

ñoApi5.327Diseño de la APIsection.5.3

Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería de Software en la Práctica Obligatorio

Eusebio Durán 202741

Nicolás Eirin 200111

Ana Clara Rodríguez 210472

Entregado como requisito de la materia Ingeniería de
Software en la Práctica

18 de junio de 2019

Declaraciones de autoría

Nosotros, Eusebio Durán, Nicolás Eirin, Ana Clara Rodríguez, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el segundo obligatorio de la materia Ingeniería de Software en la Práctica;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Índice general

I	Obligatorio 1	5
1.	Introducción	6
1.1.	Descripción de la aplicación	6
2.	Forma de Trabajo	7
2.1.	Manejo de versionado	7
3.	Planificación	8
3.1.	Release Plan	8
3.2.	Eventos	10
3.2.1.	Daily Meetings	10
3.2.2.	Sprint Review	10
3.2.3.	Retrospectivas de cada Iteración	10
3.3.	Sprint Planning	10
3.4.	Definition of Ready	11
3.5.	Definition of Done	11
3.6.	Gestión de riesgos	12
4.	Historias de Usuario	15
4.1.	Estimación	15
4.1.1.	Priorización	15
5.	Descripción de diseño	21
5.1.	Back End	21
5.1.1.	Mantenibilidad	21
5.1.2.	Eficiencia	22
5.1.3.	Simplicidad	22
5.1.4.	Diagramas de paquetes	23
5.1.5.	Diagramas de componentes	24
5.1.6.	Diagramas de despliegue	26
5.2.	Front-End	26
5.2.1.	Comunicación con Back-End	26
5.2.2.	Performance	27
5.3.	Diseño de la API	27
5.3.1.	Usuario	27

5.3.2. Compra	29
5.3.3. Grupo	30
6. Prototipos	31
7. Aseguramiento de la Calidad	40
7.1. Calidad el Código	40
7.1.1. Evidencia de Clean Code	40
7.2. Testing	43
7.2.1. Pruebas Unitarias	43
7.2.2. Pruebas de Integración	44
7.2.3. Pruebas de Front-End	44
8. Anexo	45
8.1. Tablero Principal	45
8.2. Tablero de Estimación	45
8.3. Tablero de Bugs	45
II Obligatorio 2	47
9. Introducción	48
10. Manejo de versionado	49
11. Planificación	50
12. Descripción de diseño	53
12.1. Calidad del Diseño	54
12.1.1. Front-End	54
12.2. Modelo de Tablas	55
12.3. Diseño UI UX	55
12.3.1. Integración con Sensores	55
12.3.2. Descripción general	56
12.3.3. Barra de Navegación Superior	57
12.3.4. Manejo del Stack	59
12.3.5. RecyclerViews	59
12.3.6. Carga de Imágenes	60
12.3.7. Validación de los Formularios	60
12.3.8. Mensajes de Carga	61
12.3.9. Configuraciones Espontáneas	63
12.4. Diseño de la API	64
12.4.1. Diseño de la API real	65
12.5. Eficiencia	67
13. Aseguramiento de la calidad	68
13.0.1. Evidencia de Clean Code	69
13.0.2. Testing	69

14. Instructivo de instalación	72
14.1. Deploy del Back End	72
14.1.1. Requisitos	72
14.1.2. Instrucciones	72
14.2. Deploy del Front End	73
14.2.1. Deploy con Android Studio	73

Parte I

Obligatorio 1

1. Introducción

Este documento contiene las especificaciones básicas, forma de trabajo, requerimientos y características técnicas que tendrá la aplicación a desarrollar para la segunda instancia de evaluación obligatoria de la materia Ingeniería de Software en la Práctica.

1.1. Descripción de la aplicación

La aplicación a desarrollar llevará por nombre QuickSplit y tendrá por objetivo resolver un problema muy común a la hora de hacer compras o efectuar gastos compartidos entre amigos. La aplicación QuickSplit permitirá que dado un grupo de personas puedan dividirse de forma transparente los gastos de compras entre los miembros pertenecientes a las mismas.

Un usuario tiene amigos que previamente deben haber aceptado su solicitud de amistad, por otro lado puede tener una o más sublistas de amigos para facilitar la operación al agregar una compra, ya que por lo general se efectúa más de una compra para los mismos integrantes.

Un amigo agrega a los demás en la lista de compra. Se ingresan los montos de la compra, pudiendo hacerlo escaneando el código QR del ticket si se está dentro de Uruguay o manualmente. Adicionalmente se puede agregar una foto de los productos y la ubicación de dónde fueron comprados de forma de que los usuarios puedan recordar dónde se hizo la compra. Una vez finalizada la compra, la aplicación hará un balance de cuánto debe pagar cada persona, pudiendo excluir un gasto si la compra está desglosada en artículos. Por ejemplo para lista "Vacaciones en La Pedrera", Juan no toma cerveza, por lo tanto en el balance no se agrega el monto de su parte de la cerveza y se divide entre los demás amigos. A cada miembro de la compra se le enviará una notificación cada vez que exista una novedad asociada a alguna compra en la que participen, algún subgrupo de amigos en el que participen o cuando sean agregados por otra persona.

2. Forma de Trabajo

El sistema será desarrollado en los lenguajes de programación C# (bajo el framework .NET Core) y Java (bajo el framework Android SDK). Para poder persistir los datos de la aplicación se utilizará una base de datos relacional gestionada mediante SQLServer.

La modalidad de trabajo será basada en algunos aspectos brindados por la metodologías tradicionales así como también aspectos brindados por las metodologías ágiles (SCRUM y el tablero Kanban).

Para un mejor desempeño a la hora de trabajar, hemos asignado los siguientes roles:

- Scrum Master: Nicolás Eirin.
- Product Owner: Eusebio Durán.
- Development Team: Nicolás Eirin, Eusebio Durán, Ana Clara Rodríguez.

Se ha utilizado la herramienta de administración de proyectos Trello. La misma nos permite manipular las historias de usuario no solo de manera sencilla sino que también digitalmente, omitiendo la necesidad de que los miembros del Development Team deban reunirse físicamente para poder manejar las mismas.

2.1. Manejo de versionado

Se utiliza un repositorio Git, alojado en el servicio Bitbucket. El mismo nos proporciona un completo seguimiento de nuestro repositorio. Dicho repositorio no sólo nos facilita el manejo del versionado del proyecto sino que también un respaldo confiable del mismo.

El flujo de trabajo se basa en la creación de una rama distinta (feature) para cada historia de usuario, utilizando la misma para la implementación de todas las capas del sistema, es decir, desde la conexión a base de datos, pasando por la lógica de la funcionalidad, hasta el front-end de la aplicación Android con la que interactuará el usuario.

Luego de que dicho código es considerado como terminado (cumpliendo con la Definition of Done, ver 3.5) cada rama es integrada a la rama develop.

Al finalizar cada release marcado en el Release Plan (ver 3.1) se integra la rama develop a master.

De ser necesario, se utilizan hotfix para arreglar aquellos bugs e inconcistencias que puedan surgir a lo largo del desarrollo de la aplicación.

3. Planificación

3.1. Release Plan

Para poder definir el Release Plan se han tenido en cuenta los siguientes datos:

- El producto será liberado el: 18/06/2019
- Las iteraciones tienen una duración de 1 semana.

Al final cada release dejaremos disponibles todos los binarios del Front-End y el Back-End, el APK de la aplicación de Android y haremos un deploy del Back-End en los servidores de Amazon Web Services. De esta manera luego de cada release la aplicación estará disponible para el uso de cualquier usuario interesado. Todos los links e información para utilizar el sistema estarán detallados en el readme de la rama master luego de cada release.

Teniendo en cuenta la fecha de liberación y la duración de cada iteración, se definen 3 releases compuestos por 3 iteraciones cada uno como se indica en las siguientes tablas:

Release 1	
Iteración 1	15/04 - 21/04
Iteración 2	22/04 - 28/04
Iteración 3	29/04 - 05/05

Release 2	
Iteración 1	06/05 - 12/05
Iteración 2	13/05 - 19/05
Iteración 3	20/05 - 26/05

Release 3	
Iteración 1	27/05 - 02/06
Iteración 2	03/06 - 09/06
Iteración 3	10/06 - 16/06

Las historias de usuario han sido estimadas en Talles de Camisetas como se menciona en la sección Historias de Usuario (ver 4), sin embargo, para poder tener un mejor manejo de las mismas se ha utilizado el siguiente cuadro de equivalencia:

Talle de camiseta	Story points
S	1
M	2
L	4
XL	6

Dado que el equipo no tiene experiencia trabajando en conjunto se ha realizado una prueba de concepto para poder estimar la cantidad de historias de usuarios a realizar por iteración obteniéndose una estimación de 6 Story points por iteración.

A continuación se encuentran las historias de usuario que fueron seleccionadas según su prioridad para cada iteración dentro de cada release:

■ **Release 1**

● Iteración 1:

1. Registro de Usuario
2. Login
3. Modificar Usuario
4. Eliminar Usuario
5. Agregar Amigos
6. Creación de Grupo

● Iteración 2:

1. Listar Amigos
2. Logout
3. Abandonar Grupo
4. Registrar Compra Manual

● Iteración 3:

1. Eliminar Grupo
2. Eliminar Compras Subgrupo
3. Dividir Gastos

■ **Release 2**

● Iteración 1:

1. Compras para Subgrupo
2. Modificar Compra
3. Reconocimiento de QR de tickets (Spike)

4. Mapa de Compras por GPS (Spike)

- Iteración 2:

1. Reconocimiento de QR de tickets

- Iteración 3:

1. Mapa de Compras por GPS

- **Release 3**

- Iteración 1:

- Desglose de Compras mediante OCR (Spike)

- Login con Google (Spike)

- Iteración 2:

- Desglose de Compras mediante OCR

- Iteración 3:

1. Implementación de Login con Google

3.2. Eventos

3.2.1. Daily Meetings

Todos los días nos juntaremos en la facultad entre o luego de las clases en un timebox de 15 minutos. En esta reunión nos preguntaremos que hicimos ayer, que vamos a hacer hoy y discutiremos problemas que tuvimos.

3.2.2. Sprint Review

Luego de cada iteración nos reuniremos a validar todas las historias de usuarios completadas durante la iteración. Ya que el Product Owner es uno de nosotros, validaremos entre el grupo todas las historias de usuario y verificaremos que aporten valor.

3.2.3. Retrospectivas de cada Iteración

Al finalizar cada iteración luego de la Sprint review nos reuniremos a hacer una retrospectiva de la iteración. No hablaremos de aspectos técnicos sino que discutiremos que aspectos de iteración anterior fueron buenos, que aspectos se podrían mejorar y que haremos diferente en la próxima iteración.

3.3. Sprint Planning

Antes de empezar cada iteración nos reuniremos para determinar que elementos de backlog vamos a incluir en la próxima iteración y además realizaremos una revisión de los elementos del backlog para ver si las prioridades y estimaciones siguen siendo vigentes. En caso de que no lo sean deberán ser actualizadas.

3.4. Definition of Ready

Se ha definido el siguiente criterio para poder identificar si una historia de usuario se encuentra lista para poder ser colocada en el backlog:

- Estimación en talles de camisetas
- Descripción
- Tareas (Back-End y Front-End de ser necesario)
- Criterios de aceptación
- Cumple con los criterios definidos por el acrónimo INVEST
- Validación grupal

La validación grupal refiere a someter la historia de usuario a una puesta en común para corroborar que todos los miembros del equipo concuerdan con las tareas y criterios de aceptación asignados a las mismas.

3.5. Definition of Done

Para cada historia de usuario hemos definido los siguientes criterios:

- Criterios de aceptación cumplidos
- Se pasan todas las pruebas unitarias
- Hay cobertura de más del 80 %
- Code review
- Se realizaron pruebas de integración
- Prueba de performance

Para poder realizar el Code review hemos tenido en cuenta la siguiente checklist:

-
- ☐ El código es legible.
 - ☐ No se presentan comentarios innecesarios.
 - ☐ No se necesita utilizar el scroll para leer el código.
 - ☐ Al leer el código la responsabilidad del mismo es evidente.
-

Por otro lado, la prueba de performance consiste en ejecutar el código correspondiente a la historia de usuario bajo las siguientes condiciones:

- Versión de Android Marshmallow 6.0.1 en adelante.
- Servidor alojado en Amazon Web Services en los servidores de Brasil.
- El servidor sera una alojado T2.micro.
- Samsung Galaxy A5 2017.

Bajo estas condiciones el servidor debe poder procesar 50 solicitudes por segundo con un tiempo de respuesta menor a 100ms para todas las solicitudes. No se tiene que tomar en cuenta la latencia que surge de la distancia entre el cliente y servidor.

Con respecto al Front-End, ninguna acción debe tardar mas de 1 segundo en completarse.

3.6. Gestión de riesgos

Los riesgos se encuentran clasificados con un identificador único y descritos con los siguientes atributos:

- **Identificador:** <número único de riesgo>
- **Especificación** <detalle de que se trata el riesgo>
- **Clasificación:** <debe tomar uno de los valores definidos en la *tabla 3.1*>
- **Probabilidad de ocurrencia** <*debe tomar uno de los siguientes valores:*
 - Poco probable
 - Probable
 - Muy probable
 - Altamente probable
 - Un problema mayor>
- **Ocurrencia en el tiempo:** <*debe tomar uno de los siguientes valores:*
 - Inmediato
 - Mediano plazo
 - Largo plazo>
- **Plan de contingencia:** <*descripción del plan*>

Escala	Costo	Tiempo	Calidad	Alcance
Muy alto	Aumento mayor a 50%	Aumento de tiempo de más de 20%	Inservible	Inservible
Alto	Aumento entre 30% y menos de 50%	Aumento de tiempo entre 10% y 20%	Reducción de calidad inaceptable	Alcance inaceptable
Medio	Aumento entre 15% y 30%	Aumento de tiempo entre 5% y 10%	Reducción de calidad requiere aprobación	Áreas principales de alcance afectadas
Bajo	Aumento de menos de 15%	Aumento de tiempo menor a 5%	Solo afectadas aplicaciones muy exigentes	Áreas secundarias de alcance afectadas
Insignificante	Insignificante	Insignificante	Insignificante	Insignificante

Figura 3.1: Escala de impactos

A continuación se detalla para cada iteración los riesgos identificados para las mismas.

Release 1

- No existen riesgos identificados para el primer release.

Release 2

- Iteración 1:
 - **Identificador:** R02.01.03 - Reconocimiento de QR de tickets (Spike)
 - **Especificación:** No se encuentra suficiente información para realizar la historia.
 - **Clasificación:** Medio
 - **Probabilidad de ocurrencia:** Poco Probable
 - **Ocurrencia en el tiempo:** Inmediato
 - **Plan de contingencia:** Contactar a un experto en el tema, suscribirse a un curso de pago.
 - **Identificador:** R02.01.04 - Mapa de Compras por GPS (Spike)
 - **Especificación:** No se encuentra suficiente información para realizar la historia.
 - **Clasificación:** Medio
 - **Probabilidad de ocurrencia:** Poco Probable
 - **Ocurrencia en el tiempo:** Inmediato

- **Plan de contingencia:** Contactar a un experto en el tema, suscribirse a un curso de pago.

Release 3

■ Iteración 1:

- **Identificador:** R03.01.01 - Desglose de Compras mediante OCR (Spike)
- **Especificación:** La tecnología de reconocimiento está licenciada y debe ser comprada.
- **Clasificación:** Alto
- **Probabilidad de ocurrencia:** Muy Probable
- **Ocurrencia en el tiempo:** Inmediato
- **Plan de contingencia:** Buscar alternativas o dejar fuera del alcance esta historia.

