScopedPlugin<Any>
createRouteScopedPlugin(name: String, configurationP… RouteScopedPlugin<Any>
SimpleCachePlugin (com.ucasoft.k… RouteScopedPlugin<SimpleCachePluginConfig>
CachingHeaders (io.ktor.server.plug… RouteScopedPlugin<CachingHeadersConfig>


No "magic". No plugin starts automatically
Use the install() function to configure plugins
fun main() {
0", module = Application::module)
}
fun Application.module() {
install()
Sessions (io.ktor.
CORS (io.ktor.serv
HSTS (io.ktor.serv
AuthenticationInterceptors
DefaultHeaders (io.ktor.server.plug… RouteScopedPlugin<DefaultHeadersConfig>
DropwizardMetrics (io.ktor.serve… ApplicationPlugin<DropwizardMetricsConfig>
FreeMarker (io.ktor.server.freemarker) ApplicationPlugin<Configuration>
IgnoreTrailingSlash (io.ktor.server.routing) ApplicationPlugin<Unit>
createRouteScopedPlugin(name: String, createConfigur… RouteScopedPlugin<Any>
createRouteScopedPlugin(name: String, configurationP… RouteScopedPlugin<Any>
SimpleCachePlugin (com.ucasoft.k… RouteScopedPlugin<SimpleCachePluginConfig>
CachingHeaders (io.ktor.server.plug… RouteScopedPlugin<CachingHeadersConfig>

~~Hands-on time!~~ Warmup

~~Hands-on time!~~ Warmup

Clone the repository, import the project in IntelliJ IDEA, **Run it**!!!

`https://github.com/nomisRev/ktor-workshop-2025`

```
% git branch
* main
  branch01
  branch02
  branch03
```

~~Hands-on time!~~ Warmup

Clone the repository, import the project in IntelliJ IDEA, **Run it**!!!

`https://github.com/nomisRev/ktor-workshop-2025`

```
% git branch
* main
  branch01
  branch02
  branch03
```

This is our starting point

Hands-on time!

Implement endpoints for CRUD operations for the User data class.

```
@Serializable
data class Customer(
    val id: Int,
    val name: String,
    val email: String,
    val createdAt: Instant
)

@Serializable
data class CreateCustomer(val name: String, val email: String)

@Serializable
data class UpdateCustomer(val name: String? = null, val email: String? = null)
```

Hands-on time!

Tests

Implement ~~endpoints for~~ CRUD operations for the User data class.

```
@Serializable
data class Customer(
    val id: Int,
    val name: String,
    val email: String,
    val createdAt: Instant
)

@Serializable
data class CreateCustomer(val name: String, val email: String)

@Serializable
data class UpdateCustomer(val name: String? = null, val email: String? = null)
```

```kotlin
import kotlin.test.*

class ApplicationTest {
    @Test
    fun testRoot() = testApplication {
        application {
            routing {
                get("/") {
                    call.respondText("Hello World!")
                }
            }
        }
        client.get("/").apply {
            assertEquals(HttpStatusCode.OK, status)
            assertEquals("Hello World!", bodyAsText())
        }
    }
}
```

Creates a TestApplication

Define functionality to be tested

```
import kotlin.test.*

class ApplicationTest {
    @Test
    fun testRoot() = testApplication {
        application {
            routing {
                get("/") {
                    call.respondText("Hello World!")
                }
            }
        }
        client.get("/").apply {
            assertEquals(HttpStatusCode.OK, status)
            assertEquals("Hello World!", bodyAsText())
        }
    }
}
```

```
import kotlin.test.*

class ApplicationTest {
    @Test
    fun testRoot() = testApplication {
        application {
            routing {
                get("/") {
                    call.respondText("Hello Wo
                }
            }
        }
        client.get("/").apply {
            assertEquals(HttpStatusCode.OK, status)
            assertEquals("Hello World!", bodyAsText())
        }
    }
}
```

Use client instance interaction with the TestApplication

```kotlin
import kotlin.test.*

class ApplicationTest {
    @Test
    fun testRoot() = testApplication {
        application {
            routing {
                get("/") {
                    call.respondText("Hello World!")
                }
            }
        }
        client.get("/").apply {
            assertEquals(HttpStatusCode.OK, status)
            assertEquals("Hello World!", bodyAsText())
        }
    }
}
```

```kotlin
import kotlin.test.*

class ApplicationTest {
    @Test
    fun testRoot() = testApplication {
        application {
            routing {
                get("/") {
                    call.respondText("Hello World!")
                }
            }
        }
        client.get("/").apply {
            assertEquals(HttpStatusCode.OK, status)
            assertEquals("Hello World!", bodyAsText())
        }
    }
}
```

It creates an application instance for each test.

What if we want to reuse the application instance for several tests?

```
class ApplicationTest {

    val testApp = TestApplication {
        application {
            ...
        }
    }

    val client = testApp.createClient {}

    @Test
    fun testRoot() = runBlocking<Unit> {
        val response = client.get("/")
        assertEquals(HttpStatusCode.OK, response.status)
        assertEquals("Hello World!", response.bodyAsText())
    }
}
```

```
class ApplicationTest {

    val testApp = TestApplication {
        application {
            ...
        }
    }

    val client = testApp.createClient {}

    @Test
    fun testRoot() = runBlocking<Unit> {
        val response = client.get("/")
        assertEquals(HttpStatusCode.OK, response.status)
        assertEquals("Hello World!", response.bodyAsText())
    }
}
```

Alternatively, create TestApplication instance

```
class ApplicationTest {

    val testApp = TestApplication {
        application {
            ...
        }
    }

    val client = testApp.createClient {}

    @Test
    fun testRoot() = runBlocking<Unit> {
        val response = client.get("/")
        assertEquals(HttpStatusCode.OK, response.status)
        assertEquals("Hello World!", response.bodyAsText())
    }
}
```

Configure HttpClient for the TestApplication

```
class ApplicationTest {

    val testApp = TestApplication {
        application {
            ...
        }
    }

    val client = testApp.createClient {}

    @Test
    fun testRoot() = runBlocking<Unit> {
        val response = client.get("/")
        assertEquals(HttpStatusCode.OK, response.status)
        assertEquals("Hello World!", response.bodyAsText())
    }
}
```

Need to use `runBlocking` to run in a coroutine scope

```
class ApplicationTest {

    val testApp = TestApplication {
        application {
            ...
        }
    }

    val client = testApp.createClient {
        install(io.ktor.client.plugins.contentnegotiation.ContentNegotiation) {
            json()
        }
    }

    @Test
    fun testRoot() = runBlocking<Unit> {
        ...
```

ContentNegotiation needs to be configured for the HttpClient

```
class ApplicationTest {

    val testApp = TestApplication {
        application {
            ...
        }
    }

    val client = testApp.createClient {
        install(io.ktor.client.plugins.contentnegotiation.ContentNegotiation) {
            json()
        }
    }

    @Test
    fun testRoot() = runBlocking<Unit> {
        ...
```

ContentNegotiation needs to be configured for the HttpClient

```
implementation("io.ktor:ktor-server-content-negotiation:$ktor_version")
implementation("io.ktor:ktor-client-content-negotiation:$ktor_version")
```

HttpClient

https://ktor.io/docs/client-requests.html

```
private val testApp = TestApplication { ... }

private val client = testApp.createClient { ... }

val response: HttpResponse = client.get("/data")
assertEquals(HttpStatusCode.OK, response.status)
assertEquals(..., response.bodyAsText())
```

Make request, check response

HttpClient

https://ktor.io/docs/client-requests.html

```
private val testApp = TestApplication { ... }

private val client = testApp.createClient { ... }

val response: HttpResponse = client.get("/data")
assertEquals(HttpStatusCode.OK, response.status)
assertEquals(..., response.bodyAsText())

val data = Json.decodeFromString<List<User>>(bodyAsText())
assertEquals(..., data.size)
```

Make request, check response

Deserialize response body into objects

HttpClient

https://ktor.io/docs/client-requests.html

```
private val testApp = TestApplication { ... }

private val client = testApp.createClient { ... }

val response: HttpResponse = client.get("/data")
assertEquals(HttpStatusCode.OK, response.status)
assertEquals(..., response.bodyAsText())

val data = Json.decodeFromString<List<User>>(bodyAsText())
assertEquals(..., data.size)

val response = client.post("/data") {  this: HttpRequestBuilder
    contentType(ContentType.Application.Json)
    setBody(...)
}
```

Make request, check response

Deserialize response body into objects

POST with HttpRequestBuilder

Hands-on time!

