UNIVERSIDAD DE TARAPACÁ

FACULTAD DE INGENIERÍA

Departamento de Ingeniería en Computación e Informática

PROYECTO IV

"Desarrollo de aplicación *EPA MÓVIL*"

2da Parte, Diseño y Modelado inicial del Sistema

Autores:	Patricio Chang Reyes Francisco Pantoja González
Nombre Empresa:	Empresa Portuaria Arica
Profesor:	Diego Aracena Pizarro

Arica, 10 de Noviembre de 2025

Índice

Índice de Figuras

1. Introducción

El presente informe tiene como fin presentar la segunda parte de diseño y modelamiento del sistema requerido por la Empresa Portuaria Arica, aplicando efectivamente los conocimientos adquiridos por el grupo de trabajo a lo largo de su formación académica.

Se espera integrar efectivamente los elementos más importantes y relevantes de la fase de planeación. Considerando cada uno de los hitos y convenciones establecidas por el grupo de trabajo en colaboración con el cliente. De manera adicional se mostrará cada uno de los nuevos elementos esperados para esta fase, considerando aspectos relevantes como las tecnologías y frameworks a utilizar, el alcance y limitaciones del producto en base a estas tecnologías, además de cada uno de los diagramas necesarios para detallar lo más prudente posible la propuesta informática. Finalmente se dará paso a mostrar resultados de las primeras iteraciones y bosquejos de código aplicados al diseño del equipo.

1.1. Consideraciones Fase Anterior

1.1.1. Objetivos General y Específicos

Objetivo General

Dar solución a una problemática real desarrollando la aplicación móvil, "EPA Móvil" , que integre información técnica, operativa y turística para mejorar el acceso a los servicios de la Empresa Portuaria Arica.

Objetivos Específicos

- Especificar el planteamiento, análisis, diseño y planificación del desarrollo de la aplicación “EPA Móvil”, definiendo el contexto, problemas, requisitos y etapas necesarias para su implementación.

- Implementar cada uno de los módulos necesarios para garantizar el correcto funcionamiento del sistema en base a sus requisitos iniciales, alcances y restricciones.

- Probar y testear las distintas iteraciones de producto de software en colaboración con el cliente, recopilando observaciones y propuestas de mejora razonables.

- Consolidar la iteración final de software para el periodo de trabajo determinado por la asignatura, poniendo el producto a disposición de la empresa y cliente.

1.1.2. Corrección a Diagrama de Casos de Uso

Se han hecho las correcciones pertinentes al diagrama de Casos de Uso, considerando las observaciones del profesor en la presentación e informe de la fase anterior. Cabe destacar que la totalidad de las descripciones de los casos de usos propuestos y obtenidos de la fase de análisis fueron aprobados por el cliente.

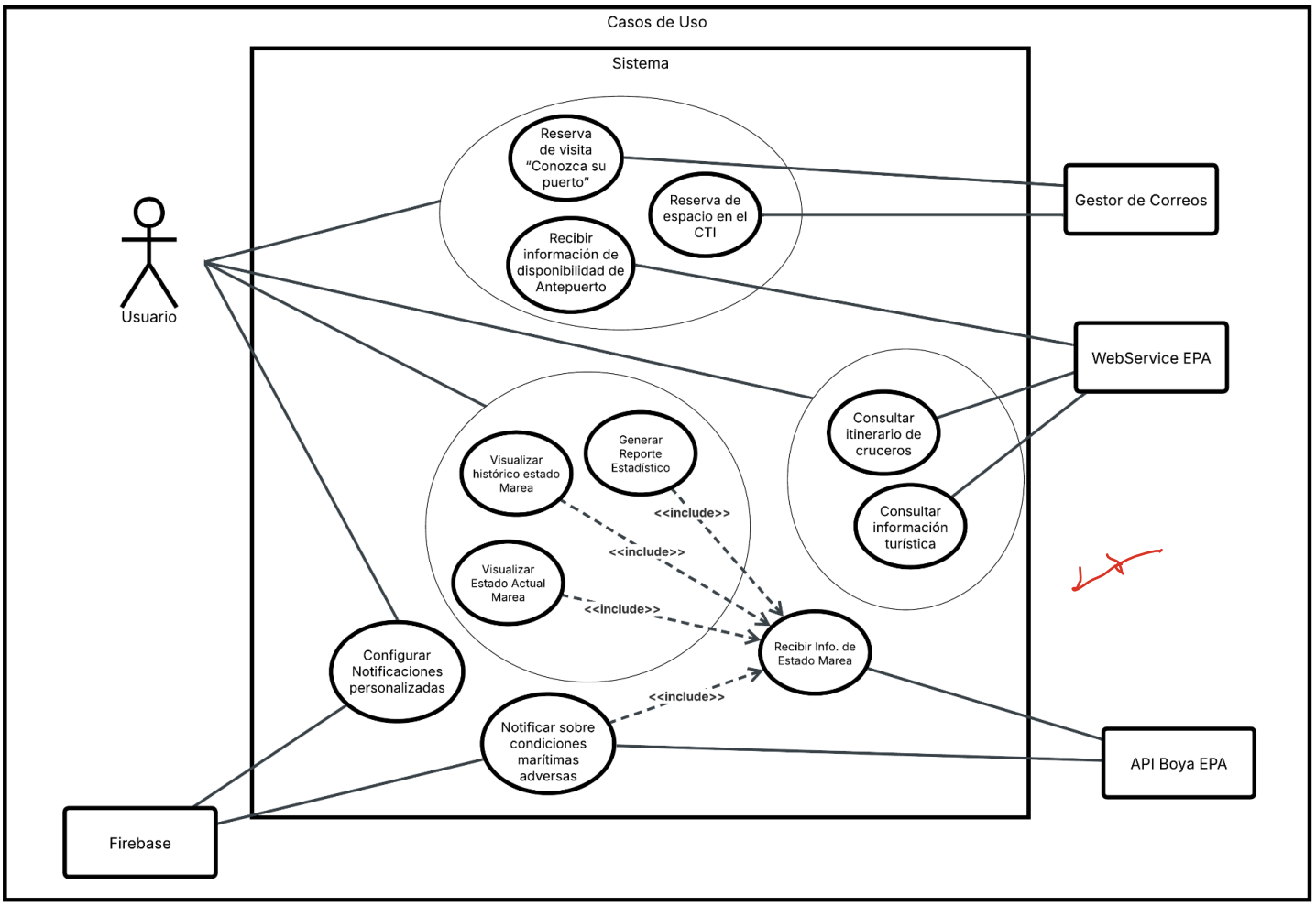


Figura 1: Diagrama de casos de uso

hay detalles de flechitas

1.3. Tecnologías a utilizar

1.3.1. Frameworks de Desarrollo

A continuación se procede a explicar los Frameworks a utilizar para cada componente del sistema.

Componente	Framework	Lenguaje Base	Razón
Aplicación Móvil	Kotlin Multiplatform Mobile (CMP/KMP)	Kotlin	Permite un único desarrollo de la lógica de la aplicación con el framework Kotlin Multiplatform, luego la interfaz de usuario utiliza CMP, basado en Jetpack Compose. El nuevo estándar de codificación de interfaces gráficas de manera declarativa
Backend Ligero (Proxy)	NestJS	Typescript	Al estar lleno de funcionalidades permite crear soluciones flexibles y a medida según los requerimientos del usuario. Además, permite la integración de otro tipo de librerías relevantes para el proyecto como el SDK de Firebase y Redis.
	Docker	—	Permitirá el despliegue rápido y compatible en todas las plataformas donde se aloje el Backend del sistema.

1.3.2. Herramientas de Desarrollo

Herramienta	Componente	Razón
Android Studio	Aplicación Móvil	Creado por Google en colaboración con Jetbrains. Es el mejor entorno de desarrollo integrado para trabajar con aplicaciones móviles. Permite acceso a emuladores, visualizador de vista previa y depurador USB y Wifi para hacer pruebas
Visual Studio Code	Backend Ligero	Es uno de los entornos más completos para trabajar con lenguajes como Typescript debido a su alta cantidad de plugins e integraciones directas a la herramienta.
Firebase	Aplicación Móvil y Backend Ligero	Permite integrar la lógica de manejo de notificaciones y llamadas especiales del servidor backend a cada usuario de la aplicación móvil, evitando pasos extra en el desarrollo y garantizando que el envío de mensajes sea seguro para cada uno de los usuarios.
UIzard	Aplicación Móvil	Es una herramienta que permite realizar Mock-ups interactivos, los cuales permitirán validar las ideas del equipo de desarrollo en colaboración al cliente.
Lucid Charts	Aplicación Móvil y Backend Ligero	Permite el desarrollo de diagramas de manera rápida y ordenada.
Github	Aplicación Móvil y Backend Ligero	Permite guardar los estados y versiones del proyecto durante todo su ciclo de vida.

1.3.3. Repositorio de Github

Actualmente el Repositorio del Proyecto está en el siguiente url: https://github.com/CHOP4NCHO/EPA_Movil/tree/ . Y actualmente se encuentra en privado por razones de seguridad.

