AAA Planificación ☝️

Requisitos

Funcionales

- Obtener información climática y de olas desde la API del CPTEC.
- Enviar notificaciones a los usuarios con las temperaturas previstas para los próximos 4 días.
- Para localidades costeras, incluir previsión de olas del día actual.
- Permitir programar el envío de notificaciones.
- Respetar la solicitud de opt-out de los usuarios.

No funcionales

- Alta escalabilidad y rendimiento.
- Resiliencia y bajo nivel de latencia.
- Capacidad de soporte para futuros canales de notificación (Push, SMS, correo electrónico).
- Uso de buenas prácticas en arquitectura de microservicios.

Problemas a Resolver

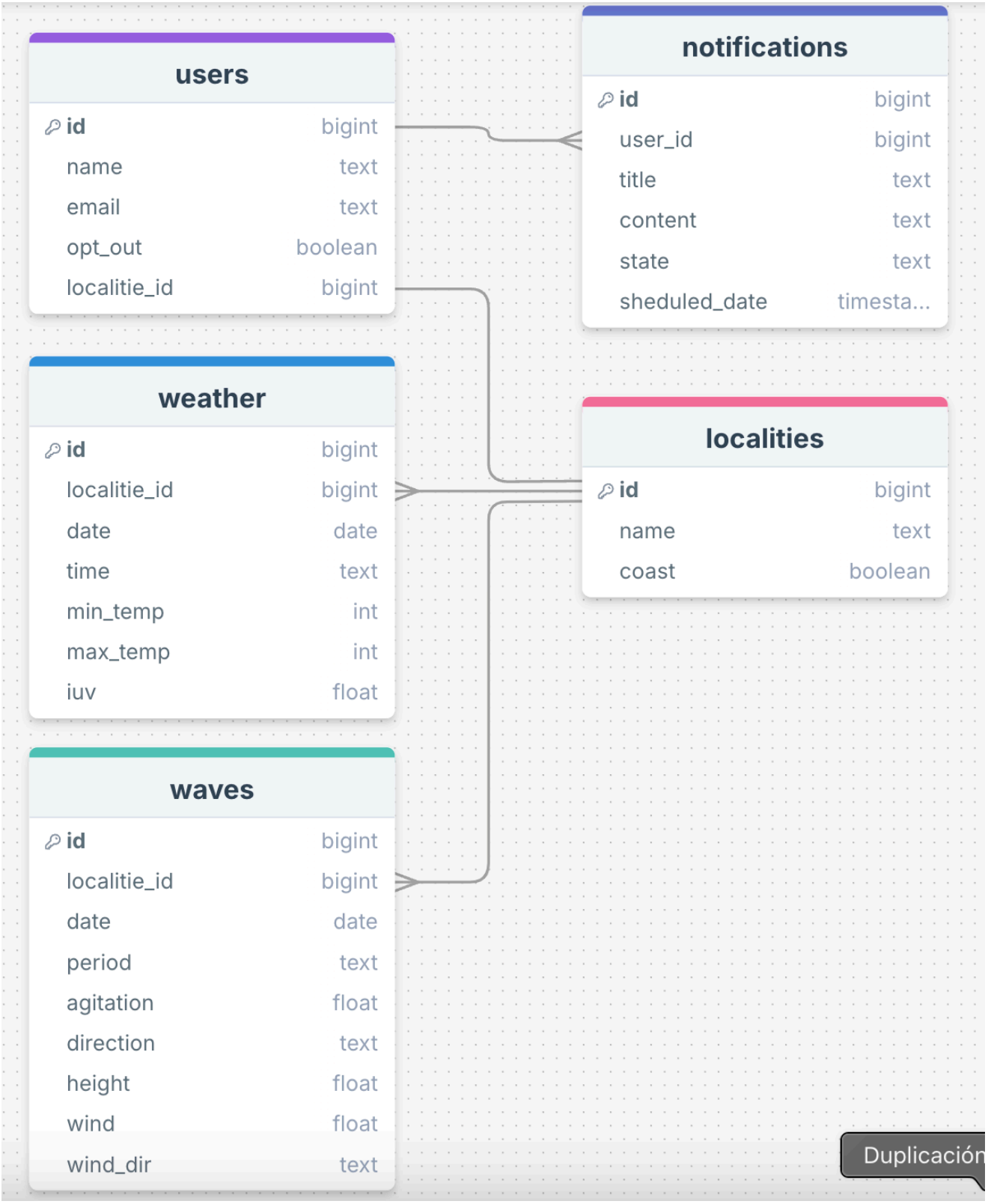
1. Integración con la API externa del CPTEC.
2. Gestión de usuarios y sus preferencias de notificación.
3. Programación y envío de notificaciones en horarios definidos.
4. Identificación de localidades costeras.
5. Escalabilidad para soportar un alto volumen de notificaciones.
6. Cacheo de datos climáticos para reducir dependencia de la API externa.