Implement endpoints for CRUD operations for the User data class.

```
@Serializable
data class Customer(
    val id: Int,
    val name: String,
    val email: String,
    val createdAt: Instant
)

@Serializable
data class CreateCustomer(
      val name: String,
      val email: String)

@Serializable
data class UpdateCustomer(
      val name: String? = null, val email: String? = null)
```

Hands-on time!

Implement these tests

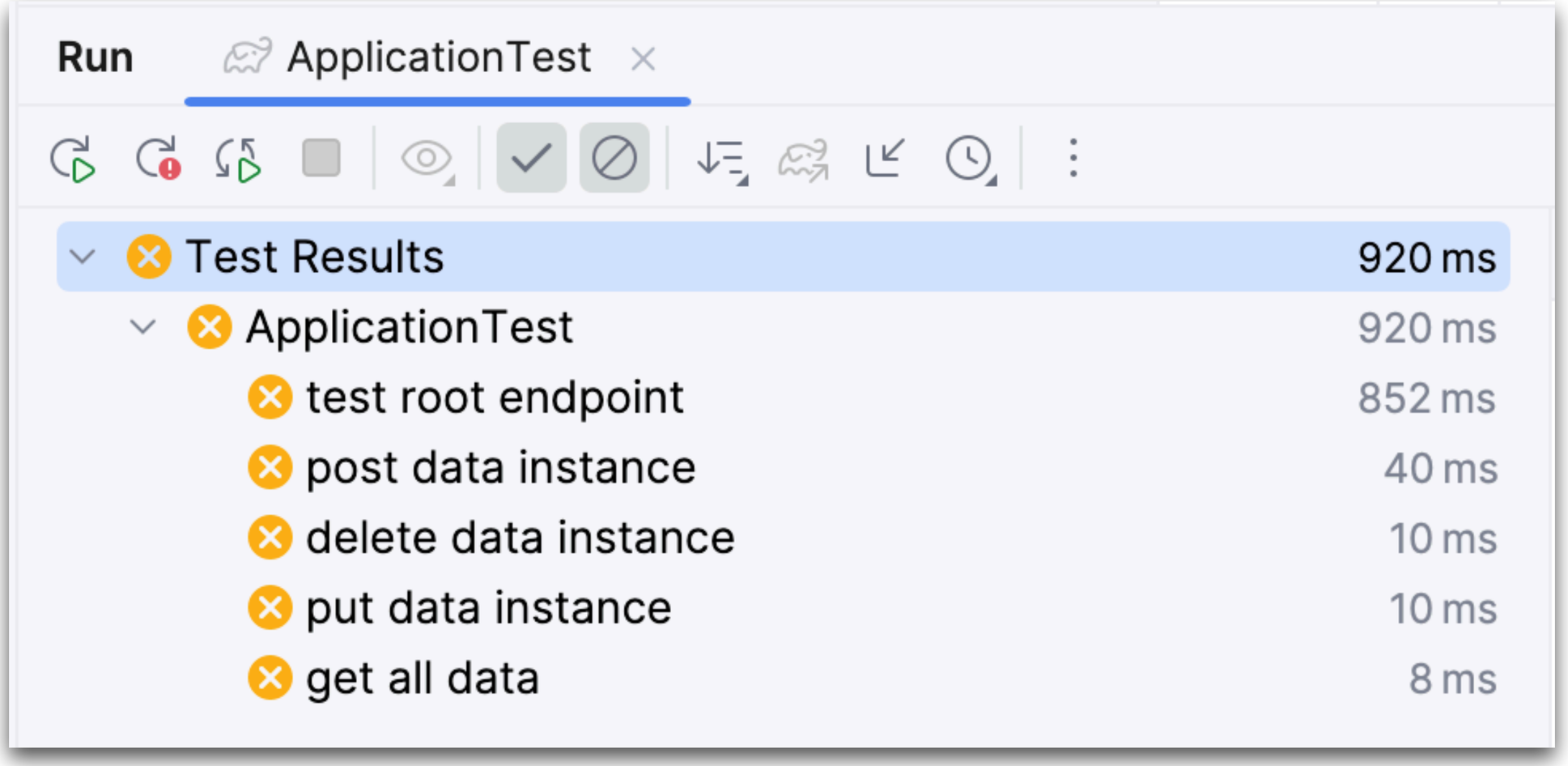
```
git branch
* branch01
```

Implement endpoints for CRUD operations for the User data class

```kotlin
@Serializable
data class Customer(
    val id: Int,
    val name: String,
    val email: String,
    val createdAt: Instant
)

@Serializable
data class CreateCustomer(
      val name: String,
      val email: String)

@Serializable
data class UpdateCustomer(
      val name: String? = null, val email: String? = null)
```

```kotlin
@Test
fun `get all customers`(): Unit = runBlocking {
    ...
}

@Test
fun `create customer`(): Unit = runBlocking {
    ...
}

@Test
fun `update customer`(): Unit = runBlocking {
    ...
}

@Test
fun `delete customer`(): Unit = runBlocking {
    ...
}
```

```
git branch
* branch02
```



Let's take a look at the solution

Receiving requests & sending responses

https://ktor.io/docs/server-requests.html

https://ktor.io/docs/server-responses.html

Sending responses

```
get("/") {
    call.respondText("Hello World!")
}
```

Respond something

Sending responses

```
get("/") {
    call.respondText("Hello World!")
}

post("/") {
    val user = call.receive<User>()
    ...
    call.respond(status = HttpStatusCode.Created, message = "Data added successfully")
}
```

Respond with HTTP code

Add response content

Receiving requests

```
get("/") {
    call.respondText("Hello World!")
}

post("/") {
    val user = call.receive<User>()
    ...
    call.respond(status = HttpStatusCode.Created, message = "Data added successfully")
}
```