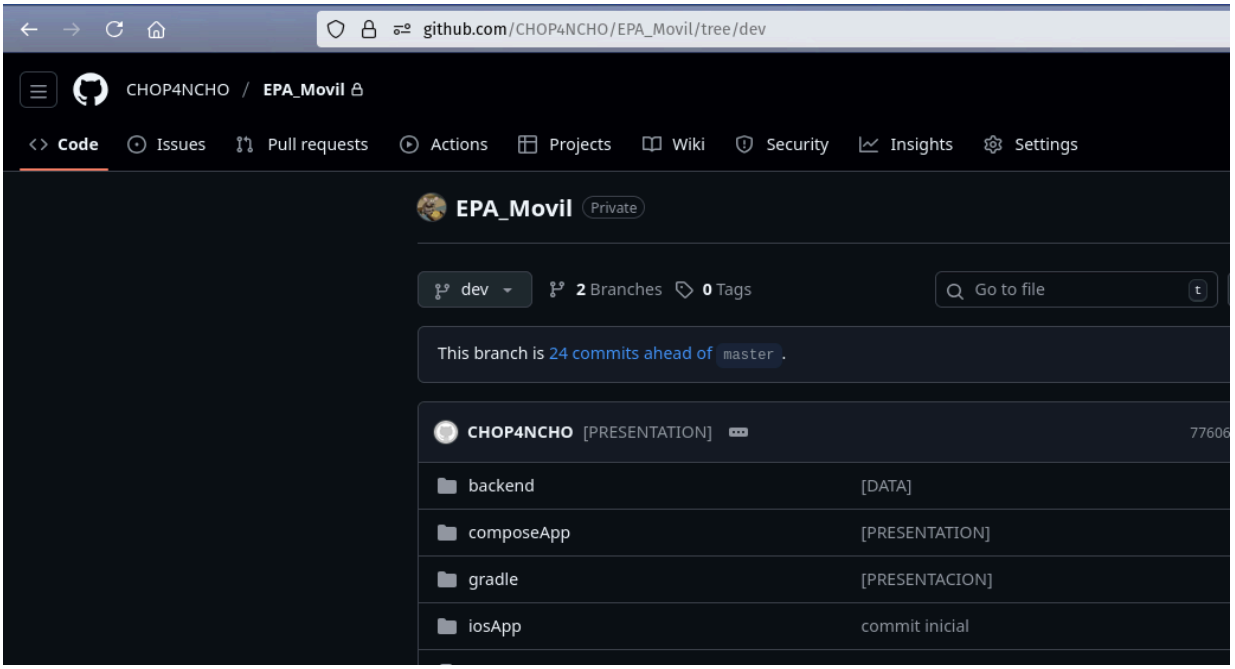


Figura 2: Repositorio en GitHub

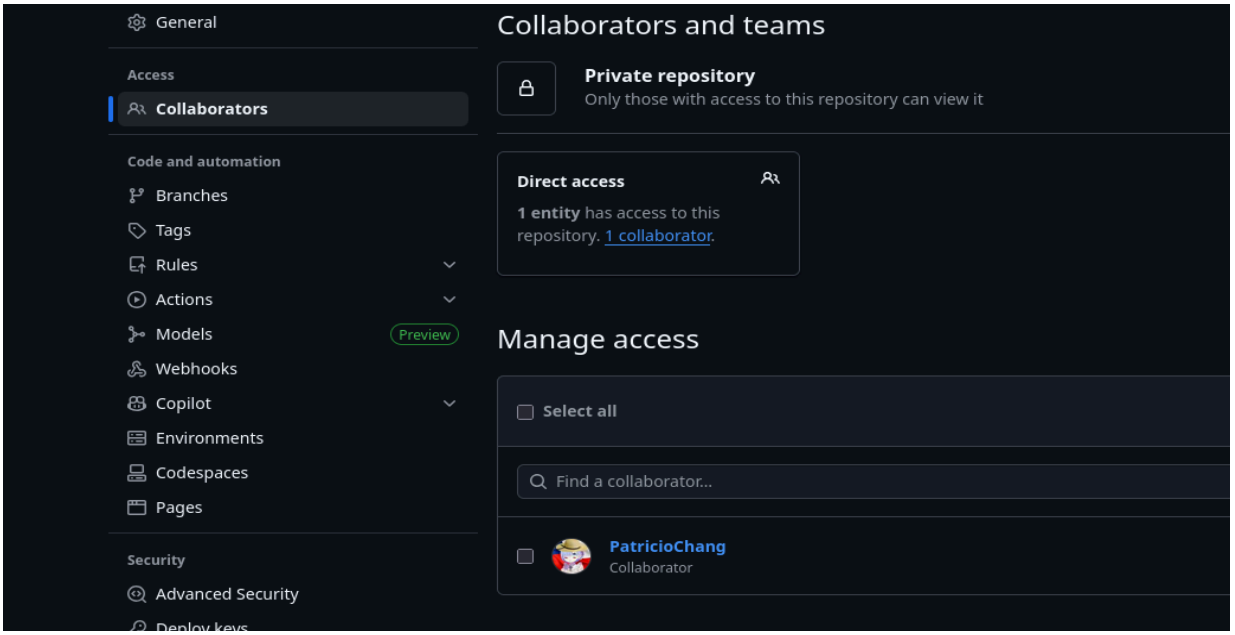


Figura 3: Colaboradores del Repositorio

1.4. Alcance del Producto con tecnologías

Una vez descritas las tecnologías de desarrollo, se espera que el producto logre cumplir con los siguientes alcances establecidos

ID	Tipo	Descripción
A1	Eficiencia	Las conexiones y comunicación entre cada uno de los componentes (app móvil - backend y viceversa) no deben exceder al mínimo de llamadas necesarias para garantizar la última información al usuario.
A2	Integración	Cada uno de los actores externos al sistema (Firebase, Gestor de Correos, Webservice EPA y la API Boya) deben ser integrados correctamente al componente de software que corresponda.
A3	Disponibilidad	La aplicación móvil debe funcionar adecuadamente tanto con como sin conexión a internet, haciendo uso del almacenamiento local. Por otro lado, el servidor debe operar correctamente incluso cuando no sea posible obtener la última información de la boya.
A4	Integridad	Cada uno de los componentes implementados de la aplicación deben coincidir con los diseños y modelamientos previamente hechos en cada una de las fases previas al desarrollo.
A5	Seguridad	El Backend debe actuar como un escudo de seguridad, centralizando y ocultando las credenciales sensibles (tokens de API Boya, credenciales SMTP) para que nunca queden expuestas en el código de la aplicación móvil.
A6	Portabilidad	El sistema Backend debe estar contenerizado mediante Docker para garantizar que pueda ser desplegado rápidamente y sin conflictos de dependencias en cualquier servidor que la Empresa Portuaria Arica disponga.

2. Desarrollo

2.1. Diagrama de Componentes

Se ha propuesto el siguiente diagrama de componentes considerando cada una de las tecnologías a utilizar.

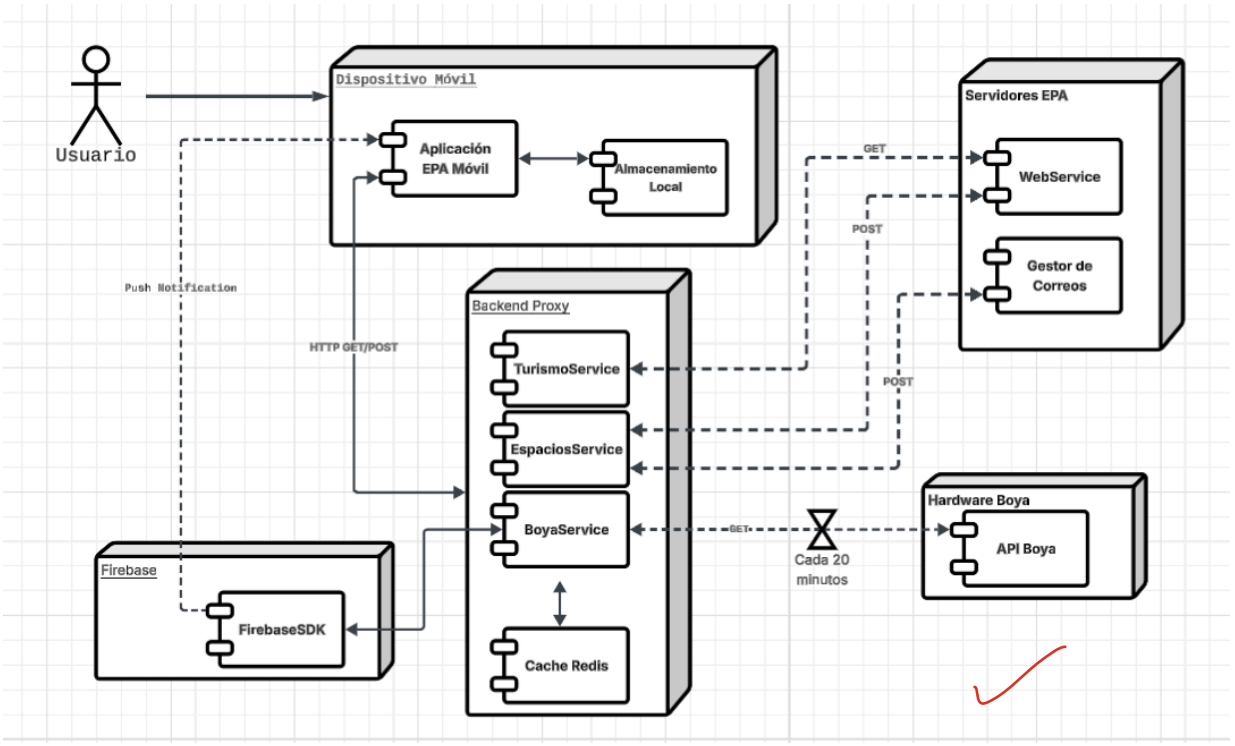


Figura 4: Diagrama de Componentes

Adicionalmente, en esta fase ya se integró un componente que no fue visto en la fase anterior el cual es Redis. Esta tecnología permitirá al servidor mantener la información necesaria para su funcionamiento en caché de manera rápida y segura.

