UNIVERSIDAD DE TARAPACÁ

FACULTAD DE INGENIERÍA

Departamento de Ingeniería en Computación e Informática

PROYECTO IV

Manual de Usuario

Autores: Patricio Chang Reyes
Francisco Pantoja González

Nombre Empresa: Empresa Portuaria Arica

Profesor: Diego Aracena Pizarro

Arica, 29 de Diciembre de 2025

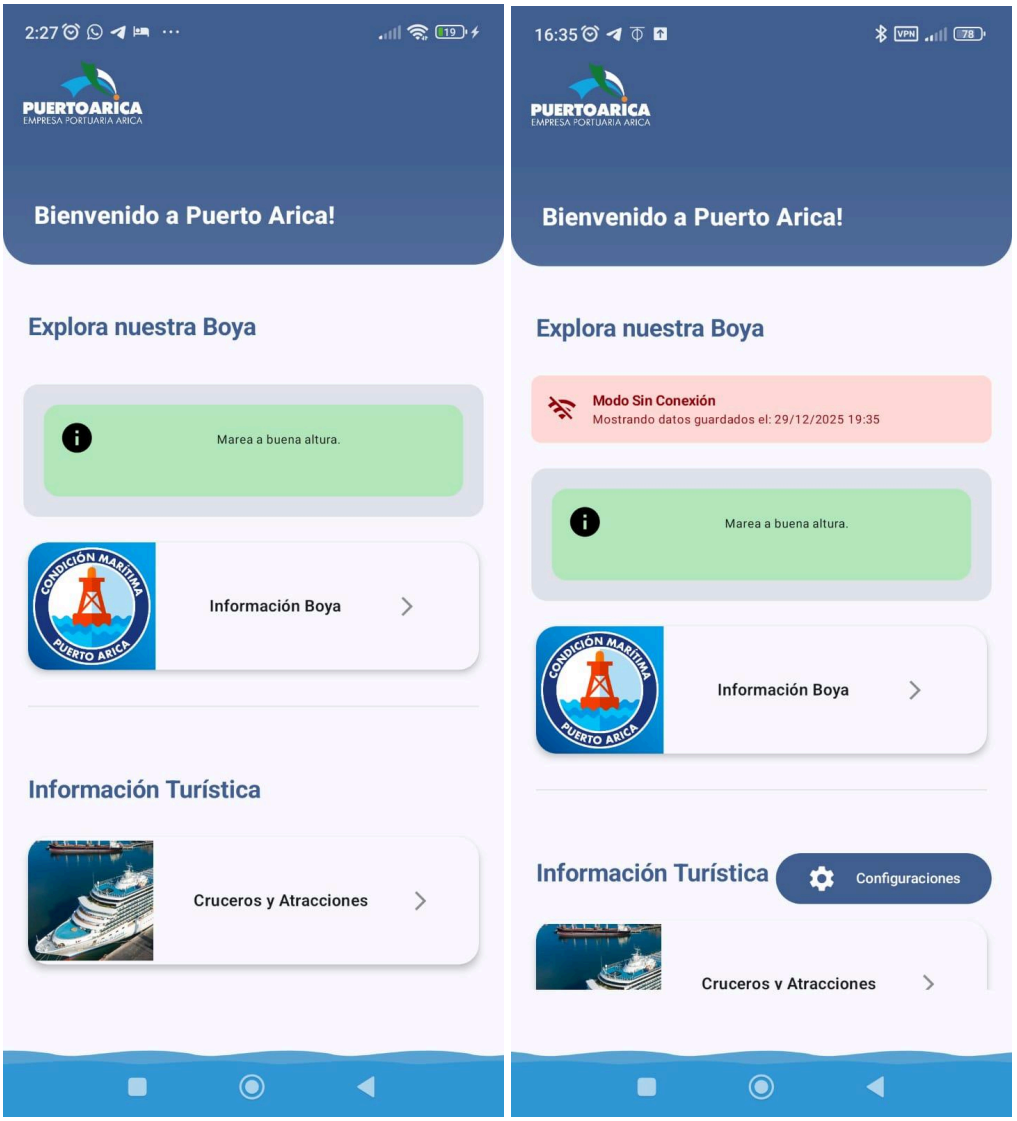
Índice

1. Pantallas de la Aplicación

La aplicación cuenta con cuatro pantallas principales: Principal, Boya, Turismo, Espacios. A continuación se detallan los aspectos centrales de cada una de ellas

