Documentación de Backend API

Version 1.0.0

Jorge Cueva - Michael Lata

September 18, 2025

Contents

List of Code Listings

1 Introducción

El presente documento muestra la documentación de la Interfaz de comunicación API creada para **Livingnet**.

Con la intención de fortalecer la gestión dentro de la empresa Livingnet se propone y crea una base sólida de software con la que se puede gestionar los reportes de la empresa. Como base para este proyecto se implementa una interfaz desarrollada en **Angular**, y un sistema de comunicación escalable para la gestión de solicitudes.

Si bien por el tiempo y las necesidades de la empresa se presenta una base de datos de lo más simple, esta permite la **gestión de usuarios** y la **gestión de reportes**, además de algunos métodos adicionales de validación y gestión que serán útiles para el funcionamiento específico de la aplicación web.

Para esto se hace uso de una base de datos, y la API respectiva de la cual se hablará en este documento.

2 Descripción

El API presentado ha sido desarrollado en *Java* utilizando *Maven* y gestionado para iniciarse con *Spring Boot*. Su objetivo principal es servir como una herramienta de apoyo para los técnicos de la empresa **Livingnet**, permitiéndoles agilizar los procesos relacionados con la carga de datos y la gestión de información.

Además, esta API se complementa con una interfaz sencilla e intuitiva, lo que facilita su adopción y uso, brindando una experiencia práctica y eficiente tanto para quienes realizan la subida de datos como para quienes administran la información.

3 Organización

Para lograr facilidad de mantenimiento y adaptabilidad, el proyecto utiliza una arquitectura de software basada en el patrón **DAO** (Data Access Object), que organiza el código en varias secciones principales:

- **DAO:** Son las clases que tienen acceso directo a la base de datos, tanto mediante las unidades de persistencia como en el uso de sentencias SQL. Su función principal es gestionar la interacción con los datos.
- **Gestión:** Estas clases se encargan de la lógica de negocio, incluyendo validaciones, transiciones y operaciones adicionales que deben realizarse tras alguna acción.
- **Servicio:** Son las clases dedicadas exclusivamente a la API. Aquí se manejan las solicitudes HTTP y se define la interfaz de comunicación con el cliente.

Adicionalmente, se ha implementado una clase de **JWT** para manejar la autenticación y autorización. Esta clase se encarga de:

- Crear y leer tokens JWT.
- Generar un filtro que ejecute ciertas acciones en cada solicitud.
- Configurar un filtro de solicitudes para determinar qué peticiones se permiten y cuáles se bloquean.
- Gestionar la configuración de CORS, especificando quién puede realizar solicitudes y de qué manera.

Esta organización permite mantener el código modular, claro y fácil de mantener, facilitando la escalabilidad y la integración de nuevas funcionalidades.

4 Clases Modelo

A continuación se detallan las clases utilizadas en la API, con sus atributos, persistencia en la base de datos y características relevantes.

4.1 ReporteModel

Clase de utilidad histórica, que permite conocer el flujo de trabajo durante las jornadas de los equipos. Persistida en la tabla `reportes`.

- **id** (long) – `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`: Identificador único del reporte.
- **fecha** (Date) – `nullable = false`: Fecha del reporte.

- **horainicio** (Date) – `nullable = false`: Hora de inicio de la actividad.
- **horafin** (Date) – `nullable = false`: Hora de finalización de la actividad.
- **agencia, actividad, cuadrilla, jefe_cuadrilla, tipo_actividad, formato_actividad, complejidad_actividad, estado_actividad, clima, foto_url** (String) – `nullable = false`: Atributos descriptivos del reporte.
- **ayudante_tecnico, motivo_retraso, observaciones** (String) – `nullable = true`: Información adicional opcional.
- **kilometraje_inicio, kilometraje_fin** (double) – `nullable = false`: Datos de kilometraje.
- **router, onu, drop, roseta, tensores, conectores, camara** (int) – `nullable = false`: Contadores de equipos o elementos técnicos.

4.2 ReporteRequest

Clase de petición para filtrar o paginar reportes (no persistida). Permite realizar solicitudes específicas y optimizar la carga de datos, sacrificando un poco de procesamiento en cada solicitud.

- **datos** (int): Cantidad de datos solicitados.
- **pagina** (int): Página de la consulta.
- **fecha** (Date), **agencia, cuadrilla, jefe_cuadrilla, ayudante_tecnico, tipo_actividad, formato_actividad, complejidad_actividad, estado_actividad** (String/Date): Parámetros opcionales de filtrado.
- **retraso** (boolean): Filtrado según retraso.

4.3 UsuarioModel

Clase persistida en la tabla `usuarios`. Gestiona las sesiones de los empleados, ajustando la validez del token JWT según su rol y configuración del servidor.