- **Identificador:** R03.01.02 - Login con Google (Spike)
- **Especificación:** Se dificulta la integración del login de google, con el login tradicional que se implementará en la aplicación, pudiendo ocurrir problemas de seguridad.
- **Clasificación:** Medio
- **Probabilidad de ocurrencia:** Probable
- **Ocurrencia en el tiempo:** Inmediato
- **Plan de contingencia:** No implementar login con Google. Intentar con otros proveedores de identidad como Facebook o Microsoft.

4. Historias de Usuario

Para poder representar los requerimientos del sistema se han utilizado historias de usuario. Las mismas se han creado siguiendo el modelo Card-Conversation-Confirmation.

4.1. Estimación

Las historias de usuario han sido estimadas mediante los métodos Planning Poker (utilizando talles de camisetas) y triangulación. Los talles utilizados fueron: S, M, L y XL.

4.1.1. Priorización

Ordenamos las historias de usuario según un índice de prioridad. Una historia con prioridad i solamente puede ser empezada a desarrollar una vez que todas las historias con prioridad $i - 1$ hayan sido completadas.

■ Registro de Usuario

- Prioridad: 1
- Tamaño: M
- Descripción:
Como un usuario no registrado
Quiero registrarme en el sistema
Para poder acceder al sistema con mis datos
- Criterios de aceptación:
 1. El usuario debe tener nombre y contraseña
 2. El usuario puede tener foto de perfil
 3. Los nombres de usuarios no se pueden repetir
 4. Se debe confirmar la contraseña en el registro

■ Eliminar Usuario

- Prioridad: 2
- Tamaño: M

- Descripción:
Como un usuario registrado
Quiero poder eliminar mi cuenta
Para darme de baja de la aplicación
- Criterios de aceptación:
 1. Los datos del usuario se borran permanentemente
 2. Otro usuario con el mismo mail puede registrarse
 3. No se puede eliminar un usuario si hay gastos pendientes para dividir
- Agregar Amigos
 - Prioridad: 2
 - Tamaño: M
 - Descripción:
Como un usuario logueado
Quiero poder agregar mis amigos
Para poder agregarlos a grupos fácilmente
 - Criterios de aceptación:
 1. Debo disponer de un buscador de usuarios registrados para agregarlos como amigos
 2. Cuando agrego a alguien esta persona puede aceptar o denegar la solicitud
 3. Debo ser notificado cuando el usuario que invité acepta mi solicitud
- Login
 - Prioridad: 2
 - Tamaño: M
 - Descripción:
Como un usuario registrado
Quiero poder iniciar sesión
Para utilizar el sistema
 - Criterios de aceptación:
 1. Si se ingresa el mail y/o contraseña mal, se notifica al usuario
 2. La contraseña debe estar hasheada en la base de datos
 3. El login debe guardarse, para no tener que loguearse cada vez que se entra a la aplicación
- Modificar Usuario
 - Prioridad: 2
 - Tamaño: S

- Descripción:
Como un usuario registrado
Quiero poder modificar mis datos
Para actualizarlos
- Criterios de aceptación:
 1. El usuario puede modificar su nombre, apellido, mail y contraseña
 2. Solo lo puede modificar el mismo usuario
- Crear un Grupo
 - Prioridad: 3
 - Tamaño: S
 - Descripción:
Como un usuario logueado
Quiero poder crear un grupo de usuarios
Para realizar compras y dividir los gastos
 - Criterios de aceptación:
 1. El grupo debe tener un nombre
 2. El grupo debe tener un administrador, que por defecto es el creador del grupo
 3. El administrador es intercambiable
 4. Puede existir más de un administrador
- Listar Amigos
 - Prioridad: 3
 - Tamaño: S
 - Descripción:
Como un usuario logueado
Quiero poder ver mis amigos
Para ver que amigos tengo en la app
 - Criterios de aceptación:
 1. La lista debe mostrar el nombre de cada amigo y los grupos en común que tienen
- Logout
 - Prioridad: 3
 - Tamaño: S
 - Descripción:
Como un usuario logueado
Quiero poder cerrar sesión
Para salir del sistema

- Criterios de aceptación:
 1. La sesión se cierra
 2. Al hacer logout aparecen los campos para loguearse como otro usuario
- Abandonar Grupo
 - Prioridad: 4
 - Tamaño: M
 - Descripción:

Como miembro de un grupo

Quiero poder abandonar el mismo

Para no compartir mas gastos con el mismo
 - Criterios de aceptación:
 1. El usuario ya no está incluido en el balance de los montos del grupo
- Eliminar Grupo
 - Prioridad: 5
 - Tamaño: S
 - Descripción:

Como usuario administrador de un grupo

Quiero poder eliminar el grupo

Para que éste ya no exista
 - Criterios de aceptación:
 1. Solo el administrador del grupo puede eliminarlo
- Registrar Compra Manual
 - Prioridad: 5
 - Tamaño: M
 - Descripción:

Como usuario dentro de un grupo

Quiero registrar una compra

Para dividir los gastos
 - Criterios de aceptación:
 1. La compra contiene nombre y monto
 2. No se puede ingresar monto negativo
 3. Se puede ingresar monto en Peso Uruguayo, Peso Argentino, Euro, Dolar o Reales
- Reconocimiento de QR de Tickets
 - Prioridad: 5

- Tamaño: M
- Tipo: Spike
- Mapa de Compras por GPS
 - Prioridad: 5
 - Tamaño: M
 - Tipo: Spike
- Compras para un Subgrupo
 - Prioridad: 6
 - Tamaño: S
 - Descripción:

Como usuario dentro de un grupo
Quiero elegir los usuarios del grupo que participan en la compra
Para no tener que incluir a todos en todas las compras
 - Criterios de aceptación:
 1. Por defecto deben estar seleccionados todos los participantes del grupo
 2. Se puede desmarcar uno o más usuarios del grupo para que no participen en la compra
 3. Se debe poder desmarcar la propia persona que realizó la compra
- Dividir Gastos
 - Prioridad: 6
 - Tamaño: L
 - Descripción:

Como usuario dentro de un grupo
Quiero hacer la división de gastos
Para saber debe cada miembro del grupo
 - Criterios de aceptación:
 1. El balance calculado debe ser el correcto para las personas que se está calculando
 2. El administrador puede saldar las compras
- Mapa de Compras por GPS
 - Prioridad: 6
 - Tamaño: XL
 - Descripción:

Como un usuario perteneciente a un grupo
Quiero ver en un mapa donde se realizaron las compras
Para saber donde se realizaron las compras

- Criterios de aceptación:
 1. Visualizar un mapa con puntos donde se realizaron las compras
- Modificar Compra
 - Prioridad: 6
 - Tamaño: S
 - Descripción:

Como un usuario que realizó la compra
Quiero poder modificar la compra
Para cambiar el monto y/o el nombre
 - Criterios de aceptación:
 1. Es posible modificar el nombre de la compra
 2. Es posible modificar el monto de la compra
 3. Solo el usuario que realizó la compra puede modificarlo
- Reconocimiento de QR de Tickets
 - Prioridad: 6
 - Tamaño: S
 - Descripción:

Como usuario dentro de un grupo
Quiero escanear códigos QR de tickets
Para poder ingresar compras mas rápidamente
 - Criterios de aceptación:
 1. No es obligatorio usar QR para realizar la compra
 2. Si no se consiguen los datos mediante QR se le debe avisar al usuario que lo tiene que agregar manual
- Compras mediante OCR
 - Prioridad: 6
 - Tamaño: L
 - Descripción: Investigación
 - Tipo: Spike
- Compras mediante OCR
 - Prioridad: 7
 - Tamaño: XL
 - Descripción:

Como usuario dentro de un grupo
Quiero escanear un ticket
Para obtener el desglose de una compra
 - Criterios de aceptación:
 1. El reconocimiento mediante OCR de la compra falla como máximo un 30 % de las veces

5. Descripción de diseño

5.1. Back End

El Back-End consistirá en una API REST programada en C# utilizando el framework ASP.NET Core y Apache como servidor HTTP. Utilizamos .NET Core en vez de .NET Framework ya que .NET Core es multiplataforma y puede ser corrido en servidores Linux que son mejores y más baratos.

El diseño de el Back-End tiene el objetivos de maximizar los siguientes atributos de calidad: la **mantenibilidad**, **eficiencia** y **simplicidad**.

5.1.1. Mantenibilidad

Con el objetivo de maximizar la mantenibilidad nos basaremos en la arquitectura "Clean Architecture" que establece los siguientes principios básicos:

- **Independencia de la UI:** La arquitectura no debe depender de la interfaz de usuario (en nuestro caso una Web API). Es decir se debería poder cambiar la UI si impactar al resto del sistema.
- **Independencia de frameworks:** La arquitectura no debe depender de los frameworks utilizados.
- **Independencia de la base de datos:** La arquitectura no debe depender de que base de datos utiliza. Se debería poder cambiar por ejemplo de MySQL a MSSQL sin impactar al resto del sistema.
- **Independencia de factores externos:** La arquitectura no debe depender de factores externos como librerías, APIs externas etc.
- **Testeable:** Se tiene que poder probar el sistema sin tener que utilizar UI, base de datos, servidores web o ningún otro tipo de elementos externos.

Otro aspecto que mejora la mantenibilidad es el patrón CQRS implementado usando el patrón Event sourcing. CQRS en general es utilizado para mejorar la eficiencia pero nuestra principal motivación para usarlo es en realidad la mantenibilidad ya que siguiendo este patrón utilizando una librería de Event sourcing, que explicaremos en más detalle en la siguiente sección (ver 5.1.2), resulta en un diseño sumamente desacoplado. Esto se podrá ver en la capa de aplicación ya que esta capa consistirá en una serie de servicios (commands o queries) que serán totalmente

independientes entre ellas y por lo tanto tendrán solo una responsabilidad lo que cumple con el Principio de responsabilidad única.

Lectura de tickets

Nuestra aplicación necesitará de algún algoritmo que extraiga la información de un ticket al realizar una compra. Este algoritmo es muy probable que cambie ya que es probable que en el futuro DGI cambie el formato de la pagina vinculada al código QR de las boletas o puede ser que en un futuro soportemos diferentes tipos de boletas o de otros países.

Para minimizar el impacto de estos posibles cambios utilizaremos el patrón Strategy. Esto se puede ver en la interfaz de usuario ITicketReader. De esta manera nuestro core del sistema (las capas de aplicación y dominio) no dependerán de la implementación específica de los lectores de tickets. Aquí se estaría usando el Dependency Injection Principle y el Open Close Principle ya que se puede extender la cantidad y tipo de tickets que son soportados por el sistema fácilmente y sin impactar al resto del sistema.

5.1.2. Eficiencia

Con el objetivo de mejorar la eficiencia tomamos dos decisiones de diseño importantes. Utilizar un modelo no bloqueante para los pedidos y utilizar el patron CQRS.

El patrón CQRS consiste en representar todos los pedidos al sistema como un command o un query. Los commands son funciones que implican la escritura de datos mientras que los queries sólo lectura. Separando la lectura de la escritura resulta en mejor eficiencia. Aunque este patrón suele ser implementado utilizando varias bases de datos o distintos modelos dentro de la base de datos para poder escalar los modelos de lectura y escritura nosotros decidimos sólo tener un modelo de base de datos y una sola base de datos ya que no tenemos necesidad de escalar en manera independiente la lectura y la escritura. La eficiencia en nuestro caso viene simplemente de no mezclar lectura y escritura en una sola conexión a la base de datos.

Utilizar un modelo no bloqueante de hilos en los pedidos tiene el beneficio de que nuestros hilos que se encargan de responder los pedidos de la Web API no se bloqueen cuando estén esperando que se complete una operación de la base de datos, escritura a disco o cualquier otro tipo de operación asíncronica. Ya que las operaciones asíncronicas suelen llevar un tiempo considerable, tener un sistema en el cual los hilos puedan atender otros pedidos mientras que se espera a que termine la operación asíncronica resulta en un sistema que puede atender a muchos más pedidos por segundo.

5.1.3. Simplicidad

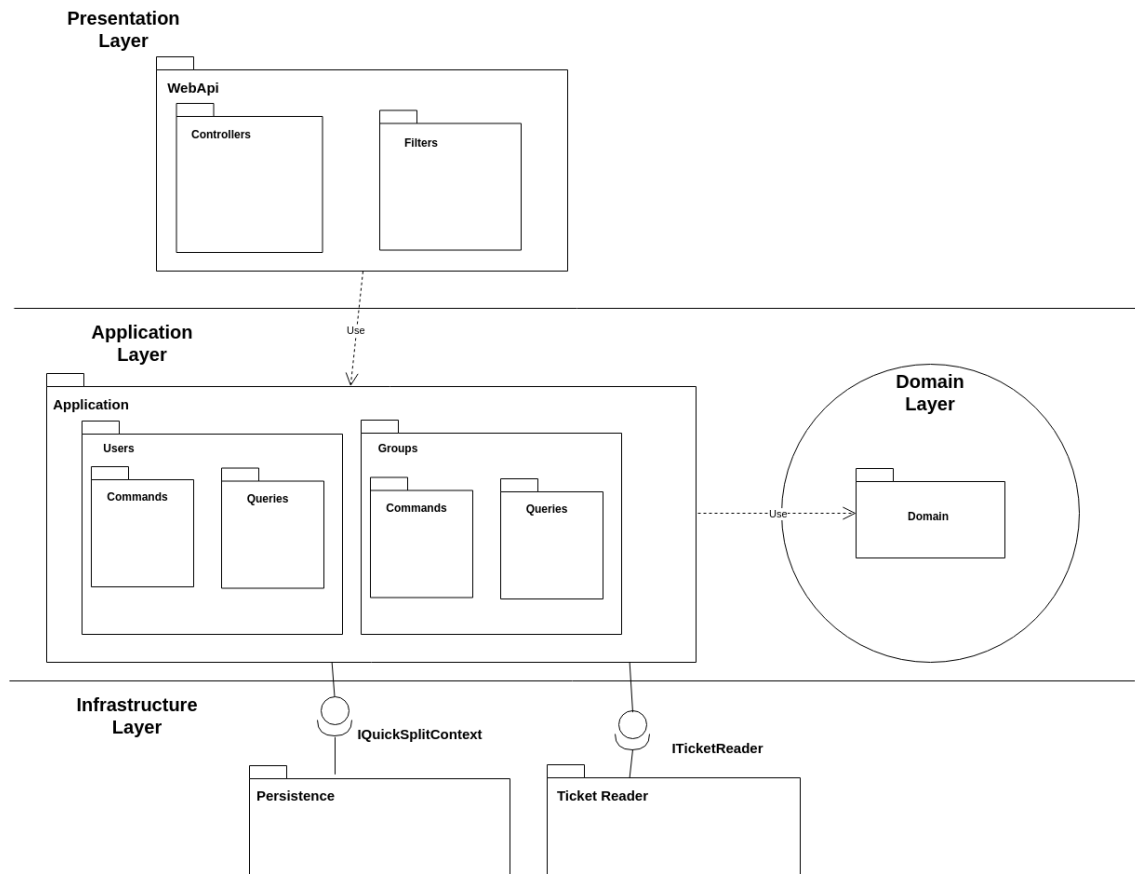
El patrón CQRS implementado con la librería MediatR también favorece la simplicidad del sistema ya que componer la capa de aplicación solamente de commands y queries resulta en un sistema mucho más simple.

Aunque depender de EntityFramework y MediatR (librería de event sourcing) no cumple con el principio de no depender de frameworks de clean architecture, abstraernos de ellos implicaría hacer el sistema mucho más complejo y además es muy poco probable que decidamos cambiarlos en el futuro. Por esto no consideramos que vale la pena abstraernos de estas dos librerías.

