Bitácora - Plataforma de Gestión de Actividades

[![Backend CI](https://github.com/Evincere/bitacora-mpd/actions/workflows/backend-ci.yml/badge.svg)](https://github.com/Evincere/bitacora-mpd/actions/workflows/backend-ci.yml)
[![Frontend CI](https://github.com/Evincere/bitacora-mpd/actions/workflows/frontend-ci.yml/badge.svg)](https://github.com/Evincere/bitacora-mpd/actions/workflows/frontend-ci.yml)
[![Quality Gate Status](https://sonarcloud.io/api/project_badges/measure?project=Evincere_bitacora-mpd&metric=alert_status)](https://sonarcloud.io/dashboard?id=Evincere_bitacora-mpd)

Descripción

Bitácora es una plataforma moderna para la gestión de actividades y tareas del Ministerio Público de la Defensa que reemplaza el sistema actual basado en CSV. Permite a los equipos registrar, dar seguimiento y analizar las actividades realizadas de manera eficiente y colaborativa.

Características principales

- **Registro estructurado de actividades**: Formularios inteligentes adaptados a diferentes tipos de actividades
  - Autocompletado inteligente basado en datos históricos
  - Sugerencias contextuales que se adaptan según las relaciones entre campos
  - Validación en tiempo real con mensajes específicos
  - Sistema de plantillas para agilizar la creación de actividades recurrentes
- **Dashboard personalizable**: Visualización de métricas y actividades relevantes
  - Dashboard específico para SOLICITANTE con resumen de solicitudes y tiempos de respuesta
  - Dashboard específico para ASIGNADOR con bandeja de entrada, distribución de carga y métricas
  - Dashboard específico para EJECUTOR con tareas asignadas y progreso
- **Flujo de trabajo optimizado**: Interfaces específicas para cada rol
  - SOLICITANTE: Creación de solicitudes, seguimiento de estado y visualización de tiempos de respuesta
  - ASIGNADOR: Bandeja de entrada de solicitudes, asignación a ejecutores y monitoreo de carga de trabajo
  - EJECUTOR: Gestión de tareas asignadas, actualización de progreso y registro de resultados
- **Estadísticas y resúmenes**: Visualización de datos por tipo y estado de actividades
- **Colaboración entre equipos**: Espacios de trabajo compartidos con permisos granulares
  - Indicadores de presencia en tiempo real
  - Notificaciones de edición simultánea
  - Sistema de comentarios y menciones
- **Reportes y análisis**: Visualizaciones avanzadas y métricas personalizables
- **Interfaz moderna**: Diseño intuitivo con modo oscuro y experiencia de usuario optimizada
  - Skeleton loaders para mejorar la percepción de velocidad
  - Transiciones animadas entre páginas
  - Diseño glassmorphism para una apariencia moderna
- **Alta disponibilidad**: Arquitectura robusta con monitoreo en tiempo real
- **Seguridad integrada**: Autenticación JWT, roles y permisos granulares
- **Experiencia offline**: Caché local para funcionamiento sin conexión

Arquitectura

La aplicación sigue una arquitectura moderna y modular, diseñada para facilitar el mantenimiento, la escalabilidad y la colaboración entre desarrolladores. Para más detalles, consulte la [documentación de arquitectura](docs/ARCHITECTURE.md).

Backend
- Java 21
- Arquitectura Hexagonal
- Spring Boot 3.x
- PostgreSQL / H2 (desarrollo)
- API RESTful
- Caché con Caffeine
- Migraciones con Flyway
- Documentación con OpenAPI
- Generación automática de tipos TypeScript desde OpenAPI
- Spring Data JPA Specifications para filtros dinámicos
- Projections para optimizar consultas
- Monitoreo con Prometheus y Grafana
- Tracing distribuido con Zipkin

Frontend

- React con TypeScript
- Arquitectura modular basada en features
- Styled Components
- React Query para gestión de estado del servidor y caché
- Redux para estado global
- Hooks personalizados para operaciones CRUD
- Sistema de notificaciones toast accesible y personalizable
- Validación de parámetros para prevenir errores
- Sistema de caché en localStorage para experiencia offline
- Debounce para optimizar búsquedas
- Lazy loading y code splitting
- Tema oscuro y diseño responsive
- Virtualización para listas grandes
- Skeleton loaders para mejorar la percepción de velocidad
- Transiciones animadas entre páginas
- Componentes estilizados con Styled Components

Estructura del Frontend

```
frontend/
├── public/              # Archivos estáticos
├── src/                 # Código fuente
│   ├── core/            # Núcleo de la aplicación
│   │   ├── api/         # Configuración y utilidades para API
│   │   ├── hooks/       # Hooks personalizados globales
│   │   ├── store/       # Estado global (Redux)
│   │   ├── types/       # Tipos e interfaces globales
│   │   └── utils/       # Utilidades y funciones auxiliares
│   ├── features/        # Características/módulos de la aplicación
│   │   ├── activities/  # Módulo de actividades
│   │   │   ├── components/  # Componentes específicos
│   │   │   ├── hooks/       # Hooks específicos
│   │   │   ├── pages/       # Páginas/vistas
│   │   │   ├── schemas/     # Esquemas de validación
│   │   │   └── services/    # Servicios específicos
│   │   ├── auth/        # Módulo de autenticación
│   │   ├── calendar/    # Módulo de calendario
│   │   └── notifications/ # Módulo de notificaciones
│   ├── shared/          # Componentes y utilidades compartidas
│   │   ├── components/  # Componentes reutilizables
│   │   │   ├── common/  # Componentes básicos (Loader, ErrorBoundary, etc.)
│   │   │   ├── layout/  # Componentes de layout (Layout, Header, Sidebar)
│   │   │   └── ui/      # Componentes de UI (StatusBadge, PageTransition, etc.)
│   │   └── styles/      # Estilos globales y temas (theme.ts, statusColors.ts, etc.)
│   ├── components/       # Alias para componentes compartidos (exporta desde shared)
│   ├── hooks/           # Alias para hooks compartidos (exporta desde core/hooks)
│   ├── styles/           # Alias para estilos compartidos (exporta desde shared/styles)
│   ├── App.tsx          # Componente principal
│   └── main.tsx         # Punto de entrada
├── index.html           # Plantilla HTML
└── vite.config.js       # Configuración de Vite
```

> **Nota**: Las carpetas `components`, `hooks` y `styles` en la raíz de `src` son alias que exportan desde las ubicaciones reales en `shared` y `core`. Esto facilita la migración y mantiene la compatibilidad con el código existente.

