Студенческая
ИТ-лаборатория

Написание сетевого и бизнес слоёв для Android и кроссплатформы на Kotlin

Груздев Артём
Android Developer в Effective

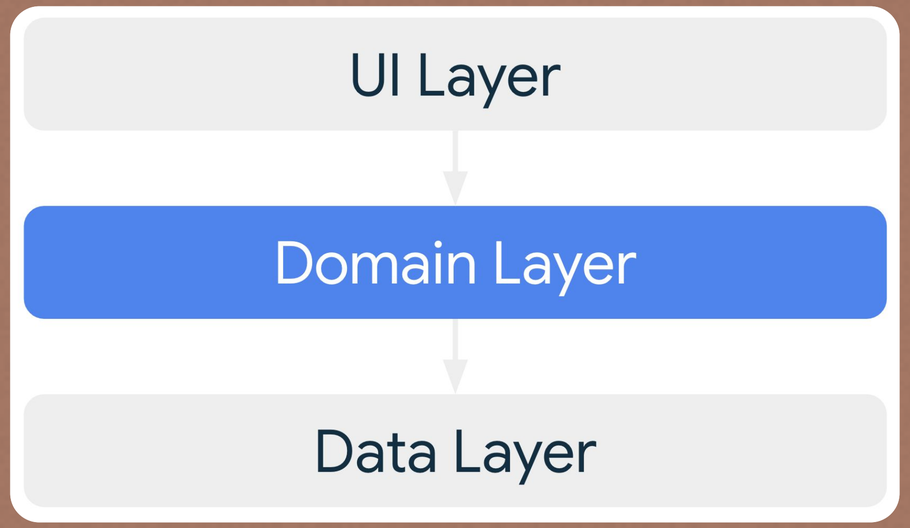
Студенческая
ИТ-лаборатория

effectiveband

Введение.
Data, Domain, UI слои

Слои архитектуры

effectiveband



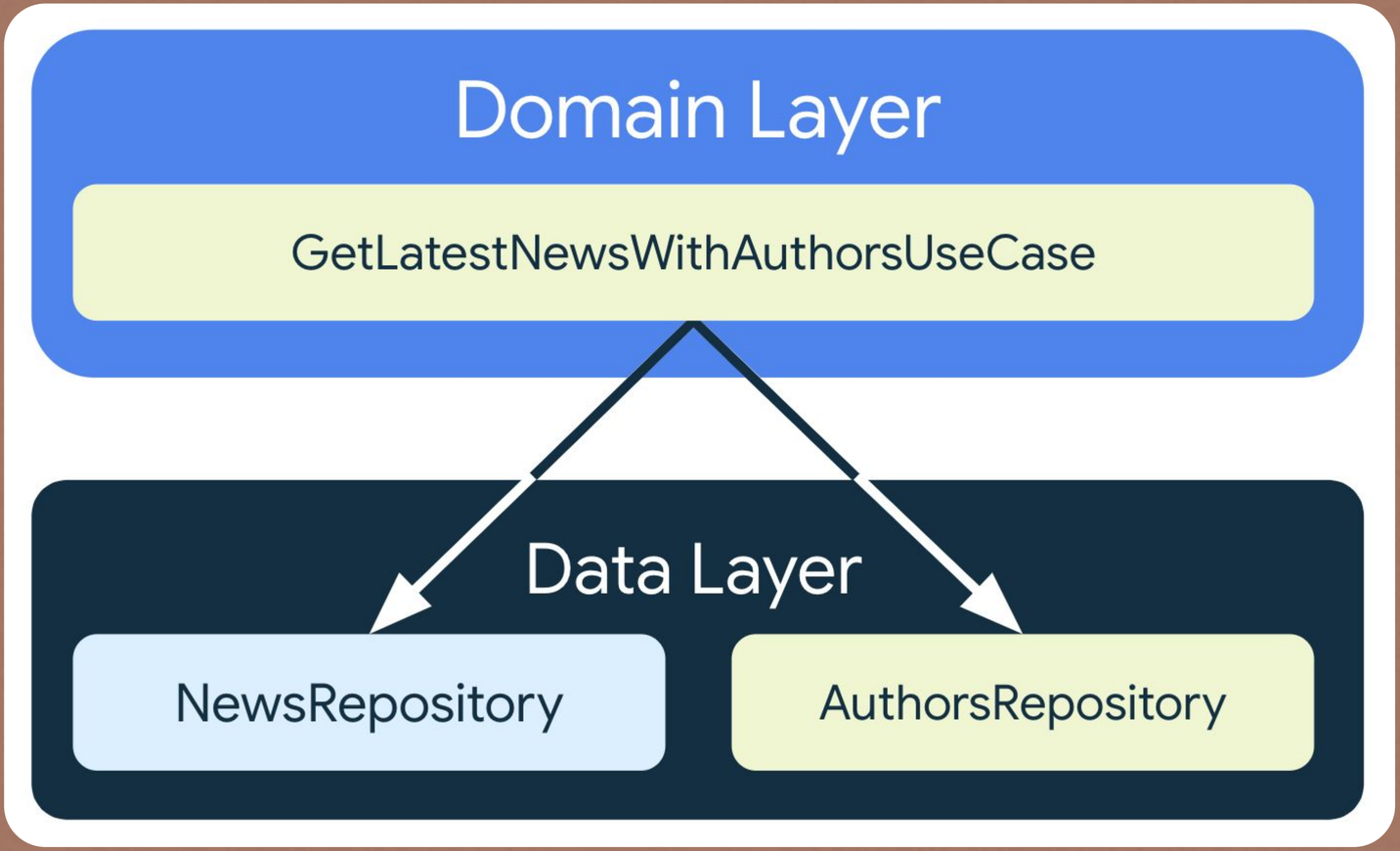
UI Layer – слой отображения данных (composable функции, Fragment, Activity).

