Implementación de Arquitectura Limpia en una aplicación móvil. Registros de ingresos y egresos de personas

Maria Laura Godoy[1], Mgter. Anita del Carmen Lopez[1], Dra. Sonia I. Mariño[1].

[1] Universidad Nacional del Nordeste, Corrientes, Argentina

marialgodoy15@gmail.com,aalegrelopez@yahoo.com.ar,simarinio@yahoo.com

Resumen. Es importante que los organismos públicos tengan acceso a una herramienta para digitalizar y centralizar la información de forma sencilla e intuitiva. Últimamente, se incrementó en importancia el teléfono celular, así las aplicaciones móviles mejoran cada vez para facilitar diversas actividades. Por lo que, es de suma importancia tener en cuenta la mantenibilidad a la hora de diseñar una aplicación. En este documento se indaga en referencia a la Arquitectura Limpia y su adaptación para generar una aplicación móvil de registros de los ingresos y egresos de ciudadanos en organismos públicos. Se presenta la adaptación de esta arquitectura en el marco de un proyecto de fin de carrera e implementado en una pasantía en TelCo, se incluye una descripción y los respectivos diagramas de los distintos comportamientos de la aplicación.

Palabras clave: Aplicación móvil, Clean Architecture, Ingeniería del Software.

1 Introducción

1.1 Breve estado del arte

Hoy en día algo que llegó a ser indispensable en nuestra vida es el teléfono celular y todas las posibilidades que las aplicaciones dentro de él pueden ofrecernos durante la vida cotidiana.

“Las aplicaciones móviles se han convertido en herramientas de gran ayuda en la actualidad en cualquier campo o actividad tales como en el comercio, educación, investigación, etc. Cada día las aplicaciones mejoran para convertirse en programas indispensables que nos facilitan cualquier actividad que realicemos, nos aportan facilidad en el ámbito laboral permitiéndonos así administrar de manera eficiente cualquier proceso [1]”.

Teniendo en cuenta lo mencionado anteriormente, es importante destacar la importancia del diseño de una aplicación para que pueda ser fácilmente mantenible. Una característica importante para lograr esta mantenibilidad es el buen diseño e implementación de una arquitectura de software.

“La Arquitectura Limpia es una filosofía de diseño de software que separa los elementos de un diseño en niveles de anillo. La regla principal de la Arquitectura Limpia es que las dependencias de código sólo pueden provenir de los niveles externos hacia adentro. El código en las capas internas no puede tener conocimiento de las funciones en las capas externas... [2]”. Esta regla es llamada “regla de dependencia” y quiere decir que el círculo interno no debe de reconocer nada acerca de algo dentro de un círculo exterior. En particular, el nombre de alguna declaración dentro de un círculo externo no debería ser mencionada en un círculo interno. No queremos que nada de un círculo exterior impacte en el círculo interior [3].

Los principios que sigue esta arquitectura son los llamados, por sus siglas en inglés, SOLID y significan [4]:

- S: El principio de responsabilidad única (SRP), que todo objeto debería de tener una única responsabilidad.
- O: El principio de abierto/cerrado (OCP), donde todas las entidades deben estar abiertas para su extensión, pero cerradas para su modificación.
- L: El principio de sustitución Liskov (LSP), todo objeto debe poder ser reemplazable por instancias de sus subtipos sin alterar el funcionamiento del programa.

- I: El principio de segregación de la interfaz (ISP), el uso de muchas interfaces específicas es mejor que utilizar una única interfaz general.
- D: El principio de inversión de la dependencia (DIP), se debe depender de las abstracciones más no de las implementaciones.

La ventaja de utilizar Arquitectura Limpia es la posibilidad de cambiar dependencias externas o detalles de la implementación tanto de la capa de datos como la capa de visualización sin que esto afecte de alguna manera a la lógica de negocio, además de simplificar la aplicación de pruebas y la implementación de posibles nuevos requerimientos siguiendo las reglas de dependencia [5].

1.2 Aspectos de ética profesional vinculados con el proyecto

Se consideró dentro del Régimen Autónomo Argentino en materia de protección de la propiedad intelectual tratada por la ley 11.723 "Régimen Legal de la Propiedad Intelectual", que estipula que todo autor o inventor es propiedad exclusiva de su obra, invento o descubrimiento por el término que le acuerde la ley.

- Artículo 4. Titulares del derecho de la propiedad intelectual:
    - inciso "d": "Las personas físicas o jurídicas cuyos dependientes contratados para elaborar un programa de computación hubiesen producido un programa de computación en el desempeño de sus funciones laborales, salvo estipulación en contrario."

1.3 La Ley de datos personales en relación al PFC

En cuanto a la privacidad y seguridad de los datos, se usó como referencia lo establecido en la ley 25.326 sobre la Protección de Datos Personales en Argentina estableciendo los principios generales relativos a la protección de los datos, así como también el control y los derechos que le corresponden al usuario. Aquellos artículos que más se adecuan al proyecto, se detallarán en breve.

