Challenge Escuela Rural (Carlos Gil)

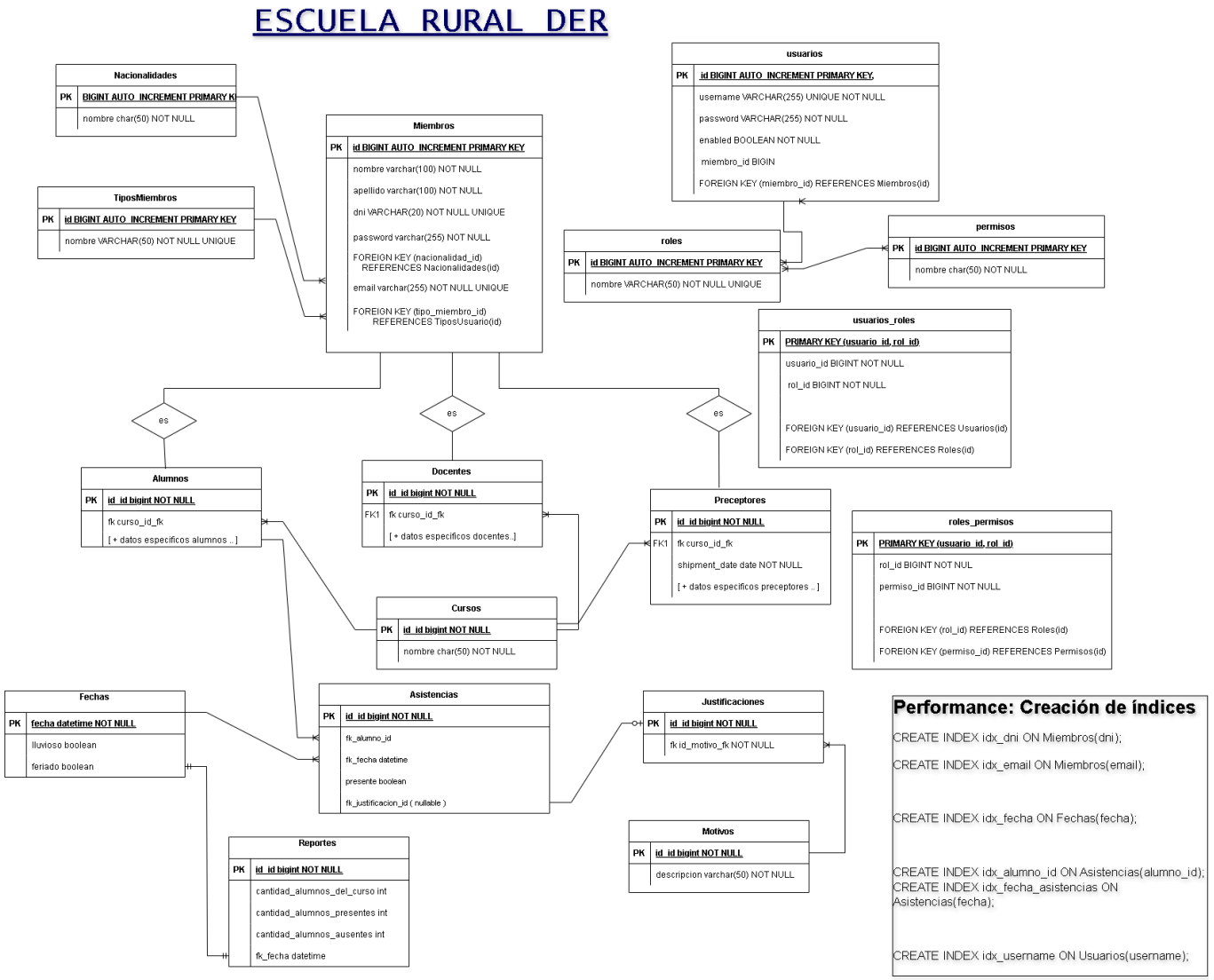
Documento de Entrega y Recomendaciones

Definición de Casos de Uso del Challenge

Para este desafío, se identificaron y desarrollaron los siguientes casos de uso principales:

1. **Gestión de Alumnos**:
    - Registro y actualización de información de alumnos.
    - Consulta de alumnos por curso y fecha de asistencia.
2. **Gestión de Docentes y Preceptores**:
    - Registro y actualización de información de docentes y preceptores.
    - Toma de lista de asistencia diaria y generación de reportes.
3. **Gestión de Asistencia**:
    - Registro de asistencias diarias.
    - Justificación de inasistencias con motivos.
4. **Autenticación y Autorización**:
    - Registro y autenticación de usuarios con JWT.
    - Autorización basada en roles para acceso a funcionalidades específicas.

Modelo de Entidad Relacion (normalizado) como solución a los requerimientos del challenge



Clases Entity Modeladas

Se crearon las siguientes entidades para representar los datos:

1. **Alumno**
2. **Docente**
3. **Preceptor**
4. **Asistencia**
5. **Fecha**
6. **Justificación**
7. **Motivo**
8. **Curso**
9. **Reporte**
10. **Usuario** (para autenticación y autorización)
11. **Role** (enumeración para los roles de usuario)

Clases en el Sistema

Listado de clases:

- AuthControllerTest.java

- EscuelaRuralApplicationTest.java

- TomarLista_GenerarReporte_RolePreceptor_ControllerTest.java

- EscuelaRural.java

- AppConfig.java

- JwtFilter.java

- JwtService.java

- SecurityConfig.java

- AdminController.java

- AuthController.java

- Role.java

- User.java

- Alumno.java

- AlumnoController.java

- AlumnoDTO.java

- AlumnoMapper.java

- AlumnoRepository.java

- AlumnoResponse.java

- AlumnoService.java

- AlumnoServiceImpl.java
- Asistencia.java
- AsistenciaController.java
- AsistenciaDTO.java
- AsistenciaRepository.java
- AsistenciaService.java
- AsistenciaServiceImpl.java
- CalificacionDTO.java
- TareaDTO.java
- TomarListaRequestDTO.java
- UsuarioDTO.java
- Calificacion.java
- Nota.java
- Tarea.java
- CalificacionRepository.java
- NotaRepository.java
- TareaRepository.java
- DataInitializationConfig.java
- JacksonConfig.java
- Curso.java
- CursoController.java
- CursoDTO.java
- CursoRepository.java
- CursoService.java
- CursoServiceImpl.java
- Docente.java
- DocenteController.java
- DocenteRepository.java
- DocenteService.java
- DocenteServiceImpl.java

- EmailAlreadyExistsException.java
- GlobalExceptionHandler.java
- ResourceNotFoundException.java
- Fecha.java
- FechaController.java
- FechaDTO.java
- FechaMapper.java
- FechaRepository.java
- FechaService.java
- FechaServiceImpl.java
- Justificacion.java
- JustificacionController.java
- JustificacionRepository.java
- JustificacionService.java
- JustificacionServiceImpl.java
- Miembro.java
- MiembroController.java
- MiembroDTO.java
- MiembroRepository.java
- MiembroService.java
- MiembroServiceImpl.java
- Motivo.java
- MotivoController.java
- MotivoRepository.java
- MotivoService.java
- MotivoServiceImpl.java
- Nacionalidad.java
- NacionalidadController.java
- NacionalidadRepository.java
- NacionalidadService.java

- NacionalidadServiceImpl.java
- Preceptor.java
- PreceptorController.java
- PreceptorRepository.java
- PreceptorService.java
- PreceptorServiceImpl.java
- Reporte.java
- ReporteController.java
- ReporteDTO.java
- ReporteMapper.java
- ReporteRepository.java
- ReporteService.java
- ReporteServiceImpl.java
- TipoUsuario.java
- TipoUsuarioController.java
- TipoUsuarioRepository.java
- TipoUsuarioService.java
- TipoUsuarioServiceImpl.java
- WeatherService.java
- WeatherServiceMock.java
- AuthenticationRequest.java
- AuthResponse.java
- RegisterRequest.java
- UserRepository.java
- AuthService.java
- AuthServiceImpl.java
- UserService.java
- UserServiceImpl.java
- application.properties
- banner.txt

Recomendaciones y Mejoras

1. **Índices en Bases de Datos**:
   - Agregar índices en los campos que son comúnmente consultados o utilizados en filtros y joins, como `alumnoId`, `fecha`, `cursoId`, `docenteId`, etc.
2. **Carga Inicial de Datos y Pruebas**:
   - La aplicación se autoprueba al iniciar a través de la clase `DataInitializationConfig`. En esta clase se crean datos iniciales y se prueban las funcionalidades básicas como el registro de alumnos, la toma de asistencia y la generación de reportes.
3. **Seguridad**:
   - Implementada con Spring Security y JWT.
   - Roles de usuario: Alumno, Docente, Preceptor, Admin.
   - Controladores protegidos por roles específicos para acceso a funcionalidades.
4. **Controladores y Servicios**:
   - Controladores, servicios, entidades y repositorios creados para cada entidad.
   - Controladores por caso de uso para gestionar funcionalidades específicas.
5. **Pruebas con Postman**:
   - Se han realizado pruebas de endpoints utilizando Postman, incluyendo los endpoints de autenticación y los métodos `get` de todas las entidades.
   - Los endpoints específicos por rol se probaron asegurando que solo los roles adecuados puedan acceder a ellos.
6. **Swagger**:
   - Integrado y configurado dentro del esquema de seguridad para ser accesible y documentar la API.
7. **Consumo del API de Weather**:
   - Implementado el consumo del API de Open Meteo para determinar si es lluvioso o no.
   - Se recomienda implementar un mecanismo de reintentos y un sistema alternativo en caso de fallo de la API o ausencia de conexión a internet.
8. **Dockerización**:
   - Recomendado dockerizar la solución, aunque no se implementó por falta de tiempo.
   - La aplicación se subirá a un repositorio en GitHub junto con este documento y un instructivo de uso.
9. **Instructivo de Uso**:
   - Requisitos previos: Tener MySQL y un esquema llamado `escuelarural`.
   - Instrucciones paso a paso para ejecutar la aplicación y ver las salidas de prueba.
   - Archivo de pruebas de Postman incluido para levantar y probar las APIs.
10. **Pruebas Exhaustivas**:
    - Pendiente realizar un test exhaustivo y robusto con pruebas unitarias por capa y pruebas de integración.
    - Implementación de DTOs para manejar resultados y evitar problemas con JPA.
    - Uso de consultas Hibernate con fetch para mejorar el rendimiento en relaciones complejas.
    - Implementación futura de WebFlux y Kafka para comunicaciones asíncronas y mejora de rendimiento.
11. **Recomendaciones para Aplicaciones Escalables**:
    - Seguridad implementada en un microservicio separado actuando como API Gateway.
    - Microservicios separados para alumnos, docentes y preceptores.
    - Uso de GraphQL para mejorar la performance en consultas.
    - Implementación de procesos batch con Spring Batch para reportes largos y costosos.
    - Uso de programación reactiva con WebFlux para tareas demandantes.