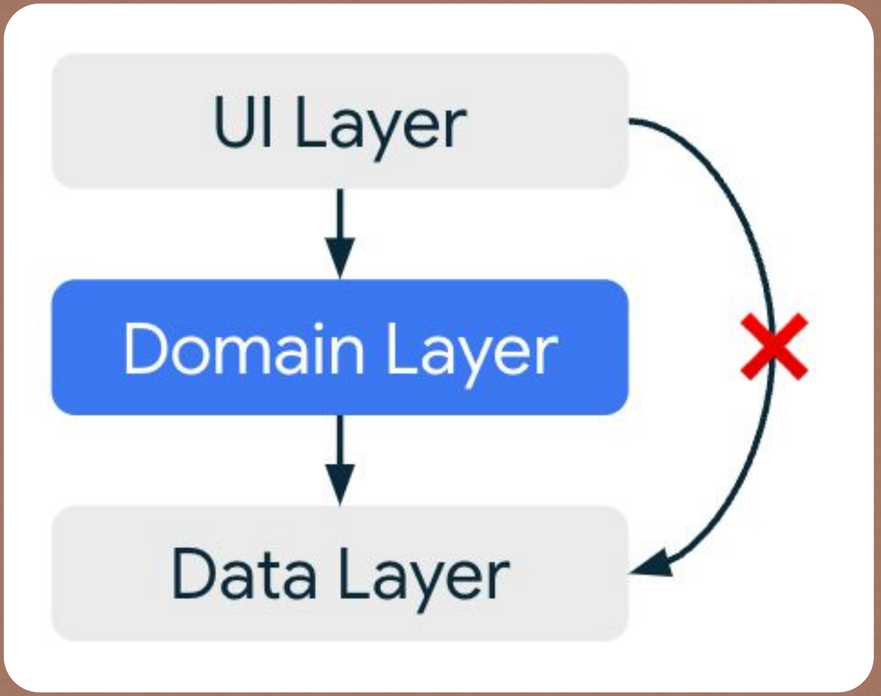
Domain Layer – слой бизнес-логики (UseCase, Interactor).

Data Layer – слой данных. Доступ в сеть, базу данных.