DevOps
- CI/CD con GitHub Actions
- Contenedores Docker
- Orquestación con Docker Compose
- Análisis de código con SonarCloud

```
backend/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── bitacora/
│   │   │           ├── application/         # Casos de uso
│   │   │           │   ├── auth/             # Casos de uso de autenticación
│   │   │           │   └── session/          # Casos de uso de sesiones
│   │   │           ├── domain/              # Entidades y reglas de negocio
│   │   │           │   ├── model/            # Modelos de dominio
│   │   │           │   │   ├── notification/  # Modelos de notificaciones
│   │   │           │   │   └── session/       # Modelos de sesiones
│   │   │           │   └── port/             # Puertos (interfaces)
```

```
│   │   │   │   │   ├── notification/  # Puertos de notificaciones
│   │   │   │   │   └── session/       # Puertos de sesiones
│   │   │   │   └── infrastructure/      # Adaptadores
│   │   │   │       ├── config/          # Configuraciones
│   │   │   │       ├── messaging/       # Adaptadores de mensajería
│   │   │   │       │   ├── adapter/      # Adaptadores de notificaciones
│   │   │   │       │   └── controller/   # Controladores WebSocket
│   │   │   │       ├── persistence/     # Adaptadores de persistencia
│   │   │   │       │   ├── adapter/      # Adaptadores de repositorios
│   │   │   │       │   ├── entity/       # Entidades JPA
│   │   │   │       │   ├── mapper/       # Mappers entre entidades y modelos
│   │   │   │       │   └── repository/   # Repositorios JPA
│   │   │   │       ├── rest/            # Adaptadores REST
│   │   │   │       │   ├── controller/   # Controladores REST
│   │   │   │       │   └── dto/          # DTOs para la API REST
│   │   │   │       └── security/        # Configuración de seguridad
│   │   │   └── resources/
│   │   │       └── db/
│   │   │           └── migration/
│   │   │               └── V2__create_user_sessions_table.sql
│   │   └── test/
│   └── pom.xml
```

Requisitos

Para desarrollo

- Node.js 20+
- Java 21 JDK
- PostgreSQL 14+
- Maven 3.8+
- xmllint (para verificación de archivos XML)

Para despliegue con Docker

- Docker 24+
- Docker Compose v2+

Despliegue con Docker

1. Configurar variables de entorno:
   ```bash
   cp .env.example .env
   # Editar .env con valores seguros
   ```

2. Compilar el backend:
   ```bash
   cd backend
   mvn clean package -DskipTests
   cd ..
   ```

3. Iniciar los contenedores:
   ```bash
   docker-compose up -d
   ```

4. Verificar el estado de los contenedores:
   ```bash
   docker-compose ps
   ```

5. Acceder a la aplicación:
   - Frontend: http://localhost:8090
   - Grafana: http://localhost:3000
   - Prometheus: http://localhost:9090
   - Zipkin: http://localhost:9411

Para más detalles, consultar la [documentación de despliegue con Docker](docs/docker-deployment.md).

Scripts de mantenimiento

El proyecto incluye varios scripts para facilitar el mantenimiento:

- `scripts/verify-configs.sh`: Verifica la validez de los archivos de configuración

- `scripts/setup-hooks.sh`: Configura hooks de git para verificar configuraciones antes de commit
- `scripts/check-dependencies.sh`: Verifica y actualiza dependencias
- `backend/generate-openapi.bat`: Genera la especificación OpenAPI manualmente
- `backend/generate-typescript.bat`: Genera tipos TypeScript desde la especificación OpenAPI

Generación de tipos TypeScript

Para generar los tipos TypeScript desde la especificación OpenAPI manualmente:

```bash
cd backend
.\generate-openapi.bat  # Genera el archivo openapi.json
.\generate-typescript.bat  # Genera los tipos TypeScript desde openapi.json
```

Esto creará los tipos TypeScript en `frontend/src/api/generated/`. El script `start-dev.bat` ejecuta estos pasos automáticamente al iniciar la aplicación.

Instalación y ejecución

Usando Docker (recomendado)

Para iniciar toda la aplicación con Docker Compose:

```bash
# Crear archivo de variables de entorno
cp .env.example .env

# Iniciar los servicios
docker-compose up -d
```

Esto iniciará todos los servicios: backend, frontend, base de datos, y herramientas de monitoreo.

Inicio rápido (Windows)

Para iniciar la aplicación en modo desarrollo:

```bash
.\start-app.bat
```

Esto iniciará tanto el backend como el frontend y abrirá la aplicación en el navegador. El script también ofrece la opción de generar los tipos TypeScript desde la especificación OpenAPI antes de iniciar la aplicación.