- Artículo 2. Define a datos personales como "información de cualquier tipo referida a personas físicas o de existencia ideal determinadas o determinables."
- Artículo 4. Calidad de los Datos:
    - inciso 1: "Los datos personales que se recojan a los efectos de su tratamiento deben ser ciertos, adecuados, pertinentes y no excesivos en relación al ámbito y finalidad para los que se hubieren obtenido."
    - Inciso 3: "Los datos objeto de tratamiento no pueden ser utilizados para finalidades distintas o incompatibles con aquellas que motivaron su obtención."
- Artículo 9. Seguridad de los Datos
    - inciso 1: "El responsable o usuario del archivo de datos debe adoptar las medidas técnicas y organizativas que resulten necesarias para garantizar la seguridad y confidencialidad de los datos personales, de modo de evitar su adulteración, pérdida, consulta o tratamiento no autorizado, y que permitan detectar desviaciones, intencionales o no, de información, ya sea que los riesgos provengan de la acción humana o del medio técnico utilizado."

1.4 Objetivos

Diseñar una aplicación móvil utilizando Arquitectura Limpia que permita automatizar el proceso de registro de ingresos y egresos de ciudadanos en los organismos públicos.

Objetivos específicos:

- Investigar sobre Arquitectura Limpia en el diseño de aplicaciones móviles.
- Desarrollar una aplicación móvil que registre los ingresos y egresos de las personas dentro los organismos públicos de la provincia de Corrientes.

1.5 Fundamentación

Al analizar la situación dentro de los diversas instituciones públicas de la ciudad de Corrientes se pudo observar que la mayoría no poseen un proceso formal que realice el registro de sus visitantes o bien, si lo poseen, estos son procesos dificultosos a la hora de la obtención de un reporte de estos registros, generando demoras y muchas veces errores.

"Todas las instituciones del estado requieren una herramienta informática para controlar y registrar la información de las personas que ingresan a realizar trámites documentación para un mayor control de las misma con el fin de tener un soporte de seguridad ante situaciones adversas que se podrían presentar en cualquier momento por diferentes índoles… [6]". Es importante que los organismos públicos posean estas herramientas de fácil acceso y uso para sus empleados para que puedan digitalizar y centralizar toda la información de los registros puesto que "…la información ordenada y sistematizada facilita y ayuda a la toma de decisión en una empresa o institución, con el fin de apoyar en el alcance de los objetivos de la empresa. El valor de la información disponible ayuda a alcanzar nuevos retos a la empresa [6]".

Esta propuesta se desarrolla en el marco del Proyecto Final de Carrera de la Licenciatura en Sistemas de Información de la Facultad de Ciencias Exactas y Naturales y Agrimensura. Universidad Nacional del Nordeste.

2 Metodología

Para el desarrollo de este sistema se siguió una adaptación de distintos métodos ágiles, como Kanban y Scrum, puesto que "…son más adecuados para el diseño de aplicaciones en que los requerimientos del sistema cambian, por lo general, rápidamente durante el proceso de desarrollo… [7]". De cada una de estas metodologías se utilizó algún concepto específico. Por ejemplo, del método Kanban, se hizo uso del concepto de "tableros" con el fin de anotar y llevar un seguimiento de las tareas. En cambio, del método Scrum, se hizo uso del concepto de Sprints para realizar las presentaciones y retroalimentaciones de los avances del proyecto.

A continuación, se detallan las etapas seguidas:

1. Planificación: En esta etapa se procedió a consolidar el objetivo del proyecto y los requerimientos del sistema en conjunto con el usuario final/cliente. Se elaboró una lista de tareas de funcionalidades principales a implementar.
2. Diseño: En esta etapa se realizó el diseño del prototipo y los elementos del mismo para construir el sistema.
3. Desarrollo: En esta etapa se procedió a construir el sistema a través de distintos "sprints" o iteraciones en los cuales se incluyeron distintas tareas y funcionalidades a realizar en un periodo de tiempo determinado para su posterior demostración al final de cada una de estas iteraciones. Estas tareas fueron planteadas en la plataforma utilizada para el seguimiento del proyecto. A continuación, una tabla con los sprints definidos y las distintas tareas incorporadas dentro de cada uno:

Tabla 1. Tabla de Sprints Definidos

SPRINT	TAREA	ESTADO	ESTIMACIÓN
1	Inicio de Sesión	FINALIZADA	3
	Pantalla Principal	FINALIZADA	3
	Registro Manual	FINALIZADA	6
	Lista de Registros	FINALIZADA	6
	Eliminar Registro Local	FINALIZADA	6
	Sincronización Remota Manual	FINALIZADA	4

2	Lista de Subsistemas	FINALIZADA	6
	Lista de Lugares	FINALIZADA	4
	Registro por Búsqueda de DNI	FINALIZADA	4
	Edición de Registros	FINALIZADA	4
	Botones de Pase Ingreso/Egreso	FINALIZADA	2
	Usuario con Lugar por defecto	FINALIZADA	4

4. Pruebas: Se realizaron pruebas durante todo el proceso de desarrollo con el fin de comprobar que el producto cumple con el requisito propuesto por el usuario además de funcionar de manera correcta.
5. Entrega: En esta etapa se procedió a realizar la entrega del producto finalizado al cliente, el cual revisó el mismo y aportó algunas mejoras y ajustes para el producto.
6. Retroalimentación: En esta etapa se recopilaron y analizaron las propuestas obtenidas del cliente en la etapa anterior con el fin de verificar su viabilidad para poder implementarlas.
7. Iteración: Finalmente, en esta última etapa, se procedió a realizar el nuevo sprint del proyecto en donde se definió las nuevas tareas y mejoras a realizar dentro del sistema en función de la retroalimentación recibida.