Identificar Entidades Claves

1. **Usuario**:
    - Atributos: id, nombre, email, preferencias de notificación (opt-in/opt-out), localidad.
2. **Localidad**:

- Atributos: id, nombre, costera (booleano).

3. **Notificación**:
   - Atributos: id, usuario_id, contenido, estado (pendiente, enviado, fallido), fecha_programada.
4. **Clima**:
   - Atributos: id, localidad_id, fecha, temperaturas (mínima, máxima).
5. **Olas** (para localidades costeras):
   - Atributos: id, localidad_id, fecha, altura, período, dirección.



Relación entre Entidades

- Un **Usuario** está asociado a una **Localidad**.
- Una **Localidad** puede tener múltiples registros de **Clima** y **Olas**.
- Un **Usuario** puede tener múltiples **Notificaciones** que contiene datos de **Clima** y **Olas**.

Relación con Servicios Externos

1. **API del CPTEC**: Proveerá los datos climáticos y de olas necesarios para generar notificaciones.
2. **Servicio de Notificaciones**: Se desarrollará internamente para manejar el envío de notificaciones (email en esta versión).

Modelo de Datos y Esquema

- Base de datos: PostgreSQL.

Tablas:

- **usuarios**: id (PK), nombre, email, opt_out (booleano), localidad_id (FK).
- **localidades**: id (PK), nombre, costera (booleano).
- **clima**: id (PK), localidad_id (FK), fecha, temp_min, temp_max.
- **olas**: id (PK), localidad_id (FK), fecha, altura, periodo, direccion.
- **notificaciones**: id (PK), usuario_id (FK), contenido, estado, fecha_programada.

Estructura Backend

1. **Directorios principales**:
   - `/cmd`: Puntos de entrada de la aplicación.
   - `/internal`: Lógica de negocio y entidades (implementación DDD).
   - `/pkg`: Componentes reutilizables.
   - `/configs`: Archivos de configuración.
   - `/scripts`: Scripts para tareas como migraciones o configuración inicial.
2. **Microservicios**:
   - **User Service**: Gestión de usuarios y preferencias.

- **Notification Service**: Gestión y envío de notificaciones.
- **Weather Service**: Integración con la API del CPTEC y cacheo de datos.

Estructura Frontend

- **Repositorio separado** para una aplicación en React y Next.js.
- Componentes:
  - **Gestión de usuarios**: Formulario para crear y editar preferencias.
  - **Vista de notificaciones**: Historial de notificaciones enviadas.

Elección de Tecnologías

- **Backend**: Golang.
- **Base de Datos**: PostgreSQL.
- **Mensajería**: RabbitMQ para orquestación de tareas y envío de notificaciones.
- **Cacheo**: Redis para almacenar datos climáticos temporalmente.
- **Contenedores**: Docker.
- **Gestión de endpoints**: Postman.
- **Pruebas**: Testify (framework de testing para Go).

Arquitectura del Sistema

- **Patrones de Diseño**:
  - Event-Driven Architecture para el envío de notificaciones.
  - Repository Pattern para el acceso a datos.
  - Circuit Breaker para la resiliencia al interactuar con la API del CPTEC.
- **Diagrama de Arquitectura**:
  - Microservicios comunicándose a través de RabbitMQ.
  - Redis como capa de cache.
  - PostgreSQL como base de datos central.
  - API Gateway para manejar peticiones externas.