Para detener el entorno de desarrollo:

```bash
# Presiona Ctrl+C en la ventana de comandos donde se ejecuta la aplicación
```

Instalación manual

Backend

```bash
cd backend
mvn clean install

# Ejecutar en modo desarrollo (con H2 en memoria)
.\run-dev.bat

# O ejecutar normalmente (requiere PostgreSQL)
# mvn spring-boot:run
```

Solución de problemas del backend

Si encuentras problemas al ejecutar el backend, consulta el archivo `backend/TROUBLESHOOTING.md` para ver soluciones a problemas comunes.

Frontend

```bash
cd frontend
npm install

# Ejecutar en modo desarrollo
.\run-dev.bat

# O ejecutar directamente
# npm start
```

Acceso a la aplicación

Con Docker:
- **Aplicación**: http://localhost
- **Prometheus**: http://localhost:9090
- **Grafana**: http://localhost:3000 (admin/admin)
- **Zipkin**: http://localhost:9411

En desarrollo:
- **Frontend**: http://localhost:3000
- **Backend API**: http://localhost:8080/api
- **Swagger UI**: http://localhost:8080/api/swagger-ui/index.html
- **Consola H2**: http://localhost:8080/api/h2-console

Endpoints Principales

Actividades
- **GET /api/activities**: Listar actividades con paginación y filtros
- **GET /api/activities/{id}**: Obtener una actividad por ID
- **POST /api/activities**: Crear una nueva actividad
- **PUT /api/activities/{id}**: Actualizar una actividad existente
- **DELETE /api/activities/{id}**: Eliminar una actividad

Flujo de Trabajo de Actividades
- **POST /api/activities/request**: Crear solicitud (SOLICITANTE)
- **POST /api/activities/{id}/assign**: Asignar tarea (ASIGNADOR)
- **POST /api/activities/{id}/start**: Iniciar tarea (EJECUTOR)
- **POST /api/activities/{id}/complete**: Completar tarea (EJECUTOR)
- **POST /api/activities/{id}/approve**: Aprobar tarea (ASIGNADOR)
- **POST /api/activities/{id}/reject**: Rechazar tarea (ASIGNADOR)
- **POST /api/activities/{id}/cancel**: Cancelar tarea
- **POST /api/activities/{id}/comment**: Agregar comentario a una actividad

Estadísticas y Resúmenes
- **GET /api/activities/stats/by-type**: Obtener estadísticas por tipo de actividad
- **GET /api/activities/stats/by-status**: Obtener estadísticas por estado de actividad
- **GET /api/activities/summaries**: Obtener resúmenes de actividades

Módulo de Actividades

El módulo de actividades permite gestionar las actividades del sistema, incluyendo la creación, edición, visualización y eliminación de actividades.

Componentes principales

- **Activities.tsx**: Componente principal que muestra la lista de actividades
- **ActivityForm.tsx**: Formulario para crear y editar actividades
  - Implementa autocompletado inteligente basado en datos históricos
  - Ofrece sugerencias contextuales que se adaptan según las relaciones entre campos
  - Proporciona validación en tiempo real con mensajes específicos
  - Incluye sistema de plantillas para agilizar la creación de actividades recurrentes
  - Permite guardar borradores automáticamente en localStorage
- **ActivityList.tsx** y **ActivityGrid.tsx**: Componentes para mostrar las actividades en formato lista o cuadrícula
- **ExpandableActivityDetail.jsx**: Componente para mostrar detalles de actividades de forma expandible en la vista de lista
- **ActivityFilters.tsx**: Componente para filtrar actividades
- **Calendar.tsx**: Componente para visualizar actividades en formato de calendario
- **AutocompleteInput.tsx**: Componente reutilizable para campos con autocompletado

Flujo de datos

1. El componente `Activities.tsx` utiliza el hook `useActivities` para obtener los datos
2. El hook `useActivities` utiliza el servicio `activitiesService.ts` para hacer las peticiones
3. El servicio `activitiesService.ts` hace peticiones a la API mediante `api-ky.ts`

Optimizaciones de Rendimiento

1. **Adaptación de Respuestas**:
   - Sistema que detecta automáticamente el formato de la respuesta del backend
   - Soporta múltiples formatos: estándar, array simple, y respuesta paginada de Spring
   - Normaliza los datos a un formato común para garantizar la compatibilidad

2. **Validación y Normalización**:
   - Cada actividad recibida es validada para verificar campos mínimos requeridos
   - Se proporcionan valores por defecto para campos faltantes
   - Se registran advertencias para actividades con formato inválido

3. **Caché y Reducción de Peticiones**:
   - Configuración optimizada de React Query para toda la aplicación
   - Sistema de prevención de peticiones duplicadas
   - Debounce en búsquedas para evitar peticiones mientras el usuario escribe

Autenticación
- **POST /api/auth/login**: Iniciar sesión
- **POST /api/auth/logout**: Cerrar sesión
- **POST /api/auth/refresh**: Refrescar token JWT

Notificaciones y Colaboración
- **POST /api/announcements/global**: Enviar anuncio global
- **POST /api/announcements/department**: Enviar anuncio a un departamento
- **POST /api/announcements/event**: Enviar anuncio de evento
- **POST /api/collaboration/view/{activityId}**: Registrar usuario viendo actividad
- **POST /api/collaboration/edit/{activityId}**: Registrar usuario editando actividad
- **POST /api/collaboration/comment/{activityId}**: Registrar comentario en actividad
- **DELETE /api/collaboration/{activityId}**: Eliminar usuario como visor/editor
- **GET /api/collaboration/viewers/{activityId}**: Obtener usuarios viendo actividad
- **GET /api/collaboration/editor/{activityId}**: Obtener usuario editando actividad

También se proporcionan endpoints equivalentes en la ruta `/auth` para compatibilidad con clientes que no utilizan el prefijo `/api`:

- **POST /auth/login**: Iniciar sesión
- **POST /auth/logout**: Cerrar sesión
- **POST /auth/refresh**: Refrescar token JWT

