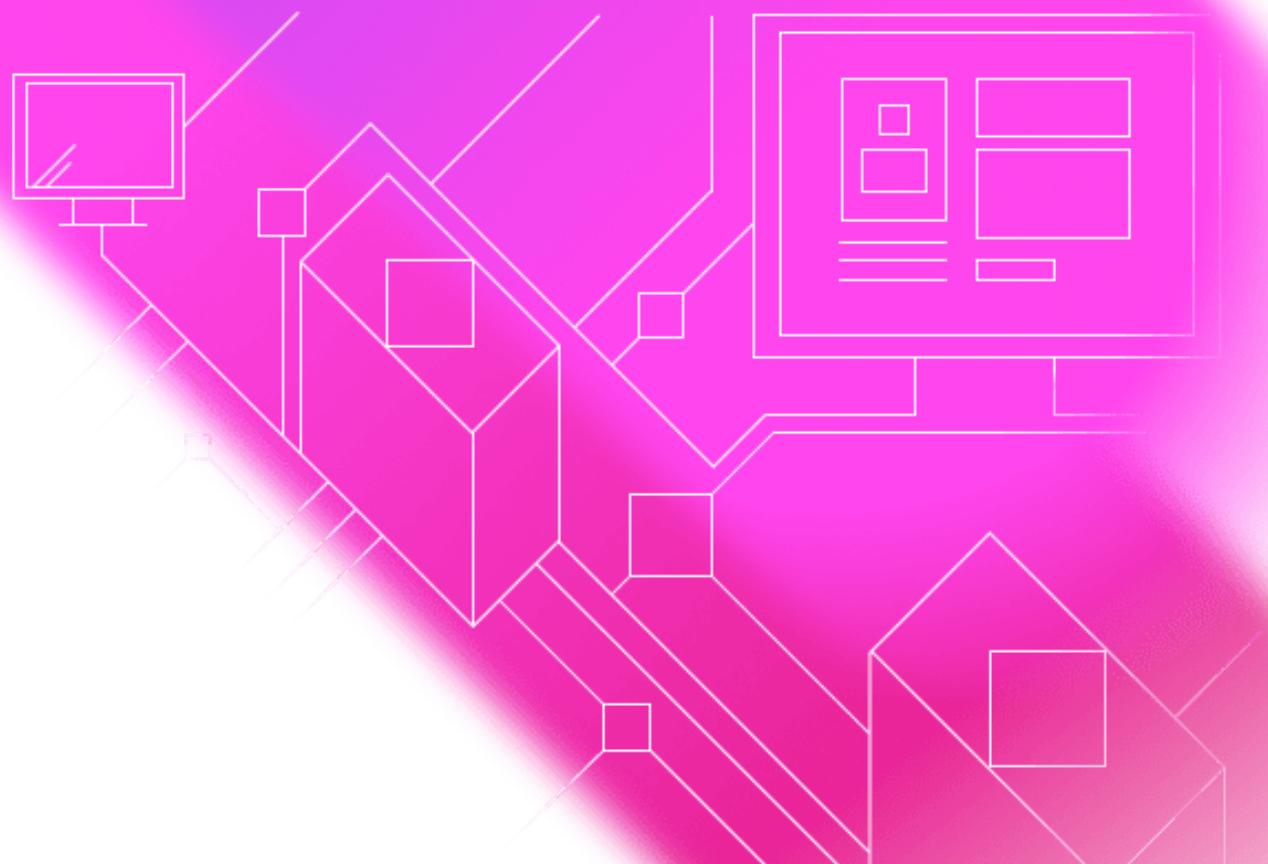


JETBRAINS

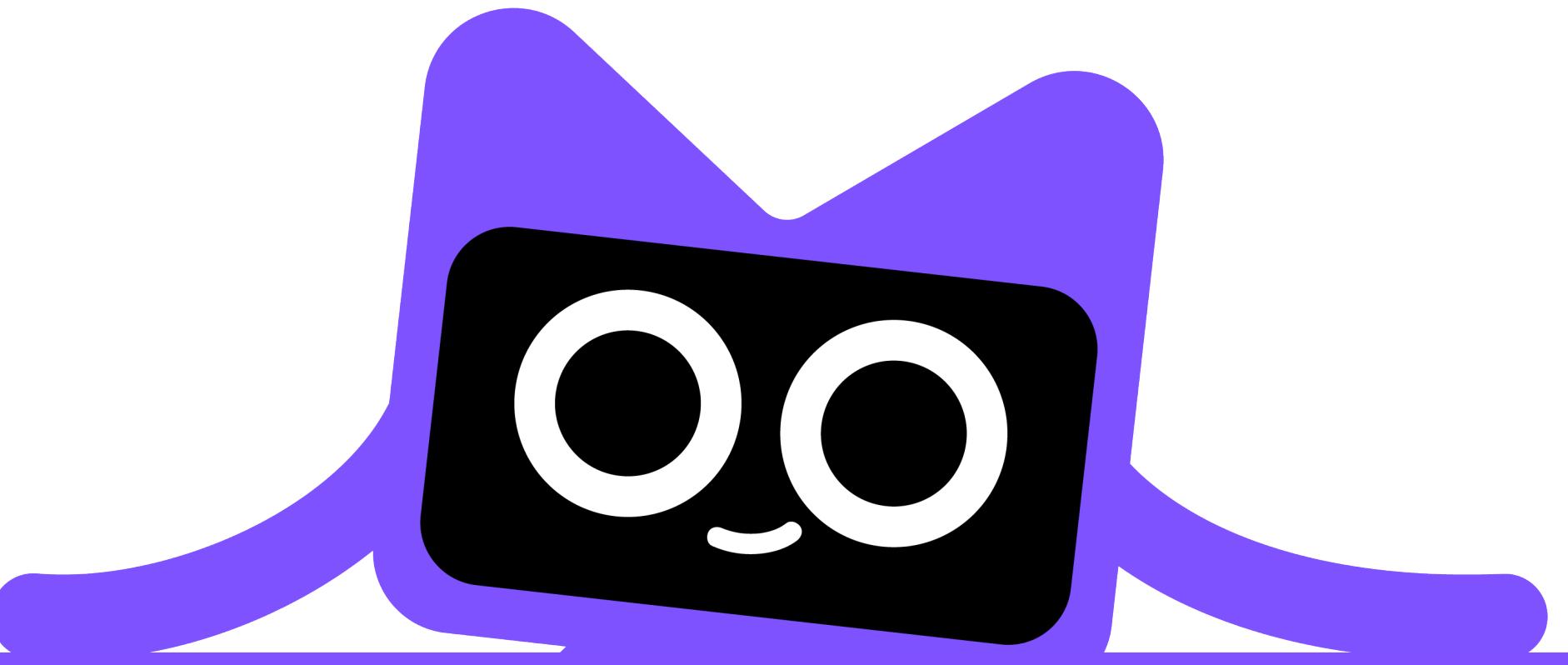
Full Stack Ktor

Simon Vergauwen
Developer Advocate

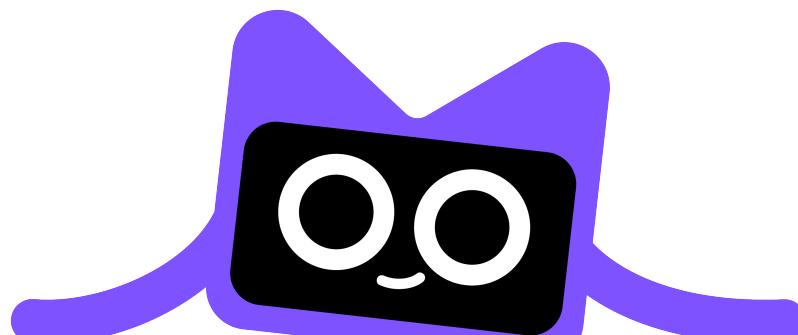
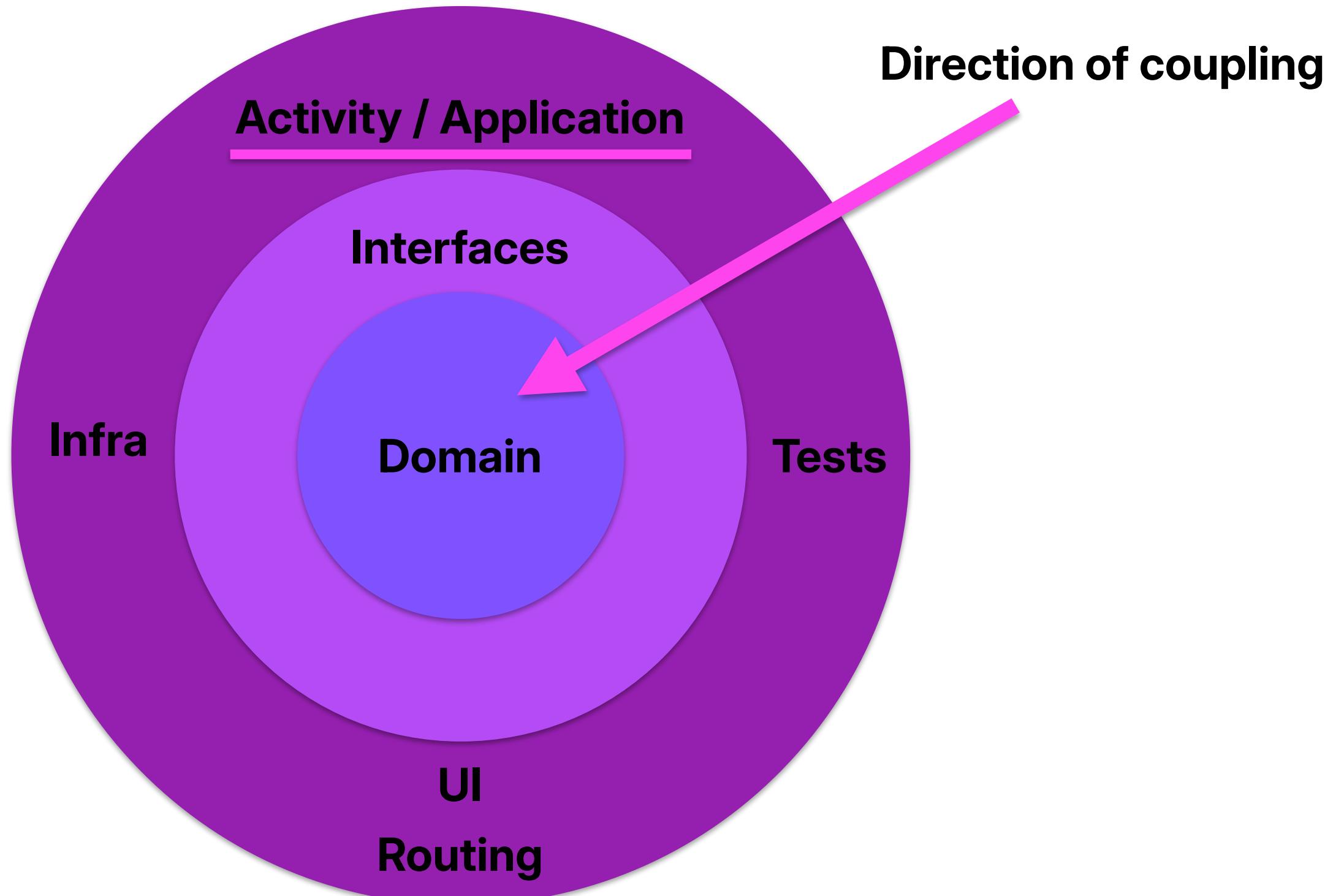


Why Ktor?

- Kotlin 1st
- Coroutines
- DSL
- Unopinionated on architecture
- Re-use and transfer existing knowledge!



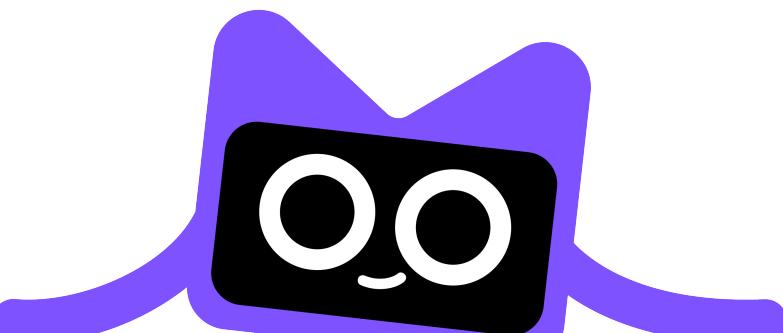
Architecture



Application.kt

```
fun main() {  
    embeddedServer(Netty, host = "0.0.0.0", port = 8080) {  
        // suspend Application.() → Unit  
    }.start(wait = true)  
}
```

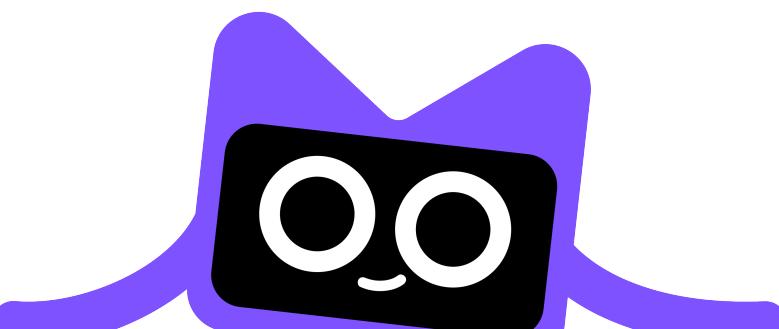
Android Emulator ⇒ **10.0.2.2:8080**



Java's main-class

```
plugins {  
    alias(libs.plugins.kotlin.jvm)  
    alias(ktorLibs.plugins.ktor)  
}  
application {  
    mainClass.set("org.jetbrains.demo.ApplicationKt")  
}
```

./gradlew run

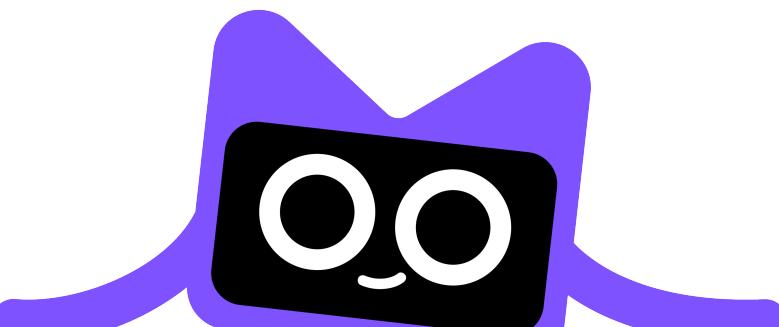


fatJar

A fatJar is a jar that includes all its dependencies

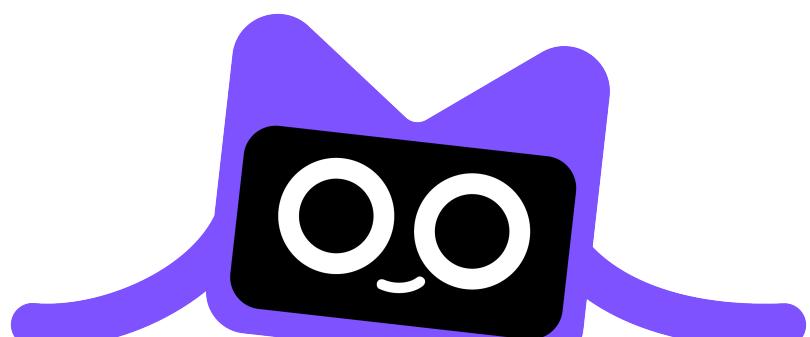
```
ktor {  
    fatJar {  
        allowZip64 = true  
        archiveFileName.set("gdg-capetown.jar")  
    }  
}
```

```
./gradlew buildFatJar  
java -jar build/libs/gdg-capetown.jar
```



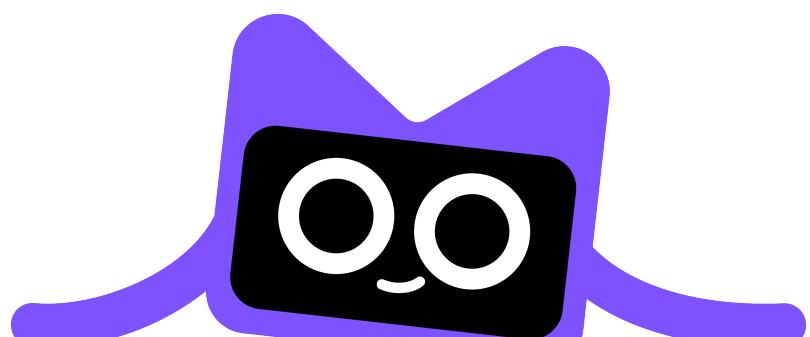
Docker image

```
ktor {  
    docker {  
        localImageName = "ktor-server"  
        imageTag = project.version.toString()  
        externalRegistry =  
            googleContainerRegistry(  
                projectName = provider { "GDG Capetown" },  
                appName = providers.environmentVariable("APP_ID"),  
                username = providers.environmentVariable("USERNAME"),  
                password = providers.environmentVariable("PASSWORD"),  
            )  
    }  
}
```