- **id** (Long) – `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`: Identificador único.

- **mail** (String) – `@Column(unique = true, nullable = false)`: Correo electrónico único del usuario.
- **password** (String) – `nullable = false`: Contraseña del usuario.
- **rol** (String) – `nullable = false`: Rol del usuario.

4.4 UsuarioRequest

Clase de petición para manejo de usuarios (no persistida). Realiza validaciones adicionales, como la confirmación de contraseña, para reforzar la seguridad en caso de fallo de validación en el frontend.

- **id** (Long): Identificador del usuario.
- **mail, password, rol, confirmPassword** (String): Datos del usuario para creación o modificación.

4.5 UsuarioSend

Clase de respuesta para usuarios (no persistida). Se utiliza para enviar información de usuario en solicitudes `GET`, evitando exponer contraseñas u otros datos sensibles.

- **id** (Long): Identificador del usuario.
- **mail** (String): Correo electrónico del usuario.
- **rol** (String): Rol del usuario.

5 Clases DAO

A continuación se describen las clases DAO utilizadas en la API, responsables del acceso y manipulación de los datos persistidos en la base de datos.

5.1 ReportesDAO

Clase de acceso a datos para la entidad `ReporteModel`. Gestiona la persistencia de los reportes históricos y proporciona métodos de filtrado, conteo y eliminación.

- **Atributos:**
  - `EntityManager em` – `@PersistenceContext`: Gestiona las operaciones con la base de datos.
- **Métodos principales:**
  - `findAll(): List<ReporteModel>` – Devuelve todos los reportes.
  - `addReporte(ReporteModel reporte): ReporteModel` – Persiste un nuevo reporte (`@Transactional`).
  - `updateReporte(ReporteModel reporte): ReporteModel` – Actualiza un reporte existente (`@Transactional`).
  - `deleteReporte(Long id): boolean` – Elimina un reporte según su ID (`@Transactional`).
  - `getCantidadReportes(ReporteRequest request): Long` – Devuelve el conteo de reportes según filtros proporcionados en `ReporteRequest`. el atributo de retraso ha sido colocado como String para procesarlas como todas, en caso de null, si existe retraso o no con valores 'yes' o 'no'
  - `getReportesFiltrado(ReporteRequest request): List<ReporteModel>` – Devuelve reportes filtrados y paginados según los parámetros de `ReporteRequest`.
  - `deleteImagen(Long id): Boolean` – Elimina el archivo de imagen asociado a un reporte, si existe.
    /a ruta de UPLOAD_DIR es un valor estático en imagProcessing
  - Métodos de obtención de listas únicas para desplegables: `getAgencias()`, `getTiposActividad()`, `getComplejidades()`, `getEstados()`, `getJefesCuadrilla()`, `getCuadrillas()`, `getAyudantesTecnico()`, `getFormatosActividad()` – Devuelven listas de strings distintos según los reportes existentes.

5.2 UsuarioDAO

Clase de acceso a datos para la entidad `UsuarioModel`. Gestiona la persistencia y consulta de usuarios dentro de la aplicación.

- **Atributos:**
  - `EntityManager em` – `@PersistenceContext`: Gestiona las operaciones con la base de datos.
- **Métodos principales:**
  - `buscarPorId(Long id): Optional<UsuarioModel>` – Busca un usuario por su ID.
  - `buscarPorEmailYPassword(String mail, String password): UsuarioModel` – Busca un usuario según correo y contraseña.
  - `getUsuarios(): List<UsuarioModel>` – Devuelve todos los usuarios.
  - `save(UsuarioModel usuario): UsuarioModel` – Persiste un nuevo usuario (`@Transactional`).
  - `updateUsuario(UsuarioModel usuario): UsuarioModel` – Actualiza un usuario existente (`@Transactional`).
  - `deleteUsuario(Long id): boolean` – Elimina un usuario por ID (`@Transactional`).

6 Clases Gestión

A continuación se describen las clases de gestión utilizadas en la API, encargadas de la lógica de negocio y de coordinar las operaciones entre los DAO y los servicios de la aplicación.

6.1 ReportesGestion

Clase de gestión para la entidad `ReporteModel`. Se encarga de coordinar la lógica de negocio relacionada con los reportes y de interactuar con `ReportesDAO`.