Usuarios del Sistema

La aplicación incluye múltiples usuarios con diferentes roles:

1. **Usuario Administrador**:
   - **Nombre**: Semper Evincere
   - **Usuario**: admin
   - **Contraseña**: Admin@123
   - **Rol**: ADMIN
   - **Permisos**: Todos los permisos (lectura, escritura y eliminación de actividades y usuarios, generación de informes)

2. **Usuario Asignador**:
   - **Nombre**: Adriana Sanchez
   - **Usuario**: 32345678 (DNI)
   - **Contraseña**: Test@1234
   - **Rol**: ASIGNADOR
   - **Permisos**: Lectura y escritura de actividades, lectura de usuarios, generación de informes

3. **Usuarios Solicitantes y Ejecutores**:
   - Múltiples usuarios con roles SOLICITANTE y EJECUTOR
   - Utilizan su DNI como nombre de usuario y la contraseña "Test@1234"

Para ver la lista completa de usuarios, roles y credenciales, consulte el archivo README-USERS.md.

Autenticación y Seguridad

La aplicación utiliza autenticación basada en JWT (JSON Web Tokens) con los siguientes flujos:

Inicio de sesión

1. El usuario ingresa sus credenciales (nombre de usuario y contraseña) en el formulario de inicio de sesión.

2. El backend valida las credenciales y, si son correctas, genera un token JWT.
3. El token JWT se devuelve al frontend y se almacena en el localStorage del navegador.
4. El frontend incluye el token JWT en todas las solicitudes posteriores al backend.

Cierre de sesión

1. El usuario puede cerrar sesión haciendo clic en su avatar en la esquina superior derecha.
2. En el menú desplegable, selecciona "Cerrar Sesión".
3. El sistema envía una solicitud al backend para invalidar el token JWT.
4. El backend añade el token a una lista negra para evitar su uso posterior.
5. El sistema elimina el token JWT del localStorage y redirige al usuario a la página de inicio de sesión.

Cierre automático por inactividad

1. El sistema monitorea la actividad del usuario (movimientos del mouse, pulsaciones de teclas, etc.).
2. Si el usuario permanece inactivo durante 30 minutos, se muestra un diálogo de confirmación.
3. Si el usuario no responde o elige cerrar sesión, el sistema cierra la sesión automáticamente.
4. Si el usuario elige continuar, se reinicia el temporizador de inactividad.

Mecanismo de Login y Rutas Protegidas

Mecanismo de Autenticación

Backend (Spring Security)

La aplicación utiliza **JWT (JSON Web Tokens)** para la autenticación, con las siguientes características:

1. **Configuración de Seguridad**:
   - Definida en `SecurityConfig.java`
   - Utiliza autenticación stateless (sin sesión en el servidor)
   - CSRF está deshabilitado
   - CORS está configurado para permitir solicitudes desde `http://localhost:3000` y otros orígenes

2. **Rutas Públicas** (no requieren autenticación):
   - `/api/auth/**` y `/auth/**` - Endpoints de autenticación
   - `/api/api-docs/**`, `/api/swagger-ui/**` - Documentación de la API
   - `/api/actuator/**` - Endpoints de monitoreo
   - `/api/h2-console/**` - Consola de la base de datos H2
   - `/api/ws/**` - Endpoints de WebSocket
   - `/api/activities/**` - Endpoints de actividades (temporalmente públicos para pruebas)

3. **Filtro JWT**:
   - Implementado en `JwtAuthenticationFilter.java`
   - Extrae el token JWT de la cabecera de autorización
   - Verifica si el token está en la lista negra
   - Valida el token y establece la autenticación en el contexto de seguridad

4. **Controladores de Autenticación**:
   - `AuthController.java` - Maneja solicitudes a `/api/auth/**`
   - `RootAuthController.java` - Maneja solicitudes a `/auth/**` (sin el prefijo `/api`)
   - Ambos exponen endpoints para login, refresh token y logout

Frontend (React)

1. **Componentes de Autenticación**:
   - `Login.tsx` - Formulario de inicio de sesión
   - `authSlice.ts` - Estado de autenticación en Redux
   - `authService.ts` - Servicios para comunicarse con la API de autenticación

2. **Almacenamiento de Tokens**:
   - Los tokens JWT se almacenan en `localStorage`:
     - `bitacora_token` - Token JWT principal
     - `bitacora_refresh_token` - Token de refresco
     - `bitacora_user` - Información del usuario

3. **Protección de Rutas**:
   - Implementada en `ProtectedRoute` en `App.tsx`
   - Verifica la existencia de tokens en localStorage
   - Redirige a `/login` si no hay tokens

Rutas Protegidas

Backend

1. **Rutas Protegidas por Configuración**:
   - Todas las rutas excepto las mencionadas como públicas requieren autenticación

2. **Rutas Protegidas por Anotaciones**:
   - Utilizan `@PreAuthorize` para verificar permisos específicos

3. **Controladores con Protección**:

   a. **UserController**:
   - `GET /api/users` - `@PreAuthorize("hasAuthority('READ_USERS')")`
   - `GET /api/users/{id}` - `@PreAuthorize("hasAuthority('READ_USERS')")`
   - Otros endpoints también protegidos con permisos específicos

   b. **ActivityController**:
   - `POST /activities` - `@PreAuthorize("hasAuthority('WRITE_ACTIVITIES')")`
   - `PUT /activities/{id}` - `@PreAuthorize("hasAuthority('WRITE_ACTIVITIES')")`
   - `DELETE /activities/{id}` - `@PreAuthorize("hasAuthority('DELETE_ACTIVITIES')")`
   - Algunos endpoints como `GET /activities` están temporalmente sin protección para pruebas

   c. **AnnouncementController**:
   - `POST /department` - `@PreAuthorize("hasRole('ADMIN')")`
   - Requiere rol de administrador

