Proyecto: Desarrollo de una interfaz

Melba Pérez Santana y Raúl Medina Rodríguez

Índice

Introducción

Nuestra aplicación llamada Vitalize, está relacionada con el área de la salud y el fitness. Pretende ofrecer a sus usuarios la facilidad de controlar las kilocalorías, proteínas, grasas… que consumen, así como sus objetivos. Asimismo, brinda la posibilidad de inspirar o aconsejar que productos consumir de su despensa para completar su dieta u objetivos. Para hacer uso de las funcionalidades que nuestra aplicación brinda es necesario registrarse e iniciar sesión. A continuación, se explicará el proceso de la creación de nuestra aplicación, así como las herramientas utilizadas y el desarrollo en sí.

Desarrollo

Fases del desarrollo

Intervalo de tiempo: Septiembre – octubre

Estudio

Nos formamos en Kotlin y seguimos diferentes Codelabs, descritos en el read me del GitHub del siguiente enlace: https://github.com/melbaperez/CodeLabs . Se realizó la lección 1: Compila tu primera app, lección 2: Diseños, lección 3: Navegación y la lección 4: Ciclos de vida de actividades y fragmentos. Estas lecciones son de los antiguos Codelabs.

Intervalo de tiempo: 24 – 30 octubre

Elección del proyecto

En primer lugar, además de, se estudió y analizó diferentes ideas para aplicar como objetivo de nuestro proyecto. Finalmente se decidió por una aplicación relacionada con el área de fitness y salud. Esta idea, consideramos que era la más apropiada ya que, podía ser usada por cualquier persona, no estimamos que esté solamente enfocada a un grupo de edad.

Elección de la metodología a seguir

Se ha decidido seguir la metodología SCRUM. De esta forma, se fomentará el trabajo en equipo y podremos llevar a cabo la realización de diferentes tareas con el objetivo de la aplicación.

Creación del proyecto. Desarrollo inicial

Habiendo decidido realizar el proyecto con la metodología SCRUM, se creó la tabla del Trello para la creación de la pila de productos. Así como el proyecto en Android Studio y la subida de este a GitHub. Por otro lado, se creó un proyecto en Firebase.

Análisis y establecimiento de los requisitos de la aplicación

Estableciendo lo que queríamos que realizase nuestra aplicación, cuál era el objetivo de esta y que pretendíamos con ella, se determinaron las historias de usuario y los requisitos de nuestra aplicación. Llegando a una compresión adecuada de los requerimientos del sistema.

Intervalo de tiempo: 31 octubre – 13 noviembre

Diseños

Se realizaron los diseños esenciales o al menos de los que nos íbamos a encargar en estas dos primeras semanas. Cabe destacar, que antes que nada fue necesario establecer los colores y tipografías a utilizar. Asimismo, tuvimos que diseñar un fondo, el logo de nuestra aplicación y también el diseño de nuestros botones. Teniendo en cuenta lo que queríamos ofrecer, diseñamos una pantalla principal con una llamada a la acción. Además, se ha considerado necesario que nuestros usuarios se registren para poder hacer uso de Vitalize.

Estudio

De forma paralela se continuó realizando los codelabs. No obstante, de los nuevos Codelabs que ofrece Android realizamos la unidad 3: Navegación y la unidad 4: Conexión a internet. Cabe destacar que, al empezar con la parte de los usuarios de nuestra aplicación, se le dedicó tiempo extra al estudio de Firebase Auth/Firestore/Storage para Android.

Desarrollo

En el desarrollo, primero se tuvieron que realizar los templates de los diseños realizados. Luego, la funcionalidad necesaria para acceder al perfil, ver sus datos, iniciar sesión, registrarse, cerrar sesión. Teniendo en cuenta siempre controlar los accesos o registros fallidos.

Documentación

En el *read me* del proyecto en GitHub se puede observar todos los vídeos o páginas visitados o vistos para obtener el conocimiento necesario para realizar las tareas. Asimismo, como datos importantes a conocer de nuestro proyecto, ya sea bases de datos utilizadas.

Configuración de base de datos

Creamos la estructura de nuestra base de datos y habilitamos la autentificación de usuarios mediante correo y contraseña. Asimismo, para poder tener información que mostrar en nuestra aplicación, se empezó rellenando datos sobre los alimentos, sus kilocalorías, grasas, proteínas, carbohidratos...

Reunión

Los jueves en clase hablábamos y compartíamos lo realizado como íbamos y dificultades que podría sufrir el otro.

Análisis y establecimiento de los requisitos de la aplicación

Al finalizar el Sprint analizamos la pila de productos y observábamos nuestro rendimiento. Entonces determinamos las nuevas tareas que íbamos a asignar para el siguiente Sprint. Así como creábamos posibles nuevas historias de usuario a realizar.

Intervalo de tiempo: 14 – 27 noviembre

Diseños

Se realizaron los diseños de otras páginas de la aplicación para poder seguir con la lógica y las funcionalidades de nuestro proyecto.

