Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería de Software en la Práctica

Matías Morales - 223375
Florencia Mosquera - 193737
Manuel Vilaró - 223777

Docente: `Luis Barragué`

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

17 de febrero de 2021

Índice general

1. Análisis del Problema **3**
- 1.1. Solución Propuesta . . . 3
- 1.2. Repositorios del Proyecto . . . 3
- 1.3. Mejoras y Defectos Conocidos . . . 3
- 1.4. Especificación de Historias de Usuario . . . 4
- 1.5. Imágenes de Interfaz . . . 7

2. Gestión de Proyecto **11**
- 2.1. Release Plan . . . 11
- 2.2. Sprint Planning . . . 11
  - 2.2.1. Sprint 1, comienzo: 8/04/2020 . . . 12
  - 2.2.2. Sprint 2, comienzo: 22/04/2020 . . . 13
  - 2.2.3. Sprint 3, comienzo: 20/05/2020 . . . 14
  - 2.2.4. Sprint 4, comienzo: 04/06/2020 . . . 16
  - 2.2.5. Sprint 5, comienzo: 16/06/2020 . . . 17
- 2.3. Gestión de Riesgos . . . 18
  - 2.3.1. Registro de Riesgos . . . 18
  - 2.3.2. Riesgos durante el transcurso del proyecto . . . 19

3. Gestión de la Configuración **20**
- 3.1. Control de Cambios y de Versiones . . . 20

4. Aseguramiento de Calidad **23**
- 4.1. Proceso y Producto . . . 23
- 4.2. Pruebas de Software . . . 23

5. Arquitectura y Diseño **26**
- 5.1. Backend . . . 26
  - 5.1.1. Principios de Diseño . . . 29
  - 5.1.2. Patrones de Diseño . . . 33
  - 5.1.3. Principios a nivel de paquetes . . . 36
- 5.2. Frontend . . . 37
- 5.3. Distribución de Proyectos . . . 37
  - 5.3.1. Clean Architecture . . . 37
  - 5.3.2. MVVM . . . 38
  - 5.3.3. Otros comentarios . . . 39
- 5.4. Cambios en Diseño y Arquitectura . . . 39

5.4.1. Sprint 0 . . . 39
5.4.2. Sprint 1 . . . 41
5.4.3. Sprint 2 . . . 43
5.5. Evolución en los Sprints 3, 4 y 5 . . . 44
5.6. Modelo de la Base de Datos . . . 45
5.7. Principios de REST API . . . 45

6. Tecnologías Utilizadas **49**
6.1. Funcionalidades del dispositivo . . . 49

7. Implementación **50**
7.1. Clean Code . . . 50
7.1.1. Nombres Significativos. . . . 50
7.1.2. Funciones. . . . 50
7.1.3. Errores. . . . 51
7.1.4. Comentarios. . . . 52
7.1.5. Formato. . . . 52
7.1.6. Objetos y Estructuras de Datos. . . . 53
7.1.7. Manejo de Errores . . . 54
7.1.8. Límites. . . . 55
7.1.9. Clases. . . . 55
7.2. Código . . . 57
7.3. Datos de Prueba . . . 58
7.4. Manual de instalación . . . 59
7.4.1. Frontend . . . 59
7.4.2. Backend . . . 59
Bibliografía . . . 60

1. Análisis del Problema

1.1. Solución Propuesta

La aplicación surge de la necesidad de poder ubicar a contactos de confianza en situaciones donde estos no contestan el teléfono o no responden mensajes. Las alternativas existentes proveen métodos de localización pero requieren de la participación de la persona a la que se quiere localizar para que envíe su localización.

FindMe busca alterar la dinámica existente pudiendo obtener la localización de un contacto amigo aún cuando este no se encuentra disponible. Al iniciar un pedido de localización el usuario solicitante enviará una notificación a su amigo y deberá esperar un tiempo pactado para poder recibir su localización. En el caso de que antes de que expire el tiempo, su amigo responda la notificación, este podrá decidir entre enviar su ubicación o responder un mensaje notificando que se encuentra bien, descartando la necesidad de enviar su ubicación.

El proyecto será llevado a cabo utilizando Android Studio con el lenguaje de programación Kotlin en lo que refiere a Frontend. El Backend será desarrollado en .NET Core utilizando el lenguaje C#. Todo esto, en un marco de desarrollo de ágil como lo es Scrum.

1.2. Repositorios del Proyecto

A continuación se presentan los links a los correspondientes repositorios de Backend y Frontend del Proyecto.

- Backend: `https://github.com/ORTISP/FindMe`
- Frontend: `https://github.com/ORTISP/FindMeApp`

1.3. Mejoras y Defectos Conocidos

- Si se crea un usuario que nunca es verificado, no se elimina. Debería esperarse cierto tiempo y eliminarse dicho usuario.

- Si el token expira en la app (tras 24 hs), primero fallaría y luego una vez que se abriese la aplicación nuevamente, cerraría la sesión. Este es el bug con mayor impacto en la app, pero, por tiempos, no se logró completar la tarea.
- Faltaron realizar mejoras en la UI en la sección del mapa. Los markers en los diseños eran más prometedores de lo que lo son en la actualidad.
- Ciertas acciones donde el body de la request no es válido mandan errores que no son bien manejados. Esto se debe a que el backend responde a la solicitud con un simple string y el frontend espera un JSON. Esto funcionaba bien inicialmente pero en algún momento comenzó a fallar y no dio el tiempo para solucionarlo.

A pesar de que quedaron algunos bugs por arreglar y mejoras a realizar, se llegó a un resultado más que satisfactorio, cumpliendo con todas las historias de usuario planteadas, y una aplicación con buena usabilidad y muchas funcionalidades del dispositivo android en sí como lo son la cámara, la lista de contactos, etc.

Esto se puede ver más en detalle en la sección 6.

1.4. Especificación de Historias de Usuario

Las historias de usuario serán estimadas en Story Points a medida que son seleccionadas para formar parte de los distintos Sprint Backlogs. La escala de valores a utilizar como referencia será la serie de Fibonacci.

1. **Registro de usuario**.
   Como usuario quiero registrarme para poder utilizar las funcionalidades que la aplicación brinda.
   Descripción: Un usuario puede registrarse ingresando su número de celular. Se enviará un mensaje de texto al celular con un código de verificación con el cual el usuario podrá finalizar el registro. Los datos necesarios para el registro, aparte del número de celular ya mencionado, son nombre y apellido.

2. **Inicio de sesión**.
   Como usuario quiero iniciar sesión para poder utilizar las funcionalidades que la aplicación brinda.
   Descripción: Un usuario puede iniciar sesión ingresando su número de celular. Se enviará un mensaje de texto al celular con un código de verificación con el cual el usuario podrá finalizar el inicio de sesión.

3. **Solicitar ubicación**.
   Como usuario registrado quiero enviar una solicitud de ubicación a un amigo para tener la posibilidad de ver dónde se encuentra un amigo.
   Descripción: Enviar una solicitud de ubicación a un amigo.

4. **Agregar amigo mediante número**.
   Como usuario registrado quiero enviar una solicitud de amistad utilizando un número de celular para poder conectarme con mis contactos de confianza.
   Descripción: Enviar una solicitud de amistad a otro usuario registrado utilizando su número de celular.

5. **Agregar amigo mediante código QR**.
   Como usuario registrado quiero enviar una solicitud de amistad utilizando un código QR para poder conectarme con mis contactos de confianza.
   Descripción: Enviar una solicitud de amistad a otro usuario registrado mediante la lectura de un código QR.

6. **Eliminar amigo**.
   Como usuario registrado quiero poder eliminar a un amigo para que pierda la posibilidad de acceder a mi ubicación.
   Descripción: El usuario podrá eliminar cualquiera de sus amistades.

7. **Responder solicitud de amistad**.
   Como usuario registrado quiero responder una solicitud de amistad para poder aceptar o rechazar a un usuario a mi lista de amigos.
   Descripción: El usuario decidirá si aceptar o no al solicitante a su lista de amigos.

8. **Manejo de permisos de ubicación**.
   Como usuario registrado quiero poder activar o desactivar los permisos de ubicación para tener el control sobre cuando la aplicación accede a mi ubicación.
   Descripción: El usuario podrá activar o desactivar los permisos de ubicación en todo momento.

9. **Configuración de tiempo de expiración entre amistades**.
   Como usuario registrado quiero poder configurar el tiempo en el que cada amigo puede acceder a mi ubicación para poder personalizar los tiempos de respuesta automática.
   Descripción: El usuario podrá configurar el tiempo en el que un amigo puede acceder a su ubicación automáticamente si no hay respuesta previa.

10. **Actualización de información personal**.
    Como usuario registrado quiero actualizar la información de mi perfil para poder personalizar mi experiencia en la aplicación .
    Descripción: El usuario podrá actualizar la información de su perfil en cualquier momento.

11. **Actualización de foto de perfil**.
    Como usuario registrado quiero poder actualiza la foto en mi perfil para poder personalizar mi experiencia en la aplicación.
    Descripción: El usuario podrá actualizar su foto de perfil a la que acceden sus amigos en todo momento, accediendo desde la galería o tomando una foto.

12. **Listado de Amistades**.
    Como usuario registrado quiero poder ver a todas mis amistades para poder interactuar con ellas.
    Descripción: El usuario podrá acceder a un listado de sus amigos y de allí interactuar a gusto con ellos.

13. **Envío de ubicación**.
    Como usuario registrado quiero poder enviar mi ubicación a un amigo para poder notificarle donde me encuentro actualmente.
    Descripción: El usuario podrá enviar su ubicación a cualquiera de sus amigos en todo momento.

14. **Vista de permisos**.
    Como usuario registrado quiero poder ver si mis amigos tienen los permisos de ubicación activados para poder saber si es posible acceder a su ubicación actual.
    Descripción: El usuario podrá ver si cada uno de sus amigos tiene los permisos de ubicación activados.

15. **Mensaje de invitación**.
    Como usuario registrado quiero poder invitar a otras personas a que se descarguen la aplicación para poder ampliar mi red de contactos.
    Descripción: El usuario podrá enviar un mensaje de invitación a cualquier número de celular.

16. **Actualización de ubicación**.
    Como usuario registrado quiero poder respaldar mi ubicación cada cinco minutos para que mis amistades puedan acceder a la última ubicación conocida en caso de perder la conexión.
    Descripción: El usuario podrá respaldar su ubicación automáticamente cada cinco minutos.

17. **Ver solicitudes de amistad**.
    Como usuario registrado quiero poder visualizar las solicitudes de amistad que se me han enviado.
    Descripción: El usuario podrá visualizar las solicitudes en una lista.

18. **Leer el código de verificación**.
    Como usuario registrado quiero poder leer automáticamente el código de verificación enviado por sms.
    Descripción: El usuario podrá elegir autocompletar el campo del código de verificación.

19. **Ver información personal**.
    Como usuario registrado quiero ver la información de mi perfil.
    Descripción: El usuario podrá ver la información de su perfil en cualquier momento.

20. **Recibir notificaciones**.
    Como usuario registrado quiero poder recibir notificaciones.
    Descripción: El usuario podrá recibir notificaciones y ver la lista de notificaciones en todo momento. Los tipos de notificaciones serían solicitudes de amistad y solicitudes de ubicación, también se notificará cuando las mismas son aceptadas y declinadas (esto último únicamente en el caso de las solicitudes de ubicación).

21. **Cerrar sesión**.
    Como usuario registrado quiero poder cerrar mi sesión.
    Descripción: El usuario podrá cerrar su sesión en cualquier momento.

22. **Iniciar sesión con Google**.
    Como usuario registrado quiero poder iniciar sesión con mi cuenta de Google.
    Descripción: El usuario podrá iniciar sesión con su cuenta de Goolge brindando su número de teléfono.

23. **Iniciar sesión con huella**.
    Como usuario registrado quiero poder iniciar sesión con mi huella si mi dispositivo lo permite.
    Descripción: El usuario podrá iniciar sesión con su huella dactilar una vez esté registrado.

1.5. Imágenes de Interfaz

A continuación se muestran imágenes de la aplicación desarrollada una vez finalizado el proyecto.

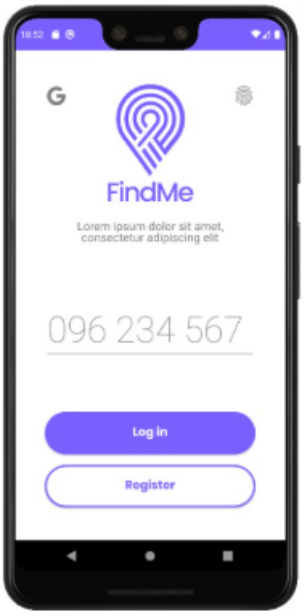

Figura 1.1: Pantalla de Log in.

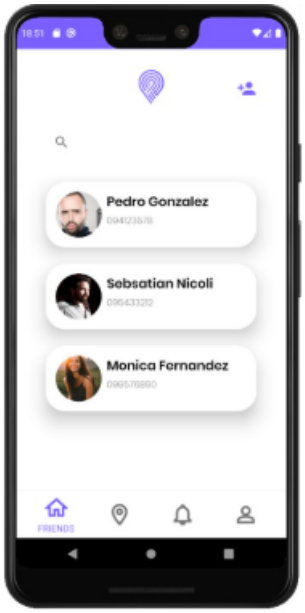

Figura 1.2: Pantalla de Inicio.

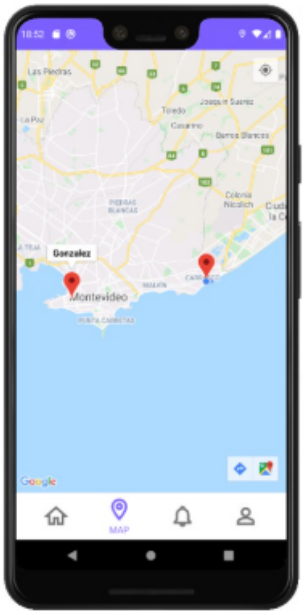

Figura 1.3: Pantalla del Mapa.

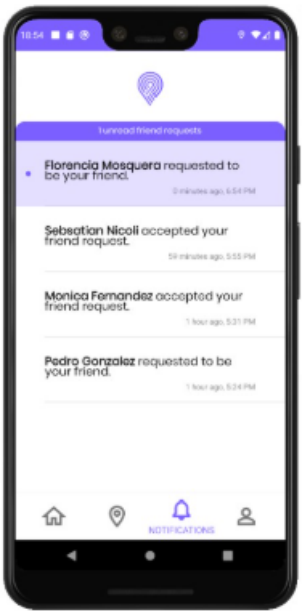

Figura 1.4: Pantalla de Notificaciones.