Para el desarrollo de esta aplicación móvil, el proyecto se realizó durante dos iteraciones del ciclo de vida de la metodología adaptada que se describe en el presente documento. En la Fig. 1 se puede observar más detalladamente las etapas definidas anteriormente.

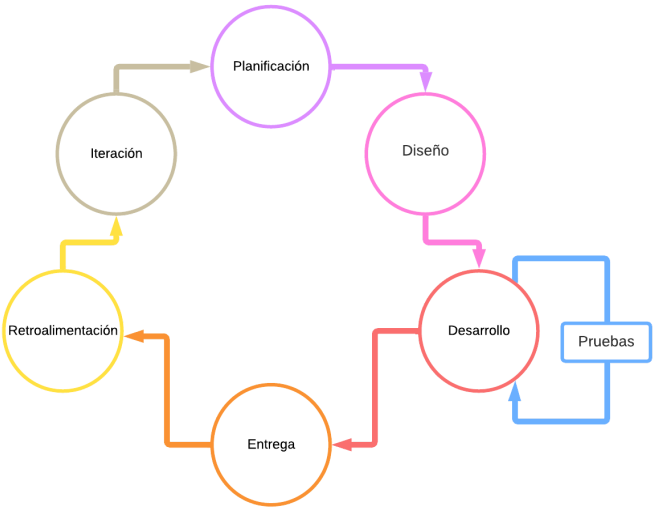

Fig. 1. Ciclo de vida de la metodología adaptada utilizada para este proyecto.

3 Herramientas y/o lenguajes de programación

Para el desarrollo de este proyecto final se utilizaron diferentes herramientas, las cuales se mencionan a continuación, junto con una breve descripción de la misma y el ámbito de uso.

3.1 Herramientas de seguimiento de proyecto

Azure DevOps: Es una herramienta de Microsoft que proporciona características integradas a las que puede acceder a través de tu explorador web o cliente IDE. Puede usar todos los servicios incluidos con Azure DevOps o elegir solo lo que necesita para complementar los flujos de trabajo existentes.De las distintas opciones que ofrece azure devops, utilizamos la de boards, la cual ofrece un conjunto de herramientas ágiles para admitir el trabajo de planificación y seguimiento, defectos de código y problemas mediante los métodos Kanban y Scrum. [8]

3.2 Herramientas de desarrollo

Lenguaje Dart: El lenguaje Dart que es un lenguaje optimizado para el cliente con el fin de poder desarrollar aplicaciones rápidas en cualquier plataforma. Su objetivo es ofrecer el lenguaje de programación más productivo para el desarrollo multiplataforma, junto con una plataforma de ejecución flexible para marcos de aplicaciones. [9]. Este lenguaje es en el que está basado el framework utilizado, el cual se describe a continuación.
Framework Flutter: El Framework utilizado para el desarrollo de la aplicación es Flutter. “Flutter es un conjunto de herramientas de interfaz de usuario multiplataforma que está diseñado para permitir la reutilización de código en sistemas operativos como iOS y Android, al tiempo que permite que las aplicaciones interactúen directamente con los servicios de la plataforma subyacente. El objetivo es permitir a los desarrolladores ofrecer aplicaciones de alto rendimiento que se sientan naturales en diferentes plataformas, aceptando las diferencias donde existan y compartiendo la mayor cantidad de código posible.”[10]
Visual Studio Code: La herramienta se utilizó para desarrollar todo el código de la aplicación. El Editor de Código y Compilador Visual Studio Code es un editor de código fuente ligero pero eficaz que se ejecuta en el escritorio y está disponible para Windows, macOS y Linux. Permite al usuario depurar código directamente desde el editor definiendo puntos de interrupción, pilas de llamadas además de poseer una consola interactiva.[11]
Android Studio: “Android Studio es el entorno de desarrollo integrado (IDE) oficial que se usa en el desarrollo de apps para Android. Basado en el potente editor de código y las herramientas para desarrolladores de IntelliJ IDEA, Android Studio ofrece aún más funciones que mejoran tu productividad cuando compilas apps para Android”[12]. En Particular, se utilizó esta herramienta con el fin de acceder al apartado de creación de dispositivos de simulación para probar la aplicación en tiempo real.
API REST en Lenguaje GOLANG: Con el fin de acceder y modificar, en el caso de los guardados de los registros, la base de datos del backoffice se hizo uso de una API REST en lenguaje GOLANG. GOLANG es un lenguaje de programación de código abierto el cual es expresivo, conciso, limpio y eficiente. Es un lenguaje compilado, rápido y de tipo estático que se siente como un lenguaje interpretado y de tipo dinámico. Go compila rápidamente en código de máquina pero tiene la conveniencia de la recolección de basura y el poder de la reflexión en tiempo de ejecución.[13]
Base de Datos SQL: Se utilizó SQLite a través de un plugin que permitió a la aplicación crear una base de datos local SQL. “SQLite es una biblioteca en lenguaje C que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta confiabilidad y con todas las funciones. SQLite es el motor de base de datos más utilizado del mundo. SQLite está integrado en todos los teléfonos móviles y en la mayoría de las computadoras y viene incluido en muchas otras aplicaciones que la gente usa todos los días”.[14]

4 Resultados

A continuación, se sintetizan aspectos relacionados con la Ética Profesional y su tratamiento en el Proyecto Final de Carrera, como así también se exponen los resultados vinculados al diseño y desarrollo de la solución.

