Tecno Ciencia — Manual Completo

```markdown

Tecno Ciencia - Manual Completo

Versión: 1.0

Paquete: com.hugoguerrero.tecno

📋 Resumen

Tecno Ciencia es una aplicación para subir, documentar y compartir proyectos tecnológicos y científicos. Este README sigue la

"Anatomía de un README.md Perfecto" y contiene: descripción del proyecto, arquitectura, código documentado (KDoc), pasos para

ejecutar, y espacio para tus capturas de pantalla en docs/screenshots/.

---

📖 Tabla de Contenidos



---

🎯 Introducción
```

Tecno Ciencia es un espacio para que desarrolladores y estudiantes suban sus proyectos, obtengan retroalimentación y colaboren.

Este README contiene ejemplos de código con KDoc, la estructura del proyecto y referencias a las capturas que debes subir

en docs/screenshots/.

📊 Estado del Proyecto

- ✅ Plantillas y estructura listas en el repositorio.
- ✅ Código ejemplo con KDoc incluido en secciones abajo (extractos disponibles).
- ⚠️ Faltan implementaciones completas de backend (ApiService es placeholder).
- 📸 Añade tus capturas en docs/screenshots/ para completar la entrega.

📱 ¿Qué es la App?

Tecno Ciencia permite crear proyectos, subir documentos, organizar por categorías, contribuir y reportar incidencias.

Está pensada para estudiantes y equipos pequeños que quieran compartir avances y recibir feedback.

Características principales:

- Autenticación con Firebase (email/Google)
- CRUD completo de proyectos
- Sistema de donaciones con comprobantes
- Perfiles de usuario con roles
- Notificaciones push
- Reporte de problemas

🔄 ¿Cómo Funciona y Quién lo Hizo?

Flujo de la Aplicación:

1. Login/Registro con Firebase Auth

2. Pantalla principal con lista de proyectos
3. Ver proyecto → documentos, contribuciones, autor
4. Perfil del usuario y gestión (admin)
5. Sistema de donaciones con aprobación manual
6. Reporte de incidencias para moderación

Equipo:

- Hugo Guerrero - Desarrollador principal
- Jorge Alberto (betoblack) - Desarrrollador y Colaborador

🛠️ Tecnologías Usadas

- Lenguaje: Kotlin 1.9.0
- UI: Jetpack Compose 1.5.0
- Arquitectura: MVVM + Clean Architecture
- Base de datos: Firebase Firestore
- Autenticación: Firebase Auth
- Almacenamiento: Firebase Storage
- Notificaciones: Firebase Cloud Messaging
- DI: Dagger Hilt
- Imágenes: Coil
- Navegación: Navigation Compose
- Serialización: Kotlin Serialization

📁 Estructura de Archivos

text

```
com/hugoguerrero/tecno/
├── 📁 data/
│   ├── model/                # Modelos de datos
```

```
│   │   ├── Project.kt
│   │   ├── UserProfile.kt
│   │   ├── Donation.kt
│   │   └── ...
│   ├── remote/                # Fuentes de datos remotas
│   │   ├── firebase/
│   │   │   ├── AuthService.kt
│   │   │   ├── ProjectRemoteDataSourceImpl.kt
│   │   │   └── ...
│   │   └── interfaces/        # Contratos de servicios remotos
│   └── repository/            # Implementaciones de repositorios
│       ├── AuthRepository.kt
│       ├── ProjectRepository.kt
│       └── ...
├── 📁 domain/
│   ├── model/                 # Modelos de dominio
│   │   ├── Project.kt
│   │   ├── User.kt
│   │   └── ...
│   ├── repository/            # Interfaces de repositorios
│   └── use_case/              # Casos de uso
│       ├── project/
│       │   ├── GetProjectsUseCase.kt
│       │   ├── CreateProjectUseCase.kt
│       │   └── ...
│       └── user/
│           ├── GetUserProfileUseCase.kt
```

```
│       └── ...
├── 📁 ui/
│   ├── screens/              # Pantallas de la aplicación
│   │   ├── login/
│   │   ├── main/
│   │   ├── project/
│   │   ├── profile/
│   │   ├── donation/
│   │   ├── management/
│   │   └── complaint/
│   ├── components/           # Componentes reutilizables
│   │   ├── ProjectCard.kt
│   │   └── BottomNavigationBar.kt
│   ├── navigation/           # Navegación
│   │   ├── Screens.kt
│   │   └── MainNavGraph.kt
│   └── theme/                # Temas y estilos
├── 📁 di/                    # Inyección de Dependencias
│   ├── AppModule.kt
│   └── RepositoryModule.kt
└── 📁 docs/                  # Documentación
    └── screenshots/          # Capturas de pantalla
```

Configuracion Build.gradle.kts(Project)

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
```

```
plugins {
    alias(libs.plugins.android.application) apply false
    alias(libs.plugins.kotlin.android) apply false
    // La siguiente línea se elimina porque no es necesaria y causa conflictos
    // alias(libs.plugins.kotlin.compose) apply false
    alias(libs.plugins.hilt) apply false
    alias(libs.plugins.ksp) apply false
    alias(libs.plugins.google.services) apply false
}
```

Configuracion Build.gradle.kts(Module)

```
import java.io.FileInputStream
import java.util.Properties

plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.ksp)
    alias(libs.plugins.hilt)
    alias(libs.plugins.google.services)
    alias(libs.plugins.compose.compiler)
}

// Leer propiedades locales
val localProperties = Properties()
val localPropertiesFile = rootProject.file("local.properties")
```

```
if (localPropertiesFile.exists()) {
    localProperties.load(FileInputStream(localPropertiesFile))
}

android {
    namespace = "com.hugoguerrero.tecno"
    compileSdk = 36

    signingConfigs {
        create("release") {
            keyAlias = localProperties.getProperty("keyAlias")
            keyPassword = localProperties.getProperty("keyPassword")
            storeFile = file(localProperties.getProperty("storeFile"))
            storePassword = localProperties.getProperty("storePassword")
        }
    }

    defaultConfig {
        applicationId = "com.hugoguerrero.tecno"
        minSdk = 24
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"
        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
```

```
        release {
            isMinifyEnabled = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
            signingConfig = signingConfigs.getByName("release")
        }
    }

    buildFeatures {
        compose = true
        buildConfig = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = "1.5.11" // Versión compatible
    }

    kotlin {
        jvmToolchain(17)
    }

    packaging {
        resources.excludes += "/META-INF/{AL2.0,LGPL2.1}"
    }
}
```

```
dependencies {

    // --- Compose BOM ---
    implementation(platform(libs.androidx.compose.bom))

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(libs.androidx.compose.ui)
    implementation(libs.androidx.compose.material3)
    implementation(libs.androidx.compose.ui.tooling.preview)

    // Navigation Compose
    implementation(libs.androidx.compose.navigation)

    // --- Firebase BOM ---
    implementation(platform(libs.firebase.bom))
    implementation(libs.firebase.auth.ktx)
    implementation(libs.firebase.firestore.ktx)
    implementation(libs.firebase.storage.ktx)
    implementation(libs.firebase.messaging.ktx)
    implementation(libs.firebase.functions.ktx)
    implementation(libs.firebase.appcheck.ktx)
    implementation(libs.firebase.appcheck.playintegrity)
    debugImplementation(libs.firebase.appcheck.debug) // <-- AÑADIDO
```

```
    // Firestore Offline Persistence
    implementation(libs.coroutines.play.services)

    // --- Hilt + KSP ---
    implementation(libs.hilt.android)
    implementation(libs.hilt.navigation.compose)
    implementation(libs.androidx.compose.foundation)
    implementation(libs.androidx.compose.ui.text)
    ksp(libs.hilt.compiler)

    // ... tus otras dependencias (Compose, Firebase, Hilt, etc.)// AÑADE ESTA LÍNEA para solucionar el error de reflexión
    implementation("org.jetbrains.kotlin:kotlin-reflect:1.9.24") // Asegúrate de que la versión coincida con la de tu proyecto

    // App Check
    implementation("com.google.firebase:firebase-appcheck-debug")


    // Debug tools (Referencias corregidas)
    debugImplementation("androidx.compose.ui:ui-tooling")
    debugImplementation("androidx.compose.ui:ui-test-manifest")

    // --- TESTING ---
    testImplementation(libs.junit)

    //Icons
    implementation("androidx.compose.material:material-icons-extended:1.7.8")
```

```
    implementation("io.coil-kt:coil-compose:2.7.0")

    //Google
    implementation("com.google.android.gms:play-services-auth:21.4.0")

    androidTestImplementation(platform(libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.test.ext.junit)
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(libs.androidx.compose.ui.test.junit4)
}
```

Configuracion libs.version.toml

```
[versions]
# Android Gradle Plugin y Kotlin
agp = "8.13.1"
kotlin = "2.2.21"
ksp = "2.2.21-2.0.4"

# Compose
activityCompose = "1.12.0"
composeBom = "2025.11.01"
composeCompiler = "2.0.0-Beta02"
navigationCompose = "2.9.6"
foundation = "1.9.5"
```

```
# AndroidX Core
coreKtx = "1.17.0"
lifecycleRuntimeKtx = "2.10.0"

# Hilt
hilt = "2.57.2"
hiltNavigationCompose = "1.3.0"

# Firebase
firebaseBom = "33.0.0"
firebaseFunctions = "21.2.1"
firebaseAppCheck = "18.0.0"

# Google y Coroutines
playServicesAuth = "21.4.0"
coroutines = "1.10.2"

# Testing
junit = "4.13.2"
junitExt = "1.3.0"
espresso = "3.7.0"

[libraries]
# AndroidX & Compose
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx"
}
```

androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version.ref = "lifecycleRuntimeKtx" }

androidx-activity-compose = { group = "androidx.activity", name = "activity-compose", version.ref = "activityCompose" }

androidx-compose-bom = { group = "androidx.compose", name = "compose-bom", version.ref = "composeBom" }

androidx-compose-ui = { group = "androidx.compose.ui", name = "ui" }

androidx-compose-material3 = { group = "androidx.compose.material3", name = "material3" }

androidx-compose-ui-tooling-preview = { group = "androidx.compose.ui", name = "ui-tooling-preview" }

androidx-compose-navigation = { group = "androidx.navigation", name = "navigation-compose", version.ref = "navigationCompose" }

androidx-compose-foundation = { group = "androidx.compose.foundation", name = "foundation", version.ref = "foundation" }

androidx-compose-ui-text = { group = "androidx.compose.ui", name = "ui-text" }

androidx-compose-foundation-layout = { module = "androidx.compose.foundation:foundation-layout", version.ref = "foundation" }

Hilt

hilt-android = { group = "com.google.dagger", name = "hilt-android", version.ref = "hilt" }

hilt-compiler = { group = "com.google.dagger", name = "hilt-compiler", version.ref = "hilt" }

hilt-navigation-compose = { group = "androidx.hilt", name = "hilt-navigation-compose", version.ref = "hiltNavigationCompose" }

Firebase

firebase-bom = { group = "com.google.firebase", name = "firebase-bom", version.ref = "firebaseBom" }

firebase-auth-ktx = { group = "com.google.firebase", name = "firebase-auth-ktx" }

firebase-firestore-ktx = { group = "com.google.firebase", name = "firebase-firestore-ktx" }

firebase-storage-ktx = { group = "com.google.firebase", name = "firebase-storage-ktx" }

firebase-messaging-ktx = { group = "com.google.firebase", name = "firebase-messaging-ktx" }

firebase-functions-ktx = { group = "com.google.firebase", name = "firebase-functions-ktx", version.ref = "firebaseFunctions" }

firebase-appcheck-ktx = { group = "com.google.firebase", name = "firebase-appcheck-ktx", version.ref = "firebaseAppCheck" }

firebase-appcheck-playintegrity = { group = "com.google.firebase", name = "firebase-appcheck-playintegrity", version.ref = "firebaseAppCheck" }

firebase-appcheck-debug = { group = "com.google.firebase", name = "firebase-appcheck-debug", version.ref = "firebaseAppCheck" }

Google & Coroutines

play-services-auth = { group = "com.google.android.gms", name = "play-services-auth", version.ref = "playServicesAuth" }

coroutines-play-services = { group = "org.jetbrains.kotlinx", name = "kotlinx-coroutines-play-services", version.ref = "coroutines" }

Debug Tools

debug-androidx-compose-ui-tooling = { group = "androidx.compose.ui", name = "ui-tooling" }

debug-androidx-compose-ui-test-manifest = { group = "androidx.compose.ui", name = "ui-test-manifest" }

Testing

junit = { group = "junit", name = "junit", version.ref = "junit" }

androidx-test-ext-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitExt" }

androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref = "espresso" }

androidx-compose-ui-test-junit4 = { group = "androidx.compose.ui", name = "ui-test-junit4" }

[plugins]

android-application = { id = "com.android.application", version.ref = "agp" }

kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }

... otros plugins

--- AÑADE ESTA LÍNEA ---

compose-compiler = { id = "org.jetbrains.kotlin.plugin.compose", version.ref = "kotlin" }

ksp = { id = "com.google.devtools.ksp", version.ref = "ksp" }

hilt = { id = "com.google.dagger.hilt.android", version.ref = "hilt" }

google-services = { id = "com.google.gms.google-services", version = "4.4.4" }

```

```

Ruta: app/src/main/java/com/hugoguerrero/tecno/data/model/

```kotlin

```

AuthResultDto

```

package com.hugoguerrero.tecno.data.model

data class AuthResultDto(
```

```
    val userId: String? = null,
    val email: String? = null,
    val isNewUser: Boolean? = null
)
```

Complaint

```
package com.hugoguerrero.tecno.data.model

import com.google.firebase.firestore.ServerTimestamp
import java.util.Date

data class Complaint(
    val id: String = "",
    val projectId: String = "",
    val reporterUid: String = "",
    val reason: String = "",
    val description: String = "",
    @ServerTimestamp
    val createdAt: Date? = null
)
```

Donation

```
package com.hugoguerrero.tecno.data.model

import com.google.firebase.firestore.ServerTimestamp
```

```
import java.util.Date

enum class DonationStatus {
    PENDING,
    COMPLETED,
    FAILED
}

data class Donation(
    val id: String = "",
    val projectId: String = "",
    val donorUid: String = "",
    val amount: Double = 0.0,
    val message: String = "",
    val status: DonationStatus = DonationStatus.PENDING,
    val proofImageUrl: String = "", // Nuevo campo para la URL del comprobante
    @ServerTimestamp
    val createdAt: Date? = null
)
```

DonationDTO

```
package com.hugoguerrero.tecno.data.model

data class DonationDto(
    val id: String? = null,
    val projectId: String? = null,
```

```
    val userId: String? = null,
    val amount: Double? = null,
    val paymentId: String? = null,
    val createdAt: Long? = null
)
```

LoginRequestDto

```
package com.hugoguerrero.tecno.data.model

data class LoginRequestDto(
    val email: String,
    val password: String
)
```

NotificationDto

```
package com.hugoguerrero.tecno.data.model

data class NotificationDto(
    val id: String? = null,
    val userId: String? = null,          // A quién va dirigida
    val title: String? = null,
    val message: String? = null,
    val type: String? = null,            // "donation", "project_approved", etc.
    val projectId: String? = null,       // opcional si aplica
    val isRead: Boolean = false,
```

```
    val createdAt: Long? = null
)
```

PaymentDto

```
package com.hugoguerrero.tecno.data.model

data class PaymentDto(
    val id: String? = null,
    val userId: String? = null,
    val projectId: String? = null,
    val amount: Double? = null,
    val method: String? = null,
    val transactionId: String? = null,
    val status: String? = null,
    val createdAt: Long? = null
)
```

Project

```
package com.hugoguerrero.tecno.data.model

import com.google.firebase.firestore.Exclude
import com.google.firebase.firestore.IgnoreExtraProperties
import com.google.firebase.firestore.ServerTimestamp
import java.util.Date
```

```
@IgnoreExtraProperties // Anotación para ignorar campos desconocidos
data class Project(
    val id: String = "",
    val title: String = "",
    val description: String = "",
    val ownerUid: String = "",
    val category: String = "",
    val institution: String = "",
    val fundingGoal: Double = 0.0,
    val currentFunding: Double = 0.0,
    val imageUrl: String = "",
    val documentUrls: List<String> = emptyList(),
    val tags: List<String> = emptyList(),
    @ServerTimestamp
    val createdAt: Date? = null
) {
    @get:Exclude
    val progress: Float
        get() = if (fundingGoal > 0) ((currentFunding / fundingGoal) * 100).toFloat() else 0f
}
```

ProjectDTO

```
package com.hugoguerrero.tecno.data.model

data class ProjectDto(
    val id: String? = null,
```

```
    val ownerId: String? = null,
    val title: String? = null,
    val description: String? = null,
    val category: String? = null,
    val imageUrl: String? = null,
    val goal: Double? = null,
    val currentAmount: Double? = null,
    val createdAt: Long? = null
)
```

RegisterRequestDto

```
package com.hugoguerrero.tecno.data.model

data class RegisterRequestDto(
    val email: String,
    val password: String,
    val name: String,
    val lastName: String,
    val phone: String
)
```

UserDto

```
package com.hugoguerrero.tecno.data.model

data class UserDto(
```

```
    val id: String? = null,
    val email: String? = null,
    val name: String? = null,
    val photoUrl: String? = null,
    val createdAt: Long? = null,
    val updatedAt: Long? = null
)
```

UserProfile

```
package com.hugoguerrero.tecno.data.model

import com.google.firebase.firestore.IgnoreExtraProperties
import com.google.firebase.firestore.ServerTimestamp
import com.hugoguerrero.tecno.domain.model.UserType
import java.util.Date

@IgnoreExtraProperties // Ignora campos desconocidos al leer de Firestore
data class UserProfile(
    val uid: String = "",
    val displayName: String = "",
    val email: String = "",
    val photoUrl: String = "",
    val bio: String = "",
    val university: String = "",
    val userType: UserType = UserType.STUDENT,
    val fcmToken: String = "",
```

```
    val paymentDetails: String = "",
    @ServerTimestamp
    val createdAt: Date = Date()
)
```

UserStats

```
package com.hugoguerrero.tecno.data.model

data class UserStats(
    val donationsCount: Int = 0,
    val successRate: Int = 0
)
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/data/remote/firebase/

```kotlin

```

AuthService

```
package com.hugoguerrero.tecno.data.remote.firebase

import com.google.firebase.auth.FirebaseAuth
import jakarta.inject.Inject
import kotlinx.coroutines.tasks.await
```

```
class AuthService @Inject constructor(
    private val auth: FirebaseAuth
) {
    suspend fun login(email: String, password: String) =
        auth.signInWithEmailAndPassword(email, password).await()

    suspend fun register(email: String, password: String) =
        auth.createUserWithEmailAndPassword(email, password).await()

    fun logout() = auth.signOut()

    fun currentUser() = auth.currentUser
}
```

DonationRemoteDataSourceImpl

```
package com.hugoguerrero.tecno.data.remote.firebase

import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.Query
import com.hugoguerrero.tecno.data.model.DonationDto
import com.hugoguerrero.tecno.data.remote.DonationRemoteDataSource
import kotlinx.coroutines.tasks.await

class DonationRemoteDataSourceImpl(
    private val firestore: FirebaseFirestore
) : DonationRemoteDataSource {
```

```
private val donationsRef = firestore.collection("donations")


override suspend fun addDonation(donation: DonationDto): Boolean {
    return try {
        val newDoc = donationsRef.document()
        val donationWithId = donation.copy(id = newDoc.id)
        newDoc.set(donationWithId).await()
        true
    } catch (e: Exception) {
        false
    }
}


override suspend fun getDonationsByProject(projectId: String): List<DonationDto> {
    return try {
        donationsRef
            .whereEqualTo("projectId", projectId)
            .orderBy("createdAt", Query.Direction.DESCENDING)
            .get()
            .await()
            .toObjects(DonationDto::class.java)
    } catch (e: Exception) {
        emptyList()
    }
```

```
    }

    override suspend fun getDonationsByUser(userId: String): List<DonationDto> {
        return try {
            donationsRef
                .whereEqualTo("userId", userId)
                .orderBy("createdAt", Query.Direction.DESCENDING)
                .get()
                .await()
                .toObjects(DonationDto::class.java)
        } catch (e: Exception) {
            emptyList()
        }
    }
}
```

FirestoreService

```
package com.hugoguerrero.tecno.data.remote.firebase

import com.google.firebase.firestore.FirebaseFirestore
import com.hugoguerrero.tecno.data.model.ProjectDto
import com.hugoguerrero.tecno.data.model.UserDto
import jakarta.inject.Inject
import kotlinx.coroutines.tasks.await

class FirestoreService @Inject constructor(
```

```
    private val db: FirebaseFirestore
) {

    suspend fun getUser(uid: String): UserDto? {
        return db.collection("users")
            .document(uid)
            .get()
            .await()
            .toObject(UserDto::class.java)
    }

    suspend fun saveProject(project: ProjectDto) {
        project.id?.let { projectId -> // Comprobación segura
            db.collection("projects")
                .document(projectId)
                .set(project)
                .await()
        }
    }
}
```

MyFirebaseMessagingService

```
package com.hugoguerrero.tecno.data.remote.firebase

import android.app.NotificationChannel
import android.app.NotificationManager
```

```
import android.content.Context
import android.os.Build
import androidx.core.app.NotificationCompat
import com.google.firebase.messaging.FirebaseMessagingService
import com.google.firebase.messaging.RemoteMessage
import com.hugoguerrero.tecno.R
import com.hugoguerrero.tecno.data.repository.AuthRepository
import com.hugoguerrero.tecno.domain.use_case.UpdateFcmTokenUseCase
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import javax.inject.Inject

@AndroidEntryPoint
class MyFirebaseMessagingService : FirebaseMessagingService() {

    @Inject
    lateinit var updateFcmTokenUseCase: UpdateFcmTokenUseCase

    @Inject
    lateinit var authRepository: AuthRepository

    override fun onMessageReceived(remoteMessage: RemoteMessage) {
        super.onMessageReceived(remoteMessage)
        remoteMessage.notification?.let { notification ->
            sendNotification(notification.title, notification.body)
```

```
        }
    }

    private fun sendNotification(title: String?, messageBody: String?) {
        val channelId = "default_channel_id"
        val notificationBuilder = NotificationCompat.Builder(this, channelId)
            .setSmallIcon(R.drawable.ic_launcher_foreground) // Asegúrate de tener este drawable
            .setContentTitle(title)
            .setContentText(messageBody)
            .setAutoCancel(true)
            .setPriority(NotificationCompat.PRIORITY_HIGH)

        val notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

        // Desde Android Oreo (API 26) se necesita un NotificationChannel.
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel(
                channelId,
                "Notificaciones Generales",
                NotificationManager.IMPORTANCE_DEFAULT
            )
            notificationManager.createNotificationChannel(channel)
        }

        notificationManager.notify(0 /* ID de la notificación */, notificationBuilder.build())
    }
```

```
    override fun onNewToken(token: String) {
        super.onNewToken(token)

        val userId = authRepository.getCurrentUser()?.uid
        if (userId != null) {
            CoroutineScope(Dispatchers.IO).launch {
                updateFcmTokenUseCase(userId, token)
            }
        }
    }
}
```

NotificationRemoteDataSourceImpl

```
package com.hugoguerrero.tecno.data.remote.firebase

import com.google.firebase.firestore.FirebaseFirestore
import com.hugoguerrero.tecno.data.model.NotificationDto
import com.hugoguerrero.tecno.data.remote.NotificationRemoteDataSource
import kotlinx.coroutines.tasks.await
import javax.inject.Inject

class NotificationRemoteDataSourceImpl @Inject constructor(
    private val firestore: FirebaseFirestore
) : NotificationRemoteDataSource {
```

```
private val notificationsCollection = firestore.collection("notifications")
private val tokensCollection = firestore.collection("user_tokens")

override suspend fun saveUserNotificationToken(userId: String, token: String) {
    try {
        val data = mapOf("token" to token, "updatedAt" to System.currentTimeMillis())
        tokensCollection.document(userId).set(data).await()
    } catch (e: Exception) {
        // Manejo de error si quieres loguear
    }
}

override suspend fun sendNotification(notification: NotificationDto): Boolean {
    return try {
        val id = notificationsCollection.document().id
        val newNotification = notification.copy(
            id = id,
            createdAt = System.currentTimeMillis(),
            isRead = false
        )
        notificationsCollection.document(id).set(newNotification).await()
        true
    } catch (e: Exception) {
        false
    }
}
```

```
    override suspend fun getNotifications(userId: String): List<NotificationDto> {
        return try {
            notificationsCollection
                .whereEqualTo("userId", userId)
                .orderBy("createdAt")
                .get()
                .await()
                .toObjects(NotificationDto::class.java)
        } catch (e: Exception) {
            emptyList()
        }
    }

    override suspend fun markAsRead(notificationId: String): Boolean {
        return try {
            notificationsCollection
                .document(notificationId)
                .update("isRead", true)
                .await()
            true
        } catch (e: Exception) {
            false
        }
    }
}
```

ProjectRemoteDataSourceImpl

```
package com.hugoguerrero.tecno.data.remote.firebase

import com.google.firebase.firestore.FirebaseFirestore
import com.hugoguerrero.tecno.data.model.ProjectDto
import com.hugoguerrero.tecno.data.remote.ProjectRemoteDataSource
import kotlinx.coroutines.tasks.await

class ProjectRemoteDataSourceImpl(
    private val firestore: FirebaseFirestore
) : ProjectRemoteDataSource {

    private val projectCollection = firestore.collection("projects")

    override suspend fun createProject(project: ProjectDto): Boolean {
        return try {
            val docRef = projectCollection.document()
            val projectWithId = project.copy(id = docRef.id)
            docRef.set(projectWithId).await()
            true
        } catch (e: Exception) {
            false
        }
    }

    override suspend fun getProjectById(id: String): ProjectDto? {
        return try {
```

```
        val document = projectCollection.document(id).get().await()
        document.toObject(ProjectDto::class.java)
    } catch (e: Exception) {
        null
    }
}

override suspend fun getAllProjects(): List<ProjectDto> {
    return try {
        val snapshot = projectCollection.get().await()
        snapshot.documents.mapNotNull { it.toObject(ProjectDto::class.java) }
    } catch (e: Exception) {
        emptyList()
    }
}

override suspend fun getProjectsByUser(userId: String): List<ProjectDto> {
    return try {
        val snapshot = projectCollection.whereEqualTo("ownerId", userId).get().await()
        snapshot.documents.mapNotNull { it.toObject(ProjectDto::class.java) }
    } catch (e: Exception) {
        emptyList()
    }
}

override suspend fun updateProjectImage(projectId: String, imageUrl: String): Boolean {
    return try {
```

```
            projectCollection.document(projectId)
                .update("imageUrl", imageUrl)
                .await()
            true
        } catch (e: Exception) {
            false
        }
    }

    override suspend fun updateProjectAmount(projectId: String, amount: Double): Boolean
{
        return try {
            projectCollection.document(projectId)
                .update("currentAmount", amount)
                .await()
            true
        } catch (e: Exception) {
            false
        }
    }
}
```