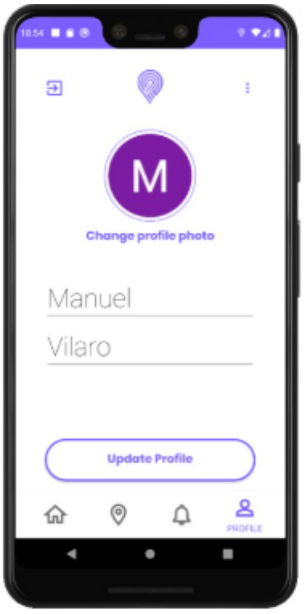

Figura 1.5: Pantalla de perfil del usuario.

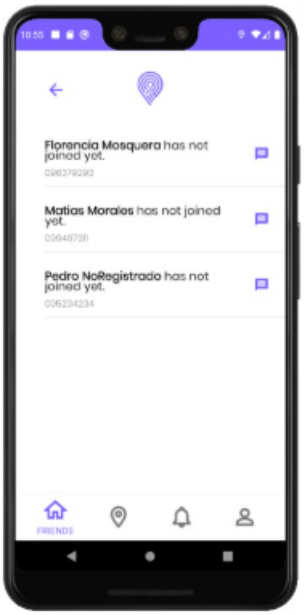

Figura 1.6: Pantalla de ver contactos.

2. Gestión de Proyecto

2.1. Release Plan

El plan de releases se vio modificado durante el transcurso del proyecto, ya que incluso se agregó una iteración, se obtuvo como resultado:

Release	Contenido	Fecha de inicio	Fecha de finalización	Epicas
1	Iteraciones 1 a 3	5/4/2020	8/5/2020	Autenticación de usuarios Manejo de usuarios y amistades
2	Iteraciones 3 a 5	9/5/2020	21/6/2020	Implementación de interfaz Administrar ubicaciones de usuarios, invitaciones y funcionalidades del dispositivo

Figura 2.1: Tabla de contenido de Releases.

2.2. Sprint Planning

Durante la planificación de los sprints en principio se estableció una duración de dos semanas cada sprint, con releases de dos sprints cada uno.

Sin embargo, durante el transcurso del proyecto se decidió realizar planificaciones para los sprints de tipo Commitment Driven, y se agregó una iteración extra a lo pactado en un comienzo. Esto, también afectó el plan de releases del proyecto, sobre el cual se decidió agregar esa iteración final al segundo release planificado.

A continuación se muestra evidencia de la estimación de las historias de usuario para este proyecto:

2.2.1. Sprint 1, comienzo: 8/04/2020

- As a logged user I should be able to activate or deactivate location permissions. (3 SP).
- As a logged user I should be able to confirm or deny a location request. (3 SP).
- As a user i should be able to choose the time before automatically sending my location to a friend (3 SP).
- As a logged user I should be able to update a profile picture (13 SP).
- As a logged user I should be able to view my friends list. (3 SP).
- As a logged user I should be able to send a friend request. (5 SP).
- As a logged user I should be able to send a location request. (3 SP).
- As a logged user I should be able to see if my contacts activated location permissions for the app. (5 SP).
- As a logged user I should be able send a text message with an invitation to join the app. (8 SP).
- As a user I should be able to receieve an sms with confirmation code provided by the application on registration and login (21 SP).
- As a logged user I should be able to send my location to a contact. (5 SP).
- As a logged user my location should be fetched every 5 minutes (8 SP).
- As a user I should be able to register (5 SP).
- As a logged user I should be able to delete an existing friend. (3 SP).
- As a user I should be able to login (13 SP).
- As a logged user I should be able to edit my profile. (3 SP).
- As a logged user I should be able to accept or deny a friend request. (3 SP).

Sprint Burndown Chart

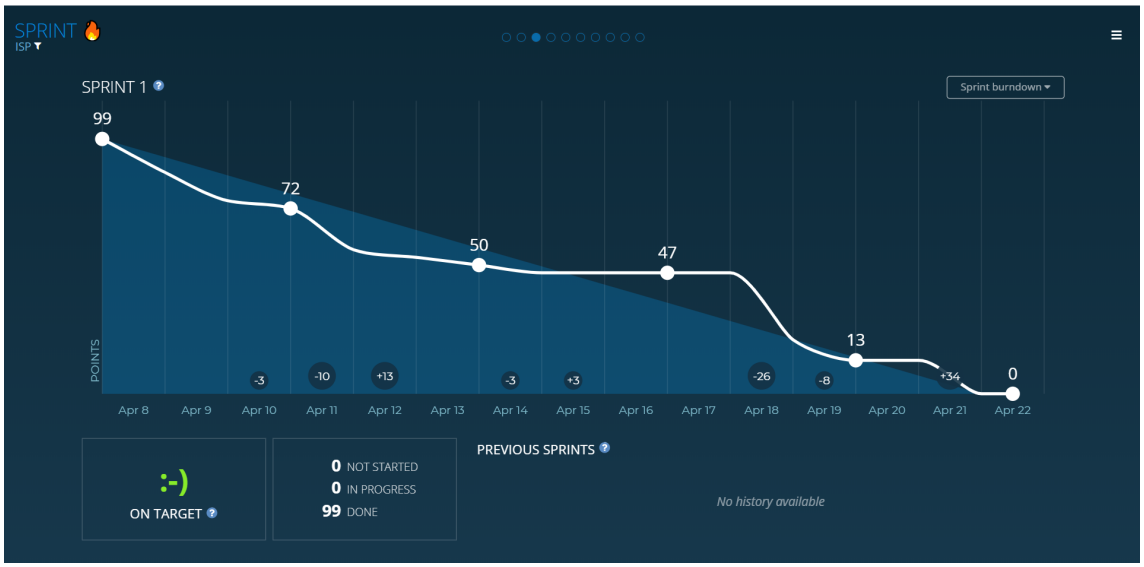

Figura 2.2: Sprint burndown chart correspondiente al Sprint 1.

2.2.2. Sprint 2, comienzo: 22/04/2020

Dados los resultados del Sprint Burndown Chart (1), con una velocidad del equipo de 99 SP, para el sprint backlog de la iteración 2 se planificaron 94 SP. Hubo intención de no superar los 95 SP debido a que se tuvo en cuenta que la velocidad del equipo sería afectada por el tiempo invertido en el desarrollo del presente documento.

- As a user i should be able to login and get a text message with a PIN. The PIN should be read from the messages. (21 SP).
- As a user I should be able to register (21 SP).
- As a logged user I should receive a location when the request response time has expired (5 SP).
- As a logged user i should be able to logout. (5 SP).
- As a logged user I should be able to view my friends list. (8 SP).
- As a logged user i should be able to send a friend request. (13 SP).
- As a logged user I should be able to view my friend requests. (13 SP).
- As a logged user i should be able to accept or deny a friend request. (8 SP).

Sprint Burndown Chart

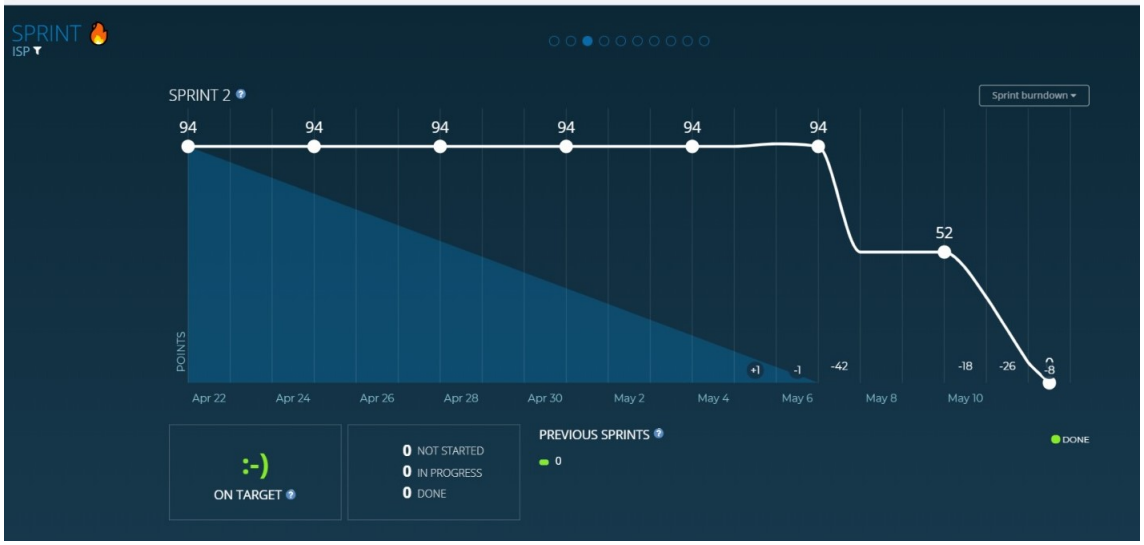

Figura 2.3: Sprint burndown chart correspondiente al Sprint 2.

2.2.3. Sprint 3, comienzo: 20/05/2020

- As a logged user I should be able to add a user with QR code. (13 SP).
- As a user, I should be able to choose the location sharing time for each contact. (21 SP).
- As a logged user I should be able to view my friend's locations. (21 SP).
- As a logged user I should be able to send a location request. (5 SP).
- As a logged user I should be able to see if my contacts activated location permissions for the app. (5 SP).
- As a logged user I should be able to select a profile picture from the gallery or take a picture .(8 SP).
- As a logged user I should be able to activate or deactivate location permissions. (13 SP).
- As a logged user I should be able to delete an existing friend. (5 SP).
- As a logged user i should be able to send my location to a contact. (13 SP).

Sprint Burndown Chart

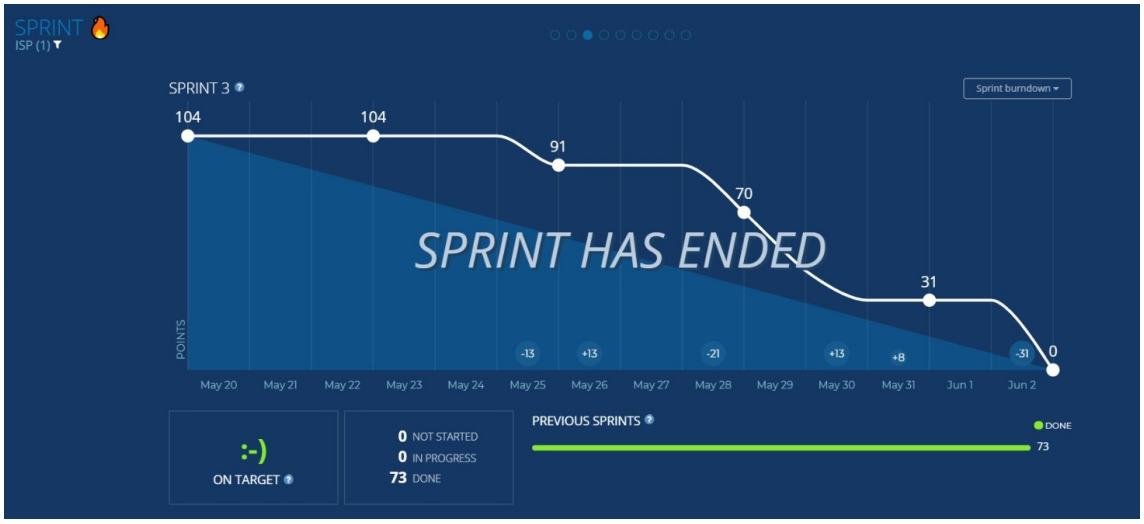

Figura 2.4: Sprint burndown chart correspondiente al Sprint 3.

Rendimiento del equipo desde el 13/04/2020 hasta 02/06/2020

Figura 2.5: Resultados de rendimiento del equipo. (RELEASE 1)

En la imagen se pueden apreciar valores relevantes sobre el rendimiento del equipo en dicho período, estos valores han sido útiles durante las ceremonias de:

- Sprint Review, al final de cada sprint en base a los datos proporcionados se identificaron algunas prácticas que podrían mejorarse por parte del equipo.
- Sprint Planning, el las siguientes ceremonias de planeación se tuvieron en cuenta estos datos como referencia para fijar fechas de finalización.

Es por tanto que con estos resultados y el intercambio de ideas entre los miembros del equipo de desarrollo, se tomó la decisión de planificar los siguientes Sprints

con la modalidad de Commitment Driven.

2.2.4. Sprint 4, comienzo: 04/06/2020

- As a logged user I should be able to recieve notifications. (34 SP).
- As a logged user I should be able to update my location every 5 minutes.(3 SP).
- As a logged user I should receive a location when the request response time has expired. (13 SP).
- As a logged user I should recieve a location when the request response time has expired. (21 SP).
- Notifications list frontend. (8 SP).
- Notifications list backend (8 SP).
- As a logged user I should be able to accept or deny a friend request. (8 SP).
- As a logged user I should be able to view my friend requests. (13 SP).

Sprint Burndown Chart

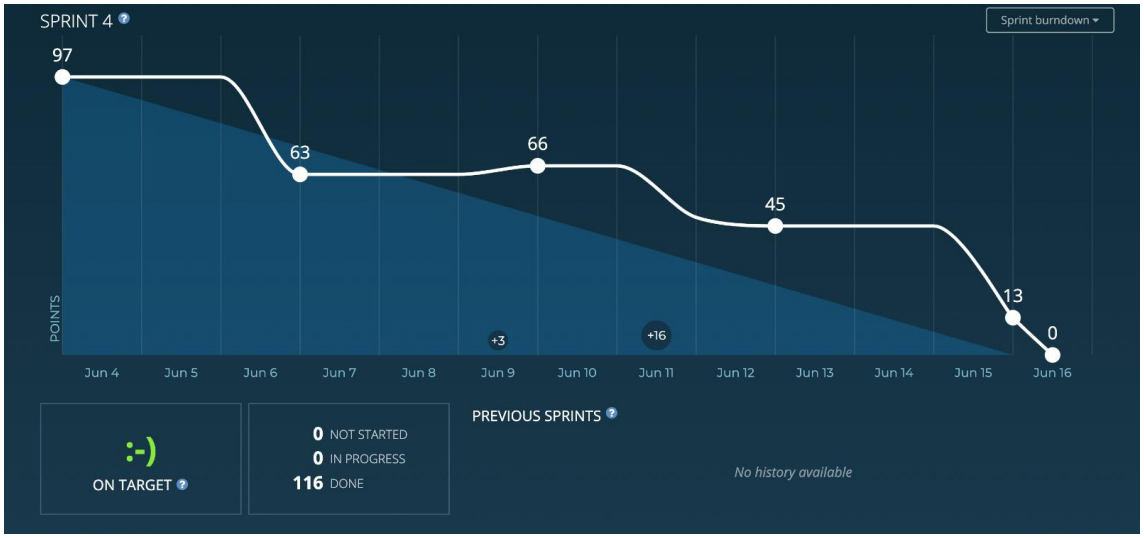

Figura 2.6: Sprint burndown chart correspondiente al Sprint 4.

