Advanced Kotlin Project: "SmartTask – Enterprise Task Manager"

Architecture Overview:

arduino

CopyEdit

smarttask/

├ domain/

│ ├ Task.kt

│ ├ TaskUseCase.kt

├ data/

│ ├ TaskRepositoryImpl.kt

├ presentation/

│ ├ TaskViewModel.kt

├ di/

│ ├ AppModule.kt

├ api/

│ ├ TaskApi.kt

├ Main.kt

---

domain/Task.kt

kotlin

CopyEdit

```
package domain

import kotlinx.serialization.Serializable
import java.util.*

@Serializable
data class Task(
    val id: UUID = UUID.randomUUID(),
    val title: String,
    val description: String,
    val isCompleted: Boolean = false,
    val createdAt: Long = System.currentTimeMillis()
)

sealed class TaskEvent {
    data class Created(val task: Task) : TaskEvent()
    data class Completed(val taskId: UUID) : TaskEvent()
    data class Deleted(val taskId: UUID) : TaskEvent()
}
```

---

domain/TaskUseCase.kt

kotlin

CopyEdit

```
package domain
```

```
import kotlinx.coroutines.flow.Flow

interface TaskUseCase {
    suspend fun createTask(title: String, description: String): Task
    suspend fun completeTask(id: UUID): Boolean
    suspend fun deleteTask(id: UUID): Boolean
    fun getAllTasks(): Flow<List<Task>>
}
```

---

data/TaskRepositoryImpl.kt

kotlin

CopyEdit

```
package data

import domain.Task
import domain.TaskUseCase
import kotlinx.coroutines.flow.*
import java.util.*
import java.util.concurrent.ConcurrentHashMap

class InMemoryTaskRepository : TaskUseCase {

    private val taskStorage = ConcurrentHashMap<UUID, Task>()
    private val tasksFlow = MutableStateFlow<List<Task>>(emptyList())
```

```
private fun emitChanges() {
    tasksFlow.value = taskStorage.values.toList()
}

override suspend fun createTask(title: String, description: String): Task {
    val task = Task(title = title, description = description)
    taskStorage[task.id] = task
    emitChanges()
    return task
}

override suspend fun completeTask(id: UUID): Boolean {
    val task = taskStorage[id] ?: return false
    taskStorage[id] = task.copy(isCompleted = true)
    emitChanges()
    return true
}

override suspend fun deleteTask(id: UUID): Boolean {
    val result = taskStorage.remove(id) != null
    emitChanges()
    return result
}
```

```
    override fun getAllTasks(): Flow<List<Task>> = tasksFlow.asStateFlow()
}
```

---

presentation/TaskViewModel.kt

kotlin

CopyEdit

```
package presentation

import domain.Task
import domain.TaskUseCase
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.*
import java.util.*

class TaskViewModel(private val useCase: TaskUseCase) {

    private val scope = CoroutineScope(Dispatchers.Default + SupervisorJob())

    private val _tasks = MutableStateFlow<List<Task>>(emptyList())
    val tasks: StateFlow<List<Task>> = _tasks

    init {
        observeTasks()
    }
```

```
private fun observeTasks() {
    scope.launch {
        useCase.getAllTasks()
            .onEach { _tasks.value = it }
            .catch { println("Error collecting tasks: ${it.localizedMessage}") }
            .collect()
    }
}

fun create(title: String, description: String) {
    scope.launch {
        useCase.createTask(title, description)
    }
}

fun complete(id: UUID) {
    scope.launch {
        useCase.completeTask(id)
    }
}

fun delete(id: UUID) {
    scope.launch {
        useCase.deleteTask(id)
    }
```

```
    }

    fun clear() {
        scope.cancel()
    }
}
```

---

api/TaskApi.kt (Ktor-based REST API)

kotlin

CopyEdit

```
package api

import domain.TaskUseCase
import io.ktor.application.*
import io.ktor.features.ContentNegotiation
import io.ktor.http.*
import io.ktor.request.*
import io.ktor.response.*
import io.ktor.routing.*
import io.ktor.serialization.*
import io.ktor.server.engine.*
import io.ktor.server.netty.*
import kotlinx.serialization.Serializable
import java.util.*
```

```
@Serializable
data class TaskRequest(val title: String, val description: String)

fun startServer(useCase: TaskUseCase) {
    embeddedServer(Netty, port = 8080) {
        install(ContentNegotiation) {
            json()
        }

        routing {
            route("/tasks") {
                post {
                    val req = call.receive<TaskRequest>()
                    val task = useCase.createTask(req.title, req.description)
                    call.respond(task)
                }

                get {
                    val tasks = useCase.getAllTasks().first()
                    call.respond(tasks)
                }

                put("/{id}/complete") {
                    val id = UUID.fromString(call.parameters["id"])
                    val success = useCase.completeTask(id)
```

```
                call.respond(HttpStatusCode.OK, mapOf("success" to success))
            }

            delete("/{id}") {
                val id = UUID.fromString(call.parameters["id"])
                val success = useCase.deleteTask(id)
                call.respond(HttpStatusCode.OK, mapOf("success" to success))
            }
        }
    }
}.start(wait = true)
}
```

---

di/AppModule.kt (Dependency Injection with Koin)

kotlin

CopyEdit

```
package di

import data.InMemoryTaskRepository
import domain.TaskUseCase
import org.koin.dsl.module
import presentation.TaskViewModel

val appModule = module {
    single<TaskUseCase> { InMemoryTaskRepository() }
```

```
    factory { TaskViewModel(get()) }
}
```

Main.kt

kotlin

CopyEdit

```
import api.startServer
import di.appModule
import org.koin.core.context.startKoin
import org.koin.java.KoinJavaComponent.inject

fun main() {
    startKoin {
        modules(appModule)
    }

    val useCase by inject(domain.TaskUseCase::class.java)
    startServer(useCase)
}
```