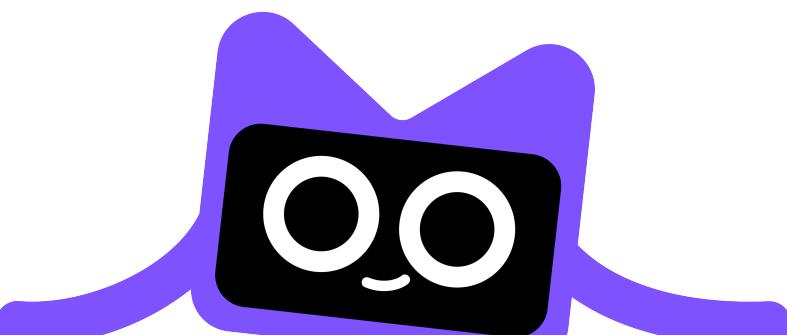
Gradle Version Catalog

```
dependencyResolutionManagement {  
    versionCatalogs {  
        create("ktorLibs") {  
            from("io.ktor:ktor-version-catalog:3.2.3")  
        }  
    }  
}
```



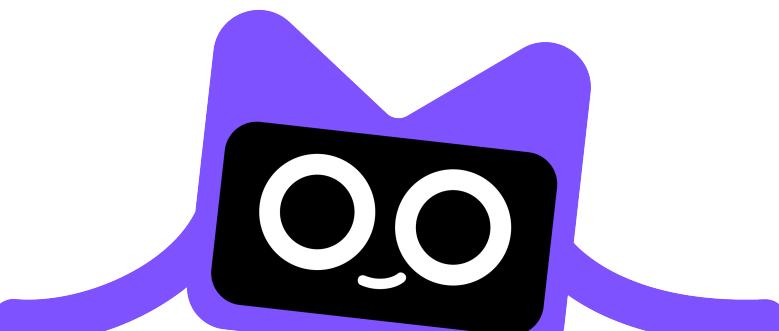
Gradle Version Catalog

```
dependencies {  
    implementation(ktorLibs.server.netty)  
    implementation(ktorLibs.server.config.yaml)  
    implementation(ktorLibs.server.auth.jwt)  
    implementation(ktorLibs.server.contentNegotiation)  
    implementation(ktorLibs.serialization.kotlinx.json)  
}
```



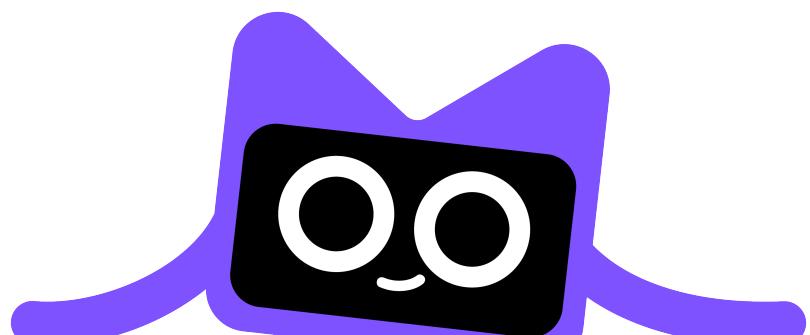
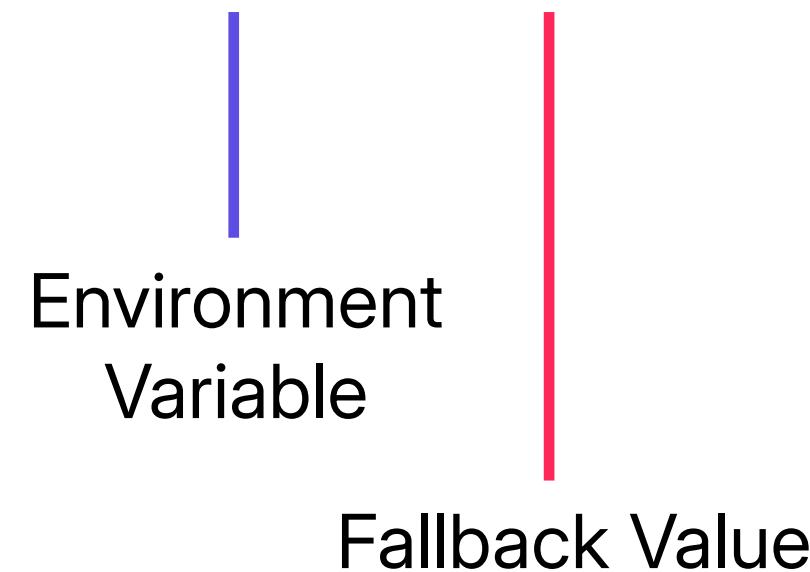
BuildConfig

```
android {  
    defaultConfig {  
        buildConfigField("String", "URL", "\"${property("URL")}\\"")  
    }  
}  
  
HttpClient(Android) {  
    defaultRequest {  
        url(BuildConfig.API_BASE_URL)  
    }  
}
```



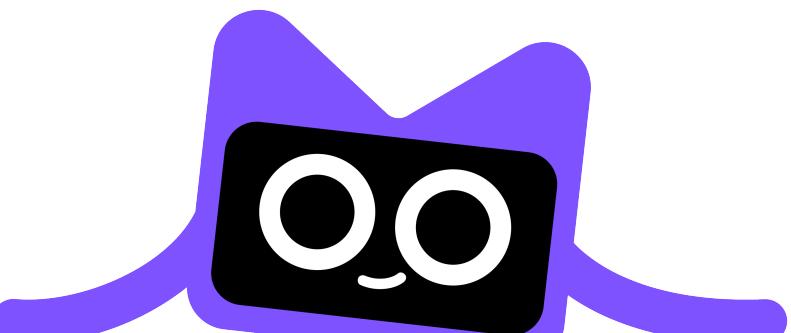
src/main/resources/application.yaml

```
app:  
  host: "$HOST:0.0.0.0"  
  port: "$PORT:8080"
```



AppConfig

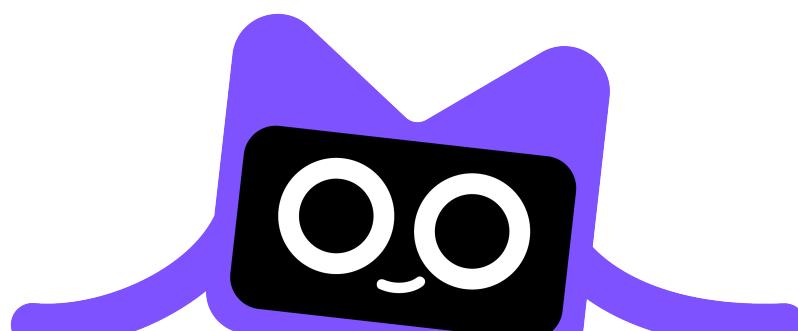
```
@Serializable  
data class AppConfig(val host: String, val port: Int)  
  
val config = ApplicationConfig("application.yaml")  
    .property("app")  
    .getAs<AppConfig>()
```



Application.kt

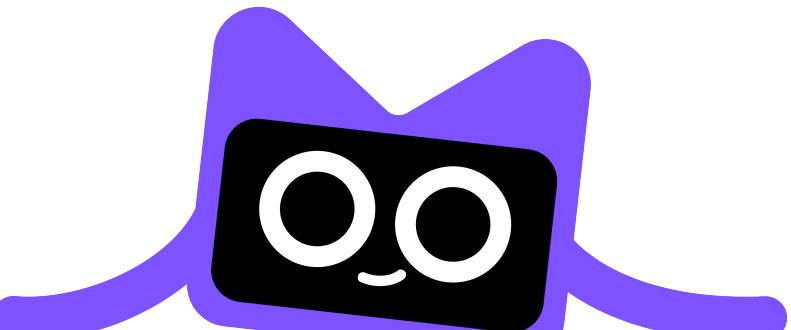
```
fun main() {  
    val config = ApplicationConfig("application.yaml")  
        .property("app")  
        .getAs<AppConfig>()  
  
    embeddedServer(Netty, host = config.host, port = config.port) {  
        // suspend Application.() -> Unit  
    }.start(wait = true)  
}
```

Yaml & HOCON Support



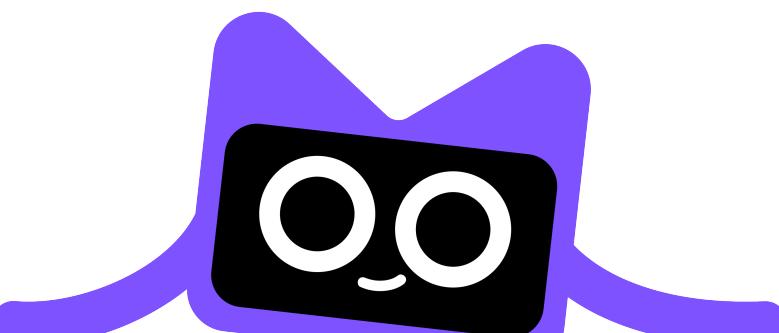
EmbeddedServer

```
fun main() {  
    val config = ApplicationConfig("application.yaml")  
        .property("app")  
        .getAs<AppConfig>()  
  
    embeddedServer(Netty, host = config.host, port = config.port) {  
        app(config)  
    }.start(wait = true)  
}  
  
suspend fun Application.app(config: AppConfig) { }
```

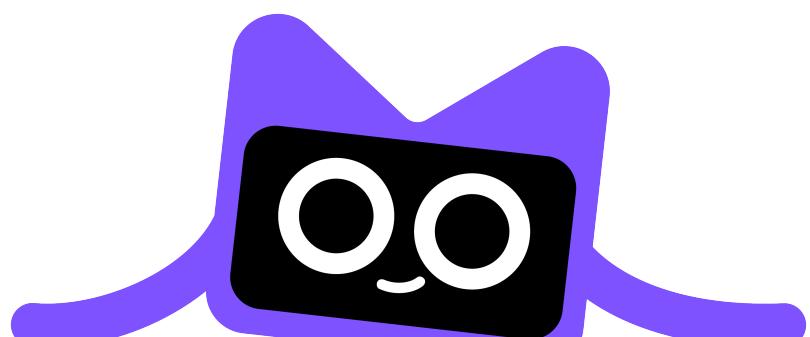
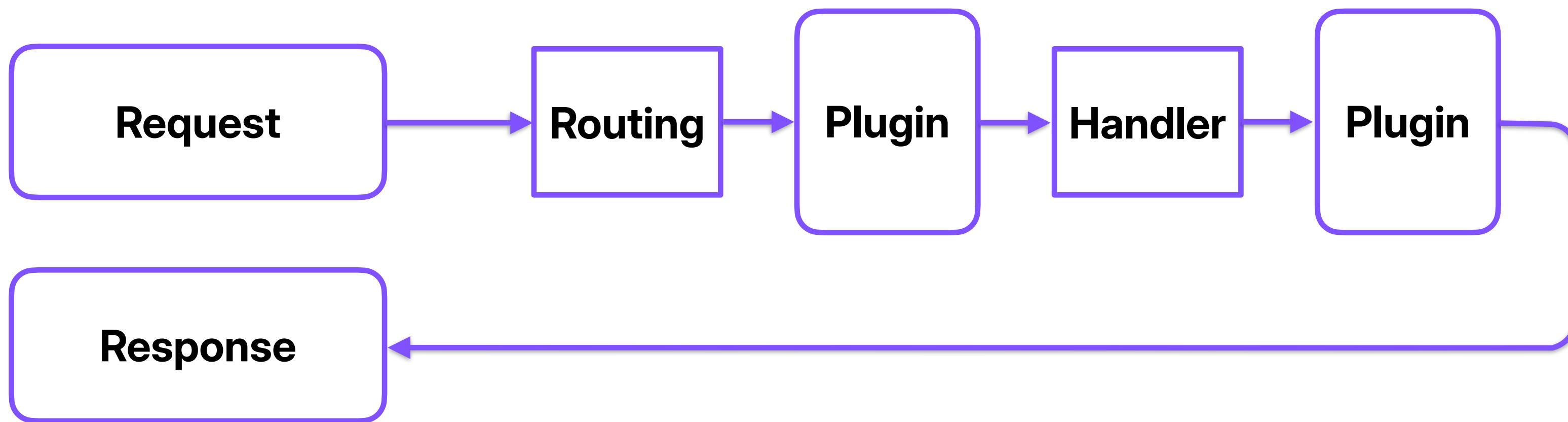


TestApplication

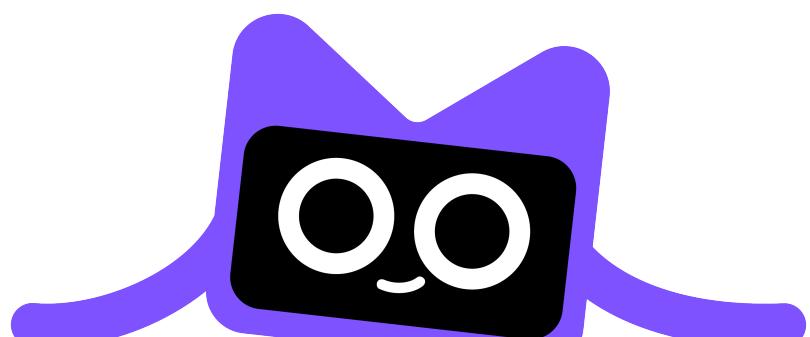
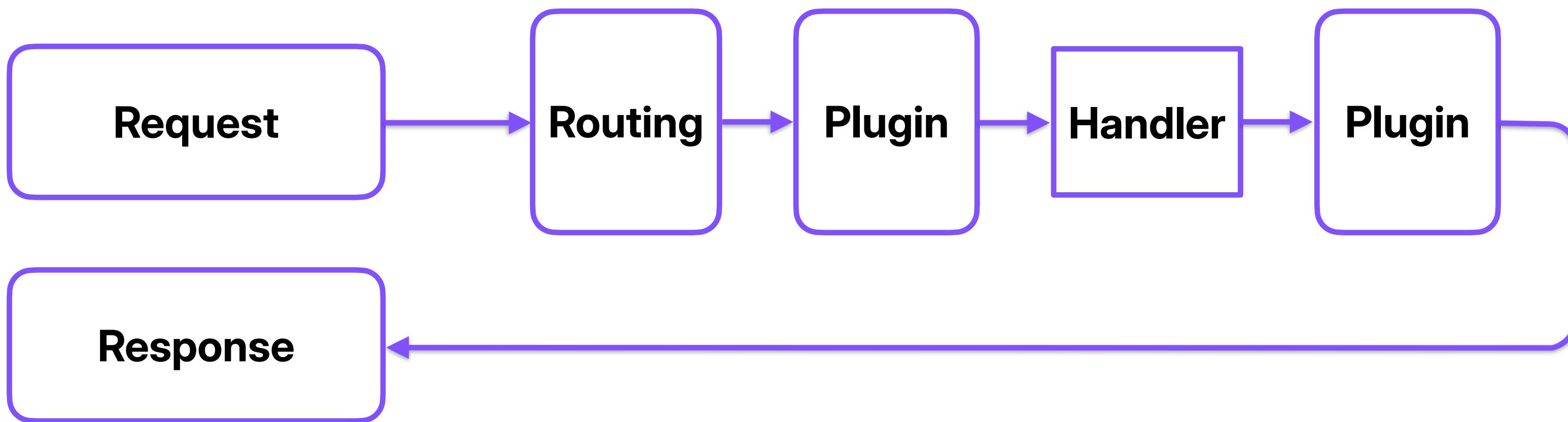
```
@Test
fun test() = testApplication {
    application {
        app(AppConfig(host = "not-used-in-test", port = 9999))
    }
    val client= createClient {
        install(ContentNegotiation) { json() }
    }
    val response = client.get("/call-some-endpoint")
    assertEquals(HttpStatusCode.OK, response.status)
}
```