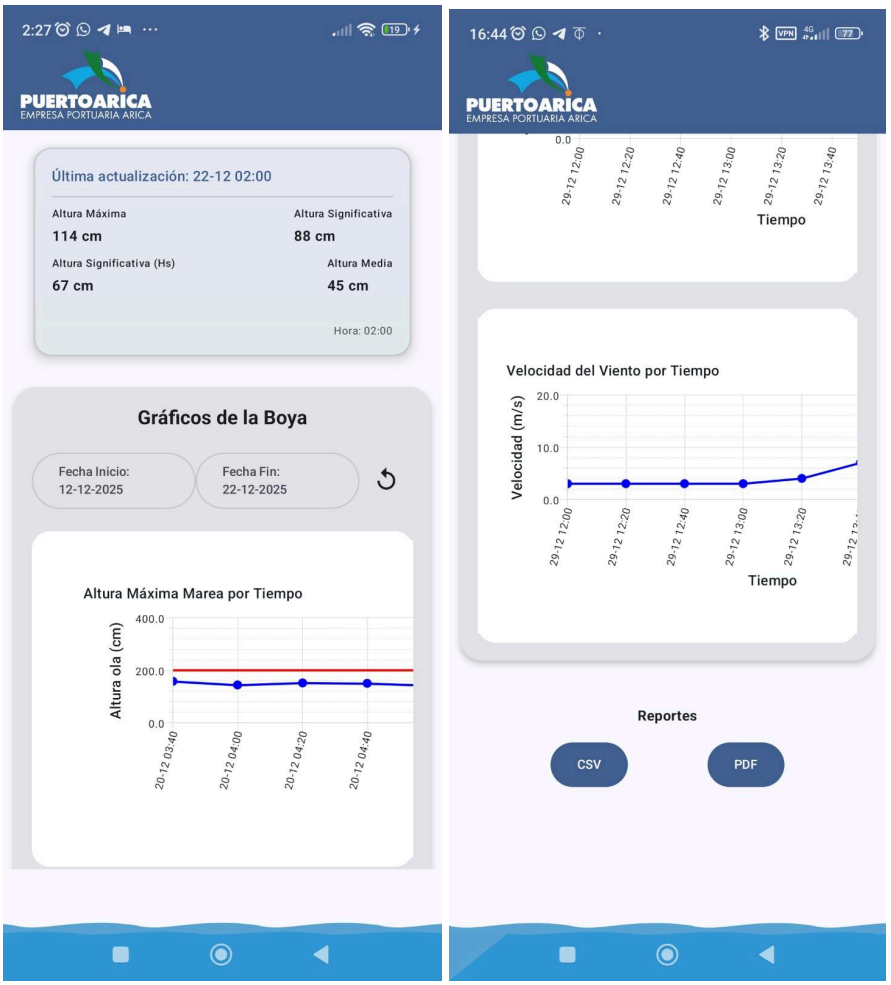
Principal

Contiene el menú principal donde se pueden acceder a los módulos de la aplicación. Cuenta con un botón en la zona inferior derecha para poder acceder a las configuraciones. En caso de no poder conectarse al backend, la aplicación funciona en modo sin conexión, con los últimos datos obtenidos del mismo.