4.1 Descripción de la solución tecnológica

Investigación y Adaptación de Arquitectura Limpia

Para llevar a cabo el desarrollo de la aplicación, se realizó una exhaustiva búsqueda de una arquitectura que fuera idónea para su implementación. Se buscó que la misma cumpliera con las especificaciones necesarias para garantizar la comprensión y escalabilidad óptimas de la aplicación.
Fue así que se decidió usar Arquitectura Limpia, puesto que, debido a su estructura en capas de anillos, ofrece la posibilidad de cambiar dependencias externas o detalles de la implementación una capa en particular sin que esto afecte de alguna manera a la lógica de negocio, además de simplificar la aplicación de pruebas y la implementación de posibles nuevos requerimientos siguiendo las reglas de dependencia.
Su autor, Robert. C. Martin, presenta un modelo conceptual para lograr la separación de responsabilidades mediante capas correctamente definidas, donde el flujo de dependencias va de afuera hacia la capa más interna, no hay flujo de manera contraria; una capa interior e inferior no deberá conocer ni depender de una capa exterior.
Se puede visualizar en la Fig. 2 el modelo general utilizado para describir la Arquitectura Limpia. Este divide el proyecto normalmente en 4 capas, según el propósito y responsabilidades específicas, las cuales, ordenadas de la más interna a la más externa, son:

- Reglas de negocio de la empresa (Entidades): Representan las entidades de negocio o conceptos fundamentales que son cruciales para la aplicación. Encapsulan las reglas más generales y de alto nivel. Son los que tienen menos probabilidades de cambiar cuando algo externo cambia. Normalmente son objetos simples o un conjunto de estructura y función de los datos.
- Reglas de negocio de la aplicación (Casos de Uso): Contienen la lógica de la aplicación y definen los casos de uso específicos que pueden ocurrir en el sistema. Estos casos de uso orquestan el flujo de datos hacia y desde las entidades, y dirigen a esas entidades a utilizar sus reglas comerciales de toda la empresa para lograr los objetivos del caso de uso.
- Adaptadores de interfaz: Adapta la información entre el formato que comprende el sistema (casos de uso y entidades) y el formato que comprende el exterior del sistema, como la interfaz de usuario.
- Frameworks y controladores de dispositivos: Contiene detalles técnicos y herramientas externas, como frameworks, bases de datos, interfaces de usuario y otros dispositivos de entrada/salida.

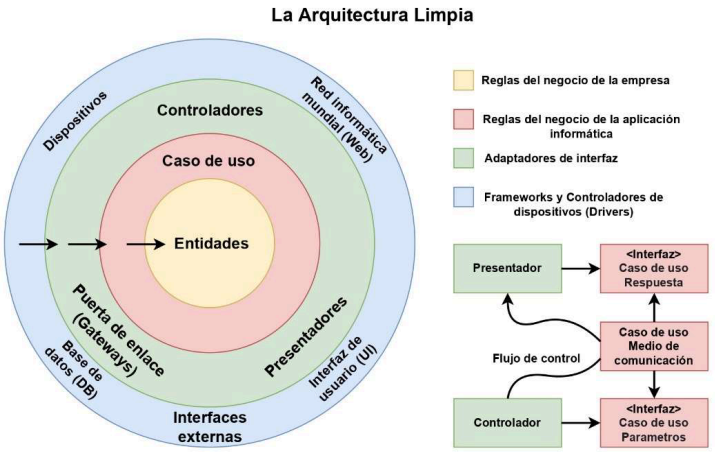

Fig. 2: Gráfico de Arquitectura Limpia según Robert. C. Martin

La regla primordial que hace que esta arquitectura funcione es la regla de dependencia. Esta regla dice que las dependencias del código fuente sólo pueden apuntar hacia adentro. Nada en un círculo interno puede saber nada acerca de algo en un círculo externo. En particular, el nombre de algo declarado en un círculo exterior no debe mencionarse en el código del círculo interior. Eso incluye funciones, clases, variables, o cualquier otra entidad de software nombrada.
Del mismo modo, los formatos de datos utilizados en un círculo exterior no deberían ser utilizados por un círculo interior, especialmente si esos formatos son generados por un marco en un círculo exterior. Es decir, se debe evitar que el contenido de un círculo exterior afecte a los círculos internos.

Ahora, conociendo más de la misma e interpretando mejor la lógica y reglas que hay que seguir para poder implementarla, se decidió adaptar la misma con el fin de poder aplicarla a la aplicación realizada en flutter con el objetivo de que sea lo más entendible posible.