- **Atributos:**
  - `ReportesDAO reportesDAO` – DAO inyectado para acceso a datos de reportes.
- **Métodos principales:**

- `getReportes(): List<ReporteModel>` – Devuelve todos los reportes.
- `addReporte(ReporteModel reporte): ReporteModel` – Añade un nuevo reporte.
- `updateReporte(ReporteModel reporte): ReporteModel` – Actualiza un reporte existente.
- `deleteReporte(Long id): Map<String,Boolean>` – Elimina un reporte y su imagen asociada, devolviendo un mapa indicando qué se eliminó.
- `getReportesFiltros(ReporteRequest rq): List<ReporteModel>` – Devuelve reportes filtrados según parámetros.
- `getCantidadReportes(ReporteRequest body): Long` – Devuelve el conteo de reportes según filtros.
- `getDesplegables(): Map<String, List<String» – Devuelve listas únicas de atributos para interfaces de filtrado y desplegables.

6.2 UsuarioGestion

Clase de gestión para la entidad `UsuarioModel`. Se encarga de coordinar la lógica de negocio relacionada con los usuarios y de interactuar con `UsuarioDAO`.

- **Atributos:**
  - `UsuarioDAO usuarioDAO` – DAO inyectado para acceso a datos de usuarios.
- **Métodos principales:**
  - `buscarPorId(Long id): Optional<UsuarioModel>` – Busca un usuario por su ID.
  - `buscarPorEmailYPassword(String email, String password): UsuarioModel` – Busca un usuario según correo y contraseña.
  - `getUsuarios(): List<UsuarioModel>` – Devuelve todos los usuarios.

- `addUsuario(UsuarioModel usuario): UsuarioModel` – Añade un nuevo usuario.
- `updateUsuario(UsuarioModel usuario): UsuarioModel` – Actualiza un usuario existente.
- `deleteUsuario(Long id): boolean` – Elimina un usuario por ID.

7 Clases Servicios

Las clases de servicio son responsables de manejar las solicitudes HTTP, interactuar con las clases de gestión y devolver las respuestas correspondientes. Todas las rutas de estas clases a excepción de /login están protegidas mediante JWT, que debe incluirse en el header de la solicitud.

7.1 LoginService

Clase de servicio para la autenticación de usuarios.

- **Ruta base:** `/`
- **Métodos principales:**
  - `login(UsuarioModel usuario): ResponseEntity<?>` – Valida las credenciales del usuario y devuelve un token JWT en caso de éxito.

  No se especifica la duración del token JWT ni los roles en caso de manejar permisos se hara de forma interna en la misma aplicación.

7.2 ReportesService

Clase de servicio para manejar solicitudes relacionadas con `ReporteModel`.
Nota: Todas las rutas requieren un JWT válido en el header para acceder.

- **Ruta base:** `/reports`
- **Métodos principales:**
  - `getReportes(): List<ReporteModel>` – Devuelve todos los reportes.

- `getReportesFiltros(ReporteRequest body): List<ReporteModel>` – Devuelve reportes filtrados incluyendo página y cantidad de reportes por página.
- `getCantidadReportes(ReporteRequest body): Long` – Devuelve la cantidad de reportes filtrados totales.
- `addReporte(ReporteModel reporte): ReporteModel` – Agrega un nuevo reporte si cumple con todos los datos obligatorios.
- `updateReporte(ReporteModel reporte): ReporteModel` – Actualiza un reporte existente según su ID y validaciones.
- `deleteReporte(Long id): ResponseEntity<?>` – Elimina un reporte y su imagen asociada.
- `getDesplegables(): ResponseEntity<?>` – Devuelve mapas con los valores únicos de atributos para filtrado.

Se realiza validación manual de todos los campos de `ReporteModel` antes de agregar o actualizar un reporte. Los detalles de JWT se encuentran en las clases de JWT

7.3 UsuariosService

Clase de servicio para manejar solicitudes relacionadas con `UsuarioModel`.
Nota: Todas las rutas requieren un JWT válido en el header.

- **Ruta base:** `/users`
- **Métodos principales:**
  - `getUsuarios(): List<UsuarioSend>` – Devuelve todos los usuarios sin exponer la contraseña.
  - `addUsuario(UsuarioRequest usuario): UsuarioModel` – Agrega un usuario nuevo si los campos obligatorios son válidos y las contraseñas coinciden.
  - `updateUsuario(UsuarioRequest usuario): UsuarioModel` – Actualiza un usuario existente si la contraseña confirmada coincide.

- `deleteUsuario(Long id): boolean` – Elimina un usuario por su ID.

Se utilizan las clases de UsuarioRequest y Usuario Send para enviar los valores justos y no exponer o recibir datos ajenos.

7.4 Validation

Clase de servicio de validación de JWT. **Nota:** Permite verificar si un token JWT es válido.

- **Ruta base:** `/validate`
- **Método principal:**
  - `validate(): String` – Retorna un mensaje de validación exitosa.

método simple para confirmar la validez del token JWT útil para que el usuario no esté en sesión más tiempo del necesario.