Estructura de datos

```
/project-root
│
├── /user-service # Microservicio de usuarios
│   ├── /cmd
│   ├── /internal
│   │   ├── /application
│   │   ├── /domain
│   │   └── /infrastructure
│   ├── /pkg
│   ├── /configs
│   ├── /events # Eventos generados o consumidos
│   └── /test
│
├── /notification-service # Microservicio de notificaciones
│   ├── /cmd
│   ├── /internal
│   │   ├── /application
│   │   ├── /domain
│   │   ├── /events # Eventos para manejar las notificaciones
│   │   └── /infrastructure
│   ├── /pkg
│   ├── /configs
│   ├── /event-handlers # Handlers para eventos
│   └── /test
│
├── /weather-service # Microservicio de clima
│   ├── /cmd
│   ├── /internal
│   │   ├── /application
│   │   ├── /domain
│   │   └── /infrastructure
│   ├── /pkg
│   ├── /configs
│   └── /test
│
├── /migrations # Migraciones compartidas
│
├── /scripts # Scripts para facilitar tareas
│   ├── start_services.sh
│   ├── migrate_db.sh
│   └── test.sh
│
```

```
├── /docker-compose.yml
└── /README.md
```

Para mayor escalabilidad

Si se desea mayor escalabilidad al largo plazo, se puede dividir los microservicios en 3 repositorios independientes

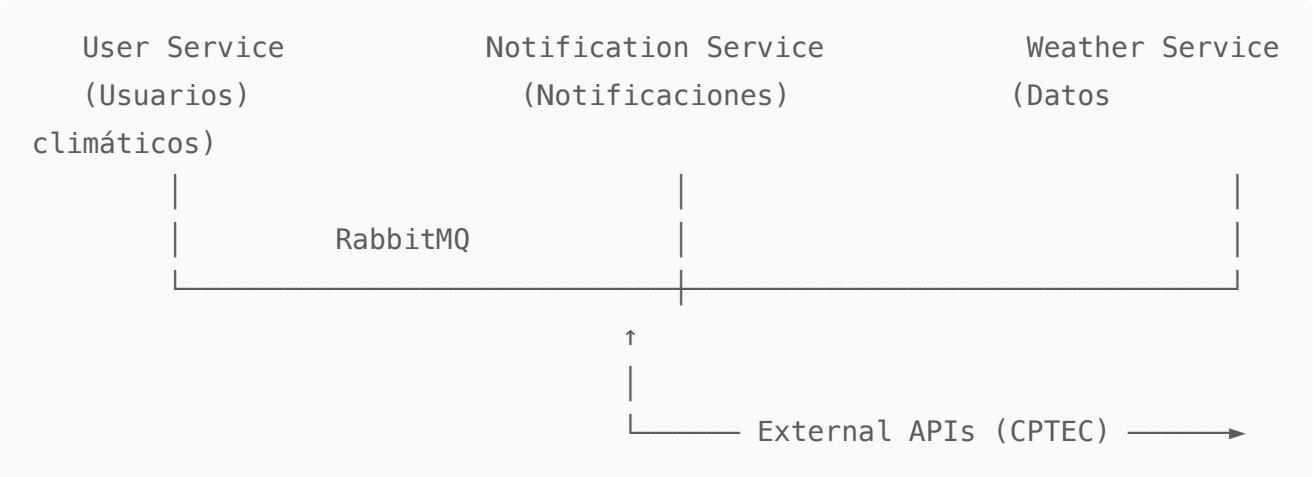
```
/user-service
├── /cmd # Punto de entrada principal
│   └── main.go # Inicia el servidor HTTP
├── /internal # Código interno no exportado
│   ├── /app # Lógica de negocio (Controllers y Servicios)
│   │   ├── user_controller.go # Manejo de solicitudes REST
│   │   └── user_service.go # Contiene lógica para usuarios
│   ├── /domain # Modelos y lógica del negocio
│   │   ├── user.go # Modelo de usuario
│   │   └── errors.go # Errores específicos del dominio
│   └── /infrastructure # Infraestructura (repositorios, DB, etc.)
│   ├── user_repository.go # Interacción con la base de datos
│   ├── db_connection.go # Configuración de PostgreSQL
│   └── http_server.go # Configuración del servidor HTTP
├── /pkg # Paquetes reutilizables
│   └── logger/logger.go # Paquete para logs
├── /configs # Archivos de configuración
│   └── config.yaml # Configuración para el microservicio
├── /events # Eventos generados o consumidos
│   └── user_updated_event.go # Evento de cambio en las preferencias del usuario
├── /test # Pruebas
│   ├── integration_test.go # Pruebas de integración
│   └── unit_test.go # Pruebas unitarias
└── go.mod # Dependencias de Go
```

```
notification-service/
├── cmd/
│   ├── notification-service/ # Punto de entrada del microservicio
│   └── main.go
├── internal/
│   ├── domain/
│   │   ├── notification/ # Entidades y lógica de negocio
│   │   │   ├── entity.go
│   │   │   ├── service.go
│   │   │   ├── event.go # Manejo de eventos
│   │   │   └── repository.go
```

```
│ ├── infrastructure/
│ │ ├── http/ # Handlers HTTP para exponer APIs
│ │ │ ├── notification_handler.go
│ │ │ └── middleware.go
│ │ ├── messaging/ # Abstracciones y manejo de cola de mensajes
│ │ │ ├── producer.go
│ │ │ ├── consumer.go
│ │ │ └── broker.go
│ │ ├── repository/ # Implementaciones de repositorios
│ │ │ ├── notification_repository.go
│ │ │ └── mocks.go
│ │ └── database/ # Conexión a la base de datos
│ │ └── db.go
│ ├── application/
│ │ └── notification_events.go # Casos de uso del dominio
└── go.mod

/weather-service
├── /cmd
│ └── main.go # Inicia el scheduler y servidores
├── /internal
│ ├── /app # Servicios de aplicación y lógica de negocio
│ │ ├── weather_service.go # Lógica para interactuar con CPTEC y el dominio
│ │ └── scheduler.go # Manejador de tareas programadas
│ ├── /domain # Modelos y lógica del negocio
│ │ ├── localidad.go # Modelo de localidad
│ │ ├── clima.go # Modelo de clima
│ │ ├── olas.go # Modelo de olas
│ │ └── errors.go # Errores específicos del dominio
│ └── /infrastructure # Infraestructura (conexiones externas)
│ ├── cptec_client.go # Cliente HTTP para consumir CPTEC
│ ├── db_repository.go # Repositorio para localidades, clima y olas
│ └── http_server.go # Servidor HTTP (para exposición de APIs)
├── /pkg
│ └── logger/logger.go # Logs
├── /configs
│ └── config.yaml # Configuración del microservicio
├── /test
│ ├── integration_test.go # Pruebas de integración
│ └── unit_test.go # Pruebas unitarias
└── go.mod
```