Se realizó una adaptación siguiendo el modelo de la Fig. 2 y el planteado por el desarrollador de aplicaciones Flutter, "Mahmoud Elbokl", el cual plantea el uso de la misma pero con las funcionalidades generales para el desarrollo de una aplicación móvil e incluso haciendo uso de manejadores de estado [15]. La adaptación de Mahmoud se presenta en la Fig. 3, se puede observar cómo se realiza el flujo de llamadas dentro de las diferentes capas de esta arquitectura implementada en el framework Flutter.

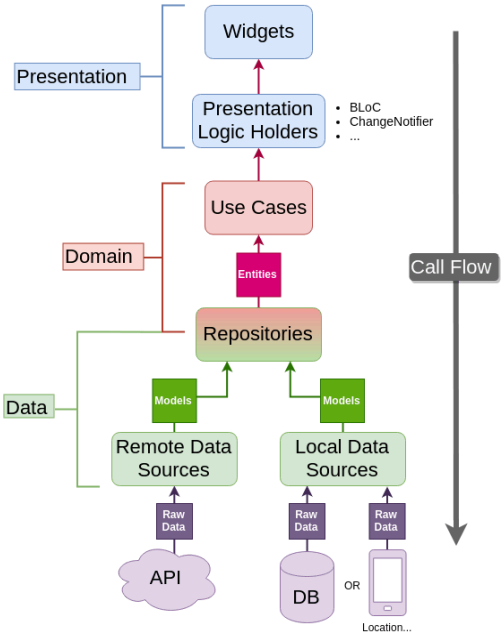

Fig. 3: Modelo del Flujo de llamadas de Clean Architecture adaptado a Flutter por Mahmoud Elbokl

En la Fig.3 se puede observar también que Mahmoud divide el proyecto de la aplicación en 3 capas en vez de 4, las cuales son:

- Capa de datos: responsable de la recuperación de datos y contiene API, almacenamiento local, objetos de datos (objetos de solicitud, respuesta y base de datos) y la implementación del repositorio.
- Capa de dominio: define la lógica empresarial de la aplicación y contiene casos de uso, entidades de dominio e interfaces de repositorio.
- Capa de Presentación: Presenta la interfaz de usuario de la aplicación y contiene pantallas, widgets y lógica de presentación.

La aplicación desarrollada hace uso de los elementos mencionados anteriormente, además del manejador de estados "BLoC", resultó más conveniente implementar este modelo de 3 capas con unos ligeros cambios en el mismo, quedando como estructura final, como se muestra en la Fig. 4. En ella se observa como el flujo de llamadas implementado es prácticamente igual al descrito por Mahmoud con la diferencia de que, en la adaptación, no se hace uso de modelos y se utilizan interfaces para los repositorios.

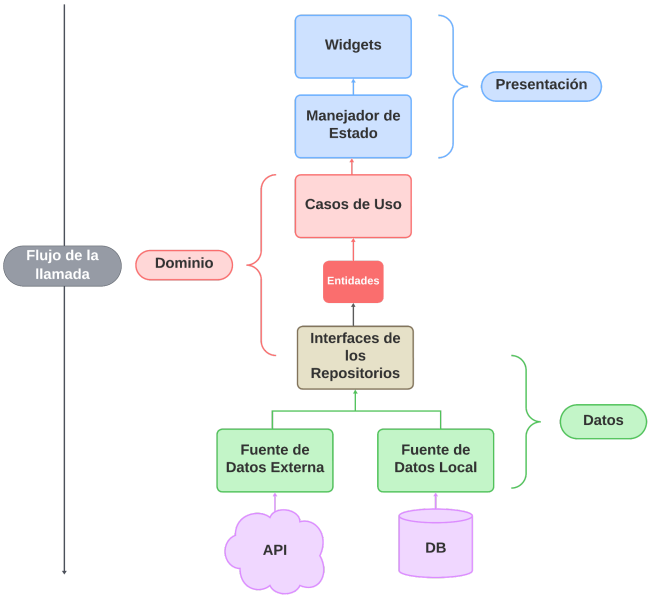

Fig. 4: Modelo adaptado del flujo de llamadas de Clean Architecture para este proyecto.

Qué implementada en directorios en el proyecto mismo se pueden visualizar en la Fig. 5 y se describen en los siguientes párrafos.

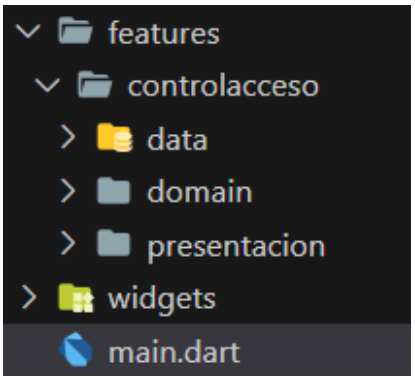

Fig. 5: Organización de los directorios del proyecto según la adaptación definida.

En primera instancia, se puede observar la carpeta features, en la cual se divide el proyecto según características o funciones principales. En este caso, solo tenemos la funcionalidad principal del control de acceso, por lo que solo tenemos un directorio dentro de la misma con toda la lógica correspondiente a esta funcionalidad.