Estudio

Mientras desarrollábamos las funcionalidades de nuestro proyecto, funciones como acceso a la galería para subir una foto o edición de los datos de autentificación o de la base de datos, fue necesario buscar más información. Por lo que fue necesario formarnos más con estas funciones.

Desarrollo

En el desarrollo, primero se tuvieron que realizar los templates de los diseños realizados. Luego, se evolucionó la aplicación, creando conexión y la posibilidad de navegar entre las páginas de nuestra aplicación. Por otro lado, se completó el perfil de usuario para poder editar datos, así como la foto de perfil. Asimismo, se implementó la modificación de la dieta diaria.

Documentación

En el *read me* del proyecto en GitHub se puede observar todos los vídeos o páginas visitados o vistos para obtener el conocimiento necesario para realizar las tareas de este segundo Sprint.

Configuración de base de datos

Se rellenó la base de datos con las kilocalorías al realizar 60 minutos de diferentes deportes.

Reunión

Los jueves de estas dos semanas compartíamos lo realizado, cómo íbamos y consejos, así como dudas.

Análisis y establecimiento de los requisitos de la aplicación

Al finalizar el Sprint analizamos la pila de productos y determinamos las últimas tareas que íbamos a realizar en el siguiente Sprint y último.

Intervalo de tiempo: 28 noviembre – 11 diciembre

Estudio

Durante el desarrollo, se buscaba ciertos aspectos o funciones necesarias para realizar las tareas, con vídeos, blogs o tutoriales nos formamos.

Desarrollo

Mayoritariamente este último Sprint todas las tareas son de desarrollo, funcionalidades como buscar un producto, la despensa del usuario, como el escaneo de un producto son funcionalidades que son necesarias para cumplir con los objetivos de nuestra aplicación, entre otros.

Documentación

En el *read me* del proyecto en GitHub apuntamos todos los vídeos o páginas webs vistos o visitados, para poder realizar las tareas de este Sprint.

Reunión

Los jueves durante las clases de prácticas, compartíamos lo que habíamos hecho, y nos explicábamos brevemente en que punto nos encontrábamos.

Intervalo de tiempo: 12 diciembre – 18 diciembre

Estudio

La fase de estudio está muy vinculada con el desarrollo, puesto que consideramos que cuando buscamos y nos informamos para desarrollar algo, estamos estudiando y formándonos.

Desarrollo

Finalizamos y mejoramos funciones, como generar la dieta a partir de los productos de la despensa, o inclusive cambiar las calorías objetivo.

Documentación

Completamos un poco el informe, así como el GitHub.

Reunión

Realizamos varias reuniones, para comentar lo necesario para la presentación del producto, así como los últimos detalles de la aplicación a tener en cuenta.

Pruebas y corrección de errores

Realizamos las acciones básicas que realizaba nuestra aplicación, encontramos errores como que se inicializaba el *userViewModel* aun no habiendo usuario registrado o iniciado sesión y la aplicación colapsaba. Entonces, corregimos todo esto, así como quitar *Logs*, y los *imports* que no se usaban.

Configuración de base de datos

Se añadió a la base de datos fotos a los productos, así como se cambió los nombres de los productos de inglés a español.

Temporización

Como se especificó anteriormente se ha seguido la metodología SCRUM para organizar las tareas. Se ha decidido que cada integrante realice 9 horas semanales, siendo 2 horas de práctica en clase y 7 horas en casa. Al ser dos integrantes en nuestro equipo y que cada sprint tendrá una duración de 2 semanas. Cada Sprint tendrá una duración de 36 horas. No obstante, se ha contado con una semana extra, que se ha incorporado para continuar las tareas del último Sprint, así como mejorar la aplicación en cuanto a organización de los ficheros. Por otra parte, es necesario indicar que se puso a prueba toda la aplicación pudiendo ver el conjunto final de esta.

Para la temporización se puede observar la tabla de la siguiente página horizontal en la que se visualiza las diferentes semanas, así como las diferentes tipos de tareas y la dedicación de horas de trabajo a cada fase por persona. Cabe destacar, que sobre todo durante los Sprints las horas de estudio estaban muy vinculadas a las horas de desarrollo. La última semana, algunas de las tareas realizadas se añadieron a la pila de productos del Sprint 3. No obstante, las pruebas por ejemplo no se adjuntaron.

Por otro lado, hemos creado un resumen de las tareas realizadas indicando la fase a la que corresponde y las horas dedicadas a dicha tarea. Cabe destacar, que la mayoría de las tareas que son parte de la fase de desarrollo también son parte de la fase de estudio. Igualmente, las tareas realizadas en la última semana no se han especificado en el Sprint 3.