StorageService

```
package com.hugoguerrero.tecno.data.remote.firebase

import com.google.firebase.storage.FirebaseStorage
```

```
import jakarta.inject.Inject
import kotlinx.coroutines.tasks.await

class StorageService @Inject constructor(
    private val storage: FirebaseStorage
) {
    suspend fun uploadImage(path: String, data: ByteArray): String {
        val ref = storage.reference.child(path)
        ref.putBytes(data).await()
        return ref.downloadUrl.await().toString()
    }
}
```

UserRemoteDataSourceImpl

```
package com.hugoguerrero.tecno.data.remote.firebase

import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import com.hugoguerrero.tecno.data.model.AuthResultDto
import com.hugoguerrero.tecno.data.model.RegisterRequestDto
import com.hugoguerrero.tecno.data.model.UserDto
import com.hugoguerrero.tecno.data.remote.UserRemoteDataSource
import kotlinx.coroutines.tasks.await

class UserRemoteDataSourceImpl(
    private val auth: FirebaseAuth,
```

```
    private val firestore: FirebaseFirestore
) : UserRemoteDataSource {

    override suspend fun login(email: String, password: String): AuthResultDto? {
        return try {
            val result = auth.signInWithEmailAndPassword(email, password).await()
            val user = result.user

            AuthResultDto(
                userId = user?.uid,
                email = user?.email,
                isNewUser = false
            )
        } catch (e: Exception) {
            null
        }
    }

    override suspend fun register(request: RegisterRequestDto): AuthResultDto? {
        return try {
            val result = auth.createUserWithEmailAndPassword(request.email,
request.password).await()
            val user = result.user ?: return null

            val userDto = UserDto(
                id = user.uid,
                email = request.email,
```

```
            name = request.name,
            photoUrl = null,
            createdAt = System.currentTimeMillis(),
            updatedAt = System.currentTimeMillis()
        )

        firestore.collection("users").document(user.uid).set(userDto).await()

        AuthResultDto(
            userId = user.uid,
            email = request.email,
            isNewUser = true
        )
    } catch (e: Exception) {
        null
    }
}

override suspend fun getUser(userId: String): UserDto? {
    return try {
        firestore.collection("users")
            .document(userId)
            .get()
            .await()
            .toObject(UserDto::class.java)
    } catch (e: Exception) {
        null
```

```
        }
    }

    override suspend fun updateUser(user: UserDto): Boolean {
        return try {
            firestore.collection("users")
                .document(user.id ?: return false)
                .set(user.copy(updatedAt = System.currentTimeMillis()))
                .await()
            true
        } catch (e: Exception) {
            false
        }
    }
}
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/data/remote/

```kotlin

```

AuthRemoteDataSource

```
package com.hugoguerrero.tecno.data.remote

import com.hugoguerrero.tecno.data.model.AuthResultDto
import com.hugoguerrero.tecno.data.model.LoginRequestDto
```

```
import com.hugoguerrero.tecno.data.model.RegisterRequestDto

interface AuthRemoteDataSource {
    suspend fun login(request: LoginRequestDto): AuthResultDto
    suspend fun register(request: RegisterRequestDto): AuthResultDto
    suspend fun loginWithGoogle(idToken: String): AuthResultDto
    suspend fun logout()
    fun getCurrentUserId(): String?
}
```

DonationRemoteDataSource

```
package com.hugoguerrero.tecno.data.remote

import com.hugoguerrero.tecno.data.model.DonationDto

interface DonationRemoteDataSource {
    suspend fun addDonation(donation: DonationDto): Boolean
    suspend fun getDonationsByProject(projectId: String): List<DonationDto>
    suspend fun getDonationsByUser(userId: String): List<DonationDto> // ✅ Agregada
}
```

NotificationRemoteDataSource

```
package com.hugoguerrero.tecno.data.remote

import com.hugoguerrero.tecno.data.model.NotificationDto
```

```
interface NotificationRemoteDataSource {
    suspend fun saveUserNotificationToken(userId: String, token: String)
    suspend fun sendNotification(notification: NotificationDto): Boolean
    suspend fun getNotifications(userId: String): List<NotificationDto>
    suspend fun markAsRead(notificationId: String): Boolean
}
```

ProjectRemoteDataSource

```
package com.hugoguerrero.tecno.data.remote

import com.hugoguerrero.tecno.data.model.ProjectDto

interface ProjectRemoteDataSource {
    suspend fun createProject(project: ProjectDto): Boolean
    suspend fun getProjectById(id: String): ProjectDto?
    suspend fun getAllProjects(): List<ProjectDto>
    suspend fun getProjectsByUser(userId: String): List<ProjectDto>
    suspend fun updateProjectImage(projectId: String, imageUrl: String): Boolean
    suspend fun updateProjectAmount(projectId: String, amount: Double): Boolean
}
```

UserRemoteDataSource

```
package com.hugoguerrero.tecno.data.remote
```

```
import com.hugoguerrero.tecno.data.model.AuthResultDto
import com.hugoguerrero.tecno.data.model.RegisterRequestDto
import com.hugoguerrero.tecno.data.model.UserDto

interface UserRemoteDataSource {
    suspend fun login(email: String, password: String): AuthResultDto?
    suspend fun register(request: RegisterRequestDto): AuthResultDto?
    suspend fun getUser(userId: String): UserDto?
    suspend fun updateUser(user: UserDto): Boolean
}
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/data/repository/

```kotlin
```

AuthRepository

```
package com.hugoguerrero.tecno.data.repository

import android.app.Application
import com.google.android.gms.auth.api.signin.GoogleSignIn
import com.google.android.gms.auth.api.signin.GoogleSignInOptions
import com.google.firebase.auth.AuthCredential
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser
import com.hugoguerrero.tecno.R
import kotlinx.coroutines.flow.Flow
```

```
import kotlinx.coroutines.flow.catch
import kotlinx.coroutines.flow.flow
import kotlinx.coroutines.tasks.await

class AuthRepository(
    private val firebaseAuth: FirebaseAuth,
    private val application: Application
) {

    // Opciones para obtener el GoogleSignInClient
    private val gso: GoogleSignInOptions
        get() = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken(application.getString(R.string.web_client_id))
            .requestEmail()
            .build()

    private val googleSignInClient by lazy { GoogleSignIn.getClient(application, gso) }

    // ... (signInWithGoogle y signInWithEmail no cambian)

    fun getCurrentUser(): FirebaseUser? {
        return firebaseAuth.currentUser
    }

    // Función de signOut actualizada
    suspend fun signOut() {
        try {
```

```
            // Cierra la sesión de Firebase
            firebaseAuth.signOut()
            // Cierra la sesión de Google
            googleSignInClient.signOut().await()
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}
```

AuthRepositoryImpl

```
package com.hugoguerrero.tecno.data.repository

import android.app.Application
import com.google.android.gms.auth.api.signin.GoogleSignIn
import com.google.android.gms.auth.api.signin.GoogleSignInOptions
import com.google.firebase.auth.AuthCredential
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser
import com.hugoguerrero.tecno.R
import com.hugoguerrero.tecno.domain.repository.AuthRepository
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow
import kotlinx.coroutines.tasks.await
import javax.inject.Inject
```

```
class AuthRepositoryImpl @Inject constructor(
    private val firebaseAuth: FirebaseAuth,
    private val application: Application
) : AuthRepository {

    private val gso: GoogleSignInOptions
        get() = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken(application.getString(R.string.web_client_id))
            .requestEmail()
            .build()

    private val googleSignInClient by lazy { GoogleSignIn.getClient(application, gso) }

    override fun signInWithEmail(email: String, password: String):
Flow<Result<FirebaseUser>> = flow {
        try {
            val userCredential = firebaseAuth.signInWithEmailAndPassword(email,
password).await()
            val user = userCredential.user!!
            emit(Result.success(user))
        } catch (e: Exception) {
            emit(Result.failure(e))
        }
    }

    override fun signInWithGoogle(credential: AuthCredential):
Flow<Result<FirebaseUser>> = flow {
        try {
```

```
            val userCredential = firebaseAuth.signInWithCredential(credential).await()
            val user = userCredential.user!!
            emit(Result.success(user))
        } catch (e: Exception) {
            emit(Result.failure(e))
        }
    }

    override fun getCurrentUser(): FirebaseUser? {
        return firebaseAuth.currentUser
    }

    override suspend fun signOut() {
        try {
            firebaseAuth.signOut()
            googleSignInClient.signOut().await()
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
}
```

ComplaintRepository

```
package com.hugoguerrero.tecno.data.repository

import com.google.firebase.firestore.FirebaseFirestore
```

```
import com.hugoguerrero.tecno.data.model.Complaint
import kotlinx.coroutines.tasks.await
import javax.inject.Inject

class ComplaintRepository @Inject constructor(
    private val firestore: FirebaseFirestore
) {

    private val complaintsCollection = firestore.collection("complaints")

    suspend fun createComplaint(complaint: Complaint): Result<Unit> {
        return try {
            val complaintRef = complaintsCollection.document()
            complaintsCollection.document(complaintRef.id).set(complaint.copy(id =
complaintRef.id)).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}
```

DonationRepository

```
package com.hugoguerrero.tecno.data.repository

import com.google.firebase.firestore.FieldValue
```

```
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.Query
import com.google.firebase.firestore.toObject
import com.hugoguerrero.tecno.data.model.Donation
import com.hugoguerrero.tecno.data.model.DonationStatus
import kotlinx.coroutines.channels.awaitClose
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.callbackFlow
import kotlinx.coroutines.tasks.await
import javax.inject.Inject

class DonationRepository @Inject constructor(
    private val firestore: FirebaseFirestore
) {

    private val donationsCollection = firestore.collection("donations")
    private val projectsCollection = firestore.collection("projects")



    suspend fun updateDonationStatus(donationId: String, projectId: String, amount:
Double, newStatus: DonationStatus): Result<Unit> {
        return try {
            val donationRef = donationsCollection.document(donationId)
            val projectRef = projectsCollection.document(projectId)

            firestore.runTransaction {
                transaction ->
```

```
            val snapshot = transaction.get(donationRef)
            val currentStatus = snapshot.get("status", DonationStatus::class.java)

            if (currentStatus == DonationStatus.PENDING) {
                transaction.update(donationRef, "status", newStatus)
                if (newStatus == DonationStatus.COMPLETED) {
                    transaction.update(projectRef, "currentFunding",
FieldValue.increment(amount))
                }
            }
            null
        }.await()

        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }
}

fun getDonationsForProject(projectId: String): Flow<List<Donation>> = callbackFlow {
    val listenerRegistration = donationsCollection
        .whereEqualTo("projectId", projectId)
        .orderBy("createdAt", Query.Direction.DESCENDING)
        .addSnapshotListener { snapshot, error ->
            if (error != null) {
                close(error)
                return@addSnapshotListener
```

```
            }
            if (snapshot != null) {
                val donations = snapshot.documents.mapNotNull { doc ->
                    doc.toObject<Donation>()?.copy(id = doc.id)
                }
                trySend(donations)
            }
        }
        awaitClose { listenerRegistration.remove() }
    }

    suspend fun createDonation(donation: Donation): Result<String> {
        return try {
            val newDonation = donation.copy(id = donationsCollection.document().id)
            donationsCollection.document(newDonation.id).set(newDonation).await()
            Result.success(newDonation.id)
        } catch (e: Exception) {
            Result.failure(e)
        }
    }
}
```

ProjectRepository

```
package com.hugoguerrero.tecno.data.repository

import android.net.Uri
```

```
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.Query
import com.google.firebase.storage.FirebaseStorage
import com.hugoguerrero.tecno.data.model.Project
import kotlinx.coroutines.channels.awaitClose
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.callbackFlow
import kotlinx.coroutines.tasks.await
import java.util.UUID

class ProjectRepository(
    private val firestore: FirebaseFirestore,
    private val storage: FirebaseStorage
) {

    private val projectsCollection = firestore.collection("projects")

    fun getProjects(): Flow<List<Project>> = callbackFlow {
        val listener = projectsCollection.orderBy("createdAt",
Query.Direction.DESCENDING)
            .addSnapshotListener { snapshot, e ->
                if (e != null) {
                    close(e)
                    return@addSnapshotListener
                }
                if (snapshot != null) {
                    val projects = snapshot.toObjects(Project::class.java)
```

```
            trySend(projects)
        }
    }
    awaitClose { listener.remove() }
}

suspend fun getProjectById(projectId: String): Project? {
    return try {
        projectsCollection.document(projectId).get().await().toObject(Project::class.java)
    } catch (e: Exception) {
        null
    }
}

fun observeProjectById(projectId: String): Flow<Project?> = callbackFlow {
    val documentRef = projectsCollection.document(projectId)
    val listener = documentRef.addSnapshotListener { snapshot, error ->
        if (error != null) {
            close(error)
            return@addSnapshotListener
        }
        if (snapshot != null && snapshot.exists()) {
            trySend(snapshot.toObject(Project::class.java))
        } else {
            trySend(null)
        }
    }
```

```
    awaitClose { listener.remove() }
}

fun getProjectsByOwner(ownerUid: String): Flow<List<Project>> = callbackFlow {
    val listener = projectsCollection.whereEqualTo("ownerUid", ownerUid)
        .orderBy("createdAt", Query.Direction.DESCENDING)
        .addSnapshotListener { snapshot, e ->
            if (e != null) {
                close(e)
                return@addSnapshotListener
            }
            if (snapshot != null) {
                val projects = snapshot.toObjects(Project::class.java)
                trySend(projects)
            }
        }
    awaitClose { listener.remove() }
}

suspend fun createProject(
    title: String,
    description: String,
    category: String,
    institution: String,
    fundingGoal: Double,
    ownerUid: String,
    imageUri: Uri,
```

```
    documentUris: List<Uri>
): Result<Unit> {
    return try {
        val projectId = projectsCollection.document().id
        val imageUrl = uploadProjectImage(imageUri, projectId)
        val documentUrls = uploadProjectDocuments(documentUris, projectId)

        val newProject = Project(
            id = projectId,
            title = title,
            description = description,
            category = category,
            institution = institution,
            fundingGoal = fundingGoal,
            ownerUid = ownerUid,
            imageUrl = imageUrl,
            documentUrls = documentUrls
        )
        projectsCollection.document(projectId).set(newProject).await()
        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }
}

suspend fun updateProject(
    projectId: String,
```

```
        title: String,
        description: String,
        category: String,
        institution: String,
        fundingGoal: Double,
        newImageUri: Uri?,
        newDocumentUris: List<Uri>?
    ): Result<Unit> {
        return try {
            val updates = mutableMapOf<String, Any>(
                "title" to title,
                "description" to description,
                "category" to category,
                "institution" to institution,
                "fundingGoal" to fundingGoal
            )

            newImageUri?.let {
                val newImageUrl = uploadProjectImage(it, projectId)
                updates["imageUrl"] = newImageUrl
            }

            newDocumentUris?.let {
                if (it.isNotEmpty()) {
                    val newDocumentUrls = uploadProjectDocuments(it, projectId)
                    updates["documentUrls"] = newDocumentUrls // Idealmente, esto debería
añadir a los existentes
```

```
            }
        }

        projectsCollection.document(projectId).update(updates).await()
        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }
}


suspend fun deleteProject(projectId: String): Result<Unit> {
    return try {
        projectsCollection.document(projectId).delete().await()
        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }
}

private suspend fun uploadProjectImage(imageUri: Uri, projectId: String): String {
    val storageRef =
storage.reference.child("project_images/$projectId/${UUID.randomUUID()}")
    val uploadTask = storageRef.putFile(imageUri).await()
    return uploadTask.storage.downloadUrl.await().toString()
}
```

```
    private suspend fun uploadProjectDocuments(documentUris: List<Uri>, projectId: String): List<String> {
        val downloadUrls = mutableListOf<String>()
        for (uri in documentUris) {
            val storageRef = storage.reference.child("project_documents/$projectId/${UUID.randomUUID()}")
            val uploadTask = storageRef.putFile(uri).await()
            val downloadUrl = uploadTask.storage.downloadUrl.await().toString()
            downloadUrls.add(downloadUrl)
        }
        return downloadUrls
    }
}
```

UserRepository

```
package com.hugoguerrero.tecno.data.repository

import android.net.Uri
import com.google.firebase.auth.FirebaseUser
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.storage.FirebaseStorage
import com.hugoguerrero.tecno.data.model.UserProfile
import com.hugoguerrero.tecno.domain.model.UserType
import kotlinx.coroutines.tasks.await
import java.util.UUID

class UserRepository(
```

```
    private val firestore: FirebaseFirestore,
    private val storage: FirebaseStorage
) {

    private val usersCollection = firestore.collection("users")

    suspend fun createUserProfile(user: FirebaseUser, name: String, userType: UserType) {
        val userProfile = UserProfile(
            uid = user.uid,
            displayName = name,
            email = user.email ?: "",
            photoUrl = user.photoUrl?.toString() ?: "",
            userType = userType
        )
        usersCollection.document(user.uid).set(userProfile).await()
    }

    suspend fun getUserProfile(uid: String): UserProfile? {
        return try {
            usersCollection.document(uid).get().await().toObject(UserProfile::class.java)
        } catch (e: Exception) {
            null
        }
    }

    suspend fun userProfileExists(uid: String): Boolean {
        return try {
```

```
            usersCollection.document(uid).get().await().exists()
        } catch (e: Exception) {
            false
        }
    }

    suspend fun updateFcmToken(userId: String, token: String) {
        usersCollection.document(userId).update("fcmToken", token).await()
    }

    suspend fun updateUserProfile(uid: String, displayName: String, bio: String, photoUri:
Uri?, paymentDetails: String): Result<Unit> {
        return try {
            val updates = mutableMapOf<String, Any>()
            updates["displayName"] = displayName
            updates["bio"] = bio
            updates["paymentDetails"] = paymentDetails

            photoUri?.let {
                val imageUrl = uploadProfilePhoto(uid, it)
                updates["photoUrl"] = imageUrl
            }

            usersCollection.document(uid).update(updates).await()
            Result.success(Unit)
        } catch (e: Exception) {
            Result.failure(e)
```

```
        }
    }

    private suspend fun uploadProfilePhoto(uid: String, photoUri: Uri): String {
        val storageRef =
storage.reference.child("profile_photos/$uid/${UUID.randomUUID()}")
        val uploadTask = storageRef.putFile(photoUri).await()
        return uploadTask.storage.downloadUrl.await().toString()
    }
}
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/di/

```kotlin
```

AppModule

```
package com.hugoguerrero.tecno.di

import android.app.Application
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.storage.FirebaseStorage
import com.hugoguerrero.tecno.data.repository.AuthRepository
import com.hugoguerrero.tecno.data.repository.DonationRepository
import com.hugoguerrero.tecno.data.repository.ProjectRepository
import com.hugoguerrero.tecno.data.repository.UserRepository
```

```
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
object AppModule {

    @Provides
    @Singleton
    fun provideFirebaseAuth(): FirebaseAuth = FirebaseAuth.getInstance()

    @Provides
    @Singleton
    fun provideFirebaseFirestore(): FirebaseFirestore = FirebaseFirestore.getInstance()

    @Provides
    @Singleton
    fun provideFirebaseStorage(): FirebaseStorage = FirebaseStorage.getInstance()

    @Provides
    @Singleton
    fun provideAuthRepository(firebaseAuth: FirebaseAuth, application: Application):
AuthRepository = AuthRepository(firebaseAuth, application)
```

```
    @Provides
    @Singleton
    fun provideUserRepository(firestore: FirebaseFirestore, storage: FirebaseStorage): UserRepository = UserRepository(firestore, storage) // Corregido aquí

    @Provides
    @Singleton
    fun provideProjectRepository(firestore: FirebaseFirestore, storage: FirebaseStorage): ProjectRepository = ProjectRepository(firestore, storage)

    @Provides
    @Singleton
    fun provideDonationRepository(firestore: FirebaseFirestore): DonationRepository = DonationRepository(firestore)

}
```

RepositoryModule

```
package com.hugoguerrero.tecno.di // Asegúrate de que el paquete sea el correcto

import com.hugoguerrero.tecno.data.repository.AuthRepositoryImpl // ¡Importa tu implementación!
import com.hugoguerrero.tecno.domain.repository.AuthRepository // ¡Importa tu interfaz!
import dagger.Binds
import dagger.Module
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
```

```
import javax.inject.Singleton

@Module
@InstallIn(SingletonComponent::class)
abstract class RepositoryModule {

    @Binds
    @Singleton
    abstract fun bindAuthRepository(
        impl: AuthRepositoryImpl // Le dices que la implementación es AuthRepositoryImpl
    ): AuthRepository          // Y que debe usarla cuando alguien pida un AuthRepository

}
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/domain/model/

```kotlin
```

Category

```
    package com.hugoguerrero.tecno.domain.model

data class Category(
    val id: String = "",
    val nombre: String = ""
)
```

Contribution

```
package com.hugoguerrero.tecno.domain.model

data class Contribution(
    val id: String = "",
    val proyectoId: String = "",
    val usuarioId: String = "",
    val cantidad: Double = 0.0,
    val fecha: Long = 0L,
    val metodo: String = "paypal" // paypal | google | tarjeta
)
```

Document

```
package com.hugoguerrero.tecno.domain.model

data class Document(
    val id: String = "",
    val proyectoId: String = "",
    val url: String = "",
    val nombre: String = "",
    val tipo: String = "", // pdf, img, zip
)
```

Project

```
```

```
package com.hugoguerrero.tecno.domain.model

data class Project(
    val id: String = "",
    val titulo: String = "",
    val descripcion: String = "",
    val categoriaId: String = "",
    val imagenPortada: String? = null,
    val fechaCreado: Long = 0L,
    val creadoPor: String = "",
    val metaFinanciamiento: Double = 0.0,
    val recaudado: Double = 0.0,
    val fechaObjetivo: Long = 0L,
    val visible: Boolean = true,
    val etiquetas: List<String> = emptyList(),
)
```

User

```
package com.hugoguerrero.tecno.domain.model

data class User(
    val id: String = "",
    val nombre: String = "",
    val correo: String = "",
    val fotoUrl: String? = null,
    val fechaRegistrado: Long = 0L,
```

```
    val rol: String = "usuario", // usuario | manager | admin
    val creditosDisponibles: Double = 0.0,
    val paymentDetails: String = "" // Nuevo campo para los detalles de pago
)
```

UserType

```
package com.hugoguerrero.tecno.domain.model

enum class UserType {
    STUDENT,
    MANAGER,
    SPONSOR
}
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/domain/repository/

```kotlin
```

AuthRepository

```
package com.hugoguerrero.tecno.domain.repository

import com.google.firebase.auth.AuthCredential
import com.google.firebase.auth.FirebaseUser
import kotlinx.coroutines.flow.Flow
```

```
interface AuthRepository {
    fun signInWithEmail(email: String, password: String): Flow<Result<FirebaseUser>>
    fun signInWithGoogle(credential: AuthCredential): Flow<Result<FirebaseUser>>
    fun getCurrentUser(): FirebaseUser?
    suspend fun signOut()
}
```

CategoryRepository

```
package com.hugoguerrero.tecno.domain.repository

import com.hugoguerrero.tecno.domain.model.Category

interface CategoryRepository {
    suspend fun getCategories(): List<Category>
}
```

ContributionRepository

```
package com.hugoguerrero.tecno.domain.repository

import com.hugoguerrero.tecno.domain.model.Contribution

interface ContributionRepository {
    suspend fun getContributionsByProject(projectId: Int): List<Contribution>
    suspend fun addContribution(contribution: Contribution): Boolean
}
```

```
DocumentRepository
```
```
package com.hugoguerrero.tecno.domain.repository

import com.hugoguerrero.tecno.domain.model.Document

interface DocumentRepository {
    suspend fun uploadDocument(projectId: Int, document: Document): Boolean
    suspend fun getDocuments(projectId: Int): List<Document>
}
```
```
DonationRepository
```
```
package com.hugoguerrero.tecno.domain.repository

import com.hugoguerrero.tecno.data.model.Donation
import com.hugoguerrero.tecno.data.model.DonationStatus
import kotlinx.coroutines.flow.Flow

interface DonationRepository {

    fun getDonationsForProject(projectId: String): Flow<List<Donation>>

    suspend fun finalizeDonation(donationId: String): Result<Unit>
```

```
    suspend fun updateDonationStatus(donationId: String, projectId: String, amount: Double, newStatus: DonationStatus): Result<Unit>
}
```

ProjectRepository

```
package com.hugoguerrero.tecno.domain.repository

import com.hugoguerrero.tecno.domain.model.Project
import com.hugoguerrero.tecno.domain.model.Document

interface ProjectRepository {
    suspend fun getAllProjects(): List<Project>
    suspend fun getProjectById(id: Int): Project?
    suspend fun getProjectsByUser(userId: Int): List<Project>

    suspend fun createProject(
        project: Project,
        documents: List<Document>
    ): Boolean

    suspend fun updateProject(project: Project): Boolean
}
```

UserRepository

```
package com.hugoguerrero.tecno.domain.repository
```

```
import com.hugoguerrero.tecno.domain.model.User

interface UserRepository {
    suspend fun login(email: String, password: String): User?
    suspend fun register(user: User): Boolean
    suspend fun logout(): Boolean
    suspend fun getUserById(id: Int): User?
    suspend fun updateUser(user: User): Boolean
}
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/domain/use_case/

```kotlin
```

CheckUserProfileExistsUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.repository.UserRepository
import javax.inject.Inject

class CheckUserProfileExistsUseCase @Inject constructor(
    private val userRepository: UserRepository
) {
    suspend operator fun invoke(uid: String): Boolean {
        return userRepository.userProfileExists(uid)
```