5.1.4. Diagramas de paquetes

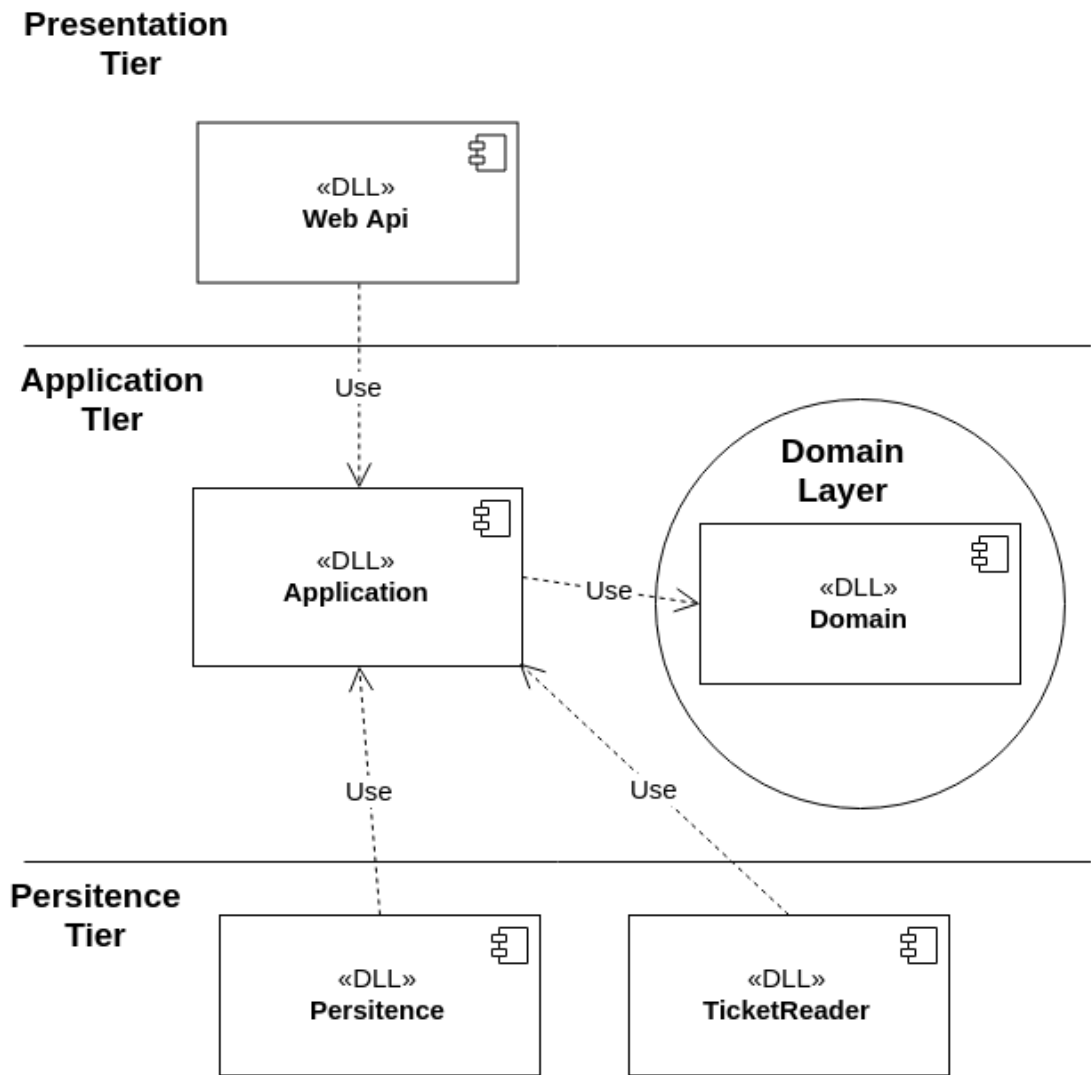
Como se puede ver en el siguiente diagrama UML dividimos nuestro sistema en las siguientes capas aplicando el patrón de diseño de Layers en orden de nivel de su funcionalidad descendiente. Las capas de más alto nivel (las localizadas en el medio del diagrama) no dependen de las de menor nivel.

- **Domain layer:** Contiene toda la lógica y reglas de negocio, se debe encontrar totalmente aislada del resto del sistema. Utiliza puntos de acoplamiento para facilitar el cambio.
- **Application layer:** Expone una serie de funciones que puede realizar el sistema y las realiza coordinando entre el dominio, el acceso a datos y posiblemente otros elementos externos.
- **Infrastructure layer:** Esta capa se encarga de proveer acceso a recursos externos como el acceso a datos o sistema de auditoría al Core del sistema (capas de aplicación y dominio).
- **Presentation Layer:** Contiene toda la lógica de interacción con el usuario.



5.1.5. Diagramas de componentes

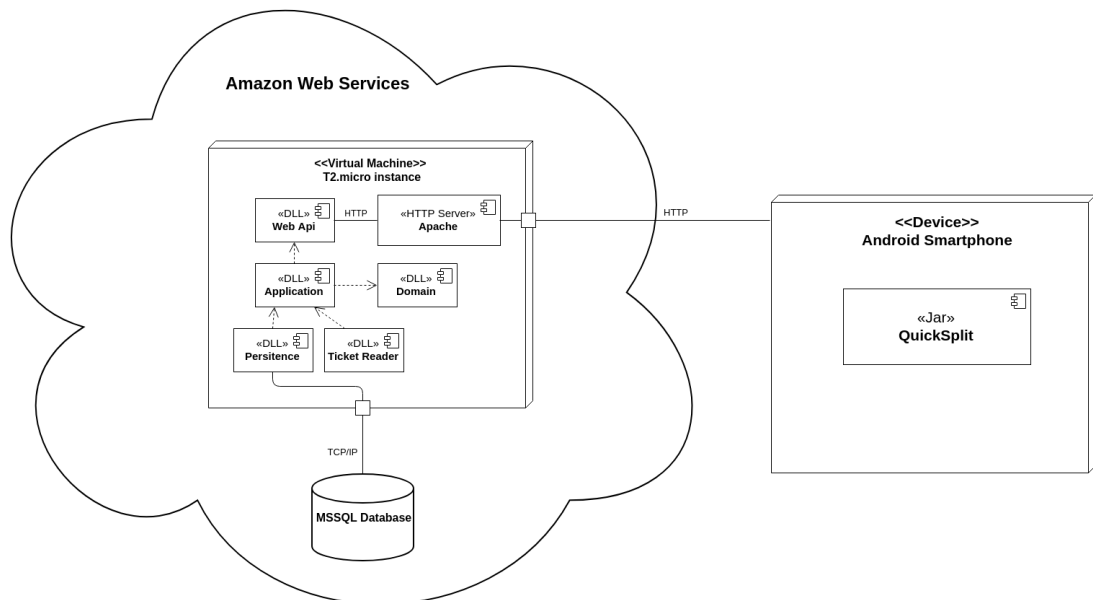
Siguiendo el mismo razonamiento de la sección anterior se dividieron los componentes en tiers aplicando el patrón Multi-tiers de manera de que los cambios de los componentes de bajo nivel no afecten a los de alto nivel.



5.1.6. Diagramas de despliegue

Desplegaremos el Back-End de nuestra aplicación en la nube utilizando Amazon Web Services ya que tenemos experiencia en esta plataforma y tenemos el servicio gratis por un año. Utilizaremos una maquina virtual Linux que correrá nuestro Back-End y utilizará el servidor HTTP Apache como un reverse proxy para comunicarse con el exterior. La base de datos estará hosteada afuera de la maquina virtual en un servidor SQL de Amazon. Para el despliegue utilizamos un container docker para manejar las dependencias al sistema. Además utilizamos un sistema de despliegue continuo que automáticamente hace el deploy del código cada vez que se hace un push en la rama master.

El Front-End de la aplicación sera una jar que sera ejecutado por cualquier smartphone Android 6.0.1 o superior.



5.2. Front-End

El Front-End consistirá en una aplicación desarrollada para Android 6.0.1 Marshmallow o superior, que se puede ejecutar en tablets o en celulares que posean dicho sistema operativo. Para el desarrollo se utilizará el IDE oficial de Google Android Studio y los repositorios oficiales de Gradle. El Front-End estará programado en Java. Para los complementos visuales se utilizarán los componentes de la biblioteca gráfica Material design. La aplicación se comunicará mediante REST con el Back-End utilizando retrofit "A type-safe REST client for Android and Java".

5.2.1. Comunicación con Back-End

Para la comunicación con el Back-End manejamos dos opciones GraphQL y REST. Dado nuestro dominio de tecnologías y el tiempo acotado con el que contamos

para aprender nuevas tecnologías decidimos utilizar REST. Esto nos permite generar completa independencia entre las tecnologías de Back-End y Front-End.

5.2.2. Performance

La UI se diseñará con fragments y activities. Para generar un código más mantenible y de mejor calidad se utilizarán fragments. Estos nos permitirán reutilizar el código y también bajar el nivel acoplamiento del mismo. Por otra parte, para mejorar la velocidad de respuesta de la aplicación se utilizarán activities en aquellas partes de la UI que tengan componentes que no puedan ser reutilizables.

Los llamados hacia el Back-End se realizan de manera asincrónica, de tal forma de que no se bloquee la interfaz de usuario mientras se espera por la respuesta de datos.

5.3. Diseño de la API

Nuestra Web API cumplirá con el estándar REST de manera de proveer una interfaz estable, clara y simple de usar para ser consumida por el Front-End.

Entonces nuestra API cumplirá con los siguientes principios:

- Cada mensaje HTTP contiene toda la información necesaria para comprender la petición.
- Se utilizara un conjunto de operaciones bien definidas (GET, POST, PUT, DELETE) que se aplican a todos los recursos de información.
- Habrán como máximo 2 capas de recursos para mantener el sistema simple.
- Cada recurso tendrá una id única.

A continuación listaremos todos los recursos son todas sus URLs, verbos disponibles y un ejemplo de sus datos.

5.3.1. Usuario

```
{
  "id": 1,
  "name": "John",
  "lastName": "Snow",
  "mail": "mail@gmail.com",
  "password": "ghost"
}
```

Verbo	Ruta	Acción
GET	/users/	Retorna todos los usuarios (paginado)
GET	/users/<id>/	Retorna el usuario con la id dada
POST	/users/	Crea un usuario con los datos dados
PUT	/users/<id>/	Modifica el usuario que tiene la id dada con los datos mandados
DELETE	/users/<id>/	Borra al usuario con la id dada
GET	/users/<id>/users/	Retorna todos los usuarios amigos del usuario con la id dada (paginado)
POST	/users/<id>/users/	Agrega al usuario con la id dada un amigo con los datos dados
DELETE	/users/<id>/users/<id>/	Borra al usuario con la id dada su amigo con una cierta id
GET	/users/<id>/memberships/	Retorna todas la membrecias a los grupos en los que pertenece
POST	/users/<id>/membership/	Crea una membresia del usuario a un grupo
DELETE	/users/<id>/membership/<id>/	Borra la membresia de un usuario a un grupo
GET	/users/<id>/purchase/	Retorna todas las compras del usuario
POST	/users/<id>/purchase/	Le agrega una compra a el usuario
DELETE	/users/<id>/purchase/<id>/	Borra una compra de un usuario

5.3.2. Compra

```
{
  "id": 1,
  "group" : 1
  "name" : "Birras",
  "cost" : 150,
  "buyer" : 1,
  "position" : "-34.8994346,-56.186796799999996",
  "participants" : [2, 3],
  "dateTime" : "2018-06-29_08:15:27.243860"
}
```

Verbo	Ruta	Acción
GET	/purchases/	Retorna todas las compras (paginado)
GET	/purchases/<id>/	Retorna la compra con esa id
POST	/purchases/	Crea una compra con los datos dados
PUT	/purchases/<id>/	Modifica la compra que tiene la id dada con los datos mandados
DELETE	/purchases/<id>/	Borra la compra con la id dada

5.3.3. Grupo

```
{
  "id": 1,
  "name": "John",
  "admin": 1
}
```

Verbo	Ruta	Acción
GET	/groups/	Retorna todos los grupos (paginado)
GET	/groups/<id>/	Retorna el grupo con la id dada
POST	/groups/	Crea un grupo con los datos dados
PUT	/groups/<id>/	Modifica el grupo que tiene la id dada con los datos mandados
DELETE	/groups/<id>/	Borra al grupo con la id dada
GET	/groups/<id>/memberships/	Retorna todos los usuarios del grupo con la id dada (paginado)
POST	/groups/<id>/memberships/	Agrega una membresia al grupo
DELETE	/groups/<id>/memberships/<id>/	Borra una membresia de un grupo
GET	/groups/<id>/purchases/	Retorna todas las compras de un grupo (paginado)
POST	/groups/<id>/purchases/	Agrega una compra a un grupo
DELETE	/groups/<id>/purchases/<id>/	Borra una compra de un grupo

6. Prototipos

La interfaz ha sido diseñada con tendencia minimalista, a partir de colores sólidos y contrastantes ya que los mismos añaden usabilidad siendo más fácil diferenciar los elementos que la componen. También se prioriza la consistencia, se utiliza Roboto como fuente tipográfica, la misma que utiliza el sistema operativo móvil Android a partir de su versión 4.0 licenciada bajo la Apache License en todas las etiquetas, campos de texto, selectores, combos y otros elementos presentes.

Se juega con los distintos tamaños de letra para destacar la jerarquía o importancia de cada elemento. Solo se muestra la mínima información necesaria para interactuar con la aplicación, este minimalismo sigue las tendencias del software actual y fortalece a la aplicación para que sea más intuitiva de usar.

A continuación se encuentran los principales prototipos del sistema relacionados a su correspondiente historia de usuario.

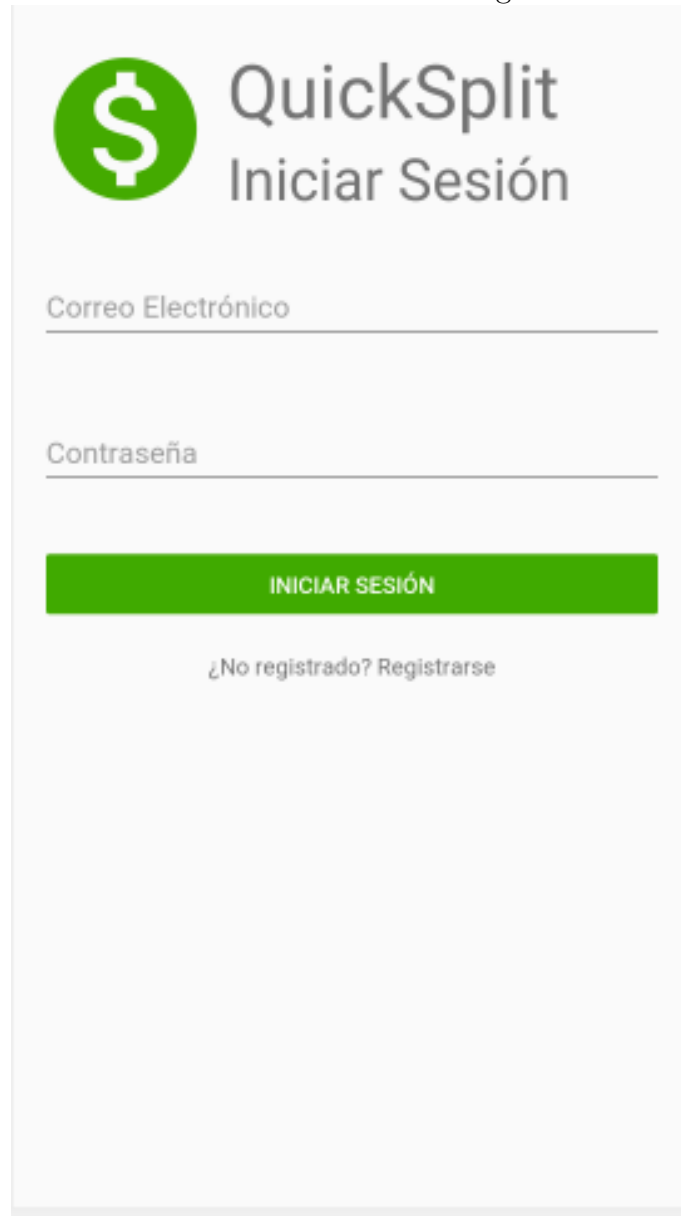
Historia de Usuario: Registro



The image shows a registration form for 'QuickSplit'. At the top left is a green circular logo with a white dollar sign. To its right, the text 'QuickSplit' is in a large, bold, dark grey font, and 'Registrarse' is in a slightly smaller, bold, dark grey font below it. Below the header, there are five input fields, each with a label above it: 'Nombre', 'Apellido', 'Correo Electrónico', 'Contraseña', and 'Repetir Contraseña'. Each label is in a light grey font. At the bottom of the form is a solid green rectangular button with the word 'REGISTRARSE' in white, uppercase letters. Below the button, centered, is a link in a small, dark grey font that reads '¿Ya registrado? Iniciar Sesión'.