2.2. Diagramas de Interacción y Secuencia

Para poder representar de mejor manera el comportamiento de cada uno de los subsistemas se dará paso a detallarlos por medio de diagramas de interacción.

2.2.1. Subsistema de Boya

Se ha optado por modelar el comportamiento de este subsistema mediante dos diagramas de secuencia, el primero representa el proceso de mantener actualizada y sincronizada la información del Backend ligero (que actúa como proxy) con la de la API de la Boya.

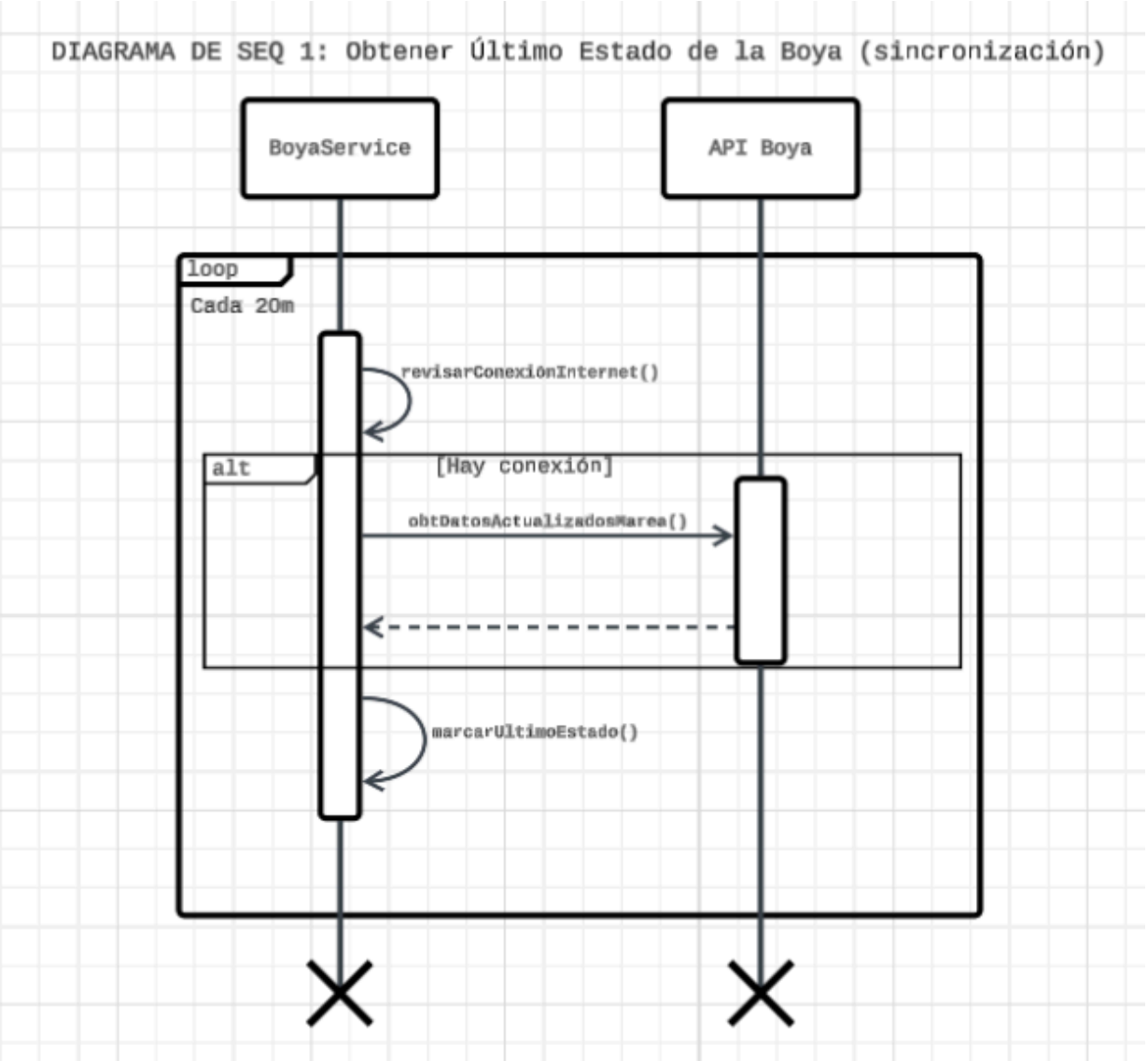


Figura 5: Diagrama de Secuencia, Obtener Último Estado de la Boya (Sincronización)

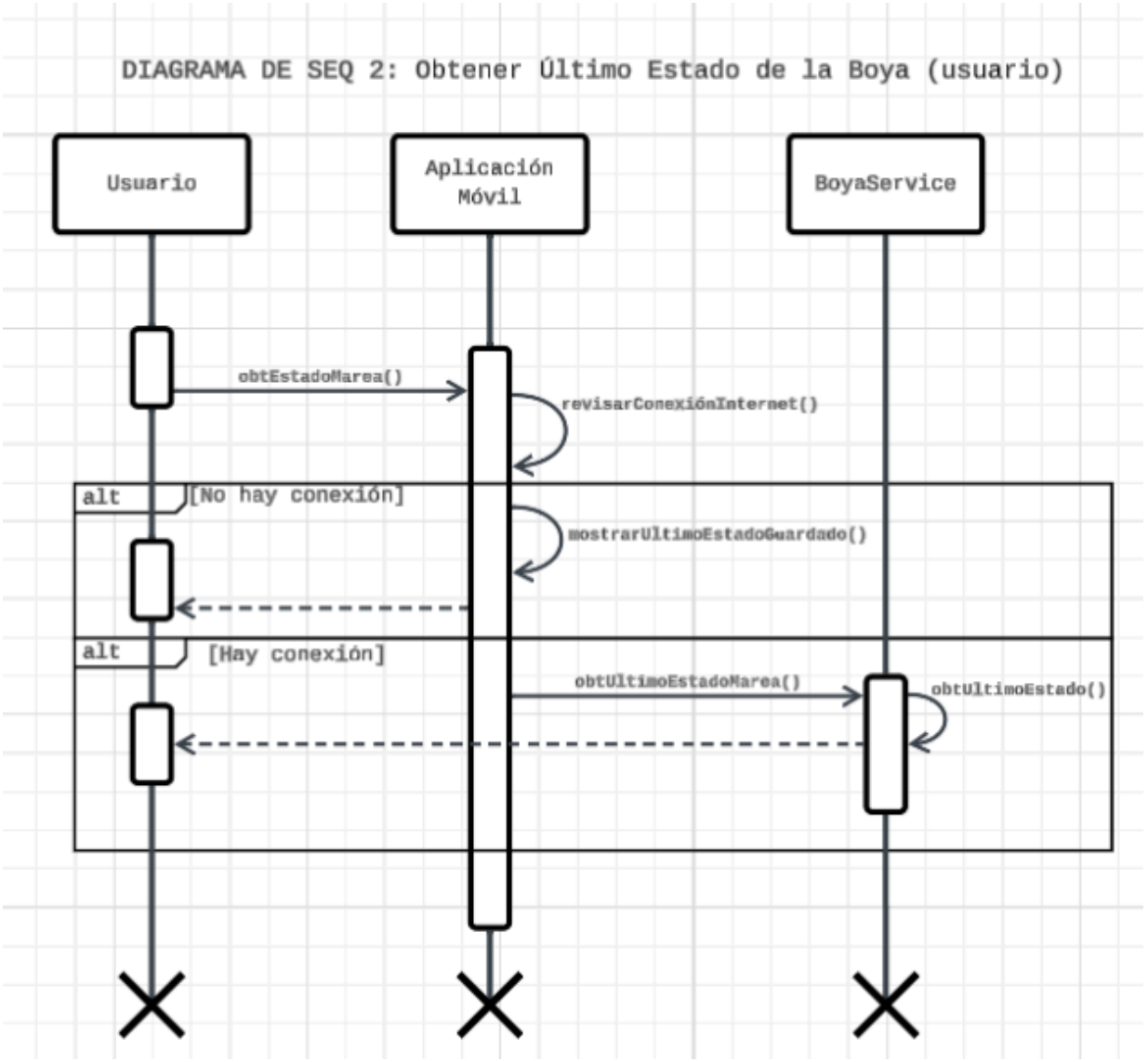


Figura 6: Diagrama de Secuencia, Obtener Último Estado de la Boya (Usuario)

Se han considerado como participantes de la interacción a BoyaService, API Boya, Aplicación Móvil y al Usuario, ya que todos ellos tienen una participación fundamental y mayoritaria en el Subsistema de Boya. Se ha dejado fuera de la secuencia a Redis, ya que su intervención en el subsistema es más auxiliar que crítica.