App - plugins

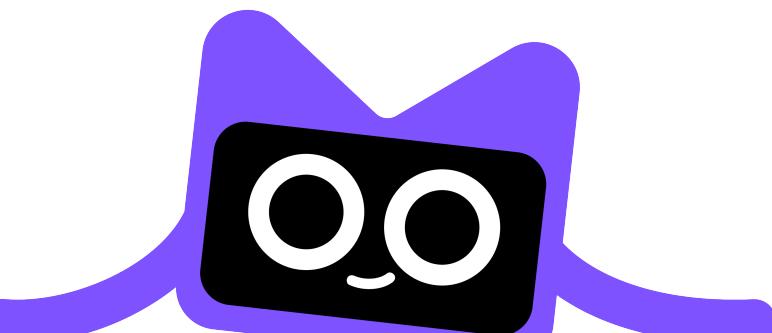


App - plugins



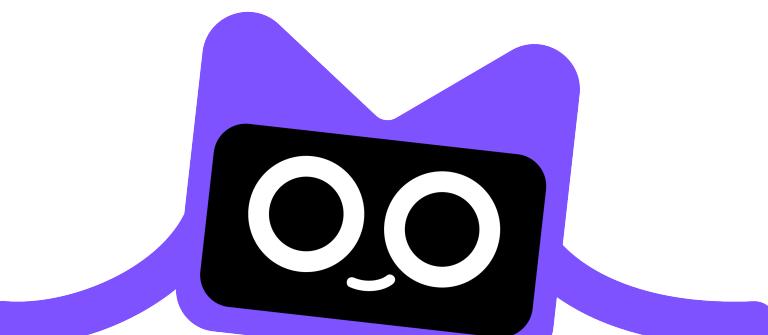
App - plugins

```
suspend fun Application.app(config: AppConfig) {  
    install(ContentNegotiation) {  
        json(Json {  
            encodeDefaults = true  
            isLenient = true  
        })  
    }  
}
```



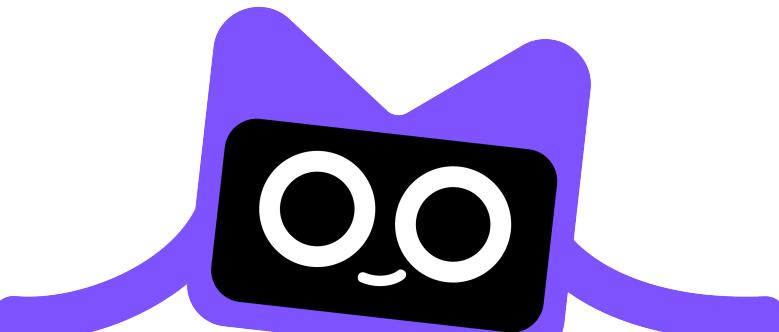
App - plugins

```
suspend fun Application.app(config: AppConfig) {  
    install(ContentNegotiation) {  
        json(Json {  
            encodeDefaults = true  
            isLenient = true  
        })  
    }  
    if (developmentMode) install(CallLogging)  
}
```

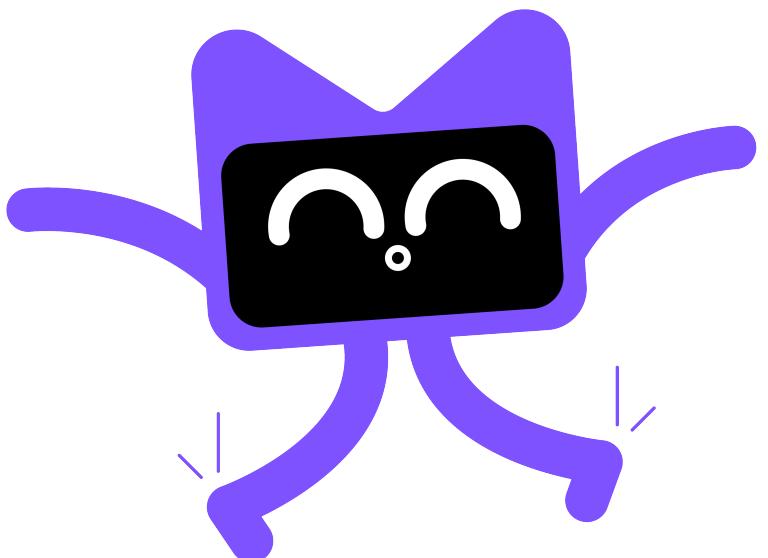
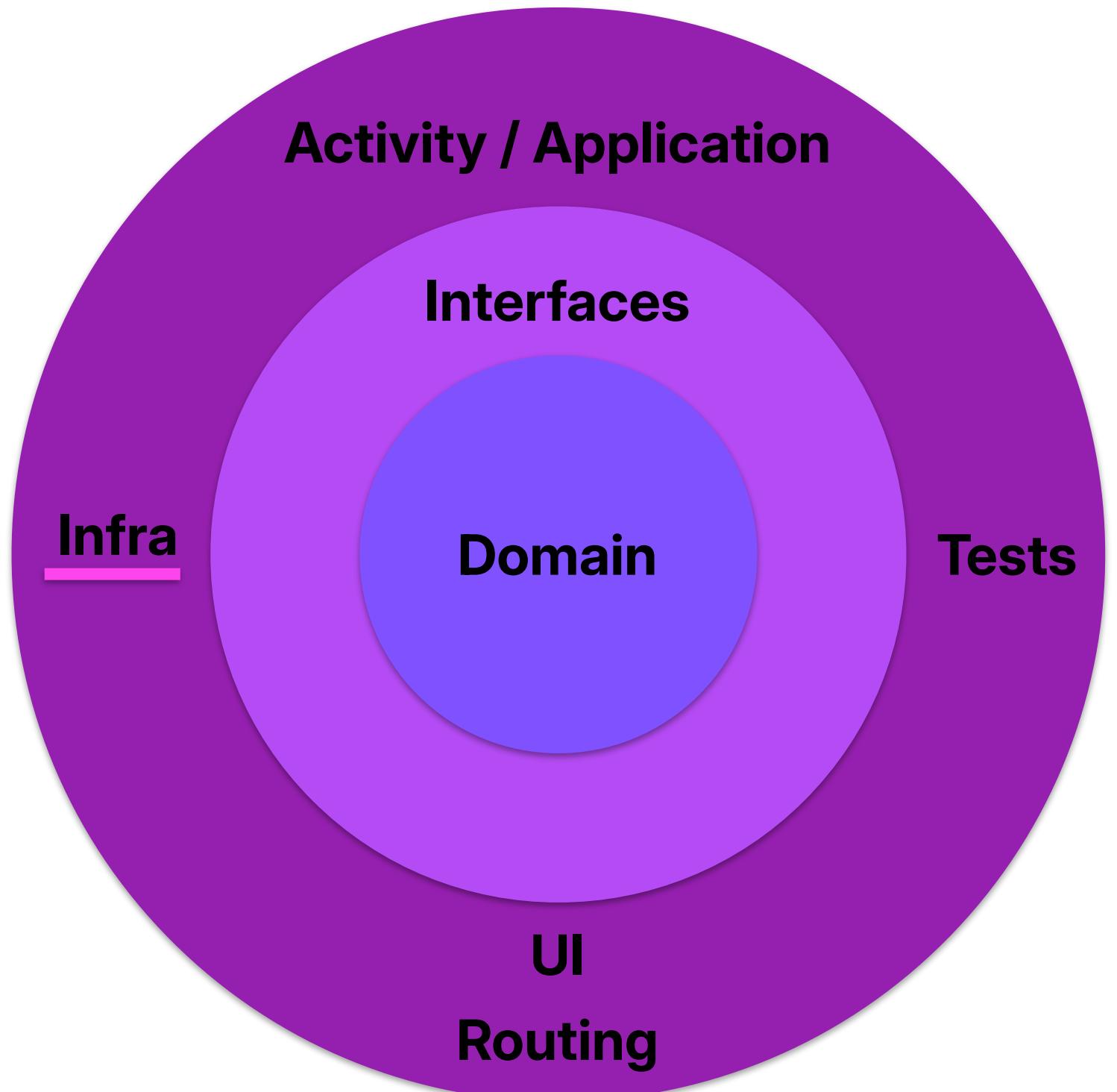


App - plugins

```
suspend fun Application.app(config: AppConfig) {  
    install(ContentNegotiation) {  
        json(Json {  
            encodeDefaults = true  
            isLenient = true  
        })  
    }  
    if (developmentMode) install(CallLogging)  
    install(SSE)  
    install(WebSockets) {  
        pingPeriod = 15.seconds  
        timeout = 15.seconds  
    }  
}
```

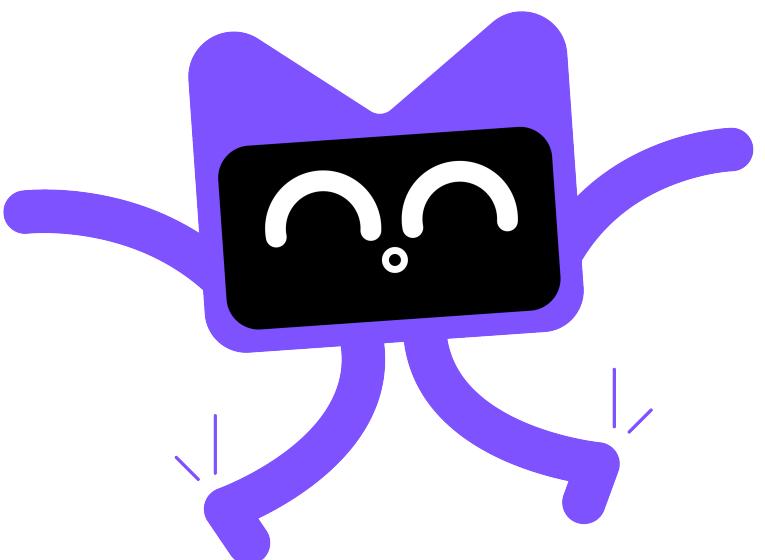


Infrastructure



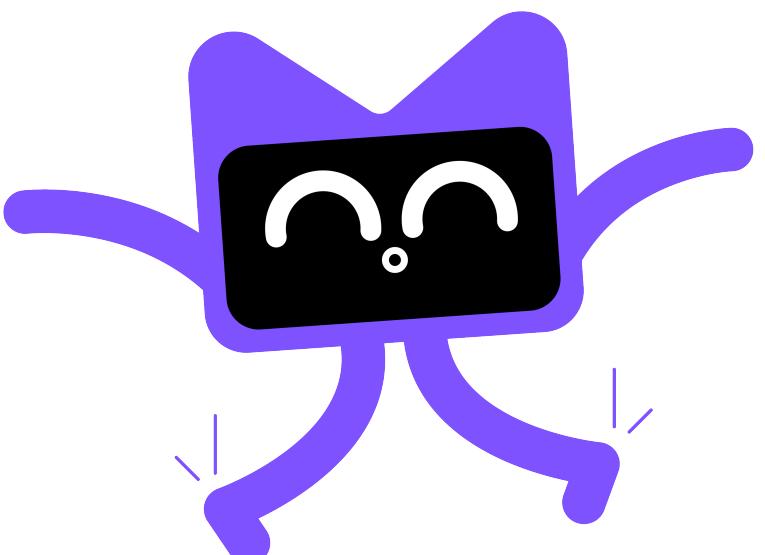
Infrastructure

```
@Serializable  
data class DatabaseConfig(  
    val url: String,  
    val username: String,  
    val password: String,  
)
```



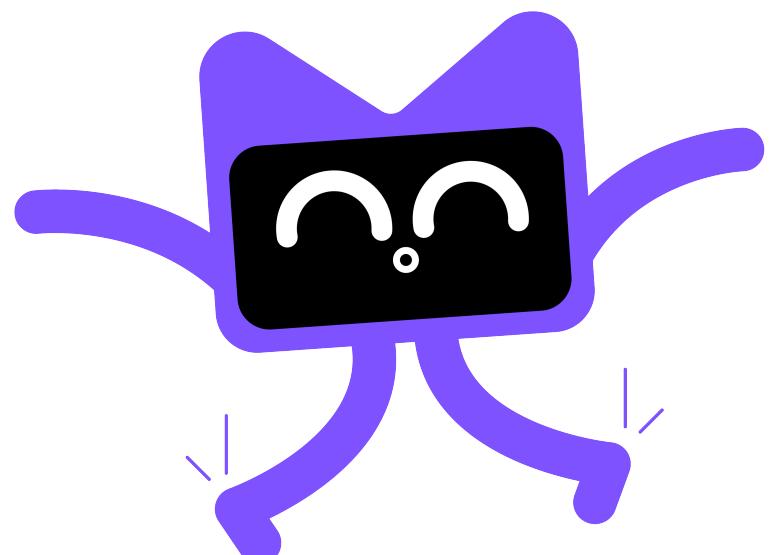
Persistence

```
app:  
    host: "$KTOR_HOST:0.0.0.0"  
    port: "$KTOR_PORT:8080"  
  
database:  
    url: "$DB_URL:jdbc:postgresql://localhost:5432/postgres"  
    username: "$DB_USERNAME:ktor_user"  
    password: "$DB_PASSWORD:ktor_password"
```



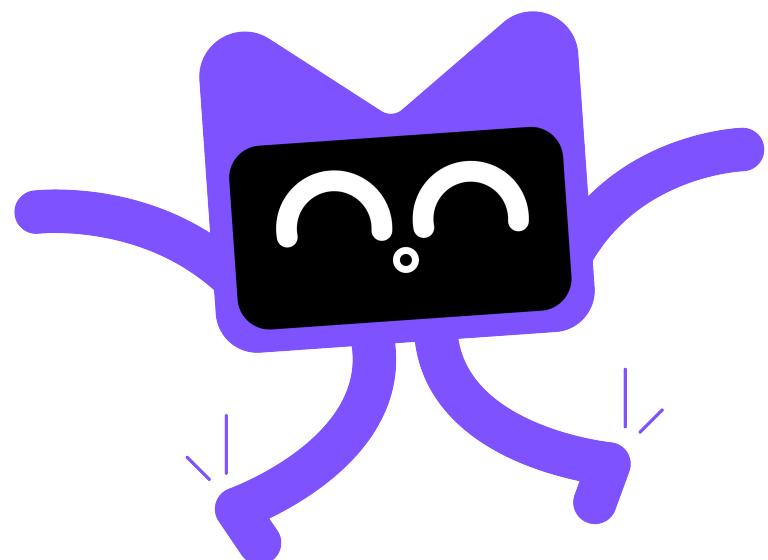
HikariDataSource

```
private fun dataSource(config: DatabaseConfig): HikariDataSource =  
    HikariDataSource(  
        HikariConfig().apply {  
            jdbcUrl = config.url  
            username = config.username  
            password = config.password  
        }  
    )
```



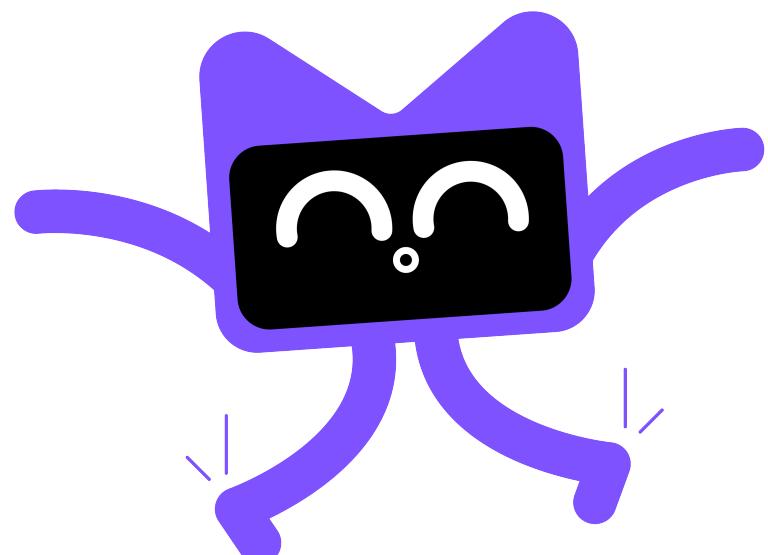
Exposed

```
fun Application.database(config: DatabaseConfig): Database {  
    val dataSource = dataSource(config)  
    val database = Database.connect(dataSource)  
    return database  
}
```



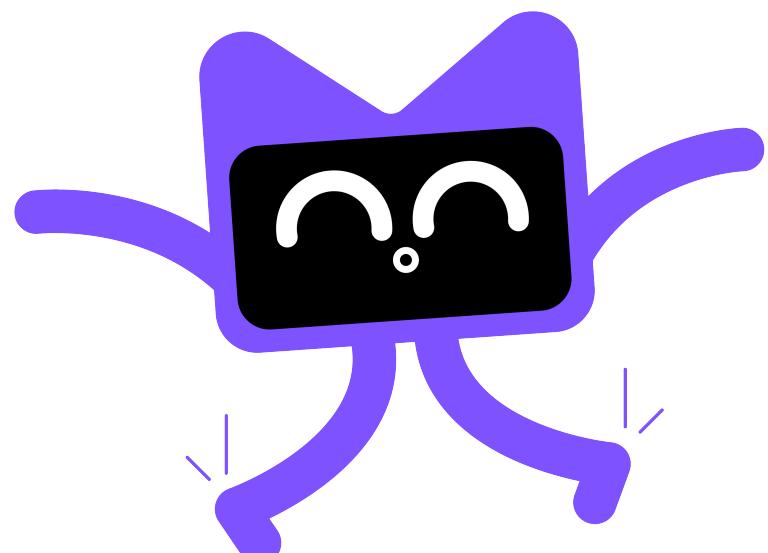
Lifecycle

```
fun Application.database(config: DatabaseConfig): Database {  
    val dataSource = dataSource(config)  
    val database = Database.connect(dataSource)  
    monitor.subscribe(ApplicationStopped) {  
        TransactionManager.closeAndUnregister(database)  
        dataSource.close()  
    }  
    return database  
}
```



