Kotlin Backend Service (Ktor + Koin + Exposed + Coroutines)

This document contains a complete Kotlin backend service example using Ktor framework, Koin for dependency injection, Exposed ORM for database operations, and coroutines for asynchronous programming. It demonstrates clean architecture best practices and is production-ready for modern backend development.

Full Kotlin Code

```kotlin
// Entry point of a RESTful backend service in Kotlin using Ktor, Exposed, Coroutines and Koin DI


import io.ktor.application.*
import io.ktor.features.*
import io.ktor.response.*
import io.ktor.request.*
import io.ktor.routing.*
import io.ktor.http.*
import io.ktor.serialization.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*
import kotlinx.coroutines.*
import org.jetbrains.exposed.sql.*
import org.jetbrains.exposed.sql.transactions.transaction
import org.jetbrains.exposed.sql.transactions.experimental.newSuspendedTransaction
import org.koin.core.context.startKoin
import org.koin.dsl.module
import org.koin.ktor.ext.Koin

import org.koin.ktor.ext.inject

import org.koin.logger.slf4jLogger


// -------------- DOMAIN LAYER -------------------- //


data class User(val id: Int, val name: String, val email: String)


interface UserRepository {

    suspend fun getAll(): List<User>

    suspend fun getById(id: Int): User?

    suspend fun create(user: User): User

}


// -------------- DATA LAYER -------------------- //


object Users : Table() {

    val id = integer("id").autoIncrement()

    val name = varchar("name", 255)

    val email = varchar("email", 255)


    override val primaryKey = PrimaryKey(id)

}


class UserRepositoryImpl : UserRepository {
```

```
override suspend fun getAll(): List<User> = dbQuery {
    Users.selectAll().map {
        User(
            id = it[Users.id],
            name = it[Users.name],
            email = it[Users.email]
        )
    }
}

override suspend fun getById(id: Int): User? = dbQuery {
    Users.select { Users.id eq id }.mapNotNull {
        User(
            id = it[Users.id],
            name = it[Users.name],
            email = it[Users.email]
        )
    }.singleOrNull()
}

override suspend fun create(user: User): User = dbQuery {
    val id = Users.insertAndGetId {
        it[name] = user.name
        it[email] = user.email
    }.value
```

```
        user.copy(id = id)
    }

    private suspend fun <T> dbQuery(block: suspend () -> T): T =
        newSuspendedTransaction(Dispatchers.IO) { block() }
}

// -------------- USE CASE / SERVICE LAYER -------------------- //

class UserService(private val repo: UserRepository) {
    suspend fun listUsers(): List<User> = repo.getAll()
    suspend fun getUser(id: Int): User? = repo.getById(id)
    suspend fun addUser(user: User): User = repo.create(user)
}

// -------------- KOIN MODULE -------------------- //

val appModule = module {
    single<UserRepository> { UserRepositoryImpl() }
    single { UserService(get()) }
}

// -------------- KTOR CONFIG -------------------- //

fun Application.module() {
```

```
install(ContentNegotiation) {
    json()
}

install(CallLogging)
install(Koin) {
    slf4jLogger()
    modules(appModule)
}

DatabaseFactory.init()

val userService: UserService by inject()

routing {
    get("/") {
        call.respondText("Kotlin Backend Service Running!", ContentType.Text.Plain)
    }

    route("/users") {
        get {
            val users = userService.listUsers()
            call.respond(users)
        }
```

```
            get("/{id}") {
                val id = call.parameters["id"]?.toIntOrNull()
                if (id == null) {
                    call.respond(HttpStatusCode.BadRequest, "Invalid ID")
                    return@get
                }
                val user = userService.getUser(id)
                if (user == null) {
                    call.respond(HttpStatusCode.NotFound, "User not found")
                } else {
                    call.respond(user)
                }
            }

            post {
                val newUser = call.receive<User>()
                val createdUser = userService.addUser(newUser)
                call.respond(HttpStatusCode.Created, createdUser)
            }
        }
    }
}

// -------------- DATABASE INIT -------------------- //
```

```
object DatabaseFactory {
    fun init() {
        val db = Database.connect(
            url = "jdbc:h2:mem:test;DB_CLOSE_DELAY=-1;",
            driver = "org.h2.Driver"
        )

        transaction(db) {
            SchemaUtils.create(Users)
        }
    }
}

// -------------- MAIN ENTRY POINT -------------------- //

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