Deserialize request content

Receiving requests

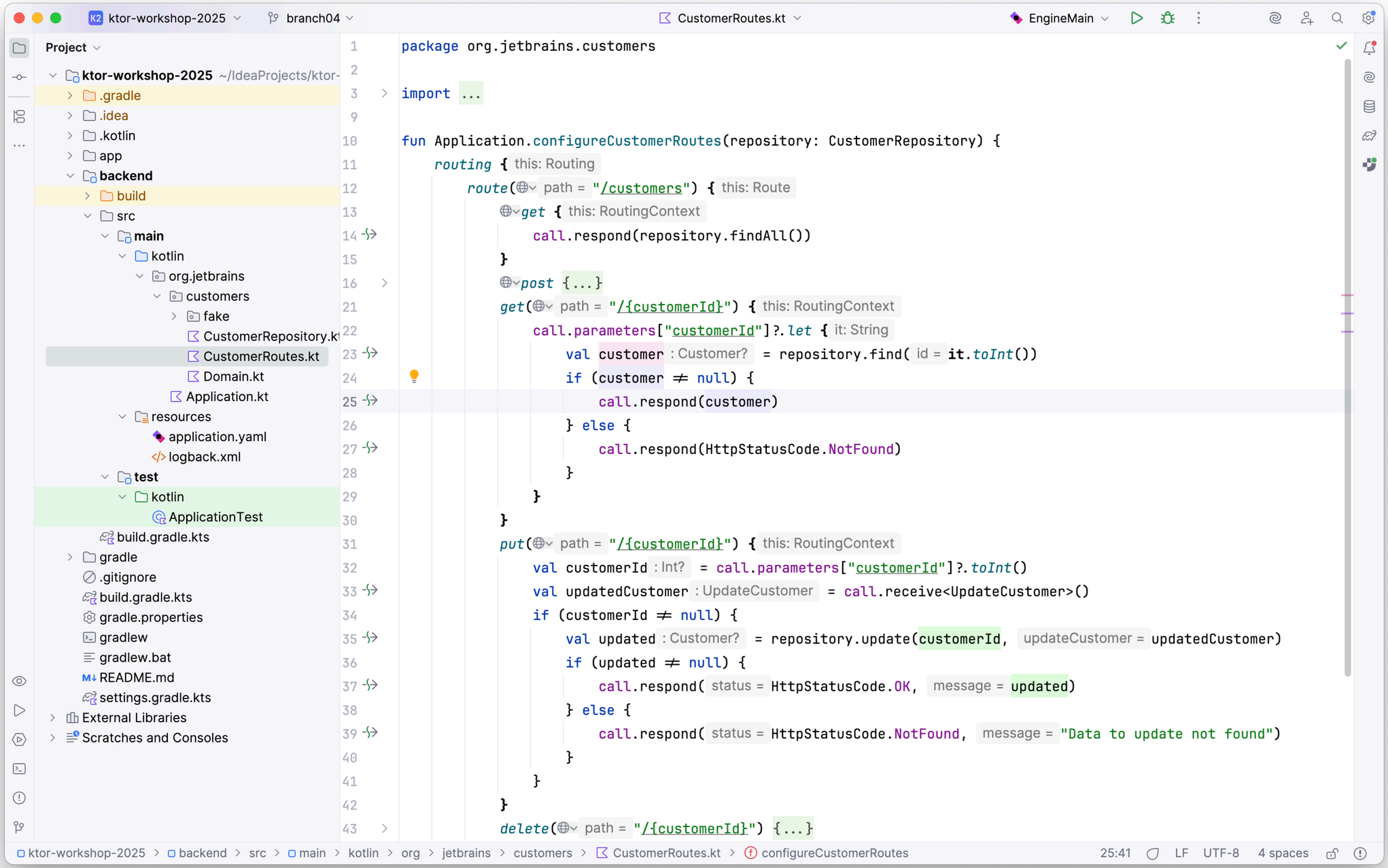
```
get("/") {
    call.respondText("Hello World!")
}

post("/") {
    val user = call.receive<User>()
    ...
    call.respond(status = HttpStatusCode.Created, message = "Data added successfully")
}

get("/{userId}") {
    val userId = call.parameters["userId"]
    ...
}
```

Access request parameters

Hands-on time!

Implement endpoints for CRUD operations for the User data class.

```
var customers = mutableListOf<Customer>()

routing {
    route("/customers") {
        get { ... }
        post { ... }
        get("/{customerId}") { ... }
        put("/{customerId}") { ... }
        delete("/{customerId}") { ... }
    }
}
```



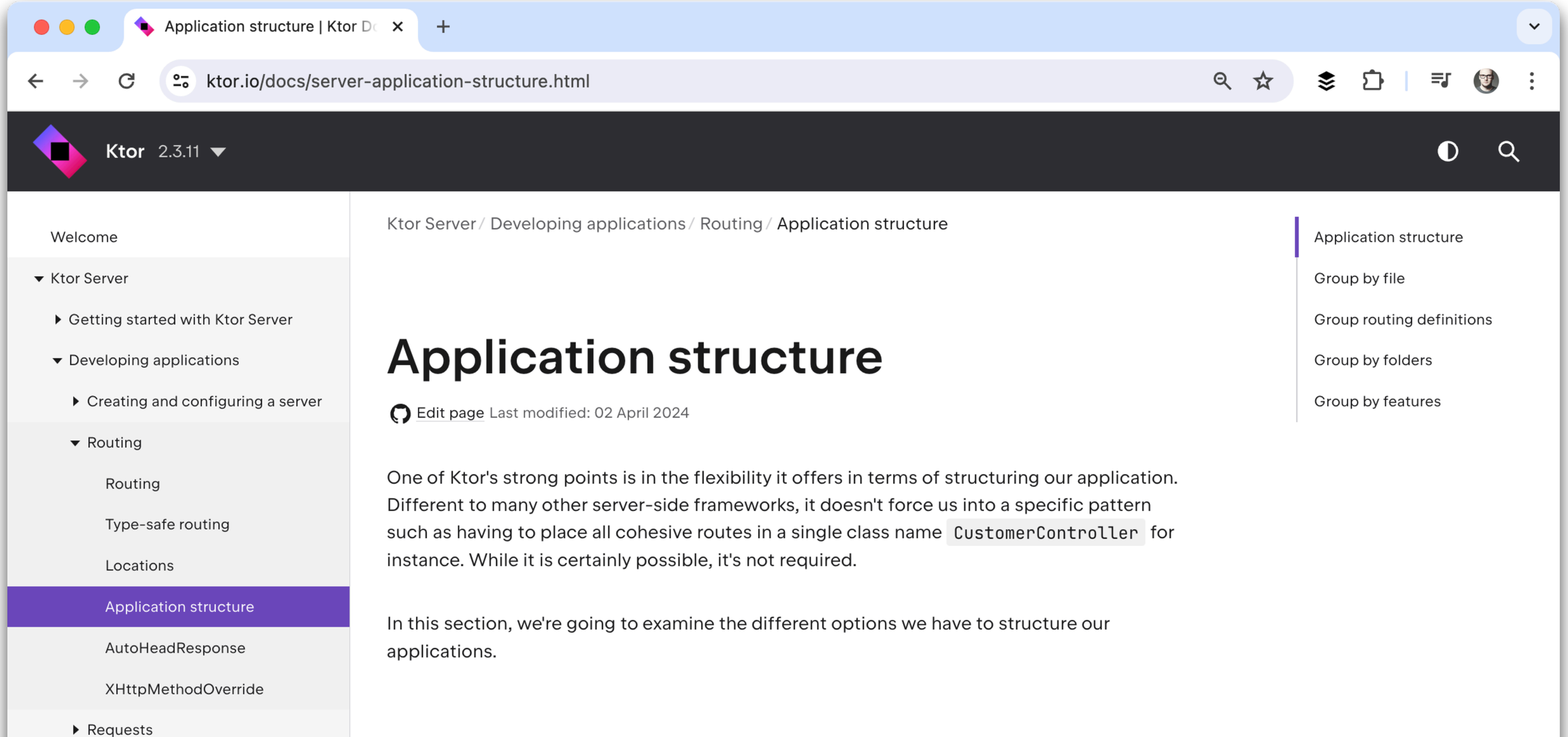

```
git branch
* branch03
```

```
git branch
* branch03
```