La capa más interna, la capa de datos, contiene los archivos correspondientes a la implementación de los repositorios de datos, tanto local como remoto. Estos repositorios se encargan de conectar, el repositorio remoto con la API del sistema web y el repositorio local con la base de datos del dispositivo, con el fin de solicitar datos a los mismos. En la Fig. 6 se puede observar el directorio descrito anteriormente.

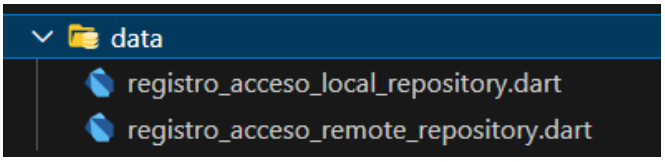

Fig. 6: Directorio “data” del Proyecto que representa la Capa de Datos.

La capa intermedia, la capa de dominio, la cual se muestra en la Fig. 7, contiene todos los archivos, separados por su responsabilidad correspondiente, que contienen los casos de uso, entidades e interfaces de los repositorios, los cuales se encargan de acceder a los repositorios correspondientes en la capa de datos. La carpeta “dtos” contiene un archivo con la definición de un modelo de respuesta para las respuestas de la api externa.

Fig. 7: Directorio “domain” del Proyecto que representa la Capa de Dominio.

Por último, la capa más externa, la capa de presentación, contiene, también separados por sus respectivas responsabilidades, todos los archivos relacionados a las vistas o interfaces de usuarios, a los manejadores de estados, o bloc’s en este caso, y a los componentes o widgets del proyecto. Este directorio se puede visualizar en la Fig. 8.

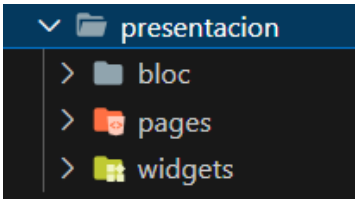

Fig. 8: Directorio “presentación” del Proyecto que representa la Capa de Presentación.

Desarrollo de la Aplicación móvil

Base de datos de la aplicación móvil

La base de datos de la aplicación se desarrolló contemplando las entidades necesarias para realizar los registros de los pases de forma offline sin tener ningún inconveniente. En la Fig. 9 se visualiza la versión final del diseño de la base de datos, la cual contiene cuatro tablas, las cuales son: registro, motivo, lugar y usuarios.

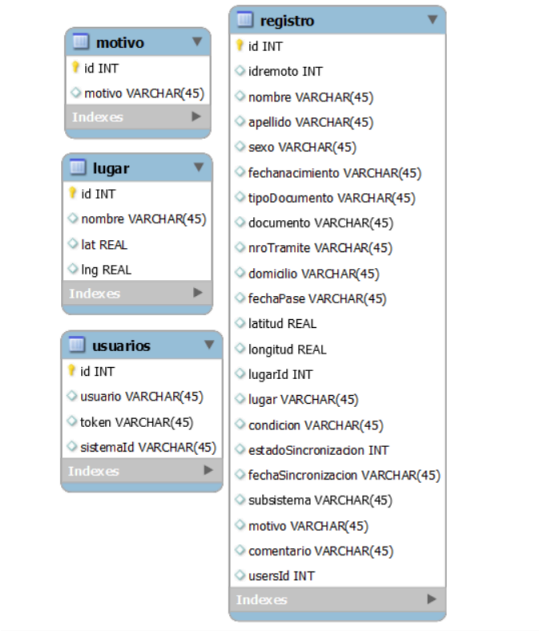

Fig. 9: Diagrama de Entidad de la Base de Datos de la aplicación.

Roles de Acceso

Para esta aplicación se especificó, un tipo de rol para tratar con todas sus funcionalidades. El rol recibe el nombre de “Usuario Flutter” y no requiere de ninguna formación o conocimiento previo. Este rol permite al usuario acceder a la aplicación por inicio de sesión y gestionar el registro y edición de pases.

Interfaces de Usuario

Módulo de Registro Manual: Apartado de la aplicación que permitirá que el usuario registre de forma manual sus datos, el lugar donde se encuentra, el motivo de su visita y alguna observación opcional (Fig. 10).

Módulo de Registro por Escaneo de DNI: Apartado de la aplicación que permitirá que el usuario registre sus datos de forma automática, teniendo que indicar solamente el lugar y motivo de su visita junto con alguna observación opcional (Fig. 11).

Fig. 10: Captura del módulo que permite registrar los datos de forma manual .

Fig. 11: Captura del módulo para escanear el código de barras del DNI con el fin de obtener los datos para autocompletar el módulo de registro.

Módulo de Carga de Subsistema: Apartado de la aplicación que permitirá elegir a un usuario entre varios subsistemas dentro del sistema general en el que se encuentra, en caso de que esta tenga más de uno (Fig. 12).

Módulo de Carga de Lugar: Apartado de la aplicación que permitirá elegir a un usuario, a partir de una lista de lugares disponibles, aquél desde el cual se encuentra para registrar los pases (Fig. 13).