Fases	**Sept-Oct**	**0** (24-30 oct)	**1** (31oct -6 nov)	**2** (7-13 nov)	**3** (14-20 nov)	**4** (21-27 nov)	**5** (28nov -4dic)	**6** (5-11 dic)	**7**(12-18 dic)
Estudio	36 horas		4 horas	3 horas	2 horas	2 horas	4 horas	4 horas	4 horas
Elección del proyecto		1 hora							
Elección de la metodología a seguir		30 minutos							
Creación del proyecto. Desarrollo inicial		1 hora							
Análisis y establecimiento de los requisitos de la aplicación		2 horas		30 minutos		30 minutos			
Diseños			2,5 horas		2,5 horas				
Desarrollo			2 horas	5 horas	4 horas	7 horas	6 horas	6 horas	6 horas
Documentación				15 minutos		15 minutos		30 minutos	30 minutos
Configuración de base de datos			5 horas		2 horas				1 hora
Reunión			15 minutos	15 minutos	15 minutos	15 minutos	15 minutos	15 minutos	15 minutos
Pruebas y corrección de errores									30 minutos

Tabla 1. Dedicación de horas a cada fase por persona

SPRINT 1	FASE	HORAS
HT - Establecer estructura de datos de la base de datos no relacional (Cloud Firestore)	Configuración de base de datos	4
HT - Diseño de Templates I (Figma)	Diseños	5
HT - Establecer conjunto mínimo base de datos alimentos	Configuración de base de datos	6
HT - Diseño de Templates I (Android)	Desarrollo	5
HT - Estudio de Firebase Auth/Firestore/Storage para Android	Estudio	6
HU - Acceso al perfil del usuario	Desarrollo	4
HU - Inicio de sesión	Desarrollo	2
HU - Registro de usuario	Desarrollo	3
HU - Logout	Desarrollo	1

Tabla 2. Fase y horas de las tareas del Sprint 1

SPRINT 2	FASE	HORAS
HT - Establecer conjunto mínimo base de datos ejercicios	Configuración de base de datos	4
HT - Diseño de Templates II (Figma)	Diseños	5
HU - Navegación	Desarrollo	6
HT - Diseño de Templates II (Android)	Desarrollo	5
HU - Edición del perfil de usuario	Desarrollo	3
HU - Eliminar foto del perfil	Desarrollo	3
HU - Añadir foto del perfil desde galería	Desarrollo	2
HU - Modificación de la dieta diaria	Desarrollo	8

Tabla 3. Fase y horas de las tareas del Sprint 2

SPRINT 3	FASE	HORAS
HU - Escaneo de producto	Desarrollo	7
HU - Búsqueda de producto	Desarrollo	6
HU - Despensa de usuario	Desarrollo	5
HU - Modificar objetivos nutricionales manualmente	Desarrollo	4
HU - Visualizar progreso de los objetivos nutricionales del día en la dieta	Desarrollo	3
HU - Generar dieta a partir de los productos de la despensa del usuario	Desarrollo	8
HT - Adaptación del diseño al móvil físico	Desarrollo	2
HT - Añadir urls a las fotos de los alimentos y cambio de nombre de inglés a español	Configuración de base de datos	1

Tabla 4. Fase y horas de las tareas del Sprint 3

Herramientas

Como se comentó anteriormente, se ha seguido la metodología SCRUM, para ello se utilizó Trello para crear las historias de usuario y repartir las tareas entre los miembros del proyecto. A continuación, adjuntamos el link de acceso para visualizar las tareas.

https://trello.com/invite/b/IxO11mcK/ATTI8fc4920c79331750668e59bf4560f1baF5239DA2/proyecto-pamn

Por otra parte, en cuanto a la documentación del trabajo los cambios que íbamos desarrollando en nuestra *app* se subían al GitHub. En el *read me* del repositorio se ha indicado el *road map* realizado. Asimismo, en este mismo documento, se ha documentado gran parte de lo realizado. Adjuntamos de nuevo el link al repositorio.

https://github.com/RaulMedR/Vitalize-PAMN

Cabe destacar, que antes de programar realizamos los diseños en Figma. Sin embargo, los diseños de las páginas finales no son fieles a los diseños en Figma. Igualmente, tener como base los diseños nos ayudó a definir las funcionalidades de nuestra aplicación, así como ir mucho más rápidos a la hora de crear los layouts.

https://www.figma.com/file/0ZWGy7TeletOhy60fIXpBS/Vitalize-PAMN?node-id=80%3A199&t=kWaxL0vFdkMVVfBa-1

En cuanto a los datos de nuestra aplicación, se utilizó Firebase. Como se comenta anteriormente para incorporar datos en nuestra base de datos utilizamos otras bases de datos cuya información era útil para el propósito de nuestra aplicación. Se añade a continuación un link, no obstante, se debe tener permisos los cuales se han cedido para poder acceder:

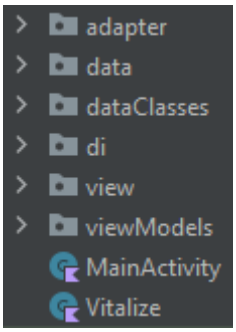
https://console.firebase.google.com/u/0/project/vitalize-93ee6/overview

Tanto en Firebase como Trello, se le ha invitado al correo de la universidad para que pueda ser visualizadora o editora.

