Manual Técnico

Diseño Apps - Ricarte Cano Héctor Emilio

DrimVoice

Integrantes:

Kevin Joel Aparicio Espino

Edwin Karim Bolaños Sánchez

Joel Alejandro Chaparro González

Brandon Enrique Espinoza Salcedo

TDM4

Indice

Contenido

Introducción

Este proyecto aborda la necesidad de muchos usuarios de registrar y gestionar ideas o recordatorios de manera práctica y accesible en su día a día. La meta es crear una aplicación móvil que permita a los usuarios grabar notas transformados a texto, ofreciendo además la opción de editarlas por escrito, para garantizar flexibilidad y adaptabilidad en su uso.

Bases del proyecto

Necesidad a atender:

Muchos usuarios encuentran complicado registrar ideas, recordatorios o notas importantes de manera inmediata cuando las condiciones no permiten escribirlas. Esto puede resultar en la pérdida de información valiosa o en un proceso engorroso de organización de las notas.

Meta:

Facilitar la captura de notas a través de la voz, transcribiéndolas automáticamente a texto editable, ofreciendo una forma eficiente y práctica de registrar ideas y recordatorios. Además de contar con un espacio único para guardar las notas, esto con ayuda de una base de datos y un login para iniciar la aplicación.

Objetivo General:

Crear una aplicación móvil que permita a los usuarios grabar sus notas mediante la voz, transcribiéndolas automáticamente a texto para ser guardadas. La aplicación también ofrecerá la opción de editar el texto registrado para garantizar flexibilidad y adaptabilidad, teniendo la seguridad de que las notas guardadas puedan verse en cualquier momento gracias a la base de datos y el inicio de sesión implementada.

Justificación:

En un mundo donde la inmediatez es clave, muchas personas enfrentan dificultades para registrar ideas rápidamente cuando no pueden escribirlas. Esto puede afectar su productividad y organización. Este proyecto busca solucionar ese problema mediante una aplicación intuitiva que permita transcribir notas de audio a texto, combinando la rapidez de las grabaciones de voz con la versatilidad del formato escrito.

Alcance del Proyecto:

El proyecto se centrará en desarrollar una aplicación móvil que facilite el registro de notas mediante la transcripción automática de voz a texto. Permitirá a los usuarios guardar y editar sus notas directamente en la aplicación. La aplicación incluirá almacenamiento de grabaciones de audio y una función de organización, priorizando la simplicidad y eficiencia en el registro de notas.

Desarrollo del Proyecto

Diagrama EDT

Requerimientos Generales

Diseño Limpio y Ordenado:

- La aplicación debe de tener un diseño limpio, ordenado e intuitivo para el usuario.
- Espacio para la grabación de voz y transcripción de texto.
- Espacio para notas guardas y edición de las mismas.

Grabación de voz:

- Se debe de implementar la funcionabilidad de grabación de voz y transcripción a texto con uso de la API SpeechRecognizer y guardar las notas en una base de datos usando la API de base de datos de FireBase

Registro e Inicio de Sesión:

- Debe permitir el registro e inicio de sesión de cuentas de usuario.

Base de Datos:

- Debe incluir una base de datos para almacenar la información necesaria. Esta base de datos estará estrechamente construida con FireBase

Optimización para Dispositivos:

- El sistema debe estar optimizado para ser ejecutado en varios dispositivos móviles.

Clasificación de SRS

FUNCIONALES	**NO FUNCIONALES**
Gestión de notas: • Guardar, editar y borrar las notas echas. • Grabación de voz y transcripción a texto	**Rendimiento:** • Tiempo de carga de la aplicación: 3 segundos.
Autenticación: • Registro de usuario. • Inicio de sesión de usuario.	**Seguridad:** • Copia de seguridad de las bases de datos (Notas y Usuarios).
Interfaz de usuario: • Interfaz funcional e intuitiva. • Interfaz organizada en base a espacios de edición, grabación y transcripción.	**Compatibilidad:** • La aplicación debe de ser compatible con varios dispositivos de Android en base a las funcionalidades.

Requerimientos Funcionales Detallados

Gestión de Notas

Requerimiento	**Función**
Grabar nota de voz	Permite grabar la nota usando las funciones de grabación de voz
Trascribir nota de voz	Con ayuda de la API SpeechRecognizer la nota de voz será transcrita en un espacio donde podremos visualizarla
Guardar nota de voz	Permite a los usuarios guardar sus notas para ser visualizadas en su cuenta.
Editar nota de voz	Permite a los usuarios editar las notas grabadas, teniendo un espacio de edición de texto

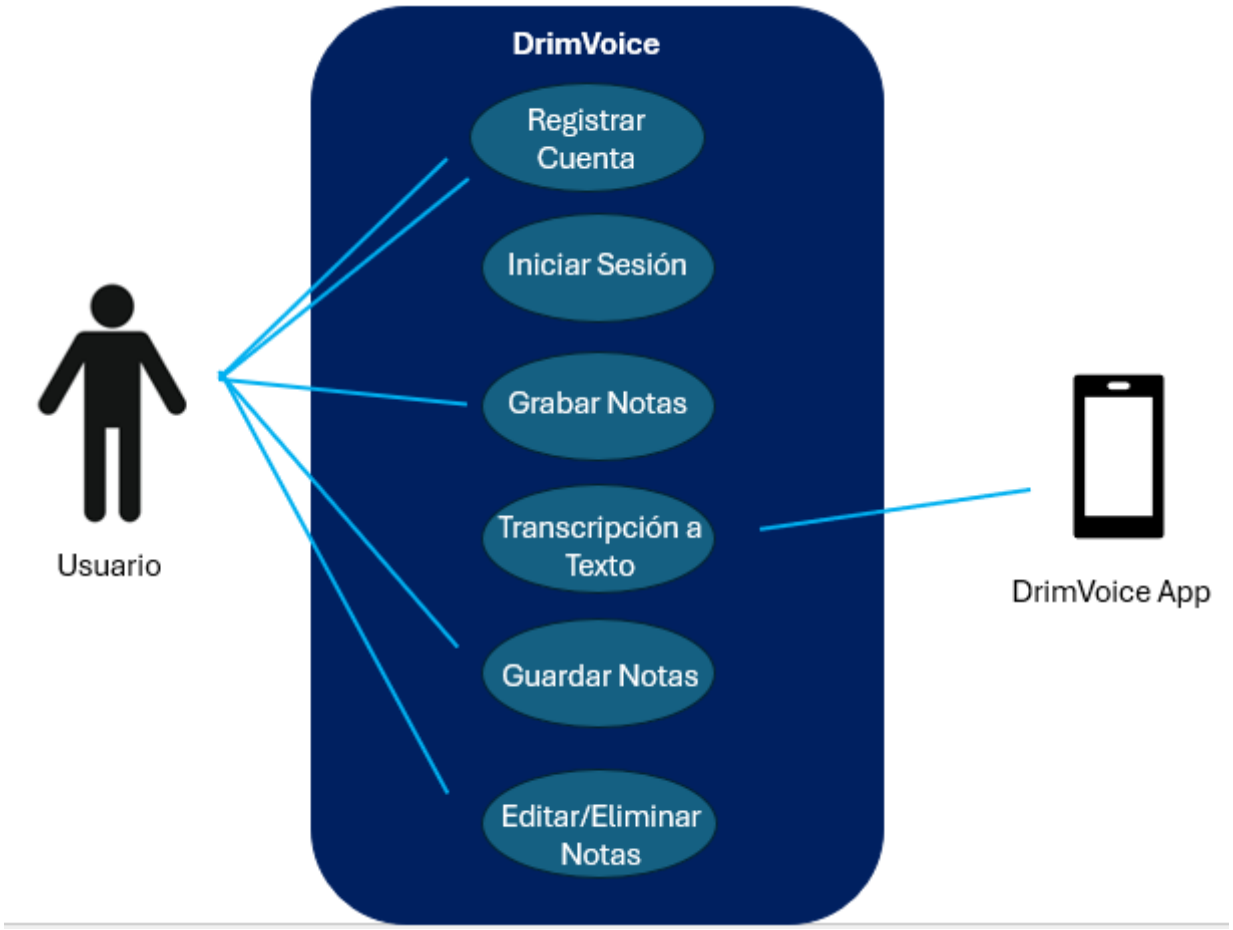
Autenticación:

Requerimiento	**Función**
Registro de usuario	La app debe permitir a los usuarios crear cuentas nuevas.
Inicio de sesión de usuario	La app debe permitir a los usuarios registrados iniciar sesión en sus cuentas.