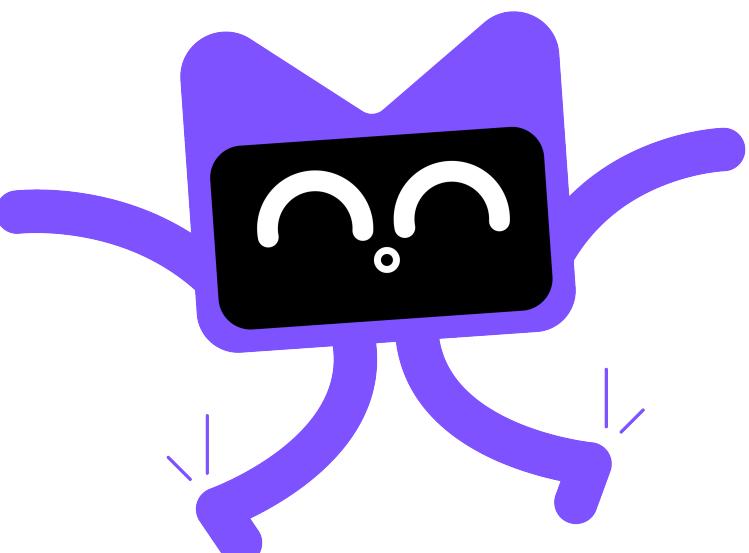
Integration Testing with Testcontainers®

```
fun PostgreSQLContainer<Nothing>.config(): DatabaseConfig =  
    DatabaseConfig(  
        url = jdbcUrl,  
        username = username,  
        password = password,  
    )
```

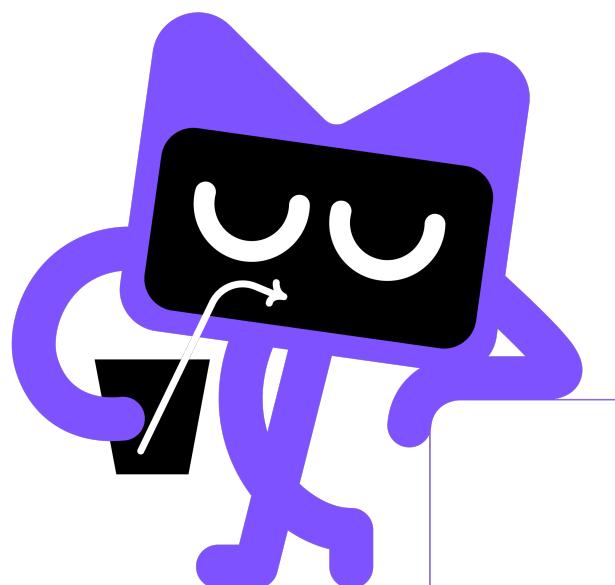
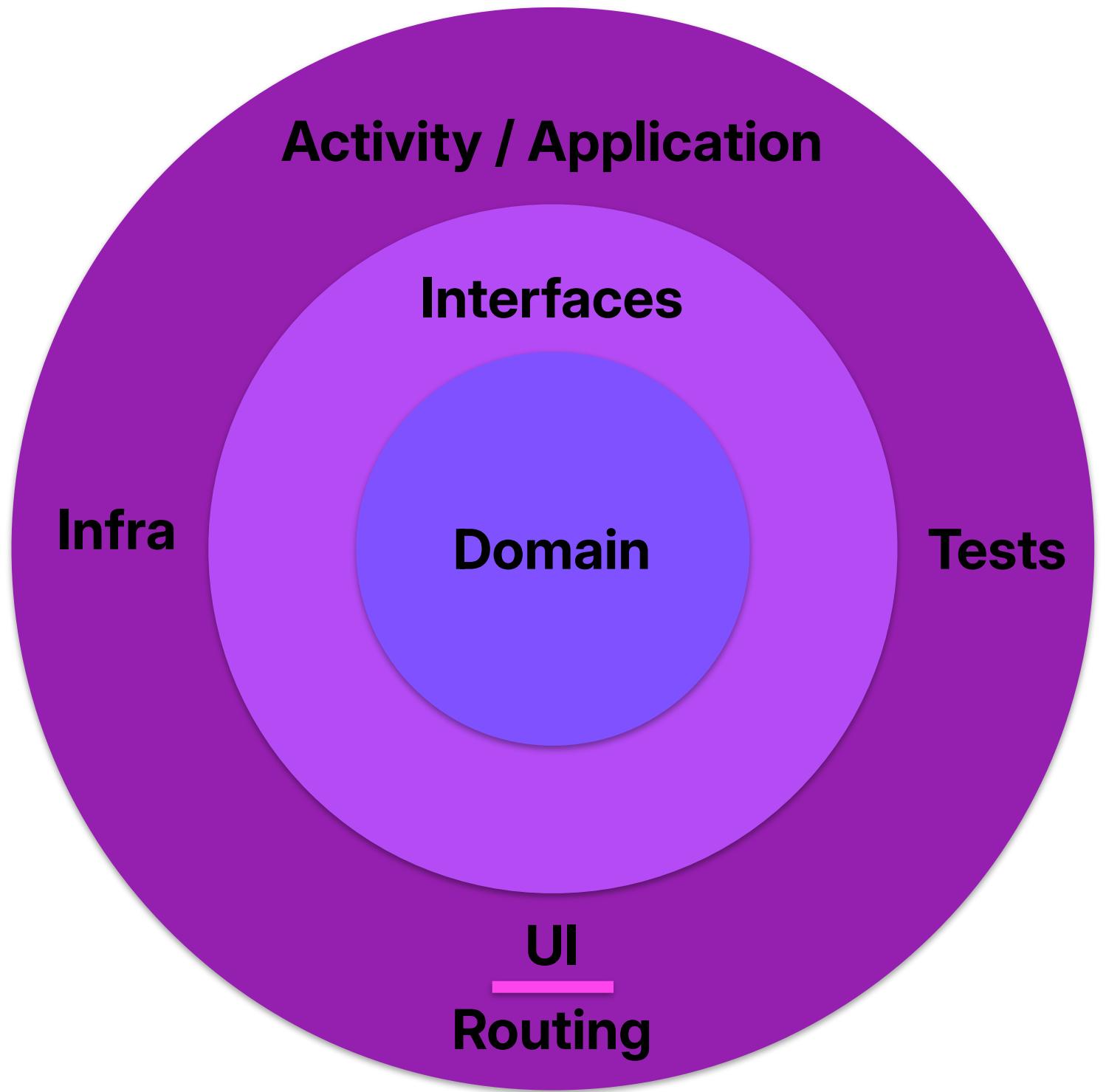


Integration Testing with Testcontainers®

```
val container by lazy {  
    PostgreSQLContainer<Nothing>("postgres:16-alpine")  
        .also { it.start() }  
}
```



Architectural layering



View / UI

14:05 53%
Speaker

Simon Vergauwen
Developer Advocate at JetBrains & OSS (arrow-kt) Maintainer



Software engineer from Antwerp, Belgium. Interested in everything functional, and Kotlin. Loves OSS, cooking, snowboarding and Fallout.

Hands-On Kotlin Web Development With Ktor 

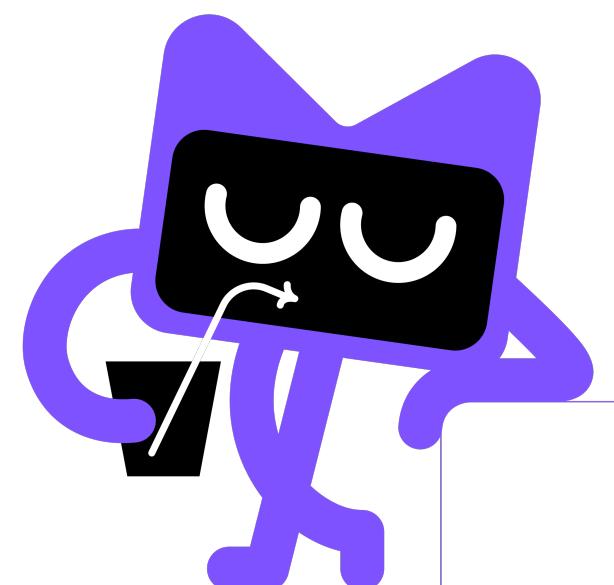
Workshop Intermediate
Anton Arhipov, Leonid Stashevskii, Simon Vergauwen

Room #176 May 21, 09:00 – 17:00

Coroutines and Structured Concurrency in Ktor 

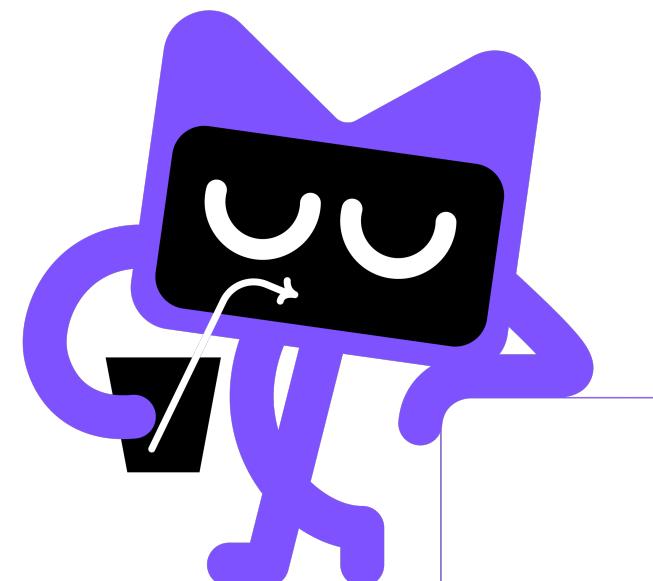
Lightning talk Intermediate Server-side
Language and best practices

Simon Vergauwen



CRUD <~> HttpMethod

- **CREATE** → • **HttpMethod.Post**
- **READ** → • **HttpMethod.Get**
- **UPDATE** → • **HttpMethod.Put**
- **DELETE** → • **HttpMethod.Delete**



HttpStatusCodes

2xx

- 200 OK
- 201 Created
- 204 No Content

4xx

- 400 Bad Request
- 401 Unauthorised
- 404 Not Found

3xx

- 301 Moved Permanently
- 302 Found

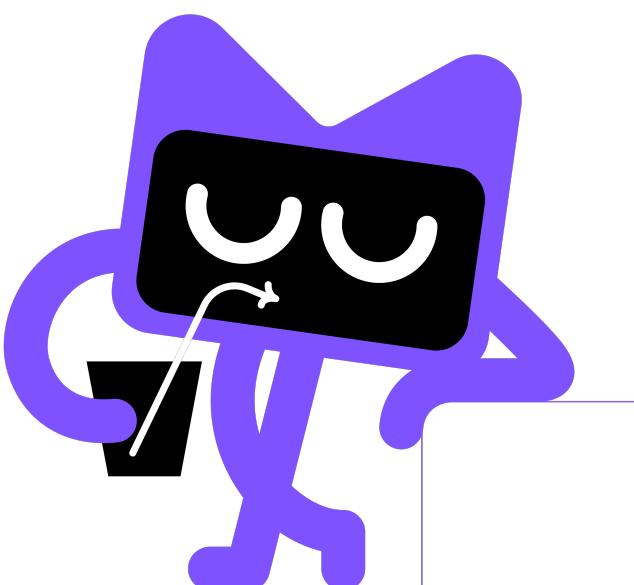
5xx

- 500 Internal Server Error



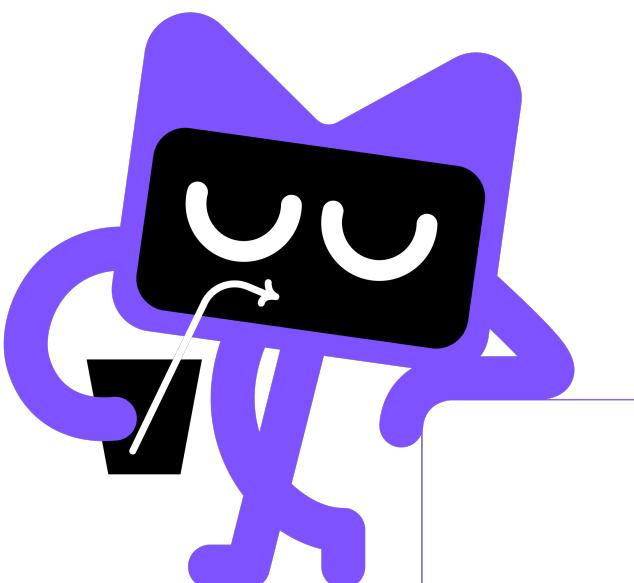
Routing

```
fun Application.userRoutes(users: UserService) =  
    routing {  
        post("/user/register") { }  
        get("/user/{userId}") { }  
        put("/user/update") { }  
        delete("/user/delete") { }  
    }
```



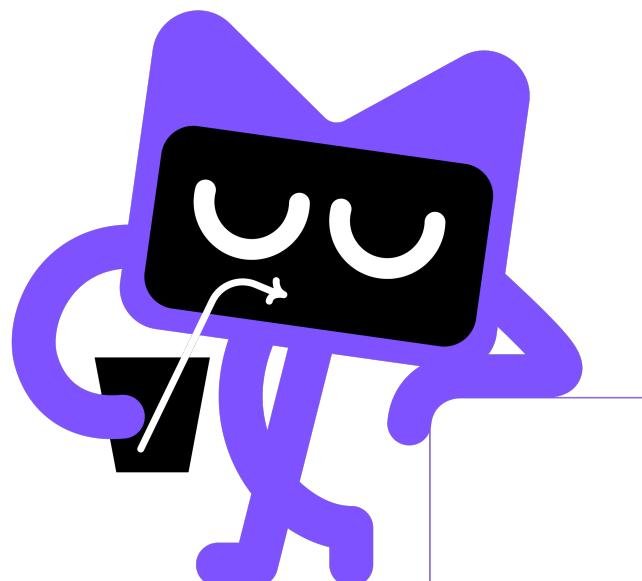
Grouping routes

```
fun Application.userRoutes(users: UserService) =  
    routing {  
        route("/user") {  
            post("/register") { }  
            get("/{userId}") { }  
            put("/update") { }  
            delete("/delete") { }  
        }  
    }
```



RouteHandler

```
get("/{userId}") {  
    val userId = call.pathParameters.getOrFail("userId")  
}
```

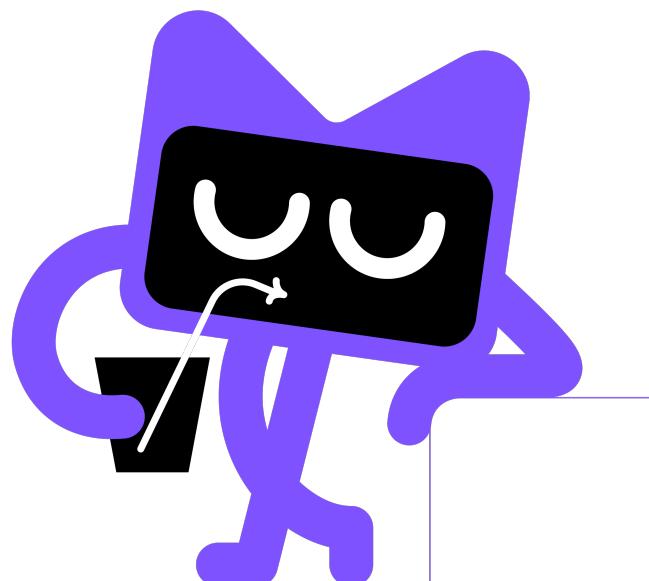


RouteHandler

```
enum class Extras { Profile, Followers; }

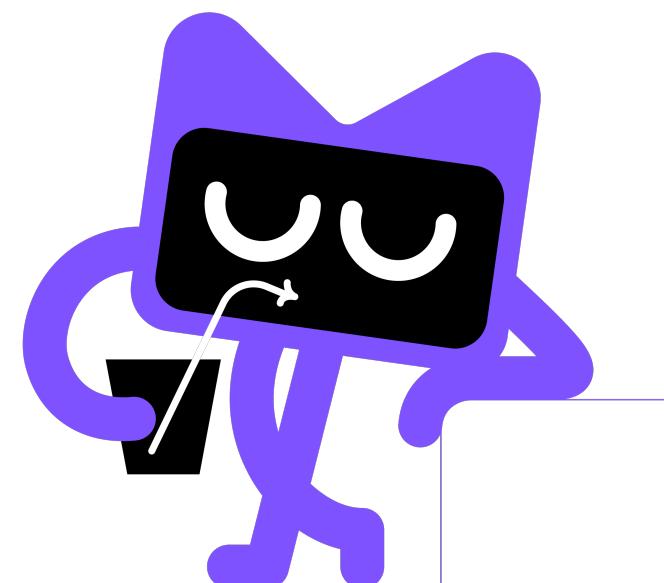
get("/{userId}") {
    val userId = call.pathParameters.getOrFail("userId")
    val extras = call.queryParameters.getOrFail<List<Extras>>("extras")
}
```