Figura 6.1: Pantalla de Registro

Esta pantalla muestra los campos que deben ser completados para que un usuario se dé de alta en el sistema. El usuario tiene un acceso directo a la pantalla de login por si ya se encuentra registrado.



The image shows a login screen for an application named 'QuickSplit'. At the top left is a green circular logo with a white dollar sign. To its right, the text 'QuickSplit' is displayed in a large, bold, dark grey font, with 'Iniciar Sesión' below it in a smaller, regular dark grey font. Below the logo and text are two input fields. The first is labeled 'Correo Electrónico' in a light grey font, followed by a horizontal line. The second is labeled 'Contraseña' in a light grey font, also followed by a horizontal line. Below these fields is a solid green rectangular button with the text 'INICIAR SESIÓN' in white, uppercase letters. At the bottom of the form, there is a link that reads '¿No registrado? Registrarse' in a small, dark grey font.

Figura 6.2: Pantalla de Inicio de Sesión

Si es la primera vez que el usuario ingresa a la aplicación, o si la sesión se cerró previamente se muestra una pantalla para ingresar las credenciales. En la misma pantalla se encuentra un acceso directo para ir a la pantalla de registro, en caso de que el usuario no se encuentre registrado.

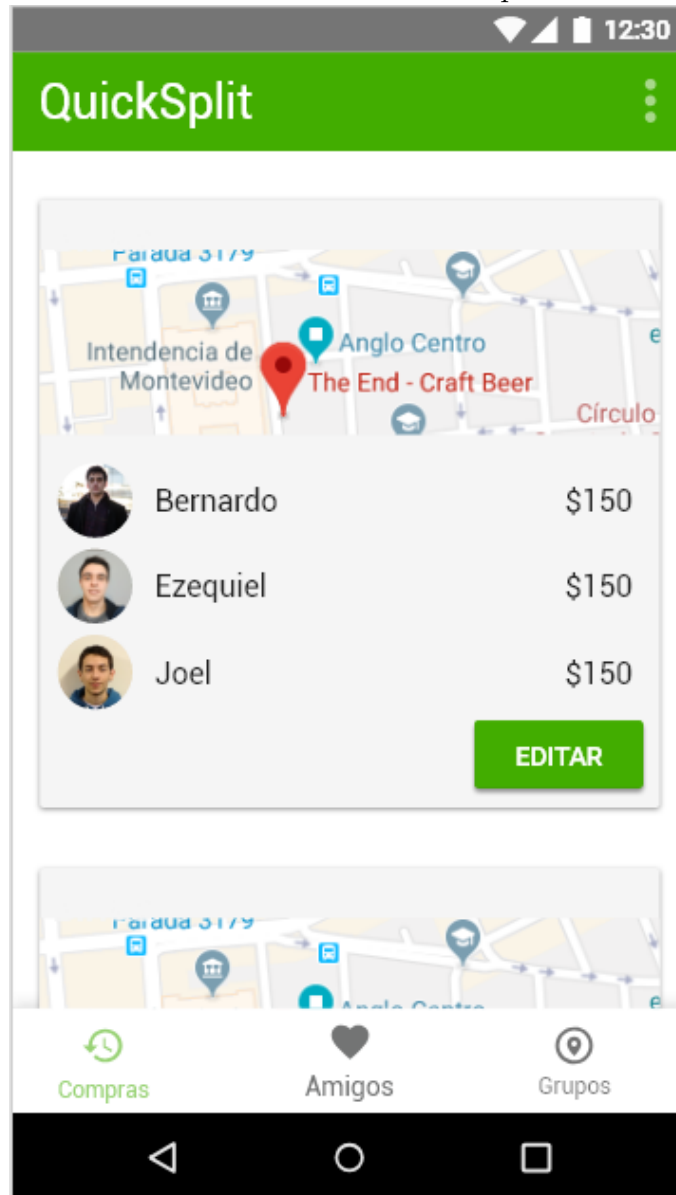


Figura 6.3: Pantalla de Compras

Luego de que el usuario inicia sesión se muestra una pantalla donde aparecen todas las compras en las que el usuario participó. En la tarjeta de la compra se muestra el desglose de costos que deberá pagar cada miembro que participó en la misma. En el banner superior aparece la ubicación donde fue realizada la compra, o las imágenes de los productos si la ubicación no fue ingresada, si no se provee ni la ubicación ni una imagen entonces se muestra una imagen por defecto.

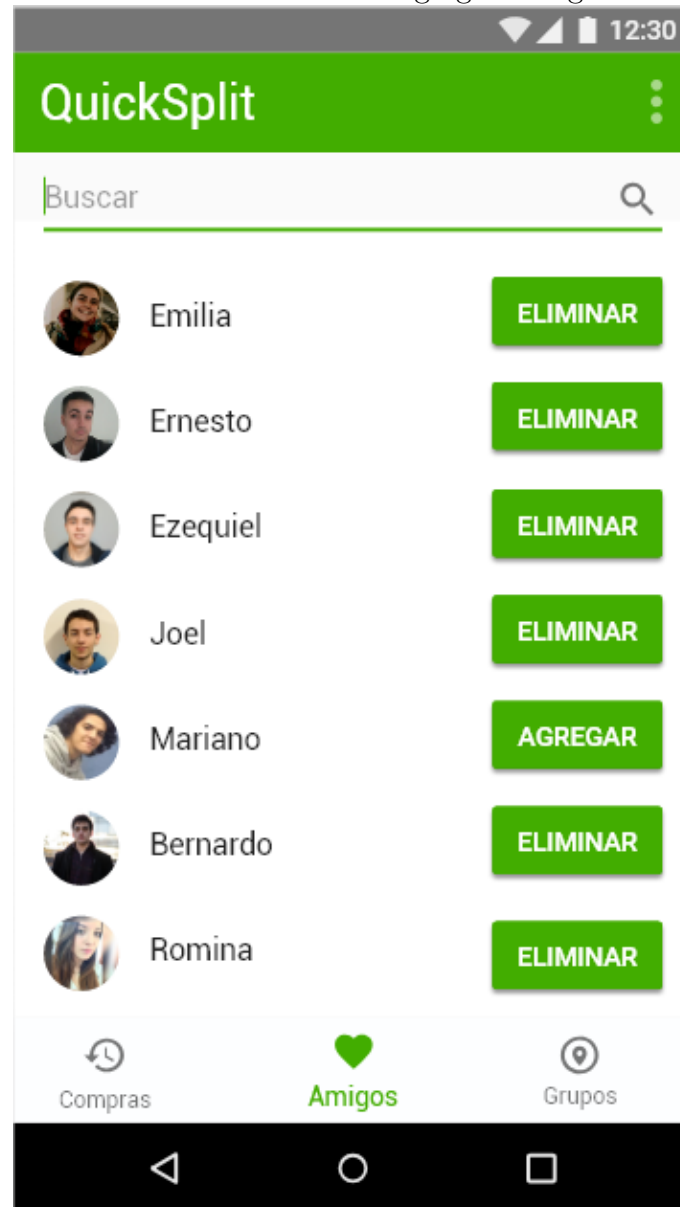


Figura 6.4: Pantalla amigos

La pestaña amigos muestra a los amigos que la persona tiene en el sistema, pudiendo buscar e invitar nuevos amigos escribiendo el nombre en el buscador integrado.

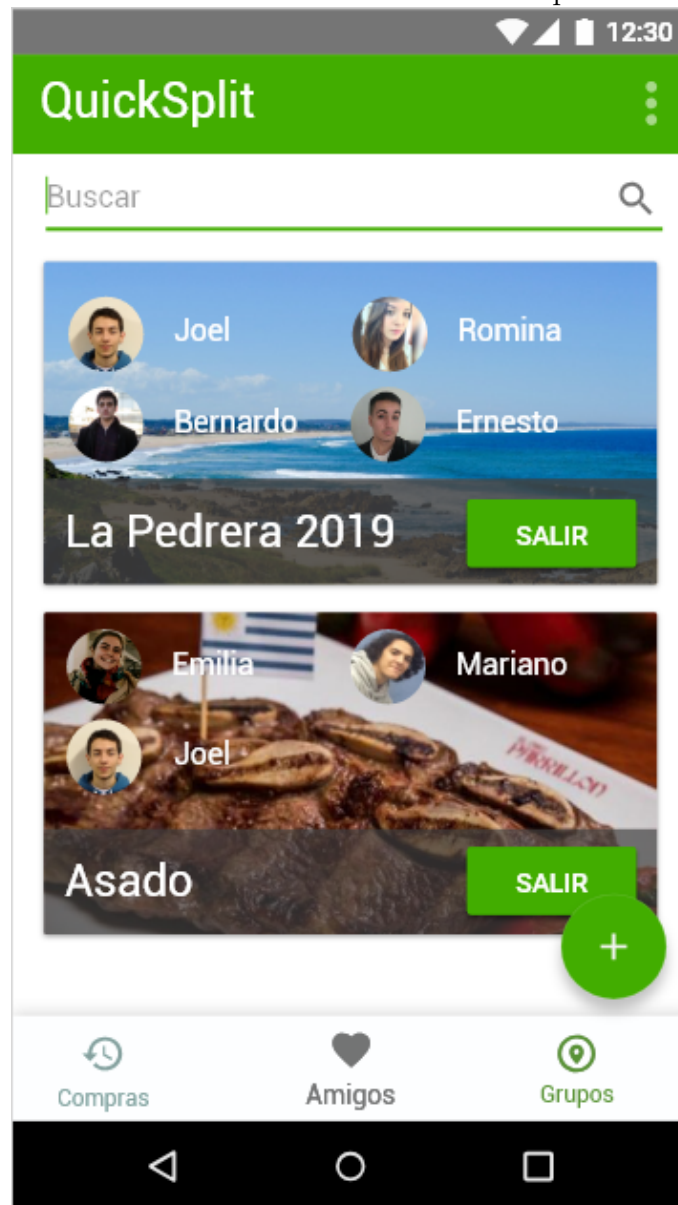


Figura 6.5: Pantalla grupos

La pestaña grupos muestra todos los grupos a los que el usuario pertenece, pudiendo salir de los grupos existentes o agregar un nuevo grupo de amigos.

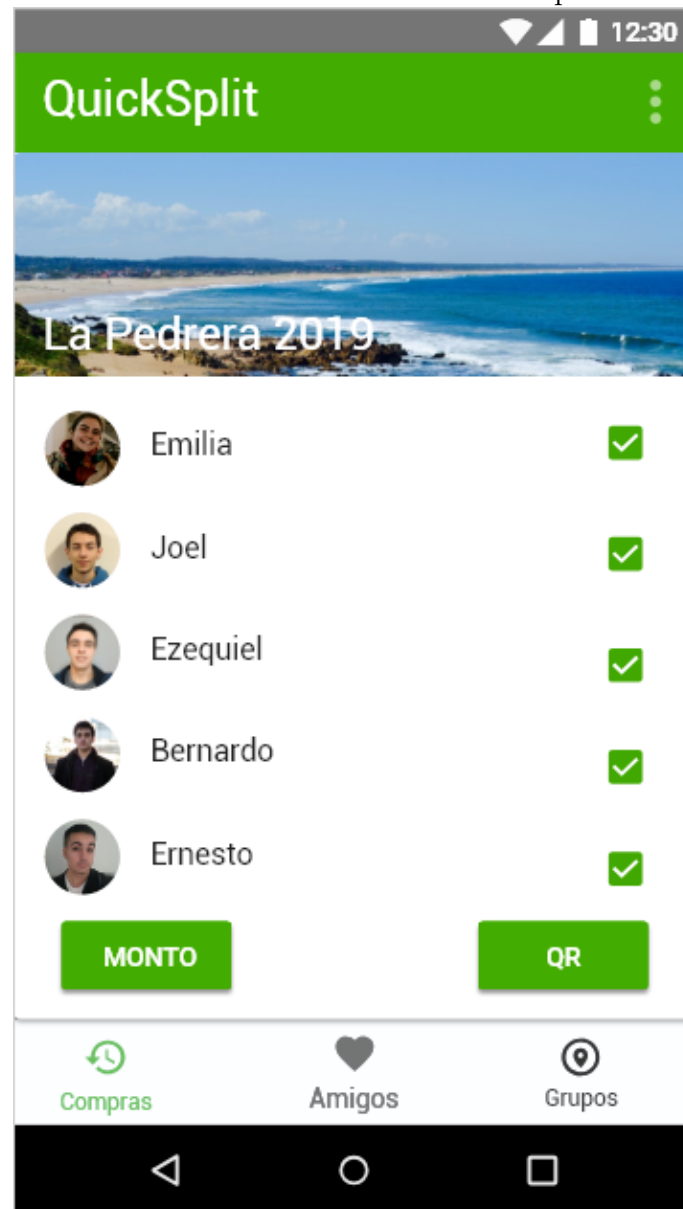


Figura 6.6: Pantalla grupo

Una vez dentro de un grupo se puede agregar una compra escaneando el código QR o ingresando el monto manualmente. Si se presiona sobre QR se despliega la cámara del dispositivo.