Interfaz de usuario:

Requerimiento	**Función**
Interfaz intuitiva	La app debe tener una interfaz de usuario intuitiva y fácil de usar.
Interfaz organizada	La app debe de ser organizada donde se tengan claros los espacios de trabajo de la app

Caso de Uso General



Caso de uso:	Registro de la cuenta
Actores:	Usuario y Sistema
Propósito:	Registrar una cuenta
Resumen:	El usuario ingresa al apartado de registro de una cuenta y completa la información requerida.
Principal escenario de éxito:	
Acciones de los actores:	Respuesta del sistema:
1- El usuario al iniciar la app se encuentra con el apartado de iniciar sesión, si este no cuenta con una cuenta, puede hacer click en un botón para crear una. 4- El usuario completa el formulario y acepta los términos y condiciones.	2- El sistema lo redirige al apartado registrar cuenta. 3- El sistema muestra un formulario que solicita la información necesaria. 5- El sistema verifica los campos. 6- El sistema crea la cuenta del usuario en la base de datos del sistema. 7- El sistema muestra un mensaje de confirmación al usuario, indicando que la cuenta se ha creado con éxito. 8- Después de confirmar el registro, el usuario es redirigido al apartado de inicio de sesión.
Caso alternativo (datos erróneos):	
Acciones de los actores:	Respuesta del sistema:
	5- El sistema verifica los campos. 6- El sistema muestra que los campos que no son válidos y le permite al usuario modificarlos.

Caso de uso:	Inicio de sesión de la cuenta
Actores:	Usuario y Sistema
Propósito:	Iniciar sesión en una cuenta
Resumen:	El usuario ingresa al apartado de inicio de sesión e ingresa sus credenciales.
Principal escenario de éxito:	
Acciones de los actores:	Respuesta del sistema:
1- El usuario ingresa al apartado de iniciar sesión. 2- El usuario proporciona sus credenciales de inicio de sesión.	4- Si las credenciales son válidas, el sistema autentica al usuario y le otorga acceso a su cuenta. 5- El sistema puede registrar eventos de inicio. 6- Después de un inicio de sesión exitoso, el sistema redirige al usuario a la página principal.
Caso alternativo (datos erróneos):	
Acciones de los actores:	Respuesta del sistema:
	4- Si las credenciales no son válidas o hay algún otro problema, el sistema muestra un mensaje de error correspondiente y proporciona al usuario la oportunidad de corregir la información ingresada.

Caso de uso:	Grabar y Guardar Notas
Actores:	Usuario y Sistema
Propósito:	Grabar una nota usando la API para grabar la voz y transcribirlo a texto
Resumen:	El usuario da click en el botón “nombre” y grabara su nota
Principal escenario de éxito:	
Acciones de los actores:	Respuesta del sistema:
1- El usuario ingresa al apartado de “Nueva Nota”. 3- El usuario da click en el botón “Pulse para hablar “. 5- El usuario grabará su nota y al finalizar tendrá un botón para guardar	2- El sistema lo redirige al apartado para grabar la nota. 4- El sistema comenzara a grabar la nota de voz 6- El sistema mostrará un recuadro para agregar nombre a la nota y guardarla
Caso alternativo (datos erróneos):	
Acciones de los actores:	Respuesta del sistema:
	4- Si el usuario no otorga permisos de micrófono a la app se mostrará un mensaje pidiendo los permisos necesarios

Caso de uso:	Editar Notas
Actores:	Usuario y Sistema
Propósito:	Poder editar y eliminar las notas guardadas y
Resumen:	En la sección donde se almacenan las notas, se podrá acceder a la función de editar la nota o eliminarla
Principal escenario de éxito:	
Acciones de los actores:	Respuesta del sistema:
1- El usuario desplegara el menú lateral que está en la página principal 3- El usuario dará click en la nota 5- El usuario visualizará un apartado con la nota guardada y podrá escribir dentro de esta sección 6- El usuario dará click en el botón “guardar”	2- El sistema mostrará las notas guardadas 4- El sistema lo mandará a una sección donde tendrá las opciones de editar y eliminar 7- El sistema guardara la nota modificada

Caso de uso:	Eliminar Notas
Actores:	Usuario y Sistema
Propósito:	Poder eliminar las notas guardadas
Resumen:	En la sección donde se almacenan las notas, se podrá acceder a la función de editar la nota o eliminarla
Principal escenario de éxito:	
Acciones de los actores:	Respuesta del sistema:
2- El usuario desplegara el menú lateral que está en la página principal 4- El usuario dará click en la nota 6- El usuario dará click en el botón “eliminar”	2- El sistema mostrará las notas guardadas 4- El sistema lo mandará a una sección donde tendrá las opciones de editar y eliminar 7- El sistema eliminara la nota