Frontend

1. **Rutas Públicas**:
   - `/login` - Página de inicio de sesión

2. **Rutas Protegidas**:
   - Todas las rutas bajo `/app/*` están protegidas por el componente `ProtectedRoute`
   - Incluyen:
     - `/app` - Dashboard principal
     - `/app/activities` - Lista de actividades
     - `/app/activities/calendar` - Calendario de actividades
     - `/app/activities/new` - Formulario para crear actividades
     - `/app/activities/:id` - Detalles de una actividad
     - `/app/activities/:id/edit` - Formulario para editar actividades
     - `/app/profile` - Perfil de usuario

Flujo de Autenticación

1. **Inicio de Sesión**:
   - Usuario ingresa credenciales en `/login`
   - Frontend envía solicitud a `/api/auth/login` o `/auth/login`
   - Backend verifica credenciales y genera tokens JWT
   - Frontend almacena tokens en localStorage
   - Usuario es redirigido a `/app`

2. **Acceso a Rutas Protegidas**:
   - Componente `ProtectedRoute` verifica tokens en localStorage
   - Si hay tokens, permite acceso a la ruta
   - Si no hay tokens, redirige a `/login`

3. **Solicitudes a API Protegida**:
   - Frontend incluye token JWT en cabecera de autorización
   - Backend valida token con `JwtAuthenticationFilter`
   - Si el token es válido, permite acceso al recurso
   - Si el token no es válido o está en lista negra, rechaza la solicitud

4. **Cierre de Sesión**:
   - Usuario hace clic en "Cerrar sesión"
   - Frontend envía solicitud a `/api/auth/logout`
   - Backend añade token a lista negra
   - Frontend elimina tokens de localStorage
   - Usuario es redirigido a `/login`

Permisos y Roles

1. **Roles de Usuario**:
   - `ADMIN` - Administrador del sistema
   - `ASIGNADOR` - Asigna tareas a ejecutores
   - `SOLICITANTE` - Solicita tareas

   - `EJECUTOR` - Ejecuta tareas asignadas

2. **Permisos**:
   - `READ_ACTIVITIES` - Leer actividades
   - `WRITE_ACTIVITIES` - Crear/editar actividades
   - `DELETE_ACTIVITIES` - Eliminar actividades
   - `READ_USERS` - Leer usuarios
   - `WRITE_USERS` - Crear/editar usuarios
   - `DELETE_USERS` - Eliminar usuarios
   - `GENERATE_REPORTS` - Generar reportes

Seguridad y Manejo de Credenciales

Archivos Sensibles

El proyecto utiliza `.gitignore` para evitar que archivos con información sensible se suban al repositorio. Algunos archivos que nunca deben subirse al repositorio incluyen:

- **Archivos de credenciales**: `gcp.oauth.json`, `*credentials*.json`, `*secret*.json`, etc.
- **Archivos de configuración con secretos**: `.env` (excepto `.env.example`)
- **Claves privadas**: `*.key`, `*.pem`, certificados, etc.
- **Tokens de acceso**: Archivos que contengan tokens de acceso a servicios

Configuración Segura

1. **Variables de Entorno**:
   - Copie `.env.example` a `.env` y complete con sus valores reales
   - Nunca suba el archivo `.env` al repositorio
   - Use valores seguros y complejos para contraseñas y secretos

2. **Credenciales de Google OAuth**:
   - Si necesita usar autenticación con Google, cree un archivo `gcp.oauth.json` basado en la plantilla proporcionada
   - Mantenga este archivo fuera del control de versiones
   - Considere usar variables de entorno en lugar de archivos JSON para entornos de producción

3. **Rotación de Secretos**:
   - Cambie regularmente las contraseñas y secretos
   - Utilice secretos diferentes para entornos de desarrollo, prueba y producción

Buenas Prácticas

- No hardcodee credenciales en el código fuente
- Use variables de entorno o archivos de configuración externos para secretos
- Implemente el principio de privilegio mínimo para accesos
- Revise regularmente los permisos y accesos
- Utilice herramientas de análisis de seguridad como parte del CI/CD

Principios de desarrollo

- Arquitectura hexagonal en el backend
- Arquitectura modular basada en features en el frontend
- Principios SOLID y Clean Code
- Patrones de diseño apropiados para cada contexto
- Testing automatizado
- Seguridad por diseño

Implementación de Sprints

Sprint 1: Arquitectura Hexagonal

- ✅ Nueva estructura de directorios según arquitectura hexagonal
- ✅ Configuración de dependencias Maven
- ✅ Implementación de la capa de dominio (entidades, value objects, eventos)
- ✅ Configuración de infraestructura básica (base de datos, caché, etc.)

Sprint 2: Capa de Infraestructura y Testing

- ✅ Implementación de persistencia con JPA
- ✅ Desarrollo de API REST con Spring MVC
- ✅ Configuración de seguridad con JWT
- ✅ Implementación de tests unitarios y de integración

Sprint 3: Frontend con TypeScript

- ✅ Migración del frontend a TypeScript
- ✅ Implementación de React Query para caché y manejo de datos
- ✅ Optimización de rendimiento con lazy loading y code splitting
- ✅ Mejora de la experiencia de usuario con skeleton loading

Sprint 4: CI/CD y Monitoreo

- ✅ Configuración de CI/CD con GitHub Actions
- ✅ Implementación de contenedores Docker y Docker Compose
- ✅ Configuración de monitoreo con Prometheus y Grafana
- ✅ Implementación de tracing distribuido con Zipkin

Sprint 5: Documentación y Calidad

- ✅ Documentación completa de la API con OpenAPI
- ✅ Mejora de la documentación del código
- ✅ Implementación de análisis de calidad con SonarCloud
- ✅ Corrección de problemas de calidad detectados

