

“DAILYSYNC”

**PROYECTO DE (LÍNEA 3. ELABORACIÓN DE UNA APLICACIÓN
PARA DISPOSITIVO MÓVIL)**

**CICLO FORMATIVO DE GRADO SUPERIOR DE (DESARROLLO DE
APLICACIONES MULTIPLATAFORMA)**

**LARRY SEBASTIAN RODRIGUEZ PATIÑO
ALFONSO GARCIA**

Curso 2023-2024

ÍNDICE

1. Resumen.....	3
2. Abstract.....	5
3. Introducción.....	6
4. Descripción.....	8
4.1 Objetivos.....	8
5. Contenidos.....	10
5.1. Diagrama de Pantallas.....	10
5.2. Pantalla de Introducción:.....	10
5.3. Pantalla de Inicio de Sesión/Registro:.....	11
5.4. Pantalla Principal:.....	12
5.5. Fragmento de Notas:.....	12
5.6. Fragmento de Calendarios:.....	13
5.7. Fragmento de Ajustes:.....	14
6. Metodología y planificación.....	15
6.1. Metodología:.....	15
6.2. Planificación:.....	16
7. Desarrollo.....	19
7.1. Tecnologías usadas:.....	19
7.1.2. IDE (Entorno de desarrollo):.....	19
7.1.3. Lenguaje de programación:.....	19
7.1.4. Bases de datos:.....	20
7.1.5. Elementos gráficos, fuentes y colores:.....	20
7.2. Implementación del desarrollo:.....	21
7.2.1. Instalación del IDE:.....	21
7.2.2. Diseño de interfaces:.....	21
7.2.3 Diseño e implementación de las base de datos:.....	24
7.2.4. Desarrollo de la lógica de la aplicación:.....	25
8. Conclusiones y proyección a futuro.....	26
8.1. Conclusiones:.....	26
8.2. Proyección a futuro:.....	26
9. Bibliografía.....	27
10. Anexos.....	28

1. Resumen

Los móviles se han vuelto cada vez más indispensables en nuestras vidas. Si aprendemos a utilizarlos correctamente, pueden ser una herramienta indispensable en nuestras actividades diarias. Un ejemplo claro de esto es aprovechar el móvil como nuestro propio gestor y agenda personal, ayudándonos a tomar notas de nuestras ideas, establecer recordatorios y gestionar eficientemente nuestro tiempo en todos los aspectos de nuestra vida, de ahí nace la iniciativa de crear una aplicación que nos ayude a gestionar nuestra vida en diferentes aspectos.

DayliSync es una aplicación móvil para dispositivos Android desarrollada en Android Studio utilizando Java, con el fin de brindar una solución completa para la organización diaria de la vida personal. Su propósito principal es simplificar la gestión del tiempo y las tareas diarias, priorizando la facilidad de uso y la intuitividad para los usuarios.

La aplicación ofrece dos funciones principales: la creación de notas y la gestión de calendarios. La función de creación de notas permite a los usuarios capturar de manera rápida y sencilla ideas, tareas y recordatorios importantes, proporcionando un espacio digital organizado para almacenar información relevante. Por otro lado, el componente de calendario proporciona una herramienta completa para planificar y organizar eventos, citas, plazos, recordatorios y rutinas, facilitando la administración de compromisos diarios, semanales y mensuales.

Una característica destacada de DayliSync es su capacidad para crear un número ilimitado de calendarios, permitiendo a los usuarios organizar y planificar actividades en diversas áreas de sus vidas, como trabajo, familia, ocio, educación y deporte. Esto proporciona una organización personalizada adaptada a las necesidades individuales de cada usuario.

Además, la aplicación cuenta con una función de seguimiento del ciclo menstrual, que se suma a la versatilidad de los calendarios disponibles, ayudando a las usuarias a llevar un registro fácil y efectivo.

Palabras claves:

Smartphones, Gestión del tiempo, Productividad, Simplificación, Aplicación.

2. Abstract

"Smartphones have become increasingly indispensable in our lives. If utilized correctly, they can serve as invaluable tools in our daily activities. One clear example is using the mobile device as our personal manager and agenda, aiding in note-taking, setting reminders, and efficiently managing our time in all aspects of life. From this initiative arises the development of an application that can manage our calendars and allow us to write notes all in one place. DayliSync is a mobile application for Android devices developed in Android Studio using Java, aiming to provide a comprehensive solution for daily personal organization. Its main purpose is to simplify time management and daily tasks, prioritizing user-friendliness and intuitiveness. The application offers two main functions: note creation and calendar management. The note creation function allows users to quickly and easily capture ideas, tasks, and important reminders, providing an organized digital space to store relevant information. Additionally, the calendar component provides a comprehensive tool for planning and organizing events, appointments, deadlines, reminders, and routines, facilitating the management of daily, weekly, and monthly commitments. A standout feature of DayliSync is its ability to create an unlimited number of calendars, enabling users to organize and plan activities in various areas of their lives, such as work, family, leisure, education, and sports. This provides a personalized organization tailored to the individual needs of each user. Furthermore, the application includes a menstrual cycle tracking feature, adding to the versatility of the available calendars and helping users maintain an easy and effective record of their health and well-being. The development methodology focused on user experience, ensuring a clear, simple, and intuitive interface design. The results demonstrate that DayliSync is an effective tool that meets the demands of daily user organization, improving productivity and simplifying life management. In conclusion, DayliSync offers a satisfactory user experience and personalized organization adaptable to individual needs, becoming a valuable ally for day-to-day activities."

Keywords:

Smartphones, Android, development, Dayli tasks, Calendars

3. Introducción

Hoy en día, los teléfonos inteligentes se han vuelto indispensables en nuestra vida cotidiana, desempeñando un papel fundamental como herramientas multifuncionales que nos acompañan en diversas áreas de nuestra vida. El constante desarrollo de aplicaciones móviles ha transformado la manera en que llevamos a cabo nuestras actividades diarias, ofreciendo soluciones innovadoras que mejoran nuestra productividad y facilitan nuestras tareas en una amplia gama de áreas.

Entre la vasta variedad de aplicaciones disponibles en el mercado, encontramos una diversidad de categorías que abarcan desde redes sociales, herramientas de aprendizaje, entretenimiento, música, aplicaciones de fotografía, juegos hasta herramientas de productividad. A pesar de esta amplia oferta, aún existen carencias en cuanto a aplicaciones que integren múltiples funciones esenciales para la organización personal en un solo lugar.

Si bien existen aplicaciones de agenda, calendarios, notas y diarios, no hay muchas aplicaciones que implementen estas funcionalidades en una misma aplicación, que permita a los usuarios crear notas, crear y gestionar múltiples calendarios para diversos propósitos y llevar un registro detallado de la actividad diaria todo eso en una misma plataforma. Es aquí donde surge la necesidad de crear una aplicación móvil más completa que satisfaga esa necesidad de poder gestionar nuestra vida más eficaz y simple a la vez.

El objetivo principal de este proyecto es precisamente desarrollar una aplicación móvil para android que ofrezca una experiencia integral al usuario, sumando diversas funcionalidades en un solo entorno. Esta aplicación no solo proporcionará un espacio para crear y gestionar notas donde el usuario plasmará sus ideas, recordatorios y tareas, sino que también dará la posibilidad de crear y gestionar multitud de calendarios personalizables e independientes el uno del otro, cada uno para su propio propósito.

En la mayoría de las aplicaciones disponibles en la Play Store, no encontramos ninguna que nos permita llevar a cabo todas estas funciones en un único lugar. Por ejemplo, si necesitamos tener un calendario dedicado al trabajo y otro para recordatorio de cumpleaños de familiares y amigos, podríamos ingresar los eventos y horarios, pero todo se mostraría en un mismo calendario. Además, aunque existen numerosas aplicaciones para tomar notas, estas suelen centrarse únicamente en esa función. Con esta aplicación se tiene como objetivo unir varias funcionalidades en una sola plataforma, permitiendo al usuario organizar todas sus tareas en un único espacio, lo que en mi opinión es mucho más eficaz e intuitivo a la vez.

4. Descripción

La aplicación a desarrollar será una herramienta que reúna varias funcionalidades dentro de un mismo entorno para la organización personal. Deberá permitir crear calendarios para diferentes propósitos, dentro de los cuales se añadirán tareas y recordatorios. Además, la aplicación también tendrá un apartado de notas que permitirá crear, visualizar, editar y eliminar las mismas. Además, se podrá añadir las notas a algunos de los calendarios creados si así se desea. Será una plataforma diseñada para usuarios que buscan una solución completa y centralizada para gestionar sus tareas diarias, y les servirá como una agenda completa que tendrá todo lo necesario para planificar sus vidas en diferentes ámbitos.

Los usuarios podrán crear cuantos calendarios quieran, cada uno independiente para diferentes propósitos, como trabajo, eventos sociales, actividades familiares, o para cualquier función que desee darle el usuario. Esto les dará a los usuarios una visión clara y organizada de sus compromisos en distintos ámbitos. La aplicación también ofrecerá la posibilidad de visualizar en cada calendario los recordatorios o tareas que el usuario ponga, así como planificar avisos para que la aplicación le recuerde el día y momento que el usuario programe. Al terminar el proyecto, se espera que la aplicación esté totalmente funcional y cumpla con su objetivo de ayudar a los usuarios en su organización de una forma intuitiva y eficaz.

4.1 Objetivos

El objetivo del proyecto es crear una aplicación para dispositivos Android que sea altamente intuitiva, eficiente y fácil de usar, y que cumpla con los siguientes requisitos:

- Permitir el registro de usuarios: La aplicación debe permitir a los nuevos usuarios registrarse, almacenando sus credenciales y creando un perfil de usuario (correo electrónico y contraseña) que les permita iniciar sesión en cualquier dispositivo Android.
- Almacenamiento seguro de la información en una base de datos: Tanto el perfil del usuario como la información generada dentro de la aplicación deben ser guardados de forma segura en una base de datos. Esto garantizará que el

usuario pueda acceder a sus tareas, recordatorios y calendarios desde cualquier dispositivo Android.

- Posibilitar la creación, visualización y gestión de notas: La aplicación debe permitir al usuario crear notas y acceder a ellas fácilmente a través de una sección dedicada para este fin. Además, deberá ofrecer herramientas para la gestión y organización efectiva de las notas.
- Permitir la creación, visualización y gestión de calendarios: Los usuarios podrán crear y administrar calendarios de manera independiente y personalizada, cada uno con su propia sección dedicada. Esto les permitirá organizar sus eventos y compromisos de manera eficiente.

5. Contenidos

La aplicación cuenta con una variedad de funciones destinadas a cumplir la tarea de ayudar a gestionar el tiempo de sus usuarios, proporcionando herramientas útiles para organizar las actividades diarias del usuario.

A continuación se mostraran las diferentes pantallas de la aplicación y una breve explicación de su funcionamiento

5.1. Diagrama de Pantallas

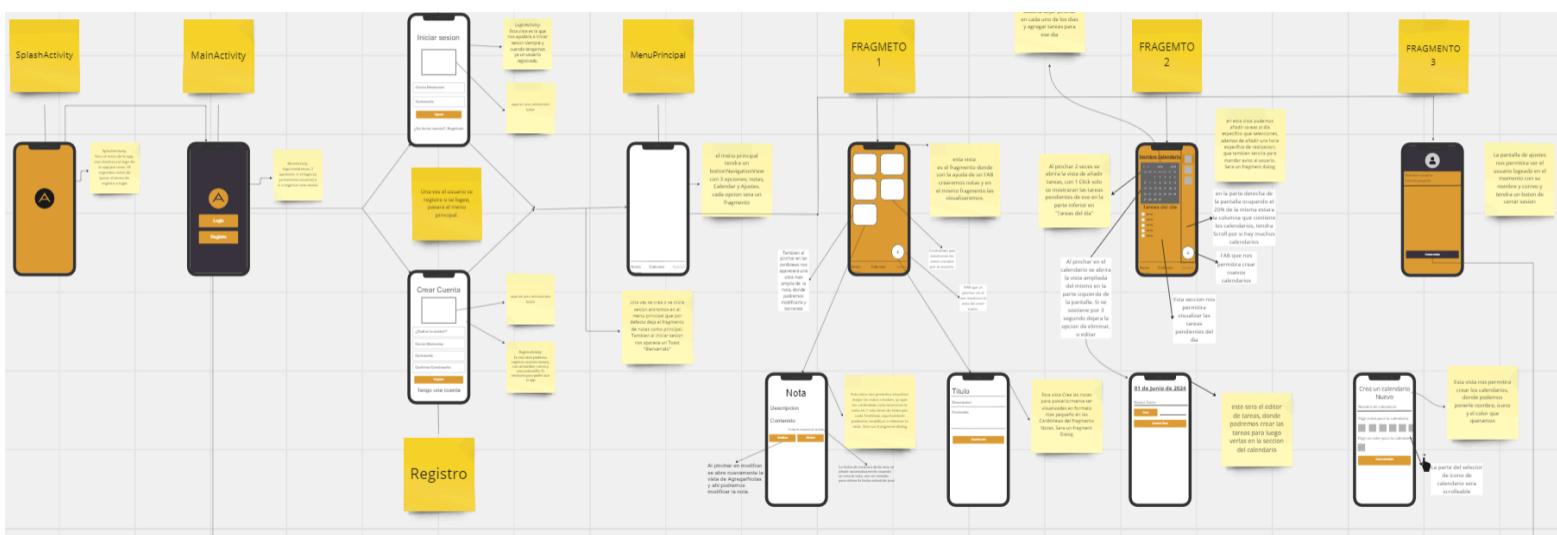


Imagen 1. Flujo de pantallas

5.2. Pantalla de Introducción:

Esta pantalla da la bienvenida al usuario y proporciona una visión del logo de DailySync durante 10 segundos, este logo se ha implementado con un Lottie obtenido de Lottiefiles [1]. Esta pantalla sirve como punto de entrada para la ejecución de la aplicación.

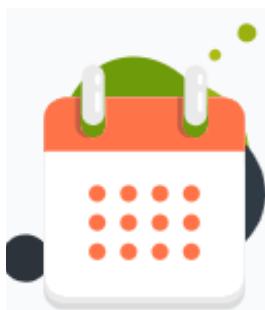


Imagen 2. Logo App



Imagen 3. Pantalla SplashActivity

5.3. Pantalla de Inicio de Sesión/Registro:

En esta pantalla, los usuarios tienen la opción de iniciar sesión si ya tienen una cuenta o registrarse si son nuevos en la aplicación. Lo primero que se les presentará será la pantalla de MainActivity que tiene 2 botones de Iniciar sesión y Registro. También contendrá el logo Lottie de la aplicación. Dependiendo de la elección del usuario, serán dirigidos a iniciar sesión que permitirá al usuario que ya tiene una cuenta registrada en la base de datos, autenticarse y poder acceder a las funciones de la aplicación. La otra opción es registrar un usuario nuevo con nombre, correo y contraseña. Ambas vistas también tendrán una animación Lottie que pretende hacer más atractiva la interfaz de usuario.



Imagen 4. Pantallas Main, Registro y Login

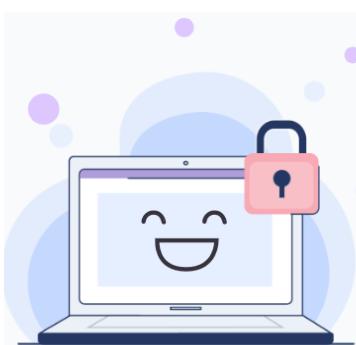


Imagen 5. Lottie de pantalla registro



Imagen 6. Lottie de pantalla Login

5.4. Pantalla Principal:

La pantalla principal se divide en tres fragmentos principales: Notas, Calendarios y Ajustes, a ellos se podrá acceder mediante una barra de navegación inferior que tendrá 3 botones, 1 para cada pantalla. Cada fragmento proporciona acceso rápido a las funciones clave de la aplicación



Imagen 7. Pantalla MenuPrincipal

5.5. Fragmento de Notas:

En esta sección, los usuarios pueden crear, modificar, eliminar y visualizar sus notas. El fragmento tendrá dentro un botón de creación de notas que abrirá la vista de agregar notas, el usuario podrá poner un título, descripción y contenido a la nota y al crearse esta nota esta se añadirá al fragmento en forma de tarjetas. Una vez aquí el usuario podrá ver la nota en pantalla completa al seleccionarla, aquí podrá también modificarla o eliminarla.

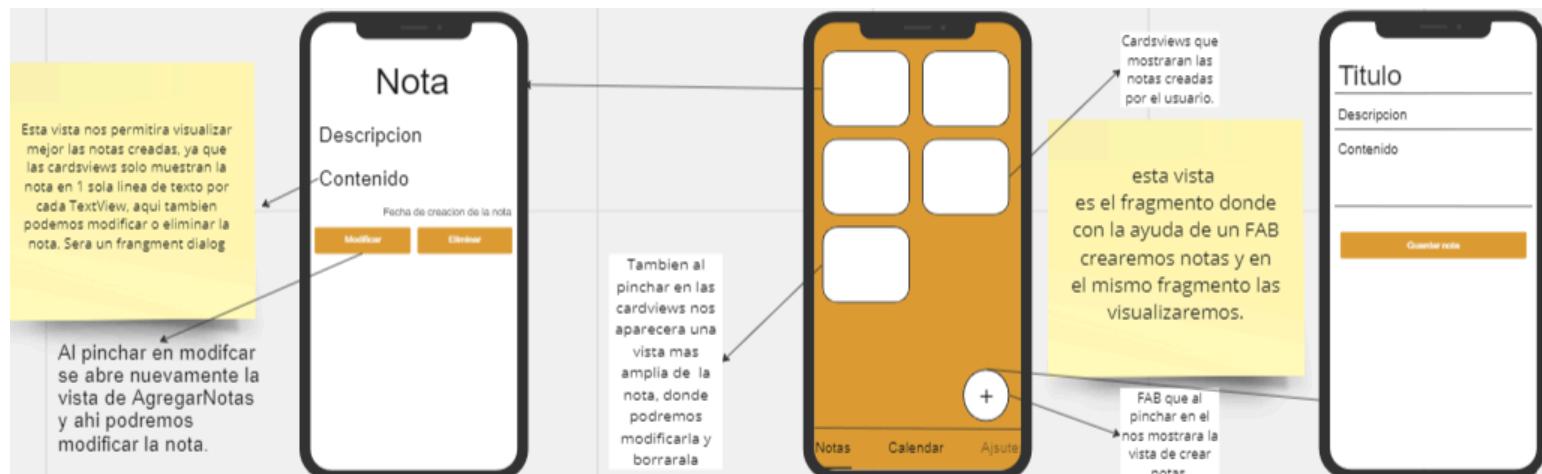


Imagen 8. Pantallas notas, Agregar notas y vista ampliada notas

5.6. Fragmento de Calendarios:

Aquí, los usuarios pueden crear, modificar, eliminar y visualizar sus calendarios. Pueden agregar eventos, establecer recordatorios y gestionar sus horarios de manera eficiente. De la misma forma que en notas el usuario dispondrá de un botón de creación de calendarios que mostrará una nueva vista donde allí el usuario podrá personalizar sus calendarios poniendo un nombre, eligiendo un ícono y un color. Luego de creados se mostrará en el fragmento en la parte derecha de la pantalla el ícono del calendario que el usuario creó. Al seleccionar cualquiera de los íconos de calendarios que haya este mostrará su nombre y vista de calendario en la parte izquierda de la pantalla, aquí el usuario podrá crear eventos, recordatorios y tareas para un día en específico. Al posicionarse sobre un día abrirá una nueva pantalla que será la encargada de la creación de las nuevas tareas del día. Ya por último estas tareas se podrán visualizar debajo del calendario y aparecerán en el día que se han creado. La aplicación también mandará aviso al usuario el dia y hora que él haya puesto una tarea o recordatorio



Imagen 9. Pantallas Calendario, agregar calendario y agregar tareas

5.7. Fragmento de Ajustes:

En este fragmento, los usuarios pueden acceder a su información de inicio de sesión, como su nombre de usuario y dirección de correo electrónico. También tienen la posibilidad de cerrar sesión.



Imagen 10. Pantalla ajustes

6. Metodología y planificación

6.1. Metodología:

La metodología de desarrollo se basa en combinar los enfoques de Scrum y Kanban en el desarrollo de la aplicación de Android. Consta de las siguientes fases:

Fase 1: Elección del tipo de aplicación a desarrollar:

se debe seleccionar cuidadosamente el tipo de aplicación que se va a desarrollar. Dado el amplio tipo de aplicaciones que utilizamos en nuestra vida cotidiana para diversos propósitos, es esencial determinar qué problema resolverá nuestra aplicación o qué valor añadido aportará a las aplicaciones existentes que abordan ese propósito.

Fase 2: Planificación y diseño inicial:

En esta fase se lleva a cabo la planificación específica del proyecto estableciendo los objetivos y los recursos necesarios para el desarrollo de la aplicación, es aquí también donde se da inicio a los primeros bocetos de como se verá la aplicación, qué funciones tendrán, como será su diseño de interfaz gráfica, como serán los flujos de navegación, de cuantas vistas consta la aplicación y cómo será su funcionamiento en sí.

Fase 3: Elección de tecnologías a usar:

En esta etapa, se procede a la selección de las tecnologías que se utilizarán en el proyecto. Se elige el lenguaje de programación, el IDE para el desarrollo, la base de datos, los métodos de autenticación de usuarios, las animaciones y la versión mínima de Android en la que funcionará la aplicación.

Fase 4: Desarrollo:

Aquí se da inicio al desarrollo de la aplicación en base a las fases anteriores, dando vida a los bocetos y diagramas que mostraban la planificación de la app, es aquí donde se le da vida a la aplicación. Aquí se aplicara la lógica para que la aplicación cumpla con sus objetivos

Fase 5: Pruebas y evaluación:

En esta fase, se llevan a cabo pruebas para asegurar la calidad y el correcto funcionamiento de la aplicación desarrollada. Se realizan diferentes tipos de pruebas, incluyendo pruebas de funcionalidad, rendimiento, usabilidad, compatibilidad y seguridad.

Las pruebas de funcionalidad se centran en verificar que todas las características y funciones de la aplicación funcionen como se esperaba de acuerdo a los requisitos establecidos. Las pruebas de rendimiento se encargan de evaluar el rendimiento de la aplicación bajo diferentes condiciones de carga y uso.

6.2. Planificación:

Para asegurar el desarrollo exitoso de mi proyecto de fin de grado, es esencial realizar una buena planificación del tiempo asignado a cada tarea, teniendo en cuenta su relevancia y complejidad. En primer lugar, elaboré un plan de trabajo para clarificar las diferentes etapas de desarrollo de la aplicación y establecer límites de tiempo para su finalización, como se detalla en los anexos 1, 2 y 3.

Posteriormente, investigué la documentación de Android para identificar los requisitos necesarios antes de iniciar el desarrollo de la aplicación. Utilicé la página oficial de Android Studio [2]. Para acceder a la información sobre la instalación del IDE requerido, instalándose (Android Studio Hedgehog // 2023.1.1). También es aquí donde se eligen los lenguajes de programación para el desarrollo en este caso se eligió java y de base de datos SQL, como método de autenticación de usuarios investigando vi que la una de las formas más eficaces para esta tarea es Firebase. Firebase es una plataforma de desarrollo de aplicaciones móviles y web creada por Google. Proporciona una variedad de servicios y herramientas para ayudar a los desarrolladores a construir, mejorar y hacer crecer sus aplicaciones de forma más rápida y eficiente. Algunos de los servicios que ofrece Firebase incluyen alojamiento

web, bases de datos en tiempo real, autenticación de usuarios, mensajería en la nube, análisis de aplicaciones y mucho más [3].

Luego procedí a revisar los requisitos de la aplicación, evaluando las funciones deseadas, el flujo de información entre las diferentes vistas y el flujo de datos. También establecí las expectativas sobre el resultado final de la aplicación. Durante esta etapa, analicé diversas aplicaciones como referencia, entre las cuales se encuentra el calendario de Google Calendar [4]. También hay un planificador semanal que se encuentra en la tienda de aplicación de google Play store[5], probé varios días algunas de estas aplicaciones y empecé a idear las mejoras que haría ya en mi propia aplicación.

Luego de tener el entorno de desarrollo de android y los requisitos que quería para mi app se empieza con el análisis de la aplicación. Se empieza con un boceto de lo que se quiere realizar para ello conté con la ayuda de un programa para realizar bocetos, Miro[6]. “Miro es una pizarra digital colaborativa en línea, que puede ser usada para la investigación, la ideación, la creación de lluvias de ideas, mapas mentales y una variedad de otras actividades colaborativas.”[7].

Ya teniendo una clara idea de la aplicación a desarrollar se empieza a desarrollar la interfaz gráfica se eligen los colores y fuentes y se empieza a desarrollar y dar vida a la app. Aquí en esta fase se trabajan las vistas, la relación que hay entre ellas, como se comunicaran con el usuario y como se presentará al usuario la app.