2.2.2. Subsistema de Turismo

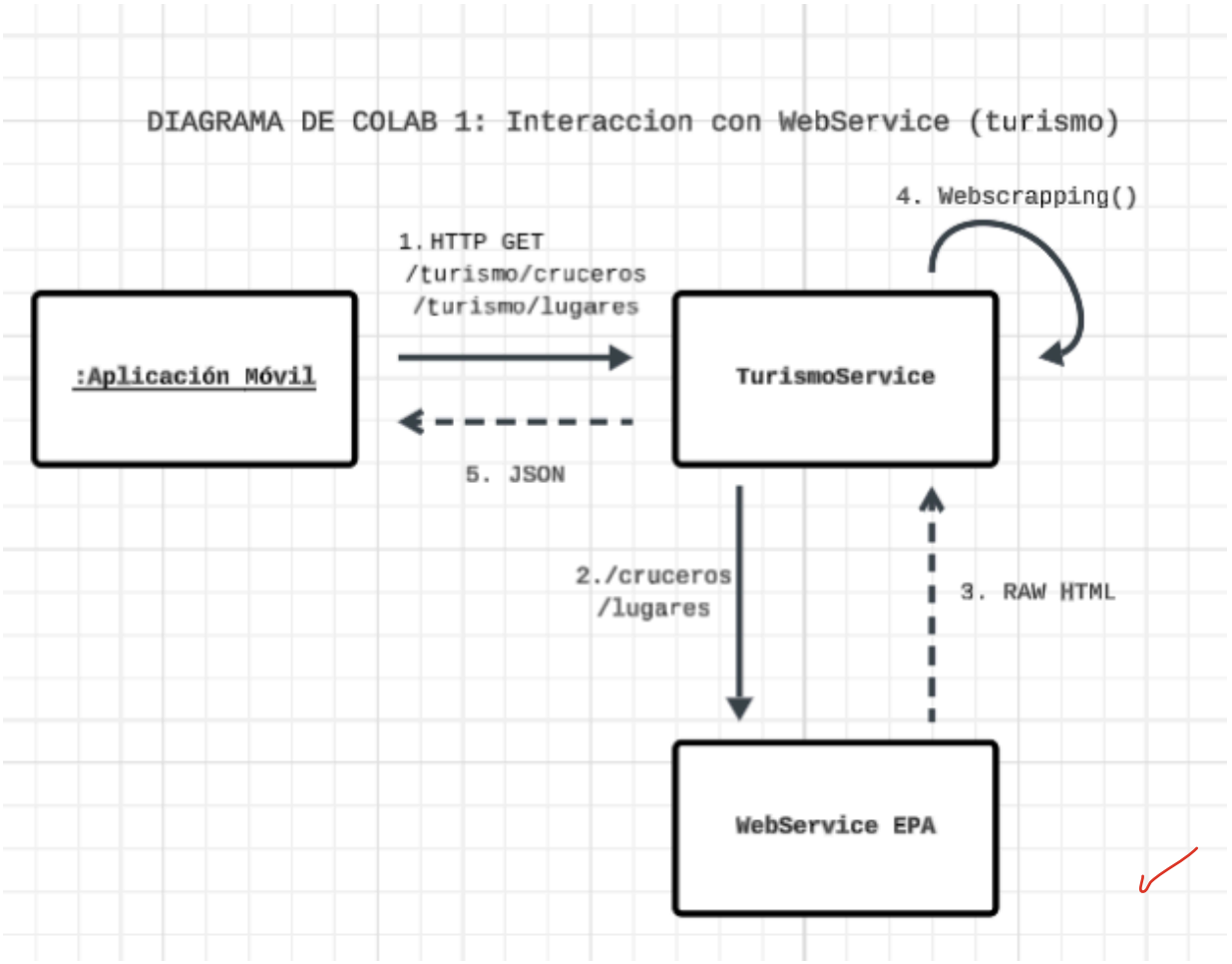


Figura 7: Diagrama de Colaboración, Interacción con WebService (Turismo)

Se consideraron los componentes de Aplicación Móvil, TurismoService y WebService EPA, ya que son los que más incidencia tienen en este subsistema. Además, el motivo de elección de un diagrama de colaboración en lugar de uno de secuencia radica en la simplicidad que este ofrece para este tipo de interacciones las cuales no tienen tantas bifurcaciones ni secuencias complejas.

2.2.3. Subsistema de Espacios

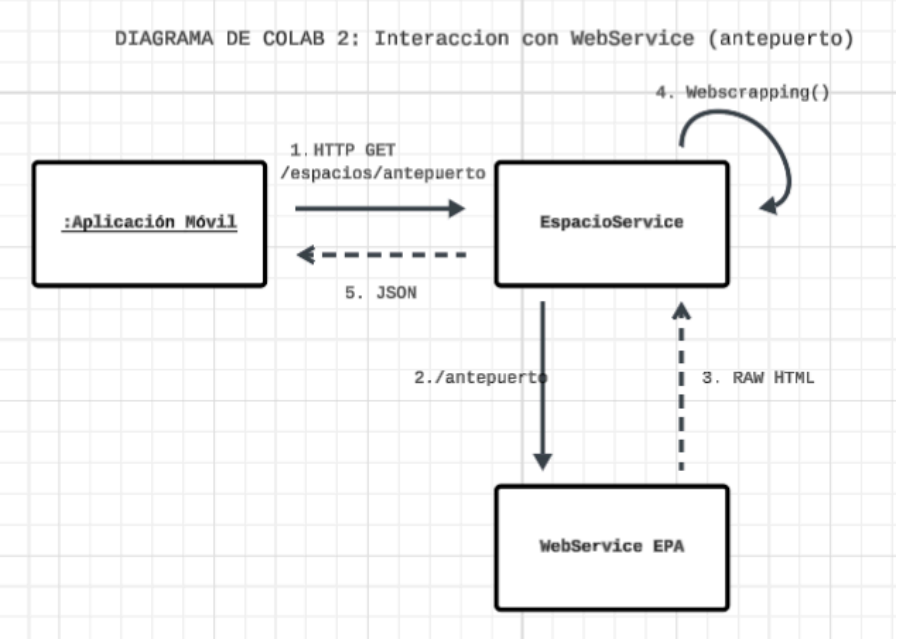


Figura 8: Diagrama de Colaboración, Interacción con WebService (AntePuerto)

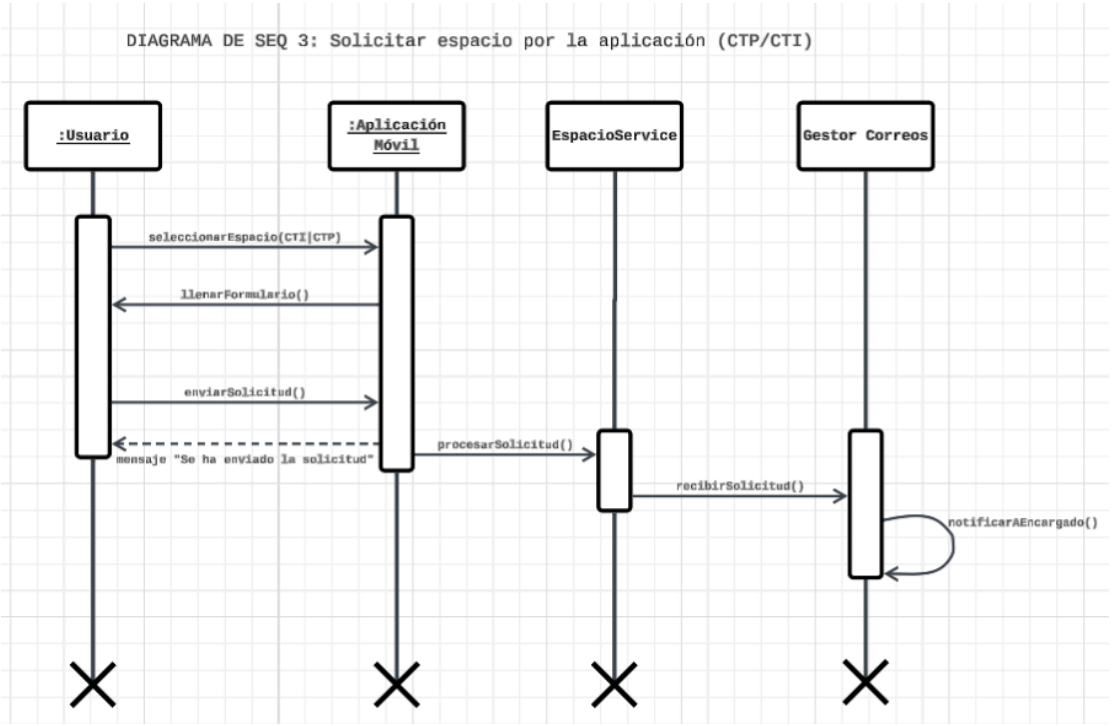


Figura 9: Diagrama de Secuencia, Solicitar Espacio por la Aplicación (CTP/CTI)

En este subsistema se han considerado un diagrama de colaboración y otro de secuencia, separando los dos tipos de interacciones que ocurren en este subsistema, uno asociado al proceso de revisar el estado del antepuerto (muy similar al subsistema anterior) y la interacción asociada al manejo de los formularios de solicitud en el CTI y el programa Conozca su Puerto.