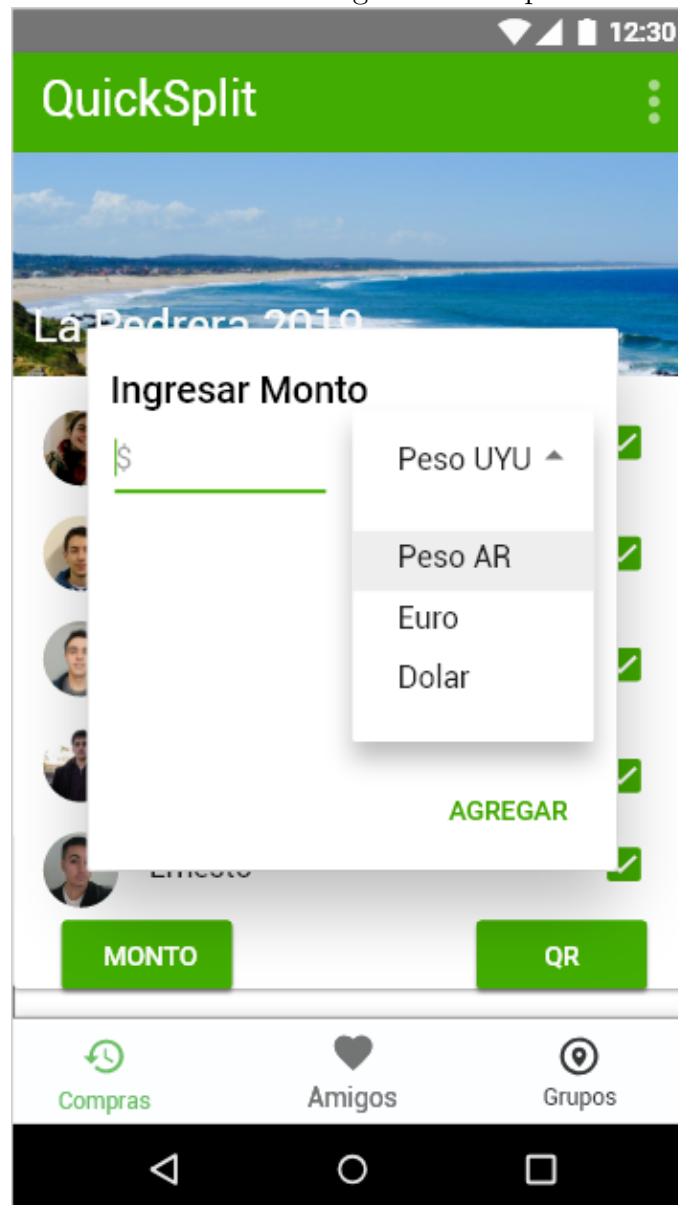


Figura 6.7: Pantalla grupo

Si se elige ingresar el monto manualmente el usuario puede seleccionar el tipo de moneda en una ventana modal, luego ingresa el monto a través de un teclado numérico.

Historia de Usuario: Logout y Editar Datos de Usuario

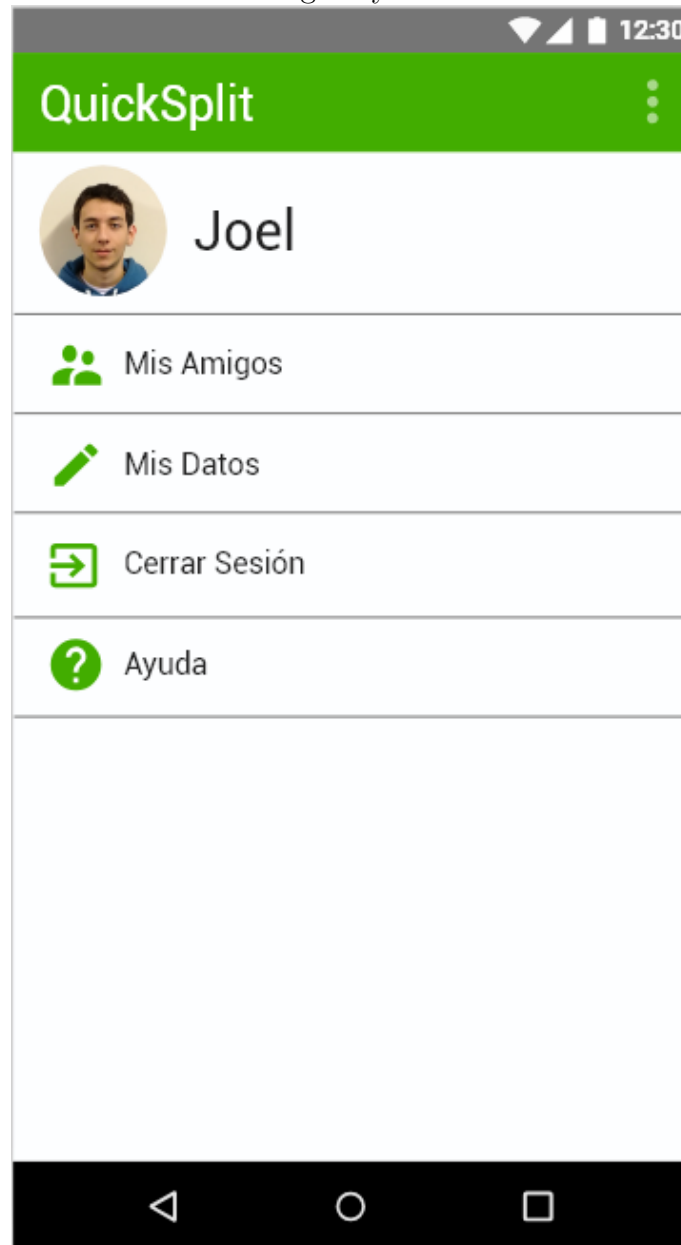


Figura 6.8: Cerrar Sesión - Editar Datos

Si el usuario logueado quiere cerrar la sesión o modificar sus datos puede acceder al menú de opciones de configuración.

7. Aseguramiento de la Calidad

7.1. Calidad el Código

Durante todo el desarrollo de la aplicación se trata de respetar los estándares de los lenguajes C# y Java, teniéndose también en cuenta aspectos considerados fundamentales para aplicar las buenas prácticas y obtener de este modo un código limpio, entendible, reutilizable y mantenible. Por parte del equipo se definieron los siguientes lineamientos a tener en cuenta a la hora de programar:

- Nombres nemotécnicos y pronunciables.
- Variables en minúscula.
- Clases en mayúscula.
- Métodos en minúscula para C# y en mayúscula Java.
- Llaves estilo Kernighan y Ritchie.
- CamelCase para diferenciar palabras.
- Los métodos deben recibir no más de tres argumentos por parámetro.
- Se listan primero los métodos públicos y luego los privados.
- Métodos con una única responsabilidad.
- Métodos con hasta 50 líneas de código.
- Clases con hasta 100 líneas de código.

7.1.1. Evidencia de Clean Code

Para poder cumplir con los aspectos mencionados respecto al Code review determinado en la Definition of Done (ver 3.5), los métodos tienen una única responsabilidad, son legibles sin la necesidad de hacer scroll y no contienen comentarios.

```
private async Task<UserModel> TryToHandle(CreateUserCommand request)
{
    User toCreate = new User()
    {
        Name = request.Name,
        LastName = request.LastName,
        Mail = request.Mail,
        Telephone = request.Telephone,
        Password = request.Password
    };

    await context.Users.AddAsync(toCreate);
    await context.SaveChangesAsync();

    return new UserModel(toCreate);
}
```

Cada clase cumple con el siguiente esqueleto: atributos públicos, atributos privados, constructor, métodos públicos y por ultimo métodos privados correspondientes al mismo.

```
public class GetUsersQueryHandler : IRequestHandler<GetUsersQuery, IEnumerable<UserModel>>
{
    private IQuickSplitContext context;

    public GetUsersQueryHandler(IQuickSplitContext context)
    {
        this.context = context;
    }

    public async Task<IEnumerable<UserModel>> Handle(GetUsersQuery request, CancellationToken cancellationToken)
    {
        return await context
            .Users
            .Select(user => MapToModel(user))
            .ToListAsync();
    }

    private UserModel MapToModel(User user)
    {
        return new UserModel()
        {
            Id = user.Id,
            Name = user.Name,
            LastName = user.LastName,
            Telephone = user.Telephone,
            Mail = user.Mail
        };
    }
}
```

Se han utilizado las llaves al estilo Kernighan y Ritchie así como también se mantuvo la misma indentación en todas las clases.

```

public class UserModel
{
    public UserModel()
    {
    }

    public UserModel(User user)
    {
        Id = user.Id;
        Name = user.Name;
        LastName = user.LastName;
        Mail = user.Mail;
        Telephone = user.Telephone;
    }

    public int Id { get; set; }

    public string Name { get; set; }

    public string LastName { get; set; }

    public string Mail { get; set; }

    public string Telephone { get; set; }
}

```

Se ha tomado la convención de que cada controlador se compone por los métodos para modificar los recursos en el siguiente orden: GET, POST, PUT, DELETE.

```

[ApiController]
public class UsersController : BaseController
{
    // GET
    [HttpGet(Name = "GetUser")]
    public async Task<ActionResult<IEnumerable<UserModel>>> Get()
    {
        IEnumerable<UserModel> users = await Mediator.Send(new GetUsersQuery());
        return Ok(users);
    }

    // GET
    [HttpGet("{id}")]
    public ActionResult<string> Get(int id)
    {
        return "value";
    }

    // POST
    [HttpPost]
    public async Task<ActionResult> Post([FromBody] CreateUserCommand user)
    {
        UserModel created = await Mediator.Send(user);
        return CreatedAtRoute("GetUser", created);
    }

    // PUT
    [HttpPut("{id}")]
    public async Task<ActionResult<UserModel>> Put(int id, [FromBody] UpdateUserCommand command)
    {
        command.Id = id;
        UserModel updated = await Mediator.Send(command);
        return Ok(updated);
    }

    // DELETE
    [HttpDelete("{id}")]
    public void Delete(int id)
    {
    }
}

```

7.2. Testing

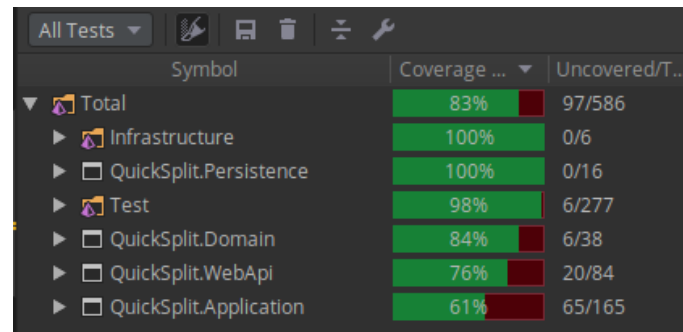
7.2.1. Pruebas Unitarias

Las pruebas unitarias serán creadas utilizando el framework XUnit ya que este comparado con otros frameworks como MSTest provee una forma más fácil de leer y más rápida de escribir para hacer unit tests. Para las pruebas unitarias nos basaremos en los principios denominados por el acrónimo F.I.R.S.T que establecen lo siguiente:

- **Fast (Rápido):** Las pruebas unitarias deben ser rápidas (menores a 1 segundo) ya que pueden llegar a haber cientos de ellas. Un desarrollador no debería dudar en correrlas porque son lentas.
- **Independent (Independiente):** Las pruebas no debería de depender de elementos que no son lo que queremos probar. No deberíamos depender de un orden de corrida de las pruebas.
- **Repeatable (Repetible):** Las pruebas tiene que poderse repetir siempre cuantas veces se quiera y debería siempre retornas los mismo resultados.
- **Self-Validating (Auto-Validables):** Cada prueba debe retornas un resultado. Nunca se debería tener que verificar el resultado manualmente.

- **Thorough (Completa):** Las pruebas debería cubrir todos los casos de uso y no solo el 100 % del código. Se deben probar todos los casos bordes, todas las posibles excepciones y se debe probar grandes cantidades de datos.

Para la Iteración 1 las historias de usuario que entraron fueron las mencionadas en la sección Planificación (ver 3.1). A continuación se muestran los resultados de las pruebas unitarias correspondiente a las mismas:



Symbol	Coverage ...	Uncovered/T...
Total	83%	97/586
Infrastructure	100%	0/6
QuickSplit.Persistence	100%	0/16
Test	98%	6/277
QuickSplit.Domain	84%	6/38
QuickSplit.WebApi	76%	20/84
QuickSplit.Application	61%	65/165

Figura 7.1: Cobertura de las Pruebas Unitarias

7.2.2. Pruebas de Integración

Para las pruebas de integración utilizaremos la metodología Big Bang. En esta metodología todos los componentes y módulos del proyecto (incluida la base de datos real) se prueban al mismo tiempo. Utilizamos esta metodología porque es rápida y sencilla de utilizar y dado el tamaño de nuestro sistema no consideramos que ameritaba utilizar una metodología más compleja.

Utilizaremos una parte del framework ASP.NET Core y XUnit para realizar las pruebas de integración. Esta parte del framework nos permite simular una request HTTP y verificar su resultados. Optamos utilizar este sistema sobre un cliente de HTTP real como Postman porque escribir pruebas en C# con estas librerías es más fácil y rápido y además los resultados de simular las llamadas HTTP son los mismos que los resultados reales.

7.2.3. Pruebas de Front-End

Para el Front-End consideramos que no es necesario automatizar las pruebas, ya que una vez que un integrante del grupo programa una funcionalidad deberá verificarla manualmente. Adicionalmente también probarán la aplicación varios usuarios ajenos al desarrollo de la misma y se realizarán Monkey Tests con el fin de detectar la mayor cantidad de bugs.

8. Anexo

8.1. Tablero Principal

<https://trello.com/invite/b/x3FtnWWF/d4f5f2ab21f2162ed22e64e26f35f9ea/obligatorio-ingenieria-en-la-practica>

8.2. Tablero de Estimación

<https://trello.com/invite/b/GNBA2fAD/7da8d2cad3886e4157286581f5a36965/estimaci%C3%B3n-de-us>

8.3. Tablero de Bugs

<https://trello.com/invite/b/zMtdPxoN/372ee7b0f5db257e54c45e60153b5aca/bug-tracker>

Bibliografía

- [1] Universidad ORT Uruguay. (2013) Documento 302 - Facultad de Ingeniería. [Online]. Available: <http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf>
- [2] Jason Taylor. Clean Architecture with ASP.NET Core 2.2. [Online]. Available: <https://www.youtube.com/watch?v=Zygw4UAxCdg&feature=youtu.be>
- [3] Microsoft. Integration tests in ASP.NET Core. [Online]. Available: <https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-2.2>

Parte II

Obligatorio 2

9. Introducción

Esta parte del documento corresponde a la segunda entrega del Obligatorio. La misma contiene las modificaciones, correcciones, y replanificaciones llevadas a cabo por parte del equipo para poder implementar el sistema planteado en la primera parte de este documento (ver 1.1) en su mayor totalidad. A su vez, se especifica el diseño final de dicho sistema, se evidencian las buenas prácticas de estilo y codificación utilizadas, clean code y diagramas para una mejor comprensión del mismo.

10. Manejo de versionado

Si bien el flujo de trabajo ha sido el definido en la primera sección del documento (ver 2.1) el mismo debió sufrir una pequeña modificación respecto al Front-End. Se tomo la decisión de crear una rama feature/Fragments conglomerando las activities, fragments, layouts, etcétera dentro de una misma rama con el fin de simplificar el desarrollo del mismo.