Durante la Sprint Retrospective algunas de las cosas que surgieron como mejoras fueron: el hecho de fomentar el feedback constante entre miembros del equipo (esto surge a partir de notar diferencias en las estrategias de resolución de ciertas funcionalidades, las mismas podrían haberse encarado de otra manera), y también se

planteó establecer ciertos objetivos a corto plazo dentro del sprint (en algunos casos con límites de fechas), para poder al final de cada implementación de funcionalidad realizar una serie de tests exhaustivos en las mismas. Esto último surge a partir de que se notó que a medida que la aplicación crecía, se fueron generando varios bugs descubiertos en el siguiente sprint por ejemplo.

2.2.5. Sprint 5, comienzo: 16/06/2020

- Login with google backend. (13 SP).
- Login with google frontend (21 SP).
- As a logged user I should be able to view the contacts I have that have joined the app and send them a contact request. (5 SP).
- As a logged user I should be able to view the contacts I have that haven't joined the app and send them a text message with an invitation to join the app. (1 SP).
- Seen/Not seen Notifications (13 SP).
- As a logged user I should be able to view the contacts I have that have joined the app and send them a contact request. (13 SP).
- As a logged user I should be able to view the contacts I have that haven't joined the app and send them a text message with an invitation to join the app. (13 SP).
- As a logged user I should be able to login with fingerprint, if my device allows it. (13 SP).

Sprint Burndown Chart

Figura 2.7: Sprint burndown chart correspondiente al Sprint 5.

Rendimiento del equipo final.

Figura 2.8: Resultados de rendimiento del equipo. (RELEASE 2)

Por último en la ceremonia de Sprint Retrospective final, el equipo demostró que se siente muy a gusto del alcance logrado en estas cinco iteraciones del proyecto, sobre todo se expresó gran satisfacción respecto al diseño de la UI.

2.3. Gestión de Riesgos

2.3.1. Registro de Riesgos

En la siguiente tabla fueron registrados distintos riesgos que en su momento se consideraron necesarios considerar para el desarrollo del proyecto. Se estimó su probabilidad de ocurrencia y la magnitud de pérdida que ocasionarían. De allí se calculó la exposición al riesgo.

Riesgo	Probabilidad de ocurrencia	Magnitud de la pérdida (días)	Exposición al riesgo (días)
Errores en la estimación del esfuerzo	50%	5	2,50
Algún integrante del equipo se enferma y no puede continuar trabajando por un periodo de tiempo	15%	7	1,05
Cambian las condiciones de cuarentena actuales y se modifica la rutina diaria	30%	4	1,20
Problemas con el uso de Android Studio	40%	3	1,20
Algún integrante del equipo abandona el curso	5%	10	0,50

Figura 2.9: Tabla de registro de riesgos.

2.3.2. Riesgos durante el transcurso del proyecto

Durante el transcurso del proyecto afortunadamente no ocurrió ningún riesgo fuera de los considerados en la anterior evaluación.
Sin embargo uno de estos riesgos llegó a ocurrir, ya que un miembro del equipo (Matías Morales), se encontró con problemas de conexión con su dispositivo Android al ejecutar la aplicación en Android Studio. La estrategia acordada para este riesgo fue la de mitigación , pero una vez ocurrido se buscó por diversos medios una solución hasta poder resolver el problema.

Concretamente el problema consistía que al parecer al correr la aplicación en el dispositivo, la misma no lograba conectar con el servicio backend. La solución encontrada fue deshabilitar el firewall del ordenador y cambiar el puerto donde se expone el servicio backend.

3. Gestión de la Configuración

3.1. Control de Cambios y de Versiones

Repositorio

Para el control de versiones del proyecto se decidió utilizar un repositorio dentro de la organización del curso para la API REST y otro para el Frontend, a los que tendrán acceso todos los miembros del equipo y el docente.

GitFlow

Se decidió establecer un GitFlow para el desarrollo con la finalidad de estandarizar el uso y nomenclatura de las ramas en Git.

La estructura de los repositorios consiste en una rama master (en la cual se incluirán las distintas versiones de cada release), una rama staging (donde se tendrán versiones estables luego de finalizar cada sprint), y una rama dev (la cual es utilizada para llevar las distintas versiones de desarrollo del sistema, de manera que al finalizar una funcionalidad, la misma será incluida en esta rama).

Las ramas deben ser nombradas con un prefijo definido según el tipo de acción que cumple, y deberán tener al final el número de la tarjeta de Trello a la que hacen referencia para que, en caso que más adelante surja un bug, se pueda volver a la especificación del requerimiento y ver la causa de este. Se nombran de la siguiente manera:

- Feature: ft/nombre_de_funcionalidad#nro_tarjeta
- Release: rl/descripcion_de_release#nro_tarjeta
- Hotfix: ft/descripcion_de_bug#nro_tarjeta

Pull Requests

Además se decidió trabajar con la modalidad de *Pull Request Policy*, esto básicamente consiste en que una vez finalizada la implementación de un cambio en una rama, se realiza su correspondiente Pull Request (PR), el cuál deberá ser aprobado por al menos dos de los otros integrantes del equipo para poder incluir dicho cambio en dev.

Para asegurar que esto se cumpliera, se bloquearon las ramas master, staging y dev a todos los miembros para que no se pueda subir el código sin ser mediante un Pull Request. A su vez, dicho PR no puede ser incluido en dev hasta que los otros dos integrantes del equipo lo acepten.

Esta política fomenta el Clean Code ya que cada integrante debe verificar que se estén siguiendo las prácticas definidas por el equipo. Además, ayuda a que todos los integrantes conozcan la solución más en detalle, y que puedan opinar sobre posibles cambios del diseño o arquitectura propuestos.

Evidencias (Pull Request Policy)

Las siguientes imágenes constituyen la evidencia de la incorporación de Pull Request Policy por parte del equipo.

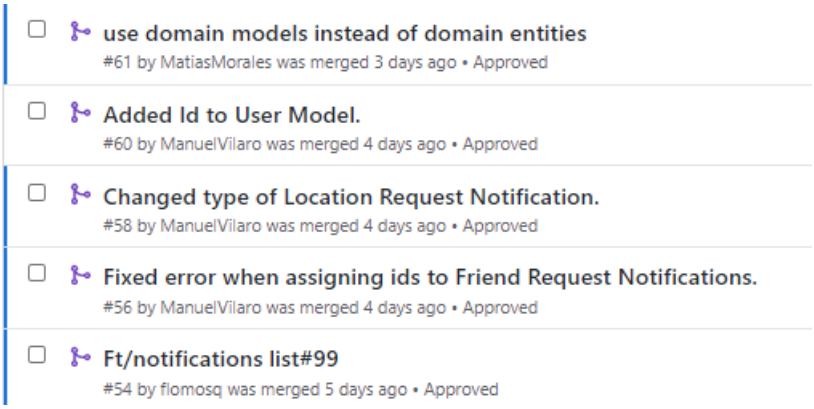

Figura 3.1: Pull Requests en el repositorio de backend.

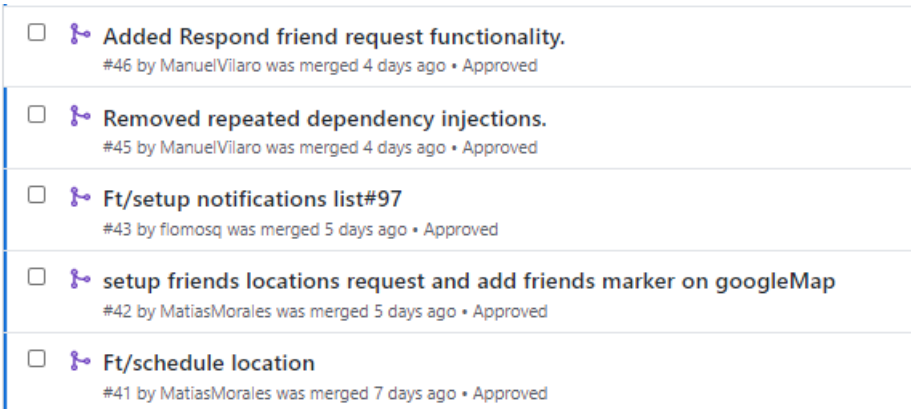

Figura 3.2: Pull Requests en el repositorio de frontend.

Modalidad de los Releases

Al final de cada release se buscará tener un MVP, el mismo será incluido como se ha mencionado anteriormente en la rama master del repositorio, siendo liberado

para producción. Mientras que para el final de cada sprint se planea mantener una versión estable en desarrollo del sistema.

Gtihub Issues

Una de las mejoras que se planteó en el equipo y se incorporó fue el uso de los Github Issues. Para cada uno de estos se realiza un Pull Request, de forma que el Issue queda marcado como resuelto una vez se mergee el Pull Request a dev. A continuación se presenta evidencia de la herramienta incorporada:

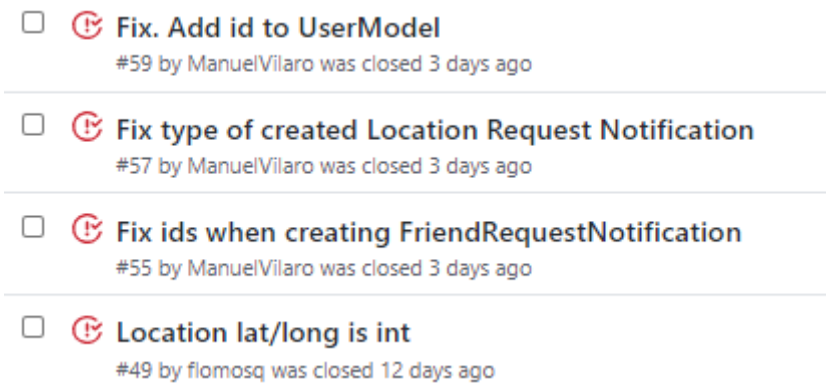

Figura 3.3: Issues de Github.

4. Aseguramiento de Calidad

4.1. Proceso y Producto

Como medidas para mejorar la calidad del producto, se ha decidido incorporar al proceso de desarrollo, el cumplimiento de los lineamientos propuestos para el desarrollo sobre calidad de código por Robert C. Martin en su libro: [2]
También se incorporó el uso de una herramienta (dotnet-format) que permite automáticamente (en base a una serie de estándares definidos en un archivo), dar formato a nuestro código para cumplir con los mismos cada vez que se crea un commit.
Como se ha mencionado anteriormente, una práctica utilizada para el aseguramiento de la calidad del producto es la llamada Pull Request Policy, que permite a todos los integrantes del equipo de desarrollo tener un seguimiento activo (y sobretodo fomentar la retroalimentación del equipo) y participación en cada una de las nuevas funcionalidades incluidas en el incremento del producto, ya sea como autor o reviewer.

4.2. Pruebas de Software

Respecto a las pruebas de aceptación: Las mismas se han llevado a cabo a medida que se produjeron incrementos en el producto que reflejen funcionalidades completas. Esto es, para cada una de las funcionalidades llevadas a cabo al momento de ser incluidas en el proyecto, se realizaron sus correspondientes pruebas de aceptación donde fueron partícipes el/los autores de dicha funcionalidad y al menos un integrante extra.

A continuación se presenta evidencia de los paquetes de test implementados en la estructura del proyecto:

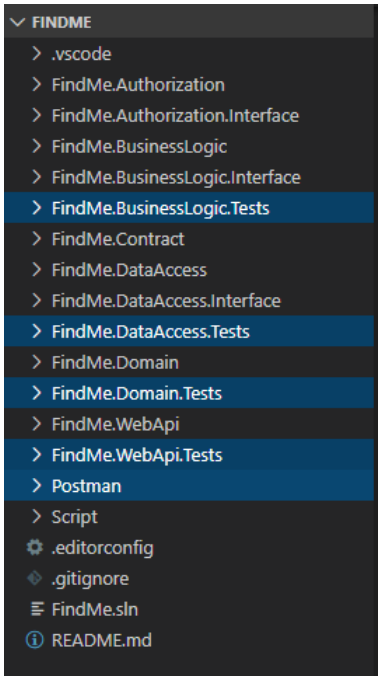

Figura 4.1: Paquetes de test contenedores de clases de test correspondientes a la unidad lógica asociada al paquete.

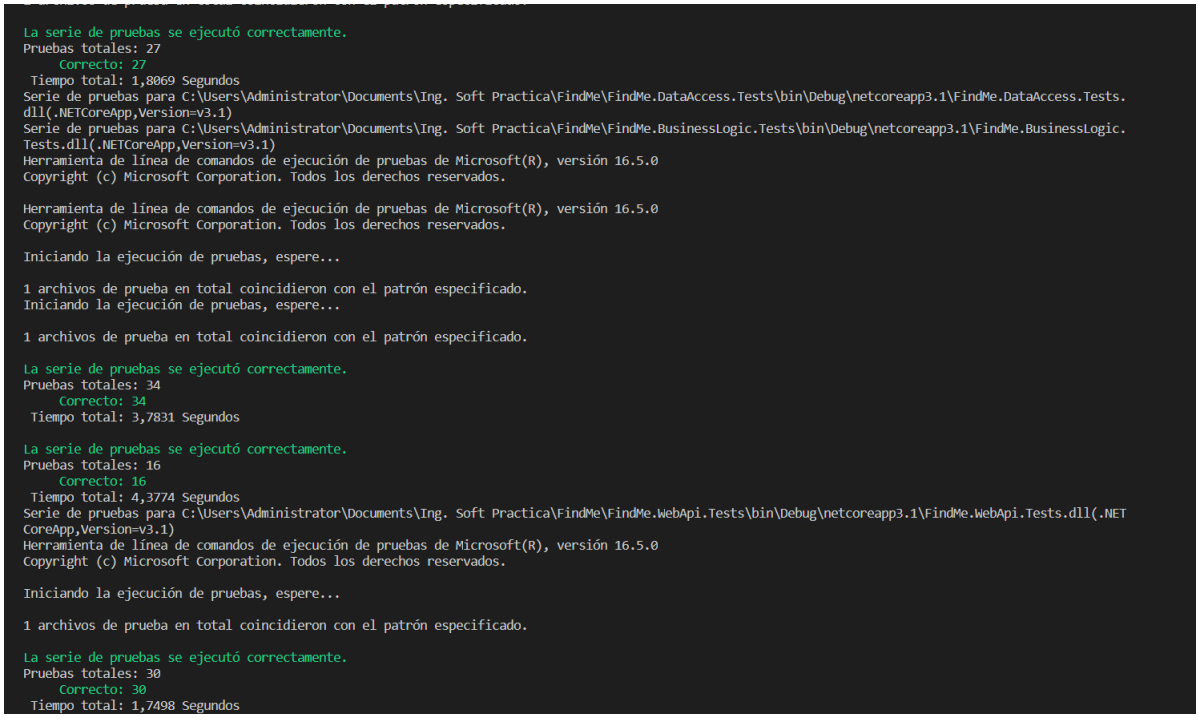

Figura 4.3: Ejecución de tests unitarios al finalizar el sprint 1.