Por último, como IDE utilizamos Android Studio. Se compiló la app mostrándola tanto en un dispositivo físico como virtual. Todos los cambios que realizábamos lo subíamos a GitHub. De esta forma podíamos trabajar a la vez, creando varias ramas para las tareas a realizar. Cabe destacar, que se hizo uso de una aplicación online que convertía imágenes de png a svg. Utilizamos las imágenes vectoriales por todas las ventajas que nos ofrecía como nitidez, dinamismo…

Desarrollo / ejecución

Estructura del proyecto



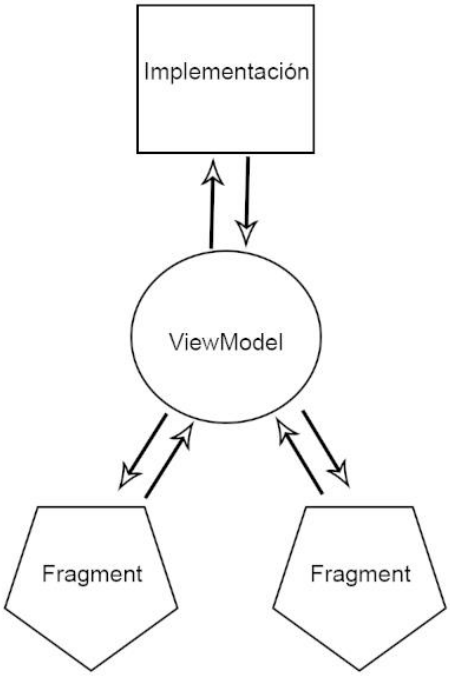
Captura 1 - Estructura del proyecto en Android Studio

El proyecto se estructura en 6 carpetas, la aplicación, y la única actividad de la aplicación.

- **Adapter:** Se encuentra la lógica de los 3 *Adapter*, el de la página principal, la despensa y el de la búsqueda, cargando el respectivo layout del *CardView* para cada uno, para usarlo posteriormente en los *RecyclerView* correspondientes.
- **Data:** Se encuentra las interfaces y las implementaciones de Firebase Auth y Firestore con los métodos que hemos necesitado de cada uno, incluyendo la creación del *Resource* que se usa para la ejecución de las corrutinas. Además, dentro se encuentra la carpeta *utils* con un fichero Kotlin en el que se ha creado una función que devuelve el resultado de la ejecución de un método asíncrono tras completarse.
- **DataClasses:** Se encuentran los distintos modelos de objetos que se trabajan en la aplicación, hasta ahora solo hemos necesitado crear uno, *Food*, que contiene la información necesaria de cada producto alimenticio.
- **Di:** Se encuentra lo relacionado con la inyección de dependencias, donde hay un archivo, el *AppModule*, en la que se define lo que se va a inyectar en la aplicación usando *Dagger* como herramienta.
- **View:** Se encuentran la lógica de las distintas vistas que contiene la aplicación, se irá detallando en ellas con más profundidad a lo largo del documento.
- **ViewModels:** Se encuentran los *ViewModel* que se han utilizado en la aplicación, en concreto el que almacena la información del usuario, el de la dieta (página principal), el de la comida (escaneo de producto), el de la búsqueda y el de la despensa del usuario. Se entrará en detalle del flujo de ejecución en el siguiente apartado.

En el apartado del diseño, se sigue la estructura de carpetas por defecto añadiendo dos carpetas, menu y navigation, la primera para la definición del menú que se usa en la barra de navegación de la aplicación, y la segunda para almacenar la definición del grafo de navegación, además, en este proyecto se han usado ficheros xml para los distintos layouts.

Flujo de ejecución



Captura 2 - Diagrama aproximado del flujo de ejecución

La aplicación en general tiene un flujo de ejecución como se puede observar en la captura, los fragmentos toman la información y se la envían al ViewModel correspondiente que es el que se encarga de almacenar y tratar los datos de la parte de la aplicación de la que es responsable, además, se ayuda de la implementación de Firebase que le corresponde para obtener cierta información o actualizarla en la base de datos. Los fragmentos comparten información entre sí a través del ViewModel para poder mostrar los datos de manera coherente en la interfaz de usuario. Esto permite que la aplicación tenga una interfaz de usuario consistente. De esta manera, el ViewModel actúa como intermediario entre los fragmentos y los datos que se almacenan en la base de datos, y permite que la aplicación funcione de manera eficiente y mantenga su integridad de datos. Además, hay que tener en cuenta que las pantallas que utilizan tarjetas para mostrar los datos, que son las principales del funcionamiento de la aplicación, utilizan un diseño de arquitectura Modelo-Vista-Presentador, siendo el modelo el o los ViewModel correspondientes, la vista sería la propia vista de la pantalla en cuestión, y el presentador sería el Adapter que se utiliza para representar la lista de tarjetas. Gracias a esta arquitectura, se logra una separación clara de responsabilidades y se mejora la escalabilidad y el rendimiento de la aplicación.

Motivación de usar fragmentos en lugar de actividades

La razón principal de usar fragmentos en lugar de actividades es la mejora del rendimiento y velocidad de la aplicación al no tener que crear y destruir constantemente actividades. Además, esto nos daba la posibilidad en el futuro de adaptar la aplicación a dispositivos más grandes mostrando varios fragmentos en la misma actividad.