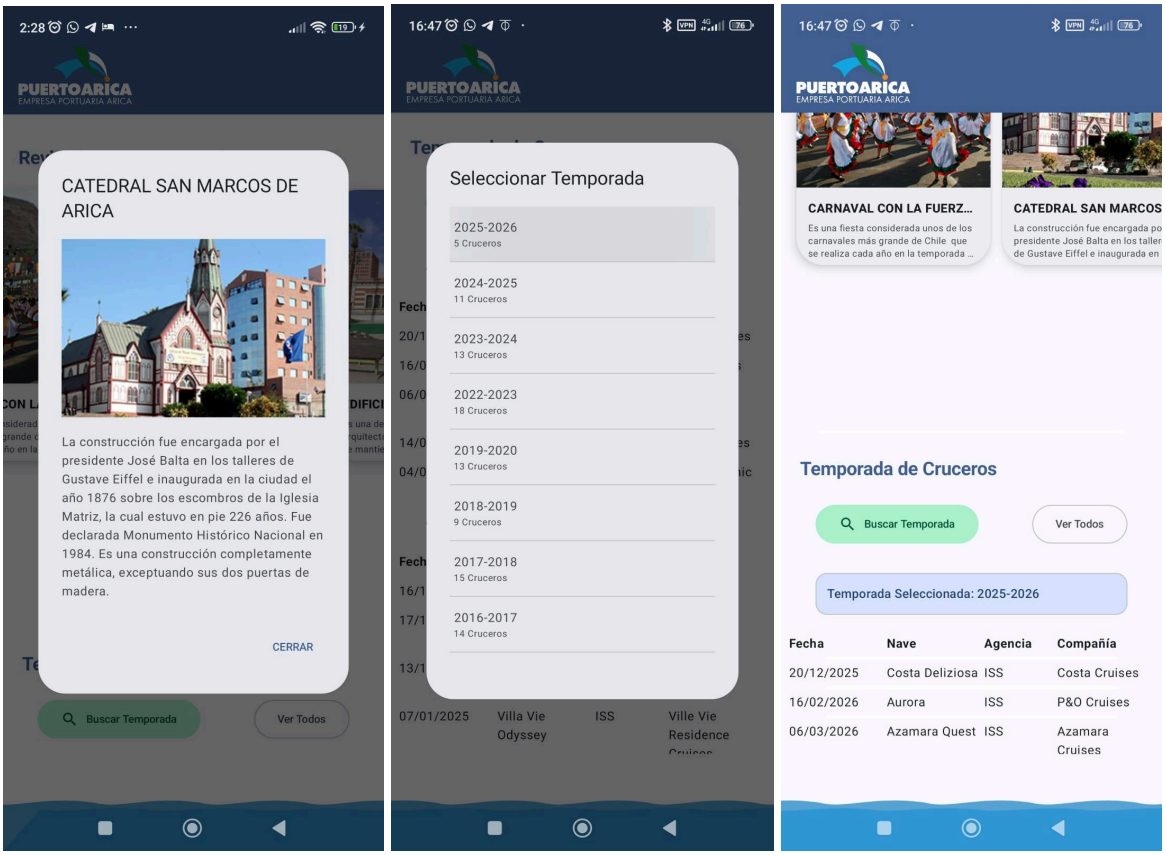
Boya

Contiene un mensaje del último estado de la marea. Dependiendo de la altura máxima obtenida, muestra un color distinto: verde, amarillo y rojo. Luego está el visualizador de gráficas por rango de tiempo, el cual se actualiza con el botón de recargar al costado derecho de los selectores de fecha. Finalmente se cuenta con dos botones para generar reportes: CSV y PDF, los cuales mostrarán un resumen de las gráficas acompañadas de dashboards extra (para PDF) de acuerdo al rango establecido por el usuario.



