SecureGraphQLEngine.kt

Purpose: Secure, reactive, encrypted GraphQL API engine with Kotlin DSL, DI, and coroutine-first I/O.

```
// File: SecureGraphQLServer.kt

package com.maangtech.securegraphql


import kotlinx.coroutines.*

import kotlinx.coroutines.flow.*

import kotlin.reflect.*

import kotlin.reflect.full.*

import java.security.*

import javax.crypto.*

import javax.crypto.spec.SecretKeySpec

import java.util.*

import kotlin.collections.set


// High-Ranked Keywords: Kotlin Secure API, Coroutine Flow, DSL, Dependency Injection, Advanced Generics, GraphQL Kotlin, Kotlin Multiplatform, Kotlin Reflection


// ============================

// Encrypted Data Class

// ============================

data class EncryptedPayload(

    val algorithm: String = "AES",
```

```
    val encryptedBase64: String
)

// ============================
// AES Encryption Utilities
// ============================
object AESCrypto {
    private const val SECRET = "maang_kotlin_secure!"

    private val keySpec = SecretKeySpec(SECRET.toByteArray(), "AES")

    fun encrypt(data: String): EncryptedPayload {
        val cipher = Cipher.getInstance("AES")
        cipher.init(Cipher.ENCRYPT_MODE, keySpec)
        val encrypted = cipher.doFinal(data.toByteArray())
        return EncryptedPayload(encryptedBase64 =
Base64.getEncoder().encodeToString(encrypted))
    }

    fun decrypt(payload: EncryptedPayload): String {
        val cipher = Cipher.getInstance("AES")
        cipher.init(Cipher.DECRYPT_MODE, keySpec)
        val decrypted = cipher.doFinal(Base64.getDecoder().decode(payload.encryptedBase64))
        return String(decrypted)
    }
```

```
}

// ============================
// Annotation for Secured Queries
// ============================
@Target(AnnotationTarget.FUNCTION)
@Retention(AnnotationRetention.RUNTIME)
annotation class SecuredGraphQL(val encrypted: Boolean = true)

// ============================
// Dependency Injection Kernel
// ============================
object DIContainer {
    private val services = mutableMapOf<KClass<*>, Any>()

    fun <T : Any> register(clazz: KClass<T>, instance: T) {
        services[clazz] = instance
    }

    @Suppress("UNCHECKED_CAST")
    fun <T : Any> resolve(clazz: KClass<T>): T =
        services[clazz] as? T ?: error("Service ${clazz.simpleName} not registered.")
}

// ============================
```

```
// Reactive In-Memory Cache
// ============================
object ReactiveCache {
    private val cache = mutableMapOf<String, MutableStateFlow<String?>>()

    fun get(key: String): Flow<String?> =
        cache.getOrPut(key) { MutableStateFlow(null) }

    suspend fun put(key: String, value: String) {
        cache.getOrPut(key) { MutableStateFlow(null) }.emit(value)
    }
}

// ============================
// GraphQL Schema DSL
// ============================
@DslMarker
annotation class GraphQLDsl

@GraphQLDsl
class SchemaBuilder {
    private val queries = mutableMapOf<String, suspend () -> Any>()

    fun query(name: String, block: suspend () -> Any) {
        queries[name] = block
```

```
    }

    fun build(): Map<String, suspend () -> Any> = queries
}

// ============================
//  Advanced GraphQL Engine
// ============================
object SecureGraphQLServer {

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

    fun schema(init: SchemaBuilder.() -> Unit): Flow<String> = flow {
        val builder = SchemaBuilder()
        builder.init()
        val resultMap = builder.build()

        resultMap.forEach { (queryName, resolver) ->
            val method = resolver::class.functions.firstOrNull()
            val secured = method?.annotations?.any { it is SecuredGraphQL } ?: false

            val result = withContext(scope.coroutineContext) {
                val res = resolver()
                if (secured) AESCrypto.encrypt(res.toString()).encryptedBase64
                else res.toString()
```

```
            }

            emit("Query: $queryName -> $result")
        }
    }
}

// ============================
// Sample Service with Secure Query
// ============================
class UserService {
    @SecuredGraphQL
    suspend fun getSecureUserData(): String = "MAANG_user_data_${UUID.randomUUID()}"

    suspend fun getPublicData(): String = "Public_Info_${System.currentTimeMillis()}"
}

// ============================
// Application Entry Point
// ============================
fun main() = runBlocking {
    // Register services
    DIContainer.register(UserService::class, UserService())

    println("SecureGraphQLServer Started...")
```

```
    SecureGraphQLServer.schema {
        val userService = DIContainer.resolve(UserService::class)

        query("publicData") {
            userService.getPublicData()
        }

        query("secureUser") {
            userService.getSecureUserData()
        }
    }.collect { println("$it") }
}
```

Highlights of Advanced Kotlin Usage

Feature	Explanation
Kotlin DSL	Custom GraphQLDsl lets you define queries in a fluent Kotlin way.
Annotations + Reflection	Marks methods as secure using @SecuredGraphQL, resolved via Kotlin reflection.
Coroutine + Flow	Asynchronous query handling using Flow and withContext.
AES Encryption	Encrypts sensitive data using symmetric encryption before output.
Reactive Cache	In-memory StateFlow-based cache with real-time update capabilities.

Feature	Explanation
Custom DI	Lightweight DI container with manual service resolution.
Hexagonal architecture	Business logic separated from transport via SecureGraphQLServer.
High-Level Security & Observability	Combines security, concurrency, and modularity.