Diagrama de Registro

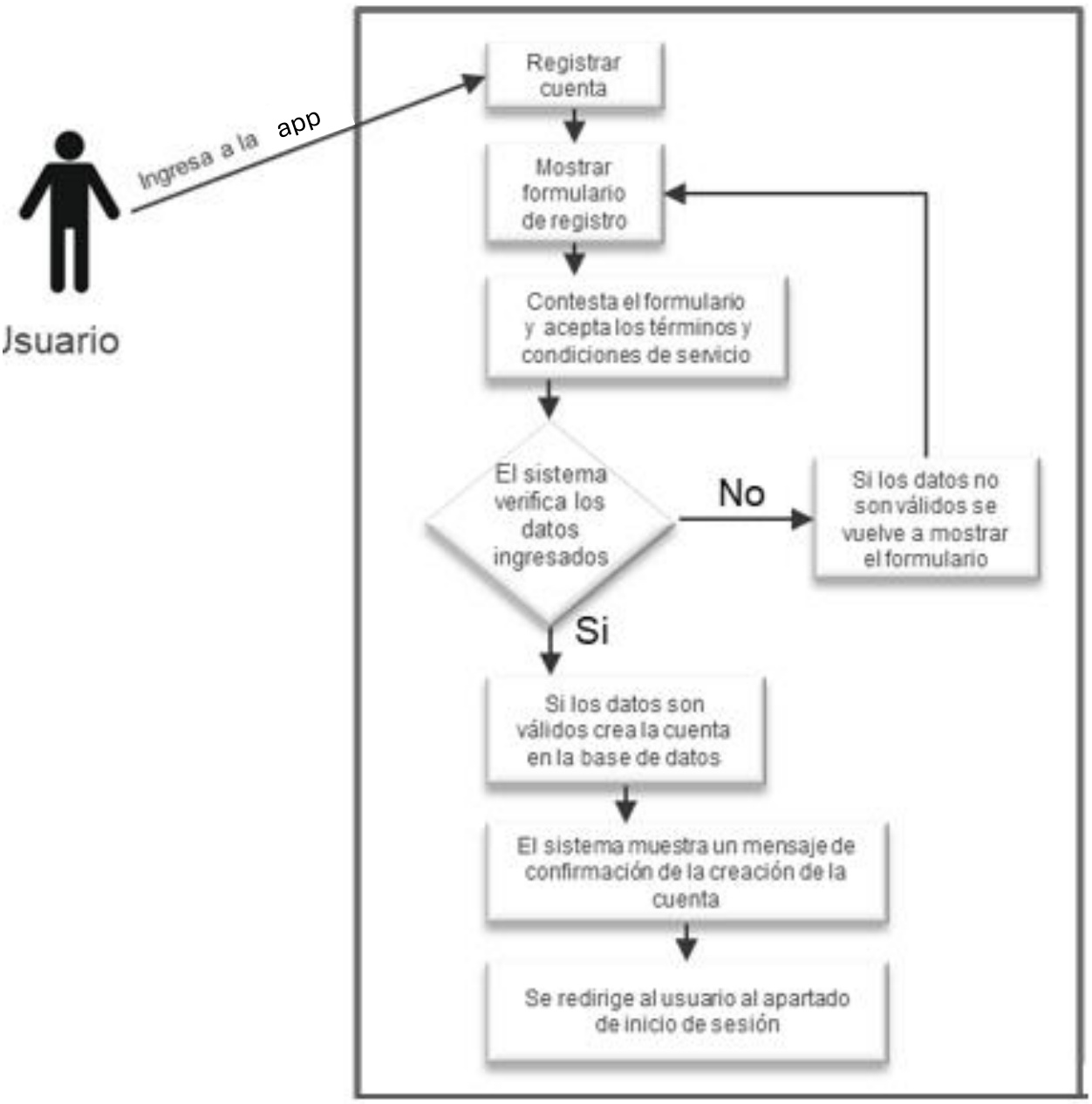


Diagrama de Inicio de Sesión

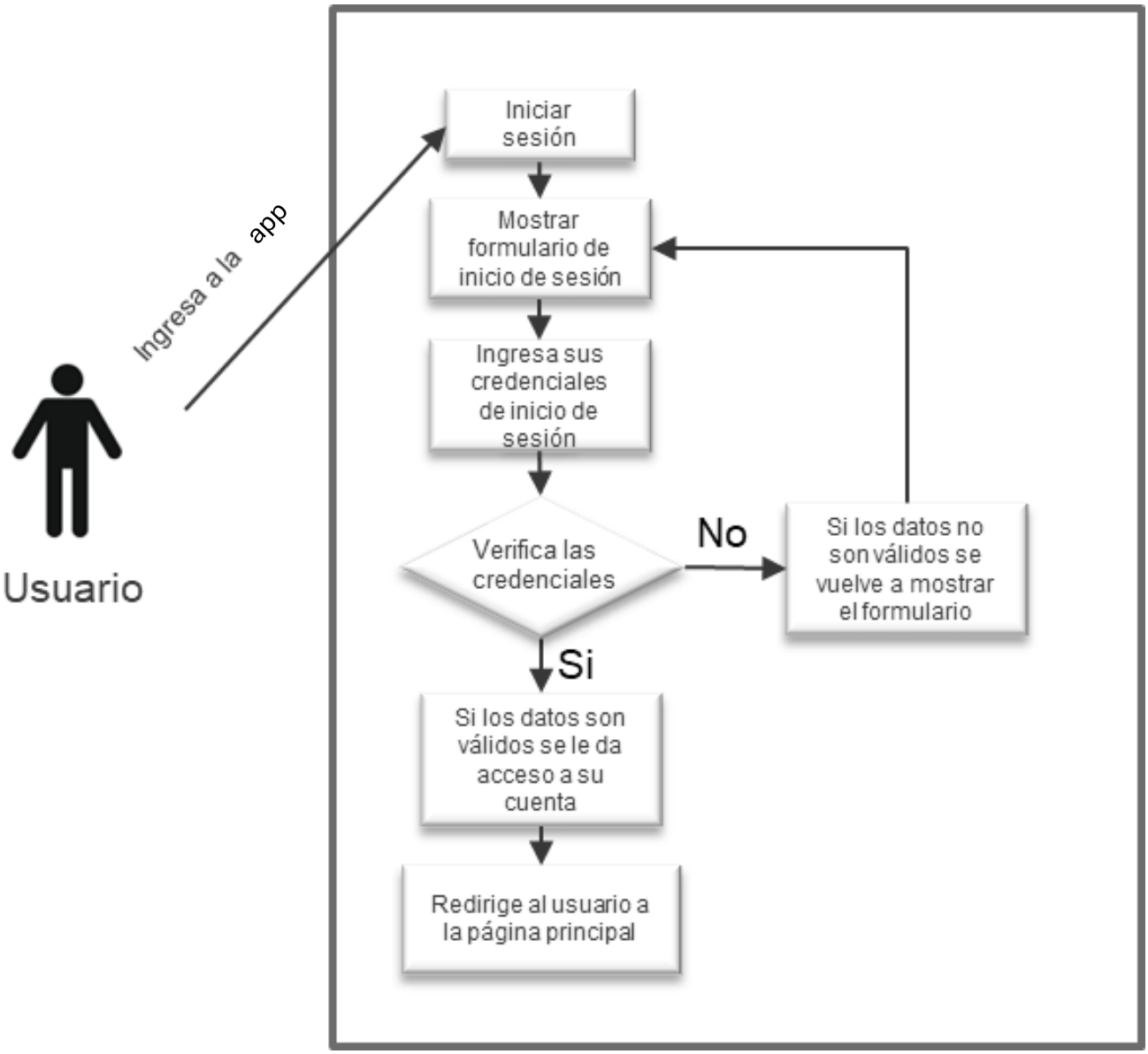


Diagrama de guardado de notas

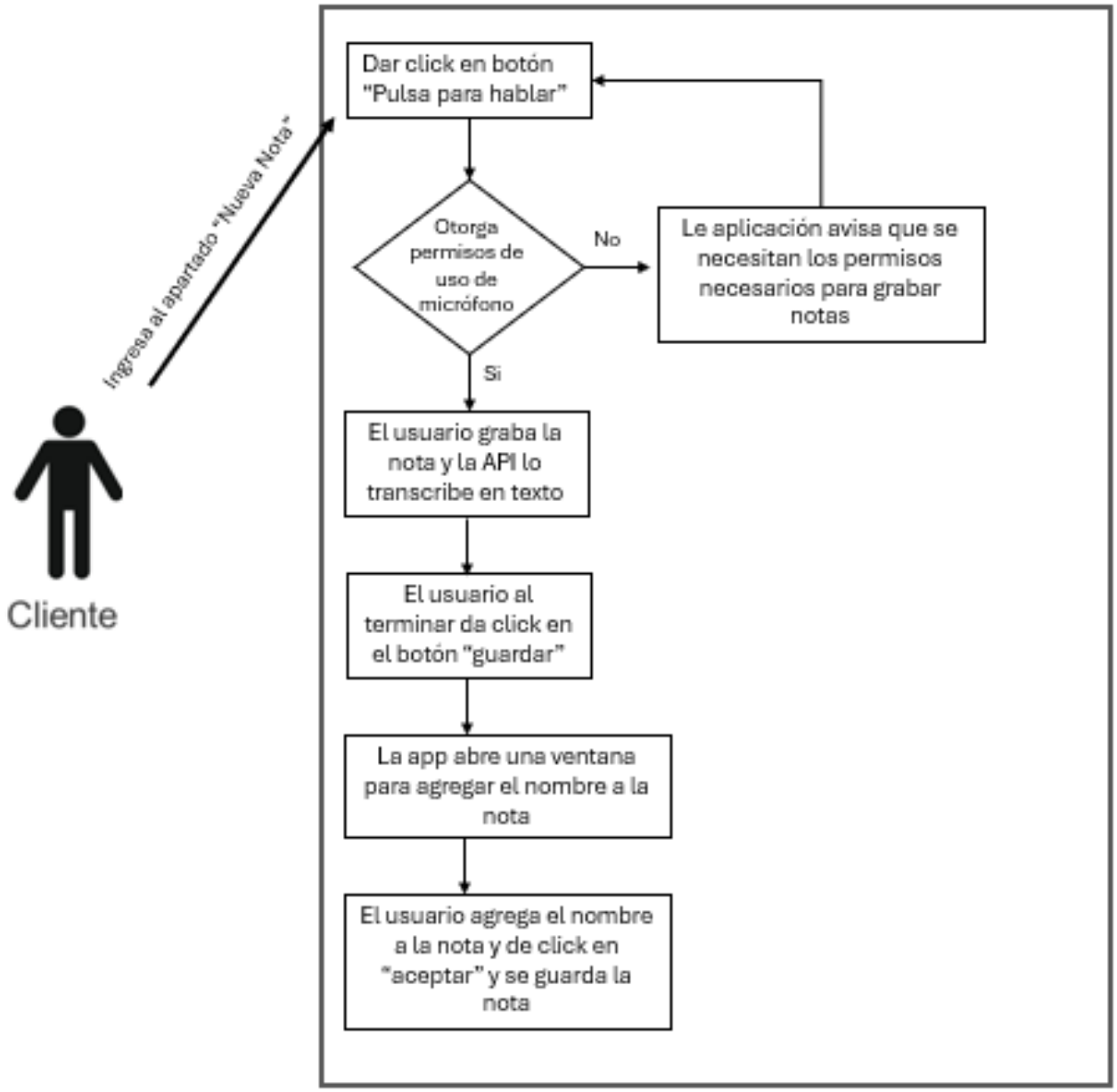
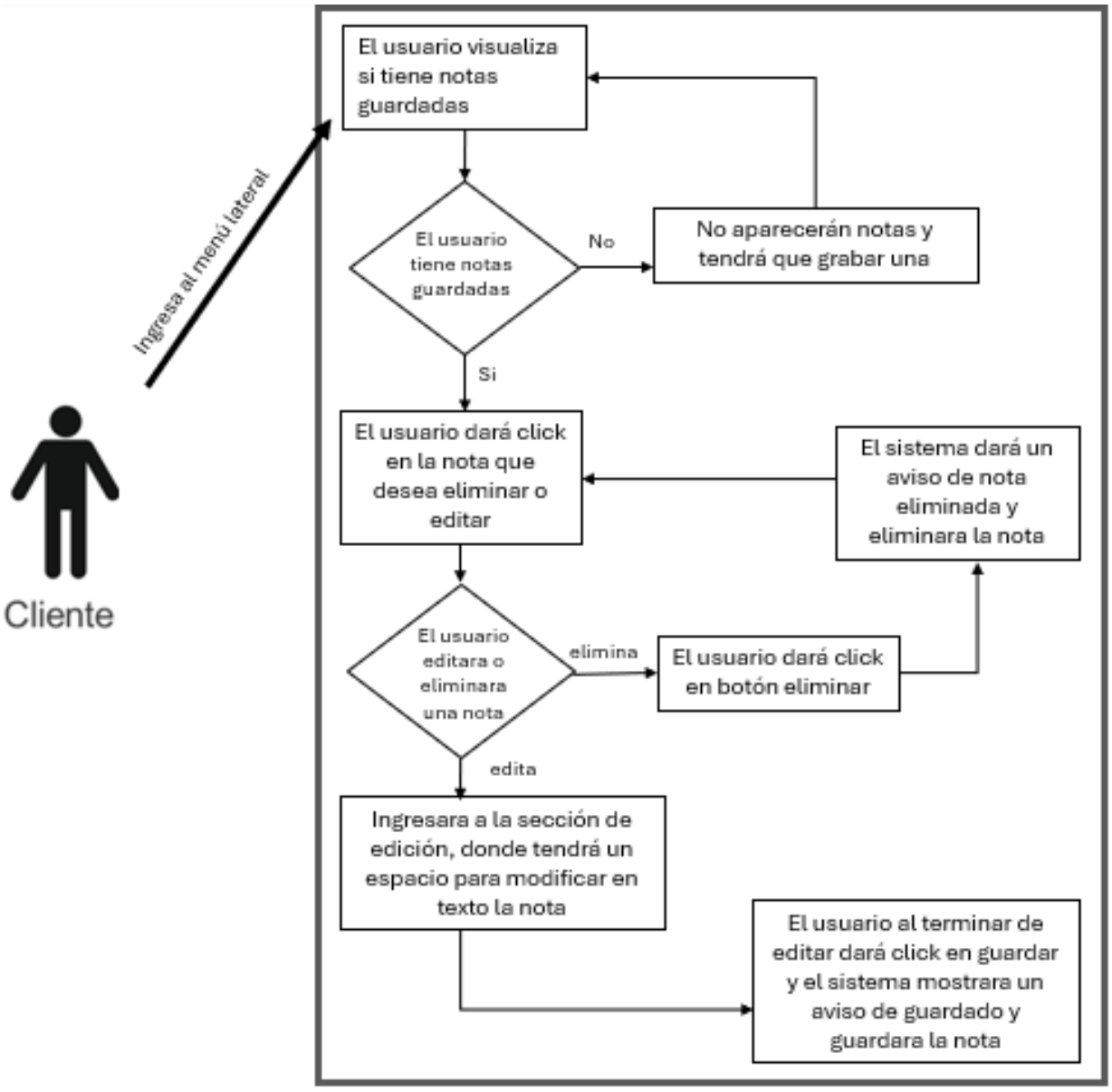


Diagrama de eliminar/editar notas



Mockups y prototipos

Paleta de colores

Justificación:

Para la paleta de colores utilizada en esta aplicación, se ha seleccionado una gradación de tonos que va desde un rojo oscuro (#5c0010) hasta un rosado claro (#ffe4ea). Esta elección está fundamentada en el significado emocional e impacto visual de los colores, alineándose con el propósito de la aplicación, que es permitir a los usuarios grabar notas de voz de manera intuitiva y emocional.

El color rojo, predominante en la paleta, está asociado con la energía, la pasión y la acción. En el contexto de esta aplicación, simboliza la importancia de las ideas y mensajes grabados, destacando

su valor y fomentando un sentido de urgencia y relevancia. La transición hacia tonos más claros, como los rosados, aporta un equilibrio visual que suaviza la intensidad del rojo, generando una experiencia más cálida y acogedora.

Además, los tonos rosados comunican calma, empatía y cercanía, cualidades que invitan a los usuarios a expresar sus pensamientos y emociones de manera cómoda y personal. La combinación de estos colores refuerza el objetivo de la aplicación, proporcionando un ambiente visual que inspira creatividad y expresión personal.