```
    }
}
```

CreateComplaintUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.Complaint
import com.hugoguerrero.tecno.data.repository.ComplaintRepository
import javax.inject.Inject

class CreateComplaintUseCase @Inject constructor(
    private val complaintRepository: ComplaintRepository
) {
    suspend operator fun invoke(complaint: Complaint): Result<Unit> {
        return complaintRepository.createComplaint(complaint)
    }
}
```

CreateDonationUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.Donation
import com.hugoguerrero.tecno.data.model.DonationStatus
import com.hugoguerrero.tecno.data.repository.DonationRepository
import javax.inject.Inject
```

```
class CreateDonationUseCase @Inject constructor(
    private val donationRepository: DonationRepository
) {
    suspend operator fun invoke(
        projectId: String,
        donorUid: String,
        amount: Double,
        message: String,
        status: DonationStatus,
        proofImageUrl: String
    ): Result<String> {
        val donation = Donation(
            projectId = projectId,
            donorUid = donorUid,
            amount = amount,
            message = message,
            status = status,
            proofImageUrl = proofImageUrl
        )
        return donationRepository.createDonation(donation)
    }
}
```

CreateProjectUseCase

```
package com.hugoguerrero.tecno.domain.use_case
```

```
import android.net.Uri
import com.hugoguerrero.tecno.data.repository.ProjectRepository
import javax.inject.Inject

class CreateProjectUseCase @Inject constructor(
    private val projectRepository: ProjectRepository
) {
    suspend operator fun invoke(
        title: String,
        description: String,
        category: String,
        institution: String,
        fundingGoal: Double,
        ownerUid: String,
        imageUri: Uri,
        documentUris: List<Uri> // Parámetro añadido
    ): Result<Unit> {
        return projectRepository.createProject(
            title = title,
            description = description,
            category = category,
            institution = institution,
            fundingGoal = fundingGoal,
            ownerUid = ownerUid,
            imageUri = imageUri,
            documentUris = documentUris // Pasamos los documentos al repositorio
```

```
        )
    }
}
```

CreateUserProfileUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.google.firebase.auth.FirebaseUser
import com.hugoguerrero.tecno.data.repository.UserRepository
import com.hugoguerrero.tecno.domain.model.UserType
import javax.inject.Inject

class CreateUserProfileUseCase @Inject constructor(
    private val userRepository: UserRepository
) {
    suspend operator fun invoke(user: FirebaseUser, name: String, userType: UserType) {
        userRepository.createUserProfile(user, name, userType)
    }
}
```

CreateUserProfileUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.repository.ProjectRepository
import javax.inject.Inject
```

```
class DeleteProjectUseCase @Inject constructor(
    private val projectRepository: ProjectRepository
) {
    suspend operator fun invoke(projectId: String): Result<Unit> {
        return projectRepository.deleteProject(projectId)
    }
}
```

GetCurrentUserUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.repository.AuthRepository
import javax.inject.Inject

class GetCurrentUserUseCase @Inject constructor(
    private val authRepository: AuthRepository
) {
    operator fun invoke() = authRepository.getCurrentUser()
}
```

GetDonationsForProjectUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.Donation
```

```
import com.hugoguerrero.tecno.data.repository.DonationRepository
import kotlinx.coroutines.flow.Flow
import javax.inject.Inject

class GetDonationsForProjectUseCase @Inject constructor(
    private val repository: DonationRepository
) {
    operator fun invoke(projectId: String): Flow<List<Donation>> {
        return repository.getDonationsForProject(projectId)
    }
}
```

GetDonationsUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.Donation
import com.hugoguerrero.tecno.data.repository.DonationRepository
import kotlinx.coroutines.flow.Flow
import javax.inject.Inject

class GetDonationsUseCase @Inject constructor(
    private val donationRepository: DonationRepository
) {
    operator fun invoke(projectId: String): Flow<List<Donation>> {
        return donationRepository.getDonationsForProject(projectId)
    }
```

}
```
GetProjectByIdUseCase
```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.data.repository.ProjectRepository
import javax.inject.Inject

class GetProjectByIdUseCase @Inject constructor(
    private val projectRepository: ProjectRepository
) {
    suspend operator fun invoke(projectId: String): Project? {
        return projectRepository.getProjectById(projectId)
    }
}
```
GetProjectsByOwnerUseCase
```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.data.repository.ProjectRepository
import kotlinx.coroutines.flow.Flow
import javax.inject.Inject

```
class GetProjectsByOwnerUseCase @Inject constructor(
    private val projectRepository: ProjectRepository
) {
    operator fun invoke(ownerUid: String): Flow<List<Project>> {
        return projectRepository.getProjectsByOwner(ownerUid)
    }
}
```

GetProjectsUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.data.repository.ProjectRepository
import kotlinx.coroutines.flow.Flow
import javax.inject.Inject

class GetProjectsUseCase @Inject constructor(
    private val projectRepository: ProjectRepository
) {
    operator fun invoke(): Flow<List<Project>> {
        return projectRepository.getProjects()
    }
}
```

GetProjectUseCase

```
```

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.repository.ProjectRepository
import javax.inject.Inject

class GetProjectUseCase @Inject constructor(
    private val repository: ProjectRepository
) {
    suspend operator fun invoke(projectId: String) = repository.getProjectById(projectId)
}
```

GetUserProfileUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.UserProfile
import com.hugoguerrero.tecno.data.repository.UserRepository
import javax.inject.Inject

class GetUserProfileUseCase @Inject constructor(
    private val userRepository: UserRepository
) {
    suspend operator fun invoke(uid: String): UserProfile? {
        return userRepository.getUserProfile(uid)
    }
}
```

GetUserStatsUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.UserStats
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow
import javax.inject.Inject

class GetUserStatsUseCase @Inject constructor() {
    // Esta es una simulación. En una app real, estos datos vendrían de tu backend.
    operator fun invoke(userId: String): Flow<Result<UserStats>> = flow {
        try {
            // Simulación de estadísticas
            val stats = UserStats(donationsCount = 15, successRate = 88)
            emit(Result.success(stats))
        } catch (e: Exception) {
            emit(Result.failure(e))
        }
    }
}
```

GetUserUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.repository.UserRepository
```

```
import javax.inject.Inject

class GetUserUseCase @Inject constructor(
    private val repository: UserRepository
) {
    suspend operator fun invoke(userId: String) = repository.getUserProfile(userId)
}
```

ObserveProjectByIdUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.data.repository.ProjectRepository
import kotlinx.coroutines.flow.Flow
import javax.inject.Inject

class ObserveProjectByIdUseCase @Inject constructor(
    private val projectRepository: ProjectRepository
) {
    operator fun invoke(projectId: String): Flow<Project?> {
        return projectRepository.observeProjectById(projectId)
    }
}
```

RegisterUserUseCase

```
```

```
package com.hugoguerrero.tecno.domain.use_case

import com.google.firebase.auth.FirebaseAuth
import com.hugoguerrero.tecno.data.repository.UserRepository
import com.hugoguerrero.tecno.domain.model.UserType
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow
import kotlinx.coroutines.tasks.await
import javax.inject.Inject

class RegisterUserUseCase @Inject constructor(
    private val firebaseAuth: FirebaseAuth,
    private val userRepository: UserRepository
) {
    operator fun invoke(name: String, email: String, password: String, userType: UserType): Flow<Result<Unit>> = flow {
        try {
            // 1. Crear usuario en Firebase Auth
            val authResult = firebaseAuth.createUserWithEmailAndPassword(email, password).await()
            val firebaseUser = authResult.user ?: throw Exception("No se pudo crear el usuario.")

            // 2. Crear perfil de usuario en Firestore
            userRepository.createUserProfile(firebaseUser, name, userType)

            emit(Result.success(Unit))
        } catch (e: Exception) {
```

```
            emit(Result.failure(e))
        }
    }
}
```

SignInWithEmailUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.google.firebase.auth.FirebaseUser
import com.hugoguerrero.tecno.domain.repository.AuthRepository // <-- CORREGIDO
import kotlinx.coroutines.flow.Flow
import javax.inject.Inject

class SignInWithEmailUseCase @Inject constructor(
    private val authRepository: AuthRepository
) {
    operator fun invoke(email: String, password: String): Flow<Result<FirebaseUser>> {
        return authRepository.signInWithEmail(email, password)
    }
}
```

SignInWithGoogleUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.google.firebase.auth.AuthCredential
```

```
import com.google.firebase.auth.FirebaseUser
import com.hugoguerrero.tecno.domain.repository.AuthRepository // <-- ¡CORREGIDO!
import kotlinx.coroutines.flow.Flow
import javax.inject.Inject

class SignInWithGoogleUseCase @Inject constructor(
    private val authRepository: AuthRepository
) {
    operator fun invoke(credential: AuthCredential): Flow<Result<FirebaseUser>> {
        return authRepository.signInWithGoogle(credential)
    }
}
```

SignOutUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.repository.AuthRepository
import javax.inject.Inject

class SignOutUseCase @Inject constructor(
    private val authRepository: AuthRepository
) {
    suspend operator fun invoke() {
        authRepository.signOut()
    }
}
```

```

UpdateDonationStatusUseCase

```

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.model.DonationStatus
import com.hugoguerrero.tecno.data.repository.DonationRepository
import javax.inject.Inject

class UpdateDonationStatusUseCase @Inject constructor(
    private val repository: DonationRepository
) {
    suspend operator fun invoke(donationId: String, projectId: String, amount: Double, newStatus: DonationStatus): Result<Unit> {
        return repository.updateDonationStatus(donationId, projectId, amount, newStatus)
    }
}
```

UpdateFcmTokenUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import com.hugoguerrero.tecno.data.repository.UserRepository
import javax.inject.Inject

class UpdateFcmTokenUseCase @Inject constructor(
    private val userRepository: UserRepository
```

```
) {
    suspend operator fun invoke(userId: String, token: String) {
        try {
            userRepository.updateFcmToken(userId, token)
        } catch (e: Exception) {
            // Manejar el error, por ejemplo, registrarlo
            e.printStackTrace()
        }
    }
}
```

UpdateProjectUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import android.net.Uri
import com.hugoguerrero.tecno.data.repository.ProjectRepository
import javax.inject.Inject

class UpdateProjectUseCase @Inject constructor(
    private val repository: ProjectRepository
) {
    suspend operator fun invoke(
        projectId: String,
        title: String,
        description: String,
        category: String,
```

```
        institution: String,
        fundingGoal: Double,
        newImageUri: Uri?,
        newDocumentUris: List<Uri>?
    ): Result<Unit> {
        return repository.updateProject(
            projectId = projectId,
            title = title,
            description = description,
            category = category,
            institution = institution,
            fundingGoal = fundingGoal,
            newImageUri = newImageUri,
            newDocumentUris = newDocumentUris
        )
    }
}
```

UpdateUserProfileUseCase

```
package com.hugoguerrero.tecno.domain.use_case

import android.net.Uri
import com.hugoguerrero.tecno.data.repository.UserRepository
import javax.inject.Inject

class UpdateUserProfileUseCase @Inject constructor(
```

```
    private val userRepository: UserRepository
) {
    suspend operator fun invoke(uid: String, displayName: String, bio: String, photoUri: Uri?, paymentDetails: String): Result<Unit> {
        return userRepository.updateUserProfile(uid, displayName, bio, photoUri, paymentDetails)
    }
}
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/domain/usecase/

```kotlin
```

Crea la carpeta category

GetCategoriesUseCase

```
package com.hugoguerrero.tecno.domain.usecase.category

import com.hugoguerrero.tecno.domain.model.Category
import com.hugoguerrero.tecno.domain.repository.CategoryRepository

class GetCategoriesUseCase(
    private val repository: CategoryRepository
) {
    suspend operator fun invoke(): List<Category> {
        return repository.getCategories()
    }
}
```

```
Crea la Carpeta contribution
GetCategoriesUseCase
```
```
package com.hugoguerrero.tecno.domain.usecase.contribution

import com.hugoguerrero.tecno.domain.model.Contribution
import com.hugoguerrero.tecno.domain.repository.ContributionRepository

class GetProjectContributionsUseCase(
    private val repository: ContributionRepository
) {
    suspend operator fun invoke(projectId: Int): List<Contribution> {
        return repository.getContributionsByProject(projectId)
    }
}
```
```
Crea la Carpeta document
GetDocumentsUseCase
```
```
package com.hugoguerrero.tecno.domain.usecase.document

import com.hugoguerrero.tecno.domain.model.Document
import com.hugoguerrero.tecno.domain.repository.DocumentRepository

class GetDocumentsUseCase(
    private val repository: DocumentRepository
```

```
) {
    suspend operator fun invoke(projectId: Int): List<Document> {
        return repository.getDocuments(projectId)
    }
}
```

GetDocumentsUseCase

```
package com.hugoguerrero.tecno.domain.usecase.document

import com.hugoguerrero.tecno.domain.model.Document
import com.hugoguerrero.tecno.domain.repository.DocumentRepository

class UploadDocumentUseCase(
    private val repository: DocumentRepository
) {
    suspend operator fun invoke(projectId: Int, document: Document): Boolean {
        return repository.uploadDocument(projectId, document)
    }
}
```

Crea la carpeta project

CreateProjectUseCase

```
package com.hugoguerrero.tecno.domain.usecase.project

import com.hugoguerrero.tecno.domain.model.Document
```

```
import com.hugoguerrero.tecno.domain.model.Project
import com.hugoguerrero.tecno.domain.repository.ProjectRepository

class CreateProjectUseCase(
    private val repository: ProjectRepository
) {
    suspend operator fun invoke(project: Project, documents: List<Document>): Boolean {
        return repository.createProject(project, documents)
    }
}
```

GetAllProjectsUseCase

```
package com.hugoguerrero.tecno.domain.usecase.project

import com.hugoguerrero.tecno.domain.model.Project
import com.hugoguerrero.tecno.domain.repository.ProjectRepository

class GetAllProjectsUseCase(
    private val repository: ProjectRepository
) {
    suspend operator fun invoke(): List<Project> {
        return repository.getAllProjects()
    }
}
```

GetProjectByIdUseCase

```
package com.hugoguerrero.tecno.domain.usecase.project

import com.hugoguerrero.tecno.domain.model.Project
import com.hugoguerrero.tecno.domain.repository.ProjectRepository

class GetProjectByIdUseCase(
    private val repository: ProjectRepository
) {
    suspend operator fun invoke(id: Int): Project? {
        return repository.getProjectById(id)
    }
}
```

GetUserProjectsUseCase

```
package com.hugoguerrero.tecno.domain.usecase.project

import com.hugoguerrero.tecno.domain.model.Project
import com.hugoguerrero.tecno.domain.repository.ProjectRepository

class GetUserProjectsUseCase(
    private val repository: ProjectRepository
) {
    suspend operator fun invoke(userId: Int): List<Project> {
        return repository.getProjectsByUser(userId)
    }
```

}
```
Crea la carpeta user
GetUserByIdUseCase
```

```
package com.hugoguerrero.tecno.domain.usecase.user

import com.hugoguerrero.tecno.domain.model.User
import com.hugoguerrero.tecno.domain.repository.UserRepository

class GetUserByIdUseCase(
    private val repository: UserRepository
) {
    suspend operator fun invoke(id: Int): User? {
        return repository.getUserById(id)
    }
}
```
LoginUserUseCase
```
package com.hugoguerrero.tecno.domain.usecase.user

import com.hugoguerrero.tecno.domain.model.User
import com.hugoguerrero.tecno.domain.repository.UserRepository

class LoginUserUseCase(
```

```
    private val repository: UserRepository
) {
    suspend operator fun invoke(email: String, password: String): User? {
        return repository.login(email, password)
    }
}
```

RegisterUserUseCase

```
package com.hugoguerrero.tecno.domain.usecase.user

import com.hugoguerrero.tecno.domain.model.User
import com.hugoguerrero.tecno.domain.repository.UserRepository

class RegisterUserUseCase(
    private val repository: UserRepository
) {
    suspend operator fun invoke(user: User): Boolean {
        return repository.register(user)
    }
}
```

UpdateUserUseCase

```
package com.hugoguerrero.tecno.domain.usecase.user

import com.hugoguerrero.tecno.domain.model.User
```

```
import com.hugoguerrero.tecno.domain.repository.UserRepository

class UpdateUserUseCase(
    private val repository: UserRepository
) {
    suspend operator fun invoke(user: User): Boolean {
        return repository.updateUser(user)
    }
}
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/ui/components/

```kotlin
```

BottomNavigationBar

```
package com.hugoguerrero.tecno.ui.components

import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Home
import androidx.compose.material.icons.filled.Person
import androidx.compose.material3.Icon
import androidx.compose.material3.NavigationBar
import androidx.compose.material3.NavigationBarItem
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
```

```
import androidx.compose.runtime.getValue
import androidx.navigation.NavController
import androidx.navigation.NavDestination.Companion.hierarchy
import androidx.navigation.NavGraph.Companion.findStartDestination
import androidx.navigation.compose.currentBackStackEntryAsState
import com.hugoguerrero.tecno.ui.navigation.Screens

@Composable
fun BottomNavigationBar(navController: NavController) {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentDestination = navBackStackEntry?.destination

    NavigationBar {
        // Pestaña de Inicio
        NavigationBarItem(
            icon = { Icon(Icons.Default.Home, contentDescription = "Inicio") },
            label = { Text("Inicio") },
            selected = currentDestination?.hierarchy?.any { it.route == Screens.Main.route } ==
true,
            onClick = {
                navController.navigate(Screens.Main.route) {
                    popUpTo(navController.graph.findStartDestination().id) {
                        saveState = true
                    }
                    launchSingleTop = true
                    restoreState = true
                }
```

```
            }
        )

        // Pestaña de Crear Proyecto
        NavigationBarItem(
            icon = { Icon(Icons.Default.Add, contentDescription = "Crear Proyecto") },
            label = { Text("Crear") },
            selected = currentDestination?.hierarchy?.any { it.route ==
Screens.UploadProject.route } == true,
            onClick = {
                navController.navigate(Screens.UploadProject.route) {
                    popUpTo(navController.graph.findStartDestination().id) {
                        saveState = true
                    }
                    launchSingleTop = true
                    restoreState = true
                }
            }
        )

        // Pestaña de Perfil
        NavigationBarItem(
            icon = { Icon(Icons.Default.Person, contentDescription = "Perfil") },
            label = { Text("Perfil") },
            selected = currentDestination?.hierarchy?.any {
                it.route == Screens.MyProfile.route || it.route == Screens.PublicProfile.route +
"/{userId}"
            } == true,
```

```
            onClick = {
                navController.navigate(Screens.MyProfile.route) {
                    popUpTo(navController.graph.findStartDestination().id) {
                        saveState = true
                    }
                    launchSingleTop = true
                    restoreState = true
                }
            }
        )
    }
}
```

ProjectCard

```
package com.hugoguerrero.tecno.ui.components

import android.util.Log
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Card
```

```
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.LinearProgressIndicator
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import coil.compose.AsyncImage
import coil.request.ImageRequest
import com.hugoguerrero.tecno.data.model.Project

@Composable
fun ProjectCard(project: Project, onClick: () -> Unit) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .clickable { onClick() },
        elevation = CardDefaults.cardElevation(defaultElevation = 4.dp)
    ) {
        Column {
            AsyncImage(
                model = ImageRequest.Builder(LocalContext.current)
```

```
            .data(project.imageUrl)
            .crossfade(true)
            .build(),
        contentDescription = "Imagen del proyecto ${project.title}",
        modifier = Modifier
            .fillMaxWidth()
            .height(150.dp)
            .clip(MaterialTheme.shapes.medium)
            .background(MaterialTheme.colorScheme.surfaceVariant),
        contentScale = ContentScale.Crop,
        onError = { error ->
            Log.e("ProjectCard", "Error al cargar la imagen: ", error.result.throwable)
        }
    )

    Column(modifier = Modifier.padding(16.dp)) {
        Text(
            text = project.title,
            style = MaterialTheme.typography.titleLarge,
            fontWeight = FontWeight.Bold,
            maxLines = 1,
            overflow = TextOverflow.Ellipsis
        )
        Text(
            text = project.description,
            style = MaterialTheme.typography.bodyMedium,
            maxLines = 3,
```

```
                overflow = TextOverflow.Ellipsis,
                modifier = Modifier.padding(vertical = 8.dp)
            )
            LinearProgressIndicator(
                progress = { project.progress / 100f },
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(vertical = 4.dp)
            )
            Row(
                modifier = Modifier.fillMaxWidth(),
                horizontalArrangement = Arrangement.SpaceBetween
            ) {
                Text("Progreso: ${project.progress.toInt()}%", style =
MaterialTheme.typography.labelSmall)
                Text("Meta: $${project.fundingGoal.toInt()}", style =
MaterialTheme.typography.labelSmall)
            }
        }
    }
}
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/ui/navigation/

```kotlin
```

BottomNavItem

```
package com.hugoguerrero.tecno.ui.navigation

import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.*
import androidx.compose.ui.graphics.vector.ImageVector

sealed class BottomNavItem(
    val route: String,
    val title: String,
    val icon: ImageVector,
    val iconContentDescription: String
) {
    object Main : BottomNavItem(
        route = Screens.Main.route,
        title = "Proyectos",
        icon = Icons.Default.List,
        iconContentDescription = "Proyectos"
    )

    object Donation : BottomNavItem(
        route = Screens.Donation.route,
        title = "Donaciones",
        icon = Icons.Default.Favorite,
        iconContentDescription = "Donaciones"
    )
```

```
    object Profile : BottomNavItem(
        route = Screens.MyProfile.route, // Corregido aquí
        title = "Perfil",
        icon = Icons.Default.Person,
        iconContentDescription = "Perfil"
    )

    object Complaint : BottomNavItem(
        route = Screens.Complaint.route,
        title = "Soporte",
        icon = Icons.Default.Warning,
        iconContentDescription = "Soporte"
    )

    companion object {
        val items = listOf(Main, Donation, Profile, Complaint)

        // Función para encontrar el item por ruta
        fun fromRoute(route: String?): BottomNavItem? {
            return items.find { it.route == route }
        }
    }
}
```

MainNavGraph

```
package com.hugoguerrero.tecno.ui.navigation
```

```
import androidx.compose.runtime.Composable
import androidx.navigation.NavGraphBuilder
import androidx.navigation.NavHostController
import androidx.navigation.NavType
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.navArgument
import androidx.navigation.navigation
import com.hugoguerrero.tecno.ui.screens.complaint.ComplaintsScreen
import com.hugoguerrero.tecno.ui.screens.donation.DonationScreen
import com.hugoguerrero.tecno.ui.screens.login.LoginScreen
import com.hugoguerrero.tecno.ui.screens.login.RegisterScreen
import com.hugoguerrero.tecno.ui.screens.main.MainScreen
import com.hugoguerrero.tecno.ui.screens.management.ManageDonationsScreen
import com.hugoguerrero.tecno.ui.screens.profile.CreditsScreen
import com.hugoguerrero.tecno.ui.screens.profile.EditProfileScreen
import com.hugoguerrero.tecno.ui.screens.profile.MyProfileScreen
import com.hugoguerrero.tecno.ui.screens.profile.PublicProfileScreen
import com.hugoguerrero.tecno.ui.screens.project.EditProjectScreen
import com.hugoguerrero.tecno.ui.screens.project.ProjectDetailScreen
import com.hugoguerrero.tecno.ui.screens.project.UploadProjectScreen



const val AUTH_GRAPH_ROUTE = "auth_graph"
const val MAIN_GRAPH_ROUTE = "main_graph"
```

```
@Composable
fun MainNavGraph(navController: NavHostController) {
    NavHost(
        navController = navController,
        startDestination = AUTH_GRAPH_ROUTE
    ) {
        // Grafo de Autenticación (Mundo Pre-Login)
        authGraph(navController)

        // Grafo Principal (Mundo Post-Login)
        mainGraph(navController)
    }
}

fun NavGraphBuilder.authGraph(navController: NavHostController) {
    navigation(
        startDestination = Screens.Login.route,
        route = AUTH_GRAPH_ROUTE
    ) {
        composable(Screens.Login.route) {
            LoginScreen(navController = navController)
        }
        composable(Screens.Register.route) {
            RegisterScreen(navController = navController)
        }
    }
}
```

```
fun NavGraphBuilder.mainGraph(navController: NavHostController) {
    navigation(
        startDestination = Screens.Main.route,
        route = MAIN_GRAPH_ROUTE
    ) {
        composable(Screens.Main.route) {
            MainScreen(navController = navController)
        }
        composable(Screens.UploadProject.route) {
            UploadProjectScreen(navController = navController)
        }
        composable(Screens.MyProfile.route) {
            MyProfileScreen(navController = navController)
        }
        composable(Screens.EditProfile.route) {
            EditProfileScreen(navController = navController)
        }
        composable(Screens.Credits.route) {
            CreditsScreen(navController = navController)
        }
        composable(
            route = Screens.EditProject.route + "/{projectId}",
            arguments = listOf(navArgument("projectId") { type = NavType.StringType })
        ) {
            EditProjectScreen(navController = navController)
        }
```

```
composable(
    route = Screens.ManageDonations.route + "/{projectId}",
    arguments = listOf(navArgument("projectId") { type = NavType.StringType })
) {
    ManageDonationsScreen(navController = navController)
}
composable(
    route = Screens.PublicProfile.route + "/{userId}",
    arguments = listOf(navArgument("userId") { type = NavType.StringType })
) {
    PublicProfileScreen(navController = navController)
}
composable(
    route = Screens.ProjectDetail.route + "/{projectId}",
    arguments = listOf(navArgument("projectId") { type = NavType.StringType })
) {
    ProjectDetailScreen(navController = navController)
}
composable(
    route = Screens.Donation.route + "/{projectId}",
    arguments = listOf(navArgument("projectId") { type = NavType.StringType })
) {
    DonationScreen(navController = navController)
}
composable(
    route = Screens.Complaint.route + "/{projectId}",
```

```
            arguments = listOf(navArgument("projectId") { type = NavType.StringType;
nullable = true })
        ) {
            ComplaintsScreen(navController = navController)
        }
    }
}
```

Screens

```
package com.hugoguerrero.tecno.ui.navigation


sealed class Screens(val route: String) {
    object Login : Screens("login")
    object Register : Screens("register")
    object Main : Screens("main")
    object UploadProject : Screens("upload_project")
    object MyProfile : Screens("my_profile")
    object PublicProfile : Screens("public_profile")
    object EditProfile : Screens("edit_profile")
    object Credits : Screens("credits") // Nueva ruta para los créditos
    object ProjectDetail : Screens("project_detail")
    object Donation : Screens("donation")
    object Complaint : Screens("complaint")
    object EditProject : Screens("edit_project")
    object ManageDonations : Screens("manage_donations")
}
```