2.3. Modelamiento de la capa de datos

A pesar de que la propuesta informática no contempla la creación de una base de datos relacional, sí es necesario modelar el comportamiento de la capa de datos. Esta capa contempla la comunicación a bajo nivel de la aplicación móvil con el Backend ligero.

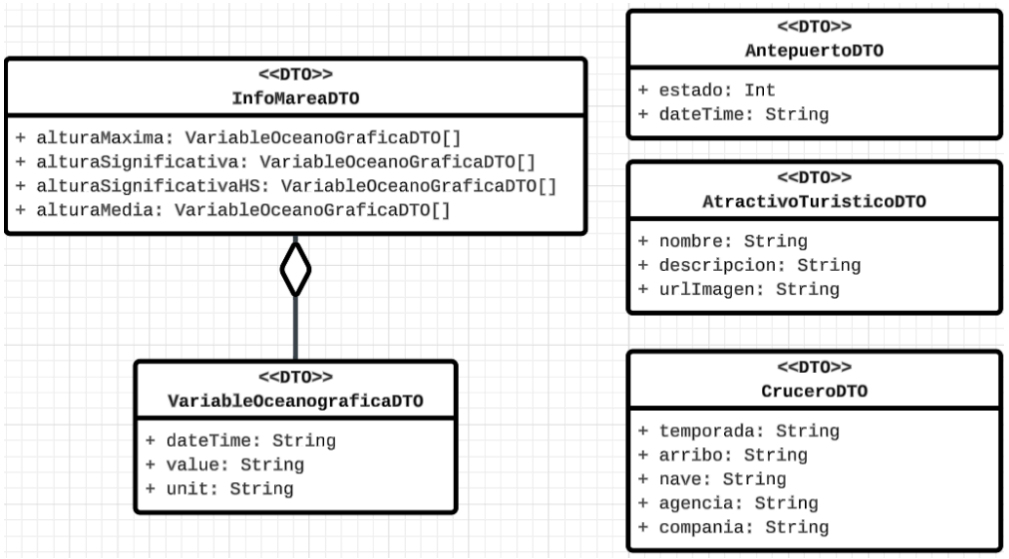


Figura 10: Capa de Datos (DTOs)

El objetivo de estos DTOs es establecer el contrato que luego tanto la aplicación como el backend se encargarán de tratar internamente. La notación intermedio para tratar con estos datos es JSON

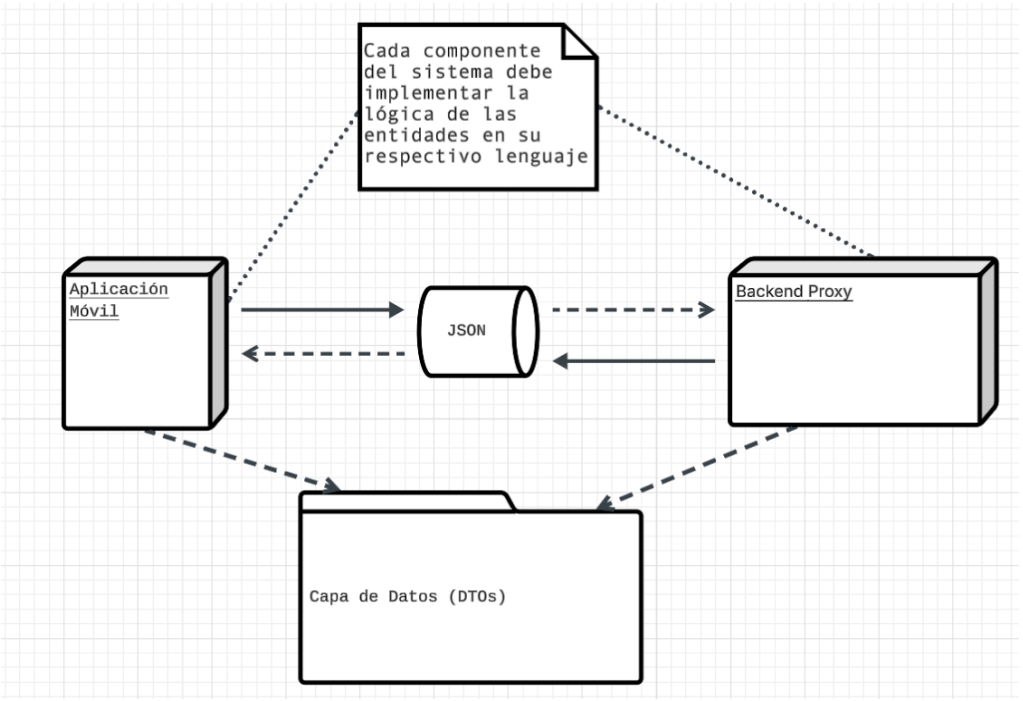


Figura 11: Integración de Capa de Datos en Comunicación de Componentes

Por otro lado. Para consumir la información de la API de la Boya, es necesario que el Backend mapee internamente los datos del formato proveniente de la API al de la capa de datos propuesta. Actualmente esta es la descripción de los datos más relevantes provenientes de la API (Por asuntos de confidencialidad, no se puede especificar todo el documento):

	Nombre variable	Definición	Unidad	Umbrales	Valores nulos
SENSOR DE OLEAJE	VMXL	Altura máxima	cm	0 a 3000	65535
	VH110	Altura significativa 1/10	cm	0 a 3000	65535
	VAVH	Altura significativa (Hs) 1/3	cm	0 a 3000	65535
	VHM0	Altura media	cm	0 a 3000	65535
	VTMX	Período máximo	seg	1 a 100.0	6553.5
	VTPK	Período ola pico	seg	1 a 100.0	6553.5
	VT110	Período ola significativa 1/10	seg	1 a 100.0	6553.5
	VAVT	Período ola significativa (Hs) 1/3	seg	1 a 100.0	6553.5
	VGTA	Período medio	seg	1 a 100.0	6553.5
	VDIR	Dirección del oleaje	º	0 a 359.9	6553.5
	VPSP	Dispersión del oleaje	º	0 a 359.9	6553.5
	VQTY	Cantidad de olas	Cantidad	1 a 1000	65535

Figura 12: Definición de Variables de la API

2.4. Arquitectura de Software (Aplicación Móvil)

Ya que la aplicación móvil es más crítica en cuanto a mantenimiento. Se consideró crucial separar cada uno de los componentes internos de la Aplicación en Capas. Esto se hizo siguiendo las convenciones establecidas por el patrón/filosofía de *Clean Architecture*, el cual se basa en los principios *SOLID* para estructurar un proyecto de software de la manera más flexible y mantenible posible.

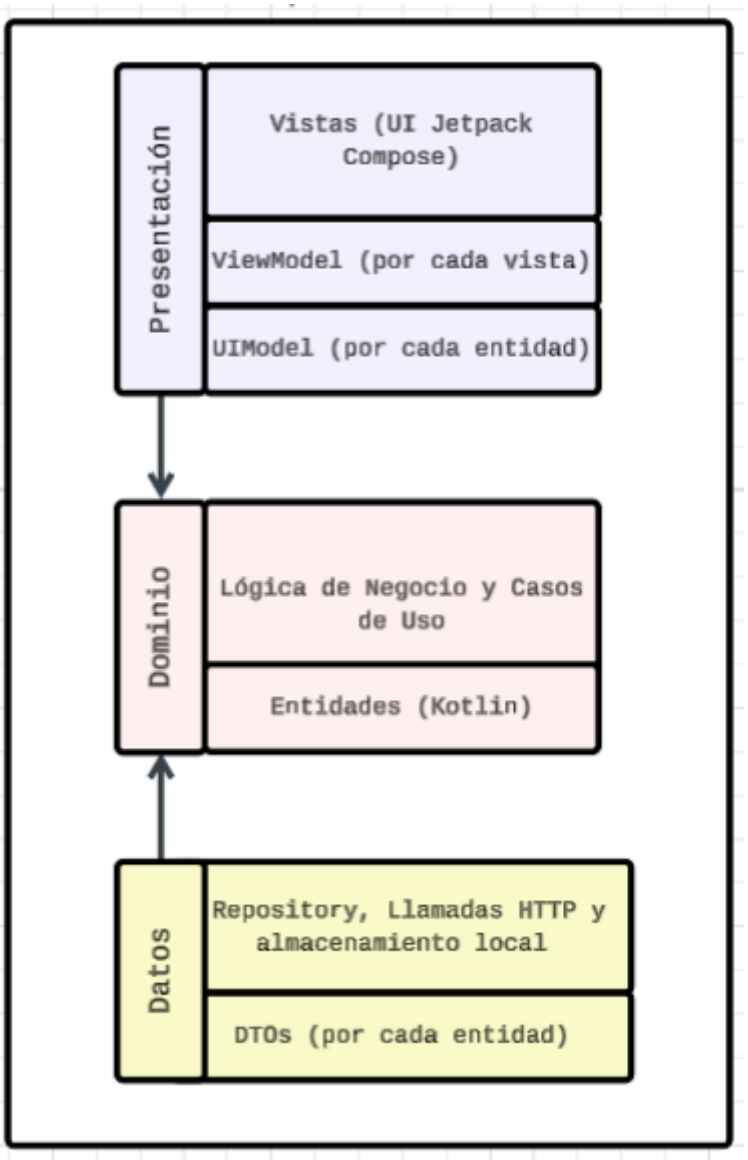


Figura 13: Arquitectura de Software del Sistema (App Móvil)

Es importante resaltar que la Capa de Dominio no depende de ninguna librería asociada a las tecnologías y frameworks que se utilizan en el desarrollo. Es decir, es una abstracción pura utilizando el lenguaje de programación en el cual esté siendo desarrollado el proyecto (en este caso, Kotlin). Por otro lado, la Capa de Presentación implementa internamente la arquitectura de Presentación MVVM, y la Capa de Datos (de la aplicación) se conecta directamente a la Capa de datos establecida por el equipo de trabajo.