11. Planificación

Durante el desarrollo de la aplicación, si bien hemos intentado acoplarnos lo mas posible al Release Plan planteado (ver 3.1), el mismo sufrió modificaciones debido a la velocidad de trabajo de cada integrante así como también aspectos que no fueron tenidos en cuenta a la hora de definir el mismo (período de exámenes, entregas de otras materias, compromisos personales, etc). Para poder mitigar dicho aspecto se realizaron los siguientes cambios respecto a las estimaciones de determinadas historias de usuario:

- Mapa de Compras por GPS
 - Prioridad: 5
 - Tamaño: S
 - Tipo: Spike
- Mapa de Compras por GPS
 - Prioridad: 6
 - Tamaño: L
 - Descripción:
Como un usuario perteneciente a un grupo
Quiero ver en un mapa donde se realizaron las compras
Para saber donde se realizaron las compras
 - Criterios de aceptación:
 1. Visualizar un mapa con puntos donde se realizaron las compras
- Reconocimiento de QR de Tickets
 - Prioridad: 5
 - Tamaño: S
 - Tipo: Spike
- Reconocimiento de QR de Tickets
 - Prioridad: 6
 - Tamaño: M

- Descripción:
Como usuario dentro de un grupo
Quiero escanear códigos QR de tickets
Para poder ingresar compras mas rápidamente
- Criterios de aceptación:
 1. No es obligatorio usar QR para realizar la compra
 2. Si no se consiguen los datos mediante QR se le debe avisar al usuario que lo tiene que agregar manual
- Compras mediante OCR
 - Prioridad: 6
 - Tamaño: L
 - Descripción: Investigación
 - Tipo: Spike
- Compras mediante OCR
 - Prioridad: 7
 - Tamaño: XL
 - Descripción:
Como usuario dentro de un grupo
Quiero escanear un ticket
Para obtener el desglose de una compra
 - Criterios de aceptación:
 1. El reconocimiento mediante OCR de la compra falla como máximo un 30 % de las veces
- Login con Google
 - Prioridad: 6
 - Tamaño: S
 - Descripción: Investigación
 - Tipo: Spike
- Login con Google
 - Prioridad: 7
 - Tamaño: M
 - Descripción:
Como usuario sin registrarse
Quiero ingresar con mi cuenta de Google
Para poder navegar en la aplicación

- Criterios de aceptación:

1. No es obligatorio utilizar una cuenta de Google para poder ingresar

Dichos cambios nos han sometido a tener que recurrir al plan de contingencia para la historia de usuario "Desglose de Compras mediante OCR" (ver 3.6) tomándose la decisión de dejar fuera del alcance dicha historia de usuario.

Por otro lado, se ha tomado como convención que a la hora de tomar una historia de usuario para ser implementada, si los criterios de aceptación dependen de otra historia de usuario, la misma debe de ser dejada de lado hasta que dicha historia de la cual se depende cumpla con la Definition of Done (ver 3.5).

A su vez, a la hora de tomar las historias de usuario de cada iteración, se dejaron para el final aquellas que incluyeran investigación, es decir, Spikes.

Como descripto en el Release Plan mencionado anteriormente, nos hemos juntado diariamente en la facultad luego de clases preguntándonos que hicimos ayer, que haremos hoy y discutimos los problemas que tuvimos. Dichas reuniones no solo sirvieron para poder informarnos de determinadas demoras a la hora de implementar historias de usuario debido a que se utilizó un entorno de desarrollo con el cual no estábamos familiarizados, sino que también, para informar de bugs detectados, pudiéndose arreglar los mismos tempranamente.

12. Descripción de diseño

El diseño del Back-End resulto ser muy bueno por lo que no tuvimos que cambiar mucho respecto a lo planeado. El único cambio significativo fue que la responsabilidad de leer los tickets fue trasladada al Front-End por un tema de usabilidad. (Es mucho mas usable escanear un QR desde el celular y conseguir una respuesta inmediata que tener que mandar una foto del QR al servidor y esperar la respuesta).

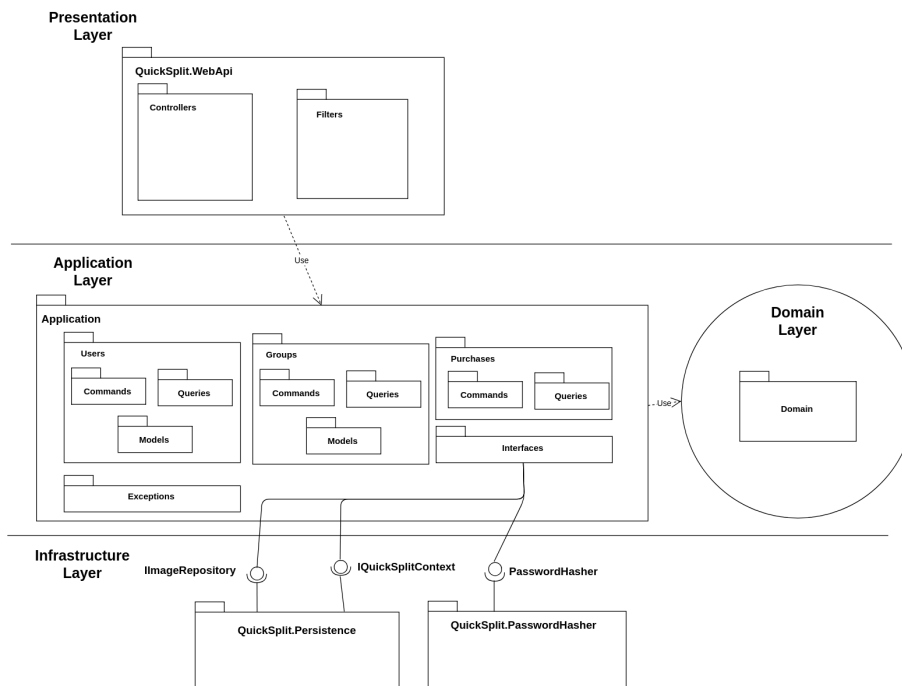


Figura 12.1: Diagrama de paquetes

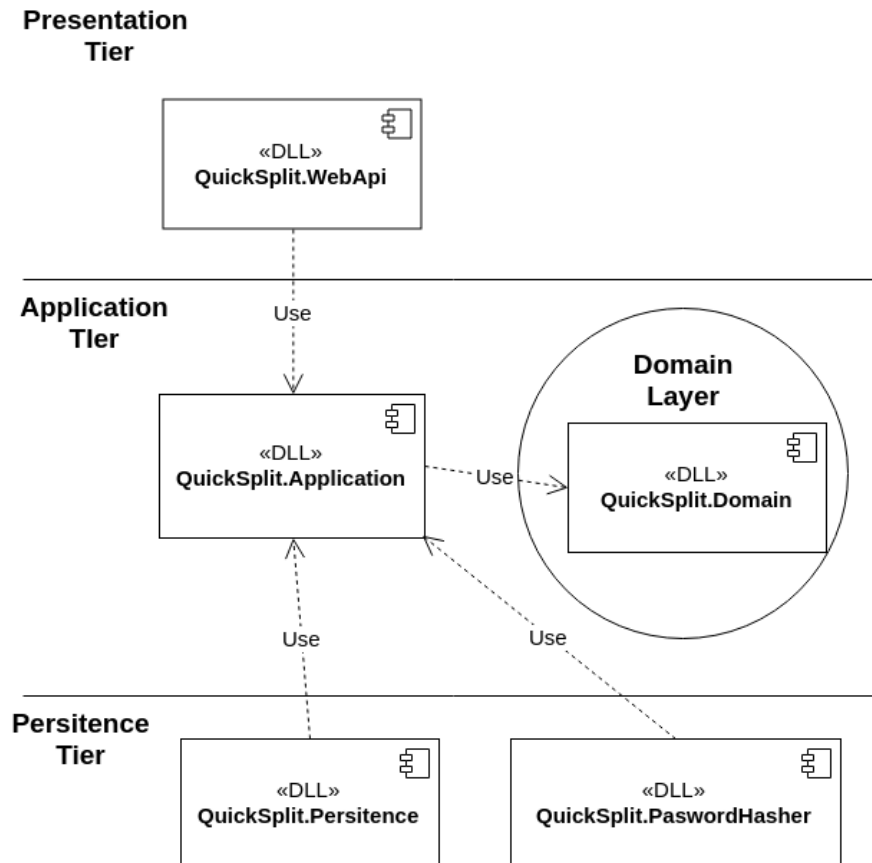


Figura 12.2: Diagrama de componentes

12.1. Calidad del Diseño

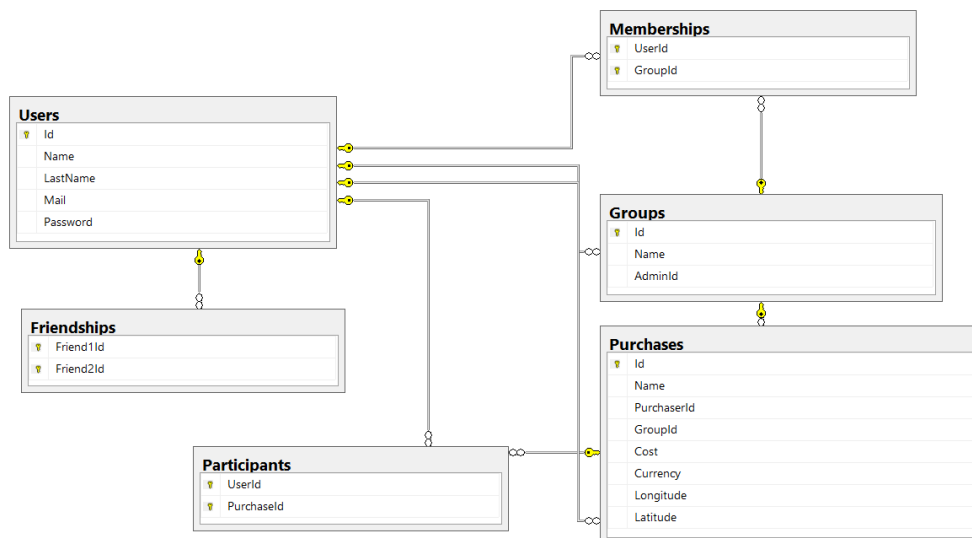
Al no haber cambiado mucho el diseño, los atributos de calidad del Back End siguen siendo lo mismos. (ver 5.1)

12.1.1. Front-End

Aunque no fue documentado en la primera instancia del obligatorio y tampoco fue un criterio de calidad, el equipo discutió la opción de utilizar el patrón Clean Architecture o el patrón de repositorio Bob Uncle para lograr un desacoplamiento de la lógica de conexión con el Back-End y el código que responde a eventos del Front-End. Lamentablemente, por falta de conocimiento y otros factores no fue posible implementar estos patrones produciendo repetición de código y una sobrecarga en las responsabilidades en las clases correspondientes al Front-End. De todas maneras, aunque no se hayan aplicado patrones específicos (a nivel de código) se programó intentando dar un orden lógico a las clases y las responsabilidades de las mismas.

12.2. Modelo de Tablas

A continuación se muestra el modelo de tablas de la base de datos.



12.3. Diseño UI UX

12.3.1. Integración con Sensores

Realizar una aplicación mobile tiene diferencias respecto de una aplicación desktop ya que los dispositivos móviles cuentan con distintos sensores (cámara, giroscopio, GPS, entre otros) que pueden ser usados para mejorar la experiencia de interacción entre el usuario y la aplicación. Teniendo esto presente, decidimos hacer uso de algunos de estos sensores para mejorar la experiencia y la forma con la que los usuarios interactúan con la misma. Más concretamente hicimos uso de la cámara, y del GPS. La cámara es usada en varios lugares, para almacenar imágenes de compras, y avatares de usuario que son enviados al Back-End para estar disponibles para todos los usuarios. Por otra parte, la aplicación cuenta con una opción para escanear códigos QR. Todo el procesamiento del QR ocurre en el Front-End, aunque en un principio evaluamos la posibilidad de trasladar parte de la lógica de esta funcionalidad al Back-End, decidimos dejarla en el Front-End ya que es más fácil, y es una funcionalidad que es propia de una aplicación mobile, es decir si se reutiliza el Back-End para una aplicación desktop, esta funcionalidad no tendría mucho sentido. La aplicación se conecta a la página de la Dirección General Impositiva¹ para obtener el monto y la moneda de una compra. Por otro lado también contamos con el GPS para localizar al usuario y guardar la ubicación de las compras de manera más rápida sin necesidad de buscar en el mapa. Si el usuario no quiere otorgar los permisos de localización puede señalar el punto manualmente. Esto nos pareció de gran importancia, ya que si no desea otorgar permisos, se puede seguir utilizando

¹Unidad Ejecutora responsable de la administración de los impuestos internos del país.

las características más importantes sin mayores problemas, viéndose restringidas algunas funcionalidades.

Además del uso de los sensores para sacar el máximo provecho, generalmente los celulares Android tienen una cuenta de Gmail sincronizada, por lo que es útil crear un registro a partir de estas credenciales otorgadas por Google aboliendo al usuario la necesidad de recordar credenciales para iniciar sesión en la aplicación. Por otro lado el inicio de sesión solo se hace durante la configuración inicial. Después del primer login ya no es necesario volver a ingresar las credenciales.

12.3.2. Descripción general

El diseño de la UI se hizo de acuerdo a lo planeado en la primer parte de obligatorio con algunas variaciones para hacer más intuitiva y comfortable la experiencia de usuario.

La aplicación está pensada para verse cómodamente en una pantalla moderna de tamaño estándar (entendiendo por resoluciones estándar 640x1136, 720x1280, 750x1334, 1080x1920, y 1440x2560). Si bien la aplicación lo soporta perfectamente el landscape, el mismo viene desactivado por defecto ya que consideramos que no es útil utilizar esta vista para mostrar información, ver y completar formularios.

Desde el punto de vista técnico la aplicación cuenta con una activity principal donde desde un menú de navegación fijo en la parte inferior de la pantalla permite navegar entre las funcionalidades más importantes (Compras, Amigos y Grupos) cuando se selecciona una opción en el menú se cambia el fragmento y se genera una recyclerview con los ítems que se deben mostrar en pantalla. Si ocurre un problema en la conectividad o si aun no existen ítems para la recyclerview se muestra un mensaje amigable al usuario en el propio fragmento.

Para agregar Compras y Grupos, se utilizó un floating action button que lanza una nueva activity con campos para llenar los datos de la compra o grupo, este botón se posiciona en la parte inferior de la pantalla siendo accesible fácilmente con el dedo pulgar.



Figura 12.3: Activity Principal - Fragment Compras

12.3.3. Barra de Navegación Superior

Para la barra de navegación se utilizó una `AppBarLayout` que es utilizada como `SupportActionBar`, el haber utilizado esta barra nos permite agregar o quitar opciones de manera dinámica. Por ejemplo cuando se cambia al fragment de amigos, en la barra aparece un campo de búsqueda para poder buscar amigos y agregarlos.

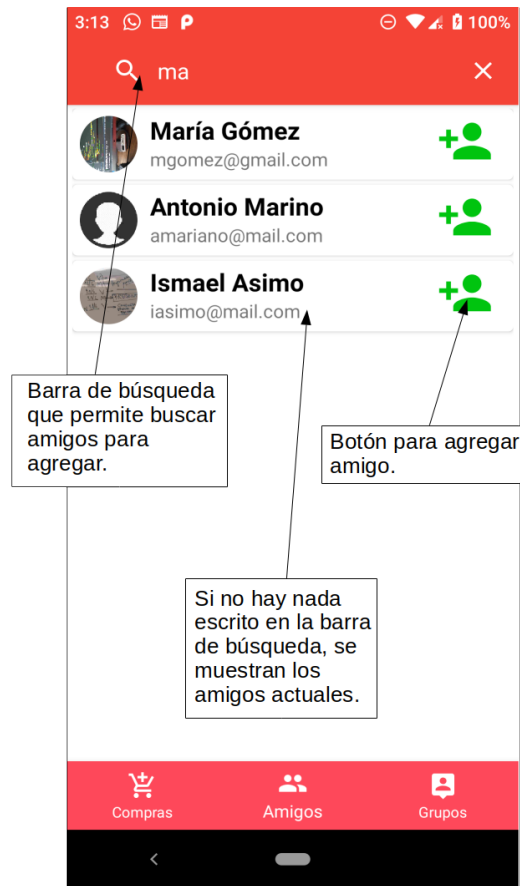


Figura 12.4: Fragment Amigos

Cuando se cambia a una pantalla secundaria, por ejemplo al Agregar Compra, Agregar Grupo, etcétera, las opciones de la barra superior cambian y se adecúan al contexto.