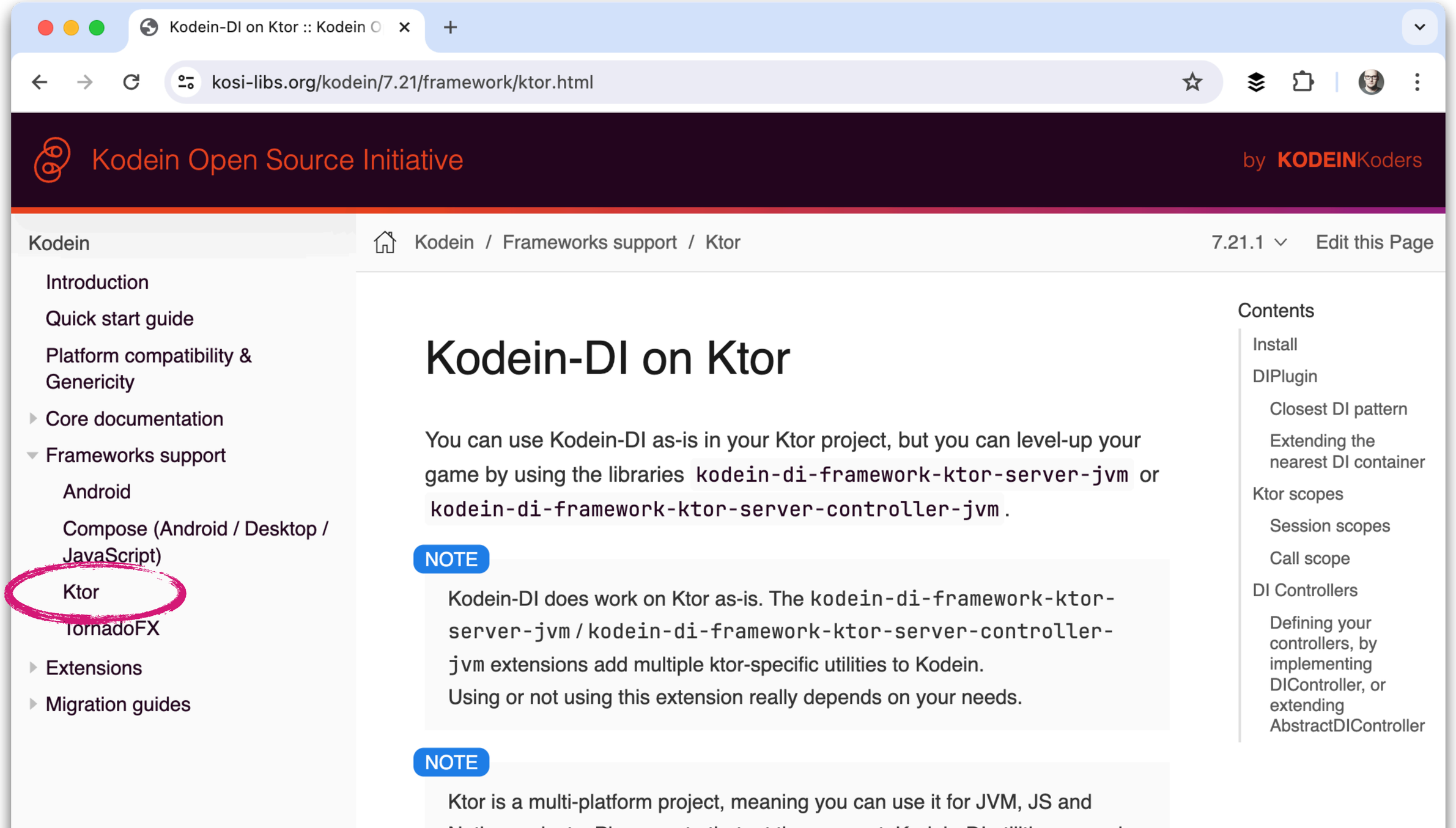
Let's take a look at the solution

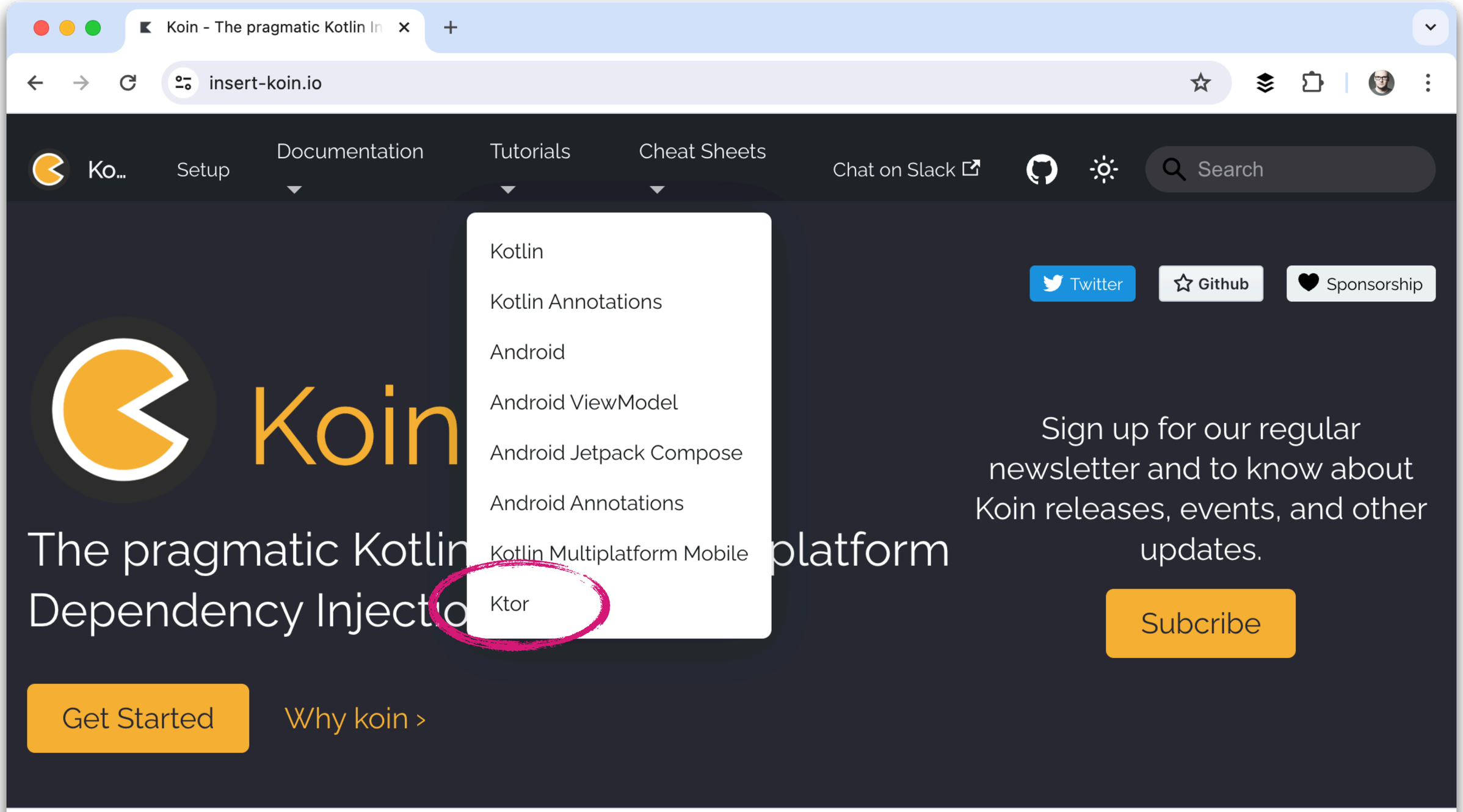
Structuring the project

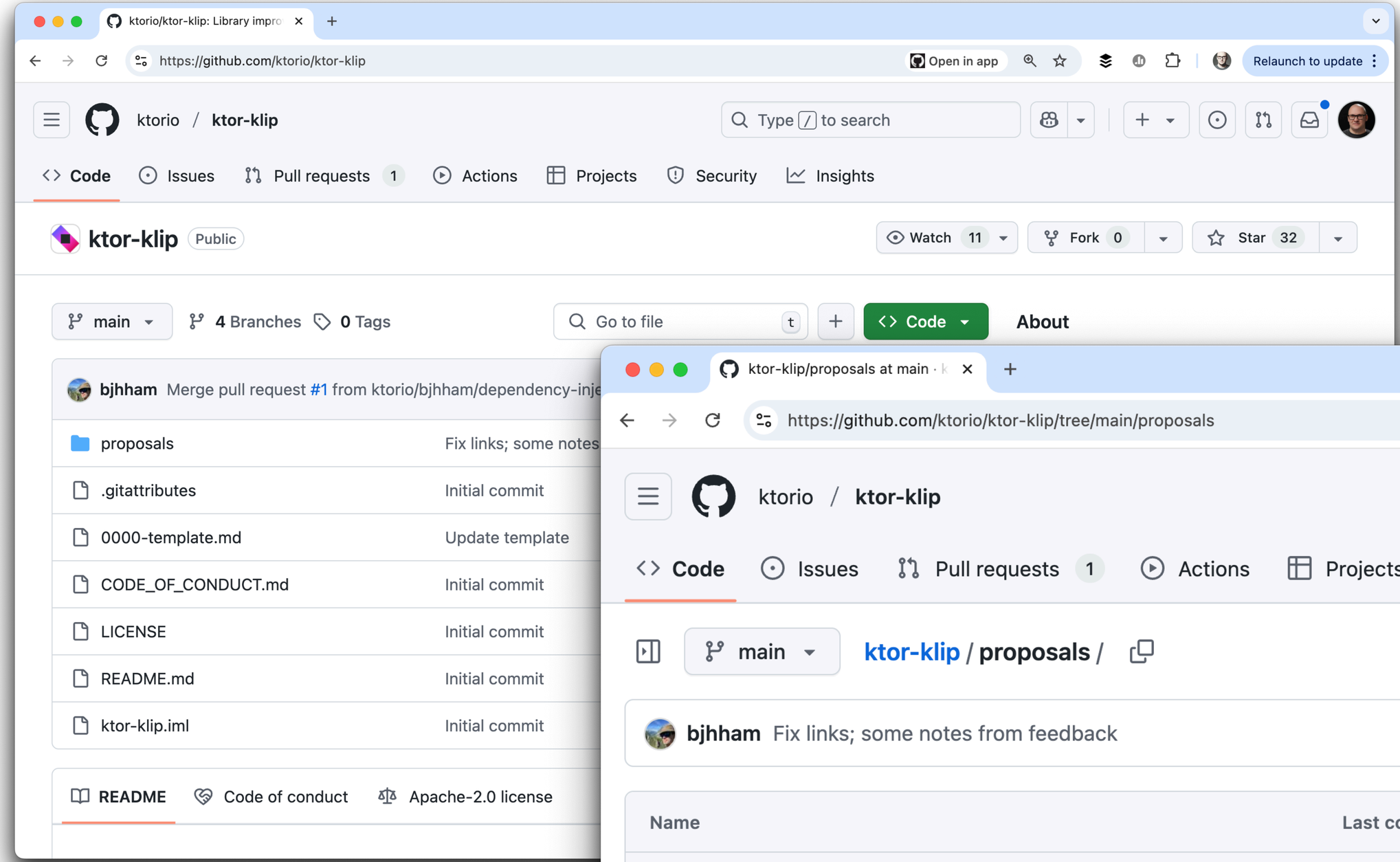
https://ktor.io/docs/server-application-structure.html



Dependency injection


Kodein-DI on Ktor :: Kodein
kosi-libs.org/kodein/7.21/framework/ktor.html
Kodein Open Source Initiative
by KODEINKoders
Kodein
Introduction
Quick start guide
Platform compatibility & Genericity
Core documentation
Frameworks support
Android
Compose (Android / Desktop / JavaScript)
Ktor
TornadoFX
Extensions
Migration guides
Kodein / Frameworks support / Ktor
7.21.1
Edit this Page
Kodein-DI on Ktor
You can use Kodein-DI as-is in your Ktor project, but you can level-up your game by using the libraries kodein-di-framework-ktor-server-jvm or kodein-di-framework-ktor-server-controller-jvm.
NOTE
Kodein-DI does work on Ktor as-is. The kodein-di-framework-ktor-server-jvm / kodein-di-framework-ktor-server-controller-jvm extensions add multiple ktor-specific utilities to Kodein.
Using or not using this extension really depends on your needs.
NOTE
Ktor is a multi-platform project, meaning you can use it for JVM, JS and
Contents
Install
DIPlugin
Closest DI pattern
Extending the nearest DI container
Ktor scopes
Session scopes
Call scope
DI Controllers
Defining your controllers, by implementing DIController, or extending AbstractDIController


Koin - The pragmatic Kotlin In
insert-koin.io
Ko...
Setup
Documentation
Tutorials
Cheat Sheets
Chat on Slack
Search
Kotlin
Kotlin Annotations
Android
Android ViewModel
Android Jetpack Compose
Android Annotations
Kotlin Multiplatform Mobile
Ktor
Twitter
Github
Sponsorship
Koin
The pragmatic Kotlin
Dependency Injecto
platform
Sign up for our regular newsletter and to know about Koin releases, events, and other updates.
Subcribe
Get Started
Why koin >


ktorio/ktor-klip: Library impro
https://github.com/ktorio/ktor-klip
Open in app
Relaunch to update
ktorio / ktor-klip
Type / to search
Code
Issues
Pull requests 1
Actions
Projects
Security
Insights
ktor-klip
Public
Watch 11
Fork 0
Star 32
main
4 Branches
0 Tags
Go to file t
Code
About
bjhham Merge pull request #1 from ktorio/bjhham/dependency-inje
proposals Fix links; some notes
.gitattributes Initial commit
0000-template.md Update template
CODE_OF_CONDUCT.md Initial commit
LICENSE Initial commit
README.md Initial commit