Una vez terminado el aspecto de la interfaz gráfica se pasa al desarrollo de la lógica que hará que la app tome forma y funcione, es una de las partes que más conlleva trabajo debido a la dificultad de hacer que la lógica que se pensó para que la app funcione se traduzca al código y realmente haga lo que esperamos que haga, aquí se inicia dar la funciones a cada una de las vistas, se procede a implementar los flujos de comunicación entre las diversas vistas, las base de datos y métodos de autenticación.

Por último se realiza una revisión exhaustiva de todos los aspectos técnicos de la app. Durante esta fase final, se evalúan los flujos entre vistas para garantizar una experiencia de usuario fluida y coherente. Además, se analiza el tiempo de respuesta de la aplicación para asegurar un rendimiento óptimo en diferentes condiciones de uso.

Finalmente, se elabora la documentación completa de la aplicación, que incluye manuales de usuario, guías de instalación y cualquier otra información relevante para su uso y mantenimiento. Esta documentación es fundamental para facilitar el soporte técnico y la gestión de la aplicación en el futuro.

7. Desarrollo

7.1. Tecnologías usadas:

Para el desarrollo de la aplicación fue necesario la implementación de varias tecnologías entre ellas están entornos de desarrollo, bases de datos, lenguajes de programación, elementos gráficos, fuentes y colores.

A continuación se describirán cada una de las partes anteriores necesarias para el desarrollo de la app:

7.1.2. IDE (Entorno de desarrollo):

Para el desarrollo de una app de android se pueden usar diferentes tipos de IDE que nos pueden ayudar en este propósito. Yo he usado Android Studio en su versión Hedgehog // 2023.1.1. Es el IDE oficial recomendado por google y en lo personal me gusta trabajar en este entorno.

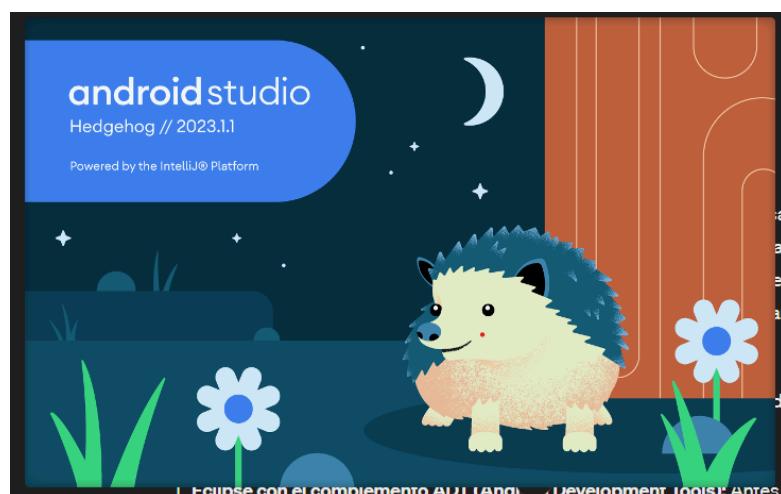


Imagen 10. Versión de android

7.1.3. Lenguaje de programación:

Necesario para la lógica de la app, tenemos 2 opciones en Android Studio, Java y Kotlin. El lenguaje recomendado por Google es Kotlin, debido a que es un lenguaje más moderno. “Si has oído hablar de Kotlin seguramente sea en el desarrollo de Android, pues desde el año 2019 Google se ha declarado Kotlin First es decir, los nuevos desarrollos se harán para Kotlin y una vez terminados se implementarán en

Java. Esto hizo que el sector móvil se diese cuenta de la importancia de dicho lenguaje y se centrara en él. Es por ello que actualmente más del 80% de las 1000 apps más populares para Android usan Kotlin.”[8].

Sin embargo dado las ventajas que ofrecía Kotlin me decidí por desarrollarlo en Java debido a que ha sido el lenguaje de programación que hemos visto durante el curso y con el que he estado aprendiendo a desarrollar.



Imagen 11. Logo de java

7.1.4. Bases de datos:

Para la implementación de las bases de datos use 2 opciones una base de datos para los usuario, autenticación de los mismos y para la guardar las notas creadas por el usuario, esta tarea la hice usando Firebase, para el guardado de calendarios y tareas use SQLite. Quise combinar ambas opciones porque ambas me ofrecían beneficios para el correcto funcionamiento de la app y son muy fácil de integrarlas en la misma.



Imagen 12. Logo Firebase



Imagen 13. Logo SQLite

7.1.5. Elementos gráficos, fuentes y colores:

Para los elementos gráficos use Canva. “Canva es una plataforma de diseño y comunicación visual que salió al mercado en 2013 y cuya misión es poner el poder del diseño al alcance de todo el mundo, para que cualquier persona pueda diseñar lo que quiera y publicarlo donde quiera.”[9]. Su facilidad para cualquier elemento gráfico ayuda mucho a la hora de crear componentes que en mi caso fueron el logo e iconos. También use Lotties para dar más intuitividad y estética a algunas vistas.

Para implementar los lotties hay que seguir algunos pasos para acoplar está al proyecto la misma página de LottieFiles muestra los pasos de como hacerlo[10].

Para la fuente de mi aplicación use Oswald Regular, que tome de Google Fonts[11]. Finalmente para los colores use también canva para obtener la paleta de colores de la app.



Imagen 14. Logo LottieFiles



Imagen 15. Logo Google Fonts



Imagen 16. Logo Canva

7.2. Implementación del desarrollo:

7.2.1. Instalación del IDE:

Para instalar el Android Studio se debe ver la documentación de instalación oficial, comprobar los requisitos mínimos que establece el entorno para poder funcionar en el ordenador y finalmente proceder a la instalación. Como también trabajaré con Java se procede a instalar el JDK 20 de java necesario para el desarrollo de aplicaciones con Java, para esto también se sigue la documentación oficial de Oracle[12]. Una vez instalados ya se puede empezar a desarrollar la aplicación Android.

7.2.2. Diseño de interfaces:

El diseño de la aplicación consta de 12 vistas diferentes y con las que el usuario interactuara en algún momento. Consta de 7 Activitys y 6 Fragmentos. A continuación se describe la relación entre estas vistas y que función cumple cada una.

- SplashActivity es donde la aplicación se inicia, da la bienvenida al usuario y se encarga también de la primera verificación de la autenticación del usuario. Se ha usado una activity porque es la primera pantalla que el usuario ve al abrir la aplicación, proporcionando una experiencia inicial agradable y

mostrando que la aplicación está cargando. Además, permite realizar cualquier inicialización necesaria, como verificar la autenticación del usuario, antes de pasar a la pantalla principal de la aplicación.

- MainActivity esta vista es donde el usuario si o si debe autenticarse bien sea haciendo inicio de sesión o registrándose en la app, esta vista tiene los botones de entrada a iniciar sesión o login.
- RegistroActivity aquí el usuario podrá registrar una cuenta nueva con nombre, correo y contraseña, esta vista maneja el registro de nuevos usuarios en la base de datos de Firebase. es la encargada de registrar correctamente los nuevos usuario y entrada al menú principal de app.
- LoginActivity esta actividad es donde si el usuario ya tiene una cuenta registrada correctamente en la base de datos podrá iniciar sesion, esta vista autentica al usuario para verificar si tiene los permisos para ingresar al menú principal de la app, para esta tarea se usa el servicio de firebase Authentication[13] que ofrece métodos eficaces y sencillos para ello.
- MenuPrincipal aquí solo accede el usuario logueado o registrado y autenticado su inicio de sesión, esta vista ofrece una barra de navegación inferior con las 3 opciones de herramientas que son las funciones esenciales de app. Esta vista solamente es el contenedor de los 3 fragmentos principales de la app, cada fragmento es un botón y se muestra en la pantalla de la actividad principal.
- FragmentoNotas este ayuda al usuario a la creación de notas, aquí también usaremos Database Firebase[14]. Para guardar en tiempo real las notas de los usuarios, esta es uan base de datos NoSql que permite almacenar en la nube datos y sincronizarlos en tiempo real, es muy fácil de implementar y es esencial en esta parte de la aplicación para que dé fluidez a la creación, edición y eliminación de las notas.
 - ActivityAgregarNota esta actividad es la encargada de la creación de las notas y lleva la lógica necesaria para guardarlas en la base de datos
 - FragmentoNota una vez creada las notas, aquí se podrán ver de forma más amplia ocupando toda la pantalla, también es posible modificar y eliminar las notas.

- FragmentoHome esta vista cumple con 3 tareas, muestra los iconos de calendarios, los calendarios y las tareas que se van asignando a los días del calendario. Para guardar los calendarios creados y las tareas se ha empleado una base de datos Sqlite que guarda estos calendarios y dentro de ellos las tareas, también con la ayuda de firebase podemos guardarlos con la autenticación del usuario.
 - ActivityAgregarCalendario esta actividad es la encargada de la creación de los calendarios con los requisitos que se pide para ello y luego de subirlos a la base de datos.
 - FragmentCalendario este fragmento es el calendario en sí, con sus días y los meses del año, esta vista se muestra en el fragmentoCalendario y es donde el usuario interactúa en él y crea sus tareas.
 - fragmentoTareas esta actividad se abre al dar 2 clicks en un dia del calendario es la encargada de la creación de las tareas y de subirlas a la base de datos.
- FragmentoAjustes este fragmento muestra al usuario sus datos de inicio de sesión y también del cierre de sesión.
- Diagrama de pantallas:

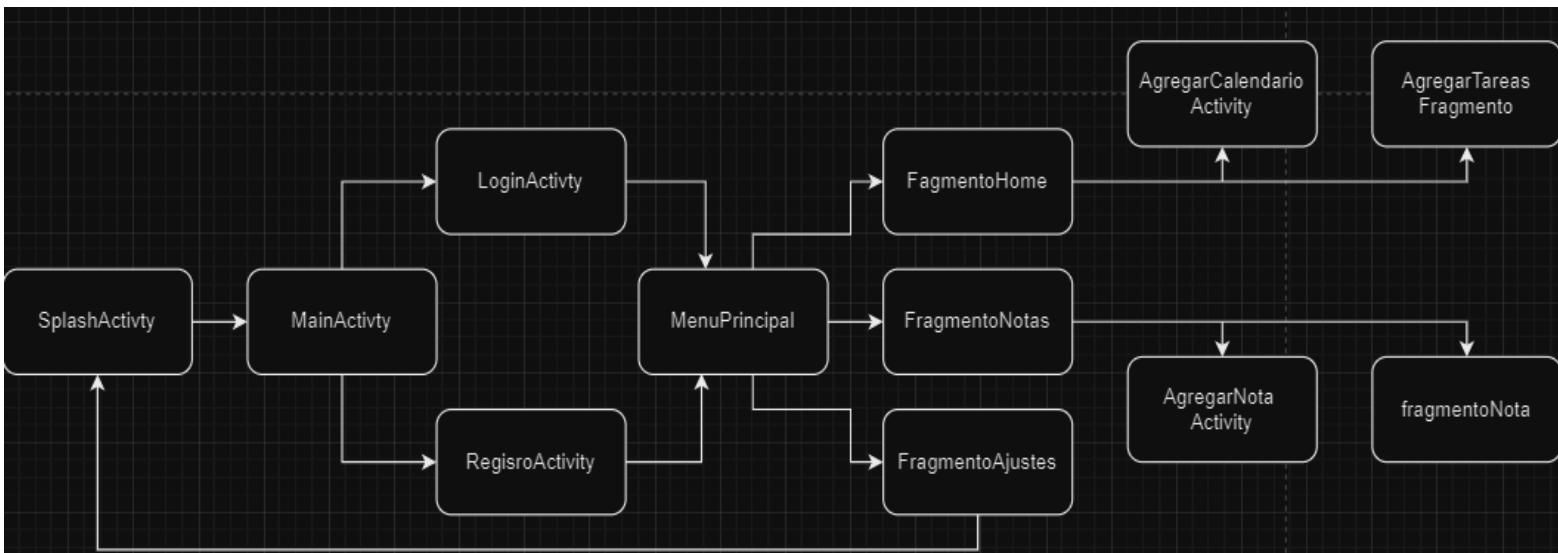


Imagen 17. Diagrama de pantallas

7.2.3 Diseño e implementación de las base de datos:

Se ha usado Firebase y Sqlite para diferentes tareas, ambas ofrecen sus propias ventajas y desventajas y es por ello que se ha implementado ambas para sacar el máximo provecho de las dos, A continuación se explicara su implementación y su diseño de cada una.

- Firebase: Para implementar Firebase en nuestro proyecto se debe crear una cuenta en este sitio y luego ver la documentación[15]. Para poder implementar este framework al proyecto de android se debe primero crear el proyecto en firebase y luego agregar este a nuestra app añadiendo los archivos de configuración de los servicios de Google en el archivo de Gradle project y finalmente en el Gradle módulo de app se agregara la implementación que queremos, en la app se han usado Authentication y Real time Database, luego con la documentación de Firebase y un video de como implementarlo[16] pude crear las base de datos de los usuarios y de las notas usando la clase objeto Usuarios con los atributos (nombre, correo, contraseña y uid) y sus métodos getters y setters que ayuda a encapsular los datos relacionados con el usuario. Esto facilita la manipulación de los datos del usuario en el contexto de la aplicación, como la creación, actualización y recuperación de información del usuario en la base de datos. Y lo mismo para la clase objeto Notas con los atributos (id, titulo, descripcion ,contenido y fecha).
- SQLite: Para implementar esta base de datos es un poco más sencillo. Para crear esta base de datos simplemente es creando una clase que extienda de SQLiteOpenHelper, esta clase nos ayuda a la creación y actualización de la base de datos, se usa el lenguaje de SQL para la creación, consultas y actualizaciones. En esta base de datos en mi app cumple con la tarea de guardar y actualizar todos los calendarios, también guardar las tareas de los mismos usando sus respectivas clases objeto. Al crear los calendarios se creará la tabla calendarios que almacenará toda la información de los calendarios creados (nombre, icono, color y id) posteriormente esta información se mostrará en el fragmentoHome. En la creación de tareas se creará una tabla tareas que guardará la información de las tareas que el

usuario cree (id, idCalendarios, descripción y fecha) esta tabla lleva también el id del calendario porque las tareas que se creen en un calendario en específico se deben guardar en ese mismo calendario.

7.2.4. Desarrollo de la lógica de la aplicación:

Aquí en esta fase, se lleva a cabo la implementación de la lógica que permite que la aplicación funcione de manera efectiva y cumpla con los requisitos especificados. Se abordaron las siguientes áreas:

- Interfaz de Usuario y Experiencia de Usuario:

Se diseñaron interfaces intuitivas y atractivas para una experiencia de usuario fluida y agradable centradas en el usuario para organizar y presentar la información de manera clara y accesible.

- Gestión de Usuarios:

Se desarrollaron funciones para el registro y autenticación de usuarios utilizando Firebase Authentication. Esto incluyó la validación de campos de entrada, el manejo de errores y la generación de tokens de autenticación seguros.

- Gestión de Notas:

Aquí se crearon mecanismos para la creación, edición y eliminación de notas, utilizando Firebase Realtime Database para almacenar y sincronizar los datos en tiempo real. Se diseñó una interfaz de usuario que permite a los usuarios interactuar de manera eficiente con sus notas, para la gestión de la información.

- Gestión de Calendarios y Tareas:

Se implementan funcionalidades para la creación y administración de calendarios, así como la asignación de tareas a fechas específicas. Se utilizó SQLite para el almacenamiento local de datos relacionados con calendarios y tareas, lo que permite un acceso rápido a la información incluso sin conexión a Internet.

8. Conclusiones y proyección a futuro

8.1. Conclusiones:

Durante el desarrollo de la aplicación diseñada para la gestión del tiempo personal, se ha concluido, tras diversas pruebas en varios dispositivos Android, tanto físicos como emuladores, que la aplicación se adapta de manera óptima a cualquier pantalla. La interfaz resulta legible, intuitiva y atractiva para el usuario.

En el apartado de notas, la aplicación cumple con su objetivo de facilitar la creación y gestión de notas para el usuario, sin presentar ningún tipo de problema significativo. Respecto a los calendarios, la aplicación permite la creación de estos y la asignación de tareas, aunque se han enfrentado limitaciones técnicas y de recursos. A pesar de ello, la aplicación logra cumplir con todas sus funciones, con excepción de la implementación del seguimiento del ciclo menstrual, que no pudo ser incorporado debido a restricciones de tiempo.

En conclusión, la aplicación es una herramienta eficaz para la toma de notas, gestión de calendarios y tareas, ofreciendo un funcionamiento fluido y una interfaz gráfica fácil de entender y atractiva. Su mayor aporte radica en proporcionar una solución integral para la gestión personal en una sola plataforma, eliminando la necesidad de utilizar múltiples aplicaciones.

8.2. Proyección a futuro:

Se contempla una serie de mejoras de la app en la parte de lógica y de interfaz gráfica, terminar de implementar la función de ciclo menstrual y añadir algunas nuevas funciones. Como principal nuevas mejoras sería implementar en el apartado de ajustes funciones que permitan al usuario personalizar la aplicación a su gusto, añadir un apartado que muestre un seguimiento de las tareas del usuario para saber si las ha cumplido o no, así como de enseñar gráficas de sus tareas en diferentes ámbitos. Otra mejora significativa sería poder hacer seguimiento al estado de ánimo del usuario donde diariamente el usuario podrá mediante emojis mostrar su estado de ánimo del día para así a final de mes o semana el usuario puede ver como estuvo su estado de ánimo.

9. Bibliografía

- [1].“Lotties”, Airbnb, (2024). Lottie. Recuperado de <https://airbnb.io/lottie/#/>
- [2].“Android Studio”, (2024). Android Developers.4. Desarrollar aplicaciones para Android. Recuperado de <https://developer.android.com/develop?hl=es-419>
- [3].“Firebase”, (2024). Qué es Firebase. Recuperado de <https://firebase.google.com/what-is-firebase>
- [4].“Google Calendar”, Google LLC. (2024). Google Calendar. Recuperado de <https://calendar.google.com>
- [5].“Play Store”, Google LLC. (2024). Google Play Store. Recuperado de <https://play.google.com/store>
- [6].“Miro”, Miro.com. (2024). Recuperado de <https://miro.com/es/>
- [7].“Universidad del Pacífico.”, (2024). Miro: Guía de uso. Recuperado de https://edutic.up.edu.pe/docs/guia_miro.pdf
- [8].“Kotlin”, OpenWebinars. (2024). ¿Qué es Kotlin? Recuperado de <https://openwebinars.net/blog/que-es->
- [9].“Canva”, (2024). Acerca de Canva. Recuperado de https://www.canva.com/es_es/about/
- [10].“LottieFiles”, (2024). Getting started with Lottie animations in Android app. Recuperado de <https://lottiefiles.com/blog/working-with-lottie-animations/getting-started-with-lottie-animations-in-android-app-es/>
- [11].“Google Fonts”, (2024). Recuperado de <https://fonts.google.com/>
- [12].“Oracle”, (2024). Descargas de Java SE y JDK. Recuperado de <https://www.oracle.com/java/technologies/downloads/#javasejdk>
- [13]. “Firebase Authentication”, Google. 2024. Firebase Authentication. Recuperado de <https://firebase.google.com/products/auth?hl=es-419>
- [14]“Database Firebase”, Google. 2024. Firebase Realtime Database. Recuperado de <https://firebase.google.com/docs/database?hl=es-419>
- [15]“Implementacion Firebase con Android”, Google. 2024. Configuración de Firebase para Android. Recuperado de <https://firebase.google.com/docs/android/setup?hl=es>
- [16]“MoureDev by Brais Moure. (2021). FIREBASE ANDROID 🔥 Curso ANDROID STUDIO desde CERO [Video]. YouTube. Recuperado de <https://www.youtube.com/watch?v=KYPc7CAYJ0w>”

10. Anexos

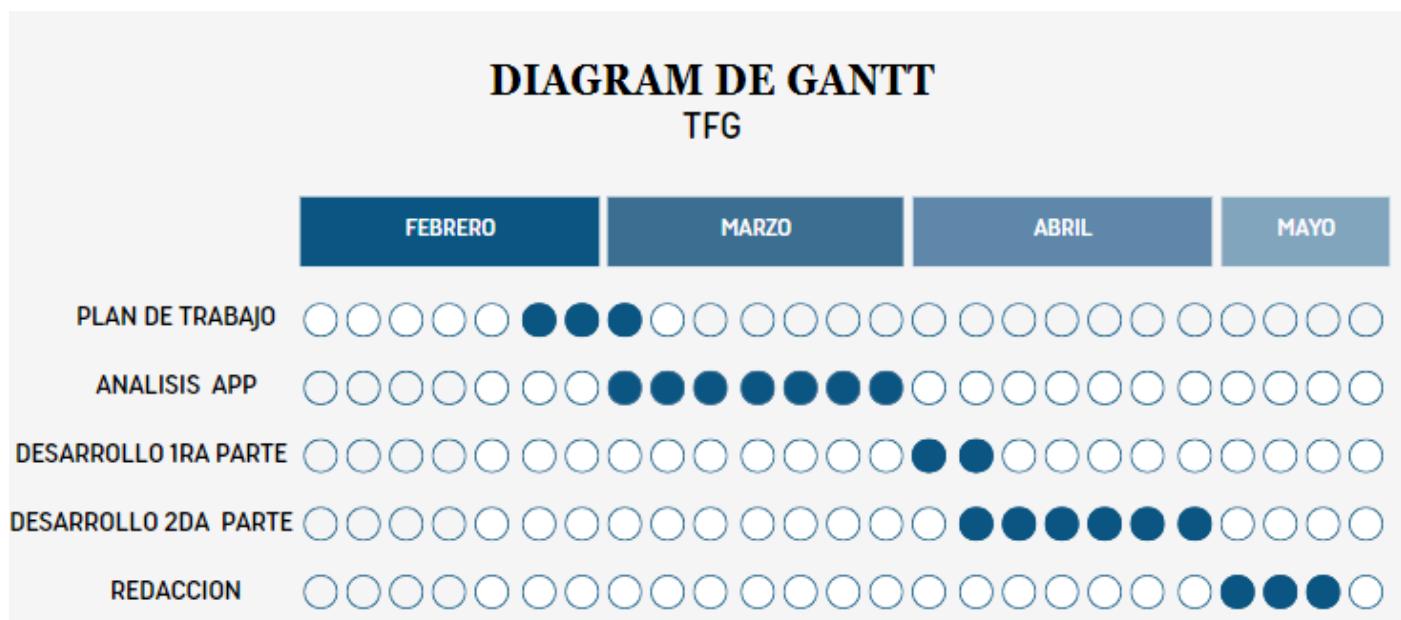
Anexo 1

Tabla planificación del proyecto

Planificación proyecto		
Entregas	Fecha Inicio	Fecha fin
Primer entrega	25/02/2024	05/03/2024
Segunda entrega	15/03/2024	09/04/2024
Entrega final	10/04/2024	05/05/2024