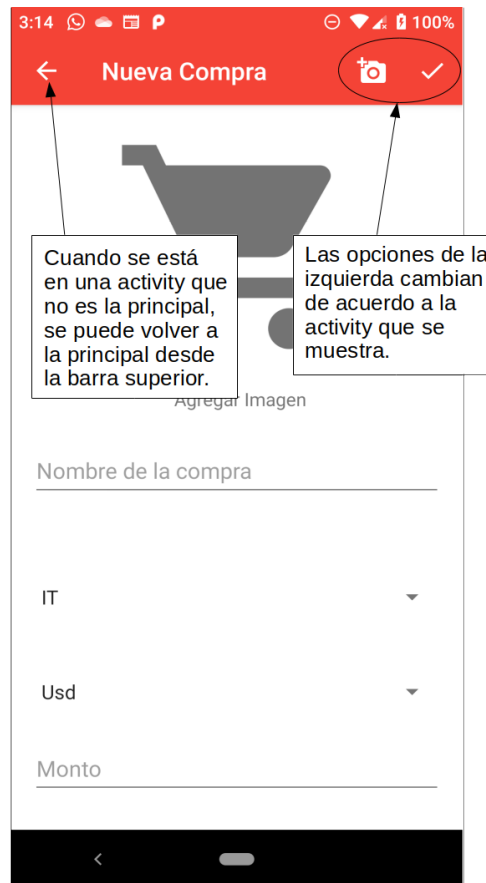


Figura 12.5: Agregar Compra

12.3.4. Manejo del Stack

El manejo del stack para volver hacia atrás es posible desde el botón de navegación del dispositivo, que de acuerdo a la documentación oficial de Google Developers todo dispositivo debe poseer uno de manera virtual o física, o desde un botón en la barra superior, que permite ir desde las activities secundarias hacia la activity principal. Una vez en la activity principal no importa el fragment que se muestra en pantalla (Compras, Amigos, Grupos) si se presiona el botón de atrás del dispositivo se cierra la aplicación completamente.

12.3.5. RecyclerViews

Para diseñar los items en la recyclerviews buscamos transmitir la mayor información necesaria para que el usuario entienda lo que está pasando sin sobrecargar demasiado la pantalla. Por ejemplo en el ítem de la compra se muestran los datos más relevantes que ayuden al usuario a recordar que compra está viendo. Se muestra el nombre, el precio y tipo de moneda, la imagen de la compra y también los miembros que participaron en la compra con su nombre y apellido, su email y su avatar.

Cada ítem tiene un botón o una serie de botones dependiendo del caso que permiten hacer operaciones sobre éste, por ejemplo, el ítem de un grupo tiene botones

para borrar, salir del grupo, modificar los datos, o ver la división de gastos de dicho grupo.

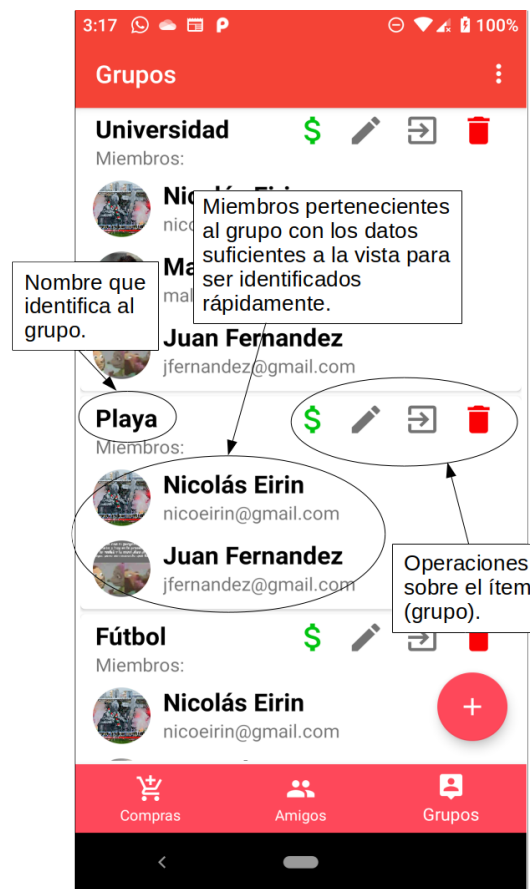


Figura 12.6: Fragment Groups

12.3.6. Carga de Imágenes

La carga de imágenes se hace a través de la biblioteca Picasso² lo que permite manejar las mismas de una manera eficiente y rápida ya que esta biblioteca se encarga automáticamente del cache de las imágenes.

12.3.7. Validación de los Formularios

Siempre que no se necesiten validaciones del Back-End, la validación de los campos se hace desde el Front-End para evitar enviar información innecesaria y evitar gastar datos, teniendo en cuenta que la aplicación podría estar usando datos móviles. Los errores se muestran de manera no invasiva debajo del campo en donde existe un error.

²Biblioteca picasso para Android: <https://square.github.io/picasso/>



Figura 12.7: Validación de Formulario - Activity Login

12.3.8. Mensajes de Carga

Si se requiere un dato que es necesario para hacer las demás operaciones se pone un mensaje de carga bloqueante que no le permite navegar por otras opciones al usuario. Si al minuto no obtuvo respuesta del servidor, se muestra el mensaje de error correspondiente.

Cuando se modifican datos, o se hacen cargas asíncronas donde no es necesario esperar por respuesta se muestra un toast con el mensaje correspondiente. En cuanto a los errores se aplica la misma lógica. Si el error no es bloqueante se muestra un toast con información de donde se produjo el mismo y adicionalmente el status code del servidor, si bien esta información no es de utilidad para el usuario, es importante que pueda verla para reportar el error y que los programadores encargados de mantener la aplicación puedan identificarlo y corregirlo lo más pronto posible. Por otro lado si el error es una falla en la comunicación y no una falla en el servidor, se muestra un mensaje a pantalla completa y se presenta un botón para recargar el fragmento o la activity.

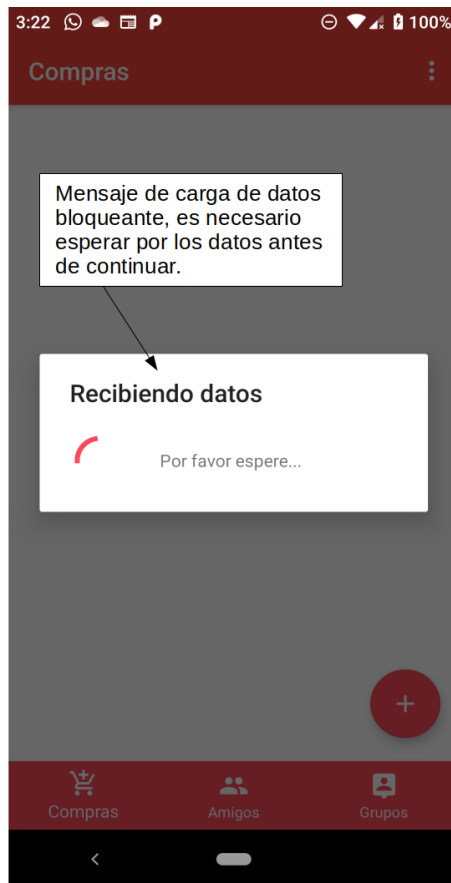


Figura 12.8: Mensaje de Carga

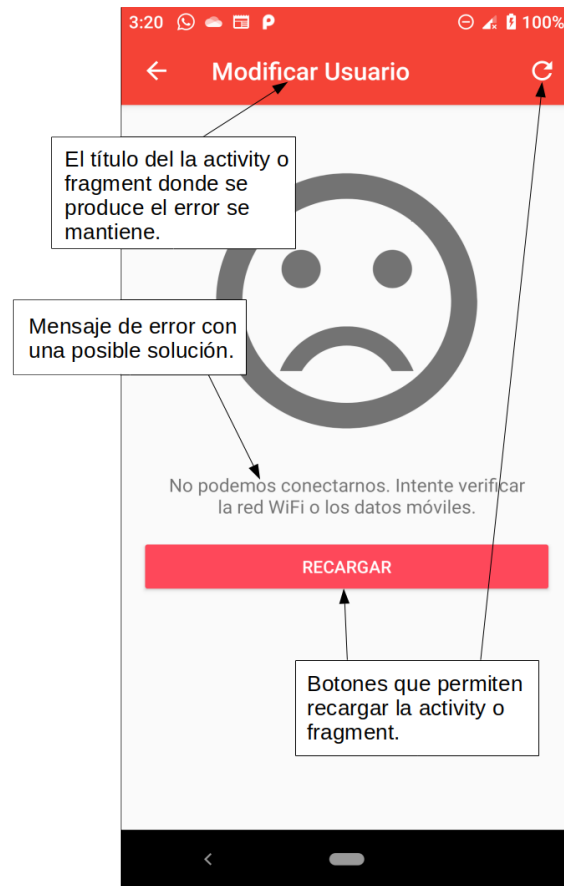


Figura 12.9: Mensaje de Error

12.3.9. Configuraciones Espontáneas

Para configuraciones espontáneas, como por ejemplo seleccionar una imagen de la cámara o la galería, o cambiar el tipo de moneda en el reporte de gastos se utilizan diálogos con las opciones necesarias.

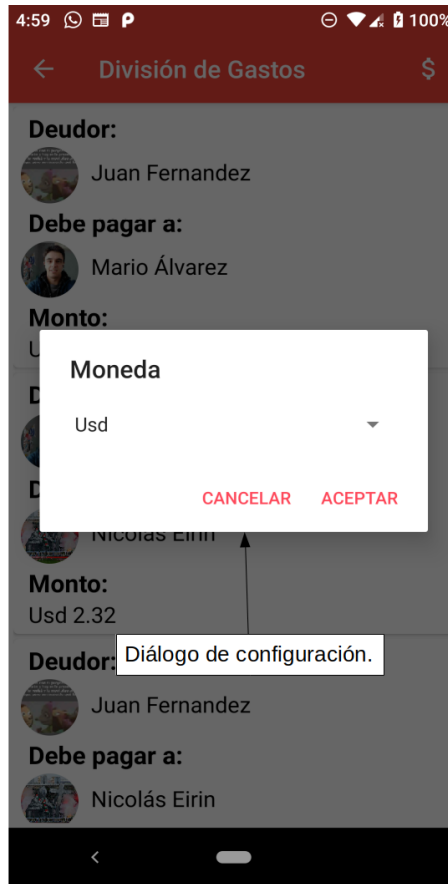


Figura 12.10: Diálogo de Configuración - Monedas - Reporte de Gastos

12.4. Diseño de la API

El diseño de la API Rest real fue cambiado un poco con respecto a lo planeado. Ver diseño planeado de la api. (ver 5.3)

El primer cambio significativo fue que no utilizamos clases de asociación entre los diferentes recursos. Por ejemplo en vez de tener un recurso membership (utilizando /group/id/memberships) que asocia User y Group, se puede acceder a los miembros de un grupo directamente (/group/id/users).

El otro cambio significativo fue incluir a las imágenes como recursos separados. Este cambio fue hecho por un tema de performance ya que pasar las imágenes en base64 dentro de un json es muy poco eficiente comparado con mandarlas como una imagen en HTTP.

El resto de los pequeños cambios fueron endpoints que fueron borrados porque no eran de utilidad y otros que fueron agregados para facilitar el desarrollo del Front end.

12.4.1. Diseño de la API real

A continuacion mostraremos el diseño de la API implementada.

Groups	
GET	/api/Groups/{id}
PUT	/api/Groups/{id}
DELETE	/api/Groups/{id}
GET	/api/Groups
POST	/api/Groups
PUT	/api/Groups/leave
GET	/api/Groups/{id}/users
GET	/api/Groups/{id}/purchases
POST	/api/Groups/{id}/purchases
GET	/api/Groups/{id}/reports

Figura 12.11: API de grupos

Authentications	
POST	/api/Authentications
Currencies	
GET	/api/Currencies

Figura 12.12: API de autenticacion y moneda

Purchases	
GET	/api/Purchases
POST	/api/Purchases
GET	/api/Purchases/{id}
PUT	/api/Purchases/{id}
GET	/api/Purchases/{id}/image
POST	/api/Purchases/{id}/image
GET	/api/Purchases/{id}/users

Figura 12.13: API de compras

Users	
GET	/api/Users
POST	/api/Users
GET	/api/Users/{id}
PUT	/api/Users/{id}
DELETE	/api/Users/{id}
GET	/api/Users/{id}/friends
POST	/api/Users/{id}/friends/{friendId}
DELETE	/api/Users/{id}/friends/{friendId}
GET	/api/Users/{id}/avatars
POST	/api/Users/{id}/avatars
GET	/api/Users/{id}/purchases

Figura 12.14: API de usuarios

12.5. Eficiencia

Luego de experimentar con la maquina virtual especificada en el plan (ver 5.2.2) llegamos a la concreción que esta no es suficientemente potente para manejar nuestra aplicación. Por lo tanto empezamos a utilizar nuestras maquinas personales para las pruebas de performance.

Bajo las condiciones especificadas en 5.2.2, llegamos a un promedio de aproximadamente 200ms por consulta. Este valor es superior al establecido en el plan pero es lo suficientemente bajo para que el tiempo de respuesta de la UI se mantenga por debajo del tiempo establecido (1 segundo).

Aunque estos resultados no son ideales son lo suficientemente buenos como para mantener la UI fluida y tener una buena experiencia de usuario.

13. Aseguramiento de la calidad

Si bien para el Back-End se han tenido en cuenta los lineamientos planteados en la primera sección de este documento (ver 7.1), a la hora de programar el Front-End, los mismos fueron más difíciles de llevar a cabo por lo que se plantearon los siguientes aspectos respecto al código de dicha capa:

- Nombres nemotécnicos
- Variables que no son públicas ni estáticas comienzan con m
- Las variables deben comenzar con su tipo para una mejor comprensión de lo que son. Esto aunque no es una práctica recomendada en la actualidad, es de mucha utilidad en este desarrollo de Android sobre Java, ya que los elementos visuales están presentes en las clases. Ejemplo: `mButtonCreateUser`
- El primer método a la vista de cada clase es el `onCreate()`

Como mencionado previamente, si bien no se pudieron aplicar los patrones de diseño deseados, se trató de representar cada requerimiento de la aplicación con una Activity o Fragment, por ejemplo para la historia de usuario *Crear Grupo* se creó una activity a parte. Esta práctica aunque puede parecer una mala idea ya que no cumple con el principio DRY¹ es buena si consideramos que en un futuro podrían existir nuevos requerimientos para la aplicación, si este fuera el caso, solo se tendría que agregar una nueva activity o fragment que represente el requerimiento deseado, sin necesidad de modificar más código existente que el manifiesto para incluirla en el caso de ser una activity.

Por otra parte hemos seguido las recomendaciones de nuestro querido amigo Martin Fowler² para mantener un estilo de codificación consistente y fácilmente comprensible. Este último factor tiene vital importancia en la mantenibilidad de código en el futuro. Complementario a esto se siguieron las prácticas recomendadas por los estándares de Java y los propuestos por Google para su desarrollo. Siendo de gran ayuda el Entorno de Desarrollo Integrado que provee Android Studio.

¹Do not Repeat Yourself: https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

²La frase hace referencia a lo planteado por Martin Fowler en su libro Clean Code.