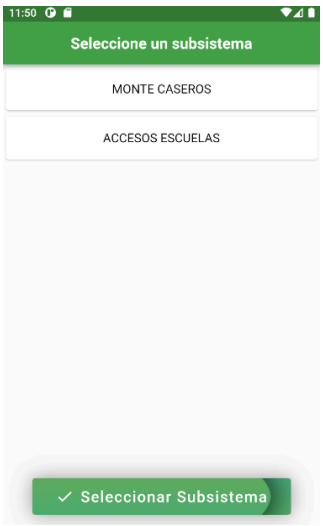

Fig. 12: Captura del módulo que lista los Subsistemas disponibles para seleccionar lugares.

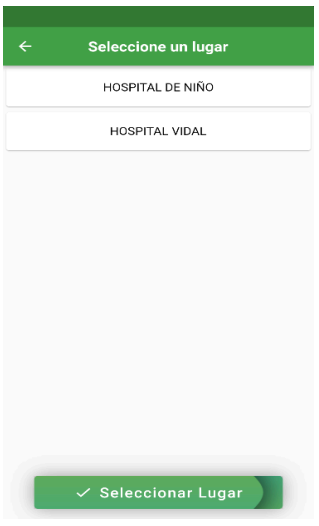

Fig. 13: Captura del módulo que lista los lugares disponibles a elegir para realizar los registros.

Módulo de Lista de Registros: Apartado de la aplicación que permitirá al usuario visualizar una lista de todos los registros que él realizó en ese dispositivo (Fig. 14).

Módulo de Edición de un Registro: Apartado de la aplicación que permitirá al usuario editar algún registro que realizó con anterioridad desde ese dispositivo (Fig. 15).

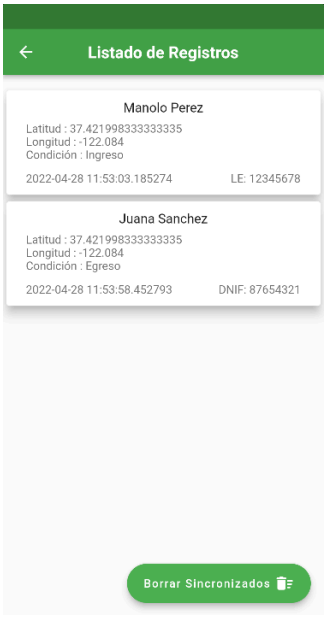

Fig. 14: Captura del módulo que lista los registros realizados en el dispositivo.

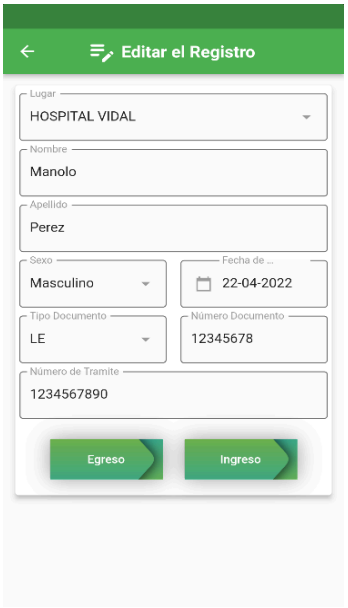

Fig. 15: Captura del módulo que permite editar un registro hecho en el dispositivo.

Módulo de Inicio de Sesión: Paquete externo que permitirá realizar todo lo relacionado a las vistas y autenticación del logueo a la aplicación (Fig. 16).

Fig. 16: Captura del módulo que permite iniciar sesión para ingresar a la aplicación .

5 Conclusiones y futuros trabajos

En este documento se pudo observar la profundización de conocimientos acerca de la Arquitectura Limpia. En específico se ahondó en una de todas las posibles aplicaciones y en cómo se puede adaptar la misma a una aplicación móvil hecha con el framework Flutter y con el manejador de estados BloC. Cómo esta arquitectura puede ser modificada tanto como sea posible para su adaptación a la conveniencia para el proyecto en cuestión, siempre y cuando se respeten los principios de la misma. Esta ofrece una gran mantenibilidad en el código ya que, aunque al principio puede ser un poco engorroso, a largo plazo, la disposición de los directorios y el tratamiento de los datos resultan ser muy eficientes para ahorrar tiempo y trabajo en el mantenimiento.
En este trabajo, se observa el desarrollo de la aplicación móvil para el registro de ingreso/egreso de personas en distintos organismos públicos de la provincia. Esta incluye, además del módulo de registro, un apartado para edición, sincronización offline y la posibilidad de poder ver y elegir de la lista de lugares disponibles para poder realizar los pases desde cualquier punto.
Una de las grandes ventajas del uso de la arquitectura se pudo observar en la realización del mantenimiento y la incorporación de nuevas funcionalidades determinadas en la retroalimentación. La estructura en capas planteada facilitó en gran medida la incorporación de los casos de uso, entidades y vistas necesarias. Permitió reducir los tiempos de implementación y así, en corto tiempo, obtener un resultado del producto final.
También se observa la ventaja del uso de esta arquitectura al realizar pruebas de funcionamiento, al seguir la regla de dependencia se facilitó identificar el camino a seguir como también el problema a solucionar.
En conclusión, la Arquitectura Limpia ofrece beneficios a largo plazo. Es adecuada para desarrollar aplicaciones móviles que sean de mantenimiento constante.