Anexo 2

Diagrama de Gantt



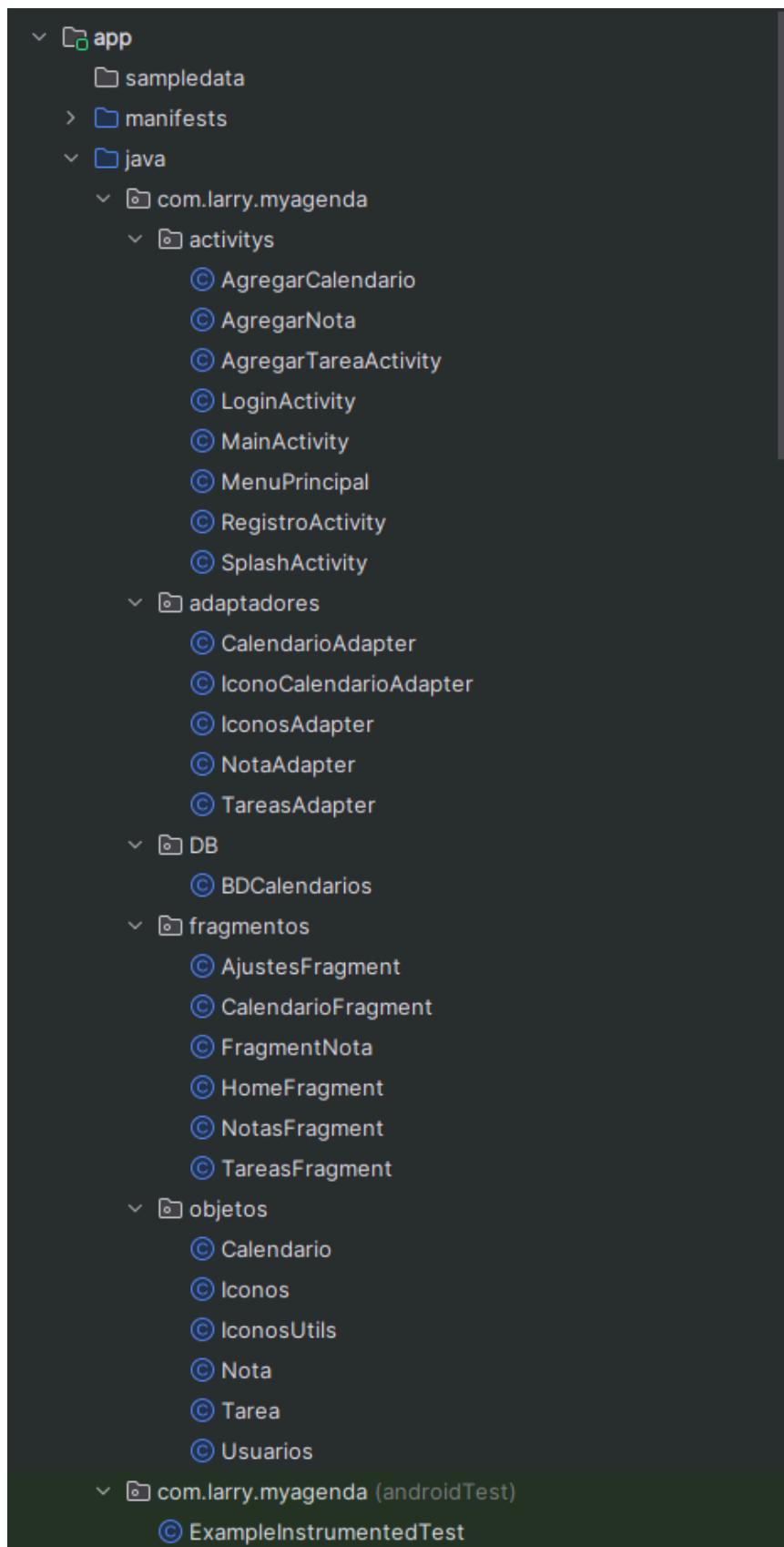
Anexo 3

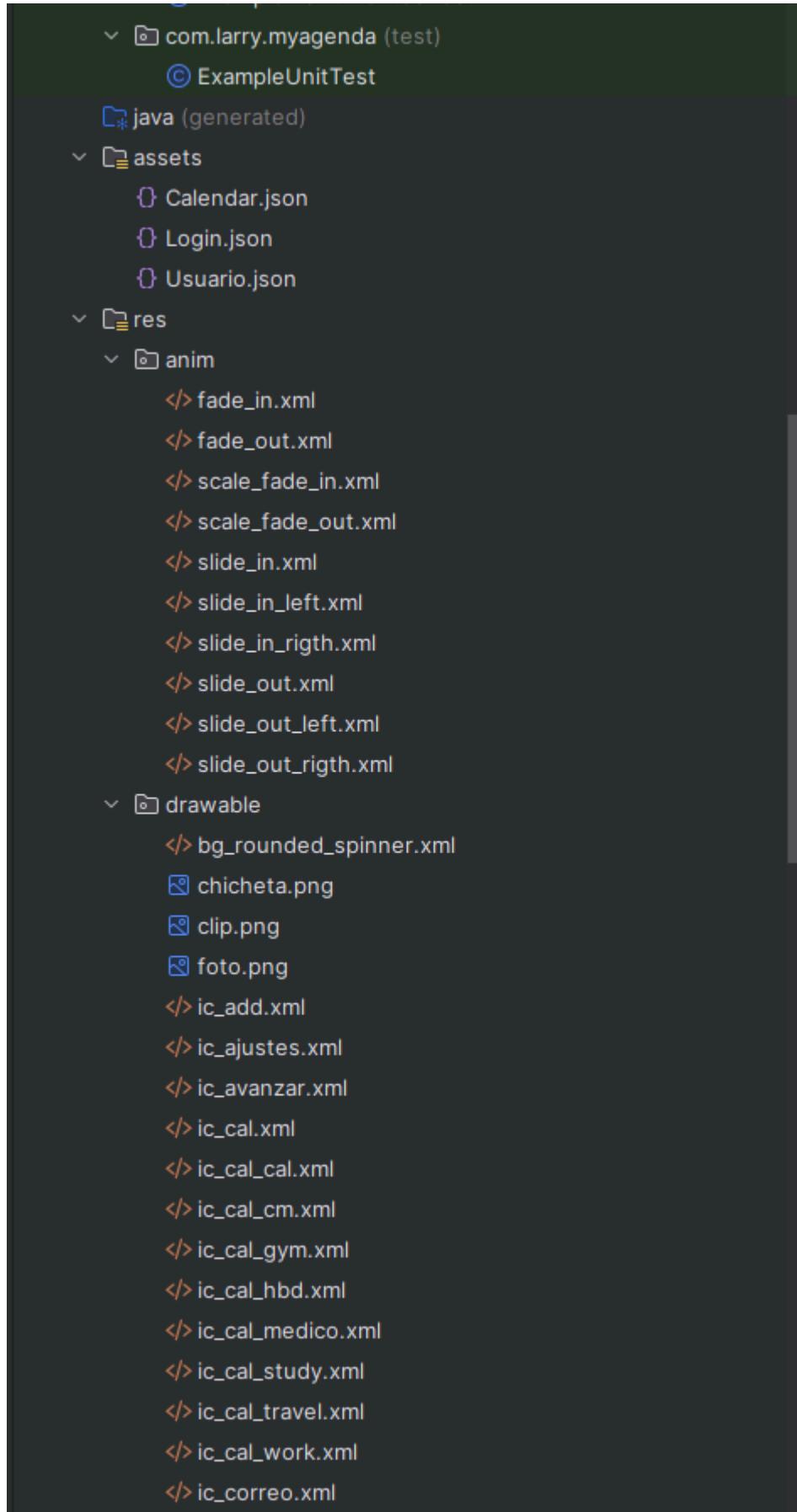
Tabla de trabajo

TABLA DE TRABAJO			
Nombre de tarea	Fecha inicio	Fecha fin	Horas
● Plan de trabajo	25/02/2024	04/03/2024	<u>30</u>
→ Documentación Android	25/02/2024	27/02/2024	10
→ Redacción 1ra parte TFG	29/02/2024	04/03/2024	20
● Análisis de la aplicación	06/03/2024	29/03/2024	<u>70</u>
→ Definición requisitos	07/03/2024	10/04/2024	10
→ Análisis	10/03/2024	18/04/2024	15
→ Diseño	12/03/2024	27/03/2024	20
→ prototipo	13/04/2024	28/03/2024	25
● Desarrollo 1ra Parte	01/04/2024	9/04/2024	<u>35</u>
→ Preparación IDE	01/04/2024	03/04/2024	5
→ Desarrollo de interfaz	03/04/2024	8/04/2024	30
● Desarrollo 2da Parte	10/04/2024	25/04/2024	<u>110</u>
→ Lógica aplicación	10/04/2024	25/04/2024	60
→ Acceso a datos	12/04/2024	21/04/2024	30
→ Pruebas	22/04/2024	25/04/2024	20
● Redacción 2da parte TFG	26/04/2024	03/05/2024	<u>45</u>
→ Redacción	26/04/2024	03/05/2024	35
→ Presentación/video	28/04/2024	01/04/2024	10
TOTAL HORAS	<u>290</u>		

Anexo 4

Estructura de carpetas del proyecto en Android Studio





```
</> ic_homee.xml
</> ic_indicador.xml
</> ic_launcher_background.xml
</> ic_launcher_foreground.xml
</> ic_nota.xml
</> ic_pass.xml
</> ic_person.xml
</> ic_retoceso.xml
☒ ic_tarea.png
▼ └ font
    └ letra.ttf
▼ └ layout
    </> activity_agregar_calendario.xml
    </> activity_agregar_nota.xml
    </> activity_agregar_tarea.xml
    </> activity_login.xml
    </> activity_main.xml
    </> activity_menu_principal.xml
    </> activity_registro.xml
    </> activity_splash.xml
    </> dia_calendario_item.xml
    </> fragment_ajustes.xml
    </> fragment_calendario.xml
    </> fragment_home.xml
    </> fragment_nota.xml
    </> fragment_notas.xml
    </> fragment_tareas.xml
    </> icon_item.xml
    </> icono_calendario_item.xml
    </> nota_item.xml
    </> tarea_item.xml
▼ └ menu
    </> menu_nav_bar.xml
> └ mipmap
> └ values
> └ xml
└ res (generated)
▼ └ Gradle Scripts
```

Anexo 5

Activity AgregarCalendario

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

Project Structure:

- app
 - sampledata
 - manifests
 - java
 - com.larry.myagenda
 - activity
 - AgregarCalendario
 - AgregarNota
 - AgregarTareaActivity
 - LoginActivity
 - MainActivity
 - MenuPrincipal
 - RegistroActivity
 - SplashActivity
 - adaptadores
 - DB
 - fragmentos
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - res (generated)

Code Editor (AregarCalendario.java):

```
1 package com.larry.myagenda.activitys;
2
3 > import ...
4
5 <> public class AgregarCalendario extends AppCompatActivity {
6
7     // Iconos
8     private RecyclerView recyclerView;
9     private IconosAdapter adapter;
10
11    // Variables
12    private Iconos iconoSeleccionado;
13    private int colorSeleccionado = Color.WHITE;
14    private int colorPorDefecto = Color.WHITE;
15    private Button btn_seleccionarColor;
16    private EditText NombreCalendarioET;
17
18    // Firebase
19    private FirebaseAuth mAuth;
20
21    // Base de datos
22    private BDCalendarios.CalendarioDBHelper dbHelper;
23
24    @Override
25    protected void onCreate(Bundle savedInstanceState) {
26        super.onCreate(savedInstanceState);
27        setContentView(R.layout.activity_agregar_calendario);
28
29        // Inicializar Firebase Auth
30        mAuth = FirebaseAuth.getInstance();
31
32        // Inicializar TextView
33        NombreCalendarioET = findViewById(R.id.NombreCalendarioET);
34
35        // Inicializar botones y EditText
36        Button btn_guardarCalendario = findViewById(R.id.Btn_guardarcalendario);
37        btn_seleccionarColor = findViewById(R.id.Btn_seleccionarcolor);
38
39        // Inicializar base de datos
40        dbHelper = new BDCalendarios.CalendarioDBHelper(this);
41
42        // Botón seleccionar color
43        btn_seleccionarColor.setOnClickListener(new View.OnClickListener() {
44            @Override
45            public void onClick(View v) { seleccionadorColor(); }
46        });
47
48        // Botón guardar
49        btn_guardarCalendario.setOnClickListener(new View.OnClickListener() {
50            @Override
51            public void onClick(View v) {
```

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

Project Structure:

- app
 - sampledata
 - manifests
 - java
 - com.larry.myagenda
 - activities
 - AgregarCalendario
 - AgregarNota
 - AgregarTareaActivity
 - LoginActivity
 - MainActivity
 - MenuPrincipal
 - RegistroActivity
 - SplashActivity
 - adapters
 - DB
 - fragmentos
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - res (generated)
- Gradle Scripts

Code Editor (AgregarCalendario.java):

```

1 // Botón guardar
2 Btn_guardarCalendario.setOnClickListener(new View.OnClickListener() {
3     @Override
4     public void onClick(View v) {
5         FirebaseUser currentUser = mAuth.getCurrentUser();
6         if (currentUser != null) {
7             String userId = currentUser.getUid();
8             String nombreCalendario = NombreCalendarioET.getText().toString().trim();
9             if (nombreCalendario.isEmpty() && iconoSeleccionado != null) {
10                 guardarCalendario(userId, nombreCalendario, iconoSeleccionado.getRuta(), colorSeleccionado);
11             } else {
12                 Toast.makeText(context, AgregarCalendario.this, text: "Por favor complete todos los campos", Toast.LENGTH_SHORT).show();
13             }
14         } else {
15             // El usuario no está autenticado, mandar el error de que el usuario no está autenticado
16             Toast.makeText(context, AgregarCalendario.this, text: "Usuario no autenticado", Toast.LENGTH_SHORT).show();
17         }
18     }
19 });
20
21 // Iconos
22 recyclerView = findViewById(R.id.recyclerViewCal);
23 LinearLayoutManager layoutManager = new LinearLayoutManager(this, LinearLayoutManager.HORIZONTAL, false);
24 recyclerView.setLayoutManager(layoutManager);
25 adapter = new IconsAdapter(getIconsList());
26 recyclerView.setAdapter(adapter);
27 adapter.setOnIconClickListener(new IconsAdapter.OnIconClickListener() {
28     @Override
29     public void onIconClick(Icons icono) { iconoSeleccionado = icono; }
30 });
31
32 Usage
33 private void seleccionarColor() {
34     ColorPickerDialogBuilder
35         .with(this)
36         .initialColor(colorOrDefecto)
37         .wheelType(ColorPickerView.WHEEL_TYPE.CIRCLE)
38         .density(15)
39         .setOnColorSelectedListener(new OnColorSelectedListener() {
40             @Override
41             public void onColorSelected(int selectedColor) {
42                 Btn_seleccionarColor.setBackground(selectedColor);
43             }
44         })
45         .setPositiveButton(text: "Seleccionar", (dialog, selectedColor, allColors) -> {
46             colorSeleccionado = selectedColor;
47             String mensaje = "Color elegido correctamente: #" + Integer.toHexString(selectedColor);
48             Toast.makeText(context, AgregarCalendario.this, mensaje, Toast.LENGTH_SHORT).show();
49         })
50         .setNegativeButton(text: "Cancelar", (dialog, which) -> {
51         })
52         .build()
53         .show();
54 }
55
56 
```

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

Project Structure:

- res
- res (generated)
- Gradle Scripts

Code Editor (res.java):

```

1 Usage
2 private List<Icons> getIconsList() { return IconsUtils.getIconsList(); }
3
4 Usage
5 private void guardarCalendario(String userId, String nombreCalendario, String iconoRuta, int color) {
6     long resultado = dbHelper.insertarCalendario(userId, nombreCalendario, iconoRuta, color);
7     if (resultado != -1) {
8         Toast.makeText(context, text: "Calendario guardado exitosamente", Toast.LENGTH_SHORT).show();
9         // Limpiar campos después de guardar
10        NombreCalendarioET.setText("");
11        iconoSeleccionado = null;
12        colorSeleccionado = Color.WHITE;
13        Btn_seleccionarColor.setBackground(colorOrDefecto);
14    } else {
15        Toast.makeText(context, text: "Error al guardar el calendario", Toast.LENGTH_SHORT).show();
16    }
17 }
18
19 
```

Anexo 6

Activity AgregarNota

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

Project Structure:

- Android
- app
 - sampledata
 - manifests
 - java
 - com.larry.myagenda
 - activity
 - AgregarCalendario
 - AgregarNota**
 - AgregarTareaActivity
 - LoginActivity
 - MainActivity
 - MenuPrincipal
 - RegistroActivity
 - SplashActivity
 - adaptadores
 - DB
 - fragmentos
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - res (generated)
 - Gradle Scripts

Code Editor (AregarNota.java):

```
1 package com.larry.myagenda.activity;
2
3 > import ...
4
5 <> public class AgregarNota extends AppCompatActivity {
6
7     3 usages
8     DatabaseReference userNotesRef;
9
10    2 usages
11    FirebaseAuth auth;
12
13    3 usages
14    FirebaseUser user;
15
16    3 usages
17    private TextView editTextTitulo, editTextDescripcion, editTextContenido;
18
19    2 usages
20    private Button btnGuardar;
21
22    11 usages
23    private Nota notaActual;
24
25    no usages
26    private static final int AGREGAR_NOTA_REQUEST_CODE = 1; // Código de solicitud único
27
28    @Override
29    protected void onCreate(Bundle savedInstanceState) {
30        super.onCreate(savedInstanceState);
31        setContentView(R.layout.activity_agregar_nota);
32
33        auth = FirebaseAuth.getInstance();
34        user = auth.getCurrentUser();
35
36        // Obtener referencia a la base de datos para las notas del usuario actual
37        if (user != null) {
38            String userId = user.getUid();
39            userNotesRef = FirebaseDatabase.getInstance().getReference("usuarios").child(userId).child("notas");
40        } else {
41            // Manejo de error: El usuario no está autenticado
42            // Puedes mostrar un mensaje o redirigir al usuario a la pantalla de inicio de sesión
43            Toast.makeText(context, "AregarNota.this", "Usuario no autenticado", Toast.LENGTH_SHORT).show();
44            finish(); // Finaliza la actividad actual
45            return; // Salir del método onCreate
46        }
47
48        // Inicializar vistas
49        editTextTitulo = findViewById(R.id.editTextTitulo);
50        editTextDescripcion = findViewById(R.id.editTextDescripcion);
51        editTextContenido = findViewById(R.id.editTextContenido);
52        btnGuardar = findViewById(R.id.btnGuardar);
53
54        // Verificar si se pasó una nota como extra en el Intent
55        if (getIntent().hasExtra("nota")) {
56            // Si se pasa una nota, cargar los detalles de la nota en los campos correspondientes
57            notaActual = (Nota) getIntent().getSerializableExtra("nota");
58            if (notaActual != null) {
59                editTextTitulo.setText(notaActual.getTitulo());
60                editTextDescripcion.setText(notaActual.getDescripcion());
61                editTextContenido.setText(notaActual.getContenido());
62            }
63        }
64
65        btnGuardar.setOnClickListener(new View.OnClickListener() {
66            @Override
67
68            1 usage
69            ...
70
71            ...
72
73        });
74    }
75}
```

The screenshot shows the Android Studio interface. On the left, the project navigation pane displays the following structure:

- app
 - sampledata
 - manifests
 - java
 - com.larry.myagenda
 - activities
 - AgregarCalendario
 - AgregarNota
 - AgregarTareaActivity
 - LoginActivity
 - MainActivity
 - MenuPrincipal
 - RegistroActivity
 - SplashActivity
 - adapters
 - DB
 - fragments
 - objects
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - (generated)
 - Gradle Scripts

The file `AgregarNota.java` is open in the main editor window. The code implements a logic for saving a note. It checks if the title is empty and if the current note's title matches the new one. It then creates a new note object with the provided data and saves it to the database. A toast message is displayed to inform the user that the note was saved successfully.

```
73     @Override
74     public void onClick(View v) { guardarNota(); }
75   });
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
```

```
private void guardarNota() {
    String titulo = editTextTitulo.getText().toString().trim();
    String descripcion = editTextDescripcion.getText().toString().trim();
    String contenido = editTextContenido.getText().toString().trim();
    String fecha = obtenerFechaActual();

    if (!TextUtils.isEmpty(titulo)) {
        // Verificar si la nota actual es nula o si hay cambios en los campos
        if (notaActual == null || !titulo.equals(notaActual.getTitulo())
            || !descripcion.equals(notaActual.getDescripcion())
            || !contenido.equals(notaActual.getContenido())) {
            String notaId;

            // Si la nota actual es nula, significa que estamos agregando una nueva nota
            if (notaActual == null) {
                notaId = userNotesRef.push().getKey();
            } else {
                // Si la nota actual no es nula, significa que estamos editando una nota existente
                notaId = notaActual.getId();
            }

            // Crear objeto Nota con los datos ingresados por el usuario
            Nota nota = new Nota(notaId, titulo, descripcion, contenido, fecha);

            // Guardar la nota en la base de datos
            userNotesRef.child(notaId).setValue(nota);
            Toast.makeText(context, AgregarNota.this, "Nota guardada exitosamente", Toast.LENGTH_SHORT).show();
            setResult(Activity.RESULT_OK);
            finish(); // Finalizar la actividad después de guardar la nota
        } else {
            // Mostrar un mensaje si no hay cambios en la nota
            Toast.makeText(context, AgregarNota.this, "No se realizaron cambios en la nota", Toast.LENGTH_SHORT).show();
        }
    } else {
        // Mostrar un mensaje de error si el título está vacío
        Toast.makeText(context, AgregarNota.this, "Por favor, ingresa un título para la nota", Toast.LENGTH_SHORT).show();
    }
}

// Método para obtener la fecha actual
private String obtenerFechaActual() {
    Calendar calendar = Calendar.getInstance();
    SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy", Locale.getDefault());
    return dateFormat.format(calendar.getTime());
}
```

Anexo 7

Activity AgregarTarea

The screenshot shows the Android Studio interface. The project structure is identical to the previous one, with the `AgregarTareaActivity.java` file selected in the navigation pane.

The code for `AgregarTareaActivity.java` defines a class that extends `AppCompatActivity`. It initializes several UI components: an `EditText` for the task name, another for the task description, a `TimePicker` for the due date, and a `Button` for saving the task. It also initializes a database helper class (`B0Calendarios.CalendarioDBHelper`). The `onCreate` method sets the content view to the layout file `activity_agregar_tarea`.

```
1 package com.larry.myagenda.activities;
2
3 > import ...
4
5 <> public class AgregarTareaActivity extends AppCompatActivity {
6
7     private EditText etNombreTarea;
8     private EditText etDescripcionTarea;
9     private TimePicker timePickerHoraTarea;
10    private Button btnGuardarTarea;
11    private B0Calendarios.CalendarioDBHelper dbHelper;
12    private String calendarioId;
13    private int diaSeleccionado;
14    private int mesSeleccionado;
15    private int añoSeleccionado;
16
17    @Override
18    protected void onCreate(Bundle savedInstanceState) {
19        super.onCreate(savedInstanceState);
20        setContentView(R.layout.activity_agregar_tarea);
21    }
22}
```

The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is a tree view of the project structure under the 'app' folder, including 'sampledata', 'manifests', 'java' (containing 'com.larry.myagenda' with 'activity' subfolder), 'DB', 'fragmentos', 'objetos', 'com.larry.myagenda (androidTest)', 'com.larry.myagenda (test)', 'Java (generated)', 'assets', 'res', 'res (generated)', and 'Gradle Scripts'. The 'activity' folder is expanded, and 'AgregarTareaActivity' is selected, highlighted with a gray background.

The main area displays the code for `AgregarTareaActivity.java`:

```

36     // Inicializar la base de datos
37     dbHelper = new BDcalendarios.CalendarioDBHelper(context, this);
38
39     // Obten el ID del calendario y los datos de fecha de los argumentos
40     if (getIntent().getExtras() != null) {
41         calendarId = getIntent().getStringExtra("calendarid");
42         diaSeleccionado = getIntent().getIntExtra("diaSeleccionado", defaultValue);
43         mesSeleccionado = getIntent().getIntExtra("mesSeleccionado", defaultValue);
44         añoSeleccionado = getIntent().getIntExtra("añoSeleccionado", defaultValue);
45     }
46
47     // Inicializar vistas
48     etNombreTarea = findViewById(R.id.editTextNombreTarea);
49     etDescripcionTarea = findViewById(R.id.editTextDescripcionTarea);
50     timePickerHoraTarea = findViewById(R.id.timePickerHoraTarea);
51     btnGuardarTarea = findViewById(R.id.buttonGuardarTarea);
52
53     // Configurar listener de clic para el botón para guardar la tarea
54     btnGuardarTarea.setOnClickListener(new View.OnClickListener() {
55         @Override
56         public void onClick(View v) { guardarTarea(); }
57     });
58
59     Usage
60
61     private void guardarTarea() {
62         String nombre = etNombreTarea.getText().toString().trim();
63         String descripcion = etDescripcionTarea.getText().toString().trim();
64
65         // Crea una instancia de Calendar para obtener la fecha completa
66         Calendar calendar = Calendar.getInstance();
67         calendar.set(añoSeleccionado, mesSeleccionado, diaSeleccionado, timePickerHoraTarea.getHour(), timePickerHoraTarea.getMinute());
68
69         // Obtener la fecha completa en milisegundos
70         long fecha = calendar.getTimeInMillis();
71
72         // Formatear la fecha
73         SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy - HH:mm", Locale.getDefault());
74         String fechaFormatada = sdf.format(calendar.getTime());
75
76         // Verificar que todos los campos están completos
77         if (nombre.isEmpty() || descripcion.isEmpty()) {
78             Toast.makeText(context, "Por favor completa todos los campos", Toast.LENGTH_SHORT).show();
79             return;
80         }
81
82         // Insertar la tarea en la base de datos con el ID del calendario y la fecha completa
83         long resultado = BDcalendarios.CalendarioDBHelper.insertarTarea(context, this, calendarId, nombre, descripcion, fechaFormatada);
84
85         if (resultado != -1) {
86             // Tarea guardada correctamente
87             Toast.makeText(context, "Tarea guardada correctamente", Toast.LENGTH_SHORT).show();
88         } else {
89             // Error al guardar la tarea
90             Toast.makeText(context, "Error al guardar la tarea", Toast.LENGTH_SHORT).show();
91         }
92
93         // Volver a la actividad anterior
94         finish();
95     }
96 }

```

Anexo 8

Activity Login

The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it is a tree view of the project structure under the 'app' folder, including 'sampledata', 'manifests', 'java' (containing 'com.larry.myagenda' with 'activity' subfolder), 'DB', 'fragmentos', 'objetos', 'com.larry.myagenda (androidTest)', 'com.larry.myagenda (test)', 'Java (generated)', 'assets', 'res', 'res (generated)', and 'Gradle Scripts'. The 'activity' folder is expanded, and 'LoginActivity' is selected, highlighted with a gray background.