Sprint 15: Sistema de Gestión de Tareas

- ✅ Implementación del modelo de datos para el sistema de gestión de tareas
  - ✅ Creación de nuevos roles de usuario (ADMIN, ASIGNADOR, SOLICITANTE, EJECUTOR)
  - ✅ Creación de nuevos estados de actividad (REQUESTED, ASSIGNED, IN_PROGRESS, COMPLETED, APPROVED, REJECTED, CANCELLED)
  - ✅ Implementación de entidades de soporte (ActivityCategory, ActivityHistory, ActivityComment, ActivityAttachment)
  - ✅ Migración de base de datos para nuevas entidades y campos
- ✅ Implementación del flujo de trabajo para SOLICITANTES, ASIGNADORES y EJECUTORES
  - ✅ Implementación del patrón State para estados de actividad
  - ✅ Creación de clases concretas para cada estado
  - ✅ Implementación de reglas de transición entre estados
  - ✅ Desarrollo de validaciones para cada transición
  - ✅ Implementación de servicio para gestionar el flujo de trabajo
- ✅ Corrección de pruebas unitarias
  - ✅ Configuración de datos de prueba para tests
  - ✅ Actualización de la configuración de pruebas para cargar datos SQL
  - ✅ Corrección de errores en la inicialización del contexto de Spring
- ✅ Implementación de sistema de categorización y priorización de tareas
  - ✅ Gestión de categorías con colores personalizables
  - ✅ Sistema de prioridades con niveles configurables
  - ✅ Interfaz de administración para categorías y prioridades
  - ✅ Integración con el flujo de trabajo para clasificar tareas
- ✅ Creación de interfaces específicas para cada rol
  - ✅ Interfaz para SOLICITANTES con formulario adaptado y seguimiento de solicitudes
  - ✅ Interfaz para ASIGNADORES con bandeja de entrada y herramientas de distribución
  - ✅ Interfaz para EJECUTORES con vista de tareas asignadas y reportes de progreso
  - ✅ Sistema de navegación adaptativo según el rol del usuario
- ✅ Implementación de sistema de notificaciones para el flujo de trabajo
  - ✅ Centro de notificaciones con filtros por tipo
  - ✅ Indicador de notificaciones no leídas
  - ✅ Panel de configuración de preferencias de notificaciones
  - ✅ Múltiples canales de notificación (app, email, push)
- ✅ Desarrollo de reportes y métricas para seguimiento de tareas
  - ✅ Dashboard con resumen de tareas y distribución por estados
  - ✅ Gráficos de distribución por categorías
  - ✅ Análisis de tendencias de tiempos de respuesta y completado
  - ✅ Métricas de rendimiento por usuario
- ✅ Preparación para integración futura con Google Calendar y Drive
  - ✅ Interfaces para Google Calendar API
  - ✅ Interfaces para Google Drive API
  - ✅ Panel de configuración para integraciones
  - ✅ Servicios mock para pruebas de integración

- ✅ Documentación completa de arquitectura (diagramas, flujos, componentes)
- ✅ Mejora de la documentación de API con OpenAPI/Swagger
- ✅ Creación de guías de desarrollo y contribución
- ✅ Documentación de decisiones arquitectónicas (ADRs)
- ✅ Configuración de herramientas adicionales de análisis estático
- ✅ Implementación de escaneo de vulnerabilidades
- ✅ Configuración de pruebas de rendimiento con JMeter
- ✅ Implementación de auditoría de seguridad OWASP Top 10

Sprint 6: Mejoras de Desarrollo Local

- ✅ Configuración de entorno de desarrollo local con H2
- ✅ Implementación de scripts para iniciar y detener la aplicación

- ✅ Creación de usuarios de prueba para desarrollo
- ✅ Mejora de la experiencia de desarrollo con hot-reloading
- ✅ Corrección de errores en componentes de React

Sprint 7: Autenticación y Seguridad

- ✅ Implementación de autenticación JWT
- ✅ Funcionalidad de cierre de sesión
- ✅ Temporizador de inactividad para cierre automático
- ✅ Lista negra de tokens JWT en el backend
- ✅ Mejora de la experiencia de usuario con menú desplegable
- ✅ Implementación de rotación de tokens JWT
- ✅ Mejora de validación de contraseñas
- ✅ Implementación de registro de auditoría de seguridad
- ✅ Implementación de validación de tipos para respuestas de API

Sprint 8: Optimización y Mejoras Técnicas

- ✅ Migración de Redux a React Query
- ✅ Implementación de hooks personalizados para operaciones CRUD
- ✅ Generación automática de tipos TypeScript desde OpenAPI
- ✅ Implementación de Spring Data JPA Specifications
- ✅ Optimización de consultas con Projections
- ✅ Nuevos endpoints para estadísticas y resúmenes
- ✅ Implementación de refresh tokens para mejorar la seguridad
- ✅ Manejo centralizado de errores con códigos estandarizados

Sprint 9: Mejoras de Rendimiento y Experiencia de Usuario

- ✅ Implementación de skeleton loaders para mejorar la percepción de velocidad
- ✅ Mejora de transiciones entre páginas con animaciones
- ✅ Implementación de notificaciones toast para acciones del usuario
- ✅ Optimización de carga de imágenes con lazy loading
- ✅ Implementación de sistema de notificaciones en tiempo real
- ✅ Implementación de gestión de sesiones múltiples
- ✅ Detección de sesiones sospechosas
- ✅ Implementación de servidor WebSocket en el backend
- ✅ Autenticación JWT para WebSockets
- ✅ Panel de preferencias de notificaciones
- ✅ Categorización de notificaciones
- ✅ Reconexión automática de WebSockets
- ✅ Indicadores visuales de presencia en actividades
- ✅ Sistema de cola para mensajes no enviados
- ✅ Compresión de mensajes para optimizar rendimiento
- ✅ Niveles de urgencia configurables para notificaciones
- ✅ Visualización especializada para cada tipo de notificación
- ✅ Acciones rápidas en notificaciones