2.5. Implementación y Refinamiento sucesivo

El propósito de esta implementación temprana es preparar la base sobre la cual se trabajará durante la fase 3. Para adelantar a la fase 3 (implementación) se ha comenzado con el desarrollo de los aspectos más importantes de la aplicación, como lo son las pantallas principales de la misma, además de asegurar la conexión a la API de la Boya. Actualmente no se cuenta con una integración directa entre ambos componentes, pero sí es posible asegurar que están preparados para establecer la conexión entre ellos sobre los distintos CUS propuestos.

2.5.1. Avances en Aplicación Móvil

Se comenzó estructurando el proyecto en cada una de las capas establecidas por la arquitectura de software de la aplicación. Esto incluye las capas de Presentación, Dominio y Datos.

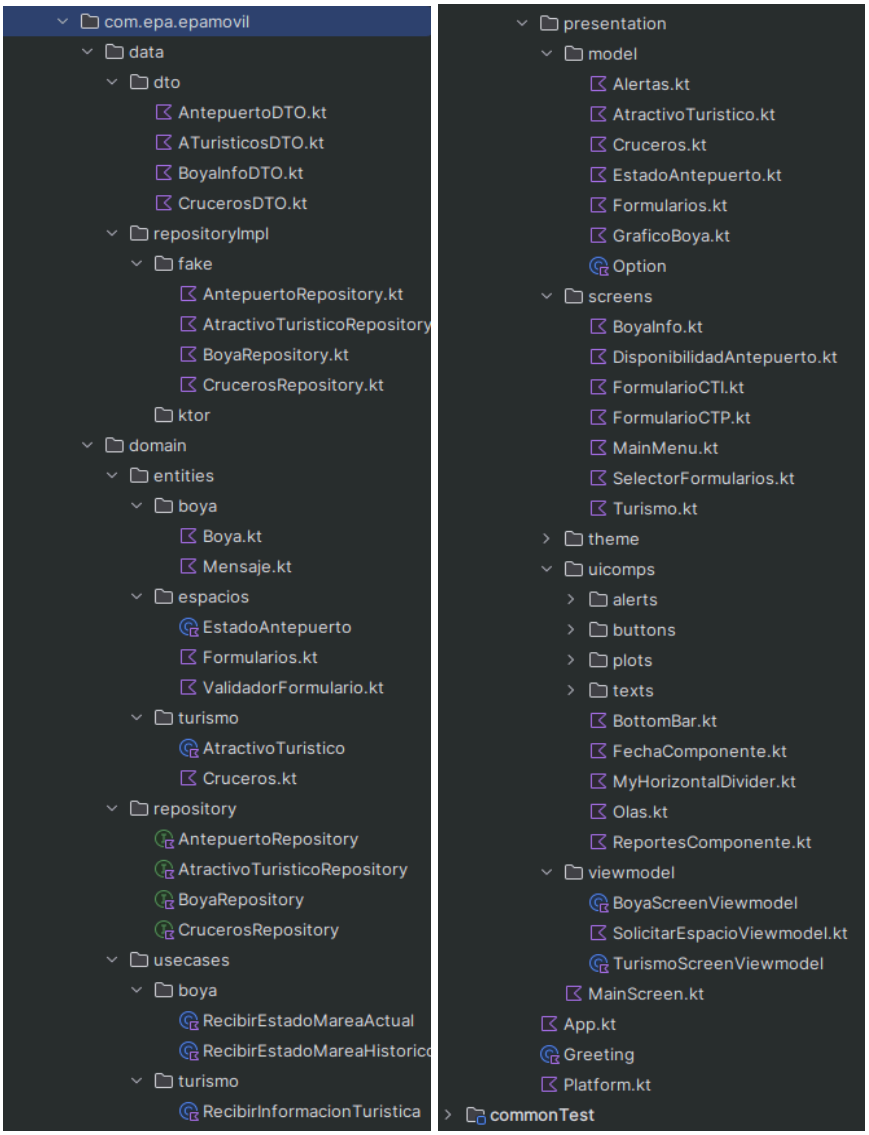


Figura 14: Avances Aplicación Móvil (1)

Refiriéndose al directorio data, este es el que implementa la capa de datos que se conectará con el Backend, por ello todas las declaraciones de data-clases son de tipo @Serializable, lo que les permite ser luego transformadas a JSON con mayor facilidad.

```kotlin
package com.epa.epamovil.data.dto

import ...

3 Usages
@Serializable
data class ATuristicosDTO (
    val atractivosTuristicos: List<AtractivoTuristicoDTO>
)

8 Usages
@Serializable
data class AtractivoTuristicoDTO (
    val nombre: String,
    val descripcion: String,
    val urlImagen: String
)

3 Usages
fun AtractivoTuristicoDTO.toAtractivoTuristico(): AtractivoTuristico {
    return AtractivoTuristico(
        nombre = nombre,
        descripcion = descripcion,
        imgReferencia = urlImagen
    )
}
```

Figura 15: Avances Aplicación Móvil (2)

Adicionalmente, se cuenta con métodos “mapper” para traspasar esta información en crudo (DTO) a una clase de la capa de dominio. Estos métodos son llamados dentro de las clases Repository, las cuales sirven como abstracciones para acceder a cada uno de los datos desde la capa de dominio. Actualmente se codificaron repositorios de prueba con el prefijo Fake, pero se espera que estos datos provengan de otras fuentes, como en este proyecto, llamadas Http.

```kotlin
package com.epa.epamovil.data.repositoryImpl.fake

import ...


class FakeAntepuertoRepository: AntepuertoRepository {
    2 Usages
    val antepuertos = listOf(
        AntepuertoDTO(
            estado = 1,
            dateTime = "2025-10-30T08:00:00"
        ),
        AntepuertoDTO(
            estado = 0,
            dateTime = "2025-10-29T18:30:00"
        ),
        AntepuertoDTO(
            estado = 1,
            dateTime = "2025-10-28T22:15:00"
        ),
        AntepuertoDTO(
            estado = 1,
            dateTime = "2025-10-27T07:45:00"
        ),
        AntepuertoDTO(
            estado = 0,
            dateTime = "2025-10-26T16:10:00"
        )
    )


    override fun getUltimoEstado(): EstadoAntepuerto {
        return antepuertos.last().toEstadoAntepuerto()
    }

    override fun getHistorico(): List<EstadoAntepuerto> {
        return antepuertos.map { it.toEstadoAntepuerto() }
    }


}
```

Figura 16: Avances Aplicación Móvil (3)

La capa de dominio contiene las entidades del negocio ya mapeadas al lenguaje Kotlin, además de casos de uso que implementan la lógica de los CUS dentro de la aplicación.