Слои архитектуры

effectiveband





Студенческая
ИТ-лаборатория

effectiveband

Network layer. Native Android

Okhttp

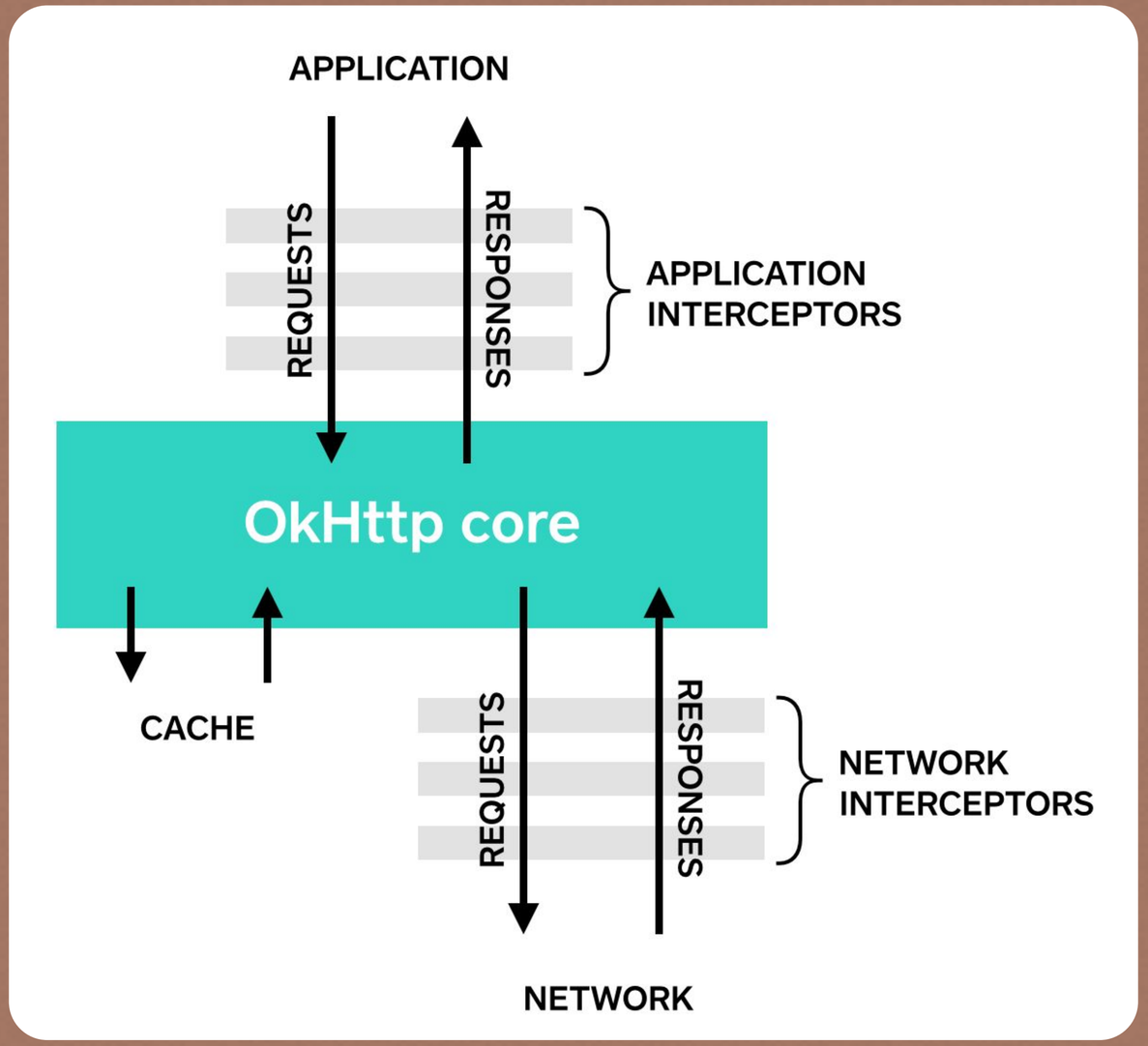
effectiveband

OkHttp-клиент – самый популярный сетевой клиент на Android, поверх которого часто используют Retrofit

🔵 записывать/читать кэш, управлять им с помощью хэдера cache-control
🔵 управлять cookie и разруливать их атрибуты
🔵 устанавливать соединение по http1 и http2 (а по http3 нет, завидуем iOS-разработчикам)
🔵 выбирать сеть, по которой будет выполнен запрос
🔵 работать с DNS и направлять трафик через Proxy

Okhttp

effectiveband



Okhttp. Interceptor

effectiveband

```
class AuthorizationInterceptor @Inject constructor(
    private val preferencesRepository: PreferencesRepository
) : Interceptor {

    override fun intercept(chain: Interceptor.Chain): Response {
        return chain.request()
            .addTokenHeader()
            .let { chain.proceed(it) }
    }

    private fun Request.addTokenHeader(): Request {
        val authHeaderName = "Authorization"
        return newBuilder()
            .apply {
                runBlocking {
                    val token = preferencesRepository.authToken.first()
                    if (token != null) {
                        header(authHeaderName, token)
                    }
                }
            }
            .build()
    }
}
```

Инициализация Okhttp

effectiveband

```
@Provides
@AppScope
fun provideBaseOkHttpClient(
    interceptors: Set<@JvmSuppressWildcards Interceptor>,
): OkHttpClient {
    return OkHttpClient.Builder()
        .apply { interceptors.forEach(::addInterceptor) }
        .callTimeout(50, TimeUnit.SECONDS)
        .build()
}
```

Сериализация JSON

effectiveband

- Moshi
- GSON
- Kotlinx Serialization
- etc...