The main area displays the code for `LoginActivity.java`:

```

1 package com.larry.myagenda.activitys;
2
3 > import ...
4
5 <> public class LoginActivity extends AppCompatActivity {
6
7     EditText CorreoLogin, ContraseñaLogin;
8     TextView Notiene cuenta;
9     ProgressDialog progressDialog;
10    Button Btn_ingresar;
11
12    @Override
13    protected void onCreate(Bundle savedInstanceState) {
14        super.onCreate(savedInstanceState);
15        setContentView(R.layout.activity_login);
16
17        CorreoLogin = findViewById(R.id.CorreoLogin);
18        ContraseñaLogin = findViewById(R.id.ContraseñaLogin);
19        Notiene cuenta = findViewById(R.id.Notiene cuenta);
20        Btn_ingresar = findViewById(R.id.Btn_ingresar);
21
22    }
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

```

The screenshot shows the Android Studio interface with the project structure on the left and the LoginActivity.java code on the right.

```

> manifests
  < java
    < com.larry.myagenda
      < activity
        < AgregarCalendario
        < AgregarNota
        < AgregarTareaActivity
        < LoginActivity
        < MainActivity
        < MenuPrincipal
        < RegistroActivity
        < SplashActivity
      < adaptadores
      < DB
      < fragmentos
      < objetos
    < com.larry.myagenda (androidTest)
    < com.larry.myagenda (test)
  < java (generated)
  < assets
  < res
  < res (generated)
< Gradle Scripts

```

LoginActivity.java (Excerpt):

```

39 progressDialog = new ProgressDialog(context);
40 progressDialog.setMessage("Iniciando sesión...");
41
42 Btn_ingresar.setOnClickListener(new View.OnClickListener() {
43     @Override
44     public void onClick(View v) {
45
46         // Obtiene el correo electrónico y la contraseña ingresados por el usuario
47         String email = CorreoLogin.getText().toString().trim();
48         String password = ContraseñaLogin.getText().toString().trim();
49
50         // Verifica si los campos están vacíos
51         if (TextUtils.isEmpty(email) || TextUtils.isEmpty(password)) {
52             Toast.makeText(context, LoginActivity.this, "Por favor, ingresa tu correo y contraseña.", Toast.LENGTH_SHORT).show();
53             return;
54         }
55         // Muestra el ProgressDialog.
56         progressDialog.show();
57
58         // Autentica al usuario con Firebase
59         FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password)
60             .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
61                 @Override
62                 public void onComplete(@NonNull Task<AuthResult> task) {
63                     progressDialog.dismiss(); // Oculta el ProgressDialog.
64
65                     if (task.isSuccessful()) {
66                         // Si la autenticación es exitosa, abre la actividad del menú principal.
67                         startActivity(new Intent(LoginActivity.this, MenuPrincipal.class));
68                         Toast.makeText(getApplicationContext(), "Bienvenido!", Toast.LENGTH_SHORT).show();
69                         finish(); // Cierra esta actividad para garantizar que el usuario solo vea el menú.
70                     } else {
71                         // Si la autenticación falla, muestra un mensaje de error.
72                         Toast.makeText(context, LoginActivity.this, "Error al iniciar sesión. Verifica tus credenciales.", Toast.LENGTH_SHORT).show();
73                     }
74                 }
75             });
76         Notiene cuenta.setOnClickListener(new View.OnClickListener() {
77             @Override
78             public void onClick(View v) {
79                 startActivity(new Intent(LoginActivity.this, RegistroActivity.class));
80                 finish();
81             }
82         });
83     }
84 }

```

Anexo 9

Activity Main

The screenshot shows the Android Studio interface with the project structure on the left and the MainActivity.java code on the right.

```

Android
  < app
    < sampledata
    < manifests
    < java
      < com.larry.myagenda
        < activity
          < AgregarCalendario
          < AgregarNota
          < AgregarTareaActivity
          < LoginActivity
          < MainActivity
          < MenuPrincipal
          < RegistroActivity
          < SplashActivity
        < adaptadores
        < DB
        < fragmentos
        < objetos
      < com.larry.myagenda (androidTest)
      < com.larry.myagenda (test)
    < java (generated)

```

MainActivity.java (Excerpt):

```

1 package com.larry.myagenda.activitys;
2
3 > import ...
4
5 <> public class MainActivity extends AppCompatActivity {
6
7     < 2 usages
8     < Button Btn_login, Btn_Registro;
9
10    @Override
11    protected void onCreate(Bundle savedInstanceState) {
12        super.onCreate(savedInstanceState);
13        setContentView(R.layout.activity_main);
14
15        Btn_login = (Button) findViewById(R.id.Btn_Login);
16        Btn_Registro = (Button) findViewById(R.id.Btn_Registro);
17
18        Btn_login.setOnClickListener(new View.OnClickListener() {
19            @Override
20            public void onClick(View v) {
21                Intent intent = new Intent(getApplicationContext(), LoginActivity.class);
22                startActivity(intent);
23            }
24        );
25        Btn_Registro.setOnClickListener(new View.OnClickListener() {
26            @Override
27            public void onClick(View v) {
28                Intent intent = new Intent(getApplicationContext(), RegistroActivity.class);
29                startActivity(intent);
30            }
31        );
32    }
33
34 }
35
36 }
37
38 }
39

```

Anexo 10

Activity MenuPrincipal

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

Project Structure:

- app/
 - sampleddata
 - manifests
 - java/
 - com.larry.myagenda/activity/
 - AgregarCalendario
 - AgregarNota
 - AgregarTareaActivity
 - LoginActivity
 - MainActivity
 - MenuPrincipal
 - RegistroActivity
 - SplashActivity
 - adaptadores
 - DB
 - fragmentos
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - res (generated)

Gradle Scripts

Code Editor (MenuPrincipal.java):

```
1 package com.larry.myagenda.activities;
2
3 import ...
4
5 public class MenuPrincipal extends AppCompatActivity {
6
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_menu_principal);
12
13         //INICIALIZAR VARIABLES
14         BottomNavigationView bottomNavigationView = findViewById(R.id.bottom_navigation);
15         FrameLayout frameLayout = findViewById(R.id.fragment_container);
16         FirebaseAuth mAuth = FirebaseAuth.getInstance();
17
18         bottomNavigationView.setOnNavigationItemSelected(new BottomNavigationView.OnNavigationItemSelectedListener() {
19             @Override
20             public boolean onNavigationItemSelected(@NonNull MenuItem item) {
21
22                 int itemId = item.getItemId();
23
24                 if (itemId == R.id.notas_menu) {
25                     LoadFragment(new NotasFragment(), false);
26                 } else if (itemId == R.id.home_menu) {
27                     LoadFragment(new HomeFragment(), false);
28                 } else if (itemId == R.id.ajustes_menu) {
29                     LoadFragment(new AjustesFragment(), false);
30                 }
31
32                 return true;
33             }
34
35             LoadFragment(new NotasFragment(), true);
36         });
37
38         LoadFragment(new NotasFragment(), true);
39
40     }
41
42     private void LoadFragment(Fragment fragment, boolean isAppInitialized) {
43
44         // Obtener el administrador de fragmentos
45         FragmentManager fragmentManager = getSupportFragmentManager();
46         // Obtener el fragmento actualmente visible en el contenedor
47         Fragment currentFragment = fragmentManager.findFragmentById(R.id.fragment_container);
48         // Verificar si el fragmento que se va a cargar es el mismo que el actual
49         boolean isSameFragment = currentFragment != null && currentFragment.getClass().getName().equals(fragment.getClass().getName());
50
51         // Crear una transacción de fragmentos
52         FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
53
54         // Si la aplicación se ha inicializado y el fragmento no es el mismo que el actual
55         if (isAppInitialized && !isSameFragment) {
56             // Agregar el nuevo fragmento al contenedor
57             fragmentTransaction.add(R.id.fragment_container, fragment);
58         } else {
59             // Reemplazar el fragmento actual con el nuevo fragmento en el contenedor
60             fragmentTransaction.replace(R.id.fragment_container, fragment);
61         }
62
63         fragmentTransaction.replace(R.id.fragment_container, fragment);
64     }
65
66     // Confirmar la transacción
67     fragmentTransaction.commit();
68 }
69
70 @Override
71 public void onBackPressed() {
72     FragmentManager fragmentManager = getSupportFragmentManager();
73     Fragment fragment = fragmentManager.findFragmentById(R.id.fragment_container);
74     if (fragment instanceof NotasFragment || fragment instanceof HomeFragment || fragment instanceof AjustesFragment) {
75         // Mueve la tarea de la aplicación a un segundo plano
76         moveTaskToBack(false);
77     } else {
78         super.onBackPressed();
79     }
80 }
81 }
```

Code Editor (Gradle Scripts):

```
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90 }
```

Anexo 11

Activity Registro

Android

```

    package com.larry.myagenda.activities;
    import ...
    public class RegistroActivity extends AppCompatActivity {
        ...
        protected void onCreate(Bundle savedInstanceState) {
            ...
            nombreET = findViewById(R.id.nombreET);
            correoET = findViewById(R.id.correoET);
            contraseñaET = findViewById(R.id.contraseñaET);
            conf_contraseñaET = findViewById(R.id.conf_contraseñaET);
            tango_cuenta = findViewById(R.id.tango_cuenta);
            Btn_Registro = findViewById(R.id.Btn_Registro);
            FirebaseAuth Auth;
            ...
            Btm_Registro.setOnClickListener(new View.OnClickListener() {
                ...
            });
            tango_cuenta.setOnClickListener(new View.OnClickListener() {
                ...
            });
        }
        ...
        private void registrarUsuario() {
            ...
            if (TextUtils.isEmpty(nombre) || TextUtils.isEmpty(correo) || TextUtils.isEmpty(contraseña) || TextUtils.isEmpty(confirmarContraseña)) {
                Toast.makeText(getApplicationContext(), "Por favor, complete todos los campos", Toast.LENGTH_SHORT).show();
                return;
            }
            ...
        }
        ...
    }

```

Android

```

    ...
    // Validar que las contraseñas coincidan
    if (!contraseña.equals(confirmarContraseña)) {
        Toast.makeText(getApplicationContext(), "Las contraseñas no coinciden", Toast.LENGTH_SHORT).show();
        return;
    }

    final ProgressDialog progressDialog = new ProgressDialog(this);
    progressDialog.setMessage("Creando su cuenta...");
    progressDialog.show();

    // Registrar usuario en Firebase Authentication
    Auth.createUserWithEmailAndPassword(correo, contraseña)
        .addOnCompleteListener(activity, new OnCompleteListener<AuthResult>() {
            ...
            if (task.isSuccessful()) {
                ...
                FirebaseUser user = Auth.getCurrentUser();
                if (user != null) {
                    ...
                    guardarDatosUsuarioFirebase(user.getUid(), nombre, correo, contraseña);
                    ...
                }
            } else {
                ...
            }
        });

```

```

137     if
138         // Datos de usuario guardados exitosamente en la base de datos
139         if(task.isSuccessful()){
140
141             } else {
142                 // Error al guardar datos del usuario
143                 Toast.makeText(getApplicationContext(), "Error al guardar datos del usuario", Toast.LENGTH_SHORT).show();
144             }
145         }
146     }
147 }
148

```

Anexo 12

Activity Splash

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

Project Structure:

- app
 - sampledadata
 - manifests
 - java
 - com.larry.myagenda
 - activity
 - SplashActivity
 - assets
 - res
 - res (generated)
 - Gradle Scripts

SplashActivity.java Code:

```

package com.larry.myagenda.activity;
import ...
public class SplashActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);

        //INICIALIZAR VARIABLES
        mAuth = FirebaseAuth.getInstance();

        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                Intent intent = new Intent(getApplicationContext(), MainActivity.class);
                startActivity(intent);
                finish();
                VerificarUsuario();
            }
        }, 1500);
    }

    private void VerificarUsuario() {
        //METODO PARA VERIFICAR SI EL USUARIO YA HA INICIADO SESION ANTES
        FirebaseUser user = mAuth.getCurrentUser();
        if (user == null){
            startActivity(new Intent(getApplicationContext(), MainActivity.class));
            finish();
        }else{
            startActivity(new Intent(getApplicationContext(), MainActivity.class));
            finish();
        }
    }
}

```

Anexo 13

Adaptadores CalendarioAdapter

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

Project Structure:

- app
 - sampledadata
 - manifests
 - java
 - com.larry.myagenda
 - activity
 - adaptores
 - CalendarioAdapter
 - IconoCalendarioAdapter
 - IconosAdapter
 - NotaAdapter
 - TareasAdapter
 - DB
 - fragmentos
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets

```

package com.larry.myagenda.adaptadores;
import ...
public class CalendarioAdapter extends BaseAdapter {
    private Context context;
    private List<Integer> daysOfMonth;
    private Spinner spinnerMes, spinnerAño;
    private String calendarioId;
    private int mesSeleccionado;
    private int añoSeleccionado;
    private int diaSeleccionado = -1;

    public CalendarioAdapter(Context context, Spinner spinnerMes, Spinner spinnerAño, String calendarioId) {
        this.context = context;
        this.daysOfMonth = new ArrayList<>();
        this.spinnerMes = spinnerMes;
        this.spinnerAño = spinnerAño;
        this.calendarioId = calendarioId;
    }
}

```

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right.

Project Structure:

- Android
- app
 - sampledata
 - manifests
 - java
 - com.larry.myagenda
 - activities
 - adaptores
 - CalendarioAdapter (selected)
 - IconoCalendarioAdapter
 - IconosAdapter
 - NotaAdapter
 - TareasAdapter
 - DB
 - fragmentos
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)- java (generated)
- assets
- res
- res (generated)
- Gradle Scripts

Code Editor (CalendarioAdapter.java):

```
65 // Obtener el mes y el año actual
66 Calendar calendar = Calendar.getInstance();
67 mesSeleccionado = calendar.get(Calendar.MONTH);
68 añoSeleccionado = calendar.get(Calendar.YEAR);
69 diaSeleccionado = calendar.get(Calendar.DAY_OF_MONTH);
70 actualizarMesYAno(mesSeleccionado, añoSeleccionado);
71
72 spinnerMes.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
73     @Override
74     public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
75         mesSeleccionado = position;
76         actualizarMesYAno(mesSeleccionado, añoSeleccionado);
77     }
78
79     @Override
80     public void onNothingSelected(AdapterView<?> parent) {
81         // No hacer nada si no se selecciona nada
82     }
83 };
84
85 // Agregar listener para el cambio de selección en el spinner de año
86 spinnerAño.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
87     @Override
88     public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
89         añoSeleccionado = Integer.parseInt(parent.getItemAtPosition(position).toString());
90         actualizarMesYAno(mesSeleccionado, añoSeleccionado);
91     }
92
93     @Override
94     public void onNothingSelected(AdapterView<?> parent) {
95         // No hacer nada si no se selecciona nada
96     }
97 };
98
99
100 @Override
101 public int getCount() { return daysOfMonth.size(); }
102
103 @Override
104 public Object getItem(int position) { return daysOfMonth.get(position); }
105
106 @Override
107 public long getItemId(int position) { return position; }
108
109 @Override
110 public View getView(int position, View convertView, ViewGroup parent) {
111     if (convertView == null) {
112         convertView = LayoutInflater.from(context).inflate(R.layout.dia_calendario_item, parent,
113             attachToRoot);
114     }
115
116     // Obtener el día actual
117     int day = daysOfMonth.get(position);
118
119     // Obtener una referencia al TextView y al indicador de tarea en el diseño del elemento del calendario
120     TextView txtDay = convertView.findViewById(R.id.txtDay);
121     CardView CVDay = convertView.findViewById(R.id.CVDay);
122     ImageView imgTaskIndicator = convertView.findViewById(R.id.imgTaskIndicator);
123
124     txtDay.setText(String.valueOf(day));
125     if (day == diaSeleccionado && año == añoSeleccionado && mes == mesSeleccionado) {
126         CVDay.setVisibility(View.VISIBLE);
127         imgTaskIndicator.setVisibility(View.VISIBLE);
128     } else {
129         CVDay.setVisibility(View.GONE);
130         imgTaskIndicator.setVisibility(View.GONE);
131     }
132 }
```

The screenshot shows the Android Studio interface with the project navigation bar on the left and the code editor on the right.

Project Structure:

- Android
- app
 - sampledata
 - manifests
 - java
 - com.larry.myagenda
 - activities
 - adapters
 - CalendarioAdapter
 - IconoCalendarioAdapter
 - IconosAdapter
 - NotaAdapter
 - TareasAdapter
 - DB
 - fragmentos
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - (generated)
 - Gradle Scripts

Code Editor (CalendarioAdapter.java):

```
// Establecer el dia en el TextView
if (day != 0) {
    txtDay.setText(String.valueOf(day));
    CVDia.setBackgroundColor(ContextCompat.getColor(context, R.color.orange));
}

// Resaltar el dia seleccionado
if (day == diaSeleccionado) {
    txtDay.setBackgroundDrawable(context.getResources().getDrawable(R.color.yellow));
    txtDay.setTextColor(Color.BLACK);
} else {
    txtDay.setBackgroundDrawable(Color.TRANSPARENT);
    txtDay.setTextColor(context.getResources().getColor(android.R.color.black));
}

// Verificar si hay tareas para el dia actual
boolean hayTareas = hayTareasParaDia(day, mesSeleccionado, añoSeleccionado, calendarioId);
if (hayTareas) {
    imgTaskIndicator.setVisibility(View.VISIBLE);
} else {
    imgTaskIndicator.setVisibility(View.INVISIBLE);
}

} else {
    txtDay.setText("");
    imgTaskIndicator.setVisibility(View.INVISIBLE);
}

return convertView;
}

// Método para actualizar el mes y el año del adaptador
public void actualizarMesAño(int mes, int año) {
    spinnerMes.setSelection(mes);
    ArrayAdapter<String> adapterAño = (ArrayAdapter<String>) spinnerAño.getAdapter();
    spinnerAño.setSelection(adapterAño.getPosition(String.valueOf(año)));

    // Limpian la lista de días del mes
    daysOfMonth.clear();

    // Actualizar la lista de días del mes
    int daysInMonth = obtenerDíasEnMes(mes, año);
    for (int i = 1; i <= daysInMonth; i++) {
        daysOfMonth.add(i);
    }

    notifyDataSetChanged();
}
```

Android

```

    app
        sampledata
        manifests
        java
            com.larry.myagenda
                activitys
                adaptadores
                    CalendarioAdapter
                    IIconoCalendarioAdapter
                    IconsAdapter
                    NotaAdapter
                    TareasAdapter
            DB
            fragmentos
            objetos
            com.larry.myagenda (androidTest)
            com.larry.myagenda (test)
            Java (generated)
            assets
            res
            res (generated)
        Gradle Scripts

```

CalendarioAdapter.java

```

101     private int obtenerDiasEnMes(int mes, int año) {
102         Calendar calendar = Calendar.getInstance();
103         calendar.set(Calendar.YEAR, año);
104         calendar.set(Calendar.MONTH, mes);
105         return calendar.getActualMaximum(Calendar.DAY_OF_MONTH);
106     }
107
108     // Método para establecer el día seleccionado
109     public void setDiaSeleccionado(int dia) { diaSeleccionado = dia; }
110
111     // Método para avanzar al siguiente mes
112     public void avanzarMes() {
113         mesSeleccionado++;
114         if (mesSeleccionado > 11) {
115             mesSeleccionado = 0;
116             añoSeleccionado++;
117         }
118         actualizarMesYAño(mesSeleccionado, añoSeleccionado);
119         // Verificar si hay tareas para el día seleccionado en el nuevo mes y año
120         hayTareasParaDia(diaSeleccionado, mesSeleccionado, añoSeleccionado, calendarioId);
121
122     }
123
124     // Método para retroceder al mes anterior
125     public void retrocederMes() {
126         mesSeleccionado--;
127         if (mesSeleccionado < 0) {
128             mesSeleccionado = 11;
129             añoSeleccionado--;
130         }
131         actualizarMesYAño(mesSeleccionado, añoSeleccionado);
132         hayTareasParaDia(diaSeleccionado, mesSeleccionado, añoSeleccionado, calendarioId);
133
134     }
135
136     3 usages
137     private boolean hayTareasParaDia(int dia, int mes, int año, String idCalendario) {
138         // Formatear la fecha del día específico en el formato esperado (dd/MM/yyyy)
139         String fechaDiaString = String.format(Locale.getDefault(), "dd/MM/yyyy", dia, mes + 1, año);
140
141         // Llamar al método estático de la clase BDCalendarios para obtener las tareas del calendario específico para esa fecha
142         Cursor cursor = BDCalendarios.CalendarioDBHelper.obtenerTareasDeCalendario(context, idCalendario);
143
144         // Verificar si se encontraron tareas para ese día, mes y año
145         boolean hayTareas = false;
146         if (cursor != null && cursor.moveToFirst()) {
147             do {
148                 // Obtener la fecha de la tarea de la base de datos
149                 String fechaTareaString = cursor.getString(cursor.getColumnIndexOrThrow(BDCalendarios.TareaEntrada.COLUMN_FECHA));
150
151                 // Comparar la fecha de la tarea con la fecha del día específico
152                 if (fechaTareaString.startsWith(fechaDiaString)) {
153                     hayTareas = true;
154                     break;
155                 }
156             } while (cursor.moveToNext());
157             // Cerrar el cursor
158             cursor.close();
159         }
160         return hayTareas;
161     }

```

Anexo 14

Adaptadores IIconoCalendarioAdapter

Android

```

    app
        sampledata
        manifests
        java
            com.larry.myagenda
                activitys
                adaptadores
                    CalendarioAdapter
                    IIconoCalendarioAdapter
                    IconsAdapter
                    NotaAdapter
                    TareasAdapter
            DB
            fragmentos
            objetos
            com.larry.myagenda (androidTest)
            com.larry.myagenda (test)
            Java (generated)
            assets
            res
            res (generated)
        Gradle Scripts

```

IIconoCalendarioAdapter.java

```

1 package com.larry.myagenda.adaptadores;
2
3 import ...
4
5 7 usages
6
7 public class IIconoCalendarioAdapter extends RecyclerView.Adapter<IIconoCalendarioAdapter.IconoViewHolder> {
8
9     1 usages
10    private Context context;
11
12    3 usages
13    private List<Calendario> calendarios;
14
15    2 usages
16    private List<Iconos> iconosList;
17
18    5 usages
19    private OnCalendarioClickListener listener;
20
21    3 usages
22    private OnCalendarioDoubleClickListener doubleClickListener;
23
24    3 usages
25    private Calendario calendarioSeleccionado;
26
27    2 usages
28    private long lastClickTime = 0;
29
30    1 usage
31    private static final long DOUBLE_CLICK_TIME_DELTA = 300;
32
33    1 usage
34    public IIconoCalendarioAdapter(List<Calendario> calendarios, Context context) {
35        this.context = context;
36        this.calendarios = calendarios;
37        this.iconosList = getIconosList();
38    }
39
40    @NotNull
41    @Override
42    public IconoViewHolder onCreateViewHolder(@NotNull ViewGroup parent, int viewType) {
43        View view = LayoutInflater.from(context).inflate(R.layout.icono_calendario_item, parent, false);
44        return new IconoViewHolder(view);
45    }

```

The screenshot shows the Android Studio interface with the project navigation bar on the left and the code editor on the right.