Sistema de Notificaciones Avanzado

El sistema incluye un mecanismo de notificaciones en tiempo real basado en WebSockets para mantener a los usuarios informados sobre eventos importantes:

Tipos de Notificaciones

- **Notificaciones de Asignación de Tareas**: Alertan cuando se asigna una nueva actividad al usuario.
- **Notificaciones de Cambio de Estado**: Informan sobre cambios en el estado de las actividades asignadas.
- **Recordatorios de Fechas Límite**: Avisan sobre actividades próximas a vencer (1 día, 4 horas, 1 hora).
- **Anuncios y Comunicados**: Permiten a los administradores enviar mensajes globales o a departamentos específicos.
- **Notificaciones de Colaboración**: Muestran quién está viendo o editando una actividad en tiempo real.

Arquitectura de Notificaciones

- **Modelo de Dominio**: Jerarquía de clases para diferentes tipos de notificaciones.
- **Eventos de Dominio**: Sistema de publicación/suscripción para desacoplar acciones de sus efectos.
- **WebSockets**: Comunicación bidireccional en tiempo real con autenticación JWT.
- **Servicios Especializados**: Servicios dedicados para cada tipo de notificación.

Características Avanzadas

- **Panel de Preferencias**: Permite a los usuarios configurar qué notificaciones recibir

y cómo.
- **Categorización**: Filtrado de notificaciones por tipo (Tareas, Fechas límite, Anuncios, Colaboración, Sistema).
- **Reconexión Automática**: Sistema de reconexión con backoff exponencial y heartbeat para detectar conexiones zombies.
- **Acciones Rápidas**: Botones de acción directa en las notificaciones para responder sin cambiar de pantalla.
- **Cola de Mensajes**: Sistema de persistencia de mensajes durante desconexiones con prioridades y reintentos automáticos.
- **Indicadores de Presencia**: Visualización en tiempo real de quién está viendo o editando una actividad.
- **Compresión de Mensajes**: Reducción automática del tamaño de los mensajes para optimizar el rendimiento y reducir el consumo de datos.
- **Niveles de Urgencia**: Configuración personalizada de la importancia de cada tipo de notificación con visualización diferenciada.
- **Visualización Especializada**: Interfaz adaptada a cada tipo de notificación con elementos visuales específicos.
- **Recordatorios Interactivos**: Opciones para posponer o descartar recordatorios de fechas límite directamente desde la notificación.
- **Cuenta Regresiva para Eventos**: Visualización de tiempo restante para eventos programados.

Próximos Pasos

Mejoras de Rendimiento y Correcciones de Errores (En Progreso - Sprint 14)

- ⏳ Corrección de errores de autenticación y permisos
  - ✅ Revisar el flujo de autenticación en el frontend
  - ✅ Corregir la forma en que se obtiene y almacena el token
  - ⏳ Implementar manejo de errores más robusto para problemas de autenticación
  - ⏳ Implementar verificación de permisos en el frontend
  - ⏳ Mostrar/ocultar elementos de la interfaz según los permisos del usuario
- ⏳ Mejoras de rendimiento
  - ⏳ Optimizar consultas de base de datos
  - ⏳ Implementar índices adicionales en tablas críticas
  - ⏳ Mejorar el rendimiento del frontend
  - ⏳ Implementar técnicas avanzadas de virtualización
- ⏳ Implementación de pruebas automatizadas
  - ⏳ Configurar Jest y React Testing Library para el frontend
  - ⏳ Configurar JUnit y Mockito para el backend
  - ⏳ Implementar pruebas unitarias para componentes comunes
  - ⏳ Crear pruebas para flujos críticos de la aplicación
- ⏳ Corrección de errores y advertencias
  - ⏳ Corregir advertencias de Checkstyle en el backend
  - ⏳ Resolver problemas de código no utilizado
  - ⏳ Corregir posibles null pointer exceptions
  - ⏳ Resolver advertencias de ESLint en el frontend
  - ⏳ Mejorar la documentación del código

Mejoras Avanzadas del Sistema de Actividades (Completado - Sprint 11)

- ✅ Implementación de filtros avanzados para actividades
  - Filtros por rango de fechas con selección visual
  - Guardado de filtros favoritos
  - Filtros rápidos predefinidos
- ✅ Desarrollo de vista de calendario para actividades
  - Vista mensual implementada con indicadores de actividades
  - Vistas semanal y diaria con indicador de tiempo actual
  - Navegación entre períodos (mes, semana, día) con botones de anterior/siguiente
  - Visualización de actividades por día con indicadores de estado
  - Acceso directo a la creación y edición de actividades desde el calendario
  - Funcionalidad de arrastrar y soltar para cambiar fechas de actividades
  - Sistema de filtros por estado y tipo de actividad
  - Confirmación al mover actividades entre fechas
- ✅ Mejoras en formularios de actividades
  - Autoguardado de borradores
  - Sistema de plantillas para actividades frecuentes
- ✅ Mejoras en la visualización de detalles de actividades
  - Panel expandible en la vista de lista
  - Modal en la vista de cuadrícula
  - Mejor experiencia de usuario al visualizar detalles
- ⏳ Optimización de rendimiento (pendiente para futuros sprints)
  - Mejoras en consultas de base de datos
  - Paginación con cursor para grandes volúmenes de datos
  - Técnicas avanzadas de virtualización
- ⏳ Nuevas funcionalidades (pendiente para futuros sprints)
  - Sistema de comentarios con menciones a usuarios