ktor-klip.iml Initial commit
README
Code of conduct
Apache-2.0 license

ktor-klip/proposals at main · k
https://github.com/ktorio/ktor-klip/tree/main/proposals
ktorio / ktor-klip
Type / to search
Code
Issues
Pull requests 1
Actions
Projects
Security
Insights
main
ktor-klip / proposals /
bjhham Fix links; some notes from feedback
Name
Last commit message
..
0001-dependency-injection.md
Fix links; some notes from feedback

```
git branch
* branch04
* branch05
```

Let's take a look at the solution

Part 2. Persistence with Exposed

What database framework should I use with Kotlin (and Ktor)?


OpenJPA


HIBERNATE


eclipse link


jdbi


EXPOSED


MyBatis


jOOQ™


SQLDelight


spring®

EXPOSED 

https://github.com/JetBrains/Exposed

Adding Exposed: dependencies

build.gradle.kts

```
implementation(libs.bundles.exposed)
```

libs.versions.toml

```
[bundles]
exposed = [
    "exposed-jdbc",
    "exposed-kotlin-datetime",
    "hikari",
    "h2database"
]
```

Using Exposed: connecting to DB

```
val database = Database.connect(
    url = "jdbc:h2:mem:test;DB_CLOSE_DELAY=-1",
    user = "root",
    driver = "org.h2.Driver", //optional
    password = ""
)
```

Using Exposed: connecting to DB

```
val database = Database.connect(
    HikariDataSource(HikariConfig().apply {
        jdbcUrl = "jdbc:h2:mem:test;DB_CLOSE_DELAY=-1"
        username = "root"
        driverClassName = "org.h2.Driver"
        password = ""
    })
)
```

Using Exposed: connecting to DB

```
val database = Database.connect(
    HikariDataSource(HikariConfig().apply {
        jdbcUrl = "jdbc:h2:mem:test;DB_CLOSE_DELAY=-1"
        username = "root"
        driverClassName = "org.h2.Driver"
        password = ""
    })
)

transaction(database) {
    SchemaUtils.createMissingTablesAndColumns(Datum)
}
```

```
@Serializable
data class Data(val id: Int, val text: String)

object Datum : Table("data") {
    val id = integer("id").autoIncrement()
    val text = varchar("text", 255)
}
```

```
@Serializable
data class Data(val id: Int, val text: String)

object Datum : IntIdTable("data", "data_id") {
    val text = varchar("text", 255)
}
```

```
@Serializable
data class Data(val id: Int, val text: String)

object Datum : Table("data") {
    val id = integer("id").autoIncrement()
    val text = varchar("text", 255)
}

Datum.selectAll().map { Data(it[Datum.id], it[Datum.text]) }
```