En términos de impacto visual, el contraste entre los tonos más oscuros y los más claros permite guiar la atención del usuario hacia elementos clave, como botones o secciones importantes. Por ejemplo, el rojo oscuro puede utilizarse en encabezados o íconos destacados, mientras que los tonos más claros, como el rosado suave, son ideales para los fondos, generando una interfaz limpia y ordenada.

Página principal

Nota

Grabar

Sección de grabación

Ingrese el titulo

Input

Texto

Editar

Guardar

Registro

DrimVoice

Registro de Usuario

Ingrese el correo

Correo electrónico

Ingrese la contraseña

Contraseña

Registrar

¿Ya tienes cuenta? Inicia sesión aquí

Inicio de sesión

DrimVoice

Inicio de sesion

Ingrese el correo

Correo electrónico

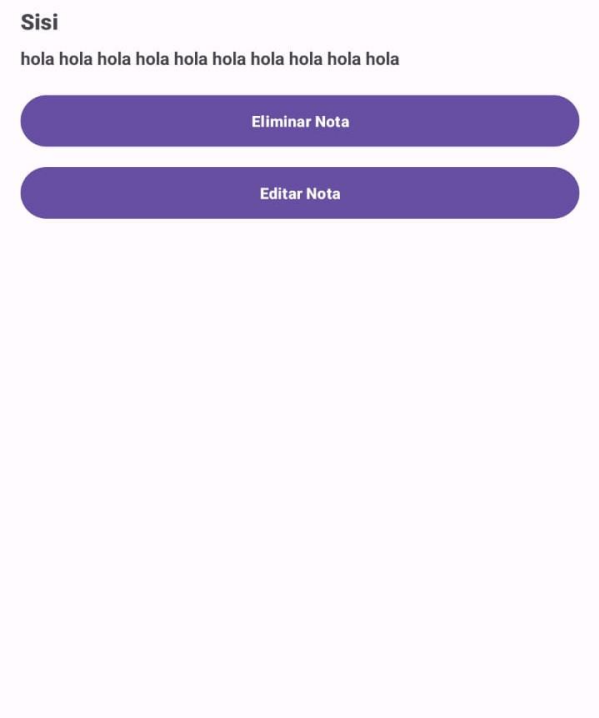
Ingrese la contraseña

Contraseña

Ingresar

¿No tienes cuenta? Regístrate aquí

otas



Principios de Gestalt utilizados

Proximidad:

Los elementos están relacionados entre sí y están agrupados de manera cercana. Esto facilita al usuario entender que estas opciones están relacionadas y forman parte de la misma funcionalidad.

Figura y fondo:

El contraste entre el fondo oscuro y los elementos más claros hace que los elementos importantes se destaquen claramente del fondo. Esto guía la atención del usuario hacia las partes interactivas de la página.

Cierre:

El uso de fondos y colores que enmarcan los contenidos.

Simetría:

La disposición de los elementos es bastante simétrica, lo que aporta un sentido de equilibrio y orden al diseño. Esto es evidente en cómo los campos de entrada y los botones están alineados uniformemente, proporcionando una estructura que es agradable a la vista.

Base de Datos

La aplicación utiliza una base de datos no relacional en Firebase para gestionar la información de los usuarios y sus respectivas notas. Este tipo de base de datos está basado en un esquema flexible que organiza los datos en colecciones y documentos, permitiendo un acceso rápido y eficiente, ideal para aplicaciones en tiempo real.

En este caso, la base de datos está estructurada con una colección principal llamada usuarios, en la que cada usuario se identifica de manera única mediante un ID. Cada documento dentro de esta colección contiene información básica del usuario, como su correo electrónico y contraseña, que se almacenan de forma segura.

Dentro de cada documento de usuario, existe una subcolección llamada notas, diseñada para almacenar todas las notas creadas por ese usuario en particular. Cada nota tiene su propio ID, además de dos campos principales: título y contenido, que representan el nombre y la información de la nota grabada, respectivamente.

Este enfoque jerárquico permite que los datos estén organizados de manera lógica, lo que facilita la consulta y manipulación de información específica. Por ejemplo, cuando un usuario inicia sesión, la aplicación puede acceder a su documento dentro de la colección de usuarios y, a partir de ahí, cargar o gestionar las notas almacenadas en su subcolección.

Además, el uso de Firebase como base de datos no relacional aporta beneficios como la sincronización en tiempo real, escalabilidad y flexibilidad en el diseño de datos, haciendo que esta solución sea ideal para una aplicación de grabación de notas. Esto asegura que las notas sean accesibles instantáneamente desde diferentes dispositivos y que los usuarios puedan interactuar con la aplicación de manera fluida y eficiente.

Explicación de Código:

MainActivty.kt:

En el siguiente código encontramos primeramente la definición del paquete com.example.aplicacionproyecto, que agrupa la funcionalidad de la aplicación, seguido de las importaciones necesarias para trabajar con elementos esenciales de Android, Firebase y el diseño de interfaz gráfica. Estas incluyen clases como Intent, FirebaseAuth, FirebaseFirestore, RecyclerView, entre otras, que serán usadas a lo largo del código.

La clase MainActivity, que hereda de AppCompatActivity, actúa como la actividad principal de la aplicación. Dentro de esta clase se declara un conjunto de variables, como drawerLayout, navigationView, auth, firestore, entre otras, que serán utilizadas para manejar la navegación, la autenticación, y el acceso a datos en Firestore.

En el método onCreate, que es el punto de inicio de la actividad, se configuran varios componentes clave:

1. Autenticación Firebase: Se verifica si hay un usuario autenticado. Si no lo hay, redirige a la actividad de inicio de sesión.
2. Firestore: Se inicializa para gestionar la base de datos en la nube.
3. RecyclerView: Configurado con un adaptador (noteAdapter) para mostrar una lista de notas recientes.
4. Interfaz gráfica: Configura el menú lateral (NavigationView) y un botón flotante para agregar nuevas notas.
5. Notas iniciales: Llama al método refreshNotes para cargar las notas del usuario desde Firestore.

El método showProfileMenu se encarga de mostrar un menú emergente cuando el usuario hace clic en la imagen de perfil. Este menú permite, por ejemplo, cerrar sesión utilizando el método logoutUser.

El método refreshNotes es responsable de actualizar la lista de notas y el menú lateral. Este se apoya en loadUserNotes, que consulta Firestore para cargar las notas almacenadas del usuario.

El manejo de resultados al agregar o visualizar notas se realiza en onActivityResult, actualizando la lista de notas o eliminando una nota si corresponde.

El método loadUserNotes realiza una consulta a Firestore para obtener las notas del usuario autenticado, las almacena en la lista local y las agrega al menú lateral mediante addNoteToDrawer.

Finalmente, el método saveNoteToFirestore guarda nuevas notas en Firestore, y addNoteToDrawer asegura que las notas también estén reflejadas en el menú lateral para facilitar la navegación.

En resumen, este código implementa una aplicación que permite a los usuarios autenticarse, visualizar y gestionar notas almacenadas en Firebase, integrando una interfaz moderna con un menú lateral, RecyclerView y soporte para operaciones CRUD.

MainActivity2.kt:

En el siguiente código tenemos la definición de una actividad en Android que se encarga de transcribir notas de voz, permitir su edición y guardarlas con un título, utilizando herramientas nativas como el reconocimiento de voz y el manejo de permisos. La clase principal **MainActivity2** extiende de **AppCompatActivity** e implementa las funcionalidades necesarias para esta operación.