A continuación se muestra evidencia de la ejecución de cobertura de pruebas para la API REST, en módulos que contienen lógica de negocio.

Se puede observar que en general los resultados de los tests de cobertura de las pruebas no son muy buenos. Esto se debe a una falta de tiempo y al hecho de que

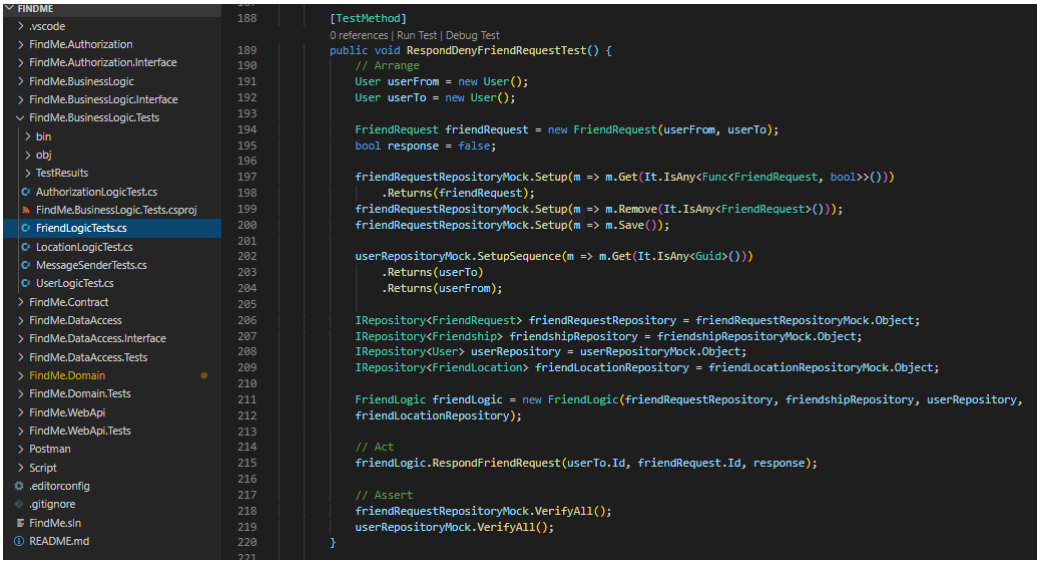

Figura 4.2: Ejemplo de método de test unitario dentro de la clase FriendLogicTests.cs

el equipo haya priorizado el alcance frente a este atributo de calidad. De hecho en el product backlog, en el tablero del equipo quedó una tarea de mejorar la cobertura de las pruebas pendiente, que no ha podido llevarse a cabo para este sprint por falta de tiempo.

Module	Line	Branch	Method
WebApi	45.63%	30.0%	52.87%
Domain	52.28%	40%	56.96%
DataAccess	12.1%	100%	46.66%
BusinessLogic	88.27%	65.21%	85.71%

Figura 4.4: Ejecución de cobertura de pruebas.

5. Arquitectura y Diseño

5.1. Backend

Para la solución del backend se optó por una arquitectura cliente-servidor en capas bien definidas según su rol.

- Primero se encuentra un paquete *Domain* que es el que contiene los modelos de los datos que se manejan en la aplicación. Estos contienen muy poca lógica y su objetivo es representar los datos a manejar.
- Por otra parte, el paquete *DataAccess* que es el encargado de acceder a la base de datos por información, y proveerla a la lógica.
- El paquete *BusinessLogic* es el responsable de la lógica de negocio de la aplicación.
- El paquete *Authorization* es quien se encarga de toda la autenticación (decodificación y codificación de tokens).
- El paquete *WebApi* se encarga de responder las peticiones de los clientes.
- El paquete *Notifications* se encarga del manejo de notificaciones y conexión a firebase.
- El paquete *Notifications* se encarga del envío de mensajes de texto a través de Twilio.
- El paquete *Utils* simplemente contiene datos predefinidos, como por ejemplo, la foto inicial para cada nuevo usuario.

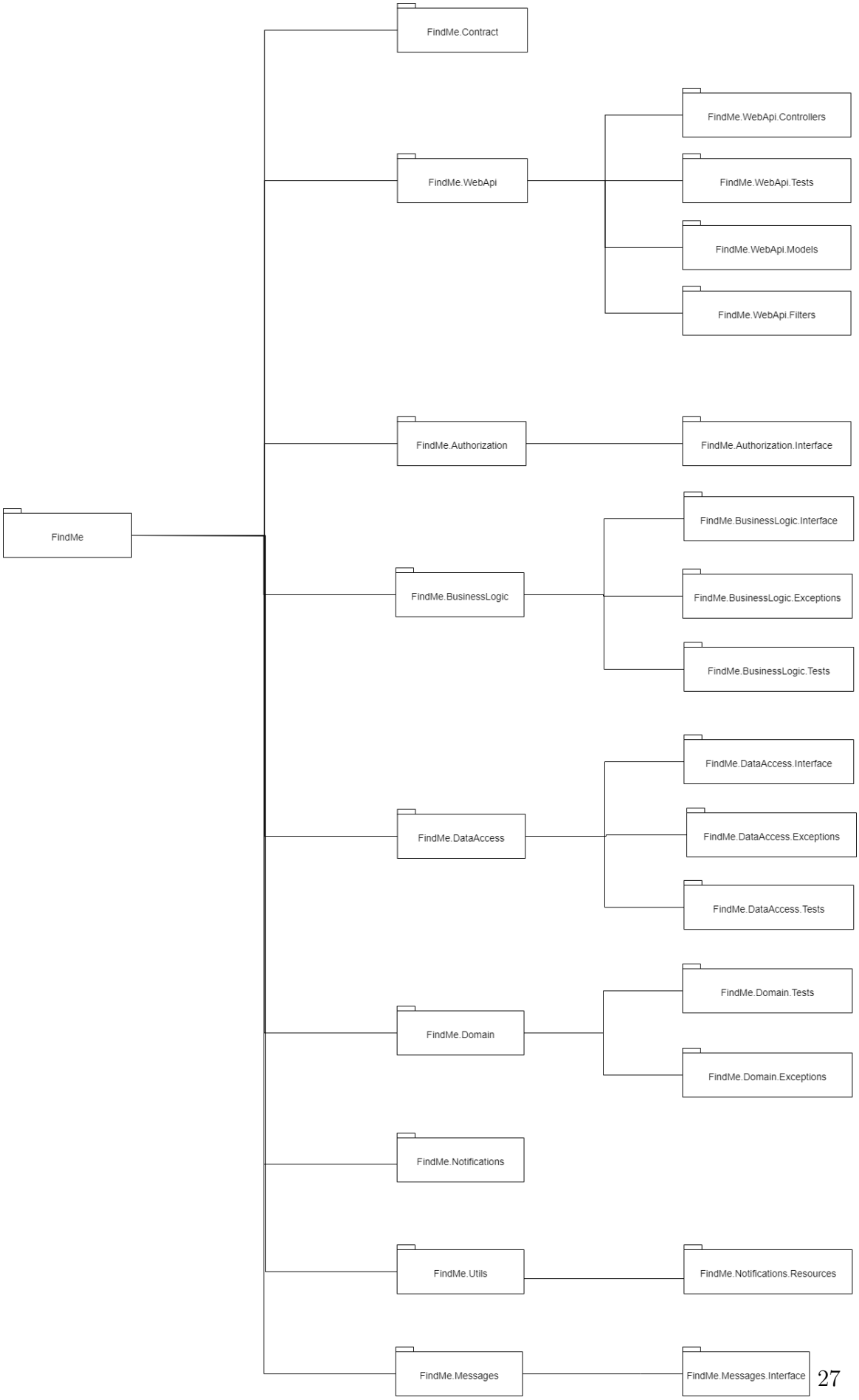

Figura 5.1: Diagrama de Descomposición de los namespaces.

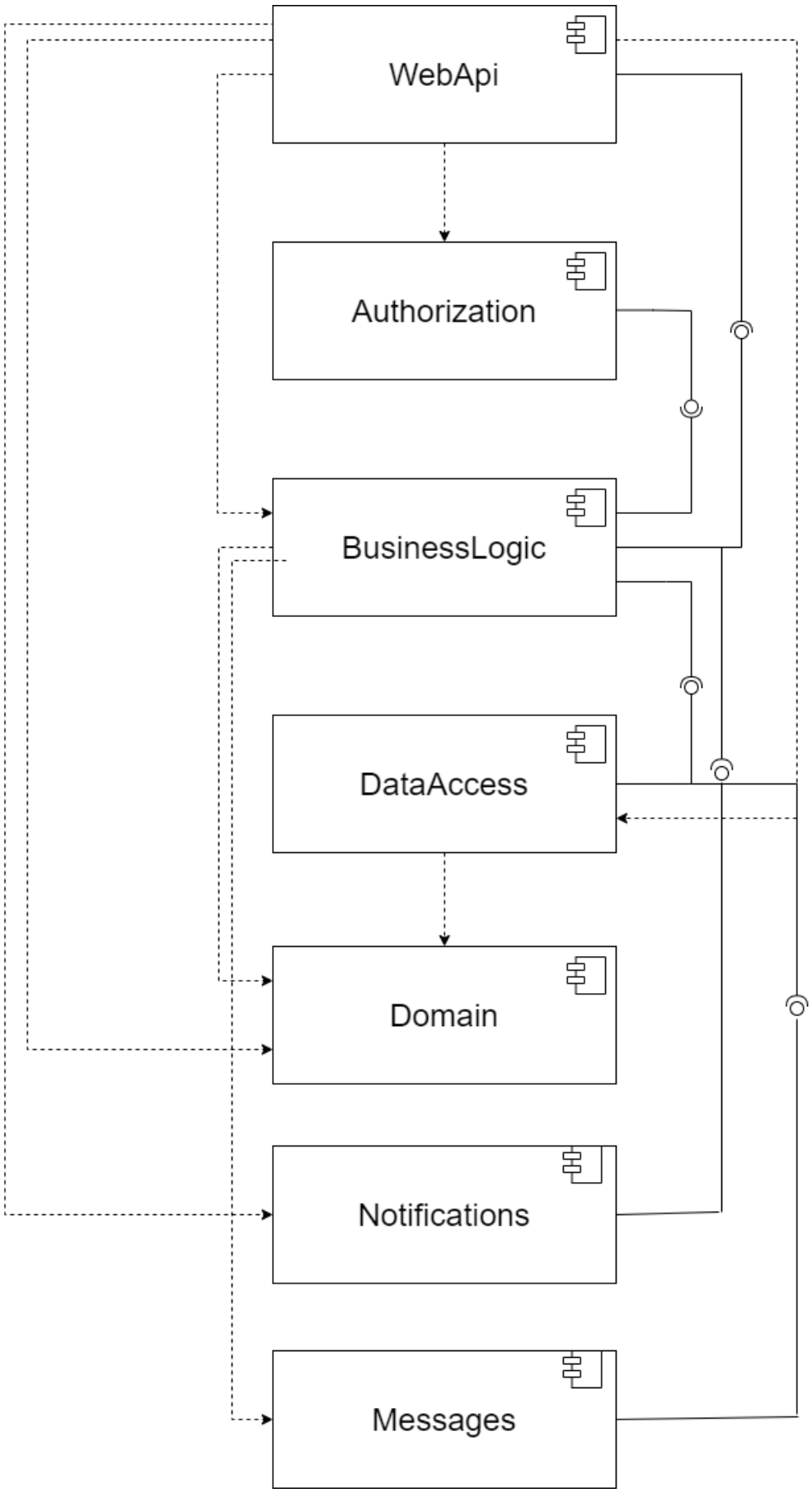

Figura 5.2: Diagrama de Componentes.

5.1.1. Principios de Diseño

Single Responsibility Principle (SRP)

> Una clase solo debe tener una razón por la cual cambiar. Si una clase tiene más de una responsabilidad tiende a tener baja cohesión.

Un ejemplo claro de esto son las clases *Code* y *Phone*. Estas podrían estar perfectamente dentro de *User*, pero no se creyó conveniente darle al usuario, aparte de sus responsabilidades actuales, la responsabilidad de generar un código de verificación y de validar un número de celular.

```
public class Code {
    1 reference
    private readonly int CODE_LENGTH = 4;

    1 reference
    public Guid Id { get; private set; }
    11 references
    public string Value { get; private set; }
    7 references
    public bool Expired { get; set; }

    7 references
    public Code() {
        this.Id = Guid.NewGuid();
        this.Generate();
    }

    8 references
    public void Generate() {
        int number = new Random().Next(0, 9999);
        this.Value = number.ToString("D" + CODE_LENGTH);
        this.Expired = false;
    }
}
```

Figura 5.3: Clase Code.

```
public class Phone {
    1 reference
    private readonly string PHONE_NUMBER_REGEX =
        @"([\+]\d{2,3})(\d{1,2})+(\d{3,3}){2}";

    2 references
    private string phoneNumber;

    2 references
    public Guid Id { get; set; }
    18 references
    public string PhoneNumber {
        get { return phoneNumber; }
        set {
            if (String.IsNullOrWhiteSpace(value)) {
                throw new NullOrEmptyException("Phone Number");
            }

            if (!IsPhoneNumberValid(value)) {
                throw new InvalidPhoneNumberException();
            }

            phoneNumber = value;
        }
    }

    1 reference
    public Phone() {
        this.Id = Guid.NewGuid();
    }

    9 references
    public Phone(string phoneNumber) {
        this.Id = Guid.NewGuid();
        this.PhoneNumber = phoneNumber;      You, 25 days ago • [Domain] Refacto
    }

    1 reference
    public bool IsPhoneNumberValid(string phoneNumber) {
        Regex matchesPhoneNumber = new Regex(PHONE_NUMBER_REGEX);

        return matchesPhoneNumber.IsMatch(phoneNumber);
    }
```

Figura 5.4: Clase Phone.

Open/Closed Principle (OCP)

Las entidades de software deben ser abiertas para la extensión y cerradas para la modificación.

- **Abierto**: Se debe poder cambiar el comportamiento de un módulo

ante cambios en la aplicación o para satisfacer nuevos requerimientos.