Explicación de por qué se ha usado inyección de dependencias y por qué con Dagger

Se ha usado inyección de dependencias para implementar Firebase Firestore y Auth para poder separar la lógica de la aplicación de la lógica de conexión a la base de datos y autenticación de Firebase. Esto permite mantener un código más legible y claro, además de permitir reutilizar estos objetos en diferentes contextos sin necesidad de inicializarlos ni duplicar código.

Además, se ha utilizado Dagger Hilt como herramienta de inyección de dependencias ya que es fácil de configurar y proporciona una gran flexibilidad automatizando la inserción de dependencias en la aplicación, usando la etiqueta *@InstallIn* para especificar que se aplicará el patrón Singleton para la creación de los componentes.

Funcionamiento del MainActivity

Esta es la única actividad de la aplicación, encargada de iniciarla y mostrar la pantalla principal. Cuando la actividad se inicia se llama al método “onCreate()” encargado de inflar la vista, configurar la navegación y comprobar si el usuario actual tiene la sesión iniciada. Si la tiene, se navega hacia la pantalla inicial que muestra la información de la dieta diaria y se muestra le componente de navegación inferior. Si el usuario no tiene la sesión iniciada, se muestra la pantalla de inicio para nuevos usuarios, para posteriormente iniciar sesión o registrarse, y se esconde el componente de navegación inferior.

El método “setupNav()” se encarga de configurar el componente de navegación inferior (BottomNavigationView) y agrega un listener que controla cuando se debe mostrar o esconder el componente de navegación en función del destino de navegación actual.

Funcionamiento pantalla sin usuario logueado

Pantalla de inicio que se muestra cuando el usuario no tiene una sesión iniciada. Cuando se pulsa el botón “Iniciar sesión” se navega hacia la pantalla de inicio de sesión. Lo mismo ocurre cuando se pulsa el botón “Registrarse”, pero en este caso se navega hacia la pantalla de registro.

Funcionamiento inicio de sesión

Esta pantalla permite al usuario iniciar sesión con un correo electrónico y una contraseña. Utiliza una instancia del ViewModel del usuario para realizar la llamada de inicio de sesión y observar el resultado.

Cuando pulsa el botón “Iniciar sesión”, se llama al método “iniciaSesion()” que valida que los campos de correo electrónico y contraseña estén rellenos y luego realiza la llamada de inicio de sesión utilizando el método “login()” del UserViewModel. Si la

llamada es exitosa, se muestra un mensaje de éxito y se navega hacia la pantalla inicial cuando hay una sesión iniciada. Si hay un error, se muestra un mensaje de error y se reinicia el flujo del UserViewModel. Cuando se pulsa en el botón “Registrarse” se navega hacia la pantalla de registro.

Funcionamiento registro de usuario

Esta pantalla permite al usuario registrarse con un nombre, correo electrónico y contraseña. Al igual que en el inicio de sesión, utiliza una instancia del UserViewModel para realizar la llamada de registro y observar el resultado.

Cuando se pulsa en el botón “Registrarse”, se llama al método “registrarse()” que valida que los campos de nombre, correo electrónico y contraseña estén rellenos y que la contraseña y su repetición sean iguales. Luego realiza la llamada de registro utilizando el método “singup()” del UserViewModel. Si la llamada es exitosa, se muestra un mensaje de éxito y se navega hacia la pantalla inicial cuando hay una sesión iniciada. Si hay un error, se muestra un mensaje de error personalizado según el tipo de error y se reinicia el flujo del UserViewModel. Cuando se pulsa en el botón “Iniciar sesión” se navega hacia la pantalla de inicio de sesión.

Funcionamiento del escáner

Para hacer uso de la cámara y escanear códigos de barra, haremos uso de zxing, una librería que nos permite de una forma muy sencilla poder leer una gran cantidad de códigos. Para ello se tuvo que añadir varias dependencias en el build.gradle (Module), así como en el AndroidManifest.xml una línea para sobrescribir la librería que sería la de zxing (tools:overrideLibrary). Cuando se accede a la página del escáner, aparece la cámara habiéndole configurado al lector de códigos, que se pueda leer cualquier tipo de código, que salga una frase en la parte interior que dice “Escanear código de barras” y que suene un pitido cuando se escanee. Si se escanea correctamente un producto, es decir que se encuentra registrado, nos aparece el nombre y la foto del producto escaneado. Así como la opción de añadirlo en la dieta o la despensa. Todos estos elementos que aparecen cambian su visibilidad gracias a las funciones “showProducto()” y “hideProducto()”. En el caso que se elija agregarlo a la dieta (“addToDiet()”) nos saldrán dos ventanas, en la primera debemos de introducir la cantidad de producto a incorporar, pudiendo cancelar con el botón “Cancelar”. En cambio, al apretar el botón “Añadir” nos aparecería la segunda ventana, donde se debe seleccionar en qué comida del día queremos añadir el producto, el desayuno, el almuerzo o la cena. Esta última ventana se crea en la función “setFoodOfDay()”.