Project Structure:

- Android
- app
 - sampledatta
 - manifests
 - java
 - com.larry.myagenda
 - activity
 - adaptores
 - CalendarioAdapter
 - IIconoCalendarioAdapter
 - IIconosAdapter
 - NotaAdapter
 - TareasAdapter
 - DB
 - fragmentos
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - (generated)
- Gradle Scripts

Code Editor (IIconoCalendarioAdapter.java):

```
42     @Override
43     public void onBindViewHolder(@NotNull IconoViewHolder holder, int position) {
44         Calendario calendario = calendarios.get(position);
45
46         holder.cardViewIcono.setCardBackgroundColor(calendario.getColor());
47         String iconoRuta = calendario.getIconoRuta();
48         Iconos icono = buscarIconoPorRuta(iconoRuta);
49         if (icono != null) {
50             holder.iconoImageView.setImageResource(icono.getId());
51         } else {
52             holder.iconoImageView.setImageResource(R.drawable.ic_no_image);
53         }
54
55         // Verificamos si este calendario es el calendario seleccionado y actualizamos su apariencia
56         if (calendario.equals(calendarioSeleccionado)) {
57             // Cambia la apariencia del calendario seleccionado
58             holder.itemView.setScaleX(1.1f);
59             holder.itemView.setScaleY(1.1f);
60         } else {
61             // Restauramos la apariencia predeterminada del calendario no seleccionado
62             holder.itemView.setScaleX(1.0f);
63             holder.itemView.setScaleY(1.0f);
64         }
65
66         // Manejar el clic en el elemento del calendario
67         holder.itemView.setOnClickListener(new View.OnClickListener() {
68             @Override
69             public void onClick(View v) {
70                 // Actualizar el calendario seleccionado y notificar los cambios
71                 calendarioSeleccionado = calendario;
72                 notifyDataSetChanged();
73
74                 // Manejar el doble clic
75                 long clickTime = System.currentTimeMillis();
76                 if (clickTime - lastClickTime < DOUBLE_CLICK_TIME_DELTA) {
77                     // Doble clic detectado
78                     if (doubleClickListener != null) {
79                         doubleClickListener.onCalendarioDoubleClick(calendario);
80                     }
81                 }
82                 lastClickTime = clickTime;
83
84                 // Llamar al método onCalendarioClick() del listener para manejar el clic del calendario
85                 if (listener != null) {
86                     listener.onCalendarioClick(calendario);
87                 }
88             }
89         });
90     }
91
92     @Override
93     public int getItemCount() { return calendarios.size(); }
94
95     public void setOnCalendarioClickListener(OnCalendarioClickListener listener) {
96         this.listener = listener;
97     }
98
99
100 }
```

Android

```

    app
        sampledata
        manifests
        java
            com.larry.myagenda
                activitys
                adaptadores
                    CalendarioAdapter
                    IIconoCalendarioAdapter
                    IIconosAdapter
                    NotaAdapter
                    TareasAdapter
                DB
                fragmentos
                objetos
            com.larry.myagenda (androidTest)
            com.larry.myagenda (test)
        java (generated)
        assets
        res
            res (generated)
    Gradle Scripts

```

IIconoCalendarioAdapter.java

```

102     public void setOnCalendarioDoubleClickListener(OnCalendarioDoubleClickListener listener) {
103         this.doubleClickListener = listener;
104     }
105
106     1 usage
107     private List<Iconos> getIconosList() { return IconosUtils.getIconosList(); }
108
109     4 usages
110     public static class IIconoViewHolder extends RecyclerView.ViewHolder {
111         2 usages
112         ImageView iconoImageView;
113         2 usages
114         CardView cardViewIcono;
115
116         1 usage
117         public IIconoViewHolder(@NotNull View itemView) {
118             super(itemView);
119             iconoImageView = itemView.findViewById(R.id.iconoImageView);
120             cardViewIcono = itemView.findViewById(R.id.cardviewIcono);
121         }
122     }
123
124     1 usage
125     private Iconos buscarIconoPorRuta(String iconoRuta) {
126         for (Iconos icono : iconosList) {
127             if (icono.getRuta().equals(iconoRuta)) {
128                 return icono;
129             }
130         }
131         return null;
132     }
133
134     1 usage
135     public void seleccionarCalendario(Calendarario calendario) {
136         calendarioSeleccionado = calendario;
137         notifyDataSetChanged();
138         if (listener != null) {
139             listener.onCalendarioClick(calendario);
140         }
141     }
142
143     3 usages 1 implementation
144     public interface OnCalendarioClickListener {
145         2 usages 1 implementation
146         void onCalendarioClick(Calendarario calendario);
147     }
148
149     3 usages 1 implementation
150     public interface OnCalendarioDoubleClickListener {
151         1 usage 1 implementation
152         void onCalendarioDoubleClick(Calendarario calendario);
153     }
154
155     1 usage
156     private void setOnIconoClickListener(OnIconoClickListener listener) {
157         this.iconoClickListener = listener;
158     }
159
160     1 usage
161     private List<Iconos> getIconosList() { return IconosUtils.getIconosList(); }
162
163     4 usages
164     public static class IIconoViewHolder extends RecyclerView.ViewHolder {
165         2 usages
166         ImageView iconoImageView;
167         2 usages
168         CardView cardViewIcono;
169
170         1 usage
171         public IIconoViewHolder(@NotNull View itemView) {
172             super(itemView);
173             iconoImageView = itemView.findViewById(R.id.iconoImageView);
174             cardViewIcono = itemView.findViewById(R.id.cardviewIcono);
175         }
176     }
177
178     1 usage
179     private Iconos buscarIconoPorRuta(String iconoRuta) {
180         for (Iconos icono : iconosList) {
181             if (icono.getRuta().equals(iconoRuta)) {
182                 return icono;
183             }
184         }
185         return null;
186     }

```

Android

```

    app
        sampledata
        manifests
        java
            com.larry.myagenda
                activitys
                adaptadores
                    CalendarioAdapter
                    IIconoCalendarioAdapter
                    IIconosAdapter
                    NotaAdapter
                    TareasAdapter
                DB
                fragmentos
                objetos
            com.larry.myagenda (androidTest)
            com.larry.myagenda (test)
        java (generated)
        assets
        res

```

IIconoCalendarioAdapter.java

```

102     public void setOnCalendarioDoubleClickListener(OnCalendarioDoubleClickListener listener) {
103         this.doubleClickListener = listener;
104     }
105
106     1 usage
107     private List<Iconos> getIconosList() { return IconosUtils.getIconosList(); }
108
109     4 usages
110     public static class IIconoViewHolder extends RecyclerView.ViewHolder {
111         2 usages
112         ImageView iconoImageView;
113         2 usages
114         CardView cardViewIcono;
115
116         1 usage
117         public IIconoViewHolder(@NotNull View itemView) {
118             super(itemView);
119             iconoImageView = itemView.findViewById(R.id.iconoImageView);
120             cardViewIcono = itemView.findViewById(R.id.cardviewIcono);
121         }
122     }
123
124     1 usage
125     private Iconos buscarIconoPorRuta(String iconoRuta) {
126         for (Iconos icono : iconosList) {
127             if (icono.getRuta().equals(iconoRuta)) {
128                 return icono;
129             }
130         }
131         return null;
132     }

```

The screenshot shows the Android Studio interface. On the left, the file browser displays the project structure:

```

> assets
> res
  < res (generated)
> Gradle Scripts

```

The right side shows a code editor with Java code for an adapter. The code defines several interfaces and methods related to calendar selection:

```

    1 package com.larry.myagenda.adaptadores;
    2
    3 import ...
    4
    5 public class IconosAdapter extends RecyclerView.Adapter<IconosAdapter.IconoViewHolder> {
    6
    7     private List<Iconos> iconosList; // Lista de íconos
    8     private OnIconoClickListener listener; // Listener para los clics en los íconos
    9
   10    private Iconos iconoSeleccionado;
   11
   12    // Constructor del adaptador
   13    public IconosAdapter(List<Iconos> iconosList) {
   14        this.iconosList = iconosList; // Inicializamos la lista de íconos
   15    }
   16
   17    // Método llamado cuando se necesita crear un nuevo ViewHolder
   18    @NotNull
   19    @Override
   20    public IconoViewHolder onCreateViewHolder(@NotNull ViewGroup parent, int viewType) {
   21        // Inflamos el diseño del ítem y lo pasamos a un nuevo íconoViewHolder
   22        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.icono_item, parent, false);
   23        return new IconoViewHolder(view);
   24    }
   25
   26    // Método llamado para asociar datos a un ViewHolder existente
   27    @Override
   28    public void onBindViewHolder(@NotNull IconoViewHolder holder, int position) {
   29        // Obtenemos el objeto Iconos correspondiente a la posición dada
   30        Iconos ícono = iconosList.get(position);
   31        // Establecemos la imagen del ícono en el ImageView del ViewHolder
   32        holder.iconoImageView.setImageResource(ícono.getIdImagen());
   33        // Establecemos el nombre del calendario en el TextView correspondiente
   34        holder.calendarioNombre.setText(ícono.getNombre());
   35
   36        // Verificamos si este ícono es el ícono seleccionado y actualizamos su apariencia
   37        if (ícono.equals(iconoSeleccionado)) {
   38
   39            // Cambiamos la apariencia del ícono seleccionado
   40            holder.itemView.setScaleX(1.1f);
   41            holder.itemView.setScaleY(1.1f);
   42
   43        } else {
   44
   45            // Restauramos la apariencia predeterminada del ícono no seleccionado
   46            holder.itemView.setScaleX(1.0f);
   47            holder.itemView.setScaleY(1.0f);
   48
   49        }
   50
   51    }
   52
   53    // Configuramos un OnClickListener en el itemView para manejar los clics en el ícono
   54    holder.itemView.setOnClickListener(new View.OnClickListener() {
   55        @Override
   56        public void onClick(View v) {
   57            // Verificamos que el listener no sea nula y llamamos al método onIconoClick del listener
   58            if (listener != null) {
   59                listener.onIconoClick(ícono);
   60            }
   61        }
   62    });
   63
   64}

```

Anexo 15

Adaptadores IconosAdapter

The screenshot shows the Android Studio code editor with the file `IconosAdapter.java` open. The code implements a RecyclerView adapter for a list of icons. It includes methods for creating and binding view holders, and an overridden `onClick` method for the item view.

```

    1 package com.larry.myagenda.adaptadores;
    2
    3 import ...
    4
    5 public class IconosAdapter extends RecyclerView.Adapter<IconosAdapter.IconoViewHolder> {
    6
    7     private List<Iconos> iconosList; // Lista de íconos
    8     private OnIconoClickListener listener; // Listener para los clics en los íconos
    9
   10    private Iconos iconoSeleccionado;
   11
   12    // Constructor del adaptador
   13    public IconosAdapter(List<Iconos> iconosList) {
   14        this.iconosList = iconosList; // Inicializamos la lista de íconos
   15    }
   16
   17    // Método llamado cuando se necesita crear un nuevo ViewHolder
   18    @NotNull
   19    @Override
   20    public IconoViewHolder onCreateViewHolder(@NotNull ViewGroup parent, int viewType) {
   21        // Inflamos el diseño del ítem y lo pasamos a un nuevo íconoViewHolder
   22        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.icono_item, parent, false);
   23        return new IconoViewHolder(view);
   24    }
   25
   26    // Método llamado para asociar datos a un ViewHolder existente
   27    @Override
   28    public void onBindViewHolder(@NotNull IconoViewHolder holder, int position) {
   29        // Obtenemos el objeto Iconos correspondiente a la posición dada
   30        Iconos ícono = iconosList.get(position);
   31        // Establecemos la imagen del ícono en el ImageView del ViewHolder
   32        holder.iconoImageView.setImageResource(ícono.getIdImagen());
   33        // Establecemos el nombre del calendario en el TextView correspondiente
   34        holder.calendarioNombre.setText(ícono.getNombre());
   35
   36        // Verificamos si este ícono es el ícono seleccionado y actualizamos su apariencia
   37        if (ícono.equals(iconoSeleccionado)) {
   38
   39            // Cambiamos la apariencia del ícono seleccionado
   40            holder.itemView.setScaleX(1.1f);
   41            holder.itemView.setScaleY(1.1f);
   42
   43        } else {
   44
   45            // Restauramos la apariencia predeterminada del ícono no seleccionado
   46            holder.itemView.setScaleX(1.0f);
   47            holder.itemView.setScaleY(1.0f);
   48
   49        }
   50
   51    }
   52
   53    // Configuramos un OnClickListener en el itemView para manejar los clics en el ícono
   54    holder.itemView.setOnClickListener(new View.OnClickListener() {
   55        @Override
   56        public void onClick(View v) {
   57            // Verificamos que el listener no sea nula y llamamos al método onIconoClick del listener
   58            if (listener != null) {
   59                listener.onIconoClick(ícono);
   60            }
   61        }
   62    });
   63
   64}

```

```

    // Actualizamos el ícono seleccionado y notificamos los cambios
    iconoSeleccionado = icono;
    notifyDataSetChanged();
}

// Método que devuelve el número total de ítems en el conjunto de datos
@Override
public int getItemCount() { return iconosList.size(); }

// Clase interna que representa un ViewHolder para los íconos
public class IconoViewHolder extends RecyclerView.ViewHolder {

    ImageView iconoImageView; // ImageView para mostrar el ícono
    TextView calendarioNombreT; // TextView para mostrar el nombre del calendario

    // Constructor del ViewHolder
    public IconoViewHolder(@NonNull View itemView) {
        super(itemView);
        // Inicializamos las vistas
        iconoImageView = itemView.findViewById(R.id.iconImageview);
        calendarioNombreT = itemView.findViewById(R.id.calendarioNombreT);
    }
}

// Interfaz que define el método para manejar los clics en los íconos
public interface OnIconoClickListener {
    void onIconoClick(Iconos icono);
}

// Método para establecer un listener para manejar los clics en los íconos
public void setOnIconoClickListener(OnIconoClickListener listener) { this.listener = listener; }
}

```

Anexo 16

Adaptadores NotasAdapter

```

    package com.larry.myagenda.adaptadores;

import ...

public class NotaAdapter extends RecyclerView.Adapter<NotaAdapter.NotaViewHolder> {

    private List<Nota> notasList;
    private Context context;
    private OnNotaClickListener onNotaClickListener; // Interfaz

    // Interfaz para manejar clics en las notas
    public interface OnNotaClickListener {
        void onNotaClick(Nota nota);
    }

    // Método para establecer el listener de clics en las notas
    public void setOnNotaClickListener(OnNotaClickListener listener) {
        this.onNotaClickListener = listener;
    }

    // Constructor que recibe la lista de notas y el contexto
    public NotaAdapter(List<Nota> notasList, Context context) {
        this.notasList = notasList;
        this.context = context;
    }

    // Método llamado cuando se necesita crear un nuevo ViewHolder
    @Override
    public NotaViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View itemView = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.nota_item, parent, false);
        return new NotaViewHolder(itemView);
    }
}

```

```

40 // Crear y retornar una nueva instancia de NotaViewHolder con la vista inflada
41 @Override
42 public void onBindViewHolder(@NonNull NotaViewHolder holder, int position) {
43     // Obtener la nota en la posición dada
44     Nota nota = notasList.get(position);
45
46     // Asignar los datos de la nota a los elementos de la vista ViewHolder
47     holder.textViewTitulo.setText(nota.getTitulo());
48     holder.textViewDescripcion.setText(nota.getDescripcion());
49     holder.textViewContenido.setText(nota.getContenido());
50     holder.textViewFecha.setText(nota.getFecha());
51
52     // Establecer un OnClickListener para cada elemento de la lista
53     holder.itemView.setOnClickListener(new View.OnClickListener() {
54         @Override
55         public void onClick(View v) {
56             // Si se ha establecido el listener de clics en las notas, llamar a su método onNotaClick
57             if (onNotaClickListener != null) {
58                 onNotaClickListener.onNotaClick(nota);
59             }
60         }
61     });
62
63     // Método que devuelve la cantidad de elementos en la lista de notas
64 @Override
65 public int getItemCount() { return notasList.size(); }
66
67 // Clase interna que representa la vista de un elemento de la lista (ViewHolder)
68 public static class NotaViewHolder extends RecyclerView.ViewHolder {
69     // Usos
70     public TextView textViewTitulo, textViewDescripcion, textViewFecha, textViewContenido;
71
72     // Constructor que recibe la vista inflada
73     public NotaViewHolder(View view) {
74         super(view);
75         // Obtener referencias a los TextViews dentro de la vista
76         textViewTitulo = view.findViewById(R.id.textViewTitulo);
77         textViewDescripcion = view.findViewById(R.id.textViewDescripcion);
78         textViewContenido = view.findViewById(R.id.textViewContenido);
79         textViewFecha = view.findViewById(R.id.textViewFecha);
80     }
81 }
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589

```

Anexo 17

Adaptadores TareasAdapter

```

1 package com.larry.myagenda.adaptadores;
2
3 import ...
4
5 // TareasAdapter.java
6 5 usages
7
8 public class TareasAdapter extends RecyclerView.Adapter<TareasAdapter.TareaViewHolder> {
9
10    // Usos
11    private Context context;
12    private List<Tarea> listaTareas;
13    private OnTareaDoubleClickListener tareaDoubleClickListener;
14
15    // Usos
16    public TareasAdapter(Context context, List<Tarea> listaTareas) {
17        this.context = context;
18        this.listaTareas = listaTareas;
19    }
20
21    @NotNull
22    @Override
23    public TareaViewHolder onCreateViewHolder(@NotNull ViewGroup parent, int viewType) {
24        View view = LayoutInflater.from(context).inflate(R.layout.tarea_item, parent, false);
25        return new TareaViewHolder(view);
26    }
27
28    @Override
29    public void onBindViewHolder(TareaViewHolder holder, int position) {
30        Tarea tarea = listaTareas.get(position);
31        holder.textViewNombreTarea.setText(tarea.getTitulo());
32        holder.textViewDescripcionTarea.setText(tarea.getContenido());
33        holder.textViewHoraTarea.setText(tarea.getFecha());
34
35        // Manejar el doble clic en la tarea
36        holder.itemView.setOnClickListener(new View.OnClickListener() {
37            @Override
38            public void onClick(View v) {
39                // Si el tiempo actual es milisegundos
40                long clickTime = System.currentTimeMillis();
41
42                // Verificar si es un doble clic
43                if (clickTime - lastClickTime < DOUBLE_CLICK_TIME_DELTA) {
44                    // Es un doble clic, llamar al método del listener
45                    if (tareaDoubleClickListener != null) {
46                        tareaDoubleClickListener.onTareaDoubleClick(tarea);
47                    }
48                }
49            }
50        });
51    }
52
53    private static final long DOUBLE_CLICK_TIME_DELTA = 300; // Intervalo de tiempo para considerar un doble clic en milisegundos
54    private long lastClickTime = 0;
55
56    @Override
57    public void onClick(View v) {
58        // Si el tiempo actual es milisegundos
59        long clickTime = System.currentTimeMillis();
60
61        // Verificar si es un doble clic
62        if (clickTime - lastClickTime < DOUBLE_CLICK_TIME_DELTA) {
63            // Es un doble clic, llamar al método del listener
64            if (tareaDoubleClickListener != null) {
65                tareaDoubleClickListener.onTareaDoubleClick(tarea);
66            }
67        }
68    }
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
89

```

```

69     // Actualizar el tiempo del último clic
70     lastClickTime = clickTime;
71   }
72 }
73
74 @Override
75 public int getItemCount() { return listaTareas.size(); }
76
77 4 usages
78 public class TareaViewHolder extends RecyclerView.ViewHolder {
79
80   2 usages
81   TextView textViewNombreTarea;
82
83   2 usages
84   TextView textViewDescripcionTarea;
85
86   2 usages
87   TextView textViewHoraTarea;
88
89   1 usage
90   public TareaViewHolder(@NonNull View itemView) {
91     super(itemView);
92     textViewNombreTarea = itemView.findViewById(R.id.txtNombreTarea);
93     textViewDescripcionTarea = itemView.findViewById(R.id.txtDescripcionTarea);
94     textViewHoraTarea = itemView.findViewById(R.id.txtFechaHora);
95   }
96
97   // Interfaz para manejar los eventos de doble clic en las tareas
98   3 usages 1 implementation
99   public interface OnTareaDoubleClickListener {
100     1 usage
101     void onTareaDoubleClick(Tarea tarea);
102   }
103
104   // Método para establecer el listener de doble clic en la tarea
105   1 usage
106   public void setOnTareaDoubleClickListener(OnTareaDoubleClickListener listener) {
107     this.tareaDoubleClickListener = listener;
108   }
109
110   1 usage
111   public void eliminarTarea(Tarea tarea) {
112     1 usage
113     int position = listaTareas.indexOf(tarea);
114     if (position != -1) {
115       listaTareas.remove(position);
116       notifyItemRemoved(position);
117     }
118   }
119 }

```

Anexo 18

Base de datos BD

```

Android
  app
    sampledata
    manifests
    java
      com.larry.myagenda
        activity
        adaptadores
        DB
          BDCalendarios
            fragmentos
            objetos
        com.larry.myagenda (androidTest)
        com.larry.myagenda (test)
      java (generated)
      assets
      res
  Gradle Scripts

```

BDCalendarios.java

```

1 package com.larry.myagenda.DB;
2
3 > import ...
4
5 36 usages
6
7 public class BDCalendarios {
8
9   // Constructor privado para evitar instancias de la clase
10  no usages
11  private BDCalendarios() {
12
13
14
15
16   // Clase interna que define las columnas de la tabla de calendarios
17   26 usages
18   public static class CalendarioEntrada implements BaseColumns {
19     6 usages
20     public static final String TABLE_NAME = "calendarios";
21     4 usages
22     public static final String COLUMN_NOMBRE = "nombre";
23     4 usages
24     public static final String COLUMN_ICONO_RUTA = "icono_ruta";
25     4 usages
26     public static final String COLUMN_COLOR = "color";
27     3 usages
28     public static final String COLUMN_USER_ID = "user_id";
29   }
30 }

```

Android

```

app
  sampledata
  manifests
  java
    com.larry.myagenda
      activities
      adaptadores
      DB
        BDCalendarios
          fragmentos
          objetos
    com.larry.myagenda (androidTest)
    com.larry.myagenda (test)
  java (generated)
  assets
  res
  res (generated)
  Gradle Scripts

```

BDCalendarios.java

```

25  // Clase interna que define las columnas de la tabla de tareas
26  public static class TareaEntrada implements BaseColumns {
27    public static final String TABLE_NAME = "Tareas";
28    public static final String COLUMN_CALENDARIO_ID = "calendario_id";
29    public static final String COLUMN_TITULO = "titulo";
30    public static final String COLUMN_CONTENIDO = "contenido";
31    public static final String COLUMN_FECHA = "fecha";
32  }
33
34  // Clase para ayudar a gestionar la base de datos de calendarios
35  public static class CalendarioDBHelper extends SQLiteOpenHelper {
36    private static final String DATABASE_NAME = "calendarios.db";
37    private static final int DATABASE_VERSION = 1;
38
39    public CalendarioDBHelper(Context context) {
40      super(context, DATABASE_NAME, null, DATABASE_VERSION);
41    }
42
43    // Método llamado cuando se crea la base de datos por primera vez
44    @Override
45    public void onCreate(SQLiteDatabase db) {
46      // Sentencia SQL para crear la tabla de calendarios
47      final String SQL_CREATE_CAL_TABLE =
48        "CREATE TABLE " + CalendarioEntrada.TABLE_NAME + "(" +
49          CalendarioEntrada._ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
50          CalendarioEntrada.COLUMN_NOMBRE + " TEXT," +
51          CalendarioEntrada.COLUMN_ICONO_RUTA + " TEXT," +
52          CalendarioEntrada.COLUMN_COLOR + " INTEGER," +
53          CalendarioEntrada.COLUMN_USER_ID + " TEXT)";
54
55      // Sentencia SQL para crear la tabla de tareas
56      final String SQL_CREATE_NOTA_TABLE =
57        "CREATE TABLE " + TareaEntrada.TABLE_NAME + "(" +
58          TareaEntrada._ID + " INTEGER PRIMARY KEY AUTOINCREMENT," +
59          TareaEntrada.COLUMN_CALENDARIO_ID + " INTEGER," +
60          TareaEntrada.COLUMN_TITULO + " TEXT," +
61          TareaEntrada.COLUMN_CONTENIDO + " TEXT," +
62          TareaEntrada.COLUMN_FECHA + " TEXT," +
63          "FOREIGN KEY(" + TareaEntrada.COLUMN_CALENDARIO_ID + ") REFERENCES " +
64          CalendarioEntrada.TABLE_NAME + "(" + CalendarioEntrada._ID + ")";
65
66      // Ejecutar las sentencias SQL para crear las tablas
67      db.execSQL(SQL_CREATE_CAL_TABLE);
68      db.execSQL(SQL_CREATE_NOTA_TABLE);
69    }
70  }
71

```

Project

```

app
  sampledata
  manifests
  java
    com.larry.myagenda
      activities
      adaptadores
      DB
        BDCalendarios
          fragmentos
          objetos
    com.larry.myagenda (androidTest)
    com.larry.myagenda (test)
  java (generated)
  assets
  res
  res (generated)
  Gradle Scripts

```

BDCalendarios.java

```

72
73  // Método llamado cuando se actualiza la versión de la base de datos
74  @Override
75  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
76    // Eliminar las tablas existentes si existen
77    db.execSQL("DROP TABLE IF EXISTS " + CalendarioEntrada.TABLE_NAME);
78    db.execSQL("DROP TABLE IF EXISTS " + TareaEntrada.TABLE_NAME);
79
80    // Crear las nuevas tablas
81    onCreate(db);
82  }
83
84  // Método para insertar un nuevo calendario en la base de datos
85  public long insertarCalendario(String userId, String nombreCalendario, String iconoRuta, int color) {
86    SQLiteDatabase db = this.getWritableDatabase();
87
88    ContentValues values = new ContentValues();
89    values.put(CalendarioEntrada.COLUMN_NOMBRE, nombreCalendario);
90    values.put(CalendarioEntrada.COLUMN_ICONO_RUTA, iconoRuta);
91    values.put(CalendarioEntrada.COLUMN_COLOR, color);
92    values.put(CalendarioEntrada.COLUMN_USER_ID, userId);
93
94    long resultado = db.insert(CalendarioEntrada.TABLE_NAME, null, values);
95
96    db.close();
97
98    return resultado;
99  }