- **Cerrado**: El código del módulo no debe cambiar debido a que otros módulos lo pueden estar utilizando.

Es bien sabido que el uso de interfaces favorece este principio, más que nada de la mano de la inyección de dependencias. Gracias a la arquitectura y el diseño de la aplicación, sus módulos son, en su gran mayoría, abiertos a la extensión y cerrados a la modificación.

Un ejemplo particular de esto puede ser la generación de tokens, para la cual se está utilizando una librería de JWT (JSON Web Tokens). Esta puede ser fácilmente intercambiable por otra sin necesidad de modificar la lógica de autenticación, cumpliendo con el principio en cuestión. Ver Figura 5.9.

Liskov Substitution Principle (LSP)

> Método que utilizan referencias a clases base deben poder utilizar sin saberlo objetos de clases derivadas.
>
> Este principio asegura que un método desconoce las subclases de las clases que referencia y por lo tanto no se puede ver afectado por las derivaciones de la clase. Se apoya en el concepto de tipo y subtipo. Es decir, cuando se hereda de una clase base, debemos ser capaces de sustituir las subclases por la clase base sin que suceda ningún tipo de problema.

Como se muestra en la Figura 5.10, se utilizó polimorfismo para los repositorios. De esta forma nosotros desconocemos qué clase concreta es la que se está utilizando y conocemos únicamente la clase abstracta. Este caso particular no el ejemplo más claro de este principio ya que en realidad desde la lógica conocemos interfaces y la clase padre *BaseRepository*. Pero es claro el punto que se quiere hacer, y de no utilizar la interfaz se podría hacer uso de dicha clase padre sin afectar en ningún momento el flujo de la aplicación.

Interface Segregation Principle (ISP)

> Los clientes no deben ser forzados a depender de interfaces que no utilizan. Como consecuencia, el cliente define la interfaz/comportamiento de lo que requiere.

En un principio, se planteó el uso de la interfaz *IRepository* (ver Figura 5.5) como una buena aplicación de este principio. Sin embargo, analizando más en detalle la solución, se pudo concluir que algunas implementaciones de esta interfaz no necesitaban eliminarse, por lo que estamos forzando a estas clases a implementar comportamiento no requerido.

La solución correcta a esto sería quitar el método *Remove* de *IRepository*, y crear una nueva interfaz, *IDeletable* por ejemplo, que tenga un método *Delete*, y que la requieran únicamente los repositorios que necesiten de ese comportamiento.

```
public interface IRepository<T> {
    51 references
    void Add(T entity);

    73 references
    void Save();

    23 references
    void Update(T entity);

    63 references
    T Get(Guid id);

    28 references
    T Get(Func<T, bool> condition);
    9 references
    void Remove(T entity);

    6 references
    IEnumerable<T> GetAll();

    26 references
    IEnumerable<T> GetAll(Func<T, bool> condition);
}
```

Figura 5.5: IRepository.

Dependency Inversion Principle (DIP)

> Los módulos que implementan funcionalidades de alto nivel no deben depender de los de bajo nivel, sino que ambos deben depender de abstracciones bien definidas.

Se crearon interfaces bien definidas para las interacciones entre las distintas capas, como se puede ver en la Figura 5.6.

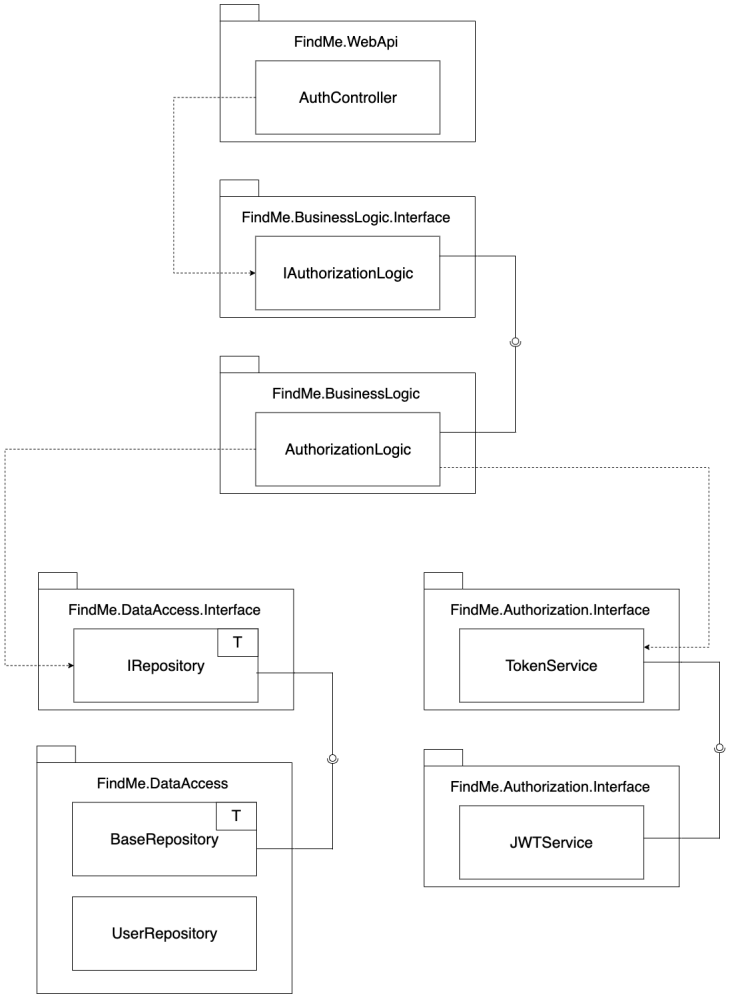

Figura 5.6: Ejemplo inversión de dependencias.

5.1.2. Patrones de Diseño

Patrón Repository

Se utilizó el patrón repository para aislar el acceso a datos de su implementación en sí, haciendo posible el cambio o extensión de cómo se accede a los datos de una manera muy simple y sin cambiar los módulos que lo utilizan.

Este patrón favorece el principio de inversión de dependencias.

Un ejemplo de esto se puede ver en la Figura 5.7.

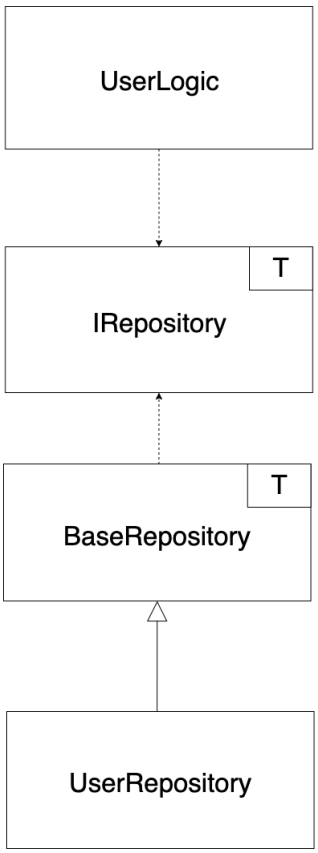

Figura 5.7: Repository Pattern.

Patrón de Inyección de Dependencias

Se utilizó el patrón de inyección de dependencias con el motivo de depender de módulos abstractos y no de sus implementaciones. Un ejemplo de esto se puede ver en la Figura 5.8.

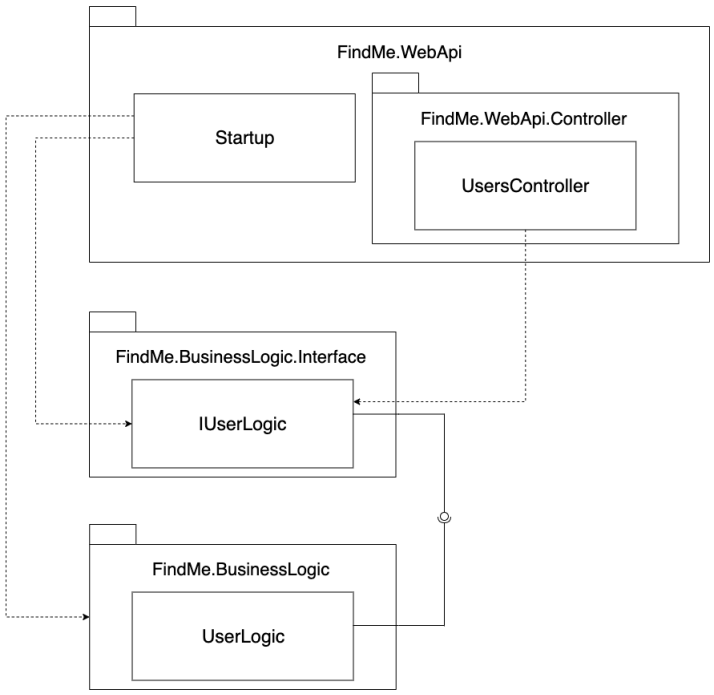

Figura 5.8: Patrón de Inyección de Dependencias.

Patrón Adapter

Utilizamos el patrón adapter para que la lógica de negocio no dependa de módulos externos que fuercen un cambio en éste. Para esto creamos un módulo que expone la interfaz requerida por la lógica, y otro módulo que la implementa. De esta manera, se puede cambiar, o en su defecto extender, fácilmente quien implementa la interfaz sin cambios en la lógica, llegando a una estructura más desacoplada.

Esto se utilizó particularmente para la generación de tokens de autenticación, para no depender del módulo que genera dichos tokens (ver Figura 5.9).

Notamos que se podría haberse implementado este patrón para el envío de mensajes, pero no quedó pronto al momento. Es un cambio planificado para los siguientes sprints.

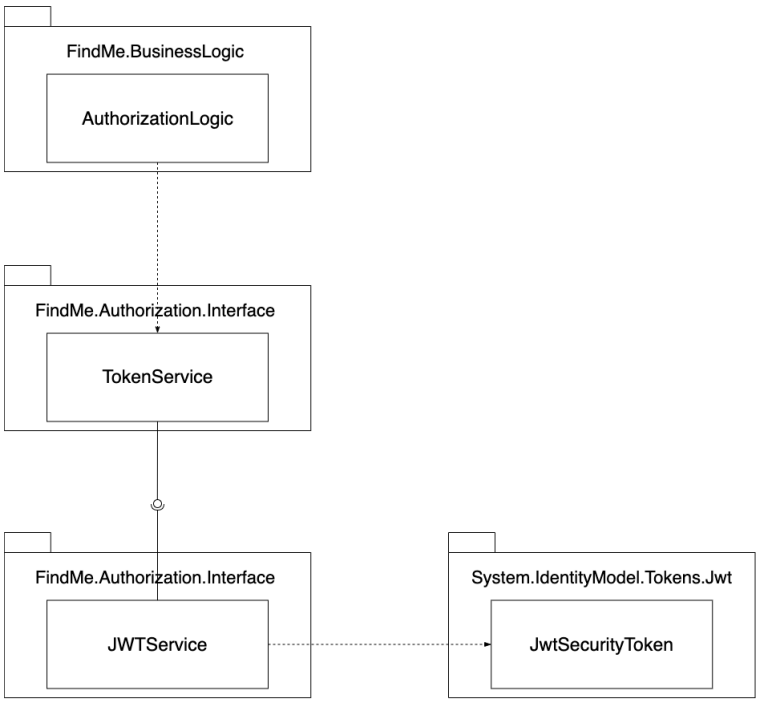

Figura 5.9: Patrón Adapter.

Patrón Polimorfismo

Se implementó el patrón polimorfismo para el acceso a datos. Para esto se provee un repositorio abstracto que implementa la ya mencionada interfaz de repositorio, de los cuales hereda cada repositorio de la solución. Ver Figura 5.10.

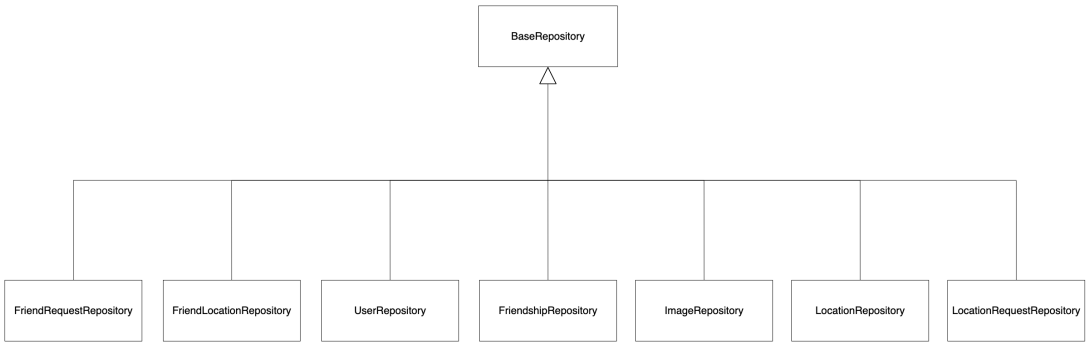

Figura 5.10: Patrón Polimorfismo.

5.1.3. Principios a nivel de paquetes

Se intentaron seguir los siguientes principios:

- Principio de dependencias estables.
- Principio de abstracciones estables.

En su gran mayoría se siguieron estos principios, aunque el Dominio rompe con esto. Esto se debe a que Entity Framework no posibilita utilizar abstracciones/interfaces.

5.2. Frontend

5.3. Distribución de Proyectos

La aplicación del frontend fue desarrollada en dos proyectos separados, uno completamente independiente y encargado del dominio y lógica de negocio, y otro que maneja UI, llamadas al backend, conexión que servicios externos, y también utiliza el primer proyecto mencionado.

Esta elección fue tomada con el fin de independizar lo que es la lógica y el dominio de factores externos como los que se mencionaron anteriormente.

5.3.1. Clean Architecture

Para la solución del Frontend se siguió la arquitectura propuesta por Robert C. Martin en el libro [1]. La arquitectura se puede ver en la Figura 5.11.

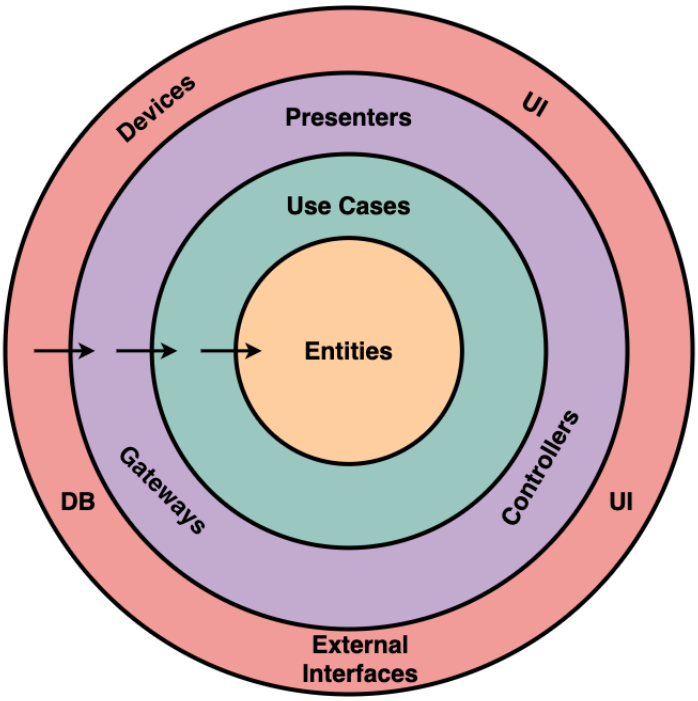

Figura 5.11: Clean Architecture.