```

Ruta: app/src/main/java/com/hugoguerrero/tecno/ui/screens/

```kotlin
```

Crea la carpeta complaint

ComplaintsScreen

```

```
package com.hugoguerrero.tecno.ui.screens.complaint

import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.filled.Send
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
```

```
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedCard
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SnackbarHost
import androidx.compose.material3.SnackbarHostState
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
import com.hugoguerrero.tecno.data.model.Complaint
import com.hugoguerrero.tecno.domain.use_case.CreateComplaintUseCase
import com.hugoguerrero.tecno.domain.use_case.GetCurrentUserUseCase
```

```
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ComplaintsScreen(
    navController: NavController,
    viewModel: ComplaintViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()
    val snackbarHostState = remember { SnackbarHostState() }
    val scope = rememberCoroutineScope()

    LaunchedEffect(state.userMessage) {
        state.userMessage?.let {
            scope.launch { snackbarHostState.showSnackbar(it) }
            viewModel.userMessageShown()
        }
    }

    LaunchedEffect(state.complaintSent) {
        if (state.complaintSent) {
            navController.popBackStack()
        }
```

```
    }

    Scaffold(
        snackbarHost = { SnackbarHost(snackbarHostState) },
        topBar = {
            TopAppBar(
                title = { Text("Reportar un Problema") },
                navigationIcon = {
                    IconButton(onClick = { navController.popBackStack() }) {
                        Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
                    }
                }
            )
        }
    ) { padding ->
        ComplaintForm(
            modifier = Modifier.padding(padding),
            state = state,
            viewModel = viewModel
        )
    }
}


@Composable
private fun ComplaintForm(modifier: Modifier, state: ComplaintState, viewModel:
ComplaintViewModel) {
    Column(
```

```
        modifier =
modifier.fillMaxSize().verticalScroll(rememberScrollState()).padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        OutlinedCard(Modifier.fillMaxWidth()) {
            Column(Modifier.padding(16.dp)) {
                Text("¿Qué problema has encontrado?", style =
MaterialTheme.typography.titleLarge)
                Spacer(Modifier.height(16.dp))
                OutlinedTextField(
                    value = state.reason,
                    onValueChange = viewModel::onReasonChange,
                    label = { Text("Motivo de la queja") },
                    modifier = Modifier.fillMaxWidth()
                )
                Spacer(Modifier.height(16.dp))
                OutlinedTextField(
                    value = state.description,
                    onValueChange = viewModel::onDescriptionChange,
                    label = { Text("Describe el problema en detalle") },
                    modifier = Modifier.fillMaxWidth(),
                    minLines = 5
                )
            }
        }
        Spacer(Modifier.height(24.dp))
        Button(
```

```
            onClick = { viewModel.submitComplaint() },
            enabled = !state.isLoading,
            modifier = Modifier.fillMaxWidth()
        ) {
            if (state.isLoading) {
                CircularProgressIndicator(modifier = Modifier.size(20.dp), strokeWidth = 2.dp)
            } else {
                Icon(Icons.Default.Send, contentDescription = null, modifier =
Modifier.padding(end = 8.dp))
                Text("Enviar Queja")
            }
        }
    }
}

@HiltViewModel
class ComplaintViewModel @Inject constructor(
    private val createComplaintUseCase: CreateComplaintUseCase,
    private val getCurrentUserUseCase: GetCurrentUserUseCase,
    savedStateHandle: SavedStateHandle
) : ViewModel() {

    private val _uiState = MutableStateFlow(ComplaintState())
    val uiState = _uiState.asStateFlow()

    private val projectId: String? = savedStateHandle.get<String>("projectId")
```

```
    fun onReasonChange(reason: String) { _uiState.value = _uiState.value.copy(reason = reason) }
    fun onDescriptionChange(description: String) { _uiState.value = _uiState.value.copy(description = description) }


    fun submitComplaint() {
        viewModelScope.launch {
            val state = _uiState.value
            val currentUser = getCurrentUserUseCase()


            if (currentUser == null) {
                _uiState.value = state.copy(userMessage = "Error de autenticación.")
                return@launch
            }
            if (state.reason.isBlank() || state.description.isBlank()) {
                _uiState.value = state.copy(userMessage = "Por favor, completa todos los campos.")
                return@launch
            }


            _uiState.value = state.copy(isLoading = true)


            val complaint = Complaint(
                projectId = projectId ?: "N/A",
                reporterUid = currentUser.uid,
                reason = state.reason,
                description = state.description
            )
```

```
            val result = createComplaintUseCase(complaint)


            result.onSuccess {
                _uiState.value = _uiState.value.copy(isLoading = false, complaintSent = true, userMessage = "Queja enviada con éxito.")
            }.onFailure {
                _uiState.value = _uiState.value.copy(isLoading = false, userMessage = it.message ?: "Error desconocido")
            }
        }
    }

    fun userMessageShown() {
        _uiState.value = _uiState.value.copy(userMessage = null)
    }
}

data class ComplaintState(
    val reason: String = "",
    val description: String = "",
    val isLoading: Boolean = false,
    val complaintSent: Boolean = false,
    val userMessage: String? = null
)
```

Crea la carpeta donation

DonationHistoryScreen

```

package com.hugoguerrero.tecno.ui.screens.donation

import android.R
import android.graphics.drawable.Icon
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DonationHistoryScreen(
    onBack: () -> Unit
) {
    // Datos de ejemplo para el historial
    val donationHistory = listOf(
        Donacion("Robótica para Ferias Científicas", "2023-10-15", 500.0),
        Donacion("BioSensor de Bajo Costo", "2023-09-20", 300.0),
        Donacion("Micro-red Solar", "2023-08-05", 1000.0)
    )

    Scaffold(
        topBar = {
            TopAppBar(
                title = {
                    Text(
                        "Historial de Donaciones",
                        color = Color.White,
```

```
                fontWeight = FontWeight.Bold,
                fontSize = 20.sp
            )
        },
        navigationIcon = {
            IconButton(onClick = onBack) {
                Icon(
                    painter = painterResource(R.drawable.ic_media_previous),
                    contentDescription = "Atrás",
                    tint = Color.White
                )
            }
        },
        colors = TopAppBarDefaults.topAppBarColors(containerColor =
Color(0xFF35002E))
      )
    },
    containerColor = Color(0xFF0F0A13)
  ) { paddingValues ->
    Column(
      modifier = Modifier
        .fillMaxSize()
        .padding(paddingValues)
        .padding(16.dp)
    ) {
      Text(
        "Tus donaciones",
```

```
    color = Color.White,
    fontSize = 18.sp,
    fontWeight = FontWeight.Bold,
    modifier = Modifier.padding(bottom = 16.dp)
)

LazyColumn(
    verticalArrangement = Arrangement.spacedBy(12.dp)
) {
    items(donationHistory) { donacion ->
        Card(
            colors = CardDefaults.cardColors(containerColor = Color(0xFF1A0D26)),
            shape = RoundedCornerShape(12.dp),
            modifier = Modifier.fillMaxWidth()
        ) {
            Column(modifier = Modifier.padding(16.dp)) {
                Text(
                    donacion.proyecto,
                    color = Color(0xFFF1E6FF),
                    fontWeight = FontWeight.Bold,
                    fontSize = 16.sp
                )
                Spacer(modifier = Modifier.height(8.dp))
                Row(
                    modifier = Modifier.fillMaxWidth(),
                    horizontalArrangement = Arrangement.SpaceBetween
                ) {
```

```
                        Text(
                            donacion.fecha,
                            color = Color(0xFFB497E5),
                            fontSize = 14.sp
                        )
                        Text(
                            "$${donacion.monto} MXN",
                            color = Color(0xFF00BFA6),
                            fontSize = 14.sp,
                            fontWeight = FontWeight.Bold
                        )
                    }
                }
            }
        }
    }
}

data class Donacion(
    val proyecto: String,
    val fecha: String,
    val monto: Double
)

@Preview(showBackground = true)
```

```
@Composable
fun PreviewDonationHistoryScreen() {
    DonationHistoryScreen(onBack = {})
}
```

DonationScreen

```
package com.hugoguerrero.tecno.ui.screens.donation

import android.net.Uri
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.filled.CloudUpload
```

```
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedButton
import androidx.compose.material3.OutlinedCard
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SnackbarHost
import androidx.compose.material3.SnackbarHostState
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
```

```
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
import com.google.firebase.storage.FirebaseStorage
import com.hugoguerrero.tecno.data.model.DonationStatus
import com.hugoguerrero.tecno.domain.use_case.CreateDonationUseCase
import com.hugoguerrero.tecno.domain.use_case.GetCurrentUserUseCase
import com.hugoguerrero.tecno.domain.use_case.GetProjectUseCase
import com.hugoguerrero.tecno.domain.use_case.GetUserUseCase
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import kotlinx.coroutines.tasks.await
import java.util.UUID
import javax.inject.Inject

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DonationScreen(
    navController: NavController,
    viewModel: DonationViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()
    val snackbarHostState = remember { SnackbarHostState() }
    val scope = rememberCoroutineScope()
```

```
val imagePickerLauncher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.GetContent(),
    onResult = { uri: Uri? -> viewModel.onProofUriChange(uri) }
)

LaunchedEffect(key1 = state.userMessage) {
    state.userMessage?.let {
        scope.launch { snackbarHostState.showSnackbar(it) }
        viewModel.userMessageShown()
    }
}

LaunchedEffect(key1 = state.donationCompleted) {
    if (state.donationCompleted) {
        navController.popBackStack()
    }
}

Scaffold(
    snackbarHost = { SnackbarHost(snackbarHostState) },
    topBar = {
        TopAppBar(
            title = { Text("Apoyar Proyecto") },
            navigationIcon = {
                IconButton(onClick = { navController.popBackStack() }) {
                    Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
```

```
                }
            }
        )
    }
) { padding ->
    DonationForm(
        modifier = Modifier.padding(padding),
        state = state,
        viewModel = viewModel,
        onSelectProof = { imagePickerLauncher.launch("image/*") }
    )
}
}

@Composable
private fun DonationForm(
    modifier: Modifier,
    state: DonationState,
    viewModel: DonationViewModel,
    onSelectProof: () -> Unit
) {
    Column(
        modifier =
modifier.fillMaxSize().verticalScroll(rememberScrollState()).padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
```

```
        if (state.isLoadingProject) {
            CircularProgressIndicator()
        } else {
            OutlinedCard(Modifier.fillMaxWidth()) {
                Column(Modifier.padding(16.dp)) {
                    Text("Datos para la transferencia", style =
MaterialTheme.typography.titleLarge)
                    Spacer(Modifier.height(8.dp))
                    Text(state.paymentDetails, style = MaterialTheme.typography.bodyLarge,
fontWeight = FontWeight.Bold)
                }
            }

            OutlinedCard(Modifier.fillMaxWidth()) {
                Column(Modifier.padding(16.dp)) {
                    OutlinedTextField(
                        value = state.amount,
                        onValueChange = viewModel::onAmountChange,
                        label = { Text("Monto Donado") },
                        prefix = { Text("$") },
                        keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
                        modifier = Modifier.fillMaxWidth()
                    )
                    Spacer(Modifier.height(8.dp))
                    OutlinedTextField(
                        value = state.message,
                        onValueChange = viewModel::onMessageChange,
```

```
                label = { Text("Mensaje (opcional)") },
                modifier = Modifier.fillMaxWidth()
            )
        }
    }

    OutlinedButton(onClick = onSelectProof) {
        Text(if (state.proofUri != null) "Comprobante seleccionado" else "Seleccionar Comprobante")
    }

    Button(
        onClick = { viewModel.sendDonationWithProof() },
        enabled = !state.isDonating && (state.amount.toDoubleOrNull() ?: 0.0) > 0 && state.proofUri != null,
        modifier = Modifier.fillMaxWidth()
    ) {
        if (state.isDonating) {
            CircularProgressIndicator(modifier = Modifier.size(20.dp), strokeWidth = 2.dp)
        } else {
            Icon(Icons.Default.CloudUpload, contentDescription = null, modifier = Modifier.padding(end = 8.dp))
            Text("Confirmar Donación")
        }
    }
  }
}
```

```
}

@HiltViewModel
class DonationViewModel @Inject constructor(
    private val createDonationUseCase: CreateDonationUseCase,
    private val getCurrentUserUseCase: GetCurrentUserUseCase,
    private val getProjectUseCase: GetProjectUseCase,
    private val getUserUseCase: GetUserUseCase,
    private val storage: FirebaseStorage,
    private val savedStateHandle: SavedStateHandle
) : ViewModel() {

    private val _uiState = MutableStateFlow(DonationState())
    val uiState = _uiState.asStateFlow()

    private val projectId: String = savedStateHandle.get<String>("projectId")!!

    init {
        loadProjectCreatorDetails()
    }

    private fun loadProjectCreatorDetails() {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(isLoadingProject = true)
            try {
                val project = getProjectUseCase(projectId)
                if (project == null) throw Exception("Proyecto no encontrado.")
```

```
            val creator = getUserUseCase(project.ownerUid)
            if (creator == null) throw Exception("Creador del proyecto no encontrado.")

            _uiState.value = _uiState.value.copy(
                paymentDetails = creator.paymentDetails ?: "El creador no ha proporcionado datos de pago.",
                isLoadingProject = false
            )
        } catch (e: Exception) {
            _uiState.value = _uiState.value.copy(userMessage = e.message ?: "Error al cargar la información.", isLoadingProject = false)
        }
    }
}

fun onAmountChange(amount: String) {
    _uiState.value = _uiState.value.copy(amount = amount.filter { it.isDigit() || it == '.' })
}

fun onMessageChange(message: String) {
    _uiState.value = _uiState.value.copy(message = message)
}

fun onProofUriChange(uri: Uri?) {
    _uiState.value = _uiState.value.copy(proofUri = uri)
}
```

```
    fun sendDonationWithProof() {
        viewModelScope.launch {
            val state = _uiState.value
            val donor = getCurrentUserUseCase()
            val amount = state.amount.toDoubleOrNull()

            if (donor == null) {
                _uiState.value = state.copy(userMessage = "Error de autenticación.")
                return@launch
            }
            if (amount == null || amount <= 0) {
                _uiState.value = state.copy(userMessage = "Por favor, introduce un monto
válido.")
                return@launch
            }
            if (state.proofUri == null) {
                _uiState.value = state.copy(userMessage = "Por favor, selecciona un
comprobante.")
                return@launch
            }

            _uiState.value = state.copy(isDonating = true, userMessage = null)

            try {
                val proofFileName = "${UUID.randomUUID()}"
                val proofRef = storage.reference.child("donation_proofs/$proofFileName")
                val uploadTask = proofRef.putFile(state.proofUri).await()
                val proofUrl = uploadTask.storage.downloadUrl.await().toString()
```

```
            val result = createDonationUseCase(
                projectId = projectId,
                donorUid = donor.uid,
                amount = amount,
                message = state.message,
                status = DonationStatus.PENDING,
                proofImageUrl = proofUrl
            )

            result.onSuccess {
                _uiState.value = _uiState.value.copy(isDonating = false, donationCompleted =
true, userMessage = "¡Gracias por tu donación!")
            }.onFailure { throw it }

        } catch (e: Exception) {
            _uiState.value = state.copy(isDonating = false, userMessage = e.message ?:
"Error al enviar la donación.")
        }
    }
}

fun userMessageShown() {
    _uiState.value = _uiState.value.copy(userMessage = null)
}
}

data class DonationState(
```

```
    val isLoadingProject: Boolean = true,
    val isDonating: Boolean = false,
    val donationCompleted: Boolean = false,
    val paymentDetails: String = "",
    val amount: String = "",
    val message: String = "",
    val proofUri: Uri? = null,
    val userMessage: String? = null
)
```

Crea la carpeta login

LoginScreen

```
package com.hugoguerrero.tecno.ui.screens.login

import android.app.Activity
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.IntentSenderRequest
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.foundation.verticalScroll
```

```
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
import com.google.android.gms.auth.api.identity.BeginSignInRequest
import com.google.android.gms.auth.api.identity.Identity
import com.google.android.gms.common.api.ApiException
import com.google.firebase.auth.AuthCredential
import com.google.firebase.auth.GoogleAuthProvider
```

```
import com.hugoguerrero.tecno.R
import com.hugoguerrero.tecno.domain.model.UserType
import com.hugoguerrero.tecno.domain.use_case.*
import com.hugoguerrero.tecno.ui.navigation.AUTH_GRAPH_ROUTE
import com.hugoguerrero.tecno.ui.navigation.MAIN_GRAPH_ROUTE
import com.hugoguerrero.tecno.ui.navigation.Screens
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import kotlinx.coroutines.tasks.await
import javax.inject.Inject

@Composable
fun LoginScreen(
    navController: NavController,
    viewModel: LoginViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()
    val context = LocalContext.current

    val googleSignInLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.StartIntentSenderForResult(),
        onResult = { result ->
```

```
        if (result.resultCode == Activity.RESULT_OK) {
            try {
                val credential =
Identity.getSignInClient(context).getSignInCredentialFromIntent(result.data)
                val googleIdToken = credential.googleIdToken
                if (googleIdToken != null) {
                    val googleCredential = GoogleAuthProvider.getCredential(googleIdToken,
null)
                    viewModel.signInWithGoogle(googleCredential)
                }
            } catch (e: ApiException) {
                viewModel.showError(e.message ?: "Error de Google Sign-In")
            }
        }
    }
  )


  LaunchedEffect(key1 = viewModel) {
    viewModel.navigationEvent.collect { navigation ->
      when (navigation) {
        is LoginNavigation.MainScreen -> {
          navController.navigate(MAIN_GRAPH_ROUTE) {
popUpTo(AUTH_GRAPH_ROUTE) { inclusive = true } }
        }
        is LoginNavigation.RegisterScreen -> {
          navController.navigate(Screens.Register.route)
        }
      }
```

```
        }
    }

    LaunchedEffect(key1 = true) {
        viewModel.checkAuthState()
    }

    Surface(modifier = Modifier.fillMaxSize()) {
        Column(
            modifier =
Modifier.fillMaxSize().padding(24.dp).verticalScroll(rememberScrollState()),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            AppLogo()
            Spacer(modifier = Modifier.height(24.dp))
            LoginForm(state = state, viewModel = viewModel, navController = navController)
            Spacer(modifier = Modifier.height(24.dp))
            SocialLoginButtons(state = state, onGoogleSignInClick = {
                viewModel.startGoogleSignIn { intentSenderRequest ->
googleSignInLauncher.launch(intentSenderRequest) }
            })
        }
    }
}

@Composable
private fun AppLogo() {
```

```
    Row(verticalAlignment = Alignment.CenterVertically) {
        Icon(
            painter = painterResource(id = R.drawable.logo),
            contentDescription = null,
            modifier = Modifier.size(32.dp),
            tint = MaterialTheme.colorScheme.primary
        )
        Text(
            text = "Tecno-Ciencia",
            style = MaterialTheme.typography.headlineSmall,
            fontWeight = FontWeight.SemiBold,
            modifier = Modifier.padding(start = 8.dp)
        )
    }
}


@Composable
private fun LoginForm(state: LoginState, viewModel: LoginViewModel, navController:
NavController) {
    OutlinedCard(modifier = Modifier.fillMaxWidth()) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text("Iniciar Sesión", style = MaterialTheme.typography.titleLarge, fontWeight =
FontWeight.Bold)
            Spacer(modifier = Modifier.height(16.dp))


            state.error?.let {
                Text(text = it, color = MaterialTheme.colorScheme.error, style =
MaterialTheme.typography.bodySmall)
```

```
    Spacer(modifier = Modifier.height(8.dp))
}

OutlinedTextField(
    value = state.email,
    onValueChange = viewModel::onEmailChange,
    label = { Text("Correo Electrónico") },
    modifier = Modifier.fillMaxWidth(),
    singleLine = true
)
Spacer(modifier = Modifier.height(8.dp))
OutlinedTextField(
    value = state.password,
    onValueChange = viewModel::onPasswordChange,
    label = { Text("Contraseña") },
    visualTransformation = PasswordVisualTransformation(),
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
    modifier = Modifier.fillMaxWidth(),
    singleLine = true
)
Spacer(modifier = Modifier.height(16.dp))
Button(
    onClick = { viewModel.signInWithEmail() },
    enabled = !state.isLoading,
    modifier = Modifier.fillMaxWidth()
) {
    if (state.isLoading) {
```

```
                CircularProgressIndicator(modifier = Modifier.size(20.dp), strokeWidth =
2.dp, color = MaterialTheme.colorScheme.onPrimary)
            } else {
                Text("Entrar")
            }
        }
        Spacer(modifier = Modifier.height(8.dp))
        TextButton(onClick = { navController.navigate(Screens.Register.route) }, modifier
= Modifier.align(Alignment.CenterHorizontally)) {
            Text("¿No tienes cuenta? Regístrate")
        }
    }
  }
}


@Composable
private fun SocialLoginButtons(state: LoginState, onGoogleSignInClick: () -> Unit) {
  Column(horizontalAlignment = Alignment.CenterHorizontally) {
    Text("O continúa con", style = MaterialTheme.typography.bodyMedium, color =
MaterialTheme.colorScheme.onSurfaceVariant)
    Spacer(modifier = Modifier.height(16.dp))
    OutlinedButton(
      onClick = onGoogleSignInClick,
      enabled = !state.isLoading,
      modifier = Modifier.fillMaxWidth(),
      shape = RoundedCornerShape(50)
    ) {
      Icon(
```

```
                painter = painterResource(id = R.drawable.google_logo),
                contentDescription = null,
                modifier = Modifier.size(20.dp)
            )
            Text("Google", modifier = Modifier.padding(start = 12.dp))
        }
    }
}

@HiltViewModel
class LoginViewModel @Inject constructor(
    private val getCurrentUserUseCase: GetCurrentUserUseCase,
    private val signInWithGoogleUseCase: SignInWithGoogleUseCase,
    private val signInWithEmailUseCase: SignInWithEmailUseCase,
    private val checkUserProfileExistsUseCase: CheckUserProfileExistsUseCase,
    private val createUserProfileUseCase: CreateUserProfileUseCase,
    private val application: android.app.Application
) : ViewModel() {

    private val _uiState = MutableStateFlow(LoginState())
    val uiState: StateFlow<LoginState> = _uiState.asStateFlow()

    private val _navigationEvent = MutableSharedFlow<LoginNavigation>()
    val navigationEvent = _navigationEvent.asSharedFlow()

    fun onEmailChange(email: String) { _uiState.value = _uiState.value.copy(email = email)
}
```

```
    fun onPasswordChange(password: String) { _uiState.value =
_uiState.value.copy(password = password) }


    fun signInWithEmail() {
        viewModelScope.launch {
            val state = _uiState.value
            _uiState.value = state.copy(isLoading = true, error = null)
            signInWithEmailUseCase(state.email, state.password).collect { result ->
                result.onSuccess {
                    _uiState.value = _uiState.value.copy(isLoading = false)
                    _navigationEvent.emit(LoginNavigation.MainScreen)
                }.onFailure {
                    _uiState.value = _uiState.value.copy(isLoading = false, error = it.message)
                }
            }
        }
    }

    fun startGoogleSignIn(launcher: (IntentSenderRequest) -> Unit) {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(isLoading = true)


            val signInRequest = BeginSignInRequest.builder()
                .setGoogleIdTokenRequestOptions(
                    BeginSignInRequest.GoogleIdTokenRequestOptions.builder()
                        .setSupported(true)
                        .setServerClientId(application.getString(R.string.web_client_id))
```

```
                .setFilterByAuthorizedAccounts(false)
                .build()
            )
            .build()

        try {
            val result =
Identity.getSignInClient(application).beginSignIn(signInRequest).await()
            launcher(IntentSenderRequest.Builder(result.pendingIntent.intentSender).build())
        } catch (e: Exception) {
            _uiState.value = _uiState.value.copy(isLoading = false, error = e.message)
        }
    }
}

fun signInWithGoogle(credential: AuthCredential) {
    viewModelScope.launch {
        _uiState.value = _uiState.value.copy(isLoading = true, error = null)
        signInWithGoogleUseCase(credential).collect { result ->
            result.onSuccess { user ->
                if (!checkUserProfileExistsUseCase(user.uid)) {
                    createUserProfileUseCase(user, user.displayName ?: "Usuario",
UserType.STUDENT)
                }
                _uiState.value = _uiState.value.copy(isLoading = false)
                _navigationEvent.emit(LoginNavigation.MainScreen)
            }.onFailure { exception ->
```

```
                _uiState.value = _uiState.value.copy(isLoading = false, error =
exception.message)
            }
        }
    }

    fun checkAuthState() {
        viewModelScope.launch {
            if (getCurrentUserUseCase() != null) {
                _navigationEvent.emit(LoginNavigation.MainScreen)
            }
        }
    }

    fun showError(message: String) {
        _uiState.value = _uiState.value.copy(error = message)
    }
}

data class LoginState(
    val email: String = "",
    val password: String = "",
    val isLoading: Boolean = false,
    val error: String? = null
)
```

```
sealed class LoginNavigation {
    object MainScreen : LoginNavigation()
    object RegisterScreen : LoginNavigation()
}
```

RegisterScreen

```
package com.hugoguerrero.tecno.ui.screens.login

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Person
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
```

```
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedCard
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.SegmentedButton
import androidx.compose.material3.SegmentedButtonDefaults
import androidx.compose.material3.SingleChoiceSegmentedButtonRow
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
import com.hugoguerrero.tecno.domain.model.UserType
import com.hugoguerrero.tecno.domain.use_case.RegisterUserUseCase
import com.hugoguerrero.tecno.ui.navigation.MAIN_GRAPH_ROUTE
```

```
import com.hugoguerrero.tecno.ui.navigation.Screens
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch
import javax.inject.Inject

sealed class NavigationEvent {
    object NavigateToMain : NavigationEvent()
    object NavigateToLogin : NavigationEvent()
}

@Composable
fun RegisterScreen(
    navController: NavController,
    viewModel: RegisterViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()

    LaunchedEffect(key1 = true) {
        viewModel.navigationEvent.collectLatest { event ->
            when (event) {
                is NavigationEvent.NavigateToMain -> {
                    navController.navigate(MAIN_GRAPH_ROUTE) {
```