Cabe destacar que cuando se escanea el producto se obtiene el código de barras. Nuestros productos deberán tener en nuestra base de datos como id el código de barras, por lo que a la hora de comprobar que ese producto se encuentra en nuestra base de datos debemos de buscarlo con la función “buscarProducto()”. Pasándole el código, comprobamos que haya un producto con ese código como id.

Por otra parte, si se desea añadir a la despensa el producto escaneado, nos aparece la misma primera ventana para introducir el peso del producto. En este caso si se da a "Añadir" aparece directamente la comida en la despensa. No obstante, si la comida escaneada no se encontrase en la base de datos, aparecería un mensaje Log informando sobre ello y desapareciendo cualquier tipo de información de algún producto anterior. En cualquier caso, encontrado o no el producto siempre habrá un botón que ponga "Scan", el cual al apretar en él nos vuelve a abrir el lector de códigos. Asimismo, para establecer que la pantalla del escaneo sea vertical, en el AndroidManifest.xml se añadió un *activity* dentro del *application* en el que se determinó el *screenOrientation* a vertical.

Funcionamiento búsqueda de producto

Cuando el usuario escribe en la barra de búsqueda se llama al método "onQueryTextChange" que filtra la lista de alimentos utilizando el texto que ha escrito el usuario. Además, tiene un observador que monitorea los cambios en la lista de alimentos y actualiza la vista de tarjetas en consecuencia.

La lógica de pasarle la información del producto seleccionado se realiza en el Adapter de la tarjeta correspondiente (CardViewAdapter). El adaptador se utiliza para mostrar la lista de alimentos de la búsqueda, además, tiene una instancia del ViewModel de la búsqueda, que se utiliza para obtener información adicional sobre los alimentos y para navegar a otras pantallas de la aplicación. Cuando se vincula una vista de tarjeta a los datos de un alimento, se establece un listener para la vista de la tarjeta. Si el usuario pulsa en la tarjeta, se muestra un cuadro de diálogo que solicita al usuario la cantidad que desea añadir. Una vez que el usuario introduce la cantidad y hace clic en "Añadir", el adaptador navega a la pantalla correspondiente según el contexto en el que se utilice la función de búsqueda de producto, ya sea para la despensa, o para la dieta que se encuentra en la pantalla principal, actualizando el producto seleccionado en el ViewModel.

Funcionamiento de la pantalla principal

Esta pantalla incluye una lista de alimentos para el desayuno, el almuerzo y la cena, además de información sobre el objetivo de consumo de calorías del usuario y el consumo actual. En la función "onViewCreated" se establecen los listener para los botones y elementos interactivos de la pantalla y se inicializan y configuran los RecyclerViews que se utilizarán para mostrar las listas de alimentos. Además, se establece un observador para la cantidad objetivo de caloría del usuario y se actualiza la pantalla en consecuencia. También se establece un observador para la lista de alimentos del desayuno, almuerzo y cena, que se utilizan para rellenar los RecyclerViews correspondientes con una instancia del adaptador "HomeFoodCardAdapter". Se irá actualizando la información y los elementos de la pantalla de acuerdo con los datos que recibe de los ViewModels. Al entrar en la pantalla se ejecuta el método "analizarEscenario()" que además de establecer los observadores, comprueba si ha

llegado algún producto nuevo que hay que añadir desde la pantalla de búsqueda, si es así, querrá decir que el usuario previamente ha pulsado uno de los botones de añadir, y se procederá a añadir el producto a la lista correspondiente, toda esta información la obtiene del ViewModel de la búsqueda. Como se ha comentado, al pulsar uno de los botones de añadir representados con un "+" en el layout, se navegará al usuario a la pantalla de búsqueda, indicando en la variable correspondiente del ViewModel qué comida ha seleccionado y de donde se solicita la búsqueda, en este caso, de la pantalla principal.

El adaptador "HomeFoodCardAdapter" se utiliza para mostrar una lista de elementos de comida en una vista de tarjeta. Se le pasa una lista de productos alimenticios, una instancia del ViewModel de la dieta y una cadena que representa el tipo de la comida. Se incluye un listener a la tarjeta, que al pulsar sobre ella se muestra un diálogo de alerta que permite al usuario modificar o eliminar el producto. Si el usuario elige modificar, se le pide que ingrese una nueva cantidad de comida en gramos y se actualiza el objeto Food correspondiente. Si el usuario elige eliminar, se elimina el elemento de la lista y de la dieta diaria.

Adicionalmente, en esta pantalla se ha implementado un método que te genera la dieta automáticamente, el método "generarDietaAuto()" llama al método correspondiente del ViewModel de la dieta para generar una dieta según los alimentos que se encuentren en la despensa y los que faltan en cada comida (desayuno, almuerzo y cena). Si falta algún alimento en una comida en particular, se envía esa lista vacía al método y se le dice que solo genere esa comida. Si faltan alimentos en todas las comidas, se envía una lista vacía para que el método genere todas las comidas. Luego, la función actualiza la despensa eliminando los alimentos que se han utilizado para generar la dieta. Por último, se muestra un mensaje de confirmación que se ha generado la dieta.