All primitives and enums



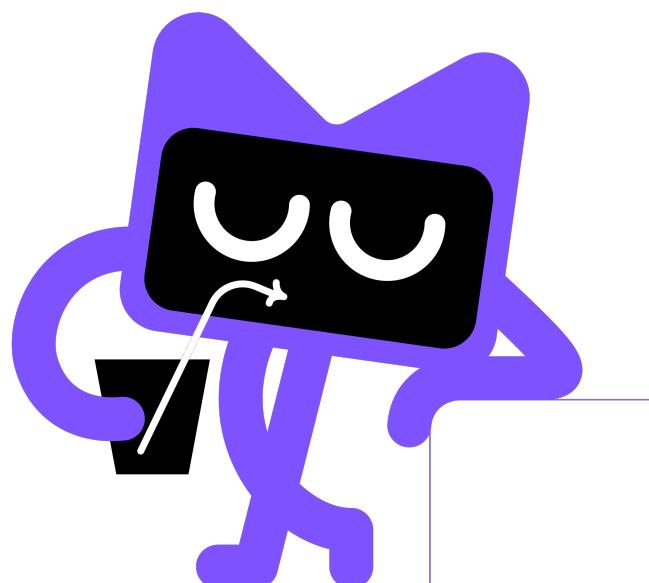
RouteHandler

```
get("/{userId}") {  
    val userId = call.pathParameters.getOrFail("userId")  
    val extras = call.queryParameters.getOrFail<List<Extras>>("extras")  
    val userAgent = call.request.headers["User-Agent"]  
}
```



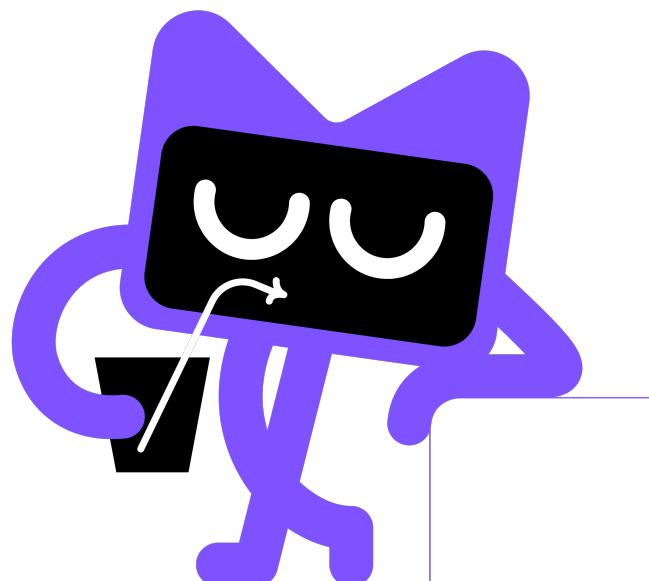
RouteHandler

```
get("/{userId}") {  
    val userId = call.pathParameters.getOrFail("userId")  
    val extras = call.queryParameters.getOrFail<List<Extras>>("extras")  
    val userAgent = call.request.headers["User-Agent"]  
    val cookie = call.request.cookies["my-cookie"]  
  
}
```



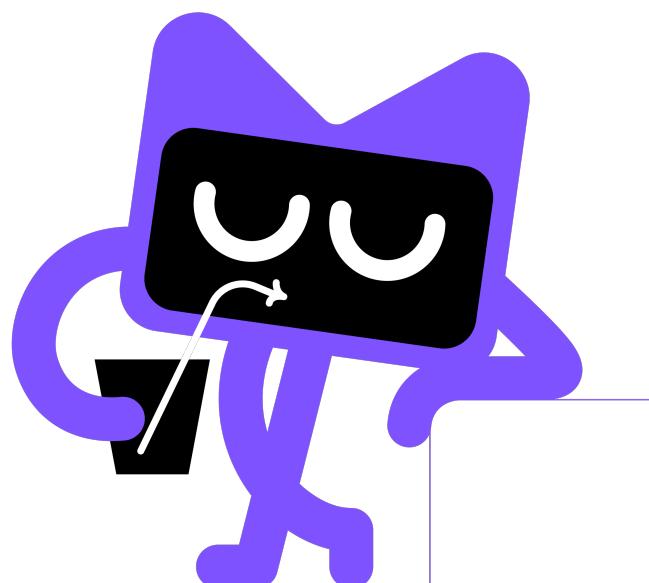
RouteHandler

```
get("/{userId}") {  
    val userId = call.pathParameters.getOrFail("userId")  
    val extras = call.queryParameters.getOrFail<List<Extras>>("extras")  
    val userAgent = call.request.headers["User-Agent"]  
    val cookie = call.request.cookies["my-cookie"]  
  
    val user = users.getUserOrNull(userId)  
}
```

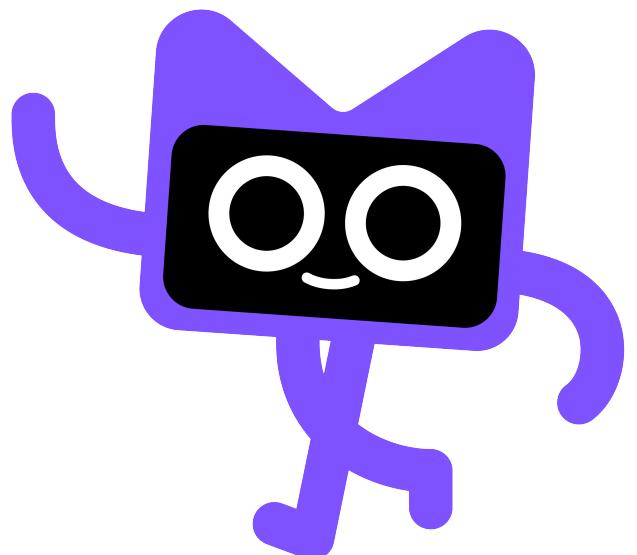
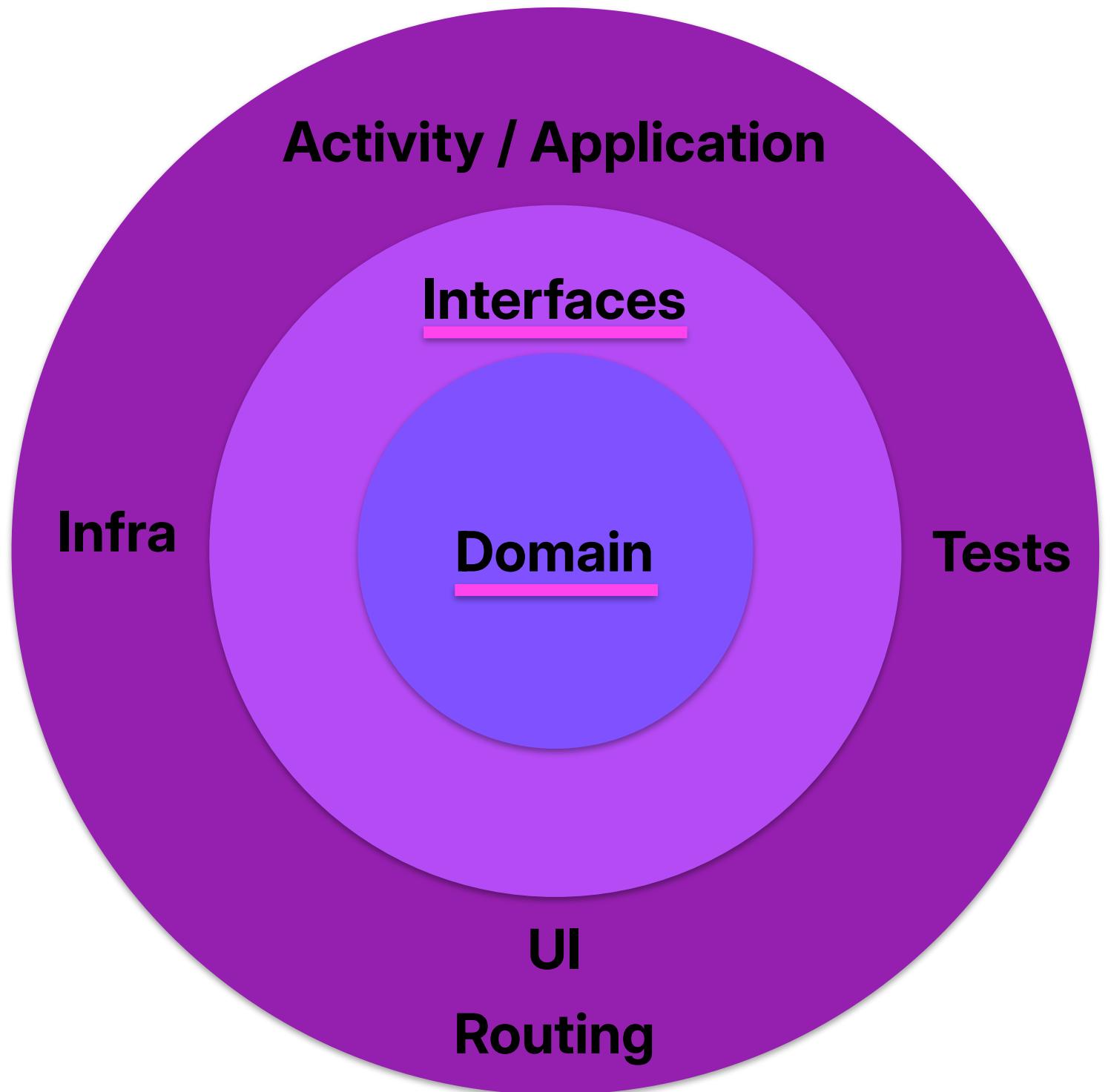


RouteHandler

```
get("/{userId}") {  
    val userId = call.pathParameters.getOrFail("userId")  
    val extras = call.queryParameters.getOrFail<List<Extras>>("extras")  
    val userAgent = call.request.headers["User-Agent"]  
    val cookie = call.request.cookies["my-cookie"]  
  
    val user = users.getUserOrNull(userId)  
  
    if (user != null) call.respond(HttpStatusCode.OK, user)  
    else call.respond(HttpStatusCode.NotFound)  
}
```



Architectural layering

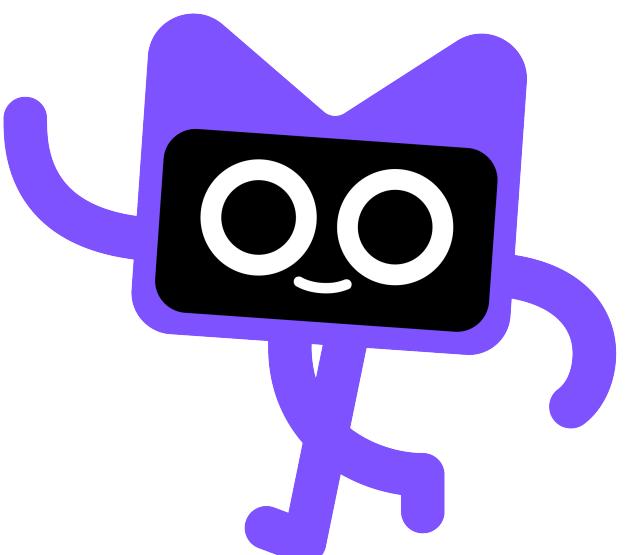


Domain / Model

Shared between client and server

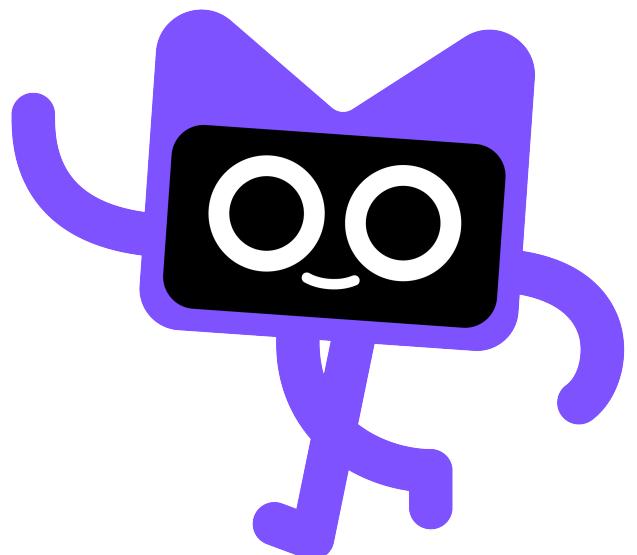
```
@Serializable  
data class UpsertUser(  
    val email: String,  
    val displayName: String?,  
    @SerializedName("about_me")  
    val aboutMe: String?,  
)
```

```
@Serializable  
data class User(  
    val userId: Long,  
    val email: String,  
    val displayName: String,  
    val aboutMe: String,  
)
```



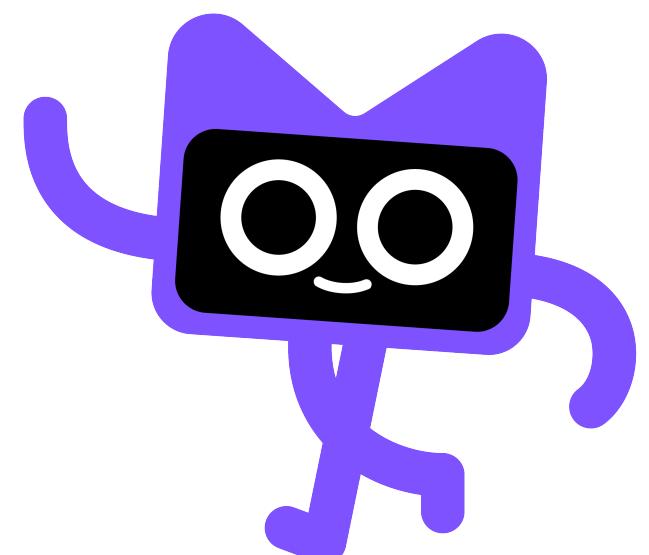
CRUD Repository

```
interface UserRepository {  
    suspend fun create(subject: String): User  
    suspend fun findOrNull(subject: String): User?  
    suspend fun findAll(): List<User>  
    suspend fun update(subject: String, updateUser: UpdateUser): User  
    suspend fun delete(subject: String): Boolean  
}
```



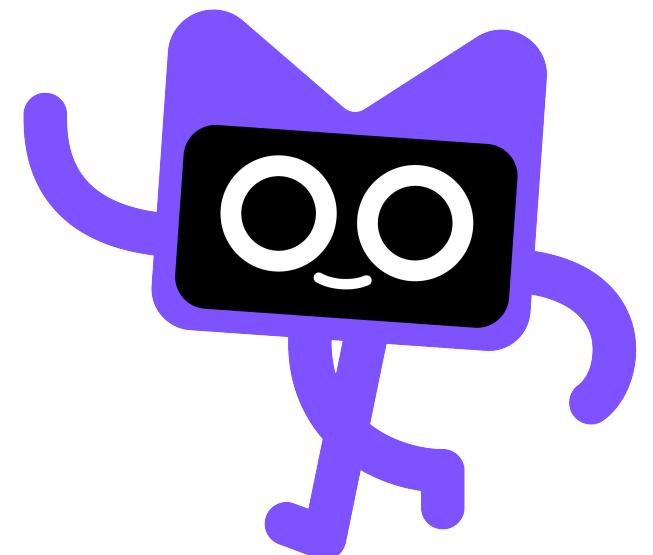
Exposed Table

```
object UserTable : LongIdTable("users", "user_id") {  
    val subject = varchar("subject", 256).uniqueIndex()  
    val email = varchar("email", 256).nullable()  
    val displayName = varchar("display_name", 256).nullable()  
    val aboutMe = varchar("about_me", 256).nullable()  
}
```