Сериализация JSON

effectiveband

```
class BlackjackHand(
  val hidden_card: Card,
  val visible_cards: List<Card>,
  ...
)

class Card(
  val rank: Char,
  val suit: Suit
  ...
)

enum class Suit {
  CLUBS, DIAMONDS, HEARTS, SPADES;
}
```

```
{
  "hidden_card": {
    "rank": "6",
    "suit": "SPADES"
  },
  "visible_cards": [
    {
      "rank": "4",
      "suit": "CLUBS"
    },
    {
      "rank": "A",
      "suit": "HEARTS"
    }
```

Call Adapter Factory

effectiveband

Приводит вызов с типом ответа R к типу T.

```
public class EitherCallAdapterFactory private constructor(
  private val coroutineScope: CoroutineScope,
) : CallAdapter.Factory() {

  override fun get(
    returnType: Type,
    annotations: Array<out Annotation>,
    retrofit: Retrofit,
  ): CallAdapter<*, *>? {
    when (getRawType(returnType)) {
      Call::class.java -> {
        val callType = getParameterUpperBound(0, returnType as ParameterizedType)
        val rawType = getRawType(callType)
        if (rawType != Either::class.java) {
          return null
        }

   .....

 }
```

Retrofit. Описание методов

effectiveband

```
interface RetrofitNiaNetworkApi {
    @GET(value = "topics")
    suspend fun getTopics(
        @Query("id") ids: List<String>?,
    ): NetworkResponse<List<NetworkTopic>>

    @POST(value = "some/url/path")
    suspend fun samePostRequest(
        someBody: RequestBody
    ) : SomeResponse
}
```

Инициализация Retrofit

effectiveband

```
    @Provides
    @AppScope
    fun provideBaseRetrofit(
        client: OkHttpClient,
        moshi: Moshi,
    ): Retrofit {
        return Retrofit.Builder()
            .client(client)
            .baseUrl(SOME_BASE_URL)
            .addConverterFactory(MoshiConverterFactory.create(moshi))
            .addCallAdapterFactory(ResultCallAdapterFactory.create())
            .build()
    }
}
```