```
                    popUpTo(Screens.Login.route) { inclusive = true }
                }
            }
            is NavigationEvent.NavigateToLogin -> {
                navController.popBackStack()
            }
        }
    }
}

Surface(modifier = Modifier.fillMaxSize()) {
    Column(
        modifier =
Modifier.fillMaxSize().padding(24.dp).verticalScroll(rememberScrollState()),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        RegisterForm(state = state, viewModel = viewModel)
    }
}
}

@Composable
private fun RegisterForm(state: RegisterState, viewModel: RegisterViewModel) {
    OutlinedCard(modifier = Modifier.fillMaxWidth()) {
        Column(modifier = Modifier.padding(16.dp), horizontalAlignment =
Alignment.CenterHorizontally) {
```

```
        Text("Crear Cuenta", style = MaterialTheme.typography.titleLarge, fontWeight =
FontWeight.Bold)

        Text("Únete a la comunidad científica", style =
MaterialTheme.typography.bodyMedium, color =
MaterialTheme.colorScheme.onSurfaceVariant)

        Spacer(modifier = Modifier.height(16.dp))


        state.error?.let {

            Text(text = it, color = MaterialTheme.colorScheme.error, style =
MaterialTheme.typography.bodySmall)

            Spacer(modifier = Modifier.height(8.dp))

        }


        UserTypeSelection(selectedType = state.userType, onTypeSelected =
viewModel::onUserTypeChange)

        Spacer(modifier = Modifier.height(16.dp))


        OutlinedTextField(

            value = state.name,

            onValueChange = viewModel::onNameChange,

            label = { Text("Nombre completo") },

            modifier = Modifier.fillMaxWidth(),

            singleLine = true

        )

        Spacer(modifier = Modifier.height(8.dp))

        OutlinedTextField(

            value = state.email,

            onValueChange = viewModel::onEmailChange,

            label = { Text("Correo electrónico") },
```

```
    modifier = Modifier.fillMaxWidth(),
    singleLine = true,
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email)
)
Spacer(modifier = Modifier.height(8.dp))
OutlinedTextField(
    value = state.password,
    onValueChange = viewModel::onPasswordChange,
    label = { Text("Contraseña") },
    visualTransformation = PasswordVisualTransformation(),
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
    modifier = Modifier.fillMaxWidth(),
    singleLine = true
)
Spacer(modifier = Modifier.height(8.dp))
OutlinedTextField(
    value = state.confirmPassword,
    onValueChange = viewModel::onConfirmPasswordChange,
    label = { Text("Confirmar contraseña") },
    visualTransformation = PasswordVisualTransformation(),
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
    modifier = Modifier.fillMaxWidth(),
    singleLine = true
)
Spacer(modifier = Modifier.height(24.dp))
Button(
    onClick = { viewModel.register() },
```

```
            enabled = !state.isLoading,
            modifier = Modifier.fillMaxWidth()
        ) {
            if (state.isLoading) {
                CircularProgressIndicator(modifier = Modifier.size(20.dp), strokeWidth =
2.dp)
            } else {
                Text("Registrarse")
            }
        }
        Spacer(modifier = Modifier.height(8.dp))
        TextButton(onClick = { viewModel.navigateToLogin() }) {
            Text("¿Ya tienes cuenta? Inicia sesión")
        }
    }
  }
}


@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun UserTypeSelection(selectedType: UserType, onTypeSelected: (UserType) ->
Unit) {
  val userTypes = UserType.values()
  SingleChoiceSegmentedButtonRow(modifier = Modifier.fillMaxWidth()) {
    userTypes.forEachIndexed { index, userType ->
      SegmentedButton(
        shape = SegmentedButtonDefaults.itemShape(index = index, count =
userTypes.size),
```

```
                onClick = { onTypeSelected(userType) },
                selected = userType == selectedType,
                icon = {
                    Icon(
                        imageVector = Icons.Default.Person,
                        contentDescription = null,
                        modifier = Modifier.size(SegmentedButtonDefaults.IconSize)
                    )
                }
            ) {
                Text(userType.name.lowercase().replaceFirstChar { it.titlecase() })
            }
        }
    }
}

@HiltViewModel
class RegisterViewModel @Inject constructor(
    private val registerUserUseCase: RegisterUserUseCase
) : ViewModel() {

    private val _uiState = MutableStateFlow(RegisterState())
    val uiState = _uiState.asStateFlow()

    private val _navigationEvent = MutableSharedFlow<NavigationEvent>()
    val navigationEvent = _navigationEvent.asSharedFlow()
```

```
    fun onNameChange(name: String) { _uiState.value = _uiState.value.copy(name = name)
}

    fun onEmailChange(email: String) { _uiState.value = _uiState.value.copy(email = email)
}

    fun onPasswordChange(password: String) { _uiState.value =
_uiState.value.copy(password = password) }

    fun onConfirmPasswordChange(password: String) { _uiState.value =
_uiState.value.copy(confirmPassword = password) }

    fun onUserTypeChange(userType: UserType) { _uiState.value =
_uiState.value.copy(userType = userType) }


    fun register() {
        viewModelScope.launch {
            val state = _uiState.value
            if (state.password != state.confirmPassword) {
                _uiState.value = _uiState.value.copy(error = "Las contraseñas no coinciden.")
                return@launch
            }


            _uiState.value = state.copy(isLoading = true, error = null)


            registerUserUseCase(state.name, state.email, state.password, state.userType)
                .collect { result ->
                    result.onSuccess {
                        _uiState.value = _uiState.value.copy(isLoading = false)
                        viewModelScope.launch {
_navigationEvent.emit(NavigationEvent.NavigateToMain) }
                    }.onFailure {
```

```
                    _uiState.value = _uiState.value.copy(isLoading = false, error = it.message ?:
"Error desconocido")
                }
            }
        }
    }

    fun navigateToLogin() {
        viewModelScope.launch { _navigationEvent.emit(NavigationEvent.NavigateToLogin)
}
    }
}

data class RegisterState(
    val name: String = "",
    val email: String = "",
    val password: String = "",
    val confirmPassword: String = "",
    val userType: UserType = UserType.STUDENT,
    val isLoading: Boolean = false,
    val error: String? = null
)
```

Crea la carpeta main

MainScreen

```
package com.hugoguerrero.tecno.ui.screens.main
```

```
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.ListAlt
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.FloatingActionButton
import androidx.compose.material3.Icon
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
```

```
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.domain.use_case.GetProjectsUseCase
import com.hugoguerrero.tecno.ui.components.BottomNavigationBar
import com.hugoguerrero.tecno.ui.components.ProjectCard
import com.hugoguerrero.tecno.ui.navigation.Screens
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.catch
import kotlinx.coroutines.flow.launchIn
import kotlinx.coroutines.flow.onEach
import javax.inject.Inject

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MainScreen(
    navController: NavController,
    viewModel: MainViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()

    Scaffold(
```

```
        topBar = { TopAppBar(title = { Text("Proyectos Recientes") }) },
        bottomBar = { BottomNavigationBar(navController) },
        floatingActionButton = {
            FloatingActionButton(onClick = {
navController.navigate(Screens.UploadProject.route) }) {
                Icon(Icons.Default.Add, contentDescription = "Subir Proyecto")
            }
        }
    ) { padding ->
        MainContent(
            modifier = Modifier.padding(padding),
            state = state,
            onProjectClick = { projectId ->
                navController.navigate(Screens.ProjectDetail.route + "/${projectId}")
            },
            onUploadClick = { navController.navigate(Screens.UploadProject.route) }
        )
    }
}

@Composable
private fun MainContent(
    modifier: Modifier = Modifier,
    state: MainState,
    onProjectClick: (String) -> Unit,
    onUploadClick: () -> Unit
) {
```

```
Box(modifier = modifier.fillMaxSize()) {
    when {
        state.isLoading -> {
            CircularProgressIndicator(modifier = Modifier.align(Alignment.Center))
        }
        state.error != null -> {
            Text(
                text = state.error,
                modifier = Modifier.align(Alignment.Center),
                textAlign = TextAlign.Center
            )
        }
        state.projects.isEmpty() -> {
            EmptyState(onUploadClick)
        }
        else -> {
            LazyColumn(
                contentPadding = PaddingValues(16.dp),
                verticalArrangement = Arrangement.spacedBy(16.dp)
            ) {
                items(state.projects) { project ->
                    ProjectCard(project = project, onClick = { onProjectClick(project.id) })
                }
            }
        }
    }
}
```

```
}

@Composable
private fun EmptyState(onUploadClick: () -> Unit) {
    Column(
        modifier = Modifier.fillMaxSize().padding(16.dp),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Icon(Icons.Default.ListAlt, contentDescription = null, modifier =
Modifier.size(64.dp))
        Text(
            text = "Aún no hay proyectos disponibles. ¡Sé el primero en crear uno!",
            modifier = Modifier.padding(top = 16.dp),
            textAlign = TextAlign.Center
        )
        Button(onClick = onUploadClick, modifier = Modifier.padding(top = 24.dp)) {
            Text("Crear un Proyecto")
        }
    }
}

@HiltViewModel
class MainViewModel @Inject constructor(
    private val getProjectsUseCase: GetProjectsUseCase
) : ViewModel() {
```

```
    private val _uiState = MutableStateFlow(MainState(isLoading = true))
    val uiState: StateFlow<MainState> = _uiState.asStateFlow()

    init {
        getProjectsUseCase()
            .onEach { projects ->
                _uiState.value = MainState(projects = projects)
            }
            .catch { exception ->
                _uiState.value = MainState(error = exception.message)
            }
            .launchIn(viewModelScope)
    }
}

data class MainState(
    val projects: List<Project> = emptyList(),
    val isLoading: Boolean = false,
    val error: String? = null
)
```

RegisterScreen

```
package com.hugoguerrero.tecno.ui.screens.main

import androidx.compose.foundation.Image
```

```
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Icon
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.OutlinedTextFieldDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
```

```
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.hugoguerrero.tecno.R

@Composable
fun RegisterScreen(
    onRegisterSuccess: () -> Unit,
    onLoginClick: () -> Unit
) {
    var name by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var confirmPassword by remember { mutableStateOf("") }

    Box(
        modifier = Modifier
            .fillMaxSize()
```

```
        .background(Color(0xFF0D1B2A))
        .verticalScroll(rememberScrollState())
        .padding(24.dp)
) {
    Column(
        horizontalAlignment = Alignment.CenterHorizontally,
        modifier = Modifier.fillMaxWidth()
    ) {
        // --- Logo y nombre ---
        Row(
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.Center,
            modifier = Modifier.padding(top = 32.dp, bottom = 24.dp)
        ) {
            Icon(
                painter = painterResource(id = R.drawable.ic_launcher_foreground),
                contentDescription = null,
                tint = Color.White,
                modifier = Modifier.size(28.dp)
            )
            Text(
                text = "Tecno-Ciencia",
                color = Color.White,
                fontSize = 18.sp,
                fontWeight = FontWeight.SemiBold,
                modifier = Modifier.padding(start = 8.dp)
            )
```

```
        }

        // --- Título y descripción ---
        Text(
            text = "Crea tu cuenta",
            color = Color.White,
            fontSize = 22.sp,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(bottom = 8.dp)
        )
        Text(
            text = "Completa tus datos para registrarte.",
            color = Color(0xFFB0BEC5),
            fontSize = 14.sp,
            modifier = Modifier.padding(bottom = 16.dp)
        )

        // --- Imagen superior ---
        Image(
            painter = painterResource(id = R.drawable.ic_launcher_foreground), // usa la
imagen correcta
            contentDescription = "Registro",
            contentScale = ContentScale.Crop,
            modifier = Modifier
                .fillMaxWidth()
                .height(180.dp)
                .clip(RoundedCornerShape(16.dp))
```

```
        .background(Color(0xFF1B263B))
        .padding(bottom = 24.dp)
)

// --- Campo de nombre ---
OutlinedTextField(
    value = name,
    onValueChange = { name = it },
    label = { Text("Nombre completo") },
    placeholder = { Text("Tu nombre completo") },
    singleLine = true,
    leadingIcon = {
        Image(
            painter = painterResource(id = R.drawable.ic_launcher_foreground),
            contentDescription = "Icono persona",
            modifier = Modifier
                .size(22.dp)
                .padding(2.dp)
        )
    },
    colors = OutlinedTextFieldDefaults.colors(
        focusedContainerColor = Color(0xFF1B263B),
        unfocusedContainerColor = Color(0xFF1B263B),
        focusedTextColor = Color.White,
        unfocusedTextColor = Color.White,
        focusedBorderColor = Color(0xFF00BFA6),
        unfocusedBorderColor = Color(0xFF455A64),
```

```
        focusedLabelColor = Color(0xFF00BFA6),
        unfocusedLabelColor = Color.White
    ),
    modifier = Modifier
        .fillMaxWidth()
        .padding(bottom = 16.dp)
)

// --- Campo de correo ---
OutlinedTextField(
    value = email,
    onValueChange = { email = it },
    label = { Text("Correo electrónico") },
    placeholder = { Text("nombre@universidad.edu") },
    singleLine = true,
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),
    leadingIcon = {
        Image(
            painter = painterResource(id = R.drawable.ic_launcher_foreground),
            contentDescription = "Icono correo",
            modifier = Modifier
                .size(22.dp)
                .padding(2.dp)
        )
    },
    colors = OutlinedTextFieldDefaults.colors(
        focusedContainerColor = Color(0xFF1B263B),
```

```
            unfocusedContainerColor = Color(0xFF1B263B),
            focusedTextColor = Color.White,
            unfocusedTextColor = Color.White,
            focusedBorderColor = Color(0xFF00BFA6),
            unfocusedBorderColor = Color(0xFF455A64),
            focusedLabelColor = Color(0xFF00BFA6),
            unfocusedLabelColor = Color.White
        ),
        modifier = Modifier
            .fillMaxWidth()
            .padding(bottom = 16.dp)
    )

    // --- Campo de contraseña ---
    OutlinedTextField(
        value = password,
        onValueChange = { password = it },
        label = { Text("Contraseña") },
        placeholder = { Text("••••••••") },
        singleLine = true,
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
        visualTransformation = PasswordVisualTransformation(),
        leadingIcon = {
            Image(
                painter = painterResource(id = R.drawable.ic_launcher_foreground),
                contentDescription = "Icono contraseña",
                modifier = Modifier
```

```
                .size(22.dp)
                .padding(2.dp)
        )
    },
    colors = OutlinedTextFieldDefaults.colors(
        focusedContainerColor = Color(0xFF1B263B),
        unfocusedContainerColor = Color(0xFF1B263B),
        focusedTextColor = Color.White,
        unfocusedTextColor = Color.White,
        focusedBorderColor = Color(0xFF00BFA6),
        unfocusedBorderColor = Color(0xFF455A64),
        focusedLabelColor = Color(0xFF00BFA6),
        unfocusedLabelColor = Color.White
    ),
    modifier = Modifier
        .fillMaxWidth()
        .padding(bottom = 16.dp)
)

// --- Campo de confirmar contraseña ---
OutlinedTextField(
    value = confirmPassword,
    onValueChange = { confirmPassword = it },
    label = { Text("Confirmar contraseña") },
    placeholder = { Text("••••••••") },
    singleLine = true,
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
```

```
    visualTransformation = PasswordVisualTransformation(),
    leadingIcon = {
        Image(
            painter = painterResource(id = R.drawable.ic_launcher_foreground),
            contentDescription = "Icono contraseña",
            modifier = Modifier
                .size(22.dp)
                .padding(2.dp)
        )
    },
    colors = OutlinedTextFieldDefaults.colors(
        focusedContainerColor = Color(0xFF1B263B),
        unfocusedContainerColor = Color(0xFF1B263B),
        focusedTextColor = Color.White,
        unfocusedTextColor = Color.White,
        focusedBorderColor = Color(0xFF00BFA6),
        unfocusedBorderColor = Color(0xFF455A64),
        focusedLabelColor = Color(0xFF00BFA6),
        unfocusedLabelColor = Color.White
    ),
    modifier = Modifier
        .fillMaxWidth()
        .padding(bottom = 24.dp)
)

// --- Botón registrarse ---
Button(
```

```
    onClick = { onRegisterSuccess() },
    colors = ButtonDefaults.buttonColors(containerColor = Color(0xFF00BFA6)),
    shape = RoundedCornerShape(12.dp),
    modifier = Modifier
        .fillMaxWidth()
        .padding(bottom = 16.dp)
        .height(48.dp)
) {
    Text(
        text = "Registrarse",
        color = Color.White,
        fontSize = 16.sp
    )
}

// --- Texto de inicio de sesión ---
Row(
    modifier = Modifier.fillMaxWidth(),
    horizontalArrangement = Arrangement.Center
) {
    Text(
        text = "¿Ya tienes cuenta? ",
        color = Color(0xFF90A4AE),
        fontSize = 13.sp
    )
    Text(
        text = "Inicia sesión",
```

```
                color = Color(0xFF00BFA6),
                fontSize = 13.sp,
                fontWeight = FontWeight.Bold,
                modifier = Modifier.clickable { onLoginClick() }
            )
        }

        // --- Footer ---
        Text(
            text = "© 2025 Tecno-Ciencia     Términos · Privacidad",
            color = Color(0xFF607D8B),
            fontSize = 11.sp,
            textAlign = TextAlign.Center,
            modifier = Modifier.padding(top = 32.dp)
        )
    }
  }
}

@Preview(showBackground = true)
@Composable
fun PreviewRegisterScreen() {
  RegisterScreen(
    onRegisterSuccess = { },
    onLoginClick = { }
  )
}
```

```

Crea la carpeta management

ManageDonationsScreen

```

package com.hugoguerrero.tecno.ui.screens.management

import android.content.Intent

import android.net.Uri

import androidx.compose.foundation.clickable

import androidx.compose.foundation.layout.Arrangement

import androidx.compose.foundation.layout.Box

import androidx.compose.foundation.layout.Column

import androidx.compose.foundation.layout.PaddingValues

import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.Spacer

import androidx.compose.foundation.layout.fillMaxSize

import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.height

import androidx.compose.foundation.layout.padding

import androidx.compose.foundation.layout.size

import androidx.compose.foundation.layout.width

import androidx.compose.foundation.lazy.LazyColumn

import androidx.compose.foundation.lazy.items

import androidx.compose.material.icons.Icons

import androidx.compose.material.icons.automirrored.filled.ArrowBack

import androidx.compose.material.icons.filled.Cancel

import androidx.compose.material.icons.filled.CheckCircle

```
import androidx.compose.material.icons.filled.HourglassEmpty
import androidx.compose.material.icons.filled.Info
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
import coil.compose.AsyncImage
import com.hugoguerrero.tecno.data.model.Donation
import com.hugoguerrero.tecno.data.model.DonationStatus
import com.hugoguerrero.tecno.domain.use_case.GetDonationsForProjectUseCase
import com.hugoguerrero.tecno.domain.use_case.UpdateDonationStatusUseCase
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.*
```

```
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.util.Locale
import javax.inject.Inject

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ManageDonationsScreen(
    navController: NavController,
    viewModel: ManageDonationsViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Gestionar Donaciones") },
                navigationIcon = {
                    IconButton(onClick = { navController.popBackStack() }) {
                        Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
                    }
                }
            )
        }
    ) { padding ->
        ManageDonationsContent(Modifier.padding(padding), state, viewModel)
    }
```

```
}

@Composable
private fun ManageDonationsContent(modifier: Modifier, state: ManageDonationsState, viewModel: ManageDonationsViewModel) {
    Box(modifier = modifier.fillMaxSize()) {
        when {
            state.isLoading -> {
                CircularProgressIndicator(modifier = Modifier.align(Alignment.Center))
            }
            state.error != null -> {
                Text(text = state.error, modifier = Modifier.align(Alignment.Center), color = MaterialTheme.colorScheme.error)
            }
            state.donations.isEmpty() -> {
                EmptyState()
            }
            else -> {
                LazyColumn(
                    modifier = Modifier.fillMaxSize(),
                    contentPadding = PaddingValues(16.dp),
                    verticalArrangement = Arrangement.spacedBy(16.dp)
                ) {
                    items(state.donations, key = { it.id }) { donation ->
                        DonationManagementItem(
                            donation = donation,
                            isUpdating = state.updatingDonationId == donation.id,
```

```
                    onConfirm = { viewModel.updateStatus(donation,
DonationStatus.COMPLETED) },
                    onReject = { viewModel.updateStatus(donation,
DonationStatus.FAILED) }
                )
            }
        }
    }
}

@Composable
private fun EmptyState() {
    Box(modifier = Modifier.fillMaxSize().padding(16.dp), contentAlignment =
Alignment.Center) {
        Column(horizontalAlignment = Alignment.CenterHorizontally, verticalArrangement =
Arrangement.spacedBy(16.dp)) {
            Icon(Icons.Default.Info, contentDescription = null, modifier = Modifier.size(48.dp),
tint = MaterialTheme.colorScheme.surfaceVariant)
            Text(
                text = "Este proyecto aún no ha recibido donaciones para gestionar.",
                style = MaterialTheme.typography.bodyLarge,
                textAlign = TextAlign.Center,
                color = MaterialTheme.colorScheme.outline
            )
        }
    }
}
```

```
@Composable
fun DonationManagementItem(donation: Donation, isUpdating: Boolean, onConfirm: () -> Unit, onReject: () -> Unit) {
    OutlinedCard(modifier = Modifier.fillMaxWidth()) {
        DonationItemHeader(donation)
        if (donation.proofImageUrl.isNotEmpty()) {
            DonationProofImage(donation.proofImageUrl)
        }
        DonationItemFooter(donation, isUpdating, onConfirm, onReject)
    }
}

@Composable
fun DonationItemHeader(donation: Donation) {
    val formattedDate = donation.createdAt?.let {
        SimpleDateFormat("dd MMM yyyy, HH:mm", Locale.getDefault()).format(it)
    } ?: "Fecha desconocida"
    Column(Modifier.padding(16.dp)) {
        Text(
            text = "$${donation.amount}",
            style = MaterialTheme.typography.headlineSmall,
            fontWeight = FontWeight.Bold
        )
        Text(
            text = formattedDate,
            style = MaterialTheme.typography.bodyMedium,
```

```
            color = MaterialTheme.colorScheme.onSurfaceVariant
        )
        if (donation.message.isNotBlank()) {
            Text(
                text = "\"${donation.message}\"",
                style = MaterialTheme.typography.bodyLarge,
                modifier = Modifier.padding(top = 8.dp)
            )
        }
    }
}

@Composable
fun DonationProofImage(imageUrl: String) {
    val context = LocalContext.current
    AsyncImage(
        model = imageUrl,
        contentDescription = "Comprobante de pago",
        modifier = Modifier
            .fillMaxWidth()
            .height(180.dp)
            .clickable {
                val intent = Intent(Intent.ACTION_VIEW, Uri.parse(imageUrl))
                context.startActivity(intent)
            },
        contentScale = ContentScale.Crop
    )
```

```
}

@Composable
fun DonationItemFooter(donation: Donation, isUpdating: Boolean, onConfirm: () -> Unit, onReject: () -> Unit) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp, vertical = 8.dp),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.SpaceBetween
    ) {
        DonationStatusBadge(status = donation.status)

        if (donation.status == DonationStatus.PENDING) {
            if (isUpdating) {
                CircularProgressIndicator(modifier = Modifier.size(36.dp))
            } else {
                Row(verticalAlignment = Alignment.CenterVertically) {
                    TextButton(onClick = onReject) {
                        Text("Rechazar")
                    }
                    Spacer(modifier = Modifier.width(8.dp))
                    Button(onClick = onConfirm) {
                        Text("Aprobar")
                    }
                }
```

```
            }
        }
    }
}


@Composable
fun DonationStatusBadge(status: DonationStatus) {
    val (text, color, icon) = when (status) {
        DonationStatus.PENDING -> Triple("Pendiente",
MaterialTheme.colorScheme.secondary, Icons.Default.HourglassEmpty)
        DonationStatus.COMPLETED -> Triple("Aprobada", Color(0xFF388E3C),
Icons.Default.CheckCircle)
        DonationStatus.FAILED -> Triple("Rechazada", MaterialTheme.colorScheme.error,
Icons.Default.Cancel)
    }

    Surface(
        shape = MaterialTheme.shapes.small,
        color = color.copy(alpha = 0.1f),
        contentColor = color,
        tonalElevation = 1.dp
    ) {
        Row(
            modifier = Modifier.padding(horizontal = 10.dp, vertical = 4.dp),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.spacedBy(6.dp)
        ) {
```

```
            Icon(icon, contentDescription = null, modifier = Modifier.size(16.dp))
            Text(text = text, style = MaterialTheme.typography.labelLarge, fontWeight = FontWeight.Bold)
        }
    }
}



@HiltViewModel
class ManageDonationsViewModel @Inject constructor(
    private val getDonationsForProjectUseCase: GetDonationsForProjectUseCase,
    private val updateDonationStatusUseCase: UpdateDonationStatusUseCase,
    savedStateHandle: SavedStateHandle
) : ViewModel() {

    private val _uiState: MutableStateFlow<ManageDonationsState> = MutableStateFlow(ManageDonationsState(isLoading = true))
    val uiState: StateFlow<ManageDonationsState> = _uiState.asStateFlow()

    private val projectId: String = savedStateHandle.get<String>("projectId")!!

    init {
        observeDonations()
    }

    private fun observeDonations() {
        getDonationsForProjectUseCase(projectId)
            .onEach { donations ->
```

```
                _uiState.value = _uiState.value.copy(
                    donations = donations,
                    isLoading = false
                )
            }
            .catch { e ->
                _uiState.value = _uiState.value.copy(
                    error = e.message,
                    isLoading = false
                )
            }
            .launchIn(viewModelScope)
    }

    fun updateStatus(donation: Donation, newStatus: DonationStatus) {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(updatingDonationId = donation.id)
            val result = updateDonationStatusUseCase(donation.id, donation.projectId,
donation.amount, newStatus)
            result.onFailure {
                _uiState.value = _uiState.value.copy(error = it.message)
            }
            // El Flow se encarga de quitar el ítem de la lista,
            // solo necesitamos quitar el indicador de carga.
            _uiState.value = _uiState.value.copy(updatingDonationId = null)
        }
    }
```