```

The screenshot shows the Android Studio interface with two main panes. The left pane displays the project structure under 'app' with several modules and resources. The right pane shows the source code for the file `BDCalendarios.java`.

```

// Método para obtener los calendarios del usuario actual de la base de datos
public static Cursor obtenerCalendariosDelUsuario(Context context, String userId) {
    CalendarioDBHelper dbHelper = new CalendarioDBHelper(context);
    SQLiteDatabase db = dbHelper.getReadableDatabase();

    String[] projection = {
        CalendarioEntrada._ID,
        CalendarioEntrada.COLUMN_NOMBRE,
        CalendarioEntrada.COLUMN_ICONO_RUTA,
        CalendarioEntrada.COLUMN_COLOR
    };

    String selection = CalendarioEntrada.COLUMN_USER_ID + " = ?";
    String[] selectionArgs = {userId};

    Cursor cursor = db.query(
        CalendarioEntrada.TABLE_NAME,
        projection,
        selection,
        selectionArgs,
        null,
        null,
        null
    );

    return cursor;
}

// Método para insertar una nueva nota en la base de datos
public static long insertarTarea(Context context, String idCalendario, String titulo, String contenido, String fecha) {
    CalendarioDBHelper dbHelper = new CalendarioDBHelper(context);
    SQLiteDatabase db = dbHelper.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(BDCalendarios.TareaEntrada.COLUMN_CALENDARIO_ID, idCalendario);
    values.put(BDCalendarios.TareaEntrada.COLUMN_TITULO, titulo);
    values.put(BDCalendarios.TareaEntrada.COLUMN_CONTENIDO, contenido);
    values.put(BDCalendarios.TareaEntrada.COLUMN_FECHA, fecha);

    long resultado = db.insert(BDCalendarios.TareaEntrada.TABLE_NAME, null, values);

    db.close();

    return resultado;
}

// Método para obtener tareas de calendario
public static Cursor obtenerTareasDeCalendario(Context context, String idCalendario) {
    CalendarioDBHelper dbHelper = new CalendarioDBHelper(context);
    SQLiteDatabase db = dbHelper.getReadableDatabase();

    String[] projection = {
        TareaEntrada._ID,
        TareaEntrada.COLUMN_TITULO,
        TareaEntrada.COLUMN_CONTENIDO,
        TareaEntrada.COLUMN_FECHA
    };
}

String selection = TareaEntrada.COLUMN_CALENDARIO_ID + " = ?";
String[] selectionArgs = {String.valueOf(idCalendario)};

Cursor cursor = db.query(
    TareaEntrada.TABLE_NAME,
    projection,
    selection,
    selectionArgs,
    null,
    null,
    null
);

return cursor;
}

// Método para eliminar un calendario específico
public void eliminarCalendario(String calendarioId) {
    SQLiteDatabase db = this.getWritableDatabase();

    // Eliminar las tareas asociadas al calendario
    db.delete(TareaEntrada.TABLE_NAME, whereClause, TareaEntrada.COLUMN_CALENDARIO_ID + " = ? ", new String[]{calendarioId});

    // Eliminar el calendario
    db.delete(CalendarEntrada.TABLE_NAME, whereClause, CalendarEntrada._ID + " = ? ", new String[]{calendarioId});

    db.close();
}

// Método para eliminar una tarea específica
public void eliminarTarea(String tareaId) {
    SQLiteDatabase db = this.getWritableDatabase();
    String selection = TareaEntrada._ID + " = ?";
    String[] selectionArgs = {tareaId};
    // Ejecutar la consulta de eliminación
    db.delete(TareaEntrada.TABLE_NAME, selection, selectionArgs);
    db.close();
}
}

```

Anexo 19

Fragmentos AjustesFragment

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `AjustesFragment.java` file, which extends the `Fragment` class. The code handles the creation of a view, setting up a button to close the session, and performing sign-out operations. It also includes logic to show a toast message and handle the intent to start the main activity.

```
1 package com.larry.myagenda.fragmentos;
2
3 import android.content.Intent;
4 import android.os.Bundle;
5 import androidx.fragment.app.Fragment;
6 import android.view.LayoutInflater;
7 import android.view.View;
8 import android.view.ViewGroup;
9 import android.widget.Button;
10 import android.widget.Toast;
11 import com.google.firebase.auth.FirebaseAuth;
12 import com.larry.myagenda.R;
13 import com.larry.myagenda.activitys.MainActivity;
14
15
16 public class AjustesFragment extends Fragment {
17
18     2 usages
19     private Button CerraSession;
20
21     @Override
22     public View onCreateView(LayoutInflater inflater, ViewGroup container,
23                             Bundle savedInstanceState) {
24         View view = inflater.inflate(R.layout.fragment_ajustes, container, false);
25
26         CerraSession = view.findViewById(R.id.btnAjustes); // Aquí está el cambio
27         CerraSession.setOnClickListener(new View.OnClickListener() {
28             @Override
29             public void onClick(View v) { cerrarSession(); }
30         });
31
32         return view;
33     }
34
35     Usage
36     private void cerrarSession() {
37         // Cerrar la sesión de Firebase
38         FirebaseAuth.getInstance().signOut();
39
40         // Mostrar el Toast
41         Toast.makeText(getActivity(), "Hasta pronto", Toast.LENGTH_SHORT).show();
42
43         // Redirigir a la actividad de inicio de sesión
44         Intent intent = new Intent(getActivity(), MainActivity.class);
45         intent.setFlags(intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK); // Borrar el historial de actividades
46         startActivity(intent);
47         getActivity().finish();
48     }
49 }
```

Anexo 20

Fragmentos CalendarioFragment

The screenshot shows the Android Studio interface with the project structure on the left and the code editor on the right. The code editor displays the `CalendarioFragment.java` file, which extends the `Fragment` class. The code inflates the layout, retrieves arguments for the calendar ID, and configures spinners for month and year selection. It also sets up an array adapter for the spinners.

```
1 package com.larry.myagenda.fragmentos;
2
3 > import ...
4
5
6 public class CalendarioFragment extends Fragment {
7
8     6 usages
9     private CalendarioAdapter adapter;
10    9 usages
11    private Spinner spinnerMes, spinnerAno;
12
13    @Override
14    public View onCreateView(LayoutInflater inflater, ViewGroup container,
15                            Bundle savedInstanceState) {
16        // Inflar el layout para este fragmento
17        View rootView = inflater.inflate(R.layout.fragment_calendario, container, false);
18        String calendarioId = getArguments().getString("calendarioId");
19
20        // Obtener una referencia al GridView
21        GridView gridViewCalendario = rootView.findViewById(R.id.gridViewCalendario);
22
23        // Obtener una referencia a los Spinners para seleccionar el mes y el año
24        spinnerMes = rootView.findViewById(R.id.spinnerMes);
25        spinnerAno = rootView.findViewById(R.id.spinnerAno);
26
27        // Configurar los adaptadores para los Spinners
28        ArrayAdapter<CharSequence> adapterMes = ArrayAdapter.createFromResource(requireContext(),
29                               R.array.meses_array, android.R.layout.simple_spinner_item);
30        adapterMes.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
31        spinnerMes.setAdapter(adapterMes);
32
33        // Obtener los años desde el año actual hasta 10 años en el futuro
34        int añoActual = Calendar.getInstance().get(Calendar.YEAR);
35        ArrayList<String> años = new ArrayList<>();
36        for (int i = añoActual; i <= añoActual + 10; i++) {
37            años.add(String.valueOf(i));
38        }
39    }
40 }
```

Android ▾

- ✓ app
 - sampledata
 - > manifests
 - > java
 - com.larry.myagenda
 - activitys
 - adaptadores
 - DB
 - fragmentos
 - AjustesFragment
 - CalendarioFragment
 - FragmentNota
 - HomeFragment
 - NotasFragment
 - TareasFragment
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - res (generated)

> Gradle Scripts

CalendarioFragment.java

```
37     ArrayAdapter<String> adapterAño = new ArrayAdapter<>(requireContext(),
38             android.R.layout.simple_spinner_item, años);
39         adapterAño.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
40         spinnerAño.setAdapter(adapterAño);
41
42         // Crea una instancia del adaptador de calendario y usar el contexto y los Spinners
43         adapter = new CalendarioAdapter(requireContext(), spinnerMes, spinnerAño, calendarioId);
44
45         // Configurar el adaptador en el GridView
46         gridViewCalendario.setAdapter(adapter);
47         gridViewCalendario.setOnItemClickListener(new AdapterView.OnItemClickListener() {
48
49             2 usages
50             private long lastClickTime = 0;
51
52             1 usage
53             private static final long DOUBLE_CLICK_TIME_DELTA = 300; // Valor de tiempo umbral para considerar un doble click
54
55             @Override
56             public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
57
58                 long clickTime = System.currentTimeMillis();
59                 if (clickTime - lastClickTime < DOUBLE_CLICK_TIME_DELTA) {
60                     if (clickTime - lastClickTime < DOUBLE_CLICK_TIME_DELTA) {
61                         // Se ha detectado un doble click
62                         int clickDia = (int) parent.getItemIdAtPosition(position);
63                         abrirAgregarTareaActivity(clickDia);
64                     } else {
65                         // Se ha detectado un click sencillo
66                         int clickDia = (int) parent.getItemIdAtPosition(position);
67                         abrirTareasFragment(clickDia);
68                     }
69                 }
70                 lastClickTime = clickTime;
71
72                 // Lógica existente para actualizar el día seleccionado y notificar al adaptador
73                 int clickDia = (int) parent.getItemIdAtPosition(position);
74                 adapter.setDiaSeleccionado(clickDia);
75                 adapter.notifyDataSetChanged();
76             }
77
78             1 usage
79             private void abrirTareasFragment(int clickDia) {
80
81                 // Obtener el ID del calendario del argumento
82                 String calendarioId = getArguments().getString("calendarioId");
83
84                 // Obtener el mes y año seleccionados de los Spinners
85                 int mesSeleccionado = spinnerMes.getSelectedItemPosition(); // El índice del mes en el Spinner
86                 int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
87
88                 // Crear un Bundle para pasar los argumentos al TareasFragment
89                 Bundle args = new Bundle();
90                 args.putInt("diaSeleccionado", clickDia);
91                 args.putInt("mesSeleccionado", mesSeleccionado);
92                 args.putInt("añoSeleccionado", añoSeleccionado);
93                 args.putString("calendarioId", calendarioId);
94
95                 // Crear una instancia de TareasFragment
96                 TareasFragment tareasFragment = new TareasFragment();
97                 tareasFragment.setArguments(args);
98
99                 // Utilizar el FragmentManager para reemplazar el contenido del FrameLayout en HomeFragment
100                FragmentManager fragmentManager = requireActivity().getSupportFragmentManager();
101                fragmentManager.beginTransaction()
102                    .replace(R.id.frameLayoutTareas, tareasFragment)
103                    .addToBackStack(null)
104                    .commit();
105            }
106
107            1 usage
108            private void abrirAgregarTareaActivity(int clickDia) {
109
110                 // Obtener el ID del calendario del argumento
111                 String calendarioId = getArguments().getString("calendarioId");
112
113                 // Obtener el mes y año seleccionados de los Spinners
114                 int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
115                 int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
116
117                 // Crear un Intent para iniciar la actividad AgregarTareaActivity
118                 Intent intent = new Intent(requireContext(), AgregarTareaActivity.class);
119
120                 // Pasa el dia, mes, año y ID del calendario como extras al Intent
121                 intent.putExtra("diaSeleccionado", clickDia);
122                 intent.putExtra("mesSeleccionado", mesSeleccionado);
123                 intent.putExtra("añoSeleccionado", añoSeleccionado);
124                 intent.putExtra("calendarioId", calendarioId);
125
126                 // Inicia la actividad con el Intent
127                 startActivity(intent);
128             }
129         });
130
131         // Obtener la vista raíz
132         rootView = inflater.inflate(R.layout.fragment_calendario, container, false);
133
134         // Asignar el adaptador a la GridView
135         gridViewCalendario.setAdapter(adapter);
136
137         // Asignar los botones de navegación anterior/siguiente
138         ImageButton btnAnterior = rootView.findViewById(R.id.btnAnterior);
139         ImageButton btnSiguiente = rootView.findViewById(R.id.btnSiguiente);
140
141         btnAnterior.setOnClickListener(new View.OnClickListener() {
142             @Override
143             public void onClick(View v) { anteriorMes(); }
144         });
145
146         btnSiguiente.setOnClickListener(new View.OnClickListener() {
147             @Override
148             public void onClick(View v) { siguienteMes(); }
149         });
150
151         return rootView;
152     }
153
154     1 of 1 >
155
156     if ( >
157
158         1 of 1 >
159
160         1 of 1 >
161
162         1 of 1 >
163
164         1 of 1 >
165
166         1 of 1 >
167
168         1 of 1 >
169
170         1 of 1 >
171
172         1 of 1 >
173
174         1 of 1 >
175
176         1 of 1 >
177
178         1 of 1 >
179
180         1 of 1 >
181
182         1 of 1 >
183
184         1 of 1 >
185
186         1 of 1 >
187
188         1 of 1 >
189
190         1 of 1 >
191
192         1 of 1 >
193
194         1 of 1 >
195
196         1 of 1 >
197
198         1 of 1 >
199
200         1 of 1 >
201
202         1 of 1 >
203
204         1 of 1 >
205
206         1 of 1 >
207
208         1 of 1 >
209
210         1 of 1 >
211
212         1 of 1 >
213
214         1 of 1 >
215
216         1 of 1 >
217
218         1 of 1 >
219
220         1 of 1 >
221
222         1 of 1 >
223
224         1 of 1 >
225
226         1 of 1 >
227
228         1 of 1 >
229
230         1 of 1 >
231
232         1 of 1 >
233
234         1 of 1 >
235
236         1 of 1 >
237
238         1 of 1 >
239
240         1 of 1 >
241
242         1 of 1 >
243
244         1 of 1 >
245
246         1 of 1 >
247
248         1 of 1 >
249
250         1 of 1 >
251
252         1 of 1 >
253
254         1 of 1 >
255
256         1 of 1 >
257
258         1 of 1 >
259
260         1 of 1 >
261
262         1 of 1 >
263
264         1 of 1 >
265
266         1 of 1 >
267
268         1 of 1 >
269
270         1 of 1 >
271
272         1 of 1 >
273
274         1 of 1 >
275
276         1 of 1 >
277
278         1 of 1 >
279
280         1 of 1 >
281
282         1 of 1 >
283
284         1 of 1 >
285
286         1 of 1 >
287
288         1 of 1 >
289
290         1 of 1 >
291
292         1 of 1 >
293
294         1 of 1 >
295
296         1 of 1 >
297
298         1 of 1 >
299
299         1 of 1 >
```

Android ▾

- ✓ app
 - sampledata
 - > manifests
 - > java
 - com.larry.myagenda
 - activitys
 - adaptadores
 - DB
 - fragmentos
 - AjustesFragment
 - CalendarioFragment
 - FragmentNota
 - HomeFragment
 - NotasFragment
 - TareasFragment
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - res (generated)

> Gradle Scripts

CalendarioFragment.java

```
114         .addToBackStack(null)
115         .commit();
116     }
117
118     1 usage
119     private void abrirAgregarTareaActivity(int clickDia) {
120
121         // Obtener el ID del calendario del argumento
122         String calendarioId = getArguments().getString("calendarioId");
123
124         // Obtener el mes y año seleccionados de los Spinners
125         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
126         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
127
128         // Crear un Intent para iniciar la actividad AgregarTareaActivity
129         Intent intent = new Intent(requireContext(), AgregarTareaActivity.class);
130
131         // Pasa el dia, mes, año y ID del calendario como extras al Intent
132         intent.putExtra("diaSeleccionado", clickDia);
133         intent.putExtra("mesSeleccionado", mesSeleccionado);
134         intent.putExtra("añoSeleccionado", añoSeleccionado);
135         intent.putExtra("calendarioId", calendarioId);
136
137         // Inicia la actividad con el Intent
138         startActivity(intent);
139     }
140
141     1 usage
142     private void abrirAgregarTareaActivity(int clickDia) {
143
144         // Obtener el ID del calendario del argumento
145         String calendarioId = getArguments().getString("calendarioId");
146
147         // Obtener el mes y año seleccionados de los Spinners
148         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
149         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
150
151         // Crear un Intent para iniciar la actividad AgregarTareaActivity
152         Intent intent = new Intent(requireContext(), AgregarTareaActivity.class);
153
154         // Pasa el dia, mes, año y ID del calendario como extras al Intent
155         intent.putExtra("diaSeleccionado", clickDia);
156         intent.putExtra("mesSeleccionado", mesSeleccionado);
157         intent.putExtra("añoSeleccionado", añoSeleccionado);
158         intent.putExtra("calendarioId", calendarioId);
159
160         // Inicia la actividad con el Intent
161         startActivity(intent);
162     }
163
164     1 usage
165     private void abrirTareasFragment(int clickDia) {
166
167         // Obtener el ID del calendario del argumento
168         String calendarioId = getArguments().getString("calendarioId");
169
170         // Obtener el mes y año seleccionados de los Spinners
171         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
172         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
173
174         // Crear un Intent para iniciar la actividad TareasFragment
175         Intent intent = new Intent(requireContext(), TareasFragment.class);
176
177         // Pasa el dia, mes, año y ID del calendario como extras al Intent
178         intent.putExtra("diaSeleccionado", clickDia);
179         intent.putExtra("mesSeleccionado", mesSeleccionado);
180         intent.putExtra("añoSeleccionado", añoSeleccionado);
181         intent.putExtra("calendarioId", calendarioId);
182
183         // Inicia la actividad con el Intent
184         startActivity(intent);
185     }
186
187     1 usage
188     private void abrirAgregarTareaActivity(int clickDia) {
189
190         // Obtener el ID del calendario del argumento
191         String calendarioId = getArguments().getString("calendarioId");
192
193         // Obtener el mes y año seleccionados de los Spinners
194         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
195         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
196
197         // Crear un Intent para iniciar la actividad AgregarTareaActivity
198         Intent intent = new Intent(requireContext(), AgregarTareaActivity.class);
199
200         // Pasa el dia, mes, año y ID del calendario como extras al Intent
201         intent.putExtra("diaSeleccionado", clickDia);
202         intent.putExtra("mesSeleccionado", mesSeleccionado);
203         intent.putExtra("añoSeleccionado", añoSeleccionado);
204         intent.putExtra("calendarioId", calendarioId);
205
206         // Inicia la actividad con el Intent
207         startActivity(intent);
208     }
209
210     1 usage
211     private void abrirTareasFragment(int clickDia) {
212
213         // Obtener el ID del calendario del argumento
214         String calendarioId = getArguments().getString("calendarioId");
215
216         // Obtener el mes y año seleccionados de los Spinners
217         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
218         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
219
220         // Crear un Intent para iniciar la actividad TareasFragment
221         Intent intent = new Intent(requireContext(), TareasFragment.class);
222
223         // Pasa el dia, mes, año y ID del calendario como extras al Intent
224         intent.putExtra("diaSeleccionado", clickDia);
225         intent.putExtra("mesSeleccionado", mesSeleccionado);
226         intent.putExtra("añoSeleccionado", añoSeleccionado);
227         intent.putExtra("calendarioId", calendarioId);
228
229         // Inicia la actividad con el Intent
230         startActivity(intent);
231     }
232
233     1 usage
234     private void abrirTareasFragment(int clickDia) {
235
236         // Obtener el ID del calendario del argumento
237         String calendarioId = getArguments().getString("calendarioId");
238
239         // Obtener el mes y año seleccionados de los Spinners
240         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
241         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
242
243         // Crear un Intent para iniciar la actividad TareasFragment
244         Intent intent = new Intent(requireContext(), TareasFragment.class);
245
246         // Pasa el dia, mes, año y ID del calendario como extras al Intent
247         intent.putExtra("diaSeleccionado", clickDia);
248         intent.putExtra("mesSeleccionado", mesSeleccionado);
249         intent.putExtra("añoSeleccionado", añoSeleccionado);
250         intent.putExtra("calendarioId", calendarioId);
251
252         // Inicia la actividad con el Intent
253         startActivity(intent);
254     }
255
256     1 usage
257     private void abrirTareasFragment(int clickDia) {
258
259         // Obtener el ID del calendario del argumento
260         String calendarioId = getArguments().getString("calendarioId");
261
262         // Obtener el mes y año seleccionados de los Spinners
263         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
264         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
265
266         // Crear un Intent para iniciar la actividad TareasFragment
267         Intent intent = new Intent(requireContext(), TareasFragment.class);
268
269         // Pasa el dia, mes, año y ID del calendario como extras al Intent
270         intent.putExtra("diaSeleccionado", clickDia);
271         intent.putExtra("mesSeleccionado", mesSeleccionado);
272         intent.putExtra("añoSeleccionado", añoSeleccionado);
273         intent.putExtra("calendarioId", calendarioId);
274
275         // Inicia la actividad con el Intent
276         startActivity(intent);
277     }
278
279     1 usage
280     private void abrirTareasFragment(int clickDia) {
281
282         // Obtener el ID del calendario del argumento
283         String calendarioId = getArguments().getString("calendarioId");
284
285         // Obtener el mes y año seleccionados de los Spinners
286         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
287         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
288
289         // Crear un Intent para iniciar la actividad TareasFragment
290         Intent intent = new Intent(requireContext(), TareasFragment.class);
291
292         // Pasa el dia, mes, año y ID del calendario como extras al Intent
293         intent.putExtra("diaSeleccionado", clickDia);
294         intent.putExtra("mesSeleccionado", mesSeleccionado);
295         intent.putExtra("añoSeleccionado", añoSeleccionado);
296         intent.putExtra("calendarioId", calendarioId);
297
298         // Inicia la actividad con el Intent
299         startActivity(intent);
300     }
301
302     1 usage
303     private void abrirTareasFragment(int clickDia) {
304
305         // Obtener el ID del calendario del argumento
306         String calendarioId = getArguments().getString("calendarioId");
307
308         // Obtener el mes y año seleccionados de los Spinners
309         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
310         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
311
312         // Crear un Intent para iniciar la actividad TareasFragment
313         Intent intent = new Intent(requireContext(), TareasFragment.class);
314
315         // Pasa el dia, mes, año y ID del calendario como extras al Intent
316         intent.putExtra("diaSeleccionado", clickDia);
317         intent.putExtra("mesSeleccionado", mesSeleccionado);
318         intent.putExtra("añoSeleccionado", añoSeleccionado);
319         intent.putExtra("calendarioId", calendarioId);
320
321         // Inicia la actividad con el Intent
322         startActivity(intent);
323     }
324
325     1 usage
326     private void abrirTareasFragment(int clickDia) {
327
328         // Obtener el ID del calendario del argumento
329         String calendarioId = getArguments().getString("calendarioId");
330
331         // Obtener el mes y año seleccionados de los Spinners
332         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
333         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
334
335         // Crear un Intent para iniciar la actividad TareasFragment
336         Intent intent = new Intent(requireContext(), TareasFragment.class);
337
338         // Pasa el dia, mes, año y ID del calendario como extras al Intent
339         intent.putExtra("diaSeleccionado", clickDia);
340         intent.putExtra("mesSeleccionado", mesSeleccionado);
341         intent.putExtra("añoSeleccionado", añoSeleccionado);
342         intent.putExtra("calendarioId", calendarioId);
343
344         // Inicia la actividad con el Intent
345         startActivity(intent);
346     }
347
348     1 usage
349     private void abrirTareasFragment(int clickDia) {
350
351         // Obtener el ID del calendario del argumento
352         String calendarioId = getArguments().getString("calendarioId");
353
354         // Obtener el mes y año seleccionados de los Spinners
355         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
356         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
357
358         // Crear un Intent para iniciar la actividad TareasFragment
359         Intent intent = new Intent(requireContext(), TareasFragment.class);
360
361         // Pasa el dia, mes, año y ID del calendario como extras al Intent
362         intent.putExtra("diaSeleccionado", clickDia);
363         intent.putExtra("mesSeleccionado", mesSeleccionado);
364         intent.putExtra("añoSeleccionado", añoSeleccionado);
365         intent.putExtra("calendarioId", calendarioId);
366
367         // Inicia la actividad con el Intent
368         startActivity(intent);
369     }
370
371     1 usage
372     private void abrirTareasFragment(int clickDia) {
373
374         // Obtener el ID del calendario del argumento
375         String calendarioId = getArguments().getString("calendarioId");
376
377         // Obtener el mes y año seleccionados de los Spinners
378         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
379         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
380
381         // Crear un Intent para iniciar la actividad TareasFragment
382         Intent intent = new Intent(requireContext(), TareasFragment.class);
383
384         // Pasa el dia, mes, año y ID del calendario como extras al Intent
385         intent.putExtra("diaSeleccionado", clickDia);
386         intent.putExtra("mesSeleccionado", mesSeleccionado);
387         intent.putExtra("añoSeleccionado", añoSeleccionado);
388         intent.putExtra("calendarioId", calendarioId);
389
390         // Inicia la actividad con el Intent
391         startActivity(intent);
392     }
393
394     1 usage
395     private void abrirTareasFragment(int clickDia) {
396
397         // Obtener el ID del calendario del argumento
398         String calendarioId = getArguments().getString("calendarioId");
399
400         // Obtener el mes y año seleccionados de los Spinners
401         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
402         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
403
404         // Crear un Intent para iniciar la actividad TareasFragment
405         Intent intent = new Intent(requireContext(), TareasFragment.class);
406
407         // Pasa el dia, mes, año y ID del calendario como extras al Intent
408         intent.putExtra("diaSeleccionado", clickDia);
409         intent.putExtra("mesSeleccionado", mesSeleccionado);
410         intent.putExtra("añoSeleccionado", añoSeleccionado);
411         intent.putExtra("calendarioId", calendarioId);
412
413         // Inicia la actividad con el Intent
414         startActivity(intent);
415     }
416
417     1 usage
418     private void abrirTareasFragment(int clickDia) {
419
420         // Obtener el ID del calendario del argumento
421         String calendarioId = getArguments().getString("calendarioId");
422
423         // Obtener el mes y año seleccionados de los Spinners
424         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
425         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
426
427         // Crear un Intent para iniciar la actividad TareasFragment
428         Intent intent = new Intent(requireContext(), TareasFragment.class);
429
430         // Pasa el dia, mes, año y ID del calendario como extras al Intent
431         intent.putExtra("diaSeleccionado", clickDia);
432         intent.putExtra("mesSeleccionado", mesSeleccionado);
433         intent.putExtra("añoSeleccionado", añoSeleccionado);
434         intent.putExtra("calendarioId", calendarioId);
435
436         // Inicia la actividad con el Intent
437         startActivity(intent);
438     }
439
440     1 usage
441     private void abrirTareasFragment(int clickDia) {
442
443         // Obtener el ID del calendario del argumento
444         String calendarioId = getArguments().getString("calendarioId");
445
446         // Obtener el mes y año seleccionados de los Spinners
447         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
448         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
449
450         // Crear un Intent para iniciar la actividad TareasFragment
451         Intent intent = new Intent(requireContext(), TareasFragment.class);
452
453         // Pasa el dia, mes, año y ID del calendario como extras al Intent
454         intent.putExtra("diaSeleccionado", clickDia);
455         intent.putExtra("mesSeleccionado", mesSeleccionado);
456         intent.putExtra("añoSeleccionado", añoSeleccionado);
457         intent.putExtra("calendarioId", calendarioId);
458
459         // Inicia la actividad con el Intent
460         startActivity(intent);
461     }
462
463     1 usage
464     private void abrirTareasFragment(int clickDia) {
465
466         // Obtener el ID del calendario del argumento
467         String calendarioId = getArguments().getString("calendarioId");
468
469         // Obtener el mes y año seleccionados de los Spinners
470         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
471         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
472
473         // Crear un Intent para iniciar la actividad TareasFragment
474         Intent intent = new Intent(requireContext(), TareasFragment.class);
475
476         // Pasa el dia, mes, año y ID del calendario como extras al Intent
477         intent.putExtra("diaSeleccionado", clickDia);
478         intent.putExtra("mesSeleccionado", mesSeleccionado);
479         intent.putExtra("añoSeleccionado", añoSeleccionado);
480         intent.putExtra("calendarioId", calendarioId);
481
482         // Inicia la actividad con el Intent
483         startActivity(intent);
484     }
485
486     1 usage
487     private void abrirTareasFragment(int clickDia) {
488
489         // Obtener el ID del calendario del argumento
490         String calendarioId = getArguments().getString("calendarioId");
491
492         // Obtener el mes y año seleccionados de los Spinners
493         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
494         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
495
496         // Crear un Intent para iniciar la actividad TareasFragment
497         Intent intent = new Intent(requireContext(), TareasFragment.class);
498
499         // Pasa el dia, mes, año y ID del calendario como extras al Intent
500         intent.putExtra("diaSeleccionado", clickDia);
501         intent.putExtra("mesSeleccionado", mesSeleccionado);
502         intent.putExtra("añoSeleccionado", añoSeleccionado);
503         intent.putExtra("calendarioId", calendarioId);
504
505         // Inicia la actividad con el Intent
506         startActivity(intent);
507     }
508
509     1 usage
510     private void abrirTareasFragment(int clickDia) {
511
512         // Obtener el ID del calendario del argumento
513         String calendarioId = getArguments().getString("calendarioId");
514
515         // Obtener el mes y año seleccionados de los Spinners
516         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
517         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
518
519         // Crear un Intent para iniciar la actividad TareasFragment
520         Intent intent = new Intent(requireContext(), TareasFragment.class);
521
522         // Pasa el dia, mes, año y ID del calendario como extras al Intent
523         intent.putExtra("diaSeleccionado", clickDia);
524         intent.putExtra("mesSeleccionado", mesSeleccionado);
525         intent.putExtra("añoSeleccionado", añoSeleccionado);
526         intent.putExtra("calendarioId", calendarioId);
527
528         // Inicia la actividad con el Intent
529         startActivity(intent);
530     }
531
532     1 usage
533     private void abrirTareasFragment(int clickDia) {
534
535         // Obtener el ID del calendario del argumento
536         String calendarioId = getArguments().getString("calendarioId");
537
538         // Obtener el mes y año seleccionados de los Spinners
539         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
540         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
541
542         // Crear un Intent para iniciar la actividad TareasFragment
543         Intent intent = new Intent(requireContext(), TareasFragment.class);
544
545         // Pasa el dia, mes, año y ID del calendario como extras al Intent
546         intent.putExtra("diaSeleccionado", clickDia);
547         intent.putExtra("mesSeleccionado", mesSeleccionado);
548         intent.putExtra("añoSeleccionado", añoSeleccionado);
549         intent.putExtra("calendarioId", calendarioId);
550
551         // Inicia la actividad con el Intent
552         startActivity(intent);
553     }
554
555     1 usage
556     private void abrirTareasFragment(int clickDia) {
557
558         // Obtener el ID del calendario del argumento
559         String calendarioId = getArguments().getString("calendarioId");
560
561         // Obtener el mes y año seleccionados de los Spinners
562         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
563         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
564
565         // Crear un Intent para iniciar la actividad TareasFragment
566         Intent intent = new Intent(requireContext(), TareasFragment.class);
567
568         // Pasa el dia, mes, año y ID del calendario como extras al Intent
569         intent.putExtra("diaSeleccionado", clickDia);
570         intent.putExtra("mesSeleccionado", mesSeleccionado);
571         intent.putExtra("añoSeleccionado", añoSeleccionado);
572         intent.putExtra("calendarioId", calendarioId);
573
574         // Inicia la actividad con el Intent
575         startActivity(intent);
576     }
577
578     1 usage
579     private void abrirTareasFragment(int clickDia) {
580
581         // Obtener el ID del calendario del argumento
582         String calendarioId = getArguments().getString("calendarioId");
583
584         // Obtener el mes y año seleccionados de los Spinners
585         int mesSeleccionado = obtenerMes(spinnerMes.getSelectedItem().toString());
586         int añoSeleccionado = Integer.parseInt(spinnerAño.getSelectedItem().toString());
587
588         // Crear un Intent para iniciar la actividad TareasFragment
589         Intent intent = new Intent(requireContext(), TareasFragment.class);
590
591         // Pasa el dia, mes, año y ID del calendario como extras al Intent
592         intent.putExtra("diaSeleccionado", clickDia);
593         intent.putExtra("mesSeleccionado", mesSeleccionado);
594         intent.putExtra("añoSeleccionado", añoSeleccionado);
595         intent.putExtra("calendarioId", calendarioId);
596
597         // Inicia la actividad con el Intent
598         startActivity(intent);
599     }
599         1 of 1 >
```

```

102     // Método para avanzar al mes siguiente
103     >     TUsage
104     >     private void siguienteMes() { adapter.avanzarMes(); }
107
108     // Método para retroceder al mes anterior
109     >     TUsage
110     >     private void anteriorMes() { adapter.retrocederMes(); }
113
114     TUsage
115     private int obtenerMes(String nombreMes) {
116         DateFormatSymbols dfs = new DateFormatSymbols();
117         String[] meses = dfs.getMonths();
118         for (int i = 0; i < meses.length; i++) {
119             if (meses[i].equalsIgnoreCase(nombreMes)) {
120                 return i;
121             }
122         }
123         return -1; // Si no se encuentra el nombre del mes
124     }