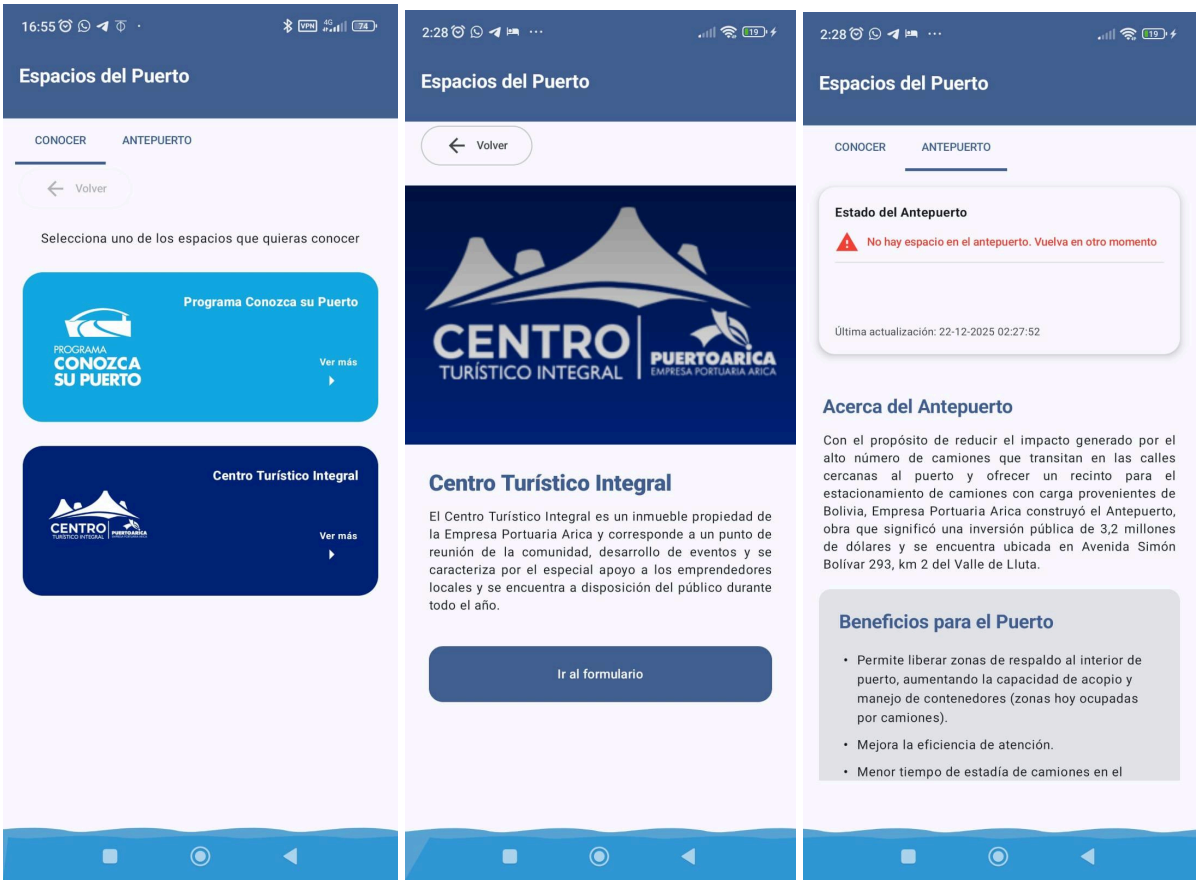
Turismo

Contiene dos secciones principales: Atractivos turísticos y el Itinerario de Cruceros. Ambas secciones se pueden recorrer un una lista horizontal y vertical respectivamente. Para filtrar los itinerarios de cruceros, utilice el botón de “Buscar Temporada”. Seleccionando en el diálogo emergente la temporada a visualizar.



Espacios

Contiene dos secciones principales, espacios del puerto y estado del antepuerto. Para navegar entre cada una de estas secciones puede hacer uso de los botones con los mismos nombres o desplazar horizontalmente la paginación. Para acceder a los formularios de solicitud de espacios del puerto, seleccione el botón correspondiente al espacio deseado. Luego llene los datos del formulario emergente.



2. Estructura del Backend

El backend actúa como un **núcleo integrador** diseñado bajo una arquitectura modular. Utiliza **NestJS** para organizar la lógica de negocio en servicios inyectables, lo que permite que cada módulo (Boya, Gestión y Turismo) opere de forma independiente pero comparta recursos globales como la base de datos, el sistema de caché (Redis) y el motor de notificaciones (Firebase).

2.1. Módulo: Boya (Monitoreo Oceanográfico)

Es el componente más robusto, encargado de la telemetría en tiempo real y la seguridad portuaria.

- **Adquisición de Datos:** Ejecuta procesos programados (**Cron Jobs**) que consumen APIs externas de sensores.
- **Procesamiento de Datos:** Recopila variables de altura de ola (máxima, significativa, media), periodos, cantidad de olas y ráfagas de viento.
- **Normalización Temporal:** Utiliza la librería **Luxon** para convertir los tiempos UTC de los sensores a la zona horaria local de Chile (America/Santiago).
- **Persistencia Híbrida:** * **Caché (Redis):** Almacena el estado actual y un historial rápido para los gráficos de la App móvil.
    - **Histórico (CSV):** Mantiene un registro físico de largo plazo para auditorías y reportes.
- **Inteligencia de Alertas:** Evalúa los datos en tiempo real. Si detecta olas peligrosas (ej. > 200 cm) en dos mediciones consecutivas, dispara notificaciones push vía **Firebase**.
- **Generador de Reportes:** Integra **PDFKit** y **QuickChart** para crear informes PDF automáticos con gráficos y tablas de riesgo con formato condicional.

2.2. Módulo: Gestión (Logística y Servicios Portuarios)

- **Gestión de Reservas (CTI y Visitas):** Maneja las solicitudes para el "Centro Turístico Integral" y el programa "Conozca su Puerto", procesando listas de visitantes y enviando confirmaciones automáticas vía **Nodemailer** tanto al administrador como al usuario.
- **Control de Antepuerto:** Implementa un **Cron Job** que simula y monitorea el estado de disponibilidad del antepuerto. Si el estado cambia (Disponible/Lleno), envía notificaciones push mediante **Firebase**, utilizando un **Cooldown de 1 hora** en Redis para evitar saturar a los transportistas.
- **Seguridad Anti-Spam:** Protege los formularios de reserva mediante el método validarSpam, que utiliza **Redis** para bloquear múltiples envíos desde un mismo dispositivo por un periodo de 30 minutos.