Select and map

```kotlin
@Serializable
data class Data(val id: Int, val text: String)

object Datum : Table("data") {
    val id = integer("id").autoIncrement()
    val text = varchar("text", 255)
}

Datum.selectAll().map { Data(it[Datum.id], it[Datum.text]) }

Datum.selectAll().where { Datum.id eq id }.singleOrNull()?.let {
    Data(it[Datum.id], it[Datum.text])
}
```

Select and map

```kotlin
@Serializable
data class Data(val id: Int, val text: String)

object Datum : Table("data") {
    val id = integer("id").autoIncrement()
    val text = varchar("text", 255)
}

Datum.selectAll().map { Data(it[Datum.id], it[Datum.text]) }

Datum.selectAll().where { Datum.id eq id }.singleOrNull()?.let {
    Data(it[Datum.id], it[Datum.text])
}

Datum.insert {
    it[Datum.id] = data.id
    it[Datum.text] = data.text
}

Datum.update({ Datum.id eq data.id }) { it[Datum.text] = data.text }
```

update / insert

Hands-on time! https://github.com/JetBrains/exposed

1. Define table objects for the `Customer`

2. Implement `CustomerRepository` using Exposed with H2

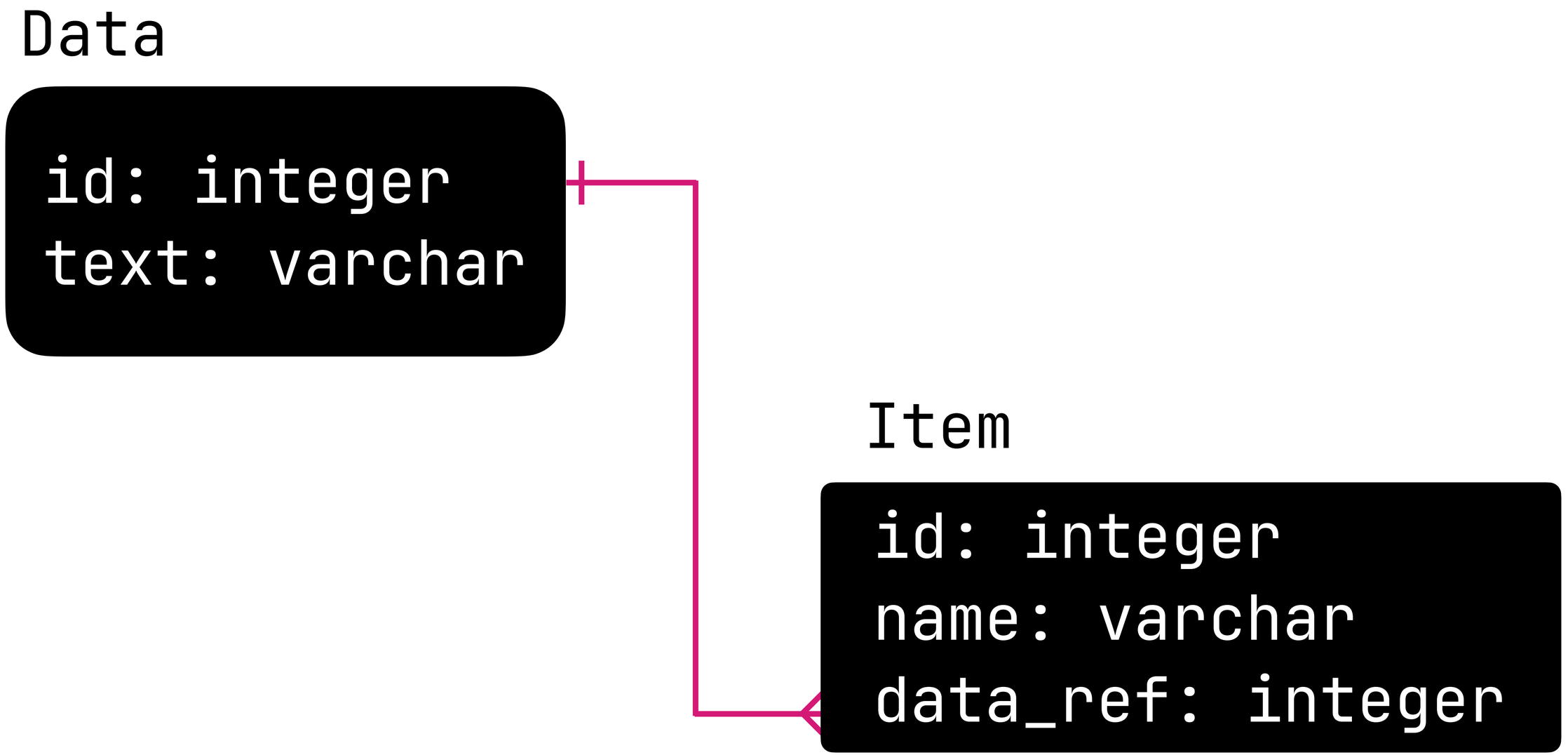
3. Integrate the new implementation into Ktor application using DI

```
object Customers : IntIdTable("CUSTOMERS", "CUSTOMER_ID") {
    val name = varchar("NAME", 255)
    val email = varchar("EMAIL", 255).uniqueIndex()
    val createdAt = timestamp("CREATED_AT").defaultExpression(CurrentTimestamp)
}
```

```
git branch
* branch06-jdbc
```