Importaciones

El código incluye importaciones esenciales para manejar la interfaz gráfica, el reconocimiento de voz, los permisos, y los elementos del sistema Android. Entre ellas:

- **SpeechRecognizer** y **RecognizerIntent**: Facilitan la captura y procesamiento de voz.
- **Manifest** y **ActivityCompat**: Manejan los permisos para grabación de audio.

- **AlertDialog**: Muestra un cuadro de diálogo para ingresar el título de la nota.

Declaración de variables

Se definen variables clave:

1. **speechRecognizer**: Maneja la funcionalidad del reconocimiento de voz.
2. **textView**: Muestra el texto transcrito al usuario.
3. **startStopButton**: Inicia y detiene la grabación de voz.
4. **saveButton**: Permite guardar la nota.
5. **isListening**: Bandera que indica si la grabación está activa.
6. **accumulatedText**: Acumula las transcripciones obtenidas.
7. **REQUEST_CODE**: Código único para la solicitud de permisos de grabación.

Método onCreate

Este método inicializa la actividad:

1. **Asociación de elementos de la interfaz**:
    - Vincula el diseño de la actividad (activity_main2) con las variables de la clase.
2. **Validación de reconocimiento de voz**:
    - Comprueba si el dispositivo soporta esta funcionalidad, mostrando un mensaje en caso contrario.
3. **Configuración del RecognitionListener**:
    - Detecta eventos clave durante el reconocimiento de voz:
        - Inicio (**onBeginningOfSpeech**): Actualiza el texto indicando que está "Escuchando...".
        - Fin (**onEndOfSpeech**): Reinicia el reconocimiento si está activo.
        - Error (**onError**): Reactiva el reconocimiento automáticamente.
        - Resultados (**onResults**): Obtiene y almacena el texto reconocido en accumulatedText, mostrándolo en pantalla.

Acciones de los botones

1. **Botón de iniciar/detener (startStopButton)**:

- Alterna entre iniciar y detener el reconocimiento de voz, cambiando el texto del botón dinámicamente.
- Llama a startListening o stopListening según el estado actual.

2. **Botón de guardar (saveButton)**:
    - Verifica si hay texto para guardar.
    - Muestra un cuadro de diálogo para ingresar un título y confirma el guardado.

Manejo del reconocimiento de voz

1. **startListening**:
    - Solicita permisos de grabación si no están otorgados.
    - Configura un intent para iniciar el reconocimiento de voz en español (**es-MX**) con resultados parciales.
    - Cambia el estado de isListening a verdadero y actualiza el texto del botón.
2. **stopListening**:
    - Detiene el reconocimiento, actualiza el estado de isListening a falso y restaura el texto del botón.

Guardado de notas

La función **showTitleInputDialog** muestra un cuadro de diálogo con un campo de texto para ingresar el título de la nota. Valida que el título no esté vacío antes de:

1. Crear un **Intent** con el título y el contenido de la nota.
2. Finalizar la actividad y regresar el resultado a la actividad que la inició.

Botón de regresar

Un botón adicional con el ID **botonAtrasSegundaPantalla** permite cerrar la actividad y regresar al menú principal.

Activity_drawer_menu.kt:

En el siguiente código encontramos la definición de una clase llamada activity_drawer_menu, que extiende de AppCompatActivity. Esta clase tiene como propósito principal gestionar una actividad con un diseño de tipo *drawer menu*

Importaciones

Se incluyen importaciones esenciales que permiten trabajar con elementos de la interfaz gráfica y las funciones de compatibilidad del sistema:

- **enableEdgeToEdge:** Configura la actividad para usar un diseño de pantalla completa con compatibilidad moderna.
- **ViewCompat y WindowInsetsCompat:** Facilitan la gestión de márgenes y rellenos asociados con las barras del sistema, como la barra de estado y la de navegación.

Método onCreate

El método onCreate es el punto de entrada de la actividad. Dentro de él encontramos las siguientes operaciones:

1. **Habilitación del modo Edge-to-Edge:**
    - **enableEdgeToEdge():** Activa el diseño de borde a borde, eliminando los márgenes tradicionales y permitiendo que la actividad utilice todo el espacio disponible en la pantalla.
2. **Establecimiento del diseño de la actividad:**
    - **setContentView:** Asocia el archivo de diseño XML
    - **activity_drawer_menu:** con esta actividad, definiendo la interfaz gráfica que se mostrará al usuario.
3. **Aplicación de rellenos dinámicos a la vista principal:**
    - **ViewCompat.setOnApplyWindowInsetsListener:** Configura un escuchador en la vista principal, identificada por el ID main.
    - El escuchador se asegura de ajustar dinámicamente los rellenos de la vista para incluir los márgenes correspondientes a las barras del sistema:
        - **getInsets(WindowInsetsCompat.Type.systemBars()):** Obtiene los márgenes actuales de las barras del sistema (barra de estado, barra de navegación, etc.).

- **setPadding:** Aplica estos márgenes como relleno a los bordes izquierdo, superior, derecho e inferior de la vista principal.

Drawer_menu.kt:

La clase principal, llamada drawer_menu, tiene como objetivo principal gestionar un diseño asociado a un menú de tipo *drawer*. A continuación, se describen sus componentes y funcionamiento:

Importaciones

Se incluyen librerías esenciales para gestionar la actividad y sus interacciones con el sistema:

- **androidx.appcompat.app.AppCompatActivity:** Proporciona compatibilidad con características modernas de Android.
- **androidx.core.view.ViewCompat y androidx.core.view.WindowInsetsCompat:** Facilitan la manipulación de márgenes y rellenos dinámicos en función de las barras del sistema (barra de estado y barra de navegación).

Definición de la clase

La clase drawer_menu hereda de AppCompatActivity, lo que le permite gestionar el ciclo de vida de una actividad estándar en Android.

Método onCreate

El método onCreate, que se ejecuta al iniciar la actividad, incluye las siguientes operaciones clave:

1. **Configuración del diseño de la actividad:**
    - **setContentView(R.layout.activity_drawer_menu):** Establece la interfaz gráfica que utilizará esta actividad, vinculándola al archivo de diseño XML activity_drawer_menu.
2. **Ajuste de márgenes dinámicos:**
    - **ViewCompat.setOnApplyWindowInsetsListener:**
        - Configura un escuchador para la vista principal identificada por el ID main.
        - Este escuchador asegura que la vista principal se ajuste dinámicamente en función de las barras del sistema.
    - **getInsets(WindowInsetsCompat.Type.systemBars()):**
        - Obtiene los valores actuales de los márgenes de las barras del sistema (superior, inferior, izquierdo y derecho).

- **v.setPadding(...):**
    - Aplica estos márgenes como relleno a los bordes de la vista principal, garantizando que el contenido no sea solapado por elementos del sistema.

EditarNota.kt:

En el siguiente código encontramos primeramente la definición de una clase llamada **EditarNota**, que extiende de **AppCompatActivity**. Su propósito principal es permitir a los usuarios editar notas almacenadas en una base de datos de Firestore y guardar los cambios correspondientes. A continuación, se describe su estructura y funcionalidad:

1. Configuración inicial

- En el método **onCreate**, se define el flujo principal de la actividad:
    - **setContentView(R.layout.activity_editar_nota)**: Establece el diseño de la actividad utilizando el archivo XML asociado (**activity_editar_nota**).
    - **Inicialización de Firestore**:
        - **firestore = FirebaseFirestore.getInstance()**: Se obtiene una instancia de Firestore para interactuar con la base de datos.

2. Configuración del botón "Atrás"

- Se vincula el botón **botonAtrasSegundaPantalla** al evento de clic para cerrar la actividad actual y regresar a la anterior mediante **finish()**.