Cabe destacar que el DietViewModel al inicializarse, además de obtener la información de la base de datos, comprueba que la fecha actual coincida con la marca de tiempo almacenada en la base de datos, de no ser así, se reinicia la información de la dieta diaria, ya que se entiende que es un nuevo día.

Funcionamiento perfil de usuario

Cuando el usuario accede a la pantalla, se muestra su nombre completo, la imagen de perfil y algunos datos adicionales, como su peso y altura actuales y la cantidad de calorías que debe consumir al día. También hay un botón de "editar perfil" que permite al usuario modificar estos datos y un botón de "cerrar sesión" que cierra la sesión del usuario en la aplicación. Además, se define una serie de observadores que se suscriben a las variables de los ViewModels de usuario y dieta diaria, manteniendo la pantalla de perfil actualizada con los datos más recientes cada vez que cambian. Por ejemplo, cuando el usuario modifica su peso en la pantalla de edición de perfil, el observador del peso se activa y actualiza el peso mostrado en la pantalla de perfil.

En la pantalla de edición de usuario se sincronizan los datos del perfil de usuario con los elementos de la interfaz. Tiene dos botones principales: “Guardar” y “Cancelar”. El botón “Guardar” actualiza los datos del perfil del usuario y los almacena en la base de datos, mientras que el botón “Cancelar” cancela la edición y vuelve a la pantalla anterior, es decir, a la del usuario. La pantalla también tiene un botón para cambiar la foto de perfil, que abre un selector de archivos y permite al usuario elegir una foto de la galería del dispositivo. Los métodos “requestPermission(), “requestPermissionLauncher” y “startForActivityGallery” son parte de la lógica de la aplicación para permitir al usuario elegir una foto de su galería y establecerla como foto de perfil. En primer lugar se solicita al usuario permiso para acceder a la galería de imágenes del dispositivo, si el usuario concede el permiso, llama al método “pickPhotoFromGallery()” para abrir la galería y permitir al usuario elegir la foto, si el usuario no concede el permiso se muestra un mensaje en pantalla indicando que se necesita habilitar los permisos. Si el usuario elige una foto, se establece como foto de perfil en el modelo de vista del usuario y se muestra en pantalla. Adicionalmente, la pantalla también tiene un botón para eliminar la foto de perfil, que muestra una ventana de alerta para confirmar la acción.

Funcionamiento despensa del usuario

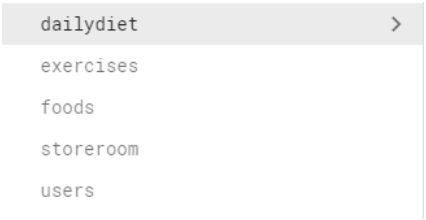
Esta pantalla muestra la despensa de los distintos alimentos del usuario. Al igual que ocurre en la búsqueda de producto, cuando se crea la vista, se configura un listener en la barra de búsqueda para filtrar los alimentos mostrados en la lista a medida que el usuario escribe en la barra de búsqueda.

Cuando se muestra la vista, se llama al método “analizarEscenario()” que se encarga de ver si esa navegación es debido al resultado de la adición de un nuevo producto desde la pantalla de búsqueda de alimento, al igual que en la pantalla inicial donde se muestra la dieta diaria del usuario, si es así, se añade el elemento a la lista. Además, se observa el LiveData que almacena la lista de la despensa del usuario que se encuentra en el ViewModel de la despensa (StoreroomViewModel) y, cuando se actualiza, se muestra en una lista utilizando un adaptador de RecyclerView (StoreroomFoodCardAdapter). Si ocurre un error al cargar los datos, se muestra un mensaje Toast.

El adaptador muestra la lista de alimentos de la despensa del usuario. Se establece un listener en el “+” de la tarjeta, para que cuando el usuario pulse en él se muestre un cuadro de diálogo para permitir modificar o eliminar el alimento de la lista, para ello al adaptador se le pasa una instancia del StoreroomViewModel.

Finalmente, hay un listener en el botón “Añadir alimentos” para navegar hacia la pantalla de búsqueda de alimentos, indicando que la búsqueda proviene de la despensa.

Estructura de los datos en Firebase Firestore



Captura 3 - Colecciones en Firestore

En Firestore se encuentran dos tipos de colecciones:

- Relacionadas con los usuarios: dailydiet, storeroom, users
- Almacenamiento de la información de los elementos que no tienen relación con los usuarios: foods, exercises

Los documentos de las colecciones relacionadas con los usuarios tienen como identificador, el identificador único (uid) del propio usuario, ya que cada documento corresponde a un solo usuario. En dailydiet se establece toda la información relacionada con la dieta diaria del usuario, 3 arrays que corresponden al desayuno, almuerzo y cena donde se almacena la información de cada producto, y un mapa que almacena la fecha de esa dieta con día, mes y año. En storeroom se encuentra un array con la información de cada producto almacenado en la despensa del usuario. Y en users se puede ver información adicional que no es almacenada en Firebase Auth, como es la edad, la altura, el peso, y el objetivo calórico del usuario.