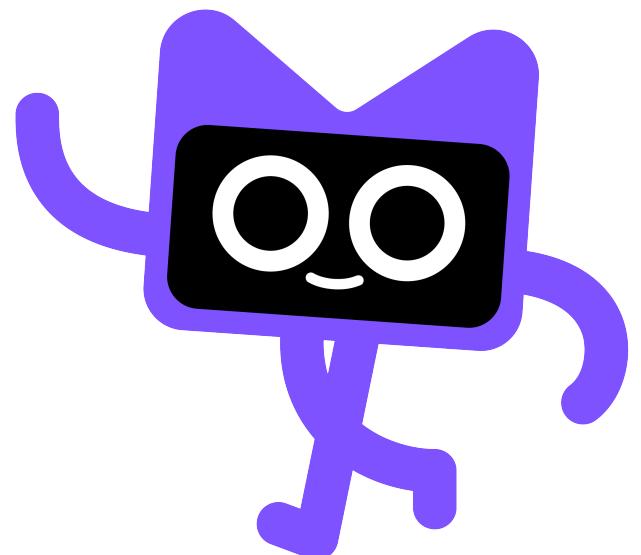
Exposed Repository

```
class ExposedUserRepository(private val db: Database) : UserRepository {
```



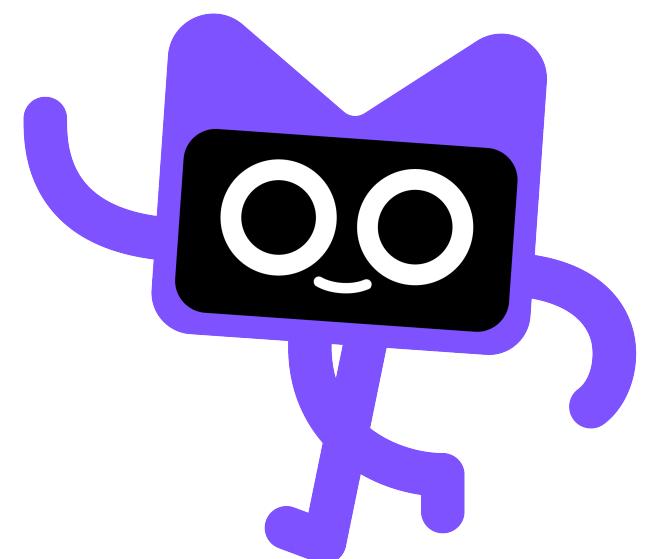
Exposed Repository

```
class ExposedUserRepository(private val db: Database) : UserRepository {  
    override suspend fun create(subj: String): User  
}
```



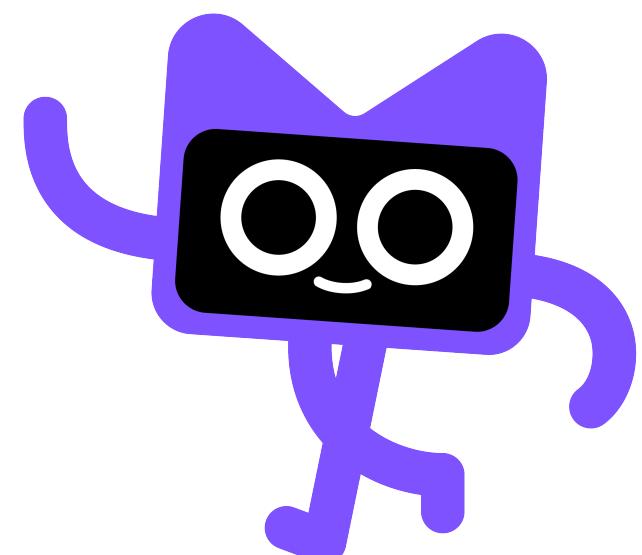
Exposed Repository

```
class ExposedUserRepository(private val db: Database) : UserRepository {  
    override suspend fun create(subj: String): User =  
        withContext(Dispatchers.IO) {  
  
    }  
}
```



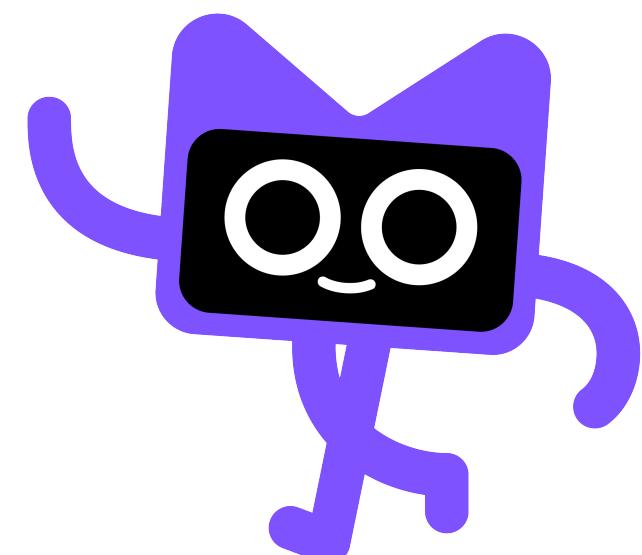
Exposed Repository

```
class ExposedUserRepository(private val db: Database) : UserRepository {  
    override suspend fun create(subj: String): User =  
        withContext(Dispatchers.IO) {  
            transaction(db) {  
                ...  
            }  
        }  
}
```



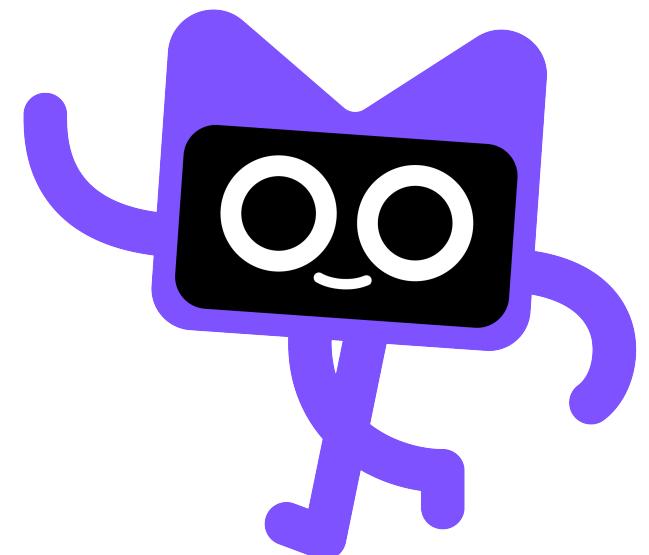
Exposed Repository

```
class ExposedUserRepository(private val db: Database) : UserRepository {  
    override suspend fun create(subj: String): User =  
        withContext(Dispatchers.IO) {  
            transaction(db) {  
                UserTable.upsertReturning(  
                    keys = arrayOf(UserTable.subject)  
                )  
            }  
        }  
}
```



Exposed Repository

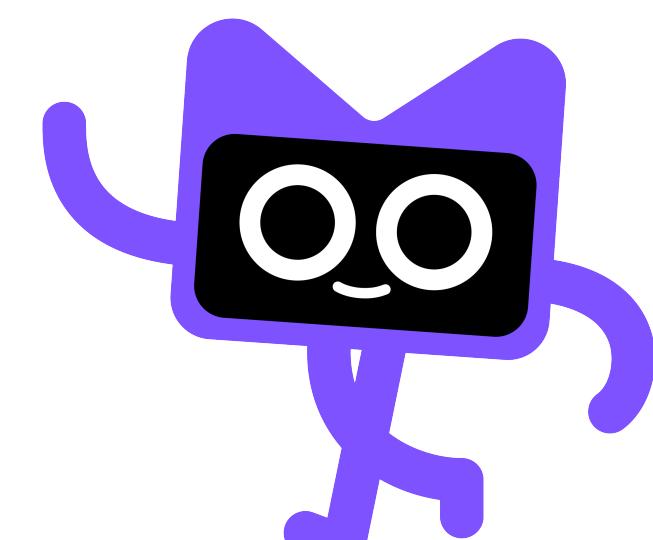
```
class ExposedUserRepository(private val db: Database) : UserRepository {  
    override suspend fun create(subj: String): User =  
        withContext(Dispatchers.IO) {  
            transaction(db) {  
                UserTable.upsertReturning(  
                    keys = arrayOf(UserTable.subject)  
                ) { upsert →  
                    upsert[subject] = subj  
                }  
            }  
        }  
}
```



Exposed Repository

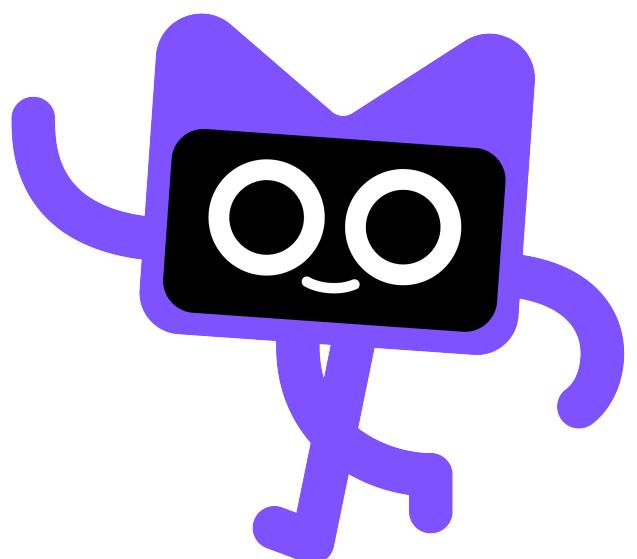
```
class ExposedUserRepository(private val db: Database) : UserRepository {  
    override suspend fun create(subj: String): User =  
        withContext(Dispatchers.IO) {  
            transaction(db) {  
                UserTable.upsertReturning(  
                    keys = arrayOf(UserTable.subject)  
                ) { upsert →  
                    upsert[subject] = subj  
                }.single().toUser()  
            }  
        }  
}
```

Iterable<ResultRow>.single(): ResultRow



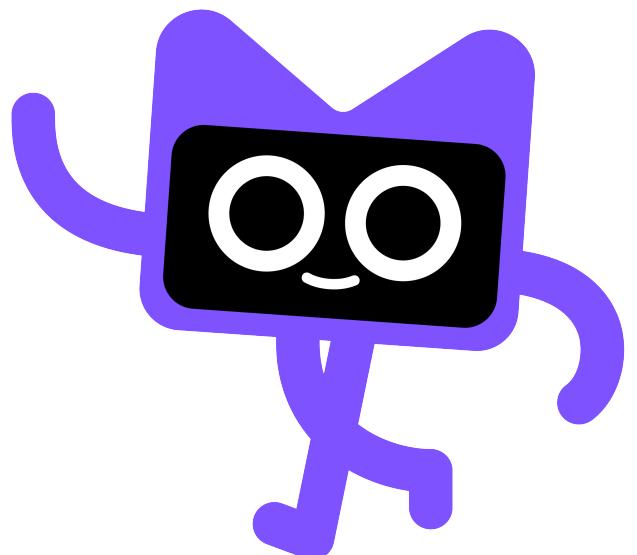
Exposed Repository

```
private fun ResultRow.toUser(): User = User(  
    this[UserTable.id].value,  
    this[UserTable.subject],  
    this[UserTable.email],  
    this[UserTable.displayName],  
    this[UserTable.aboutMe],  
)
```



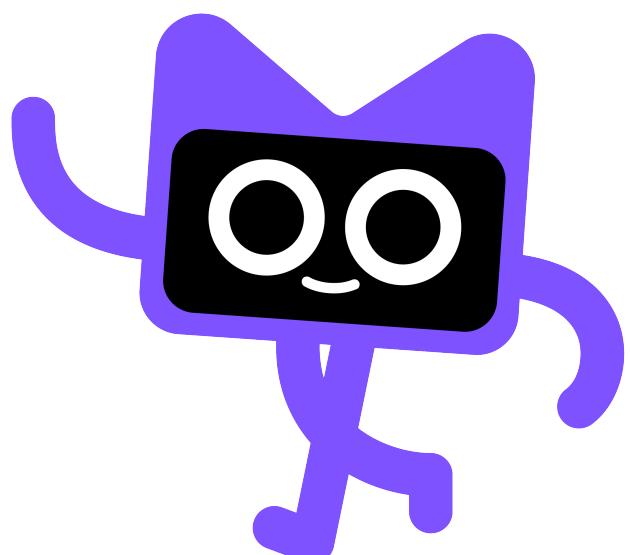
UserService = UserRepository + ...

```
interface UserService {  
    suspend fun userSessions(  
        userId: String,  
        userAgent: String?  
    ): List<String>  
  
    suspend fun deleteSessions(  
        userId: String,  
        userAgent: String?  
    ): Boolean  
}
```

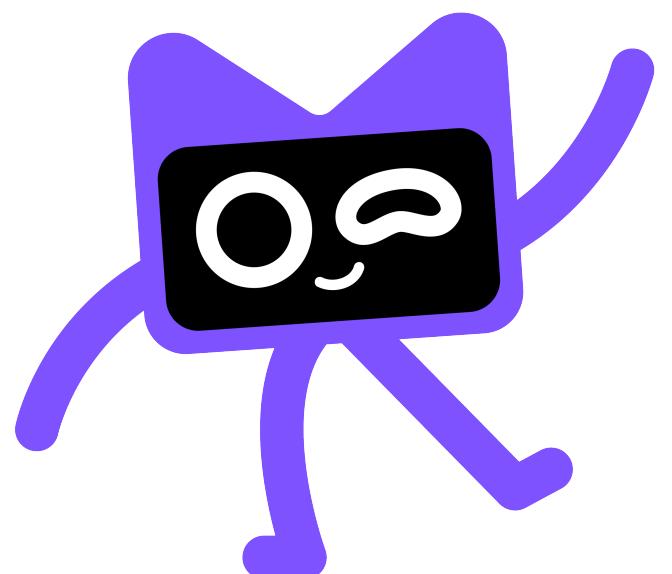
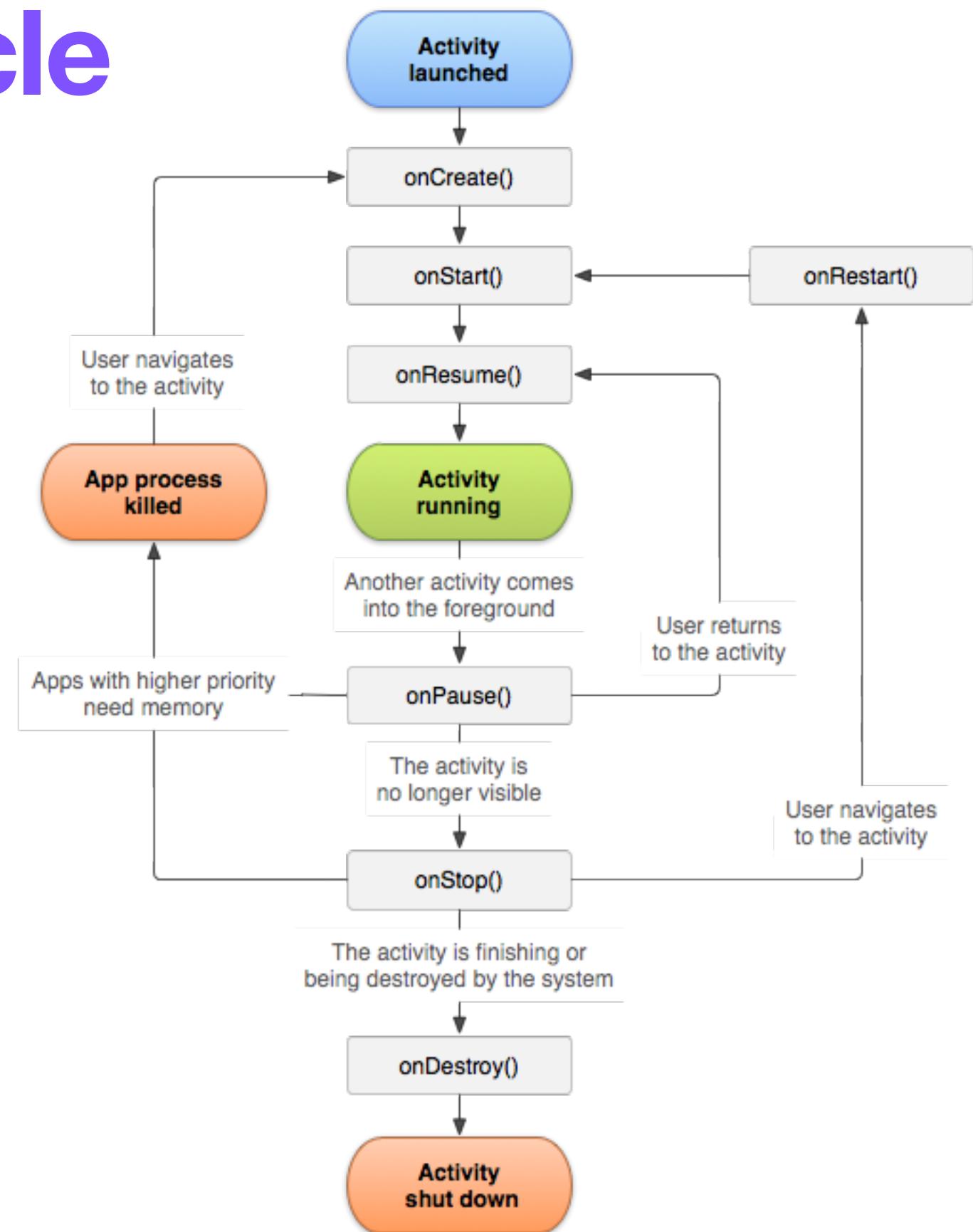