2.3. Módulo: Turismo (Información Turística Dinámica)

Este servicio proporciona información actualizada sobre los atractivos de la región y la llegada de cruceros mediante técnicas de recolección de datos.

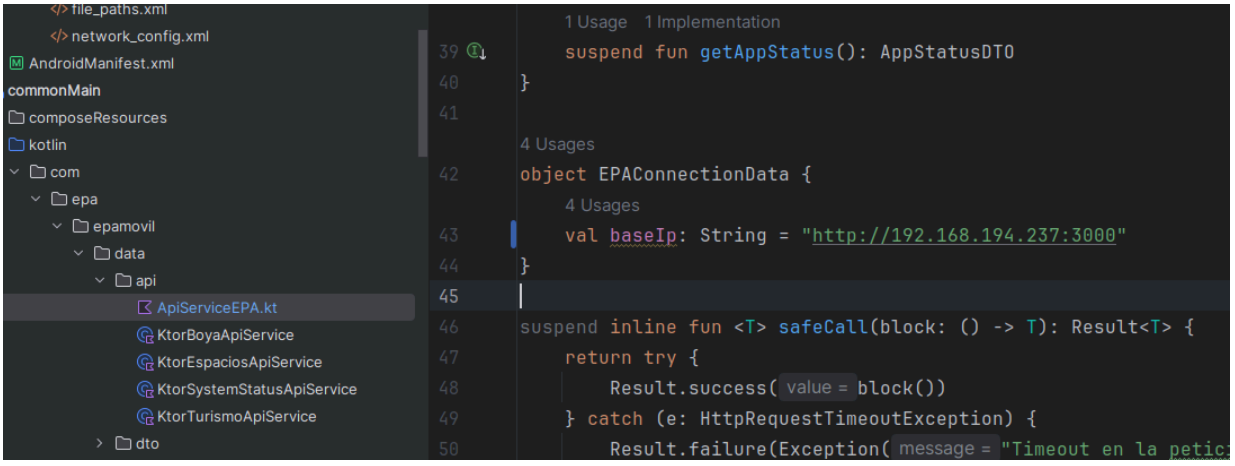
- **Web Scraping en Tiempo Real:** Utiliza la librería **Cheerio** para extraer información directamente desde el sitio oficial puertoarica.cl, evitando la necesidad de una base de datos manual.
- **Gestión de Atractivos:** Analiza el HTML de la sección de atractivos turísticos para obtener nombres, descripciones e imágenes de los destinos locales.
- **Itinerario de Cruceros:** Procesa dinámicamente las tablas de itinerarios de temporadas actuales (ej. 2025-2026), extrayendo datos de naves, agencias y fechas de arribo.

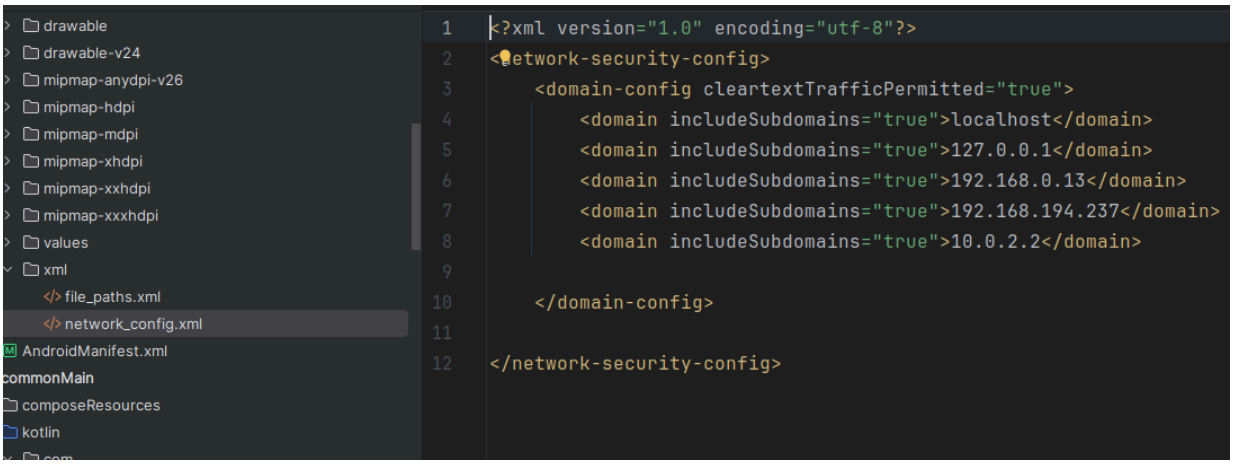
- **Configuración Segura:** Realiza peticiones HTTPS configuradas para ignorar errores de certificados no autorizados mediante un httpsAgent, garantizando la continuidad del servicio.