Microservicios



Endpoints

`user-service`

Encargado de la gestión de usuarios. Asegúrate de que los endpoints soporten operaciones necesarias para cumplir con el **opt-out** y la programación de notificaciones. Podrías agregar:

- **PATCH /users/{id}/opt-out**: Permitir que un usuario se desinscriba de las notificaciones.
- **POST /users/{id}/notifications/schedule**: Programar notificaciones para un usuario en específico.

Endpoints sugeridos finales:

- `GET /users`: Lista de usuarios.
- `GET /users/{id}`: Información de un usuario.
- `POST /users`: Crear un nuevo usuario.
- `PUT /users/{id}`: Actualizar un usuario existente.
- `DELETE /users/{id}`: Eliminar un usuario.
- `PATCH /users/{id}/opt-out`: Configurar opt-out para un usuario.

`notification-service`

Este servicio debe manejar el envío y la programación de notificaciones:

- `GET /notifications`: Listar todas las notificaciones (puedes incluir filtros).
- `GET /notifications/{id}`: Obtener detalles de una notificación específica.

- `POST /users/{id}/notifications/schedule`: Programar notificaciones para un usuario.
  Programa una nueva notificación para un usuario específico, que incluye un título, contenido y una fecha de programación. El endpoint permite a los usuarios recibir notificaciones en una fecha futura definida. Se debe validar que los campos de la notificación sean correctos y que la fecha de programación sea posterior a la fecha actual
  { "title": "Recordatorio", "content": "Tienes una cita a las 5 PM", "scheduled": "2025-01-30T15:00:00Z", "user_id": 1 }
- `PUT /notifications/{id}`: Actualizar detalles de una notificación.
- `DELETE /notifications/{id}`: Eliminar notificaciones no enviadas.

`weather-service`

Para manejar los datos del clima, asegúrate de que los endpoints aprovechen al máximo la API del CPTEC. Podrías agregar soporte para búsquedas dinámicas y filtros:

- `GET /localities`: Listar localidades (filtrar por nombre u otros parámetros).
- `GET /localities/{id}/forecast`: Obtener previsión climática de los próximos 4 días.
- `GET /localities/{id}/waves`: Obtener previsión de olas si aplica (localidades costeras).

Extras

Testing

- Pruebas unitarias para lógica de negocio.
- Pruebas de integración para microservicios.
- Mocking para endpoints externos.

Escalabilidad

- Diseño de base de datos con índices y particionamiento para alto rendimiento.
- Colas de mensajes para procesar notificaciones de forma asincrónica.

Seguridad

- Autenticación y autorización con tokens JWT.

- Cifrado de datos sensibles en tránsito y reposo.