Clean architecture se basa en 2 principios que fundan la estructura resultante.

- **Abstracción**: Los módulos de más al centro son los más abstractos y son los que contienen la lógica de negocio, mientras que los de afuera contienen código relacionado al framework y detalles de implementación.
- **Dependencias**: En la Figura se pueden apreciar flechas, estas señalan la dirección de las dependencias. Cada módulo puede depender únicamente del módulo interior.

Con lo mencionado anteriormente, las capas de implementación dependen de las capas abstractas, favoreciendo la mantenibilidad de la solución.

Las capas se pueden resumir a lo siguiente (enumeradas de la más interna a la más externa):

- Reglas de negocio de la empresa.
- Reglas de negocio de la aplicación.
- *Interface Adapters*.
- Frameworks y drivers.

5.3.2. MVVM

Model-View-ViewModel (MVVM) es un patrón de arquitectura que facilita la separación del desarrollo de interfaces gráficas (vista) del desarrollo de la lógica de negocio (modelo) haciendo ambos independientes uno del otro. Es una arquitectura *event-driven* (guiada por eventos).

Para esto se utiliza el ya mencionado *ViewModel* que actúa como intermediario entre el modelo y la vista.

El rol de cada una de las partes se lista a continuación:

- **Model**: Los datos y la lógica de negocio de la aplicación.
- **View**: Cómo se representan los datos, el responsable de la UI.
- **ViewModel**: Lo que une el modelo y la vista, se basa en llamadas asincrónicas (o *async callbacks*).

Para esto se utilizará la clase *ViewModel*[**?**] que provee Android.

Figura 5.12: MVVM.

5.3.3. Otros comentarios

Se utilizaron *Coroutines*[**?**] para el manejo de acciones asincrónicas, como por ejemplo llamadas al backend, o acciones que no pueden realizarse en el main thread, como borrar el token de Firebase.

Esto permitió darle a la aplicación mejor usabilidad, sin bloquearla con acciones que pueden llevar tiempo.

También se intentó dar el mejor posible uso a los *lifecycle methods*, haciendo uso del *onResume*, *onActivityCreated*, etc. según corresponda, y lo que mejor se acomodara a las necesidades de la app.

Para mejorar la modificabilidad de la aplicación, y reducir las dependencias entre los componentes, es que se utilizó *Koin* como librería para la inyección de dependencias. Vale mencionar que esto sigue el patrón de inyección de dependencias antes mencionado.

Se tomó como opción utilizar *Dagger* en lugar de *Koin*, pero analizando los pros y las contras, se optó por utilizar *Koin*.

5.4. Cambios en Diseño y Arquitectura

5.4.1. Sprint 0

Se describieron la arquitectura y el diseño de la solución del backend antes de comenzar el Sprint 1, justo después de definir las historias de usuario.

Se puede ver en la Figura 5.13 un primer diseño de la solución.

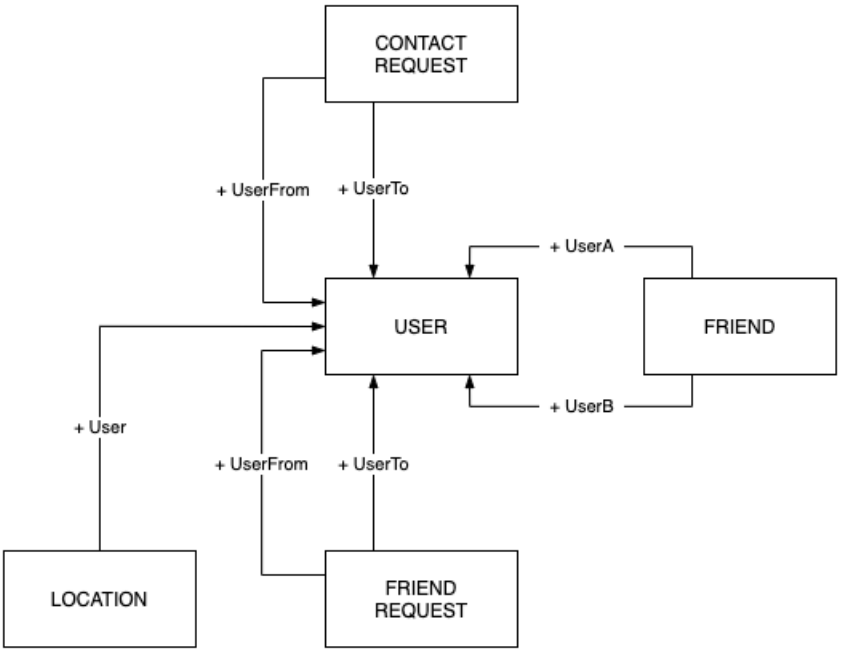

Figura 5.13: Diseño Backend Sprint 0.

En la Figura 5.14, se puede apreciar la base de la arquitectura pensada para la solución.

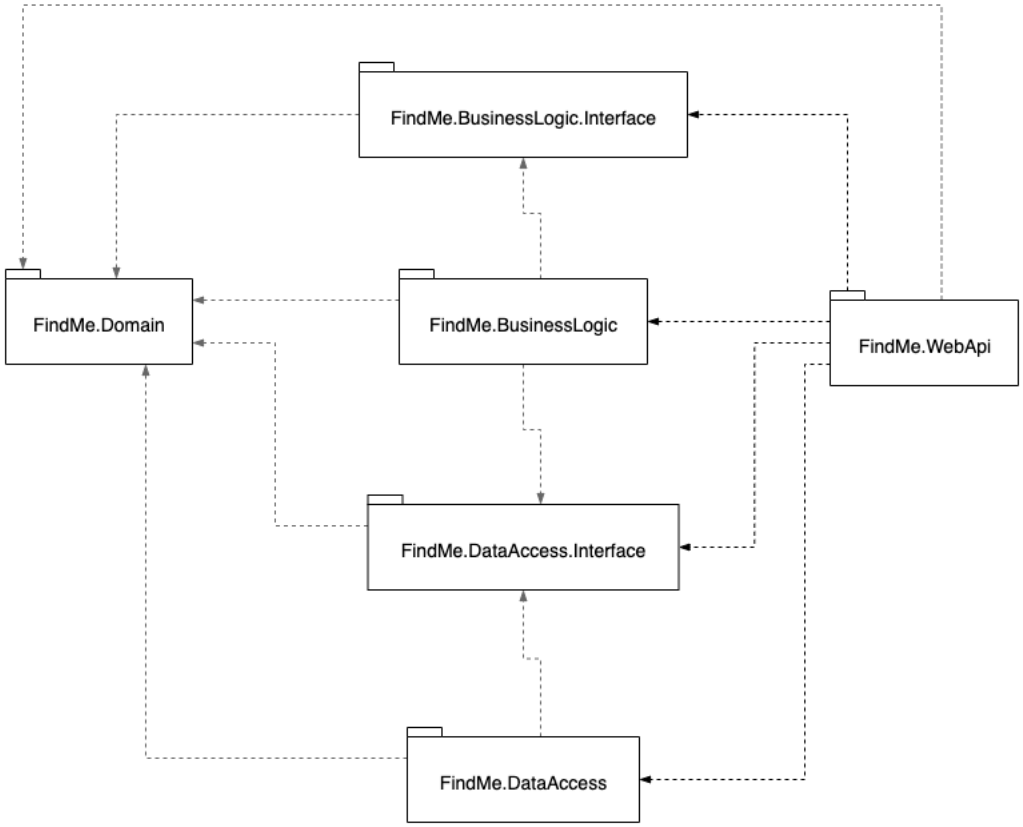

Figura 5.14: Arquitectura Backend Sprint 0.

5.4.2. Sprint 1

Durante el sprint 1, al dar con el problema, pudimos notar que tanto el diseño como la arquitectura eran bastante superficiales y necesitaban de mayor atención al detalle.

El resultado se puede ver en las Figuras 5.15 y 5.18.

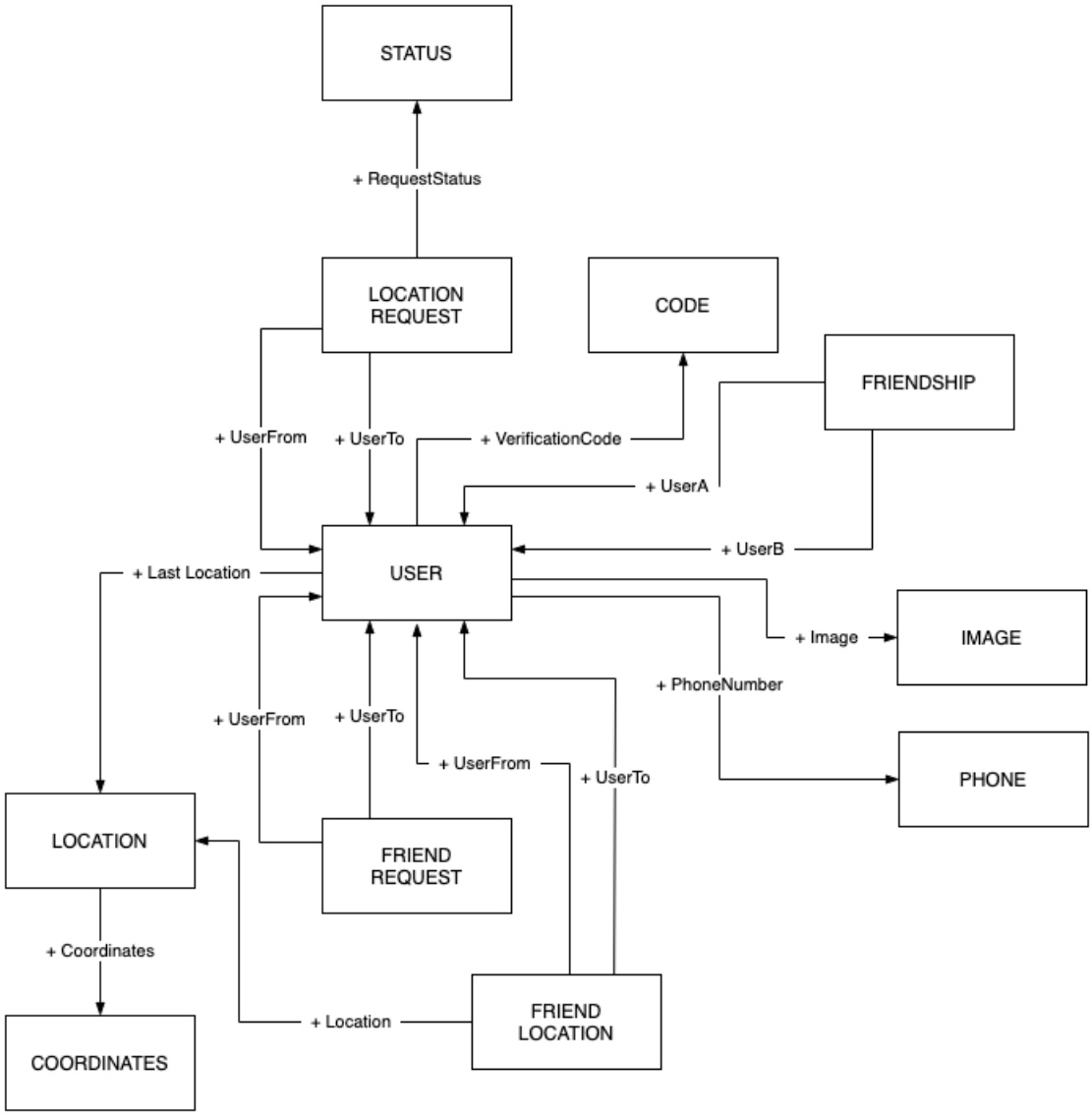

Figura 5.15: Diseño Backend Sprint 1.

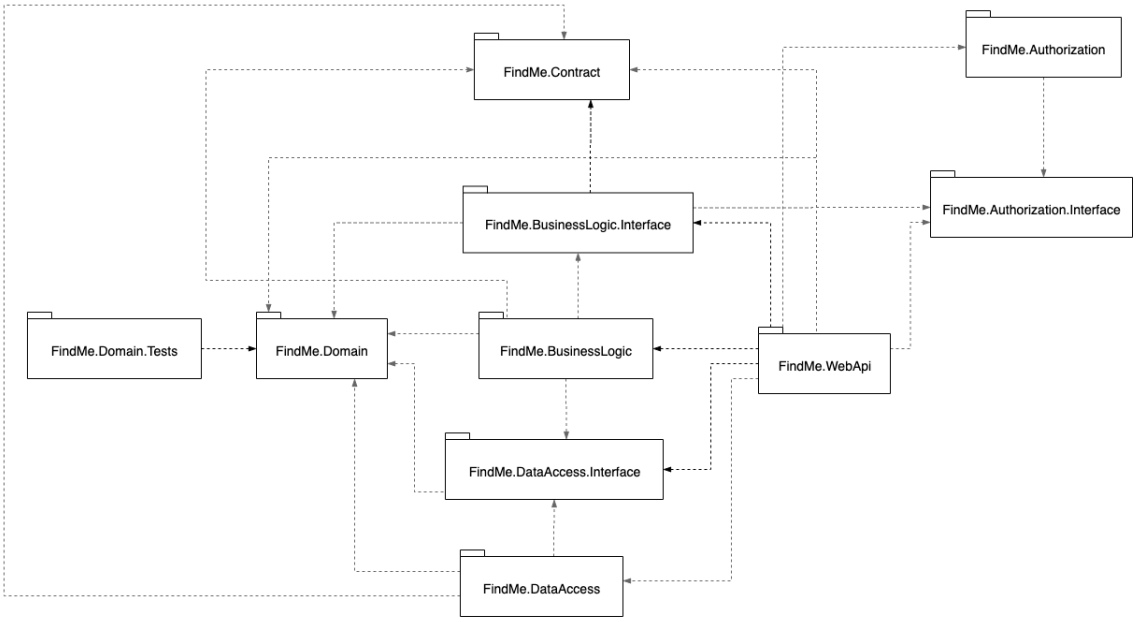

Figura 5.16: Arquitectura Backend Sprint 1.