```
}

data class ManageDonationsState(
    val isLoading: Boolean = false,
    val donations: List<Donation> = emptyList(),
    val updatingDonationId: String? = null,
    val error: String? = null
)
```

Crea la carpeta profile

CreditsScreen

```
package com.hugoguerrero.tecno.ui.screens.profile

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.icons.Icons
```

```
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedCard
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import coil.compose.AsyncImage

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun CreditsScreen(navController: NavController) {
    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("Créditos") },
```

```
            navigationIcon = {
                IconButton(onClick = { navController.popBackStack() }) {
                    Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
                }
            }
        )
    }
) { padding ->
    LazyColumn(
        contentPadding = PaddingValues(16.dp),
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.spacedBy(24.dp)
    ) {
        item {
            Text("Una aplicación desarrollada por:", style =
MaterialTheme.typography.titleLarge)
        }
        item {
            CreditProfile(
                name = "Hugo Emiliano Guerrero Campos",
                bio = "Desarrollador de la aplicación Tecno-Ciencia, apasionado por la
tecnología y la innovación.
                Estudiante de Ingeniería en Desarrollo de Software Multiplataforma.",
                photoUrl = "https://firebasestorage.googleapis.com/v0/b/tecno-
ea191.firebasestorage.app/o/project
                _credits%2Fhugo.png?alt=media&token=b4f4d585-6a49-45c0-9cdc-
4068832833fe"
```

```
            )
        }
        item {
            CreditProfile(
                name = "Jorge Alberto Carranco Ramírez",
                bio = "Colaborador y tester de la aplicación. Estudiante de Ingeniero en Desarrollo de Software Multiplataforma.",
                photoUrl = "https://firebasestorage.googleapis.com/v0/b/tecno-ea191.firebasestorage.app/o/project_
                credits%2Fjorge.jpg?alt=media&token=b0c82b42-c5ee-4234-b8a2-bc2d81b9a1c8"
            )
        }
    }
  }
}


@Composable
private fun CreditProfile(name: String, bio: String, photoUrl: String) {
  OutlinedCard(modifier = Modifier.fillMaxWidth()) {
    Column(
      horizontalAlignment = Alignment.CenterHorizontally,
      modifier = Modifier.padding(16.dp)
    ) {
      AsyncImage(
        model = photoUrl,
        contentDescription = "Foto de $name",
```

```
            modifier =
Modifier.size(120.dp).clip(CircleShape).background(MaterialTheme.colorScheme.surfaceV
ariant),
            contentScale = ContentScale.Crop
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(name, style = MaterialTheme.typography.titleLarge, fontWeight =
FontWeight.Bold, textAlign = TextAlign.Center)
        Spacer(modifier = Modifier.height(8.dp))
        Text(bio, style = MaterialTheme.typography.bodyLarge, textAlign =
TextAlign.Center)
      }
   }
}
```

EditProfileScreen

```
package com.hugoguerrero.tecno.ui.screens.profile

import android.net.Uri
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
```

```
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.filled.PhotoCamera
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedCard
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SnackbarHost
import androidx.compose.material3.SnackbarHostState
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
```

```
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import coil.compose.AsyncImage
import kotlinx.coroutines.launch

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun EditProfileScreen(
    navController: NavController,
    viewModel: EditProfileViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()
    val snackbarHostState = remember { SnackbarHostState() }
    val scope = rememberCoroutineScope()

    val imagePickerLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.GetContent(),
        onResult = { uri: Uri? -> viewModel.onPhotoUriChange(uri) }
    )
```

```
LaunchedEffect(key1 = viewModel) {
    viewModel.navigationEvent.collect { event ->
        when (event) {
            is EditProfileNavigation.NavigateBack -> navController.popBackStack()
        }
    }
}

state.userMessage?.let { message ->
    LaunchedEffect(message) {
        scope.launch { snackbarHostState.showSnackbar(message) }
        viewModel.userMessageShown()
    }
}

Scaffold(
    snackbarHost = { SnackbarHost(snackbarHostState) },
    topBar = {
        TopAppBar(
            title = { Text("Editar Perfil") },
            navigationIcon = {
                IconButton(onClick = { navController.popBackStack() }) {
                    Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
                }
            }
        )
    }
```

```
    ) { padding ->
        EditProfileContent(
            modifier = Modifier.padding(padding),
            state = state,
            onDisplayNameChange = viewModel::onDisplayNameChange,
            onBioChange = viewModel::onBioChange,
            onPaymentDetailsChange = viewModel::onPaymentDetailsChange,
            onImagePickerClick = { imagePickerLauncher.launch("image/*") },
            onSaveClick = viewModel::saveProfile
        )
    }
}

@Composable
private fun EditProfileContent(
    modifier: Modifier,
    state: EditProfileState,
    onDisplayNameChange: (String) -> Unit,
    onBioChange: (String) -> Unit,
    onPaymentDetailsChange: (String) -> Unit,
    onImagePickerClick: () -> Unit,
    onSaveClick: () -> Unit
) {
    if (state.isInitialLoading) {
        Box(modifier = modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
            CircularProgressIndicator()
        }
```

```
    } else {
        Column(
            modifier =
modifier.fillMaxSize().verticalScroll(rememberScrollState()).padding(16.dp),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            ProfileImagePicker(state.photoUri, state.initialPhotoUrl, onImagePickerClick)
            Spacer(modifier = Modifier.height(24.dp))

            OutlinedCard(Modifier.fillMaxWidth()) {
                Column(Modifier.padding(16.dp)) {
                    OutlinedTextField(
                        value = state.displayName,
                        onValueChange = onDisplayNameChange,
                        label = { Text("Nombre") },
                        modifier = Modifier.fillMaxWidth()
                    )
                    Spacer(modifier = Modifier.height(16.dp))
                    OutlinedTextField(
                        value = state.bio,
                        onValueChange = onBioChange,
                        label = { Text("Biografía") },
                        modifier = Modifier.fillMaxWidth().height(120.dp),
                        maxLines = 5
                    )
                    Spacer(modifier = Modifier.height(16.dp))
                    OutlinedTextField(
```

```
                value = state.paymentDetails,
                onValueChange = onPaymentDetailsChange,
                label = { Text("Datos de Pago (CLABE, CBU, etc.)") },
                modifier = Modifier.fillMaxWidth()
            )
        }
    }

    Spacer(modifier = Modifier.height(32.dp))

    Button(
        onClick = onSaveClick,
        modifier = Modifier.fillMaxWidth(),
        enabled = !state.isLoading
    ) {
        if (state.isLoading) {
            CircularProgressIndicator(modifier = Modifier.size(20.dp), strokeWidth =
2.dp)
        } else {
            Text("Guardar Cambios")
        }
    }
  }
}

@Composable
```

```
private fun ProfileImagePicker(selectedUri: Uri?, initialUrl: String?, onClick: () -> Unit) {
    Box(contentAlignment = Alignment.BottomEnd, modifier = Modifier.clickable(onClick = onClick)) {
        AsyncImage(
            model = selectedUri ?: initialUrl,
            contentDescription = "Foto de perfil",
            modifier = Modifier.size(140.dp).clip(CircleShape).background(MaterialTheme.colorScheme.surfaceVariant),
            contentScale = ContentScale.Crop
        )
        Icon(
            imageVector = Icons.Default.PhotoCamera,
            contentDescription = "Cambiar foto",
            modifier = Modifier.background(MaterialTheme.colorScheme.primary, CircleShape).padding(8.dp),
            tint = MaterialTheme.colorScheme.onPrimary
        )
    }
}
```

EditProfileViewModel

```
package com.hugoguerrero.tecno.ui.screens.profile

import android.net.Uri
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
```

```
import com.hugoguerrero.tecno.domain.use_case.GetCurrentUserUseCase
import com.hugoguerrero.tecno.domain.use_case.GetUserProfileUseCase
import com.hugoguerrero.tecno.domain.use_case.UpdateUserProfileUseCase
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

sealed class EditProfileNavigation {
    object NavigateBack : EditProfileNavigation()
}

data class EditProfileState(
    val displayName: String = "",
    val bio: String = "",
    val paymentDetails: String = "",
    val photoUri: Uri? = null,
    val initialPhotoUrl: String? = null,
    val isLoading: Boolean = false,
    val isInitialLoading: Boolean = true,
    val userMessage: String? = null
)

@HiltViewModel
```

```
class EditProfileViewModel @Inject constructor(
    private val getUserProfileUseCase: GetUserProfileUseCase,
    private val updateUserProfileUseCase: UpdateUserProfileUseCase,
    private val getCurrentUserUseCase: GetCurrentUserUseCase
) : ViewModel() {

    private val _uiState = MutableStateFlow(EditProfileState())
    val uiState = _uiState.asStateFlow()

    private val _navigationEvent = MutableSharedFlow<EditProfileNavigation>()
    val navigationEvent = _navigationEvent.asSharedFlow()

    init {
        loadUserProfile()
    }

    private fun loadUserProfile() {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(isInitialLoading = true)
            val user = getCurrentUserUseCase()
            if (user != null) {
                val profile = getUserProfileUseCase(user.uid)
                if (profile != null) {
                    _uiState.value = _uiState.value.copy(
                        displayName = profile.displayName,
                        bio = profile.bio ?: "",
                        paymentDetails = profile.paymentDetails ?: "",
```

```
                    initialPhotoUrl = profile.photoUrl,
                    isInitialLoading = false
                )
            } else {
                _uiState.value = _uiState.value.copy(isInitialLoading = false, userMessage = "No se pudo cargar el perfil.")
            }
        } else {
            _uiState.value = _uiState.value.copy(isInitialLoading = false, userMessage = "Error de autenticación.")
        }
    }
}

fun onDisplayNameChange(name: String) {
    _uiState.value = _uiState.value.copy(displayName = name)
}

fun onBioChange(bio: String) {
    _uiState.value = _uiState.value.copy(bio = bio)
}

fun onPaymentDetailsChange(details: String) {
    _uiState.value = _uiState.value.copy(paymentDetails = details)
}

fun onPhotoUriChange(uri: Uri?) {
    _uiState.value = _uiState.value.copy(photoUri = uri)
```

```
    }

    fun saveProfile() {
        viewModelScope.launch {
            val currentUser = getCurrentUserUseCase()
            if (currentUser == null) {
                _uiState.value = _uiState.value.copy(userMessage = "Error de autenticación. No se pudo guardar el perfil.")
                return@launch
            }

            _uiState.value = _uiState.value.copy(isLoading = true)
            val result = updateUserProfileUseCase(
                uid = currentUser.uid,
                displayName = _uiState.value.displayName,
                bio = _uiState.value.bio,
                paymentDetails = _uiState.value.paymentDetails,
                photoUri = _uiState.value.photoUri
            )
            result.onSuccess {
                _uiState.value = _uiState.value.copy(isLoading = false, userMessage = "Perfil actualizado correctamente")
                _navigationEvent.emit(EditProfileNavigation.NavigateBack)
            }.onFailure {
                _uiState.value = _uiState.value.copy(isLoading = false, userMessage = it.message)
            }
        }
```

```
    }

    fun userMessageShown() {
        _uiState.value = _uiState.value.copy(userMessage = null)
    }
}
```

MyProfileScreen

```
package com.hugoguerrero.tecno.ui.screens.profile

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.CircleShape
```

```
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.Edit
import androidx.compose.material.icons.filled.Logout
import androidx.compose.material.icons.filled.MonetizationOn
import androidx.compose.material3.AlertDialog
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Card
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.Divider
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.LinearProgressIndicator
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
```

```
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.navigation.NavController
import coil.compose.AsyncImage
import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.data.model.UserProfile
import com.hugoguerrero.tecno.ui.components.BottomNavigationBar
import com.hugoguerrero.tecno.ui.navigation.AUTH_GRAPH_ROUTE
import com.hugoguerrero.tecno.ui.navigation.Screens
import com.hugoguerrero.tecno.ui.screens.login.NavigationEvent
import kotlinx.coroutines.flow.collectLatest
import java.text.SimpleDateFormat
import java.util.Locale

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MyProfileScreen(
    navController: NavController,
    viewModel: MyProfileViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()

    LaunchedEffect(key1 = true) {
        viewModel.navigationEvent.collectLatest { event ->
            when (event) {
```

```
            is NavigationEvent.NavigateToLogin -> {
                navController.navigate(AUTH_GRAPH_ROUTE) {
                    popUpTo(navController.graph.id) { inclusive = true }
                }
            }
            else -> {}
        }
    }
}

Scaffold(
    topBar = {
        TopAppBar(
            title = { Text("Mi Perfil") },
            actions = {
                IconButton(onClick = { navController.navigate(Screens.EditProfile.route) }) {
                    Icon(Icons.Default.Edit, contentDescription = "Editar Perfil")
                }
                IconButton(onClick = { viewModel.signOut() }) {
                    Icon(Icons.Default.Logout, contentDescription = "Cerrar sesión")
                }
            }
        )
    },
    bottomBar = { BottomNavigationBar(navController) }
) { padding ->
```

```
        MyProfileContent(modifier = Modifier.padding(padding), state = state, navController = navController, viewModel = viewModel)
    }
}

@Composable
private fun MyProfileContent(modifier: Modifier, state: MyProfileState, navController: NavController, viewModel: MyProfileViewModel) {
    Box(modifier = modifier.fillMaxSize()) {
        if (state.isLoading) {
            CircularProgressIndicator(modifier = Modifier.align(Alignment.Center))
        } else if (state.error != null) {
            Text(text = state.error, modifier = Modifier.align(Alignment.Center), color = MaterialTheme.colorScheme.error)
        } else if (state.profile != null) {
            LazyColumn(contentPadding = PaddingValues(vertical = 16.dp)) {
                item {
                    ProfileHeader(profile = state.profile)
                    Spacer(modifier = Modifier.height(24.dp))
                    Text("Mis Proyectos", style = MaterialTheme.typography.titleLarge, modifier = Modifier.padding(horizontal = 16.dp))
                    Divider(modifier = Modifier.padding(vertical = 8.dp))
                }
                if (state.projects.isEmpty()) {
                    item {
                        Text(
                            "Aún no has creado ningún proyecto.",
                            style = MaterialTheme.typography.bodyMedium,
```

```
                    modifier = Modifier.padding(16.dp)
                )
            }
        } else {
            items(state.projects) { project ->
                ProjectItem(
                    project = project,
                    onUpdate = { navController.navigate(Screens.EditProject.route +
"/${project.id}") },
                    onDelete = { viewModel.onDeleteProjectClicked(project) },
                    onManage = { navController.navigate(Screens.ManageDonations.route +
"/${project.id}") }
                )
            }
        }
    }

    if (state.projectToDelete != null) {
        DeleteConfirmationDialog(state, viewModel)
    }
  }
}


@Composable
private fun ProfileHeader(profile: UserProfile) {
   Column(horizontalAlignment = Alignment.CenterHorizontally, modifier =
Modifier.fillMaxWidth().padding(horizontal = 16.dp)) {
```

```
        AsyncImage(
            model = profile.photoUrl.ifEmpty { "https://via.placeholder.com/150" },
            contentDescription = "Foto de perfil",
            modifier =
Modifier.size(120.dp).clip(CircleShape).background(MaterialTheme.colorScheme.surfaceV
ariant),
            contentScale = ContentScale.Crop
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(profile.displayName, style = MaterialTheme.typography.headlineSmall,
fontWeight = FontWeight.Bold)
        Text(profile.email, style = MaterialTheme.typography.bodyMedium)
        profile.createdAt?.let {
            val formattedDate = SimpleDateFormat("dd MMM yyyy",
Locale.getDefault()).format(it)
            Text("Miembro desde: $formattedDate", style =
MaterialTheme.typography.bodySmall, color =
MaterialTheme.colorScheme.onSurfaceVariant)
        }
    }
}

@Composable
private fun ProjectItem(project: Project, onUpdate: () -> Unit, onDelete: () -> Unit,
onManage: () -> Unit) {
    Card(modifier = Modifier.fillMaxWidth().padding(horizontal = 16.dp, vertical = 8.dp)) {
        Column {
            AsyncImage(
                model = project.imageUrl,
```

```
            contentDescription = "Imagen de ${project.title}",
            modifier = Modifier.fillMaxWidth().height(150.dp),
            contentScale = ContentScale.Crop
        )
        Column(modifier = Modifier.padding(16.dp)) {
            Text(project.title, style = MaterialTheme.typography.titleMedium, fontWeight =
FontWeight.Bold)
            Text("Meta: $${project.fundingGoal.toInt()} - Recaudado:
$${project.currentFunding.toInt()}", style = MaterialTheme.typography.bodySmall)
            LinearProgressIndicator(progress = project.progress / 100f, modifier =
Modifier.fillMaxWidth().padding(vertical = 8.dp))
        }
        Row(modifier = Modifier.fillMaxWidth().padding(horizontal = 8.dp),
horizontalArrangement = Arrangement.End) {
            TextButton(onClick = onManage) {
                Text("Gestionar")
            }
            IconButton(onClick = onUpdate) {
                Icon(Icons.Default.Edit, contentDescription = "Editar Proyecto")
            }
            IconButton(onClick = onDelete) {
                Icon(Icons.Default.Delete, contentDescription = "Eliminar Proyecto")
            }
        }
    }
  }
}
```

```
@Composable
private fun DeleteConfirmationDialog(state: MyProfileState, viewModel: MyProfileViewModel) {
    AlertDialog(
        onDismissRequest = { viewModel.cancelProjectDeletion() },
        title = { Text("Confirmar Eliminación") },
        text = { Text("¿Estás seguro de que quieres eliminar el proyecto '${state.projectToDelete?.title}'? Esta acción no se puede deshacer.") },
        confirmButton = {
            Button(
                onClick = { viewModel.confirmProjectDeletion() },
                colors = ButtonDefaults.buttonColors(containerColor = MaterialTheme.colorScheme.error)
            ) {
                Text("Eliminar")
            }
        },
        dismissButton = {
            Button(onClick = { viewModel.cancelProjectDeletion() }) {
                Text("Cancelar")
            }
        }
    )
}
```

MyProfileViewModel

```
package com.hugoguerrero.tecno.ui.screens.profile
```

```
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.data.model.UserProfile
import com.hugoguerrero.tecno.domain.use_case.DeleteProjectUseCase
import com.hugoguerrero.tecno.domain.use_case.GetCurrentUserUseCase
import com.hugoguerrero.tecno.domain.use_case.GetProjectsByOwnerUseCase
import com.hugoguerrero.tecno.domain.use_case.GetUserProfileUseCase
import com.hugoguerrero.tecno.domain.use_case.SignOutUseCase
import com.hugoguerrero.tecno.ui.screens.login.NavigationEvent
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class MyProfileViewModel @Inject constructor(
    private val getCurrentUserUseCase: GetCurrentUserUseCase,
    private val getUserProfileUseCase: GetUserProfileUseCase,
    private val getProjectsByOwnerUseCase: GetProjectsByOwnerUseCase,
    private val deleteProjectUseCase: DeleteProjectUseCase,
    private val signOutUseCase: SignOutUseCase
) : ViewModel() {
```

```
    private val _uiState = MutableStateFlow(MyProfileState(isLoading = true))
    val uiState = _uiState.asStateFlow()

    private val _navigationEvent = MutableSharedFlow<NavigationEvent>()
    val navigationEvent = _navigationEvent.asSharedFlow()

    init {
        loadProfile()
    }

    private fun loadProfile() {
        viewModelScope.launch {
            val user = getCurrentUserUseCase()
            if (user != null) {
                val profile = getUserProfileUseCase(user.uid)
                _uiState.value = _uiState.value.copy(profile = profile)
                profile?.let {
                    getProjectsByOwnerUseCase(it.uid).collect { projects ->
                        _uiState.value = _uiState.value.copy(projects = projects, isLoading = false)
                    }
                } ?: run {
                    _uiState.value = _uiState.value.copy(isLoading = false, error = "No se pudo
cargar el perfil.")
                }
            } else {
                _uiState.value = _uiState.value.copy(isLoading = false, error = "Error de
autenticación.")
```

```
        }
    }
}

fun onDeleteProjectClicked(project: Project) {
    _uiState.value = _uiState.value.copy(projectToDelete = project)
}

fun confirmProjectDeletion() {
    viewModelScope.launch {
        _uiState.value.projectToDelete?.let {
            deleteProjectUseCase(it.id)
            _uiState.value = _uiState.value.copy(projectToDelete = null)
        }
    }
}

fun cancelProjectDeletion() {
    _uiState.value = _uiState.value.copy(projectToDelete = null)
}

fun signOut() {
    viewModelScope.launch {
        signOutUseCase()
        _navigationEvent.emit(NavigationEvent.NavigateToLogin)
    }
}
```

```
}

data class MyProfileState(
    val profile: UserProfile? = null,
    val projects: List<Project> = emptyList(),
    val isLoading: Boolean = false,
    val error: String? = null,
    val projectToDelete: Project? = null
)
```

PublicProfileScreen

```
package com.hugoguerrero.tecno.ui.screens.profile

import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
```

```
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material3.Card
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.Divider
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
```

```
import androidx.navigation.NavController
import coil.compose.AsyncImage
import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.data.model.UserProfile
import com.hugoguerrero.tecno.domain.use_case.GetProjectsByOwnerUseCase
import com.hugoguerrero.tecno.domain.use_case.GetUserProfileUseCase
import com.hugoguerrero.tecno.ui.components.ProjectCard
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.catch
import kotlinx.coroutines.flow.launchIn
import kotlinx.coroutines.flow.onEach
import kotlinx.coroutines.launch
import javax.inject.Inject

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun PublicProfileScreen(
    navController: NavController,
    viewModel: PublicProfileViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()

    Scaffold(
        topBar = {
```

```
            TopAppBar(
                title = { Text(state.userProfile?.displayName ?: "") },
                navigationIcon = {
                    IconButton(onClick = { navController.popBackStack() }) {
                        Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
                    }
                }
            )
        }
    ) { padding ->
        PublicProfileContent(modifier = Modifier.padding(padding), state = state,
navController = navController)
    }
}

@Composable
private fun PublicProfileContent(modifier: Modifier, state: PublicProfileState,
navController: NavController) {
    Box(modifier = modifier.fillMaxSize()) {
        if (state.isLoading) {
            CircularProgressIndicator(modifier = Modifier.align(Alignment.Center))
        } else if (state.error != null) {
            Text(state.error, modifier = Modifier.align(Alignment.Center), color =
MaterialTheme.colorScheme.error)
        } else if (state.userProfile != null) {
            LazyColumn(contentPadding = PaddingValues(vertical = 16.dp)) {
                item {
                    PublicProfileHeader(profile = state.userProfile)
```

```
                Spacer(modifier = Modifier.height(24.dp))
                Text("Proyectos de ${state.userProfile.displayName}", style =
MaterialTheme.typography.titleLarge,
                 modifier = Modifier.padding(horizontal = 16.dp))
                Divider(modifier = Modifier.padding(vertical = 8.dp))
            }
            if (state.projects.isEmpty()) {
                item {
                    Text(
                        "Este usuario aún no ha creado ningún proyecto.",
                        style = MaterialTheme.typography.bodyMedium,
                        modifier = Modifier.padding(16.dp)
                    )
                }
            } else {
                items(state.projects) { project ->
                    ProjectCard(project = project, onClick = { /* Lógica de navegación */ })
                }
            }
        }
    }
}

@Composable
private fun PublicProfileHeader(profile: UserProfile) {
    Column(horizontalAlignment = Alignment.CenterHorizontally, modifier =
Modifier.fillMaxWidth().padding(horizontal = 16.dp)) {
```

```
        AsyncImage(
            model = profile.photoUrl.ifEmpty { "https://via.placeholder.com/150" },
            contentDescription = "Foto de perfil",
            modifier =
Modifier.size(120.dp).clip(CircleShape).background(MaterialTheme.colorScheme.surfaceV
ariant),
            contentScale = ContentScale.Crop
        )
        Spacer(modifier = Modifier.height(16.dp))
        Text(profile.displayName, style = MaterialTheme.typography.headlineSmall,
fontWeight = FontWeight.Bold)
        Text(profile.email, style = MaterialTheme.typography.bodyMedium)
        profile.bio?.let {
            Text(it, style = MaterialTheme.typography.bodyMedium, modifier =
Modifier.padding(top = 8.dp))
        }
    }
}


@HiltViewModel
class PublicProfileViewModel @Inject constructor(
    private val getUserProfileUseCase: GetUserProfileUseCase,
    private val getProjectsByOwnerUseCase: GetProjectsByOwnerUseCase,
    savedStateHandle: SavedStateHandle
) : ViewModel() {

    private val _uiState = MutableStateFlow(PublicProfileState(isLoading = true))
    val uiState: StateFlow<PublicProfileState> = _uiState.asStateFlow()
```

```
    init {
        savedStateHandle.get<String>("userId")?.let { userId ->
            loadProfile(userId)
        } ?: run {
            _uiState.value = PublicProfileState(error = "ID de usuario no proporcionado")
        }
    }

    private fun loadProfile(userId: String) {
        viewModelScope.launch {
            val userProfile = getUserProfileUseCase(userId)
            if (userProfile != null) {
                _uiState.value = _uiState.value.copy(userProfile = userProfile)
                getProjectsByOwnerUseCase(userId).onEach { projects ->
                    _uiState.value = _uiState.value.copy(projects = projects, isLoading = false)
                }.catch { e ->
                    _uiState.value = _uiState.value.copy(error = e.message, isLoading = false)
                }.launchIn(viewModelScope)
            } else {
                _uiState.value = PublicProfileState(error = "No se pudo cargar el perfil del
usuario")
            }
        }
    }
}
```

```
data class PublicProfileState(
    val userProfile: UserProfile? = null,
    val projects: List<Project> = emptyList(),
    val isLoading: Boolean = false,
    val error: String? = null
)
```

UserProfileScreen

```
package com.hugoguerrero.tecno.ui.screens.profile

import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.CircleShape
```

```
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.LinearProgressIndicator
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
```

```
import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.data.model.UserProfile
import com.hugoguerrero.tecno.data.model.UserStats
import com.hugoguerrero.tecno.domain.model.UserType
import com.hugoguerrero.tecno.domain.use_case.*
import com.hugoguerrero.tecno.ui.navigation.AUTH_GRAPH_ROUTE
import com.hugoguerrero.tecno.ui.navigation.Screens
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.catch
import kotlinx.coroutines.flow.launchIn
import kotlinx.coroutines.flow.onEach
import kotlinx.coroutines.launch
import javax.inject.Inject

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun UserProfileScreen(
    navController: NavController,
    viewModel: UserProfileViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()
```