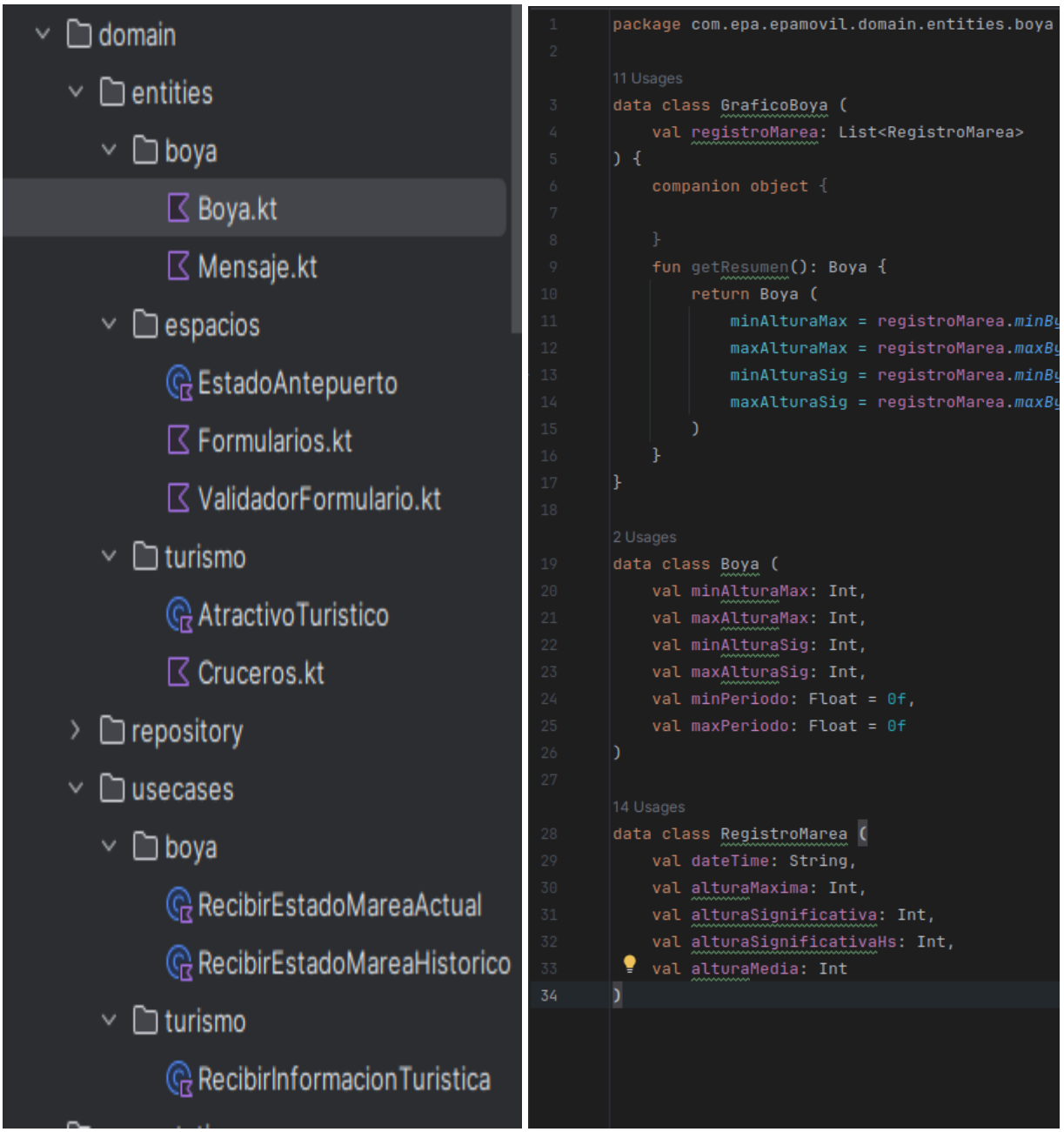
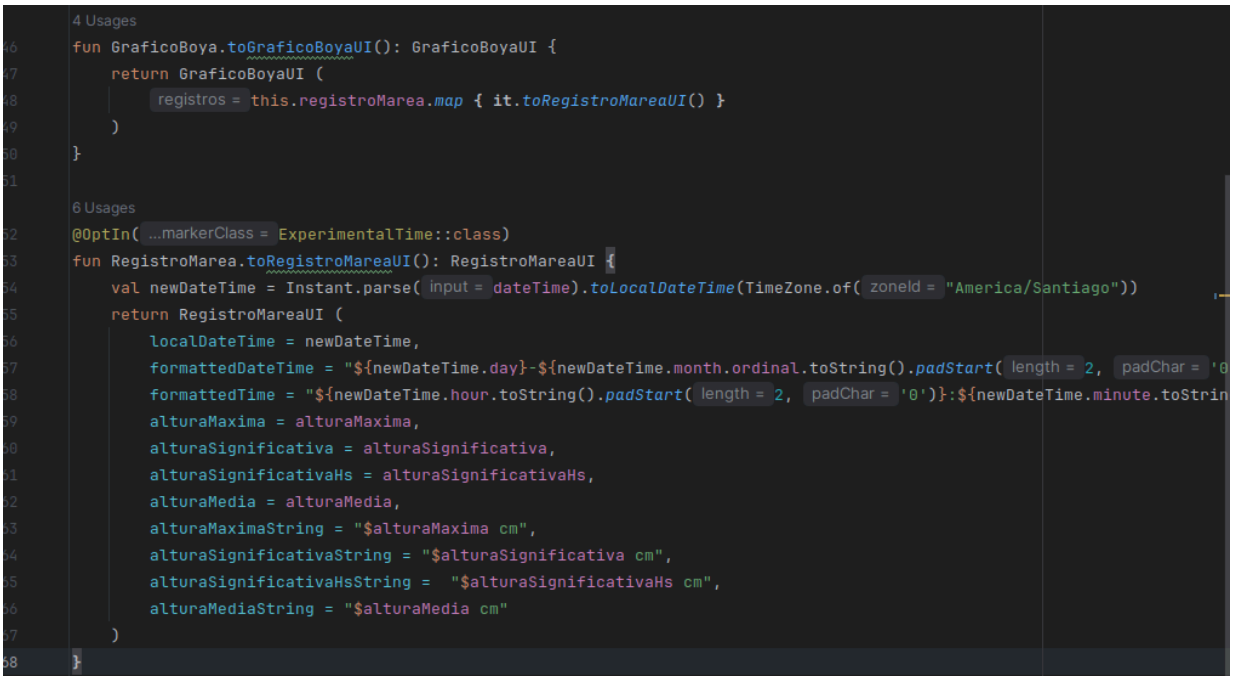


Figura 17: Avances Aplicación Móvil (4)

Finalmente, la capa de presentación contiene los modelos basados en las entidades del dominio en un formato más legible y procesable por la UI. En el caso especial del desarrollo de aplicaciones con Kotlin, la librería que maneja las fechas es externa. Por ello sólamente en esta capa se manejan datos en formato de fecha (en lugar de tratarlos como string).

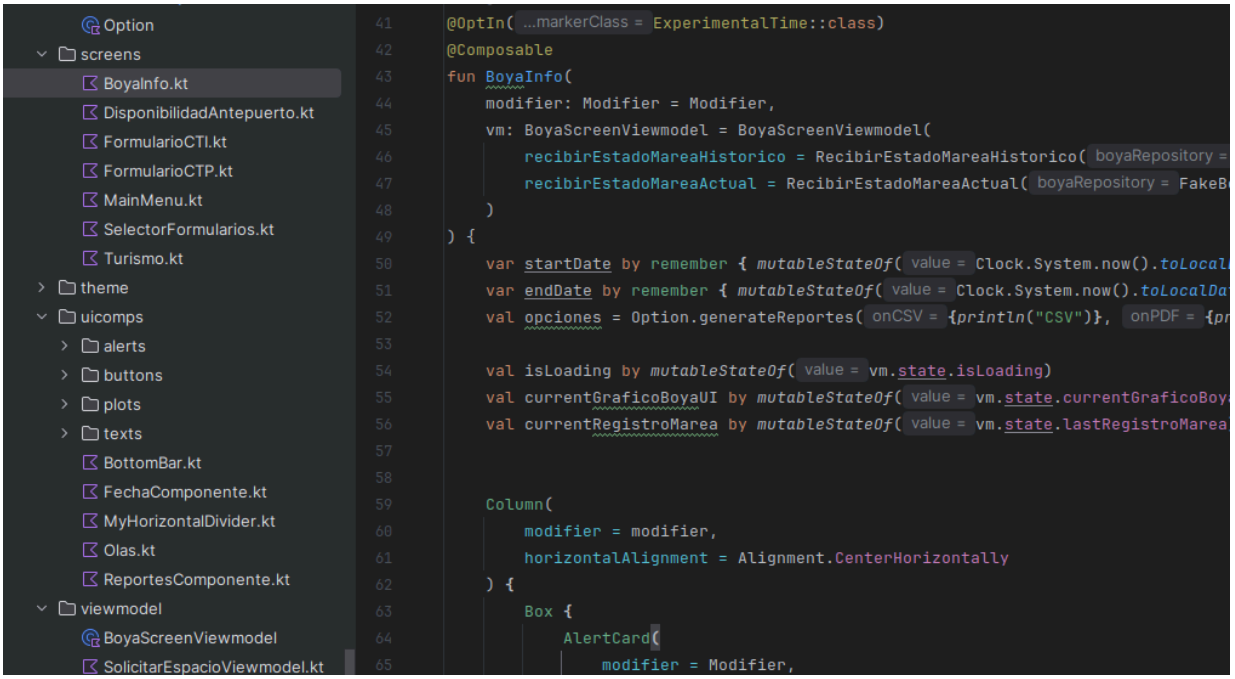


```
4 Usages
fun GraficoBoya.toGraficoBoyaUI(): GraficoBoyaUI {
    return GraficoBoyaUI (
        registros = this.registroMarea.map { it.toRegistroMareaUI() }
    )
}

6 Usages
@OptIn( ...markerClass = ExperimentalTime::class)
fun RegistroMarea.toRegistroMareaUI(): RegistroMareaUI {
    val newDateTime = Instant.parse( input = dateTime).toLocalDateTime(TimeZone.of( zoneId = "America/Santiago"))
    return RegistroMareaUI (
        localDateTime = newDateTime,
        formattedDateTime = "${newDateTime.day}-${newDateTime.month.ordinal.toString().padStart( length = 2,  padChar = '0
        formattedTime = "${newDateTime.hour.toString().padStart( length = 2,  padChar = '0')}:${newDateTime.minute.toStrin
        alturaMaxima = alturaMaxima,
        alturaSignificativa = alturaSignificativa,
        alturaSignificativaHs = alturaSignificativaHs,
        alturaMedia = alturaMedia,
        alturaMaximaString = "$alturaMaxima cm",
        alturaSignificativaString = "$alturaSignificativa cm",
        alturaSignificativaHsString =  "$alturaSignificativaHs cm",
        alturaMediaString = "$alturaMedia cm"
    )
}
```