Студенческая
ИТ-лаборатория

effectiveband

Network layer. Multiplatform

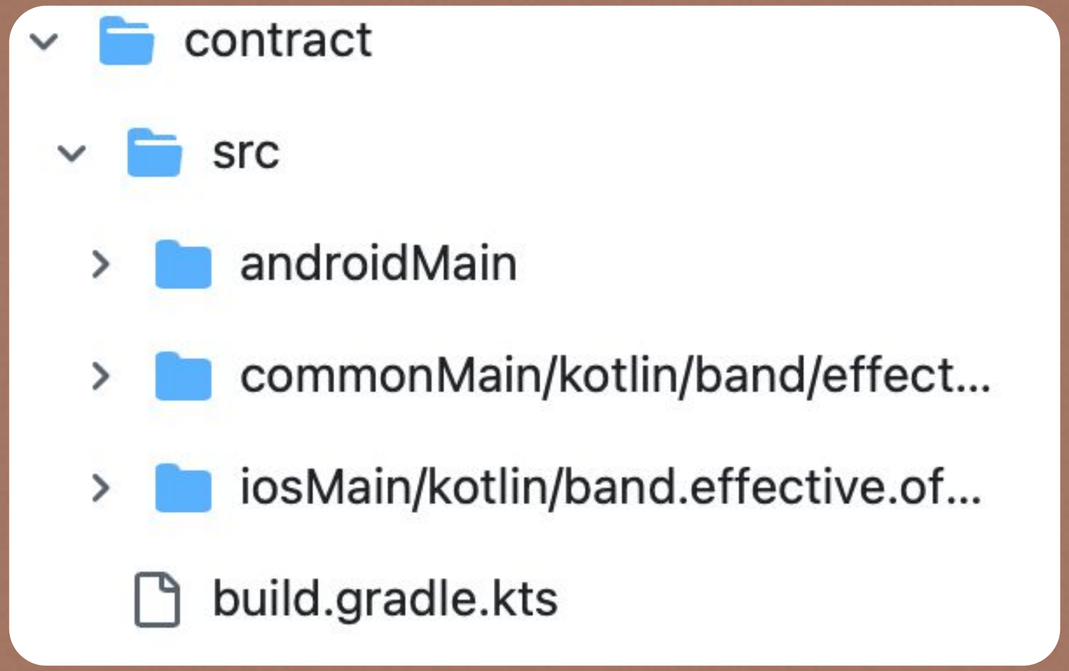
Multiplatform. Expect/Actual

effectiveband

Декларации Expected и Actual позволяют получить доступ к специфичным для платформы API из мультиплатформенных модулей Kotlin. Вы можете предоставлять платформо-агностические API в общем коде.

Multiplatform. Expect/Actual

effectiveband




```
// commonMain
expect fun helloPlatform()

//androidMain
actual fun helloPlatform() {
  Timber.d("Hello Android!")
}

//iosMain
actual fun helloPlatform() {
   NSLog("Hello IOS!")
}

//где-то в commonMain
fun doSomething(){
    helloPlatform()
}
```

Multiplatform. Ktor

effectiveband

Ktor включает в себя многоплатформенный асинхронный HTTP-клиент, который позволяет выполнять запросы и обрабатывать ответы, расширять его функциональность с помощью плагинов.

Multiplatform. Ktor

effectiveband

Engine	Platforms
Apache	JVM
Java	JVM
Jetty	JVM
Android	JVM, Android
OkHttp	JVM, Android
Darwin	Native
WinHttp	Native
Curl	Native
CIO	JVM, Android, Native
Js	JavaScript

Multiplatform. Ktor

effectiveband

```
// commonMain
expect fun createHttpEngine(): HttpClient

// androidMain
actual fun createHttpEngine(): HttpClient = HttpClient(Android)

// iosMain
actual fun createHttpEngine(): HttpClient = HttpClient(Darwin)
```

Multiplatform. Ktor

effectiveband

```
createHttpEngine().config {
    install(Auth) {
        bearer {
            loadTokens {
                BearerTokens(....)
            }
            refreshTokens {
                BearerTokens(....)
            }
        }
  }
```

Multiplatform. Ktor

effectiveband

```
val CustomHeaderPlugin = createClientPlugin("CustomHeaderPlugin") {
    onRequest { request, _ ->
        request.headers.append("X-Custom-Header", "Default value")
    }
}
```

Multiplatform. Ktor

effectiveband

```
import io.ktor.client.plugins.contentnegotiation.*
import io.ktor.serialization.kotlinx.json.*

val client = HttpClient(CIO) {
    install(ContentNegotiation) {
        json()
    }
}
```

Студенческая
ИТ-лаборатория

effectiveband

Data Base. Native and Multiplatform

Multiplatform. SQLDelight

effectiveband

SQLDelight – мультиплатформенное решение для реляционной базы данных. SQLDelight генерирует типобезопасные Kotlin API из ваших SQL-запросов. Он проверяет схему, операторы и миграции во время компиляции и предоставляет такие возможности IDE, как автозаполнение и рефакторинг, которые упрощают написание и поддержку SQL.

SQLDelight. Setup

effectiveband

```
// src/main/sqldelight/com/example/sqldelight/hockey/data/Player.sq

CREATE TABLE hockeyPlayer (
  player_number INTEGER PRIMARY KEY NOT NULL,
  full_name TEXT NOT NULL
);

CREATE INDEX hockeyPlayer_full_name ON hockeyPlayer(full_name);

INSERT INTO hockeyPlayer (player_number, full_name)
VALUES (15, 'Ryan Getzlaf');
```

SQLDelight. Setup

effectiveband

```
// commonMain
expect class DriverFactory {
  fun createDriver(): SqlDriver
}

// androidMain
actual class DriverFactory(private val context: Context) {
  actual fun createDriver(): SqlDriver {
    return AndroidSqliteDriver(Database.Schema, context, "test.db")
  }
}

// iosMain
actual class DriverFactory {
  actual fun createDriver(): SqlDriver {
    return NativeSqliteDriver(Database.Schema, "test.db")
  }
}
```

SQLDelight. Setup

effectiveband

```
fun doDatabaseThings(driver: SqlDriver) {
  val database = Database(driver)
  val playerQueries: PlayerQueries = database.playerQueries

  println(playerQueries.selectAll().executeAsList())
  // [HockeyPlayer(15, "Ryan Getzlaf")]

  playerQueries.insert(player_number = 10, full_name = "Corey Perry")
  println(playerQueries.selectAll().executeAsList())
  // [HockeyPlayer(15, "Ryan Getzlaf"), HockeyPlayer(10, "Corey Perry")]

  val player = HockeyPlayer(10, "Ronald McDonald")
  playerQueries.insertFullPlayerObject(player)
}
```

Multiplatform. Room

effectiveband

Библиотека Room предоставляет слой абстракции над SQLite, чтобы обеспечить свободный доступ к базам данных, используя при этом всю мощь SQLite.