3. Configuración y Despliegue de la Aplicación

Establecer dirección IP del Backend

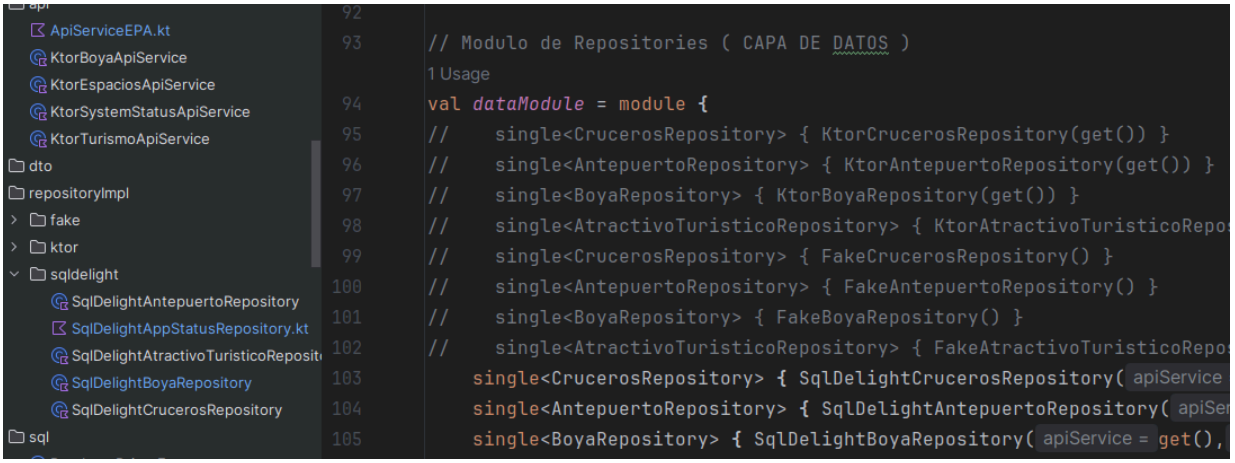
En el directorio principal, diríjase al archivo commonMain/com/epa/epamovil/data/api/ApiServiceEPA.kt. Luego ubique el objeto llamado EPAConnectionData y acceda a la variable baseIP. Este cambio afectará a todas las entradas que necesiten acceder al backend. Adicionalmente, para Android. En caso de que la dirección no esté protegida con el protocolo HTTPS, debe abrir una excepción de dominios inseguros permitidos en el archivo androidMain/res/xml/network_config.xml





Modificar Repositorios de prueba

El archivo commonMain/com/epa/epamovil/di/AppModule.kt permite modificar los Repositorios de prueba para realizar testing y debugging. Por ejemplo, todos los Repositorios con prefijo “Fake” sirven como pruebas para comprobación de funcionalidades con datos ficticios. Por defecto están habilitados todos los Repositorios con prefijo “SqlDelight”, los cuales cachean localmente la información obtenida desde Ktor.



4. Configuración y Despliegue del Backend

Para que el sistema funcione correctamente, se requiere la integración de servicios de terceros, variables de entorno específicas y una base de datos en memoria para la gestión de estados rápidos.

4.1. Requisitos Previos

Antes de iniciar la configuración, asegúrese de tener instalados los siguientes componentes:

- **Node.js:** Versión 18 o superior.
- **Redis Server:** Ejecutándose localmente o accesible vía red para la gestión de caché y bloqueos.
- **Cuenta de Firebase:** Con acceso a la consola para generar las credenciales del SDK de Administración.
- **Servidor SMTP:** Una cuenta de Gmail o similar con "Contraseñas de aplicación" activadas para el envío de correos.