```

Anexo 21

Fragmentos FragmentNota

Android

- app
 - sampledadata
 - manifests
 - java
 - com.larry.myagenda
 - activity
 - adaptadores
 - DB
 - fragmentos
 - AjustesFragment
 - CalendarioFragment
 - FragmentNota**
 - HomeFragment
 - NotasFragment
 - TareasFragment
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res (generated)

Gradle Scripts

FragmentNota.java

```

1 package com.larry.myagenda.fragmentos;
2 import ...
3
4 public class FragmentNota extends Fragment {
5
6     private TextView txtViewTitulo, txtViewDescripcion, txtViewContenido, txtViewFecha;
7
8     private Button btnModificar, btnEliminar;
9
10    private Nota nota;
11
12    private static final int AGREGAR_NOTA_REQUEST_CODE = 1;
13
14    @Override
15    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
16        View rootView = inflater.inflate(R.layout.fragment_nota, container, false);
17
18        // Inicializar vistas
19        txtViewTitulo = rootView.findViewById(R.id.TVTitulo);
20        txtViewDescripcion = rootView.findViewById(R.id.TVDescripcion);
21        txtViewContenido = rootView.findViewById(R.id.TVContenido);
22        txtViewFecha = rootView.findViewById(R.id.TVFecha);
23        btnModificar = rootView.findViewById(R.id.Btn_modificar);
24        btnEliminar = rootView.findViewById(R.id.Btn_eliminar);
25
26        // Obtener la nota de los argumentos del fragmento
27        if (getArguments() != null) {
28            nota = (Nota) getArguments().getSerializable("key_nota");
29            if (nota != null) {
30                txtViewTitulo.setText(nota.getTitulo());
31                txtViewDescripcion.setText(nota.getDescription());
32                txtViewContenido.setText(nota.getContenido());
33                txtViewFecha.setText(nota.getFecha());
34            }
35        }
36
37        btnModificar.setOnClickListener(new View.OnClickListener() {
38            @Override
39            public void onClick(View v) {
40                // Crear un Intent para iniciar la actividad AgregarNota
41                Intent intent = new Intent(getActivity(), AgregarNota.class);
42
43                // Pasar la nota como extra en el Intent para la edición
44                intent.putExtra("name", "nota", nota);
45
46                // Iniciar la actividad con el intent para la edición de la nota
47                startActivityForResult(intent, AGREGAR_NOTA_REQUEST_CODE);
48            }
49        });
50
51        btnEliminar.setOnClickListener(new View.OnClickListener() {
52            @Override
53            public void onClick(View v) {
54                // Mostrar un cuadro de diálogo de confirmación
55                AlertDialog.Builder builder = new AlertDialog.Builder(requireContext());
56                builder.setTitle("Confirmar eliminación");
57                builder.setMessage("Estás seguro de que quieres eliminar esta nota?");
58                builder.setPositiveButton("Si", new DialogInterface.OnClickListener() {
59                    @Override
60                    public void onClick(DialogInterface dialog, int which) {
61                        // Ejecutar la lógica para eliminar la nota
62                    }
63                });
64            }
65        });
66    }
67
68}

```

Android

```

app
  sampledata
  manifests
  java
    com.larry.myagenda
      activities
      adaptadores
      DB
      fragmentos
        AjustesFragment
        CalendarioFragment
        FragmentNota
        HomeFragment
        NotasFragment
        TareasFragment
      objetos
    com.larry.myagenda (androidTest)
    com.larry.myagenda (test)
    java (generated)
  assets
  res
    res (generated)
  Gradle Scripts

```

FragmentNota.java

```

1 // Ejecuto la lógica para eliminar la nota
2     eliminarNota();
3   });
4 }
5 builder.setNegativeButton( text: "No", new DialogInterface.OnClickListener() {
6   @Override
7     public void onClick(DialogInterface dialog, int which) {
8       // No hacer nada, el usuario ha cancelado la operación
9     }
10   });
11 builder.show();
12 }

13 return rootView;
14 }
15 // Método para ocultar el navigationview
16 @Override
17 public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
18   super.onViewCreated(view, savedInstanceState);
19
20   // Ocultar el BottomNavigationView
21   BottomNavigationView bottomNavigationView = getActivity().findViewById(R.id.bottom_navigation);
22   bottomNavigationView.setVisibility(View.INVISIBLE);
23   Animation slideOutAnimation = AnimationUtils.loadAnimation(getApplicationContext(), R.anim.slide_out);
24   bottomNavigationView.startAnimation(slideOutAnimation);
25 }

26
27 // Método para mostrar el BottomNavigationView al salir del FragmentNota
28 @Override
29 public void onDestroyView() {
30   super.onDestroyView();
31
32   // Mostrar nuevamente el BottomNavigationView al salir del FragmentNota
33   BottomNavigationView bottomNavigationView = getActivity().findViewById(R.id.bottom_navigation);
34   bottomNavigationView.setVisibility(View.VISIBLE);
35   Animation slideInAnimation = AnimationUtils.loadAnimation(getApplicationContext(), R.anim.scale_in);
36   bottomNavigationView.startAnimation(slideInAnimation);
37 }

38
39 // Método para configurar la animación de entrada y salida del fragment
40 @Override
41 public Animation onCreateAnimation(int transit, boolean enter, int nextAnim) {
42   if (enter) {
43     // Animación de entrada personalizada
44     return AnimationUtils.loadAnimation(getApplicationContext(), R.anim.slide_in);
45   } else {
46     // Animación de salida personalizada
47     return AnimationUtils.loadAnimation(getApplicationContext(), R.anim.slide_out);
48   }
49 }

50
51 // Método para eliminar la nota
52 private void eliminarNota() {
53   if (nota != null && isAdded()) {
54     // Obtener referencia a la base de datos para la nota específica
55     DatabaseReference notaRef = FirebaseDatabase.getInstance().getReference()
56       .child( getString("usuario") ).child(FirebaseAuth.getInstance().getCurrentUser().getUid())
57       .child( getString("notas") ).child(nota.getId());
58
59   }
60 }

61
62 // Eliminar la nota de la base de datos
63 notaRef.removeValue().addOnCompleteListener(new OnCompleteListener<Void>() {
64   @Override
65   public void onComplete(@NonNull Task<Void> task) {
66     if (task.isSuccessful()) {
67       // Notificar al usuario que la nota se eliminó con éxito
68       Toast.makeText(requireContext(), (text) "Nota eliminada correctamente", Toast.LENGTH_SHORT).show();
69
70       // Cierra el fragmento actual y vuelve al fragmento anterior
71       getParentFragmentManager().popBackStack();
72     } else {
73       // Notificar al usuario si hay algún error al eliminar la nota
74       Toast.makeText(requireContext(), (text) "Error al eliminar la nota", Toast.LENGTH_SHORT).show();
75     }
76   }
77 }

78
79 // Actividad resultado
80 @Override
81 public void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
82   super.onActivityResult(requestCode, resultCode, data);
83   if (requestCode == AGREGAR_NOTA_REQUEST_CODE && resultCode == Activity.RESULT_OK) {
84     // El resultado es RESULT_OK, lo que indica que la nota se guardó exitosamente
85     // Cerrar el fragmento
86     getParentFragmentManager().popBackStack();
87   }
88 }

```

Anexo 22

Fragmentos FragmentHome

The screenshot shows the Android Studio interface. On the left, the Project Navigators pane displays the project structure under 'app'. It includes 'sampledata', 'manifests', 'java' (containing 'com.larry.myagenda' with subfolders 'activities', 'adaptadores', 'DB', 'fragmentos', and files 'AjustesFragment', 'CalendarioFragment', 'FragmentNota', 'HomeFragment', 'NotasFragment', 'TareasFragment', 'objetos', 'com.larry.myagenda (androidTest)', and 'com.larry.myagenda (test)'), 'assets', 'res', and 'Gradle Scripts'. The 'java' section is expanded, and 'HomeFragment.java' is selected, highlighted with a grey background.

The main editor window shows the code for `HomeFragment.java`:

```
1 package com.larry.myagenda.fragmentos;
2
3 > import ...
4
5 <> public class HomeFragment extends Fragment {
6
7     2 usages
8     private static final String PREF_SELECTED_CALENDAR_ID = "selected_calendar_id";
9
10    3 usages
11    private SharedPreferences sharedPreferences;
12
13    2 usages
14    private FirebaseAuth mAuth;
15
16    4 usages
17    private RecyclerView recyclerViewCalendarios;
18
19    5 usages
20    private IconoCalendarioAdapter adapter;
21
22    6 usages
23    private List<Calendario> calendarioList;
24
25    no usages
26    private List<Tarea> tareaList;
27
28    3 usages
29    private TextView nombreCalendarioTextView;
30
31    public HomeFragment() {}
32
33    @Nullable
34    @Override
35    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
36        View rootView = inflater.inflate(R.layout.fragment_home, container, false);
37
38        // Initialize Firebase Authentication
39        mAuth = FirebaseAuth.getInstance();
40
41        // Initialize SharedPreferences
42        sharedPreferences = requireActivity().getSharedPreferences("MyPrefs", Context.MODE_PRIVATE);
43
44        // Initialize RecyclerView
45        recyclerViewCalendarios = rootView.findViewById(R.id.recyclerviewCalendario);
46        recyclerViewCalendarios.setLayoutManager(new LinearLayoutManager(getActivity()));
47
48        // Initialize la lista de calendarios
49        calendarioList = new ArrayList<>();
50
51        // Initialize el TextView
52        nombreCalendarioTextView = rootView.findViewById(R.id.nombreCalendarioTextView);
53
54        FloatingActionButton fab = rootView.findViewById(R.id.fabAddCalendario);
55        fab.setOnClickListener(new View.OnClickListener() {
56            @Override
57            public void onClick(View view) {
58                // Abrir la nueva actividad de agregar calendario
59                Intent intent = new Intent(getActivity(), AgregarCalendario.class);
60                startActivity(intent);
61            }
62        });
63
64        // Ocultan o muestran el FAB dependiendo del scroll del RecyclerView
65        recyclerViewCalendarios.addOnScrollListener(new OnScrolledListener() {
66            if (dy > 0 && fab.isShown()) {
67                fab.hide();
68            } else if (dy < 0 && !fab.isShown()) {
69                fab.show();
70            }
71        });
72    }
73}
```

HomeFragment.java

```

176
177     String id = cursor.getString(cursor.getColumnIndexOrThrow(BOCalendarios.CalendarioEntrada._ID));
178     String nombre = cursor.getString(cursor.getColumnIndexOrThrow(BOCalendarios.CalendarioEntrada.COLUMN_NOMBRE));
179     String iconoRuta = cursor.getString(cursor.getColumnIndexOrThrow(BOCalendarios.CalendarioEntrada.COLUMN_ICONO_RUTA));
180     int color = cursor.getInt(cursor.getColumnIndexOrThrow(BOCalendarios.CalendarioEntrada.COLUMN_COLOR));
181
182     // Crea un objeto Calendario y agésselo a la lista
183     Calendario calendario = new Calendario(id, nombre, iconoRuta, color);
184     calendarioList.add(calendario);
185   } while (cursor.moveToNext());
186
187   // Cerrar el cursor después de usarlo
188   cursor.close();
189 }
190
191 if (adapter == null) {
192   adapter = new IconoCalendarioAdapter(calendarioList, getContext());
193   recyclerViewCalendarios.setAdapter(adapter);
194
195   // Establecer el listener para el clic en el calendario
196   adapter.setOnCalendarioClickListener(new IconoCalendarioAdapter.OnCalendarioClickListener() {
197     @Override
198     public void onCalendarioClick(Calendario calendario) {
199       // Obtener el FragmentManager
200       FragmentManager fragmentManager = requireActivity().getSupportFragmentManager();
201
202       // Crea una instancia del CalendarioFragment y configurar cualquier dato necesario
203       CalendarioFragment calendarioFragment = new CalendarioFragment();
204
205       // Pasa cualquier dato necesario al CalendarioFragment, por ejemplo, el ID del calendario seleccionado
206       Bundle args = new Bundle();
207       args.putString("calendarioId", calendario.getId());
208       calendarioFragment.setArguments(args);
209
210       // Reemplazar el contenido del FrameLayout con el CalendarioFragment
211       fragmentManager.beginTransaction()
212         .replace(R.id.frameLayoutCalendario, calendarioFragment)
213         .addToBackStack(null)
214         .commit();
215
216       nombreCalendarioTextView.setText(calendario.getNombre());
217
218       // Guardar el ID del calendario seleccionado en SharedPreferences
219       sharedPreferences.edit().putString(PREF_SELECTED_CALENDAR_ID, calendario.getId()).apply();
220
221       // Crea una instancia del TareasFragment y pasa el ID del calendario seleccionado como argumento
222       TareasFragment tareasFragment = new TareasFragment();
223       Bundle tareasArgs = new Bundle();
224       tareasArgs.putString("calendarioId", calendario.getId());
225       tareasArgs.putBoolean("mostrarTodasLasTareas", true);
226       tareasFragment.setArguments(tareasArgs);
227
228       // Utilizar el FragmentManager para reemplazar el contenido del FrameLayout en HomeFragment con el TareasFragment
229       fragmentManager.beginTransaction()
230         .replace(R.id.frameLayoutTareas, tareasFragment)
231         .addToBackStack(null)
232         .commit();
233     }
234   });
235 }

```

CalendarioFragment.java

```

284
285   // En el método onActivityCreated se va a llamar automáticamente donde configures el adaptador
286   adapter.setOnCalendarioDoubleClickListener(new IconoCalendarioAdapter.OnCalendarioDoubleClickListener() {
287     @Override
288     public void onCalendarioDoubleClick(Calendario calendario) {
289       // Mostrar un diálogo de confirmación para borrar el calendario
290       AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
291       builder.setMessage("¿Quieres que quitemos el calendario " + "(" + nombreCalendarioTextView.getText() + ") ?");
292       builder.setPositiveButton("Sí", new DialogInterface.OnClickListener() {
293         @Override
294         public void onClick(DialogInterface dialog, int which) {
295           // Eliminar el calendario
296          消除Calendar(calendario.getId());
297           Toast.makeText(getActivity(), "Calendario eliminado exitosamente", Toast.LENGTH_SHORT).show();
298           actualizarInterfazUsuario();
299         }
300       });
301
302       builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
303         @Override
304         public void onClick(DialogInterface dialog, int which) {
305           // No hacer nada, simplemente cerrar el diálogo
306           dialog.dismiss();
307         }
308       });
309
310       builder.show();
311     }
312   });
313
314   // Notificar al adaptador sobre los cambios en los datos
315   adapter.notifyDataSetChanged();
316 }
317
318 }
319
320 }
321
322 }
323
324 }
325
326 }
327
328 }
329
330 }
331
332 }
333
334 }
335
336 }
337
338 }
339
340 }
341
342 }
343
344 }
345
346 }
347
348 }
349
350 }
351
352 }
353
354 }
355
356 }
357
358 }
359
360 }
361
362 }
363
364 }
365
366 }
367
368 }
369
370 }
371
372 }
373
374 }
375
376 }
377
378 }
379
380 }
381
382 }
383
384 }
385
386 }
387
388 }
389
390 }
391
392 }
393
394 }
395
396 }
397
398 }
399
400 }
401
402 }
403
404 }
405
406 }
407
408 }
409
410 }
411
412 }
413
414 }
415
416 }
417
418 }
419
420 }
421
422 }
423
424 }
425
426 }
427
428 }
429
430 }
431
432 }
433
434 }
435
436 }
437
438 }
439
440 }
441
442 }
443
444 }
445
446 }
447
448 }
449
450 }
451
452 }
453
454 }
455
456 }
457
458 }
459
460 }
461
462 }
463
464 }
465
466 }
467
468 }
469
470 }
471
472 }
473
474 }
475
476 }
477
478 }
479
480 }
481
482 }
483
484 }
485
486 }
487
488 }
489
490 }
491
492 }
493
494 }
495
496 }
497
498 }
499
500 }
501
502 }
503
504 }
505
506 }
507
508 }
509
510 }
511
512 }
513
514 }
515
516 }
517
518 }
519
520 }
521
522 }
523
524 }
525
526 }
527
528 }
529
530 }
531
532 }
533
534 }
535
536 }
537
538 }
539
540 }
541
542 }
543
544 }
545
546 }
547
548 }
549
550 }
551
552 }
553
554 }
555
556 }
557
558 }
559
560 }
561
562 }
563
564 }
565
566 }
567
568 }
569
570 }
571
572 }
573
574 }
575
576 }
577
578 }
579
580 }
581
582 }
583
584 }
585
586 }
587
588 }
589
590 }
591
592 }
593
594 }
595
596 }
597
598 }
599
599 }
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
749 }
750
751
752
753
754
755
756
757
758
759
759 }
760
761
762
763
764
765
766
767
768
769
769 }
770
771
772
773
774
775
776
777
778
779
779 }
780
781
782
783
784
785
786
787
788
789
789 }
790
791
792
793
794
795
796
797
798
799
799 }
800
801
802
803
804
805
806
807
808
809
809 }
810
811
812
813
814
815
816
817
818
819
819 }
820
821
822
823
824
825
826
827
828
829
829 }
830
831
832
833
834
835
836
837
838
839
839 }
840
841
842
843
844
845
846
847
848
849
849 }
850
851
852
853
854
855
856
857
858
859
859 }
860
861
862
863
864
865
866
867
868
869
869 }
870
871
872
873
874
875
876
877
878
879
879 }
880
881
882
883
884
885
886
887
888
889
889 }
890
891
892
893
894
895
896
897
898
899
899 }
900
901
902
903
904
905
906
907
908
909
909 }
910
911
912
913
914
915
916
917
918
919
919 }
920
921
922
923
924
925
926
927
928
929
929 }
930
931
932
933
934
935
936
937
938
939
939 }
940
941
942
943
944
945
946
947
948
949
949 }
950
951
952
953
954
955
956
957
958
959
959 }
960
961
962
963
964
965
966
967
968
969
969 }
970
971
972
973
974
975
976
977
978
979
979 }
980
981
982
983
984
985
986
987
988
989
989 }
990
991
992
993
994
995
996
997
998
999
999 }