Los identificadores de los documentos de las colecciones no relacionadas con los usuarios son automáticos generados por Firestore, a excepción de algunos documentos de la colección foods. Tanto los documentos de foods como de exercises han sido establecidos manualmente, aunque en el futuro se puede implementar mediante una API para obtener estos datos, o bien automatizar más el proceso. Los identificadores de los documentos de foods, o bien son el código de barras del producto, si es un producto del supermercado, o bien uno generado automáticamente por Firebase si es un producto genérico. Los documentos de la colección foods contienen el nombre, así como la información nutricional (calorías, grasas, proteínas, hidratos de carbono) y la url de la imagen de cada uno de los productos alimenticios de la aplicación. Los documentos de la colección excercises contiene el nombre de la actividad física así como las calorías quemadas a la hora realizando esa actividad física, cabe destacar que es una información meramente orientativa para el usuario, ya que depende de muchos más factores.

Extra

- La aplicación utiliza la biblioteca Glide para cargar la imagen del alimento en la vista de los CircleImageView.
- Alguna de las vistas tiene el método “goBack()” que define a dónde se navegará al pulsar el botón de retroceso o alguna opción de la pantalla que produzca esa navegación. En el caso de la pantalla principal, al retroceder se cerrará la aplicación.

Conclusiones

Como posibles mejoras, se plantearía que no solo se pueda fijar como objetivo las calorías en el perfil sino también las grasas, proteínas, carbohidratos… De forma que en la pantalla principal pudiésemos observar la evolución de los valores, es decir, poder visualizar que cantidades llevamos ingeridas y cuales son nuestros objetivos. A continuación, se adjunta una representación visual de lo comentado anteriormente, que se realizó en Figma.

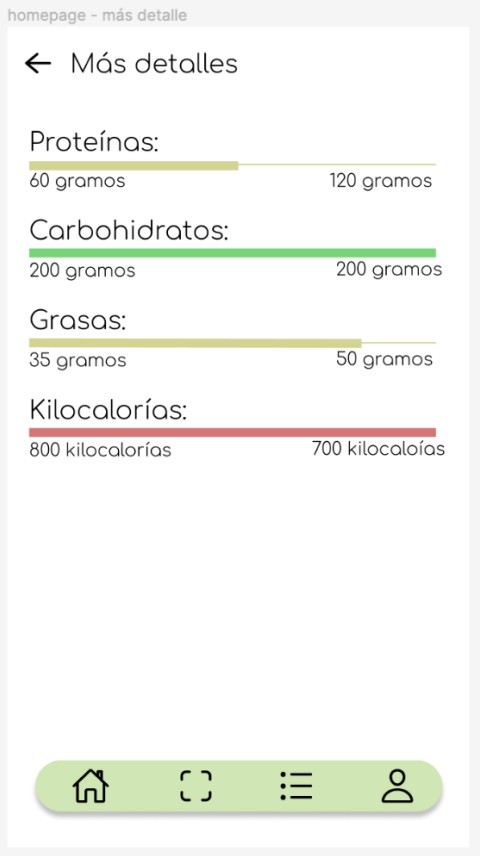


Ilustración 1. Posible mejora de evolución de los datos

Asimismo, otra posibilidad de mejora relacionado a lo comentado sería un calendario en el que obtuviésemos un resumen de la semana con colores, de forma que se pudiese obtener a simple vista si se ha conseguido alcanzar los objetivos, si nos hemos quedado a mitad o si lo hemos hecho muy mal. Para este propósito se almacenaría un registro de las dietas del usuario a lo largo de un tiempo determinado.

Ilustración 2. Posible mejora calendario

Igualmente, en cuanto a la edición del perfil del usuario, se podría dar la oportunidad de que este, al cambiar la foto de perfil, se le permita sacarse una foto en el momento.

Cuando creamos las historias de usuario se plantearon las siguientes ideas: permitir al usuario crear una solicitud de registro de producto en la base de datos, tener una lista de la compra dentro de la aplicación, registro del peso pudiendo observar un gráfico de la evolución del peso. Por otra parte, se planteaba el posible registro de actividades para descontarlas en las calorías ingeridas, dando la posibilidad de conectar la aplicación con Google Fit, para obtener estos datos. Otra de las mejoras adicionales, era poder hacer un prototipo de red social en la que los usuarios pudieran compartir su progreso, ver las dietas que han seguido otros usuarios…

Por último, cabe destacar la modificación objetivos nutricionales automatizada según propósito, es decir, brindar el servicio de que al usuario se le modifique objetivos nutricionales automáticamente según su propósito ya sea pérdida de peso, definición muscular, aumento de peso, ganancia muscular, mantenimiento...

No obstante, todo lo comentado han sido ideas que nos surgieron al definir la idea del proyecto, pero debido al corto tiempo en disposición tuvimos que descartar dichas ideas o simplificarlas o aplazarlas inclusive, centrándonos en la función principal de nuestra aplicación.

Finalmente, toda la documentación utilizada para la realización del proyecto se encuentra en el README del GitHub.