Application.kt

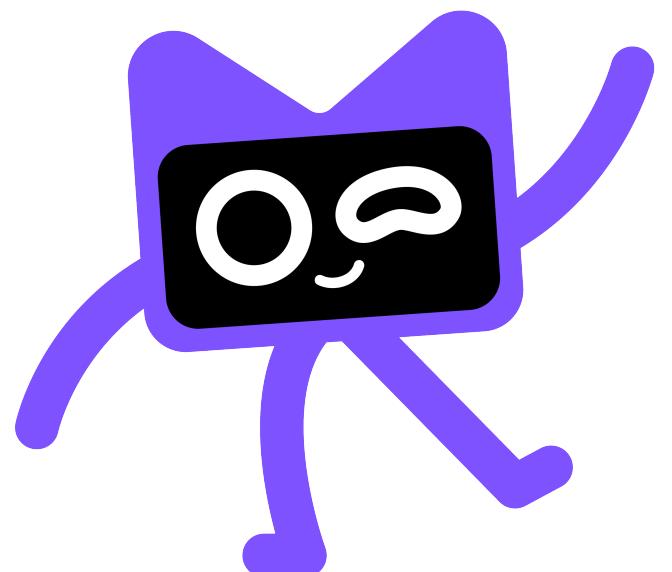
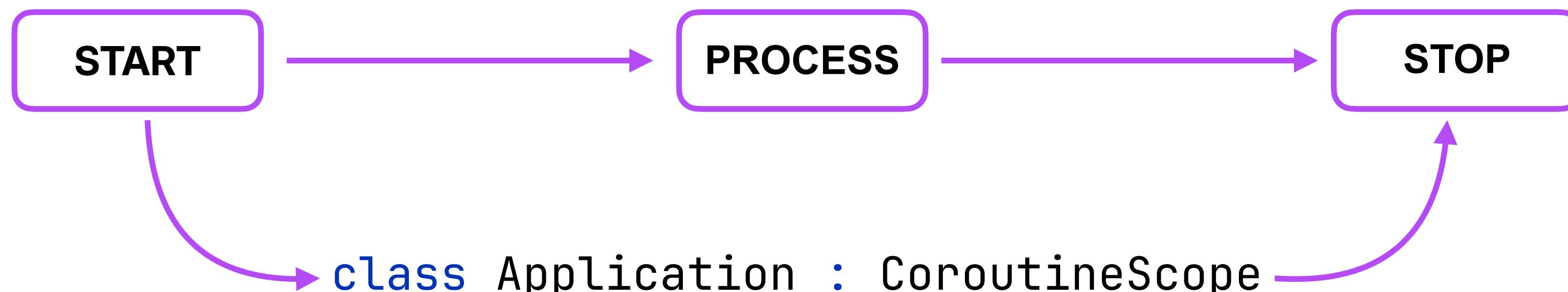
```
suspend fun Application.app(config: AppConfig) {  
    val database = database(config.database)  
    val userRepository: UserRepository = ExposedUserRepository(database)  
  
    install(ContentNegotiation) { json() }  
    if (developmentMode) install(CallLogging)  
  
    routing {  
        userRoutes(userRepository)  
    }  
}
```



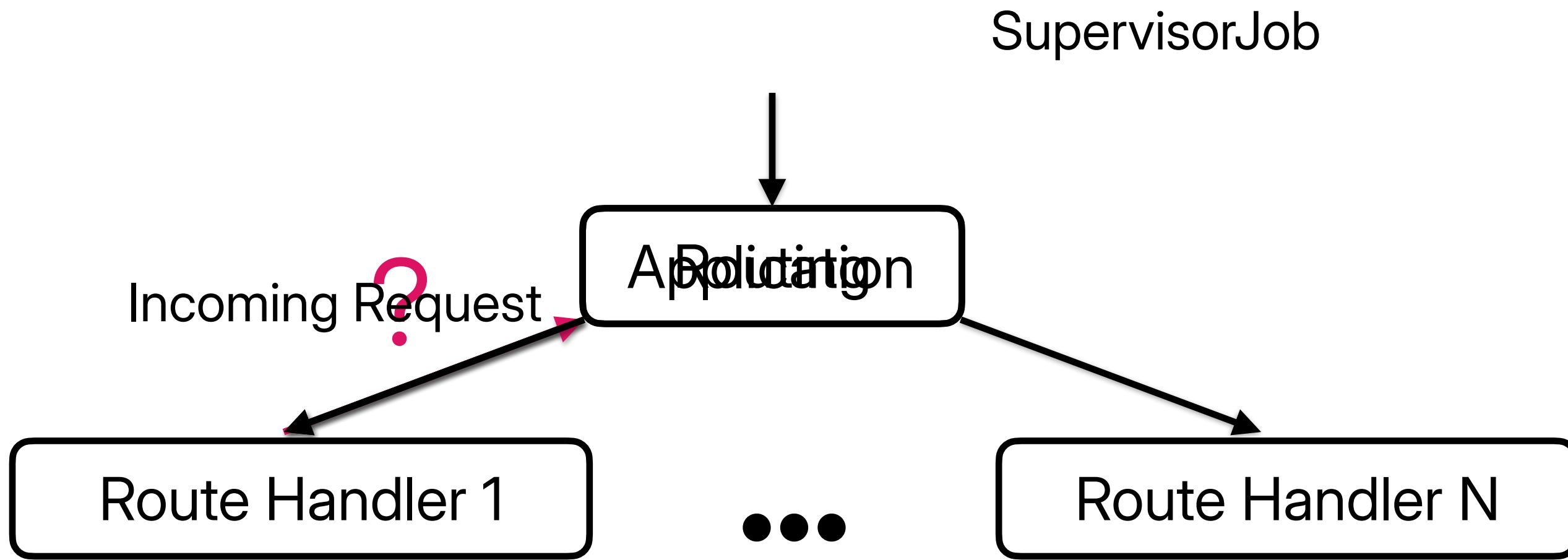
Activity Lifecycle



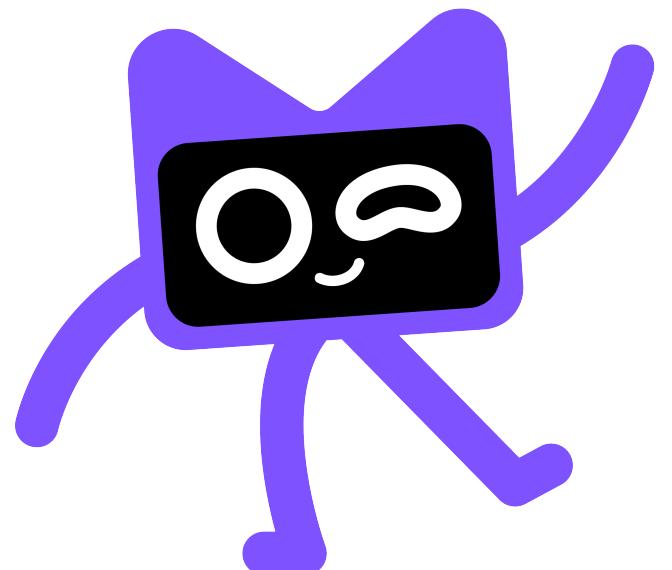
Application Lifecycle



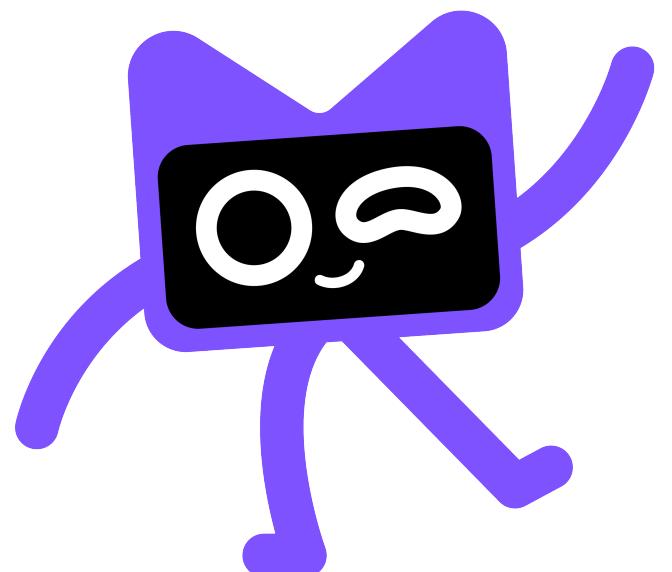
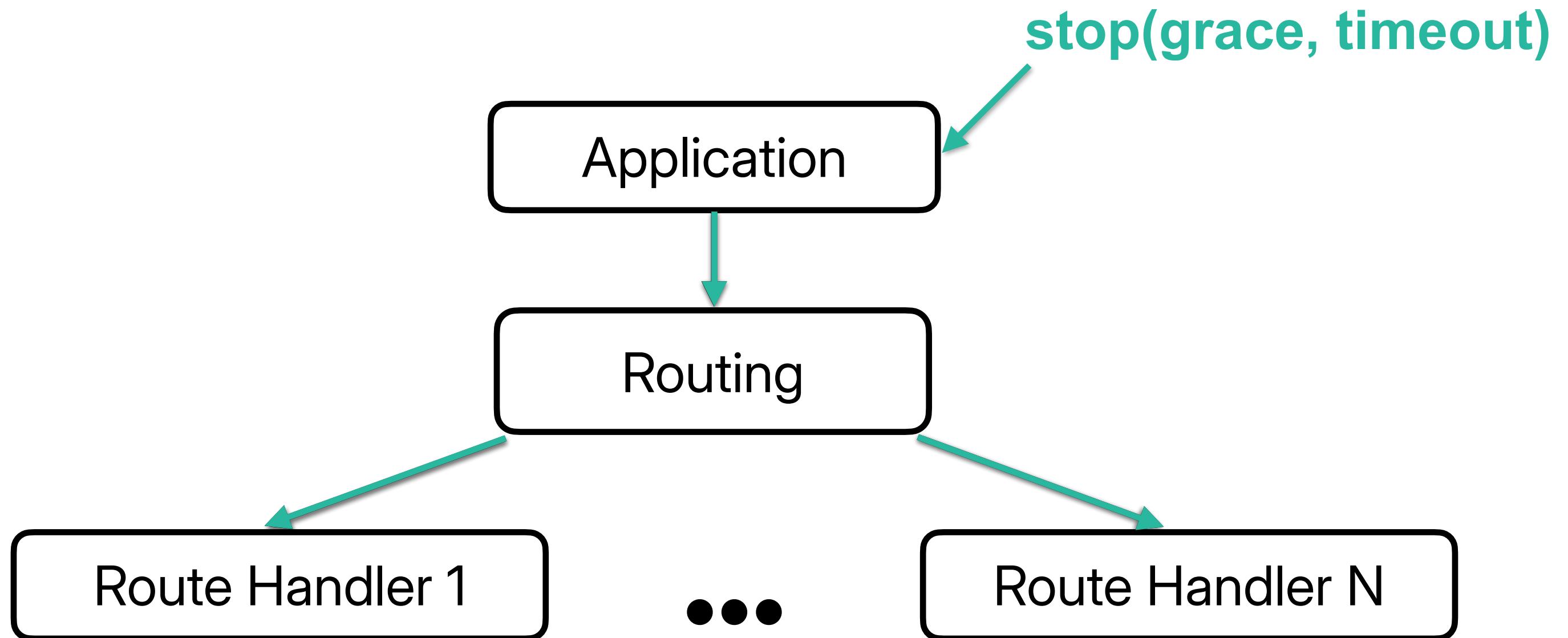
Application Hierarchy



Unhandled exception caught...
500 (Internal Server Error);

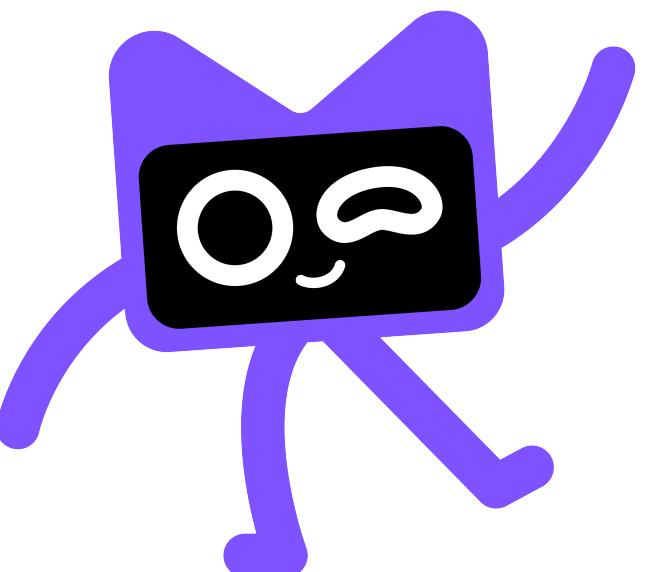


Application



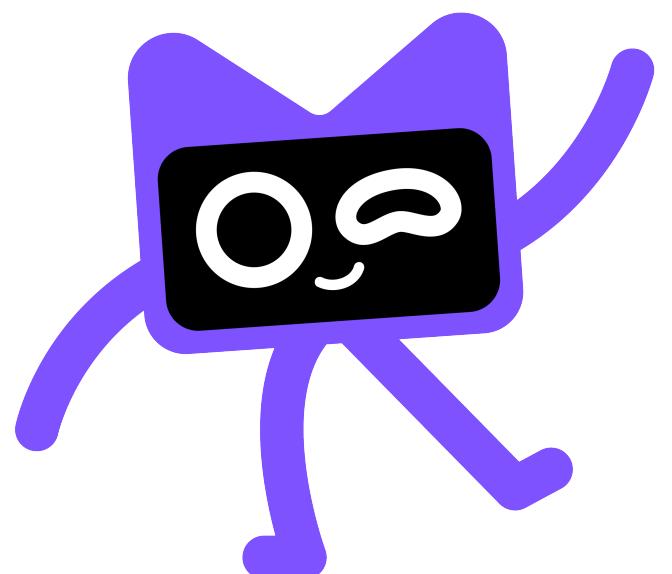
Application Lifecycle

```
fun Application.scoped() = launch {  
    try {  
        awaitCancellation()  
    } catch (e: CancellationException) {  
        println("Server closed")  
    }  
}
```



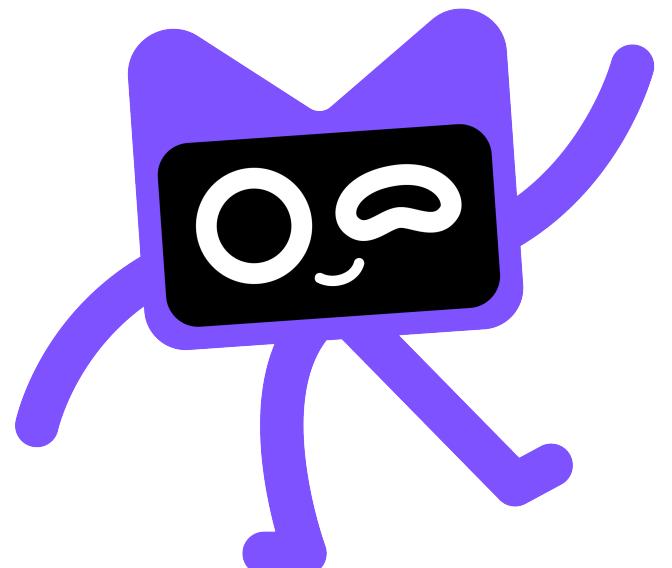
Application Lifecycle

```
monitor.subscribe(ApplicationStopped) {  
    TransactionManager.closeAndUnregister(database)  
    dataSource.close()  
}
```