5.4.3. Sprint 2

El sprint 1 dio como fruto un backend casi terminado, por lo que, a menos que se produzca algún cambio drástico en las historias de usuario y los requerimientos de la aplicación, no se espera que éste varíe mucho.

Por lo mencionado anteriormente, es que los nuevos cambios de este sprint se presentaron en el frontend de la aplicación.

La arquitectura planteada se puede ver en la Figura 5.20.

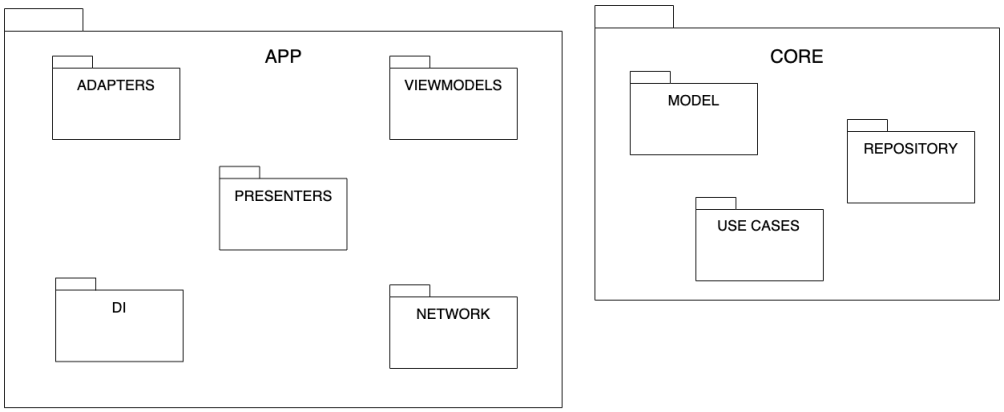

Figura 5.17: Arquitectura Frontend Sprint 2.

5.5. Evolución en los Sprints 3, 4 y 5

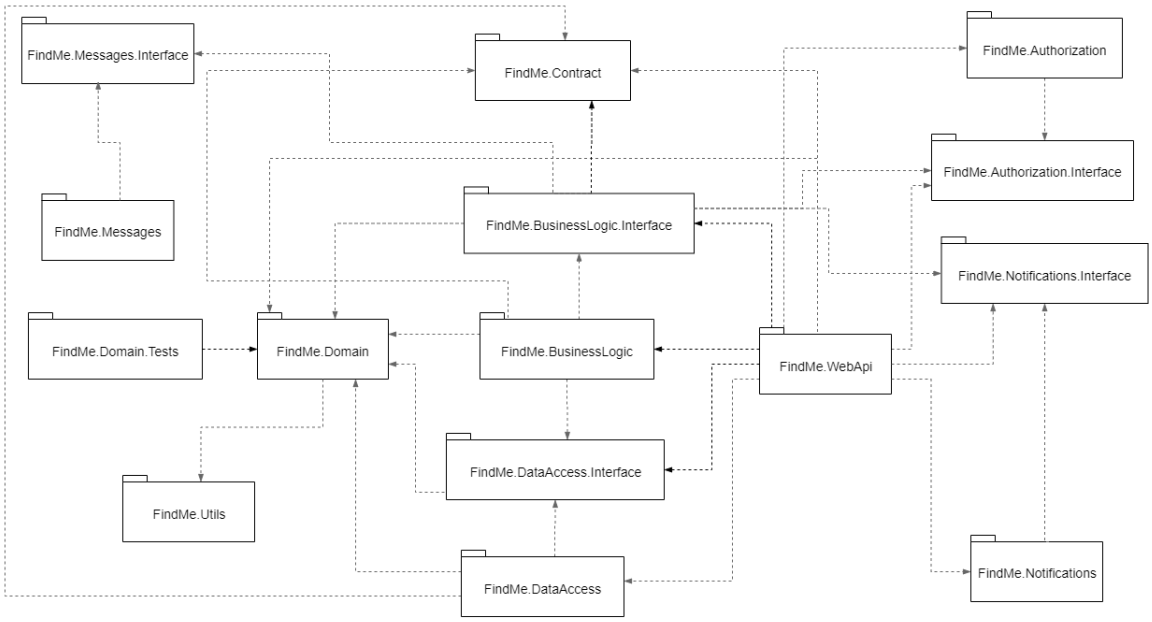

Figura 5.18: Arquitectura Backend Final.

La evolución por parte del backend en estas iteraciones es notoria, se agrega el paquete FindMe.Utils para cubrir la necesidad de establecer un icono por defecto a cada usuario recientemente registrado.

También se crea el paquete FindMe.Notifications, el cual es utilizado para el servicio de notificaciones que envía el backend, es necesario mencionar que se ha hecho uso de la herramienta brindada por Firebase.

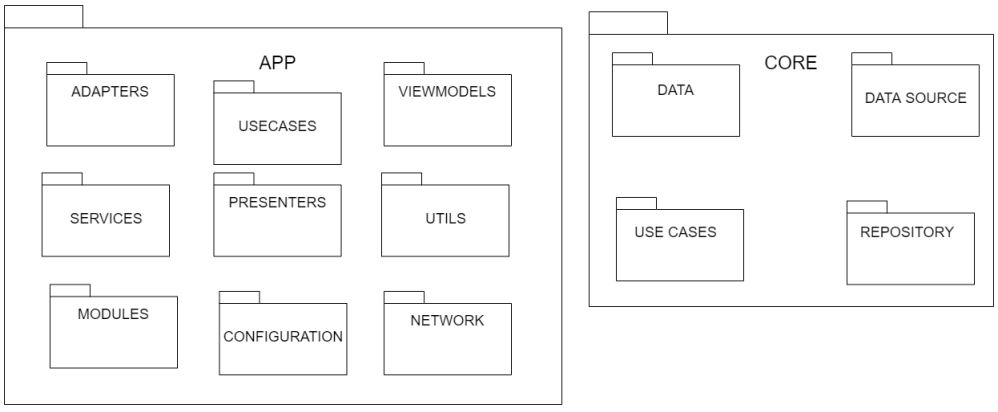

Figura 5.19: Arquitectura Frontend Final.

En lo que a frontend se refiere, dado que las primeras iteraciones se han centrado en obtener una WebApi complentamente funcional, en estas últimas iteraciones principalmente la mayor parte del esfuerzo por parte del equipo fue puesta en el frontend. Respecto a la evolución del mismo desde la iteración 2, se pueden apreciar

lo que serían nuevos espacios dentro de APP y CORE. En todo momento se ha desarrollado teniendo en cuenta el modelo MVVM anteriormente mencionado.

5.6. Modelo de la Base de Datos

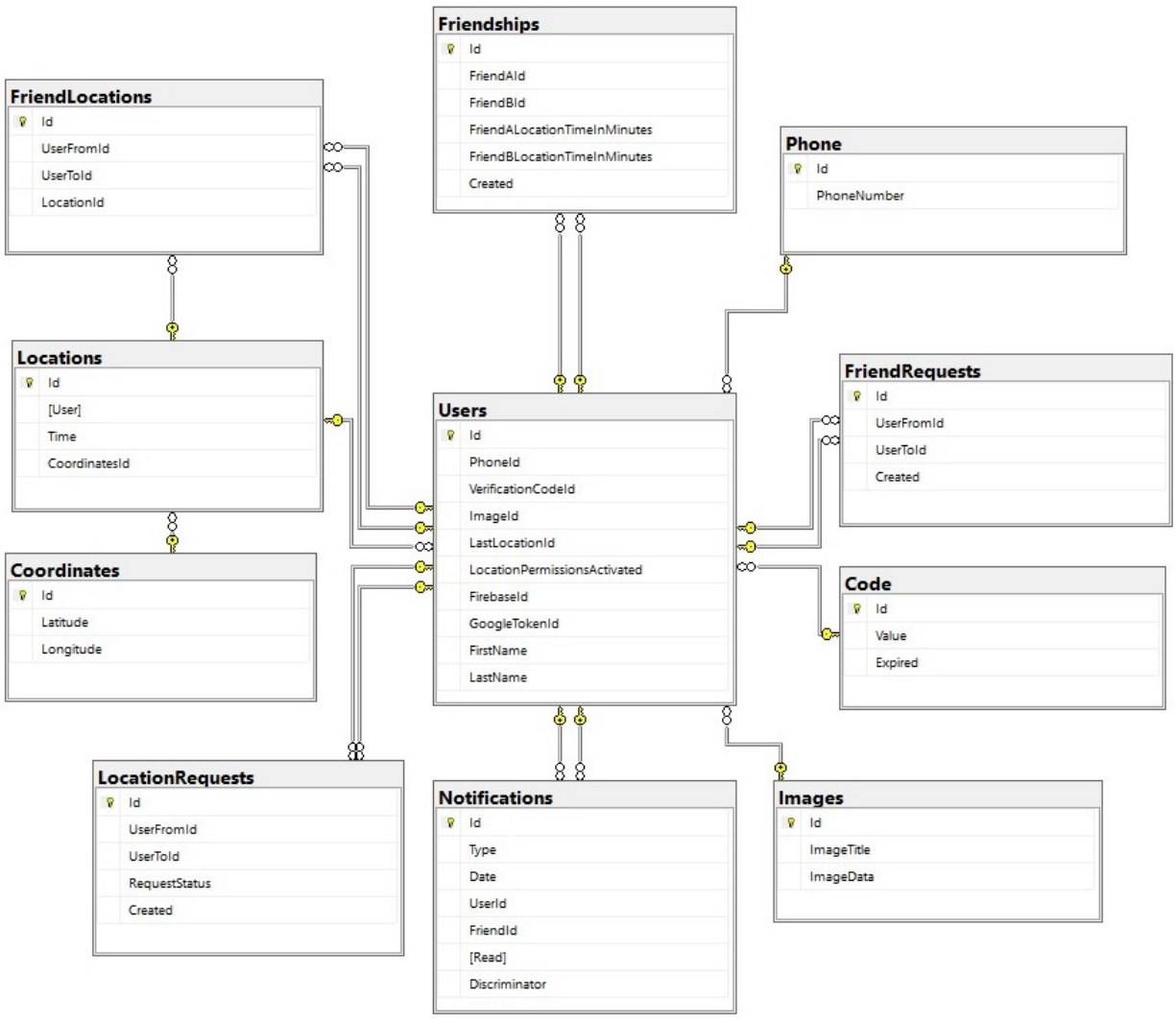

Figura 5.20: Nuevo modelo de la base de datos.

La imágen muestra una clara evolución en el diseño de la base de datos de la WebApi, donde se incluyen nuevas tablas como por ejemplo Notifications, también se cambió el tipo de dato que se guarda en la tabla Coordinates a Float.

5.7. Principios de REST API

Nuestro objetivo fue diseñar una API cumpliera con *affordance*, la propiedad de diseño que comunica cómo algo debe ser usado sin necesidad de documentación.

Para esto se deben seguir las buenas prácticas definidas. Las principales de ellas se listan a continuación:

- Tener solo 2 URLs por recurso.
- Sustantivos SI, verbos NO.
- Utilizar verbos HTTP para operar sobre las colecciones y elementos.
- Utilizar sustantivos en plural, y utilizar nombres concretos.
- Evitar usar mas que */resource/identifier/resource*.
- Complejidad a la derecha del "?".
- Los errores deben brindar contexto y visibilidad sobre cómo salieron las cosas.
- Manejar errores utilizando HTTP status codes.

Los HTTP Status Codes que se utilizaron hasta el momento son los siguientes:

- 200 - OK.
- 201 - Created.
- 400 - Bad Request.
- 404 - Not Found.
- 500 - Server Error.

A pesar de no necesitar documentación, decidimos crear utilizar las *Swagger Annotations* para generar documentación de Swagger a medida que se iba avanzando en el desarrollo de la aplicación. Ésta se puede ver corriendo el backend y accediendo a la URL base.

Una imagen a modo de ejemplo se muestra a continuación.

Figura 5.21: Documentación de swagger.

Evidencia

A continuación se puede ver un ejemplo de la aplicación de las prácticas definidas anteriormente, así como las *annotations* mencionadas cuyo rol es generar la documentación de swagger.

```
[SwaggerOperation(
    Summary = "Create User",
    Description = "Creates a user if there isn't one with the same phone number already"
)]
[SwaggerResponse(StatusCodes.Status201Created, "User", typeof(CreateUserModel))]
[SwaggerResponse(StatusCodes.Status400BadRequest)]
[SwaggerResponse(StatusCodes.Status404NotFound)]
[HttpPost]
public IActionResult CreateUser([FromBody] CreateUserModel model) {
    try {
        User user = this.users.CreateUser(model.ToEntity());
        return Ok(CreateUserModel.ToModel(user));
    } catch (LogicException e) {
        return BadRequest(e.Message);
    } catch (NotFoundException e) {
        return NotFound(e.Message);
    }
}
```

Figura 5.22: Ejemplo de buen uso de prácticas REST.

La API, al ser realizada en su gran mayoría antes que el frontend, se probó mediante Postman. La colección y el ambiente se encuentran en el repositorio con el fin de que todos los integrantes puedan acceder a ellas.

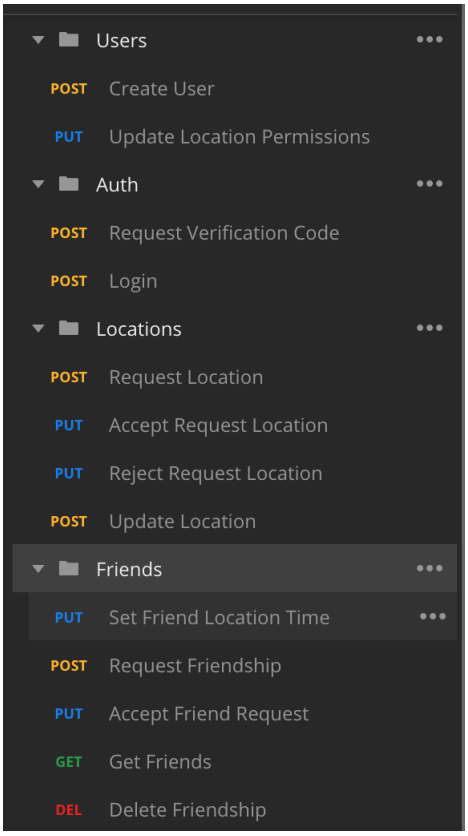

Figura 5.23: Colección postman.

6. Tecnologías Utilizadas

- Frontend en Android (Kotlin).
- Backend en .NET Core.
- Entity Framework Core.
- Notificaciones con Firebase.
- Autenticación con Google.