5.1 Trabajos Futuros

A futuro se prevé obtener un producto mejorado incorporando un gestor de turnos con escaneo de código QR o de tarjetas NFC para el registro del ingreso y posterior marcado del egreso al establecimiento.
Otra posible mejora del producto es la gestión de nuevos lugares o motivos directamente desde la aplicación. Este proceso incluiría agregar roles con distintos permisos de acuerdo a las funcionalidades asociadas a los usuarios.

Referencias

[1] J. P. Villón Barreno, “Desarrollo de una Aplicación Móvil dirigida al Proceso de Control de Ingresos y Egresos de Documentos para la Gestión de Archivo de la Empresa “Sacoplast S.A.” Ubicada en el Cantón Durán Km 8 Via Duran Yaguachi.”, Tesis de Grado, Facultad de Ing. Industrial, U. de Guayaquil, Guayaquil, 2022.
[2] J. C. Martínez Z, C. Henao, F. Henao, y E. Zapata, “Utilización de Arquitecturas Limpias para Trabajo con Buenas Prácticas en la Construcción de Aplicaciones Java.”, IDS, vol. 1, n.º 2, pp. 133 - 140, feb. 2021.
[3] R. C. Martin, “The Dependency Rule”, en Clean Architecture: A Craftsman's Guide to Software Structure and Design, Prentice Hall, 2017, cap. 22, pp. 193-194.
[4] S.G. Dellepiane Espinoza, H. A. Díaz Alvarado, “MYTRAININGGOAL: Facilitar la Adopción de un Estilo de Vida Sano en los Clientes del Gimnasio Personal Training, Mediante la Utilización de un Sistema Móvil con Android, Aplicando el Concepto de Clean Architecture”, Tesis, Facultad de Ing., U. Ricardo Palma, Lima, 2016.
[5] B. D. Tung, “Reactive Programming and Clean Architecture in Android Development,” Tesis, Helsinki Metropolia University of Applied Sciences, 2017.
[6] J. A. Gallardo Andrés, “Propuesta de implementación de un sistema de registro de visitas para mejorar el control de ingreso de las personas que realizan trámites en las sedes del Gobierno Regional de Lima”, Tesis de Maestría, Escuela de Maestría, U. N. José Faustino Sánchez Carrión, Lima, 2020.
[7] C. Lara, L. M. Figueroa, “Metodología ágil para el desarrollo de aplicaciones móviles educativas”, “XV Congreso Nacional de Tecnología en Educación y Educación en Tecnología”, TE&ET, Neuquén, 2020, pp. 206-213.

[8] Microsoft Learn, "¿Qué es Azure DevOps? - Azure DevOps.", Julio 2023. [Online]. Disponible: https://learn.microsoft.com/es-es/azure/devops/user-guide/what-is-azure-devops?view=azure-devops
[9] Dart, "Dart Overview.", 2023. [Online]. Disponible: https://dart.dev/overview
[10] Flutter, "Flutter architectural overview", 2023. [Online]. Disponible: https://docs.flutter.dev/resources/architectural-overview
[11] Microsoft, " Visual Studio: IDE y Editor de código para desarrolladores de software y Teams", 2024. [Online]. Disponible: https://visualstudio.microsoft.com/es/#vscode-section
[12] Android Developers, "Introducción a Android Studio", 2024. [Online]. Disponible: https://developer.android.com/studio/intro?hl=es-419&_gl=1*mi56ve*_up*MQ..&gclid=CjwKCAjw8diwBhAbEiwA7i_sJRjvPRnvbGa9AKD43dSj5JrxwTWiuS2g9wIwguBUl3VK3x4OrCitmBoCfOEQAvD_BwE&gclsrc=aw.ds
[13] The Go Programming Language, "Documentation", 2024. [Online]. Disponible: https://go.dev/doc/
[14] SQLite, "SQLite Home", 2024. [Online]. Disponible: https://www.sqlite.org/
[15] Mahmoud Elbokl, "Clean Architecture for Flutter", 2021. [Online]. Disponible: https://github.com/MahmoudElbokl/flutter_clean_arch_sample_demo?ref=flutterawesome.com

Trabajo enmarcado en actividades del Acta acuerdo de cooperación entre la empresa Corrientes Telecomunicaciones S.A.P.E.M. y la Facultad de Ciencias Exactas y Naturales y Agrimensura aprobado por Res. 0850-23 CD. Este trabajo se desarrolla en el marco del Proyecto Final de Carrera de la carrera Licenciatura en Sistemas de Información. (FaCENA - UNNE). Referentes:

- Mgter. Anita del Carmen Lopez, Prof. Orientadora del Proyecto Final de Carrera. (FaCENA - UNNE). Directora de Corrientes Telecomunicaciones S.A.P.E.M. (TelCo),
- Dra. Sonia I. Mariño, Prof. Titular Responsable de la asignatura Proyecto Final de Carrera. (FaCENA - UNNE)