```

Anexo 23

Fragmentos FragmentNotas

Android Studio Project Structure:

- app
 - sampledata
 - manifests
 - java
 - com.larry.myagenda
 - activities
 - adaptores
 - DB
 - fragmentos
 - AjustesFragment
 - CalendarioFragment
 - FragmentNota
 - HomeFragment
 - NotasFragment
 - TareasFragment
 - objetos
 - com.larry.myagenda (androidTest)
 - com.larry.myagenda (test)
 - java (generated)
 - assets
 - res
 - (generated)
- Gradle Scripts

NotasFragment.java (Content)

```

1 package com.larry.myagenda.fragmentos;
2 import ...
3
4 public class NotasFragment extends Fragment {
5
6     ...
7
8     public View onCreateView(@NonNull LayoutInflater inflater,
9         ViewGroup container, Bundle savedInstanceState) {
10        ...
11
12        // Inicializar RecyclerView
13        recyclerViewNotas = root.findViewById(R.id.recyclerViewNotas);
14        recyclerViewNotas.setLayoutManager(new LinearLayoutManager(getContext(), LinearLayoutManager.VERTICAL));
15        notalist = new ArrayList<Nota>; // Inicializar la lista nota
16
17        // Obtiene el usuario actualmente autenticado
18        FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
19        if (user != null) {
20            ...
21            // Obtiene referencia a las notas del usuario en la base de datos de Firebase
22            DatabaseReference notoref = FirebaseDatabase.getInstance().getReference("usuarios").child(user.getUid()).child("notas");
23
24            // Escuchar cambios en las notas del usuario
25            notoref.addValueEventListener(new ValueEventListener() {
26                ...
27                @Override
28                public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
29                    ...
30                    // Limpia la lista de notas antes de cargar nuevas notas
31                    notalist.clear();
32                    // Iterar sobre todos las notas del usuario y agregarlas a la lista
33                    for (DataSnapshot notaSnapshot : dataSnapshot.getChildren()) {
34                        Nota nota = notaSnapshot.getValue(Nota.class);
35                        notalist.add(nota);
36                    }
37
38                    // Instancia y establecer el adaptador con los datos actualizados
39                    notaAdapter = new NotaAdapter(notalist, getContext());
40                    recyclerViewNotas.setAdapter(notaAdapter);
41
42                    // Agregar clic en las notas para abrir el FragmentNota
43                    notaAdapter.setOnNotaClickListener(new NotaAdapter.OnNotaClickListener() {
44                        ...
45                        @Override
46                        public void onNotaClick(Nota nota) { abrirFragmentNota(nota); }
47                    });
48                }
49                ...
50                @Override
51                public void onCancelled(@NonNull DatabaseError databaseError) {
52                    ...
53                    // Mostrar errores de lectura de la base de datos
54                    databaseError.toException().printStackTrace();
55                }
56            });
57        }
58    }
59}

```


FloatingActionButton.java (Content)

```

79
80     ...
81     fab.setOnTouchListener(new View.OnTouchListener() {
82         ...
83         @Override
84         public void onClick(View view) {
85             ...
86             Intent intent = new Intent(getActivity(), AgregarNota.class);
87             startActivity(intent);
88         }
89     });
90
91     ...
92     recyclerViewNotas.setOnScrollListener(new ScrollListener() {
93         ...
94         if (dy > 0 && fab.isShown()) {
95             fab.hide();
96         } else if (dy < 0 && !fab.isShown()) {
97             fab.show();
98         }
99     });
100
101     return root;
102 }
103
104     ...
105     private void abrirFragmentNota(Nota nota) {
106         FragmentManager fragmentManager = getActivity().getSupportFragmentManager();
107         FragmentNota fragmentNota = new FragmentNota();
108         Bundle args = new Bundle();
109         args.putSerializable("nota", nota);
110         fragmentNota.setArguments(args);
111         fragmentManager.beginTransaction()
112             .replace(R.id.fragment_container, fragmentNota)
113             .addToBackStack(null)
114             .commit();
115     }
116 }

```

Anexo 24

Fragmentos FragmentTareas

The screenshot shows the Android Studio interface with the file `TareasFragment.java` open. The left sidebar displays the project structure under the `app` folder, including `sampledata`, `manifests`, `java` (containing `com.larry.myagenda` with subfolders `activities`, `adaptadores`, `DB`, `fragmentos` which contains `AjustesFragment`, `CalendarioFragment`, `FragmentNota`, `HomeFragment`, `NotasFragment`, and `TareasFragment`, and `objetos`), `com.larry.myagenda (androidTest)`, `com.larry.myagenda (test)`, `java (generated)`, `assets`, `res`, and `res (generated)`, and `Gradle Scripts`. The right pane shows the Java code for `TareasFragment.java`.

```
package com.larry.myagenda.fragmentos;
import ...
public class TareasFragment extends Fragment {
    ...
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        ...
        // Recupera los argumentos
        int day = getArguments().getInt("diaSeleccionado", -1);
        int month = getArguments().getInt("mesSeleccionado", -1);
        int year = getArguments().getInt("añoSeleccionado", -1);
        String calendarioId = getArguments().getString("calendarioId");
        boolean mostrarTodasLasTareas = getArguments().getBoolean("mostrarTodasLasTareas", false);
        ...
        // Inicializar bd
        dbHelper = new BDCalendarios.CalendarioDBHelper(getContext());
        ...
        // Si no se proporcionan argumentos válidos, no hacer nada
        if (calendarioId == null) {
            return rootView;
        }
        ...
        // Cargar tareas desde la base de datos
        List<Tarea> tareas;
        ...
        // Cargar todas las tareas para el calendario
        if (mostrarTodasLasTareas || (day == -1 || month == -1 || year == -1)) {
            tareas = obtenerTodasLasTareasDeCalendario(getContext(), calendarioId);
        } else {
            // Si se proporciona una fecha específica, cargar las tareas para ese día
            tareas = obtenerTareasDeCalendarioParaDia(getContext(), calendarioId, day, month, year);
        }
        ...
        // Configurar el RecyclerView para mostrar las tareas
        RecyclerView recyclerView = rootView.findViewById(R.id.recyclerViewTareas);
        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
        adapter = new TareasAdapter(getContext(), tareas); // Asignar el adaptador a la variable de clase
        recyclerView.setAdapter(adapter);
        ...
        // Establecer el listener de doble clic en la tarea
        adapter.setOnTareaDoubleClickListener(new TareasAdapter.OnTareaDoubleClickListener() {
            ...
            @Override
            public void onTareaDoubleClick(Tarea tarea) {
                ...
                AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
                builder.setMessage("Seguro que quieres eliminar la tarea " + tarea.getTitulo() + "?")
                    .setPositiveButton("Sí", new DialogInterface.OnClickListener() {
                        ...
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            ...
                            dbHelper.eliminarTarea(tarea.getId());
                            adapter.eliminarTarea(tarea);
                            Toast.makeText(getActivity(), "Tarea eliminada exitosamente", Toast.LENGTH_SHORT).show();
                            adapter.notifyDataSetChanged();
                        }
                    })
                    .setNegativeButton("No", new DialogInterface.OnClickListener() {
                        ...
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            ...
                            dialog.dismiss();
                        }
                    })
                    .show();
            }
        });
        ...
        return rootView;
    }
}
```

The screenshot continues the code for `TareasFragment.java`. It shows the continuation of the `onCreateView` method and the implementation of the `OnTareaDoubleClickListener`.

```
    ...
    public void onClick(DialogInterface dialog, int which) {
        dbHelper.eliminarTarea(tarea.getId());
        adapter.eliminarTarea(tarea);
        Toast.makeText(getActivity(), "Tarea eliminada exitosamente", Toast.LENGTH_SHORT).show();
        adapter.notifyDataSetChanged();
    }
}
.setNegativeButton("No", new DialogInterface.OnClickListener() {
    ...
    @Override
    public void onClick(DialogInterface dialog, int which) {
        ...
        dialog.dismiss();
    }
})
.show();
}
return rootView;
}
public List<Tarea> obtenerTareasDeCalendarioParaDia(Context context, String calendarioId, int day, int month, int year) {
    List<Tarea> listaTareas = new ArrayList<>();
    ...
    // Formatear la fecha del día específico en el formato esperado (dd/MM/yyyy)
    String fechaDiaString = String.format(Locale.getDefault(), "%02d/%02d/%04d", day, month + 1, year);
    ...
    // Llamar al método estático de la clase BDCalendarios para obtener las tareas del calendario específico
    Cursor cursor = BDCalendarios.CalendarioDBHelper.obtenerTareasDeCalendario(context, calendarioId);
    ...
    // Verificar si se encontraron tareas para este calendario
    if (cursor != null && cursor.moveToFirst()) {
        do {
            ...
        }
    }
}
}
```

```

110     // Obtener los datos de la tarea de la base de datos
111     String id = cursor.getString(cursor.getColumnIndexOrThrow(BDCalendarios.TareaEntrada._ID));
112     String titulo = cursor.getString(cursor.getColumnIndexOrThrow(BDCalendarios.TareaEntrada.COLUMN_TITULO));
113     String contenido = cursor.getString(cursor.getColumnIndexOrThrow(BDCalendarios.TareaEntrada.COLUMN_CONTENIDO));
114     String fecha = cursor.getString(cursor.getColumnIndexOrThrow(BDCalendarios.TareaEntrada.COLUMN_FECHA));
115
116     // Extraer solo la parte de la fecha (dd/MM/yyyy) para la comparación
117     String fechaSolo = fecha.split(" ")[0];
118
119     // Extrae las tareas para incluir solo las del dia específico
120     if (fechaSolo.equals(fechaAsString)) {
121         // Crea un objeto Tarea con los datos obtenidos y añadirlo a la lista
122         Tarea tarea = new Tarea(id, calendarioId, titulo, contenido, fecha); // Cambia calendarioId a String
123         listaTareas.add(tarea);
124     }
125 } while (cursor.moveToNext());
126 // Cerrar el cursor
127 cursor.close();
128
129
130 return listaTareas;
131 }
132
133 public List<Tarea> obtenerTodasLasTareasDeCalendario(Context context, String calendarioId) {
134     List<Tarea> listaTareas = new ArrayList<>();
135
136     // Llamar al método estático de la clase BDCalendarios para obtener todas las tareas del calendario específico
137     Cursor cursor = BDCalendarios.CalendarioDBHelper.obtenerTodasLasTareasDeCalendario(context, calendarioId);
138
139     // Verifico si se encontraron tareas para este calendario
140     if (cursor != null && cursor.moveToFirst()) {
141         do {
142             // Obtener los datos de la tarea de la base de datos
143             String id = cursor.getString(cursor.getColumnIndexOrThrow(BDCalendarios.TareaEntrada._ID));
144             String titulo = cursor.getString(cursor.getColumnIndexOrThrow(BDCalendarios.TareaEntrada.COLUMN_TITULO));
145             String contenido = cursor.getString(cursor.getColumnIndexOrThrow(BDCalendarios.TareaEntrada.COLUMN_CONTENIDO));
146             String fecha = cursor.getString(cursor.getColumnIndexOrThrow(BDCalendarios.TareaEntrada.COLUMN_FECHA));
147
148             // Crear un objeto Tarea con los datos obtenidos y añadirlo a la lista
149             Tarea tarea = new Tarea(id, calendarioId, titulo, contenido, fecha); // Cambia calendarioId a String
150             listaTareas.add(tarea);
151         } while (cursor.moveToNext());
152         // Cerrar el cursor
153         cursor.close();
154     }
155
156     return listaTareas;
157 }
158

```

Anexo 25

Objetos Calendario

```

Android
app
sampledata
manifests
java
com.larry.myagenda
activitys
adaptores
DB
fragmentos
objetos
    Calendario
    Iconos
    IconosUtils
    Nota
    Tarea
    Usuarios
com.larry.myagenda (androidTest)
com.larry.myagenda (test)
java (generated)
assets
res

Calendario.java
package com.larry.myagenda.objetos;
import ...
1 usages
public class Calendario {
2     private String id;
3     private String nombre;
4     private String iconoRuta;
5     private int color;
6
7     // Constructor para crear un objeto Calendario con todos los atributos
8     public Calendario(String id, String nombre, String iconoRuta, int color) {
9         this.id = id;
10        this.nombre = nombre;
11        this.iconoRuta = iconoRuta;
12        this.color = color;
13    }
14
15    // Métodos para acceder a los datos del calendario
16    public String getId() { return id; }
17
18    public String getNombre() { return nombre; }
19
20    public String getIconoRuta() { return iconoRuta; }
21
22    public int getColor() { return color; }
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59 usages

```

Anexo 26

Objetos Iconos

The screenshot shows the Android Studio project structure on the left and the code editor on the right. The code editor displays the `Iconos.java` file.

```
Android ▾
  ↘ app
    ↗ sampledata
    > ↗ manifests
    ↘ java
      ↗ com.larry.myagenda
        > ↗ activitys
        > ↗ adaptadores
        > ↗ DB
        > ↗ fragmentos
        > ↗ objetos
          ↗ Calendario
          ↗ Iconos
          ↗ IconosUtils
          ↗ Nota
          ↗ Tarea
          ↗ Usuarios
        > ↗ com.larry.myagenda (androidTest)
        > ↗ com.larry.myagenda (test)
        ↗ java (generated)
      > ↗ assets
      > ↗ res
```

```
Iconos.java ×
1 package com.larry.myagenda.objetos;
2
3     27 usages
4 public class Iconos {
5     3 usages
6     private int imagenId;
7     3 usages
8     private String nombre;
9     3 usages
10    private String ruta; |
11
12    8 usages
13    public Iconos(int imagenId, String nombre, String ruta) {
14        this.imagenId = imagenId;
15        this.nombre = nombre;
16        this.ruta = ruta;
17    }
18
19    2 usages
20    public int getImagenId() { return imagenId; }
21
22    17 > public String getNombre() { return nombre; }
23
24    2 usages
25    public String getRuta() { return ruta; }
26
27    no usages
28    public void setImagenId(int imagenId) { this.imagenId = imagenId; }
29
30    29 > public void setNombre(String nombre) { this.nombre = nombre; }
31
32    no usages
33    public void setRuta(String ruta) { this.ruta = ruta; }
34}
```

Anexo 27

Objetos Nota

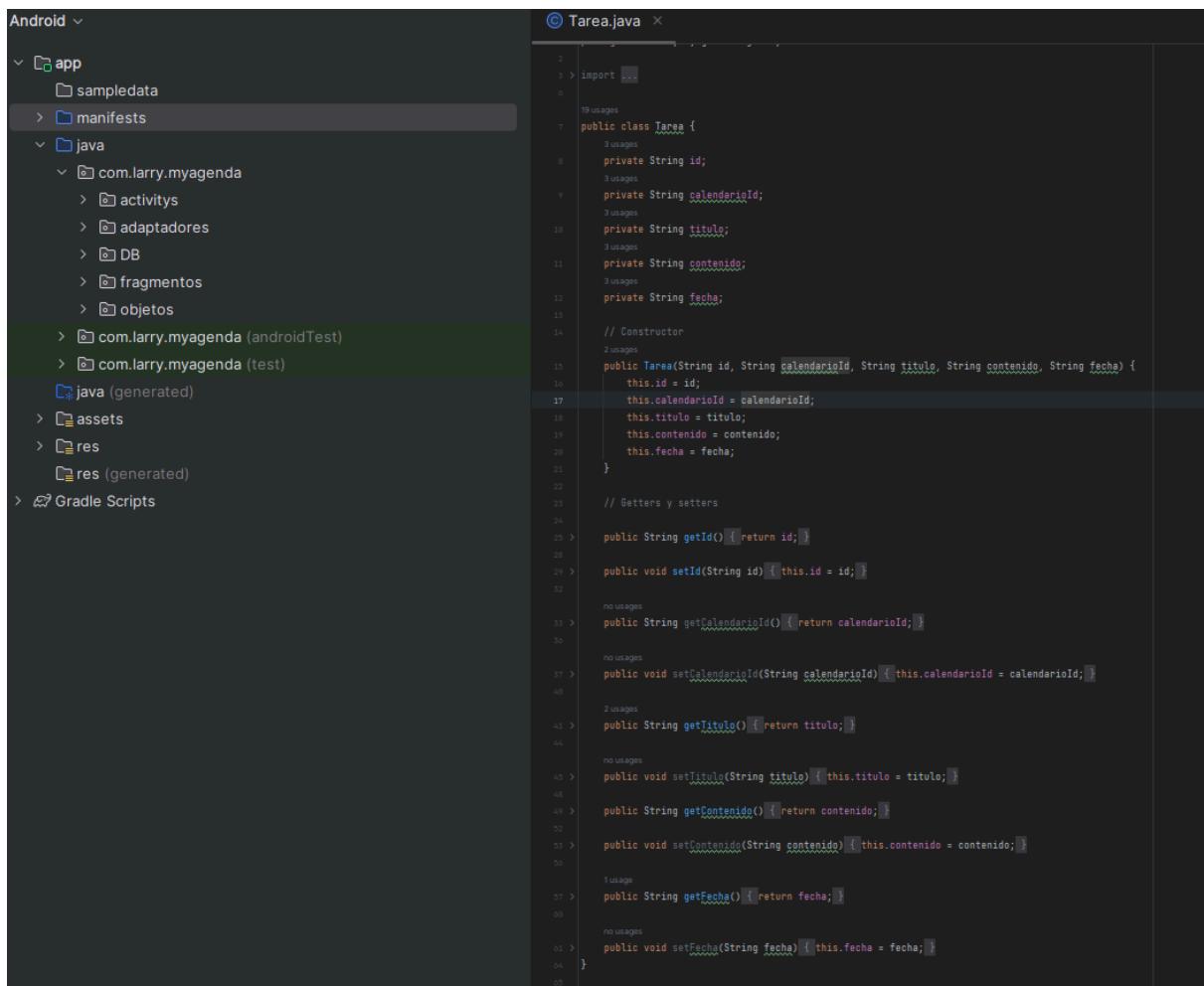
The screenshot shows the Android Studio project structure on the left and the code editor on the right. The code editor displays the `Nota.java` file.

```
Android ▾
  ↘ app
    ↗ sampledata
    > ↗ manifests
    ↘ java
      ↗ com.larry.myagenda
        > ↗ activitys
        > ↗ adaptadores
        > ↗ DB
        > ↗ fragmentos
        > ↗ objetos
          ↗ Calendario
          ↗ Iconos
          ↗ IconosUtils
          ↗ Nota
          ↗ Tarea
          ↗ Usuarios
        > ↗ com.larry.myagenda (androidTest)
        > ↗ com.larry.myagenda (test)
        ↗ java (generated)
      > ↗ assets
      > ↗ res
      > ↗ res (generated)
    > ↗ Gradle Scripts
```

```
Nota.java ×
1 19 usages
2 public class Nota implements Serializable {
3     3 usages
4     private String id;
5     3 usages
6     private String titulo;
7     3 usages
8     private String descripcion;
9     3 usages
10    private String contenido;
11    3 usages
12    private String fecha;
13
14    no usages
15    public Nota() {
16        // Constructor vacío
17    }
18
19    1 usage
20    public Nota(String id, String titulo, String descripcion, String contenido, String fecha) {
21        this.id = id;
22        this.titulo = titulo;
23        this.descripcion = descripcion;
24        this.contenido = contenido;
25        this.fecha = fecha;
26    }
27
28    // Getters y setters
29    public String getId() {return id;}
30
31    public void setId(String id) {this.id = id;}
32
33    4 usages
34    public String getTitulo() {return titulo;}
35
36    no usages
37    public void setTitulo(String titulo) {this.titulo = titulo;}
38
39    4 usages
40    public String getDescripcion() {return descripcion;}
41
42    no usages
43    public void setDescription(String descripcion) {this.descripcion = descripcion;}
44
45    public String getContenido() {return contenido;}
46
47    2 usages
48    public String getFecha() {return fecha;}
49
50    no usages
51    public void setFecha(String fecha) {this.fecha = fecha;}
52}
```

Anexo 28

Objetos Tarea



The screenshot shows the Android Studio interface. On the left, the project structure is displayed under the 'Android' tab, showing the app module with its sub-directories: sampledata, manifests, java (which is expanded to show com.larry.myagenda, assets, res, and generated), and Gradle Scripts. The 'java' directory is currently selected. On the right, the code editor displays the 'Tarea.java' file. The code defines a class 'Tarea' with private fields for id, calendarId, title, contenido, and fecha. It includes a constructor, getters and setters for each field, and usage counts for each field.

```
2 > import ...;
0
19 usages
7 public class Tarea {
8     3 usages
9     private String id;
10    3 usages
11     private String calendarId;
12    3 usages
13     private String titulo;
14    3 usages
15     private String contenido;
16    3 usages
17     private String fecha;
18
19 // Constructor
20
21 public Tarea(String id, String calendarId, String titulo, String contenido, String fecha) {
22     this.id = id;
23     this.calendarId = calendarId;
24     this.titulo = titulo;
25     this.contenido = contenido;
26     this.fecha = fecha;
27 }
28
29 // Getters y setters
30
31 public String getId() { return id; }
32
33 public void setId(String id) { this.id = id; }
34
35 no usages
36 public String getCalendarId() { return calendarId; }
37
38 no usages
39 public void setCalendarId(String calendarId) { this.calendarId = calendarId; }
40
41 public String getTitulo() { return titulo; }
42
43 no usages
44 public void setTitulo(String titulo) { this.titulo = titulo; }
45
46 public String getContenido() { return contenido; }
47
48 public void setContenido(String contenido) { this.contenido = contenido; }
49
50
51 public String getFecha() { return fecha; }
52
53 no usages
54 public void setFecha(String fecha) { this.fecha = fecha; }
55 }
```

Anexo 29

Objetos Usuarios

The screenshot shows the Android Studio interface. On the left, the project structure is displayed under 'Android'. It includes the 'app' module with its sub-directories: 'sampledata', 'manifests', 'java', and 'res'. The 'java' directory contains packages for 'com.larry.myagenda' and 'objetos'. Under 'objetos', there are several classes: 'Calendario', 'Iconos', 'IconosUtils', 'Nota', 'Tarea', and 'Usuarios'. The 'Usuarios' class is selected and shown in the main editor window. The code for 'Usuarios.java' is as follows:

```
1 package com.larry.myagenda.objetos;
2
3 usages ▾ 14rry9912
4 public class Usuarios {
5     3 usages
6     private String nombre;
7     3 usages
8     private String correo;
9     3 usages
10    private String contraseña;
11    no usages
12    private String uid;
13
14    no usages ▾ 14rry9912
15    public Usuarios() {
16        // Constructor vacío requerido por Firebase
17    }
18
19    1 usage ▾ 14rry9912
20    public Usuarios(String nombre, String correo, String contraseña) {
21        this.nombre = nombre;
22        this.correo = correo;
23        this.contraseña = contraseña;
24    }
25
26    // Métodos getters y setters
27    ▾ 14rry9912
28    public String getNombre() { return nombre; }
29
30    ▾ 14rry9912
31    public void setNombre(String nombre) { this.nombre = nombre; }
32
33    no usages ▾ 14rry9912
34    public String getCorreo() { return correo; }
35
36    no usages ▾ 14rry9912
37    public void setCorreo(String correo) { this.correo = correo; }
38
39    no usages ▾ 14rry9912
40    public String getContraseña() { return contraseña; }
41
42    no usages ▾ 14rry9912
43    public void setContraseña(String contraseña) { this.contraseña = contraseña; }
44}
```