```
    LaunchedEffect(Unit) {
        viewModel.navigationEvent.collect { event ->
            when (event) {
                is NavigationEvent.NavigateToAuth -> {
                    navController.navigate(AUTH_GRAPH_ROUTE) {
                        popUpTo(navController.graph.id) { inclusive = true }
                    }
                }
            }
        }
    }

    Scaffold(
        topBar = {
            TopAppBar(
                title = {
                    Row(
                        modifier = Modifier.fillMaxWidth(),
                        horizontalArrangement = Arrangement.SpaceBetween,
                        verticalAlignment = Alignment.CenterVertically
                    ) {
                        Text("Mi Perfil", color = Color.White, fontWeight = FontWeight.Bold,
fontSize = 20.sp)
                        Text("Tecno-Ciencia", color = Color.White, fontWeight =
FontWeight.SemiBold, fontSize = 20.sp)
                    }
                },
```

```
            colors = TopAppBarDefaults.topAppBarColors(containerColor =
Color(0xFFB30059))
        )
    },
    containerColor = Color(0xFF0F0015)
) { paddingValues ->
    Column(
        modifier =
Modifier.fillMaxSize().padding(paddingValues).background(Color(0xFF0F0015))
    ) {
        if (state.isLoading) {
            Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
                CircularProgressIndicator(color = Color(0xFFFF66CC))
            }
        } else {
            state.userProfile?.let { user ->
                Column(modifier = Modifier.padding(16.dp), horizontalAlignment =
Alignment.CenterHorizontally) {
                    Box(
                        modifier =
Modifier.size(120.dp).clip(CircleShape).background(Color(0xFF9F0D2A)),
                        contentAlignment = Alignment.Center
                    ) {
                        Text(user.displayName.take(2).uppercase(), color = Color.White, fontSize
= 32.sp, fontWeight = FontWeight.Bold)
                    }
                    Spacer(modifier = Modifier.height(12.dp))
                    Text(user.displayName, color = Color(0xFFFF66CC), fontWeight =
FontWeight.Bold, fontSize = 22.sp)
```

```
                Text(
                    when (user.userType) {
                        UserType.STUDENT -> "👨‍🎓 Estudiante"
                        UserType.MANAGER -> "👨‍💼 Manager"
                        UserType.SPONSOR -> "💰 Patrocinador"
                        else -> "👤 Usuario"
                    },
                    color = Color.LightGray, fontSize = 14.sp
                )
                Spacer(modifier = Modifier.height(16.dp))
                Row(
                    modifier = Modifier.fillMaxWidth().padding(horizontal = 32.dp),
                    horizontalArrangement = Arrangement.SpaceEvenly
                ) {
                    StatItem(value = state.userProjects.size.toString(), label = "Proyectos",
color = Color(0xFFFF66CC))
                    StatItem(value = state.userStats.donationsCount.toString(), label =
"Donaciones", color = Color(0xFFFF66CC))
                    StatItem(value = "${state.userStats.successRate}%", label = "Éxito",
color = Color(0xFFFF66CC))
                }
            }

            Button(
                onClick = { viewModel.signOut() },
                modifier = Modifier.fillMaxWidth().padding(horizontal = 16.dp, vertical =
8.dp)
            ) {
```

```
                Text("Cerrar Sesión")
            }

            state.error?.let {
                Text(text = it, color = Color.Red, modifier = Modifier.padding(horizontal = 16.dp), textAlign = TextAlign.Center)
            }

            PostsSection(
                projects = state.userProjects,
                onProjectClick = { projectId ->
                    navController.navigate(Screens.ProjectDetail.route + "/$projectId")
                }
            )

        } ?: Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
            Text("No se pudo cargar el perfil.", color = Color.Gray)
        }
    }
  }
}

@HiltViewModel
class UserProfileViewModel @Inject constructor(
  private val getCurrentUserUseCase: GetCurrentUserUseCase,
  private val getUserProfileUseCase: GetUserProfileUseCase,
```

```
    private val getUserProjectsUseCase: GetProjectsByOwnerUseCase,
    private val getUserStatsUseCase: GetUserStatsUseCase,
    private val signOutUseCase: SignOutUseCase
) : ViewModel() {

    private val _uiState = MutableStateFlow(UserProfileState())
    val uiState: StateFlow<UserProfileState> = _uiState.asStateFlow()

    private val _navigationEvent = MutableSharedFlow<NavigationEvent>()
    val navigationEvent = _navigationEvent.asSharedFlow()

    init {
        loadUserData()
    }

    private fun loadUserData() {
        viewModelScope.launch {
            _uiState.value = UserProfileState(isLoading = true)
            val currentUserId = getCurrentUserUseCase()?.uid

            if (currentUserId == null) {
                _uiState.value = UserProfileState(error = "No se ha iniciado sesión.")
                return@launch
            }

            val userProfile = getUserProfileUseCase(currentUserId)
            if (userProfile == null) {
```

```
            _uiState.value = UserProfileState(error = "No se pudo cargar el perfil.")
            return@launch
        }
        _uiState.value = _uiState.value.copy(userProfile = userProfile)

        getUserProjectsUseCase(currentUserId).onEach { projects ->
            _uiState.value = _uiState.value.copy(userProjects = projects)
        }.catch { e ->
            _uiState.value = _uiState.value.copy(error = e.message)
        }.launchIn(viewModelScope)

        getUserStatsUseCase(currentUserId).onEach { result ->
            result.onSuccess {
                _uiState.value = _uiState.value.copy(userStats = it)
            }
        }.launchIn(viewModelScope)

        _uiState.value = _uiState.value.copy(isLoading = false)
    }
}

fun signOut() {
    viewModelScope.launch {
        signOutUseCase()
        _navigationEvent.emit(NavigationEvent.NavigateToAuth)
    }
}
```

```
}

data class UserProfileState(
    val userProfile: UserProfile? = null,
    val userProjects: List<Project> = emptyList(),
    val userStats: UserStats = UserStats(),
    val isLoading: Boolean = false,
    val error: String? = null
)

sealed class NavigationEvent {
    object NavigateToAuth : NavigationEvent()
}

@Composable
fun StatItem(value: String, label: String, color: Color) {
    Column(horizontalAlignment = Alignment.CenterHorizontally) {
        Text(value, color = color, fontWeight = FontWeight.Bold, fontSize = 18.sp)
        Text(label, color = Color.LightGray, fontSize = 12.sp)
    }
}

@Composable
fun PostsSection(projects: List<Project>, onProjectClick: (String) -> Unit) {
    if (projects.isEmpty()) {
        Box(modifier = Modifier.fillMaxSize().padding(32.dp), contentAlignment =
Alignment.Center) {
```

```
            Column(horizontalAlignment = Alignment.CenterHorizontally) {
                Text("📝", fontSize = 48.sp, modifier = Modifier.padding(bottom = 16.dp))
                Text("Aún no tienes proyectos", color = Color(0xFFFF66CC), fontWeight = FontWeight.Bold, fontSize = 18.sp)
                Text("Crea tu primer proyecto para comenzar", color = Color.LightGray, fontSize = 14.sp, textAlign = TextAlign.Center, modifier = Modifier.padding(top = 8.dp))
            }
        }
    } else {
        LazyColumn(
            modifier = Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.spacedBy(8.dp),
            contentPadding = PaddingValues(bottom = 16.dp)
        ) {
            items(projects) { project ->
                ProjectPostItem(project = project, onClick = { onProjectClick(project.id) })
            }
        }
    }
}

@Composable
fun ProjectPostItem(project: Project, onClick: () -> Unit) {
    Card(
        modifier = Modifier.fillMaxWidth().padding(horizontal = 16.dp, vertical = 8.dp).clickable { onClick() },
        shape = RoundedCornerShape(16.dp),
        colors = CardDefaults.cardColors(containerColor = Color(0xFF1A001F)),
```

```
        elevation = CardDefaults.cardElevation(8.dp)
    ) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text(text = project.title, color = Color(0xFFFF66CC), fontWeight =
FontWeight.Bold, fontSize = 18.sp)
            Spacer(modifier = Modifier.height(6.dp))
            Text(text = project.description, color = Color(0xFFD1B2FF), fontSize = 14.sp,
maxLines = 2)
            Spacer(modifier = Modifier.height(12.dp))
            LinearProgressIndicator(
                progress = { project.progress / 100f },
                color = Color(0xFFB30059),
                trackColor = Color(0xFF4B0054),
                modifier = Modifier.fillMaxWidth().height(8.dp)
            )
            Spacer(modifier = Modifier.height(8.dp))
            Row(modifier = Modifier.fillMaxWidth(), horizontalArrangement =
Arrangement.SpaceBetween) {
                Text("${project.progress.toInt()}% completado", color = Color(0xFFD1B2FF),
fontSize = 12.sp)
                Text("$${project.currentFunding.toInt()} / $${project.fundingGoal.toInt()}",
color = Color(0xFFD1B2FF), fontSize = 12.sp)
            }
            Spacer(modifier = Modifier.height(8.dp))
            Row(horizontalArrangement = Arrangement.spacedBy(8.dp)) {
                project.tags.take(2).forEach { tag ->
                    Box(
```

```
                modifier =
Modifier.clip(RoundedCornerShape(8.dp)).background(Color(0xFF330033)).padding(horiz
ontal = 8.dp, vertical = 4.dp)
            ) {
                Text(text = tag, color = Color(0xFFFF66CC), fontSize = 10.sp)
            }
        }
    }
  }
}
```

Crea la carpeta project

ProfileCreditos

```
package com.hugoguerrero.tecno.ui.screens.project


import android.R
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
```

```
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.material3.TopAppBarDefaults
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

@Preview(showBackground = true)
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ProfileCreditos() {
    Scaffold(
        topBar = {
            TopAppBar(
                title = {
```

```
            Text(
                "Créditos",
                color = Color.White,
                fontWeight = FontWeight.Bold,
                fontSize = 20.sp
            )
        },
        navigationIcon = {
            IconButton(onClick = { /* sin funcionalidad */ }) {
                Icon(
                    painter = painterResource(R.drawable.ic_media_previous),
                    contentDescription = "Atrás",
                    tint = Color.White
                )
            }
        },
        colors = TopAppBarDefaults.topAppBarColors(
            containerColor = Color(0xFF0D1117)
        )
    )
},
containerColor = Color(0xFF0D1117)
) { paddingValues ->

Column(
    modifier = Modifier
        .fillMaxSize()
```

```
        .padding(paddingValues)
        .verticalScroll(rememberScrollState())
        .padding(16.dp),
    verticalArrangement = Arrangement.spacedBy(12.dp)
) {

    // Cabecera
    Column {
        Text(
            "Tecno-Ciencia",
            color = Color.White,
            fontWeight = FontWeight.Bold,
            fontSize = 20.sp
        )
        Text(
            "Conectando estudiantes, managers y patrocinadores",
            color = Color(0xFF9DA8B8),
            fontSize = 13.sp
        )
    }

    // Equipo del proyecto
    Card(
        colors = CardDefaults.cardColors(containerColor = Color(0xFF161B22)),
        shape = RoundedCornerShape(12.dp),
        modifier = Modifier.fillMaxWidth()
    ) {
```

```
            Column(modifier = Modifier.padding(16.dp)) {
                Text(
                    "Equipo del proyecto",
                    color = Color.White,
                    fontWeight = FontWeight.SemiBold,
                    fontSize = 16.sp
                )
                Spacer(modifier = Modifier.height(8.dp))
                Text("Hugo Guerrero — Producto y dirección • Líder", color =
Color(0xFFB4B4B4))
                Text("Jorge — Diseño UI/UX • Diseño", color = Color(0xFFB4B4B4))
                Text("Natalia Estefania — iOS & Android • Mobile", color =
Color(0xFFB4B4B4))
                Text("Rafa — Backend & API • Ingeniería", color = Color(0xFFB4B4B4))
            }
        }

        //  Colaboradores
        Card(
            colors = CardDefaults.cardColors(containerColor = Color(0xFF161B22)),
            shape = RoundedCornerShape(12.dp),
            modifier = Modifier.fillMaxWidth()
        ) {
            Column(modifier = Modifier.padding(16.dp)) {
                Text(
                    "Colaboradores",
                    color = Color.White,
                    fontWeight = FontWeight.SemiBold,
```

```
                fontSize = 16.sp
            )
            Spacer(modifier = Modifier.height(8.dp))
            Text("Alex — QA y testing • QA", color = Color(0xFFB4B4B4))
            Text("Emx — Comunidad • Comunidad", color = Color(0xFFB4B4B4))
        }
    }

    // Agradecimientos especiales
    Card(
        colors = CardDefaults.cardColors(containerColor = Color(0xFF161B22)),
        shape = RoundedCornerShape(12.dp),
        modifier = Modifier.fillMaxWidth()
    ) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text(
                "Agradecimientos especiales",
                color = Color.White,
                fontWeight = FontWeight.SemiBold,
                fontSize = 16.sp
            )
            Spacer(modifier = Modifier.height(8.dp))
            Text(
                "Gracias a nuestra comunidad de estudiantes, managers y patrocinadores por su apoyo y retroalimentación constante.",
                color = Color(0xFF9DA8B8),
                fontSize = 13.sp
```

```
            )
        }
    }

    // Tecnologías y recursos
    Card(
        colors = CardDefaults.cardColors(containerColor = Color(0xFF161B22)),
        shape = RoundedCornerShape(12.dp),
        modifier = Modifier.fillMaxWidth()
    ) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text(
                "Tecnologías y recursos",
                color = Color.White,
                fontWeight = FontWeight.SemiBold,
                fontSize = 16.sp
            )
            Spacer(modifier = Modifier.height(8.dp))
            Text("• Arquitectura: REST API, Realtime", color = Color(0xFFB4B4B4))
            Text("• Mobile: Swift, Kotlin, React Native", color = Color(0xFFB4B4B4))
            Text("• Backend: Node.js, PostgreSQL", color = Color(0xFFB4B4B4))
            Text("• Diseño: Figma, Lucide Icons", color = Color(0xFFB4B4B4))
            Spacer(modifier = Modifier.height(8.dp))
            Text(
                "Esta app incluye logotipos e imágenes que pertenecen a sus respectivos propietarios y se utilizan únicamente con fines de referencia.",
                color = Color(0xFF7C8592),
```

```
                    fontSize = 12.sp
                )
            }
        }

        Spacer(modifier = Modifier.height(16.dp))

        // Pie de página
        Text(
            "© 2025 Tecno-Ciencia. Todos los derechos reservados.",
            color = Color(0xFF6E7681),
            fontSize = 12.sp,
            modifier = Modifier.fillMaxWidth(),
            textAlign = TextAlign.Center
        )
    }
  }
}
```

EditProjectScreen

```
package com.hugoguerrero.tecno.ui.screens.project

import android.net.Uri
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.layout.Arrangement
```

```
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.filled.AttachFile
import androidx.compose.material.icons.filled.CloudUpload
import androidx.compose.material.icons.filled.Image
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.ExposedDropdownMenuBox
import androidx.compose.material3.ExposedDropdownMenuDefaults
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedButton
import androidx.compose.material3.OutlinedCard
import androidx.compose.material3.OutlinedTextField
```

```
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SnackbarHost
import androidx.compose.material3.SnackbarHostState
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
import coil.compose.AsyncImage
import com.hugoguerrero.tecno.domain.use_case.GetProjectByIdUseCase
import com.hugoguerrero.tecno.domain.use_case.UpdateProjectUseCase
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
```

```
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun EditProjectScreen(
    navController: NavController,
    viewModel: EditProjectViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()
    val snackbarHostState = remember { SnackbarHostState() }

    val imagePickerLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.GetContent(),
        onResult = { uri -> viewModel.onImageSelected(uri) }
    )

    val documentPickerLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.GetMultipleContents(),
        onResult = { uris -> viewModel.onDocumentsSelected(uris) }
    )

    LaunchedEffect(state.userMessage) {
        state.userMessage?.let {
            snackbarHostState.showSnackbar(it)
            viewModel.onUserMessageShown()
```

```
    }
}

LaunchedEffect(state.isProjectUpdated) {
    if (state.isProjectUpdated) {
        navController.popBackStack()
    }
}

Scaffold(
    topBar = {
        TopAppBar(
            title = { Text("Editar Proyecto") },
            navigationIcon = {
                IconButton(onClick = { navController.popBackStack() }) {
                    Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
                }
            }
        )
    },
    snackbarHost = { SnackbarHost(snackbarHostState) }
) { padding ->
    if (state.initialLoading) {
        Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
            CircularProgressIndicator()
        }
    } else {
```

```
            EditProjectForm(
                modifier = Modifier.padding(padding),
                state = state,
                viewModel = viewModel,
                onImageSelect = { imagePickerLauncher.launch("image/*") },
                onDocumentSelect = { documentPickerLauncher.launch("*/*") }
            )
        }
    }
}

@Composable
private fun EditProjectForm(
    modifier: Modifier,
    state: EditProjectState,
    viewModel: EditProjectViewModel,
    onImageSelect: () -> Unit,
    onDocumentSelect: () -> Unit
) {
    Column(modifier =
modifier.fillMaxSize().verticalScroll(rememberScrollState()).padding(16.dp)) {
        OutlinedCard(Modifier.fillMaxWidth()) {
            Column(Modifier.padding(16.dp)) {
                OutlinedTextField(value = state.title, onValueChange =
viewModel::onTitleChange, label = { Text("Título del proyecto") }, modifier =
Modifier.fillMaxWidth())
                Spacer(Modifier.height(16.dp))
```

```
            OutlinedTextField(value = state.description, onValueChange =
viewModel::onDescriptionChange, label = { Text("Descripción") }, modifier =
Modifier.fillMaxWidth(), minLines = 4)
        }
    }

    Spacer(Modifier.height(16.dp))

    OutlinedCard(Modifier.fillMaxWidth()) {
        Column(Modifier.padding(16.dp)) {
            CategoryDropdown(state.category, viewModel::onCategoryChange)
            Spacer(Modifier.height(16.dp))
            OutlinedTextField(value = state.institution, onValueChange =
viewModel::onInstitutionChange, label = { Text("Institución") }, modifier =
Modifier.fillMaxWidth())
            Spacer(Modifier.height(16.dp))
            OutlinedTextField(value = state.fundingGoal, onValueChange =
viewModel::onFundingGoalChange, label = { Text("Meta de financiamiento") }, modifier
= Modifier.fillMaxWidth())
        }
    }

    Spacer(Modifier.height(16.dp))

    ImagePicker(state.newImageUri ?: state.currentImageUrl, onImageSelect)
    Spacer(Modifier.height(16.dp))
    DocumentPicker(state.newDocumentUris, onDocumentSelect)

    Spacer(Modifier.height(32.dp))
```

```
        Button(
            onClick = viewModel::updateProject,
            enabled = !state.isLoading,
            modifier = Modifier.fillMaxWidth()
        ) {
            if (state.isLoading) {
                CircularProgressIndicator(modifier = Modifier.size(20.dp), strokeWidth = 2.dp)
            } else {
                Icon(Icons.Default.CloudUpload, contentDescription = null, modifier =
Modifier.padding(end = 8.dp))
                Text("Guardar Cambios")
            }
        }
    }
}


@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun CategoryDropdown(selectedCategory: String, onCategoryChange: (String) ->
Unit) {
    var expanded by remember { mutableStateOf(false) }
    val categories = listOf("Tecnología", "Salud", "Educación", "Medio Ambiente", "Arte y
Cultura", "Social")


    ExposedDropdownMenuBox(expanded = expanded, onExpandedChange = { expanded =
!expanded }) {
        OutlinedTextField(
```

```
            value = selectedCategory,
            onValueChange = {},
            readOnly = true,
            label = { Text("Categoría") },
            trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = expanded) },
            modifier = Modifier.menuAnchor().fillMaxWidth()
        )
        ExposedDropdownMenu(expanded = expanded, onDismissRequest = { expanded = false }) {
            categories.forEach { category ->
                DropdownMenuItem(text = { Text(category) }, onClick = { onCategoryChange(category); expanded = false })
            }
        }
    }
}

@Composable
private fun ImagePicker(image: Any?, onImageSelect: () -> Unit) {
    OutlinedButton(onClick = onImageSelect, modifier = Modifier.fillMaxWidth()) {
        Icon(Icons.Default.Image, contentDescription = null, modifier = Modifier.padding(end = 8.dp))
        Text("Cambiar Imagen de Portada")
    }
    if (image != null) {
        AsyncImage(model = image, contentDescription = "Imagen seleccionada", modifier = Modifier.fillMaxWidth().height(200.dp).padding(top = 8.dp))
    }
```

```
}

@Composable
private fun DocumentPicker(documentUris: List<Uri>, onDocumentSelect: () -> Unit) {
    OutlinedButton(onClick = onDocumentSelect, modifier = Modifier.fillMaxWidth()) {
        Icon(Icons.Default.AttachFile, contentDescription = null, modifier =
Modifier.padding(end = 8.dp))
        Text("Adjuntar Nuevos Documentos")
    }
    documentUris.forEach { uri ->
        Text("Nuevo: ${uri.lastPathSegment}", style = MaterialTheme.typography.bodySmall,
modifier = Modifier.padding(start = 16.dp, top = 4.dp))
    }
}

@HiltViewModel
class EditProjectViewModel @Inject constructor(
    private val getProjectByIdUseCase: GetProjectByIdUseCase,
    private val updateProjectUseCase: UpdateProjectUseCase,
    savedStateHandle: SavedStateHandle
) : ViewModel() {

    private val _uiState = MutableStateFlow(EditProjectState())
    val uiState: StateFlow<EditProjectState> = _uiState.asStateFlow()

    private val projectId: String = savedStateHandle.get<String>("projectId")!!

    init {
```

```
        loadProjectDetails()
    }

    private fun loadProjectDetails() {
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(initialLoading = true)
            val project = getProjectByIdUseCase(projectId)
            if (project != null) {
                _uiState.value = _uiState.value.copy(
                    projectId = project.id,
                    title = project.title,
                    description = project.description,
                    category = project.category,
                    institution = project.institution,
                    fundingGoal = project.fundingGoal.toString(),
                    currentImageUrl = project.imageUrl,
                    initialLoading = false
                )
            } else {
                _uiState.value = _uiState.value.copy(userMessage = "No se pudo cargar el proyecto.", initialLoading = false)
            }
        }
    }

    fun onTitleChange(title: String) { _uiState.value = _uiState.value.copy(title = title) }
    fun onDescriptionChange(description: String) { _uiState.value = _uiState.value.copy(description = description) }
```

```
    fun onCategoryChange(category: String) { _uiState.value = _uiState.value.copy(category = category) }

    fun onInstitutionChange(institution: String) { _uiState.value = _uiState.value.copy(institution = institution) }

    fun onFundingGoalChange(goal: String) { _uiState.value = _uiState.value.copy(fundingGoal = goal) }

    fun onImageSelected(uri: Uri?) { _uiState.value = _uiState.value.copy(newImageUri = uri) }

    fun onDocumentsSelected(uris: List<Uri>) { _uiState.value = _uiState.value.copy(newDocumentUris = uris) }


    fun updateProject() {

        viewModelScope.launch {

            val state = _uiState.value

            if (state.title.isBlank() || state.description.isBlank() || state.fundingGoal.toDoubleOrNull() == null) {

                _uiState.value = _uiState.value.copy(userMessage = "Por favor, completa todos los campos.")

                return@launch

            }


            _uiState.value = state.copy(isLoading = true)


            val result = updateProjectUseCase(

                projectId = state.projectId,

                title = state.title,

                description = state.description,

                category = state.category,

                institution = state.institution,
```

```
                fundingGoal = state.fundingGoal.toDouble(),
                newImageUri = state.newImageUri,
                newDocumentUris = state.newDocumentUris
            )

            result.onSuccess {
                _uiState.value = _uiState.value.copy(isLoading = false, isProjectUpdated = true)
            }.onFailure {
                _uiState.value = _uiState.value.copy(isLoading = false, userMessage =
it.message ?: "Error desconocido")
            }
        }
    }

    fun onUserMessageShown() {
        _uiState.value = _uiState.value.copy(userMessage = null)
    }
}

data class EditProjectState(
    val projectId: String = "",
    val title: String = "",
    val description: String = "",
    val category: String = "",
    val institution: String = "",
    val fundingGoal: String = "",
    val currentImageUrl: String? = null,
```

```
    val newImageUri: Uri? = null,
    val newDocumentUris: List<Uri> = emptyList(),
    val isLoading: Boolean = false,
    val isProjectUpdated: Boolean = false,
    val userMessage: String? = null,
    val initialLoading: Boolean = true
)
```

ProjectDetailScreen

```
package com.hugoguerrero.tecno.ui.screens.project

import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.CircleShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.filled.Description
import androidx.compose.material.icons.filled.Favorite
import androidx.compose.material.icons.filled.Group
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
```

```
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalUriHandler
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
import coil.compose.AsyncImage
import com.hugoguerrero.tecno.data.model.Donation
import com.hugoguerrero.tecno.data.model.Project
import com.hugoguerrero.tecno.data.model.UserProfile
import com.hugoguerrero.tecno.domain.use_case.GetDonationsUseCase
import com.hugoguerrero.tecno.domain.use_case.ObserveProjectByIdUseCase
import com.hugoguerrero.tecno.domain.use_case.GetUserProfileUseCase
import com.hugoguerrero.tecno.ui.navigation.Screens
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.flow.*
import javax.inject.Inject
```

```
data class DonorInfo(val profile: UserProfile, val donation: Donation)


@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ProjectDetailScreen(
    navController: NavController,
    viewModel: ProjectDetailViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()
    val uriHandler = LocalUriHandler.current


    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text(state.project?.title ?: "") },
                navigationIcon = {
                    IconButton(onClick = { navController.popBackStack() }) {
                        Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
                    }
                }
            )
        },
        floatingActionButton = {
            state.project?.let {
                ExtendedFloatingActionButton(
                    onClick = { navController.navigate(Screens.Donation.route + "/${it.id}") },
                    icon = { Icon(Icons.Default.Favorite, contentDescription = "Apoyar") },
```