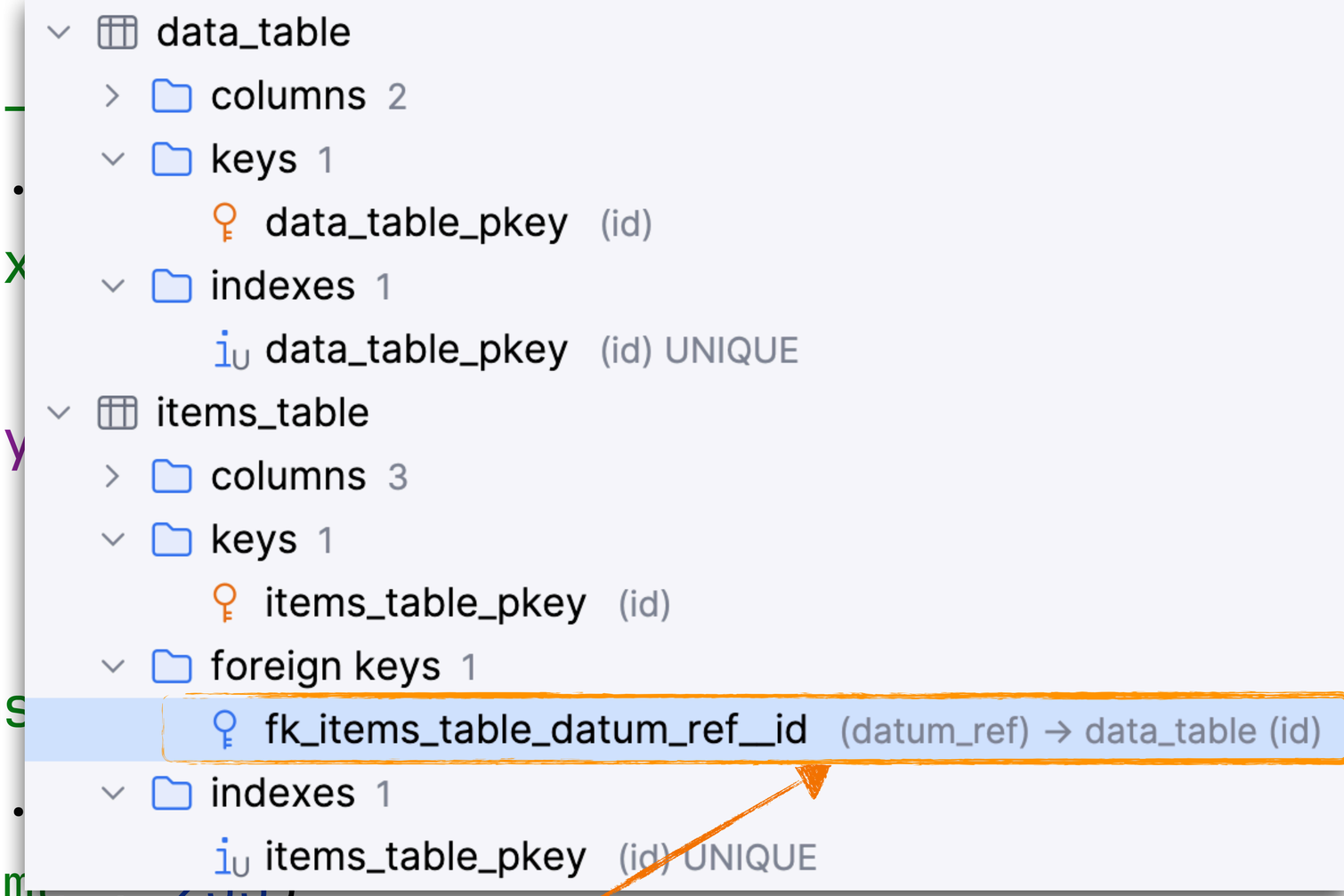
Let's take a look at the solution

Table relations


Data
id: integer
text: varchar
Item
id: integer
name: varchar
data_ref: integer

```kotlin
object Datum : Table("data_table") {
    val id = integer("id").autoIncrement()
    val text = varchar("text", 255)

    override val primaryKey = PrimaryKey(id)
}

object Items : Table("items_table") {
    val id = integer("id").autoIncrement()
    val name = varchar("name", 255)
    val datum_ref = reference("datum_ref", Datum.id)

    override val primaryKey = PrimaryKey(id)
}
```

```
object Datum : Table("data_
    val id = integer("id").
    val text = varchar("tex

    override val primaryKey
}

object Items : Table("items
    val id = integer("id").
    val name = varchar("name", 255)
    val datum_ref = reference("datum_ref", Datum.id)

    override val primaryKey = PrimaryKey(id)
}
```



```
val dataId = Datum.insert {
    it[Datum.text] = data.text
} get Datum.id

Items.insert {
    it[Items.name] = item.name
    it[Items.datum_ref] = dataId
}
```

```
val dataId = Datum.insert {
    it[Datum.text] = data.text
} get Datum.id

Items.insert {
    it[Items.name] = item.name
    it[Items.datum_ref] = dataId
}
```

```
val data = Data("Some data")

val item = Item("Some item")
```

Hands-on time!

Update the `CustomerRepository` implementation to handle the relations

```
object Customers : IntIdTable("CUSTOMERS", "CUSTOMER_ID") {
    val name = varchar("NAME", 255)
    val email = varchar("EMAIL", 255).uniqueIndex()
    val createdAt = timestamp("CREATED_AT").defaultExpression(CurrentTimestamp)
}

object Bookings : IntIdTable("BOOKINGS", "BOOKING_ID") {
    val customerId = reference("CUSTOMER_ID", Customers)
    val bookingDate = timestamp("BOOKING_DATE").defaultExpression(CurrentTimestamp)
    val amount = double("AMOUNT")
}
```

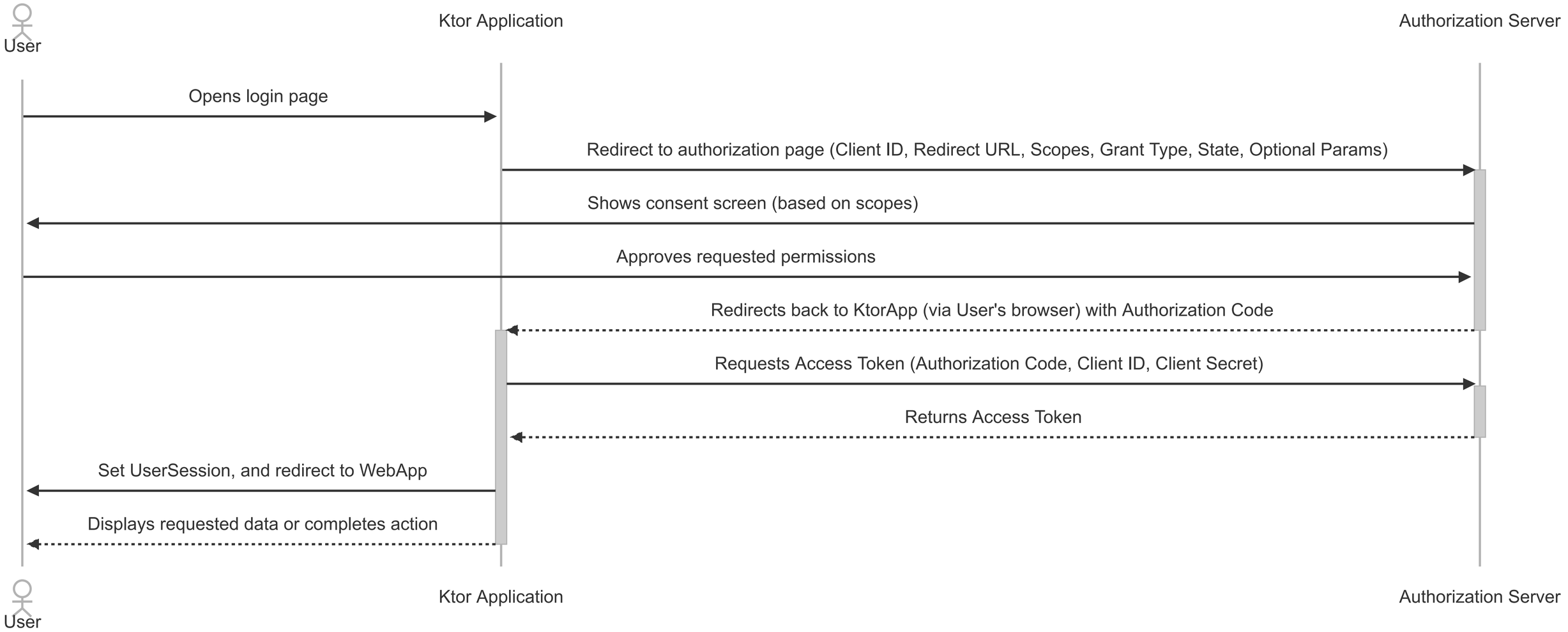
Update data classes and application code