  - Etiquetas y categorización jerárquica
  - Exportación e importación de datos

Sistema de Notificaciones Avanzado (Completado - Sprint 10)

- ✅ Refactorización del modelo de notificaciones con jerarquía de clases
- ✅ Implementación de notificaciones de asignación de tareas
- ✅ Implementación de notificaciones de cambio de estado de tareas
- ✅ Implementación de recordatorios de fechas límite
- ✅ Implementación de sistema de anuncios y comunicados
- ✅ Implementación de notificaciones de colaboración en tiempo real
- ✅ Implementación del panel de preferencias de notificaciones
- ✅ Mejora del centro de notificaciones con categorización
- ✅ Optimización de WebSockets con reconexión automática

Unificación de Estilos y Corrección de Errores TypeScript (Completado - Sprint 13)

- ✅ Unificación de archivos de estilos duplicados
  - Consolidación de archivos statusColors.ts en una única ubicación
  - Consolidación de archivos theme.ts en una única ubicación
- ✅ Corrección de errores en interfaces de tema
  - Revisión y corrección de la interfaz DefaultTheme
  - Corrección de interfaces ColorScheme, StatusColorMap y TypeColorMap
- ✅ Mejora de la estructura de carpetas
  - Reorganización de componentes compartidos
  - Reorganización de utilidades y hooks
  - Creación de archivos de barril (index.ts) para simplificar importaciones

Refactorización de Arquitectura y Solución de Problemas de Plantillas (Completado - Sprint 12)

- ✅ Consolidación de archivos duplicados
  - Identificación de versiones utilizadas realmente
  - Eliminación de versiones no utilizadas
  - Actualización de importaciones
- ✅ Migración completa a TypeScript
  - Conversión de archivos .jsx restantes a .tsx
  - Adición de tipos e interfaces para todos los componentes
- ✅ Reorganización de la estructura de carpetas
  - Creación de nueva estructura según arquitectura propuesta
  - Movimiento de componentes a ubicaciones correctas
- ✅ Mejora de la gestión de estado
  - Consolidación de estado global con Redux
  - Implementación de React Query para datos del servidor
- ✅ Solución del problema de las plantillas
  - Consolidación de archivos ActivityForm.tsx
  - Verificación de importaciones de componentes de plantillas

Mejoras de Experiencia de Usuario

- Implementar vista de sesiones activas en el frontend
- Mejorar la integración frontend-backend con WebSockets
- Optimizar la virtualización para listas grandes
- Implementar code splitting adicional para mejorar el rendimiento

Mejoras Técnicas

- Optimizar el manejo de errores en solicitudes WebSocket
- Implementar reconexión automática para WebSockets
- Implementar validación de tipos para las respuestas de la API

Documentación Adicional

- [Documentación General](docs/README.md): Índice completo de documentación
- [Arquitectura](docs/architecture/overview.md): Visión general de la arquitectura del sistema
- [Decisiones de Arquitectura](docs/adrs/): Registro de decisiones arquitectónicas (ADRs)
- [Guía de Contribución](docs/contributing.md): Cómo contribuir al proyecto
- [Guía de Estilo de Código](docs/code-style.md): Estándares de codificación
- [Referencia de la API](docs/api-reference.md): Documentación de la API REST
- [CI/CD y Monitoreo](docs/ci-cd-monitoring.md): Configuración de CI/CD y herramientas de observabilidad
- [Guía de Seguridad](docs/security-guidelines.md): Directrices y mejores prácticas de seguridad
- [Pruebas de Rendimiento](docs/performance-testing.md): Resultados y recomendaciones de pruebas de rendimiento
- [Herramientas de Análisis Estático](docs/static-analysis-tools.md): Configuración y uso

de herramientas de análisis estático
- [Despliegue con Docker](docs/docker-deployment.md): Configuración y despliegue con Docker Compose

Monitoreo y Observabilidad

El proyecto incluye una configuración completa de monitoreo y observabilidad:

Prometheus

Prometheus recopila métricas de la aplicación en tiempo real, incluyendo:

- Rendimiento de la JVM (memoria, CPU, garbage collection)
- Tiempos de respuesta de endpoints HTTP
- Tasas de error y códigos de estado HTTP
- Métricas personalizadas de negocio

Acceso: http://localhost:9090

Grafana

Grafana proporciona dashboards visuales para monitorear la aplicación:

- Dashboard principal de Spring Boot
- Visualización de métricas de negocio
- Alertas configurables

Acceso: http://localhost:3000 (usuario: admin, contraseña: admin)

Zipkin

Zipkin permite el tracing distribuido de solicitudes:

- Seguimiento de solicitudes a través de diferentes componentes
- Análisis de latencia y cuellos de botella
- Diagnóstico de problemas en producción

Acceso: http://localhost:9411

CI/CD (Integración y Despliegue Continuos)

El proyecto utiliza GitHub Actions para automatizar el ciclo de vida del desarrollo:

Flujos de trabajo configurados

- **Backend CI**: Compila, prueba y analiza el código Java
- **Frontend CI**: Compila, verifica tipos y analiza el código TypeScript
- **Deploy**: Automatiza el despliegue a producción

Características principales

- **Pruebas automatizadas**: Ejecución de pruebas unitarias y de integración
- **Análisis de código**: Integración con SonarCloud para detectar problemas
- **Construcción de imágenes Docker**: Automatización de la creación de contenedores
- **Despliegue seguro**: Proceso automatizado con verificaciones previas

Para más detalles, consulta la [documentación de CI/CD](docs/ci-cd-monitoring.md).

Equipo

- Desarrolladores backend
- Desarrolladores frontend
- Diseñadores UX/UI
- DevOps

Licencia