```
                text = { Text("Apoyar este proyecto") },
                elevation = FloatingActionButtonDefaults.elevation(8.dp)
            )
        }
    }
) { padding ->
    Box(modifier = Modifier.padding(padding).fillMaxSize()) {
        when {
            state.isLoading -> {
                CircularProgressIndicator(modifier = Modifier.align(Alignment.Center))
            }
            state.error != null -> {
                Text(text = state.error!!, modifier = Modifier.align(Alignment.Center), color =
MaterialTheme.colorScheme.error)
            }
            state.project != null -> {
                ProjectDetailContent(navController, state) { uri ->
                    try {
                        uriHandler.openUri(uri)
                    } catch (e: Exception) { /* Opcional: Manejar error */ }
                }
            }
        }
    }
}
```

```
@Composable
fun ProjectDetailContent(navController: NavController, state: ProjectDetailState,
onDocumentClick: (String) -> Unit) {
    LazyColumn(contentPadding = PaddingValues(bottom = 80.dp)) {
        item { ProjectHeader(project = state.project!!) }
        item { ProjectAuthor(profile = state.authorProfile, navController = navController) }
        item { Spacer(modifier = Modifier.height(16.dp)) }
        item { FundingInfo(project = state.project!!) }
        item { ProjectDescription(description = state.project!!.description) }

        if (state.project!!.documentUrls.isNotEmpty()) {
            item { SectionTitle(title = "Documentos Adjuntos", icon =
Icons.Default.Description) }
            items(state.project.documentUrls) {
                DocumentItem(documentUrl = it, onClick = { onDocumentClick(it) })
            }
        }

        if (state.donors.isNotEmpty()) {
            item { SectionTitle(title = "Donantes Recientes", icon = Icons.Default.Group) }
            items(state.donors) {
                DonorItem(donorInfo = it, onClick = {
navController.navigate(Screens.PublicProfile.route + "/${it.profile.uid}") })
            }
        }
    }
}
```

```
@Composable
fun ProjectHeader(project: Project) {
    AsyncImage(
        model = project.imageUrl,
        contentDescription = "Imagen del proyecto",
        modifier =
Modifier.fillMaxWidth().height(220.dp).background(MaterialTheme.colorScheme.surface
Variant),
        contentScale = ContentScale.Crop
    )
    Text(
        text = project.title,
        style = MaterialTheme.typography.headlineLarge,
        modifier = Modifier.padding(horizontal = 16.dp, vertical = 12.dp)
    )
}

@Composable
fun ProjectAuthor(profile: UserProfile?, navController: NavController) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp)
            .clickable { profile?.let { navController.navigate(Screens.PublicProfile.route +
"/${it.uid}") } },
        verticalAlignment = Alignment.CenterVertically
    ) {
        AsyncImage(
```

```
            model = profile?.photoUrl,
            contentDescription = "Autor",
            modifier =
Modifier.size(32.dp).clip(CircleShape).background(MaterialTheme.colorScheme.surfaceVa
riant)
        )
        Text(
            text = "por ${profile?.displayName ?: "..."}",
            style = MaterialTheme.typography.bodyLarge,
            fontWeight = FontWeight.SemiBold,
            color = MaterialTheme.colorScheme.primary,
            modifier = Modifier.padding(start = 12.dp)
        )
    }
}

@Composable
fun FundingInfo(project: Project) {
    OutlinedCard(modifier = Modifier.fillMaxWidth().padding(16.dp)) {
        Column(modifier = Modifier.padding(16.dp)) {
            val formattedProgress = "%.1f".format(project.progress)
            Text("Recaudado: $${project.currentFunding.toInt()} de
$${project.fundingGoal.toInt()}", style = MaterialTheme.typography.titleMedium,
fontWeight = FontWeight.Bold)
            Spacer(modifier = Modifier.height(12.dp))
            LinearProgressIndicator(
                progress = project.progress / 100f,
                modifier = Modifier.fillMaxWidth().height(8.dp).clip(CircleShape),
```

```
            strokeCap = StrokeCap.Round
        )
        Spacer(modifier = Modifier.height(4.dp))
        Text("$formattedProgress% completado", style =
MaterialTheme.typography.bodySmall, modifier = Modifier.align(Alignment.End))
    }
  }
}

@Composable
fun ProjectDescription(description: String) {
  Text(
    text = description,
    style = MaterialTheme.typography.bodyLarge,
    modifier = Modifier.padding(horizontal = 16.dp, vertical = 8.dp)
  )
}

@Composable
fun SectionTitle(title: String, icon: androidx.compose.ui.graphics.vector.ImageVector) {
  Row(verticalAlignment = Alignment.CenterVertically, modifier = Modifier.padding(start
= 16.dp, end = 16.dp, top = 24.dp, bottom = 8.dp)) {
    Icon(icon, contentDescription = title, tint = MaterialTheme.colorScheme.secondary)
    Text(
      text = title,
      style = MaterialTheme.typography.titleLarge,
      modifier = Modifier.padding(start = 12.dp)
    )
```

```
    }
}

@Composable
fun DocumentItem(documentUrl: String, onClick: () -> Unit) {
    ListItem(
        headlineContent = { Text("Ver Documento", fontWeight = FontWeight.SemiBold) },
        leadingContent = { Icon(Icons.Default.Description, contentDescription = null) },
        modifier = Modifier.clickable(onClick = onClick).padding(horizontal = 8.dp)
    )
}

@Composable
fun DonorItem(donorInfo: DonorInfo, onClick: () -> Unit) {
    ListItem(
        headlineContent = { Text(donorInfo.profile.displayName, fontWeight =
FontWeight.SemiBold) },
        leadingContent = {
            AsyncImage(
                model = donorInfo.profile.photoUrl,
                contentDescription = "Foto de perfil de ${donorInfo.profile.displayName}",
                modifier =
Modifier.size(40.dp).clip(CircleShape).background(MaterialTheme.colorScheme.surfaceVa
riant)
            )
        },
        modifier = Modifier.clickable(onClick = onClick).padding(horizontal = 8.dp)
    )
```

```
}

@OptIn(ExperimentalCoroutinesApi::class)
@HiltViewModel
class ProjectDetailViewModel @Inject constructor(
    private val observeProjectByIdUseCase: ObserveProjectByIdUseCase,
    private val getUserProfileUseCase: GetUserProfileUseCase,
    private val getDonationsUseCase: GetDonationsUseCase,
    savedStateHandle: SavedStateHandle
) : ViewModel() {

    private val _uiState = MutableStateFlow(ProjectDetailState(isLoading = true))
    val uiState: StateFlow<ProjectDetailState> = _uiState.asStateFlow()

    init {
        savedStateHandle.get<String>("projectId")?.let { projectId ->
            val projectFlow = observeProjectByIdUseCase(projectId)
            val donationsFlow = getDonationsUseCase(projectId)

            projectFlow.flatMapLatest { project ->
                if (project == null) {
                    return@flatMapLatest flowOf(ProjectDetailState(error = "Proyecto no
encontrado", isLoading = false))
                }

                val authorFlow = flow { emit(getUserProfileUseCase(project.ownerUid)) }
```

```
            val donorsFlow = donationsFlow.flatMapLatest { donations ->
                if (donations.isEmpty()) {
                    flowOf(emptyList<DonorInfo>()) // Return empty list if no donors
                } else {
                    val donorInfoFlows = donations.map { donation ->
                        flow {
                            emit(getUserProfileUseCase(donation.donorUid)?.let { profile ->
                                DonorInfo(profile, donation)
                            })
                        }
                    }
                    combine(donorInfoFlows) { donorInfos ->
                        donorInfos.filterNotNull() // Filter out any null profiles
                    }
                }
            }

            combine(authorFlow, donorsFlow) { author, donors ->
                ProjectDetailState(project = project, authorProfile = author, donors = donors,
isLoading = false)
            }
        }
        .onStart { _uiState.value = ProjectDetailState(isLoading = true) }
        .catch { e -> _uiState.value = ProjectDetailState(error = e.localizedMessage ?:
"Ocurrió un error", isLoading = false) }
        .onEach { state ->
            _uiState.value = state
        }
```

```
            .launchIn(viewModelScope)


        } ?: run {
            _uiState.value = ProjectDetailState(error = "ID de proyecto no proporcionado",
isLoading = false)
        }
    }
}


data class ProjectDetailState(
    val project: Project? = null,
    val authorProfile: UserProfile? = null,
    val donors: List<DonorInfo> = emptyList(),
    val isLoading: Boolean = false,
    val error: String? = null
)
```

ProjectListScreen

```
package com.hugoguerrero.tecno.ui.screens.project


import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
```

```
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.hugoguerrero.tecno.data.model.Project

@Composable
fun ProjectListScreen(
    projects: List<Project>,
    onProjectClick: (String) -> Unit,
    onCreateProject: () -> Unit,
    onRefresh: () -> Unit
) {
    Column(
        modifier = Modifier
            .fillMaxSize()
```

```
        .background(Color(0xFF0D1B2A))
        .padding(16.dp)
) {
    // Título y descripción
    Text(
        text = "Descubre proyectos",
        color = Color.White,
        fontSize = 20.sp,
        fontWeight = FontWeight.Bold,
        modifier = Modifier.padding(bottom = 8.dp)
    )
    Text(
        text = "Apoya iniciativas o conviértete en manager",
        color = Color(0xFFB0BEC5),
        fontSize = 14.sp,
        modifier = Modifier.padding(bottom = 16.dp)
    )

    // Lista de proyectos
    if (projects.isEmpty()) {
        Box(
            modifier = Modifier.fillMaxSize(),
            contentAlignment = Alignment.Center
        ) {
            Text(
                text = "No hay proyectos disponibles",
                color = Color(0xFFB0BEC5)
```

```
            )
        }
    } else {
        LazyColumn(
            verticalArrangement = Arrangement.spacedBy(16.dp)
        ) {
            items(projects) { project ->
                ProjectCard(
                    project = project,
                    onProjectClick = onProjectClick
                )
            }
        }
    }
}

@Composable
fun ProjectCard(
    project: Project,
    onProjectClick: (String) -> Unit
) {
    Card(
        colors = CardDefaults.cardColors(containerColor = Color(0xFF1B263B)),
        elevation = CardDefaults.cardElevation(defaultElevation = 4.dp),
        shape = RoundedCornerShape(16.dp),
        modifier = Modifier
```

```
            .fillMaxWidth()
            .clickable { onProjectClick(project.id) }
    ) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text(
                text = project.title,
                color = Color.White,
                fontWeight = FontWeight.Bold,
                fontSize = 18.sp
            )
            Text(
                text = project.description,
                color = Color(0xFFB0BEC5),
                fontSize = 14.sp,
                modifier = Modifier.padding(vertical = 8.dp)
            )

        }
    }
}
```

UploadProjectScreen

```
package com.hugoguerrero.tecno.ui.screens.project

import android.net.Uri
import androidx.activity.compose.rememberLauncherForActivityResult
```

```
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.automirrored.filled.ArrowBack
import androidx.compose.material.icons.filled.AttachFile
import androidx.compose.material.icons.filled.CloudUpload
import androidx.compose.material.icons.filled.Image
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.DropdownMenuItem
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.ExposedDropdownMenuBox
```

```
import androidx.compose.material3.ExposedDropdownMenuDefaults
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedButton
import androidx.compose.material3.OutlinedCard
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Scaffold
import androidx.compose.material3.SnackbarHost
import androidx.compose.material3.SnackbarHostState
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
```

```
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import androidx.navigation.NavController
import coil.compose.AsyncImage
import com.hugoguerrero.tecno.domain.use_case.CreateProjectUseCase
import com.hugoguerrero.tecno.domain.use_case.GetCurrentUserUseCase
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun UploadProjectScreen(
    navController: NavController,
    viewModel: CreateProjectViewModel = hiltViewModel()
) {
    val state by viewModel.uiState.collectAsState()
    val snackbarHostState = remember { SnackbarHostState() }
    val scope = rememberCoroutineScope()

    val imagePickerLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.GetContent(),
        onResult = { uri -> viewModel.onImageSelected(uri) }
    )
```

```
val documentPickerLauncher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.GetMultipleContents(),
    onResult = { uris -> viewModel.onDocumentsSelected(uris) }
)

LaunchedEffect(state.userMessage) {
    state.userMessage?.let {
        scope.launch { snackbarHostState.showSnackbar(it) }
        viewModel.onErrorShown()
    }
}

LaunchedEffect(state.isProjectCreated) {
    if (state.isProjectCreated) {
        navController.popBackStack()
    }
}

Scaffold(
    topBar = {
        TopAppBar(
            title = { Text("Crear Nuevo Proyecto") },
            navigationIcon = {
                IconButton(onClick = { navController.popBackStack() }) {
                    Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
                }
```

```
                }
            )
        },
        snackbarHost = { SnackbarHost(snackbarHostState) }
    ) { padding ->
        UploadProjectForm(
            modifier = Modifier.padding(padding),
            state = state,
            viewModel = viewModel,
            onImageSelect = { imagePickerLauncher.launch("image/*") },
            onDocumentSelect = { documentPickerLauncher.launch("*/*") }
        )
    }
}

@Composable
private fun UploadProjectForm(
    modifier: Modifier,
    state: CreateProjectState,
    viewModel: CreateProjectViewModel,
    onImageSelect: () -> Unit,
    onDocumentSelect: () -> Unit
) {
    Column(modifier =
modifier.fillMaxSize().verticalScroll(rememberScrollState()).padding(16.dp)) {
        OutlinedCard(Modifier.fillMaxWidth()) {
            Column(Modifier.padding(16.dp)) {
```

```
            OutlinedTextField(value = state.title, onValueChange =
viewModel::onTitleChange, label = { Text("Título del proyecto") }, modifier =
Modifier.fillMaxWidth())

            Spacer(Modifier.height(16.dp))

            OutlinedTextField(value = state.description, onValueChange =
viewModel::onDescriptionChange, label = { Text("Descripción") }, modifier =
Modifier.fillMaxWidth(), minLines = 4)

        }

    }


    Spacer(Modifier.height(16.dp))


    OutlinedCard(Modifier.fillMaxWidth()) {

        Column(Modifier.padding(16.dp)) {

            CategoryDropdown(state.category, viewModel::onCategoryChange)

            Spacer(Modifier.height(16.dp))

            OutlinedTextField(value = state.institution, onValueChange =
viewModel::onInstitutionChange, label = { Text("Institución") }, modifier =
Modifier.fillMaxWidth())

            Spacer(Modifier.height(16.dp))

            OutlinedTextField(value = state.fundingGoal, onValueChange =
viewModel::onFundingGoalChange, label = { Text("Meta de financiamiento") }, modifier
= Modifier.fillMaxWidth())

        }

    }


    Spacer(Modifier.height(16.dp))


    ImagePicker(state.imageUri, onImageSelect)

    Spacer(Modifier.height(16.dp))
```

```
        DocumentPicker(state.documentUris, onDocumentSelect)

        Spacer(Modifier.height(32.dp))

        Button(
            onClick = viewModel::createProject,
            enabled = !state.isLoading,
            modifier = Modifier.fillMaxWidth()
        ) {
            if(state.isLoading) {
                CircularProgressIndicator(modifier = Modifier.size(20.dp), strokeWidth = 2.dp)
            } else {
                Icon(Icons.Default.CloudUpload, contentDescription = null, modifier =
Modifier.padding(end = 8.dp))
                Text("Publicar Proyecto")
            }
        }
    }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun CategoryDropdown(selectedCategory: String, onCategoryChange: (String) ->
Unit) {
    var expanded by remember { mutableStateOf(false) }
    val categories = listOf("Tecnología", "Salud", "Educación", "Medio Ambiente", "Arte y
Cultura", "Social")
```

```
    ExposedDropdownMenuBox(expanded = expanded, onExpandedChange = { expanded = !expanded }) {
        OutlinedTextField(
            value = selectedCategory,
            onValueChange = {},
            readOnly = true,
            label = { Text("Categoría") },
            trailingIcon = { ExposedDropdownMenuDefaults.TrailingIcon(expanded = expanded) },
            modifier = Modifier.menuAnchor().fillMaxWidth()
        )
        ExposedDropdownMenu(expanded = expanded, onDismissRequest = { expanded = false }) {
            categories.forEach { category ->
                DropdownMenuItem(text = { Text(category) }, onClick = { onCategoryChange(category); expanded = false })
            }
        }
    }
}

@Composable
private fun ImagePicker(imageUri: Uri?, onImageSelect: () -> Unit) {
    OutlinedButton(onClick = onImageSelect, modifier = Modifier.fillMaxWidth()) {
        Icon(Icons.Default.Image, contentDescription = null, modifier = Modifier.padding(end = 8.dp))
        Text("Seleccionar Imagen de Portada")
    }
    if (imageUri != null) {
```

```
        AsyncImage(model = imageUri, contentDescription = "Imagen seleccionada",
modifier = Modifier.fillMaxWidth().height(200.dp).padding(top =
8.dp).clip(RoundedCornerShape(12.dp)))
    }
}

@Composable
private fun DocumentPicker(documentUris: List<Uri>, onDocumentSelect: () -> Unit) {
    OutlinedButton(onClick = onDocumentSelect, modifier = Modifier.fillMaxWidth()) {
        Icon(Icons.Default.AttachFile, contentDescription = null, modifier =
Modifier.padding(end = 8.dp))
        Text("Adjuntar Documentos")
    }
    documentUris.forEach { uri ->
        Text("Archivo: ${uri.lastPathSegment}", style =
MaterialTheme.typography.bodySmall, modifier = Modifier.padding(start = 16.dp, top =
4.dp))
    }
}



@HiltViewModel
class CreateProjectViewModel @Inject constructor(
    private val createProjectUseCase: CreateProjectUseCase,
    private val getCurrentUserUseCase: GetCurrentUserUseCase
) : ViewModel() {

    private val _uiState = MutableStateFlow(CreateProjectState())
    val uiState: StateFlow<CreateProjectState> = _uiState.asStateFlow()
```

```
    fun onTitleChange(title: String) { _uiState.value = _uiState.value.copy(title = title) }

    fun onDescriptionChange(description: String) { _uiState.value =
_uiState.value.copy(description = description) }

    fun onCategoryChange(category: String) { _uiState.value = _uiState.value.copy(category
= category) }

    fun onInstitutionChange(institution: String) { _uiState.value =
_uiState.value.copy(institution = institution) }

    fun onFundingGoalChange(goal: String) { _uiState.value =
_uiState.value.copy(fundingGoal = goal) }

    fun onImageSelected(uri: Uri?) { _uiState.value = _uiState.value.copy(imageUri = uri) }

    fun onDocumentsSelected(uris: List<Uri>) { _uiState.value =
_uiState.value.copy(documentUris = uris) }


    fun createProject() {
        viewModelScope.launch {
            val state = _uiState.value
            val currentUser = getCurrentUserUseCase()


            if (currentUser == null) {
                _uiState.value = _uiState.value.copy(userMessage = "Error de autenticación.")
                return@launch
            }
            if (state.imageUri == null) {
                _uiState.value = _uiState.value.copy(userMessage = "Por favor, selecciona una
imagen.")
                return@launch
            }
```

```
        if (state.title.isBlank() || state.description.isBlank() ||
state.fundingGoal.toDoubleOrNull() == null || state.category.isBlank()) {
            _uiState.value = _uiState.value.copy(userMessage = "Por favor, completa todos
los campos.")
            return@launch
        }

        _uiState.value = state.copy(isLoading = true)

        val result = createProjectUseCase(
            title = state.title,
            description = state.description,
            category = state.category,
            institution = state.institution,
            fundingGoal = state.fundingGoal.toDouble(),
            ownerUid = currentUser.uid,
            imageUri = state.imageUri!!,
            documentUris = state.documentUris
        )

        result.onSuccess {
            _uiState.value = _uiState.value.copy(isLoading = false, isProjectCreated = true)
        }.onFailure {
            _uiState.value = _uiState.value.copy(isLoading = false, userMessage =
it.message ?: "Error desconocido")
        }
    }
}
```

```
    fun onErrorShown() {
        _uiState.value = _uiState.value.copy(userMessage = null)
    }
}

data class CreateProjectState(
    val title: String = "",
    val description: String = "",
    val category: String = "",
    val institution: String = "",
    val fundingGoal: String = "",
    val imageUri: Uri? = null,
    val documentUris: List<Uri> = emptyList(),
    val isLoading: Boolean = false,
    val isProjectCreated: Boolean = false,
    val userMessage: String? = null
)
```

Ruta: app/src/main/java/com/hugoguerrero/tecno/ui/theme/

```kotlin
```

Color.kt

```
package com.hugoguerrero.tecno.ui.theme

import androidx.compose.ui.graphics.Color
```

```
val Purple80 = Color(0xFFD0BCFF)
val PurpleGrey80 = Color(0xFFCCC2DC)
val Pink80 = Color(0xFFEFB8C8)

val Purple40 = Color(0xFF6650a4)
val PurpleGrey40 = Color(0xFF625b71)
val Pink40 = Color(0xFF7D5260)
```

Theme.kt

```
package com.hugoguerrero.tecno.ui.theme

import android.app.Activity
import android.os.Build
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.darkColorScheme
import androidx.compose.material3.dynamicDarkColorScheme
import androidx.compose.material3.dynamicLightColorScheme
import androidx.compose.material3.lightColorScheme
import androidx.compose.runtime.Composable
import androidx.compose.runtime.SideEffect
import androidx.compose.ui.graphics.toArgb
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalView
import androidx.core.view.WindowCompat
```

```
private val DarkColorScheme = darkColorScheme(
    primary = Purple80,
    secondary = PurpleGrey80,
    tertiary = Pink80
)

private val LightColorScheme = lightColorScheme(
    primary = Purple40,
    secondary = PurpleGrey40,
    tertiary = Pink40
)

@Composable
fun TecnoTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
) {
    val colorScheme = when {
        dynamicColor && Build.VERSION.SDK_INT >= Build.VERSION_CODES.S -> {
            val context = LocalContext.current
            if (darkTheme) dynamicDarkColorScheme(context) else
dynamicLightColorScheme(context)
        }
        darkTheme -> DarkColorScheme
        else -> LightColorScheme
    }
```

```
    val view = LocalView.current
    if (!view.isInEditMode) {
        SideEffect {
            val window = (view.context as Activity).window
            window.statusBarColor = colorScheme.primary.toArgb()
            WindowCompat.getInsetsController(window, view).isAppearanceLightStatusBars = darkTheme
        }
    }

    MaterialTheme(
        colorScheme = colorScheme,
        typography = Typography,
        content = content
    )
}
```

Type.kt

```
package com.hugoguerrero.tecno.ui.theme

import androidx.compose.material3.Typography
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.sp
```

```
val Typography = Typography(
    bodyLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 16.sp,
        lineHeight = 24.sp,
        letterSpacing = 0.5.sp
    )
    /* Other default text styles to override
    titleLarge = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Normal,
        fontSize = 22.sp,
        lineHeight = 28.sp,
        letterSpacing = 0.sp
    ),
    labelSmall = TextStyle(
        fontFamily = FontFamily.Default,
        fontWeight = FontWeight.Medium,
        fontSize = 11.sp,
        lineHeight = 16.sp,
        letterSpacing = 0.5.sp
    )
    */
)
```

```

Modifica tu ManiActivity

MainActivity

```

```
package com.hugoguerrero.tecno

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.navigation.compose.rememberNavController
import com.hugoguerrero.tecno.ui.navigation.MainNavGraph
import com.hugoguerrero.tecno.ui.theme.TecnoTheme
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint // Anotación añadida
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContent {
            TecnoTheme {
                val navController = rememberNavController()
                MainNavGraph(navController = navController)
            }
        }
    }
```

}
```

Crea TecnoApp al mismo nivel del MainActivity

```
package com.hugoguerrero.tecno

import android.app.Application
import com.google.firebase.appcheck.debug.DebugAppCheckProviderFactory
import com.google.firebase.appcheck.ktx.appCheck
import com.google.firebase.appcheck.playintegrity.PlayIntegrityAppCheckProviderFactory
import com.google.firebase.ktx.Firebase
import com.hugoguerrero.tecno.BuildConfig
import dagger.hilt.android.HiltAndroidApp

@HiltAndroidApp
class TecnoApp : Application() {

    override fun onCreate() {
        super.onCreate()

        // Inicializar Firebase App Check
        if (BuildConfig.DEBUG) {
            // Usar el proveedor de depuración para las pruebas
            Firebase.appCheck.installAppCheckProviderFactory(
                DebugAppCheckProviderFactory.getInstance(),
)
        } else {
```
```

```
            // Usar el proveedor de producción para la versión de lanzamiento
            Firebase.appCheck.installAppCheckProviderFactory(
                PlayIntegrityAppCheckProviderFactory.getInstance(),
            )
        }
    }
}
```

🚀 Cómo Ejecutar

```

Prerrequisitos

- Android Studio Flamingo (2022.2.1) o superior
- JDK 17+
- Dispositivo Android con API 24+ o emulador

```

Pasos de Instalación

```

1. Clona el repositorio:

bash

git clone https://github.com/Hugo-Guerrero/Tecno.git

cd Tecno

2. Configura Firebase:

o Ve a Firebase Console

o Crea un nuevo proyecto Android

o Descarga google-services.json y colócalo en app/
```

3. Configura las credenciales (opcional para release):

o Copia local.properties.example a local.properties

o Configura tus claves de firma:

```

properties

```

storeFile=../keystore/tecno.jks

storePassword=tu_password

keyAlias=tecno

keyPassword=tu_password

4. Ejecuta la aplicación:

o Abre el proyecto en Android Studio

o Haz clic en Run ▶️

o Selecciona un dispositivo o emulador

```

## 🤝 Contribuir

```

1. Haz fork del repositorio
2. Crea una rama para tu feature:

bash

git checkout -b feature/nueva-funcionalidad

3. Commitea tus cambios:

bash

git commit -m 'feat: añade nueva funcionalidad'

4. Push a la rama:

bash

git push origin feature/nueva-funcionalidad

5. Abre un Pull Request

Guía de estilo

- Usa KDoc para documentación
- Sigue las convenciones de código Kotlin
- Mantén las pruebas unitarias actualizadas

```

Licencia

```

LICENCIA PARA PROYECTO ESCOLAR

Este código se comparte únicamente con fines de visualización y evaluación académica.

Queda estrictamente prohibido:

- Modificar, distribuir o publicar este código
- Usar este código en otros proyectos
- Realizar forks con modificaciones

Este proyecto es parte de un trabajo escolar y no está abierto a contribuciones externas.

Para cualquier consulta sobre el uso de este código, contactar a:
hugoguerrerocampo217@gmail.com

Autor

- Desarrollador: Hugo Guerrero
- Autor del repo: Hugo Guerrero
- Colaborador: Jorge Alberto

```
```