Application Lifecycle

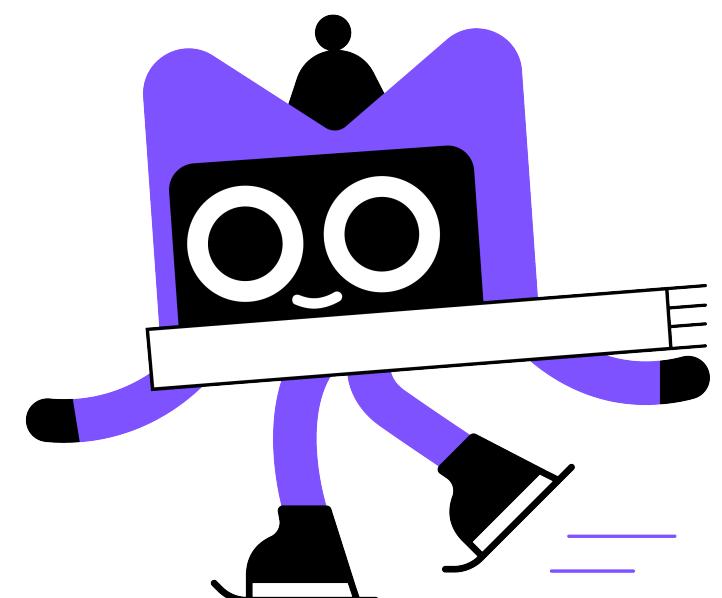
```
class MyService(private val scope: CoroutineScope) {  
    fun trackMetrics() = scope.launch {  
        /* Some code that runs in parallel */  
    }  
}  
  
fun Application.module() {  
    val service = MyService(this)  
}
```



Authorization Bearer

```
fun HttpClient(tokenProvider: TokenProvider): HttpClient =  
    HttpClient {  
        install(Auth) {  
            bearer {  
                loadTokens { tokenProvider.loadTokens() }  
                refreshTokens { tokenProvider.refreshToken() }  
            }  
        }  
    }
```

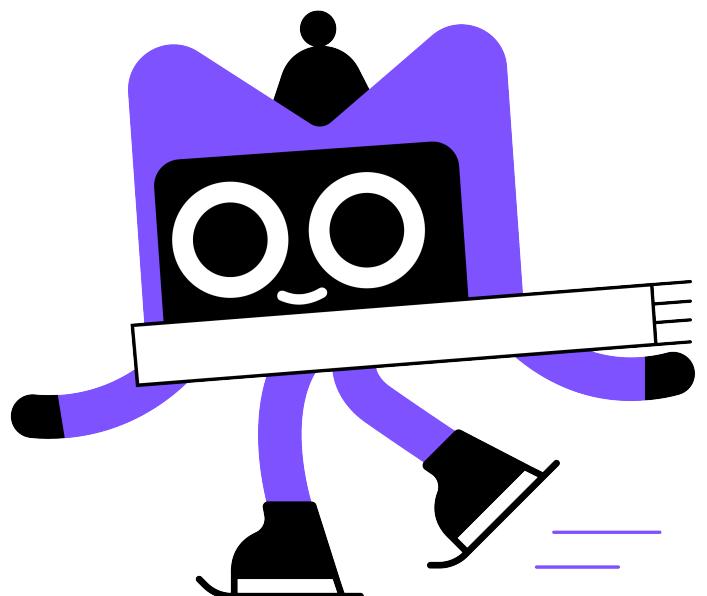
okhttp3.Authenticator



Auth

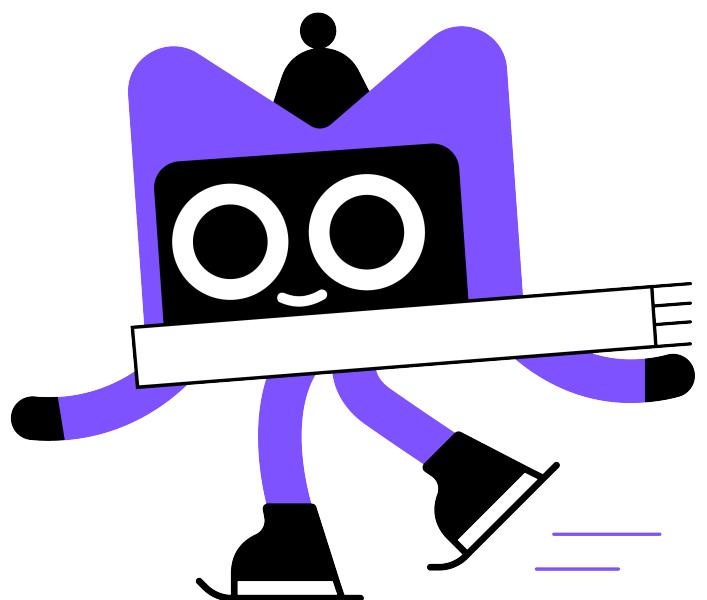
```
@Serializable  
data class JwkConfig(val issuer: String, val jwkUrl: String)
```

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



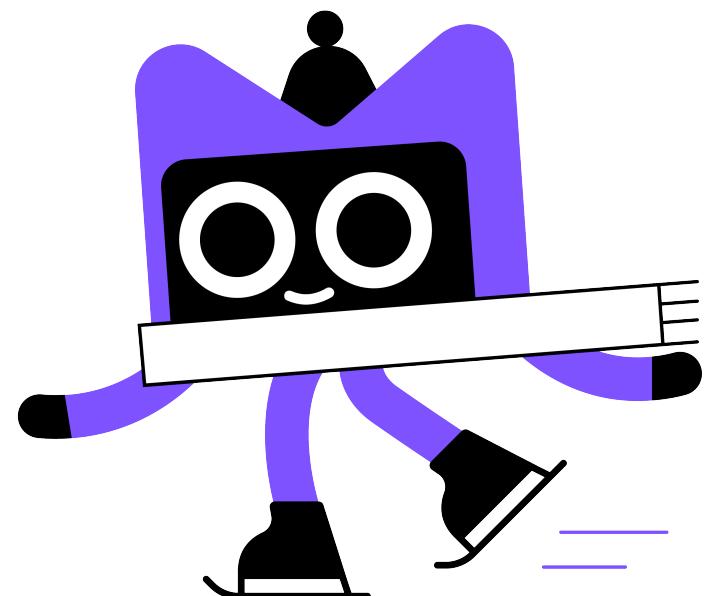
Auth

```
fun Application.configureJwtAuth(config: JwkConfig) {  
    authentication {  
    }  
}
```



Auth

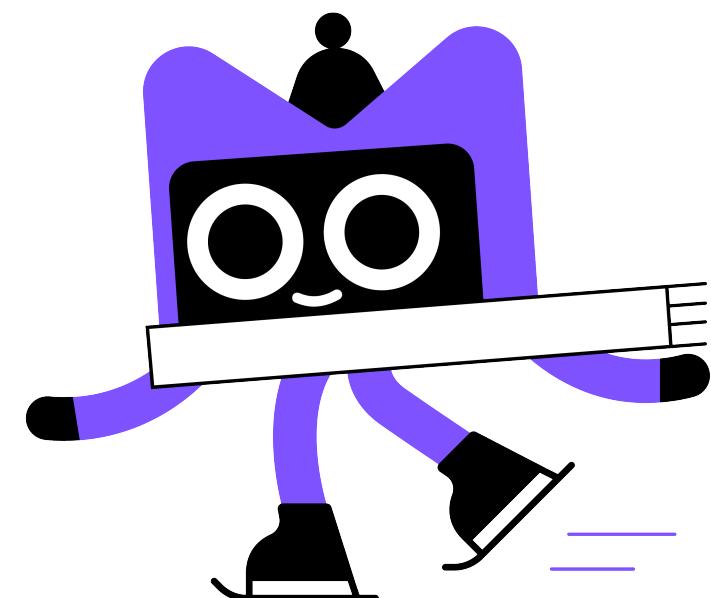
```
fun Application.configureJwtAuth(config: JwkConfig) {  
    authentication {  
        val jwk = JwkProviderBuilder(URL(config.jwkUrl))  
            .cached(10, 24, TimeUnit.HOURS)  
            .rateLimited(10, 1, TimeUnit.MINUTES)  
            .build()  
    }  
}
```



Auth

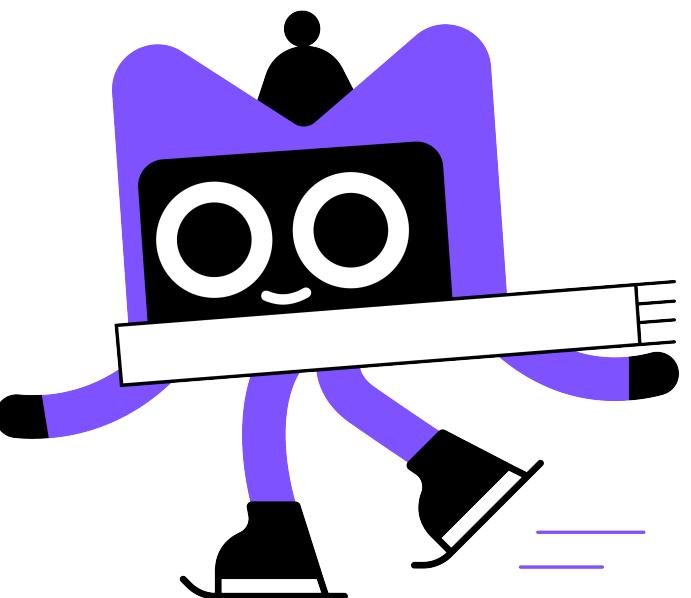
```
fun Application.configureJwtAuth(config: JwkConfig) {  
    authentication {  
        val jwk = JwkProviderBuilder(URL(config.jwkUrl))  
            .cached(10, 24, TimeUnit.HOURS)  
            .rateLimited(10, 1, TimeUnit.MINUTES)  
            .build()  
  
        jwt("google-jwt") {  
            verifier(jwk, config.issuer)  
        }  
    }  
}
```

Verifies signature, issuer, expiration



Auth

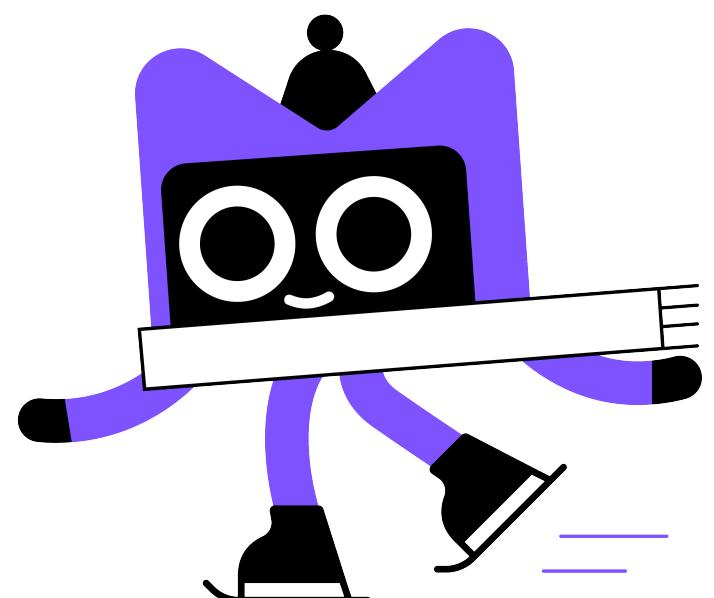
```
jwt("google-jwt") {  
    verifier(jwk, config.issuer)  
    validate { cred: JWTCredential → cred }  
}
```



Auth

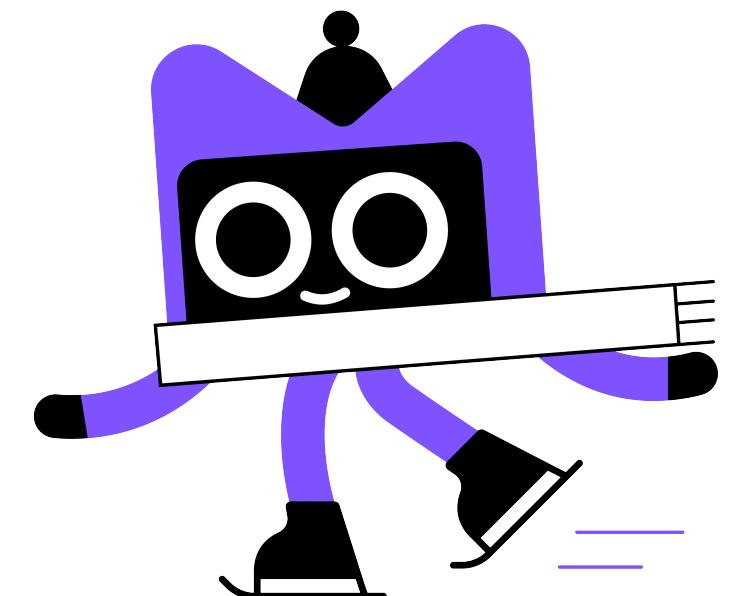
```
jwt("google-jwt") {  
    verifier(jwk, config.issuer)  
    validate { cred: JCredential →  
        val subject: String? = cred.payload.subject  
        if (subject ≠ null) GoogleIdToken(subject, cred.payload)  
        else null  
    }  
}
```

```
class GoogleIdToken(  
    val subject: String,  
    val payload: Payload  
)
```

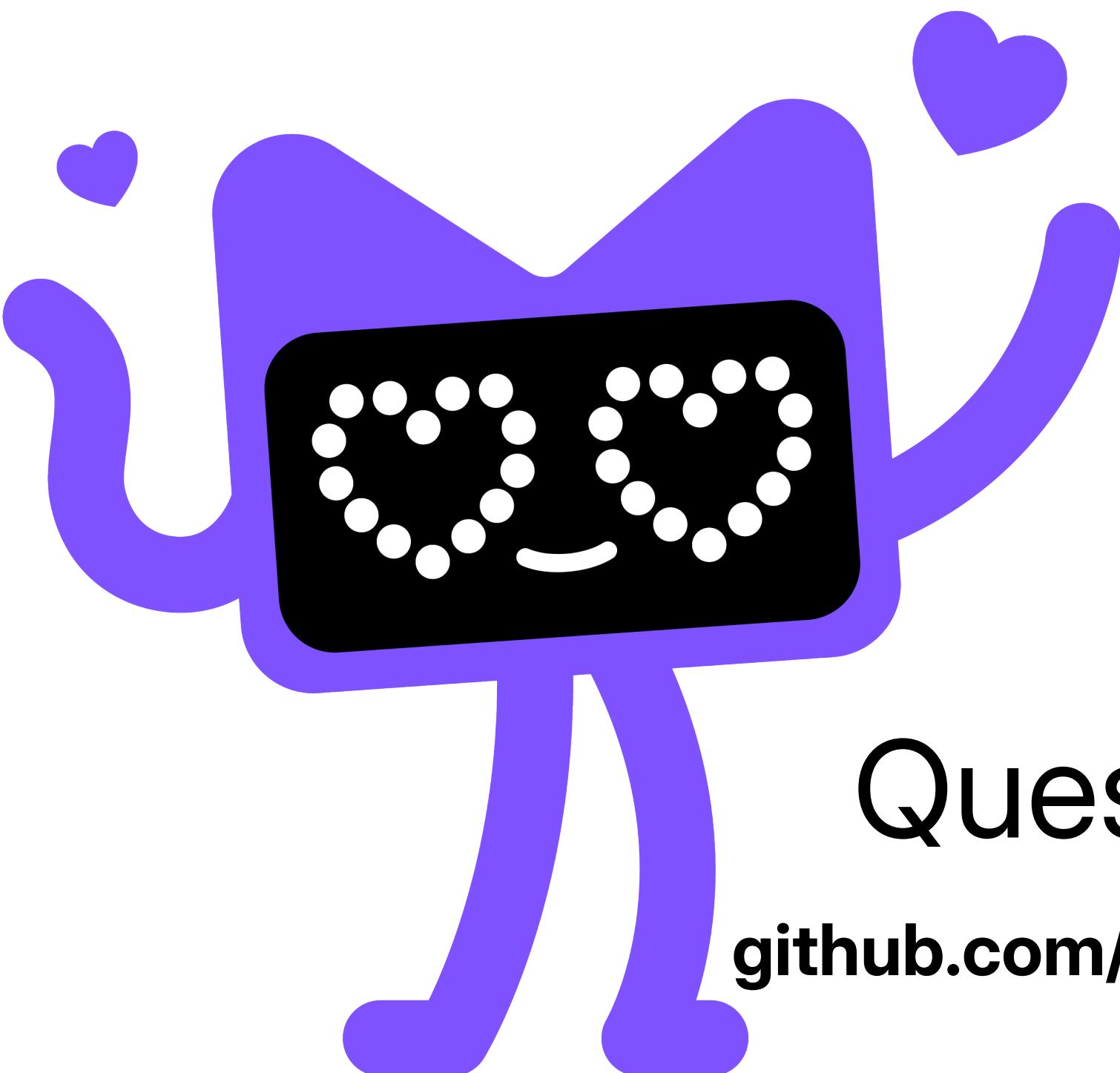


Auth

```
fun Application.userRoutes(users: UserRepository) = routing {
    authenticate("google-jwt") {
        route("user") {
            get("/") {
                val idToken = call.principal<GoogleIdToken>()!!
                val user = users.findOrNull(idToken.subject)
                if (user != null) call.respond(HttpStatusCode.OK, user)
                else call.respond(HttpStatusCode.NotFound)
            }
        }
    }
}
```



Full Stack Ktor



Questions?

github.com/nomisRev/full-stack-ktor-talk