В частности, Room обеспечивает следующие преимущества:

- Проверка SQL-запросов во время компиляции;
- Удобные аннотации, которые минимизируют повторяющийся и подверженный ошибкам код;
- Оптимизированные пути миграции баз данных.

Room?
Почему Multiplatform?

Room

effectiveband

Version 2.7.0-alpha01

May 1, 2024

`androidx.room:room-*:2.7.0-alpha01` is released. Version 2.7.0-alpha01 contains these commits.

New Features

- **Kotlin Multiplatform (KMP) Support**: In this release, Room has been refactored to become a Kotlin Multiplatform (KMP) library. Although there is still some work to be done, this release introduces a new version of Room where the majority of the functionality has been “common-ized” (made to be multiplatform). Current supported platforms are Android, iOS, JVM (Desktop), native Mac and native Linux. Any missing functionality in the newly supported platforms will be made “feature-complete” in upcoming Room releases.

Room

effectiveband

```
// commonMain

@Database(entities = [TodoEntity::class], version = 1)
@ConstructedBy(AppDatabaseConstructor::class)
abstract class AppDatabase : RoomDatabase() {
  abstract fun getDao(): TodoDao
}

// The Room compiler generates the `actual` implementations.
@Suppress("NO_ACTUAL_FOR_EXPECT")
expect object AppDatabaseConstructor : RoomDatabaseConstructor<AppDatabase> {
    override fun initialize(): AppDatabase
}
```

Room

effectiveband

```
// commonMain

@Dao
interface TodoDao {
  @Insert
  suspend fun insert(item: TodoEntity)

  @Query("SELECT count(*) FROM TodoEntity")
  suspend fun count(): Int

  @Query("SELECT * FROM TodoEntity")
  fun getAllAsFlow(): Flow<List<TodoEntity>>
}

@Entity
data class TodoEntity(
  @PrimaryKey(autoGenerate = true) val id: Long = 0,
  val title: String,
  val content: String
)
```

Room

```kotlin
// androidMain
fun getDatabaseBuilder(ctx: Context): RoomDatabase.Builder<AppDatabase> {
  val appContext = ctx.applicationContext
  val dbFile = appContext.getDatabasePath("my_room.db")
  return Room.databaseBuilder<AppDatabase>(
    context = appContext,
    name = dbFile.absolutePath
  )
}

// iosMain
fun getDatabaseBuilder(): RoomDatabase.Builder<AppDatabase> {
    val dbFilePath = documentDirectory() + "/my_room.db"
    return Room.databaseBuilder<AppDatabase>(
        name = dbFilePath,
    )
}

private fun documentDirectory(): String {
  val documentDirectory = NSFileManager.defaultManager.URLForDirectory(
    directory = NSDocumentDirectory,
    inDomain = NSUserDomainMask,
    appropriateForURL = null,
    create = false,
    error = null,
  )
  return requireNotNull(documentDirectory?.path)
}
```

effectiveband

Студенческая
ИТ-лаборатория

effectiveband

Domain Layer.
Summarize the results

Domain Layer. UseCase

effectiveband

```
class MyUseCase(
    private val defaultDispatcher: CoroutineDispatcher = Dispatchers.Default
) {

    suspend operator fun invoke(...) = withContext(defaultDispatcher) {
        // Long-running blocking operations happen on a background thread.
    }
}

class MyCoolViewModel @Inject constructor(useCase: MyUseCase) : ViewModel() {
   ....
   viewModelScope.launch {
     val result = useCase()
   }
   ....
}
```

Repository

effectiveband

```
interface UserRepository {
    suspend fun getUserById(id: Int): User
}

class UserRepositoryImpl(
  // local storage
  private val userDao: UserDao,
  // network
  private val userService: UserService
) : UserRepository {
    ....
}
```

References

effectiveband

// Тут ссылки на примеры кода в GitHub