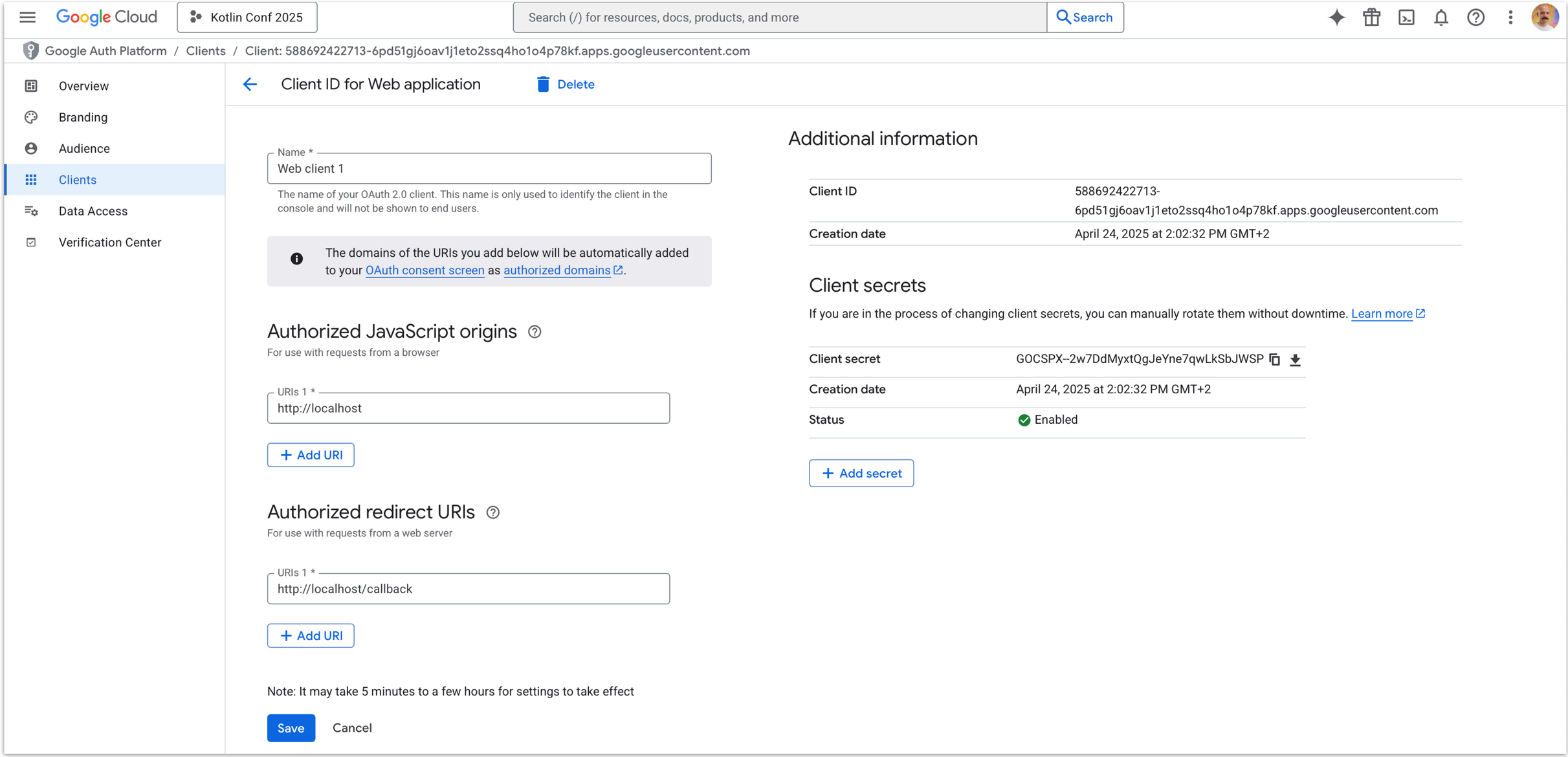
```
git branch
* branch06
```

Let's take a look at the solution

Part 3. Authentication

OAuth2


User
Ktor Application
Authorization Server
Opens login page
Redirect to authorization page (Client ID, Redirect URL, Scopes, Grant Type, State, Optional Params)
Shows consent screen (based on scopes)
Approves requested permissions
Redirects back to KtorApp (via User's browser) with Authorization Code
Requests Access Token (Authorization Code, Client ID, Client Secret)
Returns Access Token
Set UserSession, and redirect to WebApp
Displays requested data or completes action
User
Ktor Application
Authorization Server


Google Cloud
Kotlin Conf 2025
Search (/) for resources, docs, products, and more
Search
Google Auth Platform / Clients / Client: 588692422713-6pd51gj6oav1j1eto2ssq4ho1o4p78kf.apps.googleusercontent.com
Overview
Branding
Audience
Clients
Data Access
Verification Center
Client ID for Web application
Delete
Name *
Web client 1
The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.
The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorized domains.
Authorized JavaScript origins
For use with requests from a browser
URIs 1 *
http://localhost
Add URI
Authorized redirect URIs
For use with requests from a web server
URIs 1 *
http://localhost/callback
Add URI
Note: It may take 5 minutes to a few hours for settings to take effect
Save
Cancel
Additional information
Client ID
588692422713-6pd51gj6oav1j1eto2ssq4ho1o4p78kf.apps.googleusercontent.com
Creation date
April 24, 2025 at 2:02:32 PM GMT+2
Client secrets
If you are in the process of changing client secrets, you can manually rotate them without downtime. Learn more
Client secret
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
Creation date
April 24, 2025 at 2:02:32 PM GMT+2
Status
Enabled
Add secret

ktor-workshop.env

AUTH_CLIENT_ID=****
AUTH_CLIENT_SECRET=***

Non-secrets

```
redirectUrl: "http://localhost:8000/callback"
authorizeUrl: "https://accounts.google.com/o/oauth2/auth"
accessTokenUrl: "https://oauth2.googleapis.com/token"
```

Hosting Static Content

```
fun Routing.configureChatRoutes() {
    get("/") {
        val hasSession = call.sessions.get<UserSession>() ≠ null
        val redirectUrl = if (hasSession) "/home" else "login"
        call.respondRedirect(redirectUrl)
    }

    staticResources("/", "web")
    staticResources("/login", "web")
    staticResources("/home", "web") {
        modify { _, call →
            if (call.sessions.get<UserSession>() == null) call.respondRedirect("/login")
        }
    }
}
```

Sessions

```
@Serializable data class UserSession(val email: String)

fun Application.configureSession(config: AuthConfig) {
    val shouldBeSecure = !developmentMode
    install(Sessions) {
        cookie<UserSession>("SESSION") {
            cookie.secure = shouldBeSecure
            cookie.extensions["SameSite"] = "lax"
            cookie.maxAge = 5.minutes
            cookie.httpOnly = true
            transform(
              SessionTransportTransformerEncrypt(
                  hex(config.encryptionKey),
                  hex(config.signKey)
              )
            )
        }
    }
}
```

OAuth - exercise

AUTH_CLIENT_ID=*****
AUTH_CLIENT_SECRET=*****

```
redirectUrl: “http://localhost:8080/callback"
authorizeUrl: "https://accounts.google.com/o/oauth2/auth"
accessTokenUrl: "https://oauth2.googleapis.com/token"
```

Task:

1. Configure Authentication
2. Create authenticate routes (login & callback)
3. If User successfully logs in, set UserSession
4. Verify your an log into Chat using test account:
   ktor.user.kc25@gmail.com
   ktoruser25!

Docs: https://ktor.io/docs/server-oauth.html

JWT - additional exercise

Setup JWT for the Google OAuth2 issues id_token.
This requires using Json Web Key (JWK) Specification.

```
jwkUrl: "https://www.googleapis.com/oauth2/v3/certs"
issuer: "https://accounts.google.com"
```

Task:

1. Configure Authentication (JWT) with JWK
2. Validate credentials
3. If Credentials are valid set UserSession
4. Log in through Chat, and log/debug id_token
   ktor.user.kc25@gmail.com
   ktoruser25!
5. Test you can log into a route with JWT

Docs: https://ktor.io/docs/server-oauth.html

Part 4. LangChain4j

Reactive

Streaming

R2DBC

Flow

Server Sent Events (SSE)

WebSockets

```
interface TravelService {
    @SystemMessage(SYSTEM_MESSAGE)
    @UserMessage("{{question}}")
    fun answer(
      @MemoryId email: String,
      @V("question") question: String
    ): Flow<String>
}
```

SSE - exercise

AI_API_KEY=***

Task:

1. Configure SSE Plugin
2. Setup SSE Route
3. Check if User successfully logs in (JWT & UserSession)
4. Receive a Question / String
5. Delegate to TravelService & stream response to request
6. Log in through Chat, and log/debug id_token
   ktor.user.kc25@gmail.com
   ktoruser25!
7.

Docs: https://ktor.io/docs/server-oauth.html

@JetBrainsKtor

https://github.com/ktorio