3. Obtención de datos de la nota

- Los datos de la nota (ID, título y contenido) se reciben a través de la intención que inició la actividad:
    - **intent.getStringExtra("noteId")**, **noteTitle**, y **noteContent**: Extraen la información enviada desde la actividad anterior.
- Los campos de entrada (**EditText**) para el título y contenido de la nota son inicializados con los valores recibidos, utilizando **setText()**.

4. Guardar los cambios

- El botón **saveButton** permite guardar los cambios realizados en la nota:
    - Los valores actualizados se obtienen de los campos de entrada.
    - Se valida que el **noteId** sea válido antes de proceder a guardar los cambios.
    - La función **updateNoteInFirestore** se llama para realizar la actualización en la base de datos.

5. Función para actualizar Firestore

- La función **updateNoteInFirestore** realiza las siguientes acciones:
    1. **Obtención del usuario autenticado**: Se verifica si el usuario actual está autenticado con Firebase Authentication.
    2. **Estructuración de los datos**:
        - Un mapa (**HashMap**) se crea con las claves "title" y "content" para representar los datos actualizados de la nota.
    3. **Actualización en Firestore**:
        - Se accede a la colección **users**, donde cada documento corresponde a un usuario. Dentro de la colección de notas del usuario actual, se busca el documento asociado al ID de la nota y se actualizan los datos utilizando **update()**.
        - En caso de éxito:
            - Se redirige al usuario a la actividad principal (**MainActivity**) usando un **Intent** con las banderas **FLAG_ACTIVITY_CLEAR_TOP** y **FLAG_ACTIVITY_NEW_TASK**, que eliminan cualquier actividad previa del stack.
            - Un mensaje confirma que la nota se modificó correctamente.
        - En caso de error:
            - Se muestra un mensaje indicando el problema.

LoginActivity.kt:

En el siguiente código se implementa la definición de la clase **LoginActivity**, que extiende de **AppCompatActivity**. Su objetivo principal es proporcionar una interfaz para que los usuarios puedan iniciar sesión en la aplicación utilizando Firebase Authentication y verificar la existencia del usuario en Firestore. A continuación, se detalla su estructura:

1. Configuración inicial

- En el método **onCreate**, se establece el diseño mediante **setContentView(R.layout.activity_login)**, que define la estructura visual de la pantalla de inicio de sesión.
- Se inicializan las instancias de Firebase Authentication (**auth**) y Firestore (**firestore**) para gestionar el proceso de autenticación y acceso a la base de datos.

2. Referencias a elementos del diseño

- Se obtienen las referencias de los componentes del diseño:

- **EditText**:
  - **edtEmail** para el correo electrónico.
  - **edtPassword** para la contraseña.
- **Button**:
  - **btnLogin** para el botón de iniciar sesión.
- **TextView**:
  - **tvRegister** para mostrar un texto interactivo que redirige a la actividad de registro.

3. Texto interactivo para el registro

- Se crea un texto interactivo utilizando **SpannableString**:
  - El texto es **"¿No tienes cuenta? Regístrate aquí"**.
  - Se define un **ClickableSpan** para que al hacer clic en la palabra **"aquí"**, se inicie la actividad **RegisterActivity** mediante un **Intent**.
  - La funcionalidad de clic se habilita utilizando **LinkMovementMethod.getInstance()**.

4. Evento del botón "Iniciar Sesión"

- Al presionar **btnLogin**, se obtienen los valores ingresados en los campos de correo y contraseña, aplicando **trim()** para eliminar espacios en blanco.
- Si algún campo está vacío, se muestra un mensaje con **Toast** pidiendo que se completen todos los campos.

5. Proceso de inicio de sesión con Firebase

- Se utiliza **auth.signInWithEmailAndPassword(email, password)** para autenticar al usuario:
  - **Si la autenticación es exitosa**:
    - Se obtiene el ID del usuario autenticado (**userId**) y se busca en la colección **users** de Firestore.
    - Si el documento correspondiente al usuario existe en Firestore:
      - Se muestra un mensaje de éxito y se redirige a **MainActivity** mediante un **Intent**.
      - Se finaliza la actividad actual utilizando **finish()**.
    - Si no se encuentra al usuario en Firestore:
      - Se muestra un mensaje de error y se cierra la sesión con **auth.signOut()**.

- **Si la autenticación falla**:
    - Se muestra un mensaje con el error específico utilizando **task.exception?.message**.

Note.kt:

En el siguiente código podemos ver la declaración de una **data class** llamada Note, que representa una nota con tres propiedades: id, title y content. Estas propiedades tienen los siguientes significados:

- **id**: Un identificador único para la nota.
- **title**: El título de la nota.
- **content**: El contenido de la nota.

Estas propiedades son definidas con los tipos **String** y son inicializadas mediante un constructor primario. Además, se incluye un **constructor vacío**, que es necesario para que la clase sea compatible con la base de datos de Firebase. Este constructor vacío permite crear una instancia de la clase sin pasar parámetros, lo cual es requerido por Firebase cuando los objetos se serializan y deserializan de la base de datos.

En resumen, la clase Note está diseñada para manejar notas dentro de la aplicación, facilitando su almacenamiento y recuperación de Firebase.

NoteAdapter.kt:

Este código implementa un **adaptador personalizado** para un RecyclerView en Android, llamado NoteAdapter. Este adaptador permite mostrar una lista de objetos de tipo Note y manejar interacciones con ellos. A continuación, se explica cada parte del código:

1. **Propiedades principales**:
    - notes: Una lista de objetos Note que representa las notas que se mostrarán en el RecyclerView.
    - onItemClick: Una función lambda que se ejecuta cuando se hace clic en un elemento de la lista, permitiendo manejar eventos de clic desde fuera del adaptador.
2. **Métodos principales**:
    - **onCreateViewHolder**: Este método infla la vista de un elemento de la lista utilizando un diseño predeterminado (simple_list_item_1) y crea un objeto NoteViewHolder para mantener una referencia a la vista.
    - **onBindViewHolder**: Se vincula un objeto Note de la lista a la vista correspondiente, estableciendo el texto del título y configurando el evento de clic para ese elemento. Al hacer clic, se llama a la función onItemClick con el objeto Note seleccionado.

- **getItemCount**: Devuelve la cantidad de elementos en la lista notes.
- **updateNotes**: Permite actualizar la lista de notas con un nuevo conjunto de datos. Llama a notifyDataSetChanged() para notificar al RecyclerView que los datos han cambiado y deben refrescarse.

3. **Clase interna NoteViewHolder**:
    - Define un ViewHolder que almacena referencias a los elementos de la vista del diseño del elemento de la lista.
    - Aquí, se utiliza un TextView (titleTextView) para mostrar el título de la nota.