4.2. Variables de Entorno

Cree un archivo .env en la raíz del proyecto con las siguientes claves obligatorias para el correcto funcionamiento de los servicios:

```
# Configuración de API Externa (OceanCom)
API_TOKEN=tu_token_aqui # Necesario para BoyaService

# Configuración de Correo (Nodemailer)
EMAIL_USER=tu_correo@gmail.com # Cuenta emisora
EMAIL_PASS=tu_clave_de_aplicacion # Contraseña de 16 dígitos de Google
EMAIL_TO=admin_puerto@ejemplo.com # Destinatario de las solicitudes CTI

# Umbrales de Seguridad
UMBRAL_ALERTA_CM=200 # Límite para disparar notificaciones de marejada

# Configuración de Redis
REDIS_HOST=localhost
REDIS_PORT=6379
```

4.3. Configuración de Servicios Externos

A. Firebase Admin SDK

Para habilitar las notificaciones push en tiempo real:

1. Descargue el archivo firebase-service-account.json desde la consola de Firebase.
2. Colóquelo en la raíz del proyecto.
3. El sistema utilizará este archivo para mandar alertas a los tópicos de firebase de la boya y el antepuerto.

B. Persistencia de Datos

- **Directorio de Datos:** Asegúrese de que la carpeta /data exista en la raíz del proyecto, ya que allí se almacenarán el archivo historico.csv.
- **Permisos:** El usuario que ejecute el backend debe tener permisos de escritura sobre este directorio para evitar errores en el **BoyaService**.

4.4. Despliegue con Docker y Docker Compose

4.4.1. El Dockerfile (Multi-stage Build)

Usaremos una construcción en dos etapas para que la imagen final sea liviana y solo contenga lo necesario para ejecutar la aplicación, evitando que el código fuente y las herramientas de compilación ocupen espacio innecesario.

```
# --- Fase 1: Construcción (Compilar el TypeScript) ---
FROM node:22-alpine AS builder

WORKDIR /app

# Copiar solo los package.json para instalar dependencias
COPY package.json package-lock.json ./
RUN npm install

# Copiar el resto del código fuente
COPY . .

# Compilar la aplicación
RUN npm run build

# --- Fase 2: Producción (Imagen final y limpia) ---
FROM node:22-alpine

WORKDIR /app

# Instalar solo las dependencias de producción (más ligero)
COPY package.json package-lock.json ./
RUN npm install --omit=dev

# Copiar la aplicación compilada desde la fase de "builder"
COPY --from=builder /app/dist ./dist

# Exponer el puerto que usa NestJS
EXPOSE 3000

# Comando para iniciar la aplicación
CMD ["node", "dist/main"]
```

4.4.2. Orquestación con Docker Compose

Para recibir los formularios de reserva, el estado del antepuerto y el histórico rápido de la boya, usaremos docker-compose.yml para levantar ambos servicios al mismo tiempo.

```
services:
  backend:
    build:
      context: .
    container_name: epa-backend
    ports:
      - "3000:3000"
    volumes:
      - ./data:/app/data
      -
./firebase-service-account.json:/app/firebase-service-account.json
    env_file:
      - .env
    depends_on:
      - redis
    environment:
      - REDIS_HOST=redis
      - REDIS_PORT=6379

  redis:
    image: redis:alpine
    container_name: epa-redis
    ports:
      - "6379:6379"
    restart: always
```

4.4.3. Consideraciones Críticas para el Despliegue

- **Persistencia (Volumes):** Es vital mapear la carpeta ./data del host hacia /app/data en el contenedor. Si no haces esto, cada vez que reinicies el contenedor de Docker perderás el archivo historico.csv generado por el **BoyaService**.
- **Networking:** En el archivo .env, debes cambiar REDIS_HOST=localhost por REDIS_HOST=redis. Dentro de la red de Docker, los contenedores se ven entre sí por el nombre del servicio definido en el Compose.

- **Firebase:** Asegúrate de que la ruta hacia el firebase-service-account.json en el contenedor coincida con la que espera tu FirebaseService para enviar las notificaciones de alerta y antepuerto.

4.4.4. Comandos para el Despliegue

Una vez configurados los archivos, solo necesitas ejecutar:

1. **Construir y levantar todo:**

```
docker-compose up -d --build
```

2. **Ver los logs en tiempo real (útil para el Cron Job):**

```
docker logs -f epa-backend
```