6.1. Funcionalidades del dispositivo

- Envío de SMS (Twilio Client desde el servicio backend).
- Lectura de código automática desde SMS.
- Cámara.
- Galería.
- Lista de contactos.
- Escaner QR.
- Mapa y GPS (Google Play Services).
- Autenticación con huella dactilar.

7. Implementación

7.1. Clean Code

A continuación se presentan los principales estándares de Clean Code que tuvimos en cuenta durante el desarrollo.

7.1.1. Nombres Significativos.

- Usar nombres que revelen intención.
- Evitar la desinformación.
- Distinciones significativas.
- Nombres pronunciables.
- Nombres fáciles de buscar.
- **Evitar Encodings**.
  - Hungarian Notation.
  - Prefijos.
- Las clases deberían ser un sustantivo o tener sustantivo.
- Los métodos deben ser un verbo o ser una frase con verbo.
- Ser consistente: (ej: `fetch, retrieve, get`) usar una para todo.

7.1.2. Funciones.

> "The first rule of functions is that they should be small. The second rule of functions is that they should be smaller than that".

Bloques e Indentación.

- Los bloques o cuerpo de if, try, whiles, loops en general deben tener una línea (function call).
- Los niveles de indentación no deben superar uno o dos niveles.
- Los niveles de abstracción no deben superar uno o dos niveles. **STEP DOWN RULE**.

Una Sola Cosa.

- Las funciones deben hacer una sola cosa.
- *¿Cuándo una función hace solo una cosa?* Cuando ya no es posible extraer otra función sin que sea la misma con otro nombre.

Switch.

- Lo uso **solo** en un lugar por jerarquía, y para instanciar los tipos polimórficos.
- RTTI: solo aceptamos el creacional.

Parámetros.

- Nombres descriptivos (tanto la función cómo los parámetros).
- Están a otro nivel de abstracción de la definición de la función. Esto se debe a que los parámetros dan información de cómo está implementado el método, a diferencia de la definición de la función que da información acerca de su comportamiento.
- El ideal son funciones sin parámetros, ya que facilitan las pruebas. Más de tres ya es demasiado.
- Parámetros bool o banderas pueden indicar que la función hace más de una cosa.

7.1.3. Errores.

Excepciones vs. Error Codes.

- Preferimos las excepciones.
- Recordemos que las funciones deben hacer una cosa solamente. El manejo de errores es una sola cosa. Por lo tanto, se debería hacer una función para el try y otra para el catch.

Se utilizaron excepciones padre en el paquete *FindMe.Contract*, que son las que se utilizan para decidir qué código de error va a retornar el controller.

Cada paquete tiene su propio namespace de exceptions donde tiene una excepción para cada uno de los posibles errores esperados. Estas excepciones heredan de las que se nombraron anteriormente, y cada una describe su propio error.

A continuación se muestran tanto las excepciones que contiene el paquete Contracts, como las del paquete BusinessLogic.

Figura 7.1: Excepciones de Business Logic.

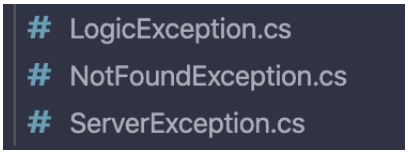

Figura 7.2: Paquete Contract.

7.1.4. Comentarios.

Motivos para no utilizarlos sin fundamentos:

- Se usan generalmente cuando fracasamos al expresarnos en el código.
- El código cambia y evoluciona. Los comentarios no siempre.

7.1.5. Formato.

Vertical.

- Cantidad máxima de líneas de código: 200 deseable, 500 máximo.
- Enters entre un método y otro.
- **Metáfora del Periódico**: "Vender la clase". Leer de lo más abstracto al detalle.

Horizontal.

- Tabs.
- Espacio entre if, while, etc.
- Máx. largo de línea.

7.1.6. Objetos y Estructuras de Datos.

Estructura de Datos.

- No tiene comportamiento.
- Solo tiene estado público.

Objeto.

- Se piensa desde el comportamiento.
- Una clase expone su comportamiento, no su estado.

Antisimetría.

- Los objetos ocultan sus datos detrás de abstracciones y exponen funciones que operan sobre esos datos.
- Las estructuras de datos exponen sus datos y no tienen funciones significante.
- **Ventajas de Estructuras de Datos**: Fácil de agregar nuevo comportamiento sin cambiar las estructuras existentes.
- **Ventajas de Clases**: Fácil de agregar un nuevo tipo sin modificar las funciones existentes.

La Ley de Demeter.

Una clase C solo debe llamar a métodos f de:

- La clase C.
- Un objeto creado por f.
- Un objeto argumento de f.
- Un objeto mantenido en una variable de instancia de C.

Train Wrecks.

Concatenar llamadas a métodos o atributos. Se debe evitar.

Data Transfer Objects.

- Su responsabilidad es transferir información.
- Sirve para agrupar parámetros.

Las utilizamos para los modelos que se utilizan en los Controllers que expone la API, como se ve en la siguiente imagen.

```
using System;
using FindMe.Domain;
using System.ComponentModel.DataAnnotations;

You, 18 days ago | 1 author (You)
namespace FindMe.WebApi.Models {
    0 references | You, 18 days ago | 1 author (You)
    public class ImageModel {
        [Required]
        1 reference
        public byte[] ImageData { get; set; }
        [Required]
        1 reference
        public Guid Id { get; set; }
        [Required]
        1 reference
        public string ImageTitle { get; set; }

        0 references
        public ImageModel() { }

        0 references
        public ImageModel(Image image) {
            this.ImageData = image.ImageData;
            this.ImageTitle = image.ImageTitle;
            this.Id = image.Id;
        }
    }
}
```

Figura 7.3: DTO de Imagen.

7.1.7. Manejo de Errores

- El código tiene que ser legible pero también robusto.
- Debemos ver el manejo de errores como una responsabilidad aparte, independiente de la lógica.
- Usar excepciones y no códigos de error.
- Definir excepciones según quién las llama.
- No retornar null, no pasar null.

El correcto manejo de errores se puede ver en la Figura 7.4. Esto funciona correctamente para todos los errores esperados.

```
[SwaggerOperation(
    Summary = "Request login",
    Description = "Generates a verification code used for login"
)]
[SwaggerResponse(StatusCodes.Status200OK)]
[SwaggerResponse(StatusCodes.Status404NotFound)]
[HttpPost("verification-code/{id}")]
2 references
public IActionResult GenerateVerificationCode(Guid id) {
    try {
        this.authorization.GenerateVerificationCode(id);
        return Ok("Code generated successfully");
    } catch (LogicException e) {
        return NotFound(e.Message);
    } catch (NotFoundException e) {
        return NotFound(e.Message);
    }
}
```

Figura 7.4: Manejo de errores.

Para los errores no esperados, se implementó un filtro de errores, donde cada error no controlado devuelve una respuesta con código de error 500, como se ve en la Figura 7.5.

```
using FindMe.WebApi.Models;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;

You, 14 days ago | 1 author (You)
namespace FindMe.WebApi.Filters {
    1 reference | You, 14 days ago | 1 author (You)
    public class ErrorFilter : IExceptionFilter {
        0 references
        public void OnException(ExceptionContext context) {
            ObjectResult response = new ObjectResult(new ErrorResponse("Internal Server Error"));
            response.StatusCode = StatusCodes.Status500InternalServerError;
            context.Result = response;
        }
    }
}
```

Figura 7.5: Filtro de errores.

7.1.8. Límites.

- Usando código de terceros ⇒ Tratar siempre de hacer un wrapper.
- Usando código que no existe aún ⇒ Adapter.

7.1.9. Clases.

- Las clases deben ser **pequeñas**.

- El largo de las clases se cuenta en responsabilidades.
  - Las clases deben tener una única responsabilidad.
  - (SRP) - Single responsibility principle.
    - Una sola razón por la cual cambiar.
    - Clases muy largas ⇒ síntoma de muchas responsabilidades.
  - Las clases deben tener alta cohesión. Muchas variables ⇒ síntoma de baja cohesión.

Organización.

1. `public static const`
2. `private static variables`
3. `private instance variables`
4. `public functions`
5. `private utility function` después de la `public function` que la usa.

Esto respeta la **metáfora del periódico**.
Un ejemplo claro de esto en el código se puede ver en la clase *Code* de la siguiente imagen.

```
public class Code {
    1 reference
    private readonly int CODE_LENGTH = 4;

    1 reference
    public Guid Id { get; private set; }
    11 references
    public string Value { get; private set; }
    7 references
    public bool Expired { get; set; }

    7 references
    public Code() {
        this.Id = Guid.NewGuid();
        this.Generate();
    }

    8 references
    public void Generate() {
        int number = new Random().Next(0, 9999);
        this.Value = number.ToString("D" + CODE_LENGTH);
        this.Expired = false;
    }
}
```

Figura 7.6: Clase Code.

Ejemplos en Frontend

```kotlin
package com.example.core.repository

import com.example.core.datasource.AuthorizationDataSource

class AuthorizationRepository(private val dataSource: AuthorizationDataSource) {
    suspend fun generateVerificationCode(mobile: String) = dataSource.generateVerificationCode(mobile)
    suspend fun login(mobile: String, code: String) = dataSource.login(mobile, code)
    suspend fun loginWithGoogle(tokenId: String) = dataSource.loginWithGoogle(tokenId)
}
```

Figura 7.7: AuthorizationRepository.

```kotlin
package com.example.findmeapp.network.services

import ...

class UserService(private val userApiService: UserApiService) :
    UserDataSource {
    override suspend fun register(user: User): User {
        try {
            val userModel = UserModel.fromUser(user)
            val result: BaseModel<UserModel> = userApiService.register(userModel)
            if (!result.success) {
                throw Exception(result.message)
            }
            return result.data.toUser()
        } catch (e: HttpException) {
            val errorResponse = getErrorResponse(e)
            throw Exception(errorResponse.message)
        }
    }
```

Figura 7.8: UserService.

7.2. Código

Como se ha mencionado en secciones anteriores para asegurar el cumplimiento de los estándares de clean code mencionados en el libro, se ha implementado un formateador para que en cada commit de formato a nuestro código según definimos en el siguiente archivo:

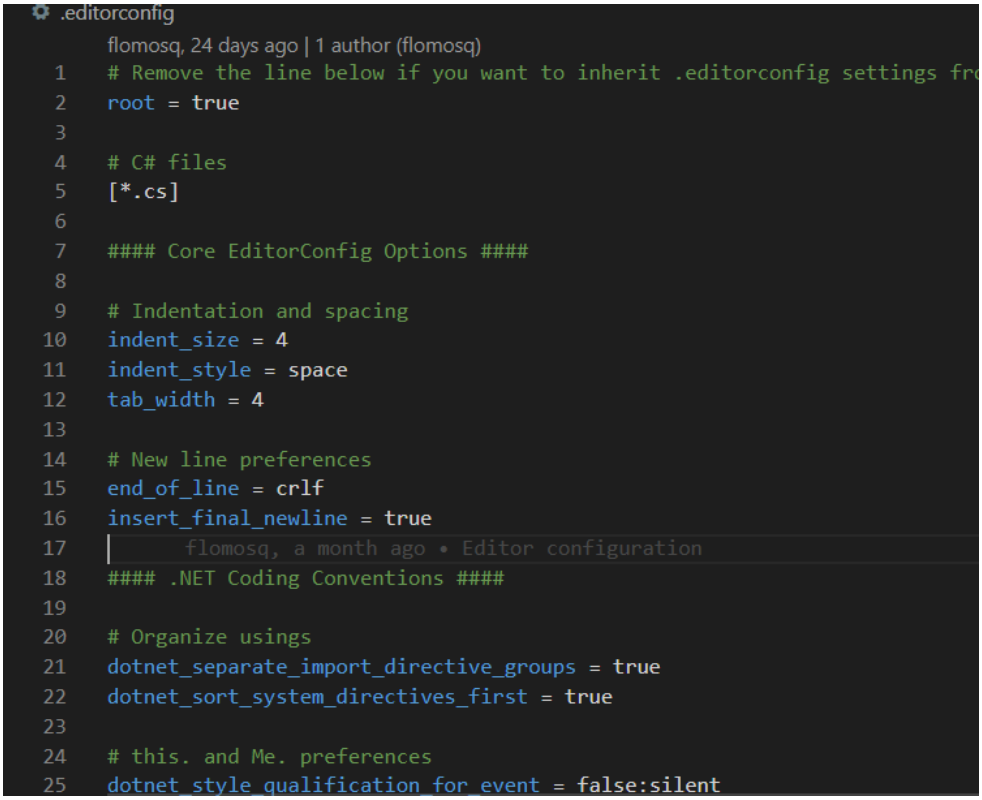

Figura 7.9: Archivo de configuración para dar formato al código.

También para el frontend de la aplicación se ha utilizado ktlint, lo cual ayuda a mantener los estándares definidos para la programación en android con kotlin por la comunidad.

7.3. Datos de Prueba

Es importante mencionar que si se desea probar el registro de un nuevo usuario, o loguearse con un usuario ya existente a la aplicación, se deben utilizar ciertos números de teléfonos que serán provistos a continuación, o como alternativa notificar a los autores del proyecto si se desea ingresar un nuevo número.

Esto se debe a que el log in de la aplicación se maneja con códigos de verificación que son generados en el backend y enviados por SMS mediante Twilio Client. Como la aplicación está en una versión preliminar, se ha decicido utilizar la versión gratis de Twilio Client, la cual nos permite enviar mensajes con un saldo limitado, y con números de destinatarios registrados en la cuenta de Twilio Client. Por eso pedimos que si se desea registrar un nuevo número se notifique a los autores para agregarlos a la cuenta de Twilio Client o en otro caso, utilizar uno de los siguientes ya registrados:

- 099 497 311
- 096 379 293
- 095 272 903

7.4. Manual de instalación

A continuación se mostrará un manual paso a paso para instalar la aplicación. Igualmente también en los correspondientes repositorios de Frontend y Backend se encuentran archivos README.txt que explican su instalación.

El APK creado estará configurado para funcionar en un AVD.

7.4.1. Frontend

1. Descargar APK en el repositorio FindMeApp en el dispositivo.
2. Abrir el archivo.
3. Continuar guía de instalación (debe aparecer un modal).

7.4.2. Backend

1. Descargar archivo .bak del repositorio FindMe (el full contiene datos de prueba, el otro está vacío).
2. Restaurar base de datos con el archivo .bak
3. Luego ejecutar el comando 'dotnet build –configuration Release'
4. Finalmente ejecutar el comando 'dotnet build –configuration Publish'

Bibliografía

[1] R. C. Martin, *Clean Architecture*, 14th ed. Pearson, 2008.

[2] ——, *Clean Code*, 14th ed. Prentice Hall, 2008.