13.0.1. Evidencia de Clean Code

Como previamente descripto en el documento anterior, a la hora de programar siempre se tuvo en cuenta que los métodos deben tener una única responsabilidad y deben ser legibles sin la necesidad de hacer scroll.

```
private void AddDebtToReport(User debtor, User debtee, double portion)
{
    (User, User) debtorDebtee = (debtor, debtee);
    if (!_dictionary.ContainsKey(debtorDebtee))
    {
        _dictionary[debtorDebtee] = 0d;
    }

    _dictionary[debtorDebtee] += portion;
}
```

Se ha utilizado la librería Linq ya que la misma bajo distintas situaciones nos permite reducir las líneas de código. Por ejemplo, a la hora de hacer consultas al contexto, o recorrer listas para aplicar determinado método a los elementos de la misma. Se evitan de esta forma los iteradores for o foreach, los cuales molestan a la hora de leer el código.

```
private async Task<User> GetPurchaserIfValid(CreatePurchaseCommand request)
{
    return await _context.Users.FindAsync(request.Purchaser) ?? throw new InvalidCommandException("El comprador no existe");
}
```

```
public async Task<IEnumerable<UserModel>> Handle(GetPurchaseParticipantsQuery request, CancellationToken cancellationToken)
{
    Purchase purchase = await _context
        .Purchases
        .Include(p => p.Participants)
        .ThenInclude(p => p.User)
        .FirstOrDefaultAsync(r => r.Id == request.PurchaseId, cancellationToken: cancellationToken)
        ?? throw new InvalidQueryException("No existe la compra");

    return purchase
        .Participants
        .Select(p => new UserModel(p.User));
}
```

Se ha tomado la convención de que si una línea de código excede el largo de la pantalla, mediante un salto de línea, la misma debe ser reubicada.

```
public async Task<IEnumerable<UserModel>> Handle(GetFriendsQuery request, CancellationToken cancellationToken)
{
    User user = await _context
        .Users
        .FindAsync(request.UserId);

    if (user == null)
        throw new InvalidQueryException($"No existe usuario con id {request.UserId}");

    List<User> friends = await _context.Friendships
        .Where(friendship => friendship.Friend2Id == request.UserId)
        .Select(friendship => friendship.Friend1)
        .ToListAsync(cancellationToken: cancellationToken);

    return friends.Select(f => new UserModel(f));
}
```

Se ha puesto total énfasis en que para que el día de mañana la mantenibilidad del sistema no sea engorrosa, acoplarse a las buenas prácticas de estilo y codificación es esencial.

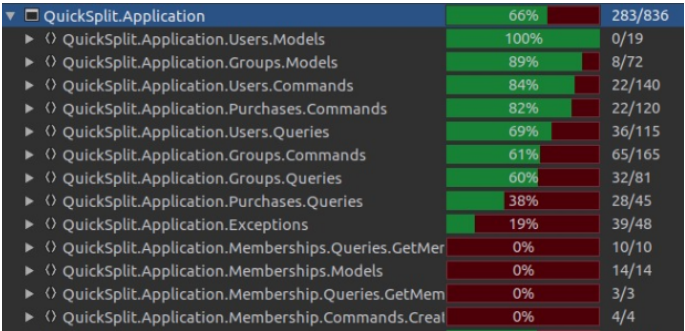
13.0.2. Testing

Pruebas unitarias

Dado que uno de los aspectos que determinan si una historia de usuario puede ser catalogada como terminada refiere a que la misma debe tener implementada

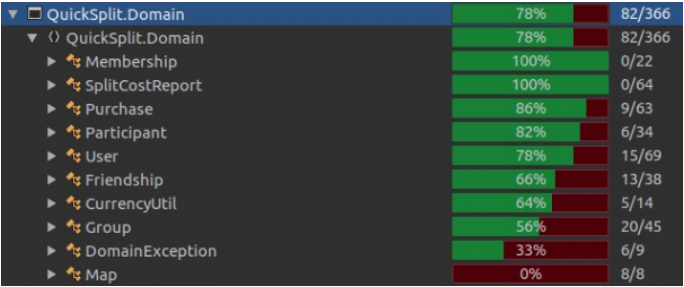
sus correspondientes pruebas unitarias, el código de dichas pruebas fue desarrollado conforme al desarrollo del código de implementación de cada historia de usuario. Se utilizó el framework XUnit el cual provee una forma fácil de leer y mas rápida de escribir el código. Siempre se tuvieron en cuenta los principios denominados por el acrónimo F.I.R.S.T previamente descriptos en la primera parte de este documento (ver 7.2.1). A continuación se detalla la cobertura obtenida para cada modulo:

Application:



QuickSplit.Application	66%	283/836
QuickSplit.Application.Users.Models	100%	0/19
QuickSplit.Application.Groups.Models	89%	8/72
QuickSplit.Application.Users.Commands	84%	22/140
QuickSplit.Application.Purchases.Commands	82%	22/120
QuickSplit.Application.Users.Queries	69%	36/115
QuickSplit.Application.Groups.Commands	61%	65/165
QuickSplit.Application.Groups.Queries	60%	32/81
QuickSplit.Application.Purchases.Queries	38%	28/45
QuickSplit.Application.Exceptions	19%	39/48
QuickSplit.Application.Memberships.Queries.GetMer	0%	10/10
QuickSplit.Application.Memberships.Models	0%	14/14
QuickSplit.Application.Membership.Queries.GetMem	0%	3/3
QuickSplit.Application.Membership.Commands.Creat	0%	4/4

Domain:



QuickSplit.Domain	78%	82/366
QuickSplit.Domain	78%	82/366
Membership	100%	0/22
SplitCostReport	100%	0/64
Purchase	86%	9/63
Participant	82%	6/34
User	78%	15/69
Friendship	66%	13/38
CurrencyUtil	64%	5/14
Group	56%	20/45
DomainException	33%	6/9
Map	0%	8/8

WebApi:



QuickSplit.WebApi	59%	86/210
QuickSplit.WebApi	76%	10/42
QuickSplit.WebApi.Filters	73%	3/11
QuickSplit.WebApi.Controllers	54%	73/157

Persistence:

QuickSplit.Persistence	52%	58/122
QuickSplit.Persistence	52%	58/122
QuickSplitContext	100%	0/49
ImageRepository	21%	58/73

Cobertura total:

Symbol	Coverage ...	Uncover...
Total	79%	530/2485
Infrastructure	100%	0/6
Test	95%	47/960
QuickSplit.Domain	78%	82/366
QuickSplit.Application	67%	273/833
QuickSplit.WebApi	65%	70/198
QuickSplit.Persistence	52%	58/122

Pruebas de integración

Para las pruebas de integración utilizamos la metodología Big Bang, probándose todos los componentes y módulos del proyecto (incluida la base de datos real) al mismo tiempo. Esta metodología no sólo es sencilla de utilizar sino que también es rápida, lo cual fue un factor indispensable para nuestro desarrollo ya que como mencionado anteriormente el mismo sufrió cambios en lo que al calendario respecta. A su vez, se utilizó una parte del framework ASP.NET Core y XUnit para realizar las pruebas de integración. Esta parte del framework nos permitió simular una request HTTP y verificar sus resultados. Optamos utilizar este sistema sobre un cliente de HTTP real como Postman porque escribir pruebas en C# con estas librerías es más fácil y rápido y además los resultados de simular las llamadas HTTP son los mismos que los resultados reales.

Pruebas de FrontEnd

Las pruebas para esta capa no fueron automatizadas (a excepción de una prueba) sino que fueron llevadas a cabo por parte del equipo de desarrollo al terminar cada historia de usuario. Adicionalmente, la aplicación fue probada en dos instancias distintas por personas ajenas al desarrollo de la misma. Gracias a éstas, se mejoraron aspectos de la experiencia de usuario y diseño de la interfaz.

La prueba sobre la lectura de código QR se hace desde el Front-End ya que como fue mencionado con anterioridad es aquí donde ocurre el procesamiento del código QR.

14. Instructivo de instalación

14.1. Deploy del Back End

Si la aplicación fuese a ser lanzada al mercado, se debería hacer el despliegue en Amazon Web Services como especifica el diagrama de despliegue original (Ver 5.1.6) pero a medida de facilitar el despliegue para este obligatorio estas instrucciones describen un despliegue simplificado que correrá sobre una sola PC.

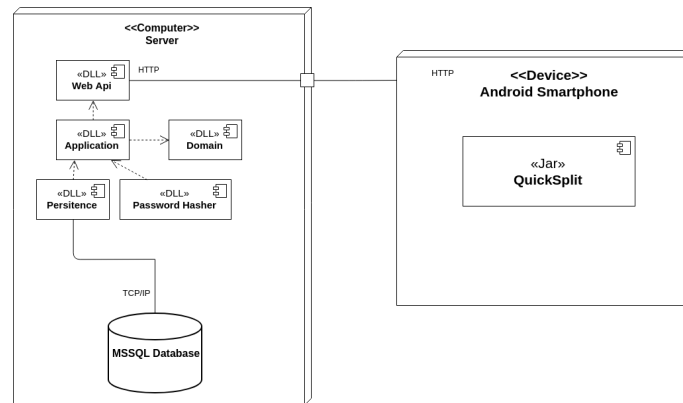


Figura 14.1: Diagrama de despliegue simplificado

14.1.1. Requisitos

1. Se recomienda que el servidor utilice Linux, pero también debería funcionar en Windows (se recomienda bajar el firewall en este caso).
2. El servidor debe tener instalado Microsoft SQL Server.
3. El servidor debe tener instalado el runtime de .Net Core.

14.1.2. Instrucciones

1. Correr el archivo SQL "CrearUsuarioDeBase.sql" incluido con los binarios en la base de datos.
2. Desde la terminal o cmd (en Windows), ejecutar el siguiente comando **dotnet *ruta a el archivo "QuickSplit.WebApi.dll" ubicado en los binarios***.

14.2. Deploy del Front End

Debido a que no contamos con las licencias correspondientes y por el momento no tenemos interés de lanzar la aplicación al mercado, nuestra aplicación no estará disponible en la tienda oficial de Android, Google Play Store. Por otro lado el Back-End se encontrará ejecutándose en una computadora que dependerá de la red a la que esté conectada ésta el número de IP asignado al Back-End, por lo que tampoco es factible generar una APK (ya que no se podría configurar la ruta).

14.2.1. Deploy con Android Studio

1. Clonar el repositorio que se encuentra en <https://bitbucket.org/EusebioDM/obligatorio-ingenieria-en-la-practica/>.
2. Abrir Android Studio, una vez que carguen todos los módulos dirigirse a *File* → *Open...* y seleccionar el directorio donde fue clonado el git luego navegar a *src* y dar clic sobre FrontEnd (Android Studio debería reconocerlo como un proyecto de Android sin necesidad de configuraciones adicionales).
3. Una vez abierto el proyecto, dirigirse a *Build* → *Rebuild Project*, hecho esto debería aparecer una estructura de proyecto similar a la de la figura 14.2.
4. Dirigirse a *app/java/org.quicksplit/ServiceGenerator*, una vez posicionados en esta clase cambiar el valor de la variable *BASE_URL* por la dirección en la que se encuentra el Back-End de la aplicación.
5. Navegar hacia *app/res/xml/network_security_config.xml* y cambiar el valor entre las etiquetas *domain* por la dirección donde se encuentra el Back-End.
6. Lo último que queda es ejecutar la aplicación en un emulador o en un celular físico que cumpla con los requerimientos de la aplicación antes definidos.
7. En Android Studio dar al botón verde de play (figura 14.3) o presionar el shortcut *Mayus + F10* aparecerá una ventana similar a la de la figura 14.4, si existe algún dispositivo físico conectado, el mismo debería aparecer en la lista (El celular debe tener activado el modo desarrollador, activar este modo depende de cada fabricante).
8. Seleccionar el dispositivo y confirmar, la aplicación debería instalarse en el dispositivo físico o virtual existente.

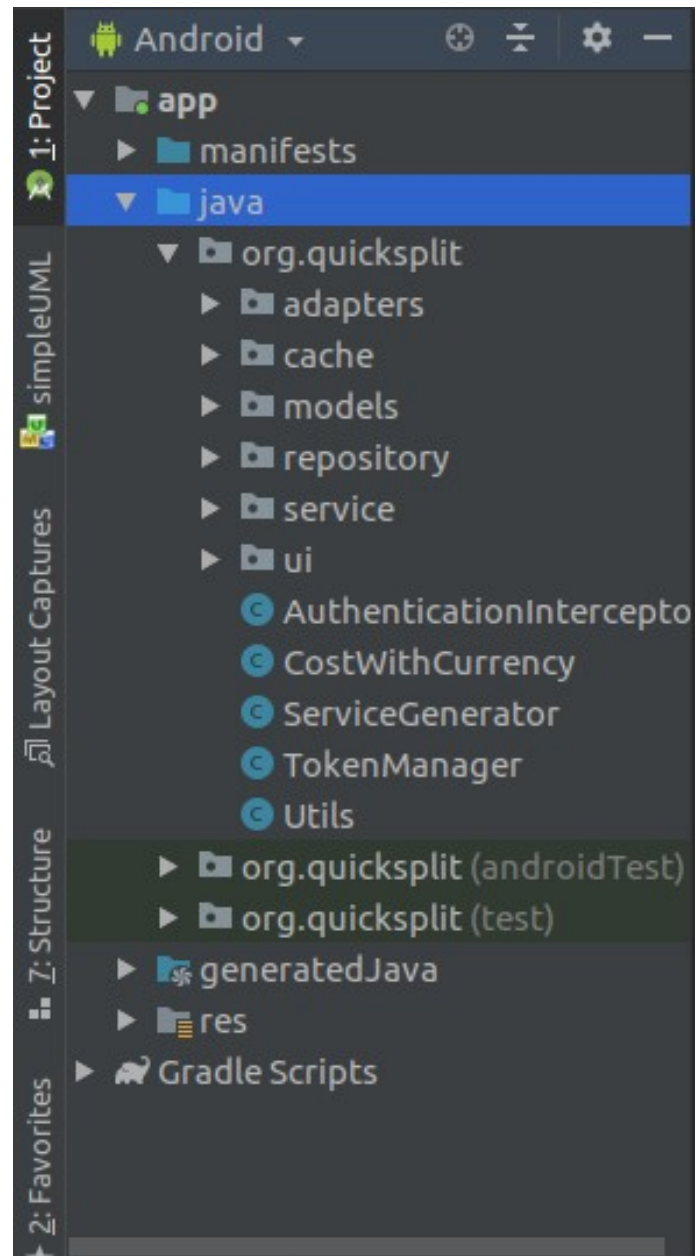


Figura 14.2: Estructura de Directorios del Proyecto



Figura 14.3: Botones para Correr el Proyecto

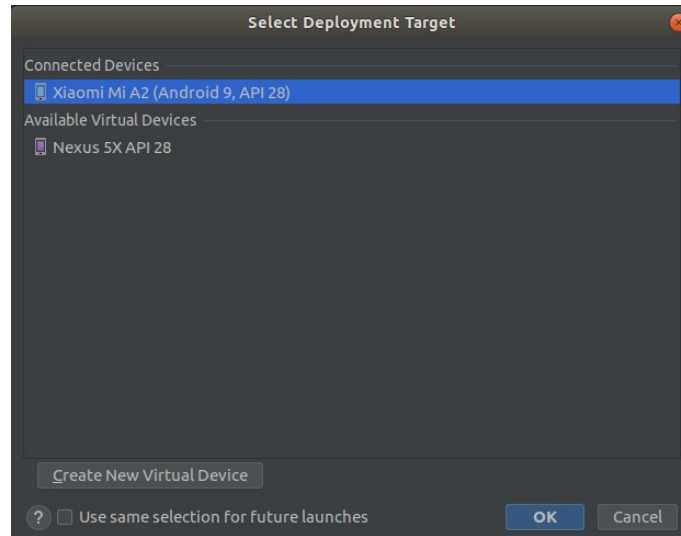


Figura 14.4: Lista de Dispositivos Disponibles