Figura 18: Avances Aplicación Móvil (5)

Para finalmente pasar a las vistas, cada una de ellas son de tipo @Composable. Las interacciones y datos mutables dentro de las vistas son tratadas por los Viewmodel.



```
Option
screens
  BoyaInfo.kt
  DisponibilidadAntepuerto.kt
  FormularioCTI.kt
  FormularioCTP.kt
  MainMenu.kt
  SelectorFormularios.kt
  Turismo.kt
theme
uicomps
  alerts
  buttons
  plots
  texts
  BottomBar.kt
  FechaComponente.kt
  MyHorizontalDivider.kt
  Olas.kt
  ReportesComponente.kt
viewmodel
  BoyaScreenViewmodel
  SolicitarEspacioViewmodel.kt
```

```
@OptIn( ...markerClass = ExperimentalTime::class)
@Composable
fun BoyaInfo(
    modifier: Modifier = Modifier,
    vm: BoyaScreenViewmodel = BoyaScreenViewmodel(
        recibirEstadoMareaHistorico = RecibirEstadoMareaHistorico( boyaRepository =
        recibirEstadoMareaActual = RecibirEstadoMareaActual( boyaRepository = FakeB
    )
) {
    var startDate by remember { mutableStateOf( value = Clock.System.now().toLocal
    var endDate by remember { mutableStateOf( value = Clock.System.now().toLocalDa
    val opciones = Option.generateReportes( onCSV = {println("CSV")}, onPDF = {pr

    val isLoading by mutableStateOf( value = vm.state.isLoading)
    val currentGraficoBoyaUI by mutableStateOf( value = vm.state.currentGraficoBoy
    val currentRegistroMarea by mutableStateOf( value = vm.state.lastRegistroMarea


    Column(
        modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Box {
            AlertCard(
                modifier = Modifier,
```

Figura 19: Avances Aplicación Móvil (6)

Figura 20: Avances Aplicación Móvil (7)

2.5.2. Avances en Backend Ligero

Se ha creado un proyecto de NestJS con los tres componentes principales en formato de directorios. Además de los archivos Dockerfile y docker-compose-yml para su rápido despliegue.

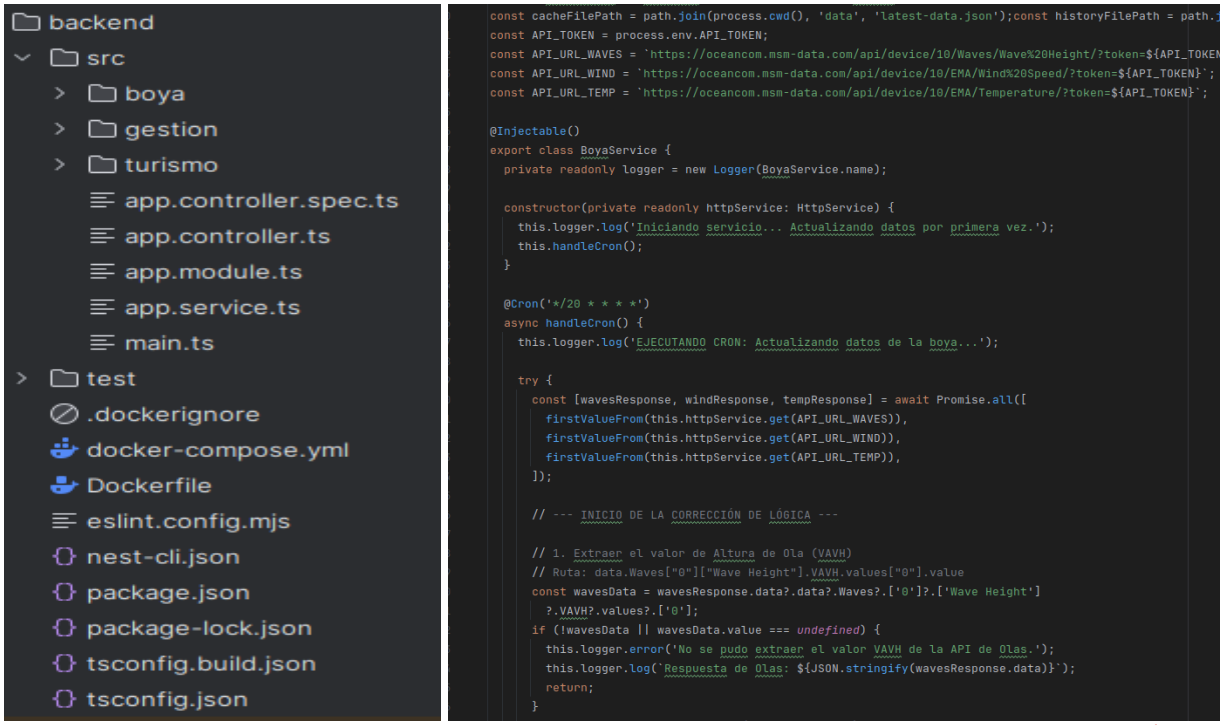


Figura 21: Avances Backend Ligero (8)

Se han implementado algunas tareas de tipo Cron para simular el llamado a la API de la Boya cada 20 minutos. Actualmente se tiene abierta la ruta /boya/actual, la que será llamada constantemente desde la aplicación móvil. Cabe destacar que la información devuelta por el Backend es cacheada previamente, por lo que la API de la Boya no se satura con llamadas de pruebas como lo estableció el grupo de trabajo.

Figura 22: Avances Backend Ligero (9)

2.5.3. Avances con planificación ScrumBan

Para la gestión de la fase de implementación y el refinamiento sucesivo, el equipo ha adoptado la metodología híbrida ScrumBan. Esta combina la estructura de planificación iterativa de Scrum (necesaria para cumplir con los hitos académicos) con la visualización de flujo continuo de Kanban (ideal para el trabajo de mantenimiento y pequeñas tareas evolutivas).

Se ha dispuesto un tablero de trabajo que permite visualizar el estado actual del desarrollo, limitar el trabajo en progreso para evitar cuellos de botella entre el desarrollo móvil y backend, y asegurar que las tareas críticas, como la conexión a la API de la Boya, se completen prioritariamente.

El tablero se estructura en las siguientes columnas:

- Product Backlog: Lista maestra de todos los requisitos pendientes (derivados de los Casos de Uso).
- Sprint Backlog (Fase 3): Tareas seleccionadas para la próxima entrega de implementación.
- In Progress (Backend/Mobile): Tareas que están siendo activamente desarrolladas. Se separa para visualizar si un área está bloqueando a la otra.
- Testing/QA: Tareas terminadas que requieren validación con los datos reales de la boya o pruebas de integración.
- Done: Tareas completadas y validadas funcionalmente.

Actualmente, se puede apreciar que las tareas fundamentales de arquitectura (Docker, estructura NestJS y KMP) ya se encuentran en la columna "Done", dando paso a las tareas de integración de servicios.

Figura 23: Avances con Planificación ScrumBan

3. Conclusiones

La segunda fase del proyecto "EPA Móvil" ha finalizado exitosamente, validando las decisiones arquitectónicas clave. La implementación de un Backend Proxy ha demostrado ser eficaz para centralizar la seguridad y optimizar la gestión de datos mediante caché, mientras que el diseño móvil basado en Clean Architecture asegura una base escalable para el desarrollo futuro.

Un logro fundamental de esta etapa ha sido la mitigación temprana del riesgo técnico más crítico: la conexión funcional con la API de la Boya Oceanográfica mediante tareas programadas. Con estos cimientos operativos y la adopción de la metodología ScrumBan, el equipo se encuentra en una posición óptima para afrontar la fase final de implementación e integración.

Obs:
Buen informe, con bastante avance, hay detalles, y conclusiones cortas
acostúmbrense a decir el trabajo futuro a seguir..
revisen los sentidos de las flechitas de su informe..

4. Referencias

[1] JetBrains, "Compose Multiplatform," jetbrains.com. [Online]. Disponible: https://www.jetbrains.com/compose-multiplatform/. [Accedido: 10-Nov-2025].

[2] JetBrains, "Kotlin Programming Language," kotlinlang.org. [Online]. Disponible: https://kotlinlang.org/. [Accedido: 10-Nov-2025].

[3] NestJS, "Documentation | NestJS - A progressive Node.js framework," nestjs.com. [Online]. Disponible: https://docs.nestjs.com/. [Accedido: 10-Nov-2025].

[4] Docker Inc., "What is Docker?," docker.com. [Online]. Disponible: https://www.docker.com/resources/what-container/. [Accedido: 10-Nov-2025].

[5] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston, MA, USA: Prentice Hall, 2017.

[6] R. C. Martin, "Design Principles and Design Patterns," objectmentor.com, 2000. [Online]. Disponible: https://web.archive.org/web/20150906155800/http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf. (Referencia original para principios SOLID).

[7] Atlassian, "Qué es Trello," trello.com. [Online]. Disponible: https://trello.com/tour [Accedido: 10-Nov-2025].