NotesShareedPreferencs.kt:

Este código define una clase llamada NotesSharedPreferences que se encarga de guardar y cargar una lista de notas en el almacenamiento compartido (SharedPreferences) de Android utilizando la librería Gson para convertir entre objetos de tipo Note y su representación en formato JSON. A continuación, se explican los detalles principales del código:

1. **Propiedades y constructor**:
    - sharedPreferences: Obtiene la instancia de SharedPreferences que permite guardar datos de manera persistente en el dispositivo. Se usa en modo privado (MODE_PRIVATE) para que solo la aplicación pueda acceder a estos datos.
    - gson: Se instancia un objeto de la librería Gson para realizar las conversiones entre objetos Java y JSON.
2. **Constantes**:
    - PREFS_NAME: El nombre del archivo de preferencias donde se almacenarán los datos (en este caso, "notes_prefs").
    - NOTES_KEY: La clave utilizada para almacenar y recuperar la lista de notas en las preferencias.
3. **Métodos**:
    - **saveNotes(notes: List<Note>)**: Este método convierte una lista de objetos Note en una cadena JSON usando Gson (gson.toJson(notes)) y luego la guarda en SharedPreferences mediante el método putString(). Finalmente, se aplica el cambio con apply(), lo que asegura que los datos se guarden de manera asíncrona.
    - **loadNotes()**: Este método obtiene la cadena JSON almacenada en SharedPreferences mediante getString(). Si la cadena JSON no es null, se convierte nuevamente en una lista de objetos Note usando gson.fromJson(). Si no hay datos guardados, devuelve una lista vacía (emptyList()).

RegisterActivity.kt:

Este código corresponde a la actividad de registro de un usuario en la aplicación, utilizando Firebase Authentication para la creación de cuentas y Firestore para almacenar los datos del usuario. A continuación, se describe el flujo y los componentes principales del código:

1. **Inicialización de variables**:
    - Se inicializan los elementos del layout como EditText para el correo y la contraseña, y un Button para el registro. Además, se inicializan las instancias de FirebaseAuth y FirebaseFirestore para la autenticación y el almacenamiento de datos.
2. **Configuración de la interacción con el texto de "Iniciar sesión"**:
    - Se configura un TextView para redirigir a los usuarios a la pantalla de inicio de sesión (LoginActivity). Usando un SpannableString, se hace clickeable la frase "aquí" dentro del texto "¿Ya tienes cuenta? Inicia sesión aquí". Al hacer clic, se inicia la actividad de inicio de sesión.
3. **Registro del usuario con Firebase**:
    - Cuando el usuario presiona el botón de registro (btnRegister), se obtiene el correo y la contraseña ingresados. Si alguno de estos campos está vacío, se muestra un Toast solicitando completar la información.
    - Si ambos campos están completos, se utiliza FirebaseAuth para crear una cuenta con el correo y la contraseña proporcionados. Si la creación de la cuenta es exitosa, se procede a almacenar los datos del usuario en Firestore.
4. **Guardar los datos del usuario en Firestore**:
    - Una vez creado el usuario, se almacena su correo y contraseña en la colección users de Firestore utilizando el UID del usuario como el identificador del documento.
    - Se crea una subcolección llamada notes en el documento del usuario, y dentro de esta, se guarda una nota de ejemplo con un ID, título y contenido predefinidos.
5. **Redirección a la pantalla de inicio de sesión**:
    - Tras guardar exitosamente los datos del usuario y la nota de ejemplo, el código redirige a la pantalla de inicio de sesión (LoginActivity) y finaliza la actividad de registro, cerrando la pantalla de registro.
6. **Manejo de errores**:
    - Se incluyen bloques addOnFailureListener para manejar posibles errores en la creación de la cuenta, el almacenamiento de datos en Firestore y la creación de la nota de ejemplo, mostrando un Toast con el mensaje de error correspondiente.

VisualizarNota.kt:

Este código corresponde a una actividad en la aplicación, donde se permite visualizar y editar una nota específica, o eliminarla de la base de datos de Firestore. La actividad está diseñada para interactuar con Firebase para obtener y gestionar las notas de un usuario autenticado. A continuación, se describe el funcionamiento y las principales funcionalidades del código:

1. **Inicialización de variables y Firebase**:
    - Se inicializa una instancia de FirebaseFirestore para interactuar con la base de datos de Firestore.
    - Se obtiene información pasada a la actividad a través de un Intent, específicamente el ID, título y contenido de la nota, que se utilizarán para mostrar los detalles de la nota seleccionada.
2. **Configuración de la interfaz de usuario**:
    - Los valores de la nota (título y contenido) se asignan a los TextView correspondientes (noteTitleTextView y noteContentTextView), para que el usuario pueda verlos. Si no hay contenido disponible, se muestra un mensaje predeterminado.
    - Se configuran dos botones: uno para eliminar la nota (deleteButton) y otro para editarla (editButton).
3. **Acción de eliminar la nota**:
    - Al presionar el botón de eliminación, se verifica que el noteId no sea nulo. Si es válido, se llama a la función deleteNoteFromFirestore para eliminar la nota correspondiente de Firestore.
    - Si la eliminación es exitosa, se muestra un mensaje de confirmación (Toast) y se devuelve el ID de la nota eliminada a la actividad anterior mediante un Intent. Luego, se finaliza la actividad actual con finish().
    - Si ocurre un error durante la eliminación, se muestra un mensaje de error.
4. **Acción de editar la nota**:
    - Al presionar el botón de edición, se abre la actividad EditarNota pasando el ID, el título y el contenido de la nota a través de un Intent. Esto permite que el usuario pueda modificar los detalles de la nota en otra pantalla.
5. **Eliminación de la nota de Firestore**:
    - La función deleteNoteFromFirestore se encarga de eliminar la nota de la base de datos. Utiliza el UID del usuario autenticado para buscar la nota dentro de la colección notes de Firestore y eliminarla.
    - Si el usuario no está autenticado, se muestra un mensaje de error.

En resumen, esta actividad permite visualizar los detalles de una nota, editarla o eliminarla, interactuando con Firebase para almacenar y eliminar datos. La interfaz está diseñada para facilitar

al usuario la navegación entre las opciones de gestionar sus notas, manteniendo la interacción fluida y manejando adecuadamente los casos de error.

Conclusiones individuales

Kevin Aparicio: Podemos tener como conclusión que, a través de este proyecto, se ha logrado crear una plataforma eficiente para registrar y gestionar notas, integrando funcionalidades de Firebase para almacenar y editar información. El uso de Firebase Authentication y Firestore asegura que los datos se manejen de manera segura y accesible, mientras que la posibilidad de editar y eliminar notas aporta una gran flexibilidad para el usuario. Este proyecto refuerza la importancia de crear aplicaciones móviles prácticas que optimicen el tiempo y esfuerzo del usuario al registrar sus pensamientos.

Edwin Bolaños: El aprendizaje final de este proyecto es que la creación de una aplicación para gestionar notas no solo depende de la implementación de funciones básicas, sino también de asegurar una experiencia de usuario fluida y confiable. La integración de Firebase como backend permite una gestión de datos eficaz, mientras que las opciones de edición y eliminación proporcionan al usuario el control total sobre sus notas. Esta aplicación demuestra cómo el uso adecuado de las tecnologías móviles puede mejorar la organización personal.

Brandon Espinoza: Podemos concluir que este proyecto permite entender cómo la creación de una aplicación para gestión de notas puede ser tanto intuitiva como poderosa. La integración con Firebase no solo facilita el almacenamiento de notas, sino que también proporciona un entorno seguro y accesible para el usuario. El proyecto subraya la importancia de ofrecer opciones de personalización, como la edición y eliminación de notas, para que los usuarios puedan gestionar de manera eficiente sus ideas y recordatorios.

Joel Chaparro: El aprendizaje final de este proyecto es que, al desarrollar una aplicación de gestión de notas, se deben combinar tanto las funcionalidades prácticas como la experiencia de usuario. La implementación de Firestore y Firebase Authentication otorga a la aplicación la capacidad de manejar datos de manera segura, mientras que las funciones de edición y eliminación proporcionan un alto grado de flexibilidad. Este proyecto resalta cómo una simple aplicación de notas puede ser altamente funcional y valiosa cuando se integra correctamente con tecnologías móviles.