Documentación
Pokemón Bytes

Índice de la Documentación Técnica: Fase I (Backend - Seguridad)

- → **4.2.2. POST /login: Flujo de autenticación y devolución del Token JWT**

- **4.3. Implementación y Verificación del Filtro JWT**
  - → **4.3.1. Estructura del JwtAuthenticationFilter.java**
  - → **4.3.2. Cableado del filtro en SecurityFilterChain (addFilterBefore)**
  - → **4.3.3. Pruebas funcionales: Verificación de las rutas protegidas (/api/v1/juego/estado)**

1. Introducción y Estado del Proyecto

1.1. Propósito de la Fase I.

El propósito de la fase I fue establecer una “Plataforma de Servicio de Autenticación de Alta Disponibilidad” mediante Java/SpringBoot. El objetivo principal era migrar la autenticación de un sistema basado en sesiones sin estado (stateless) haciendo uso de JWT, garantizando que las rutas críticas del juego (/api/v1/juego/**) fueran inaccesibles sin un token válido.

1.2. Resumen de la Arquitectura Implementada (Cliente/Servidor/BD)

Se estableció una arquitectura de microservicio REST de autenticación, siguiendo el patrón de arquitectura en 3 capas para la diferenciación de responsabilidades:

$$Presentación \Leftrightarrow Lógica \Leftrightarrow Datos$$

- **Lógica y Seguridad (Spring Boot):** Responsable del hashing de contraseñas (BCrypt), la gestión de usuarios (JPA), y la emisión/validación de Tokens (JWT).
- **Presentación (React/Postman):** Cliente que consume la API REST, enviando peticiones POST para autenticación.

1.3. Tecnologías de la Fase I (Java/Spring Boot, MySQL, JWT)

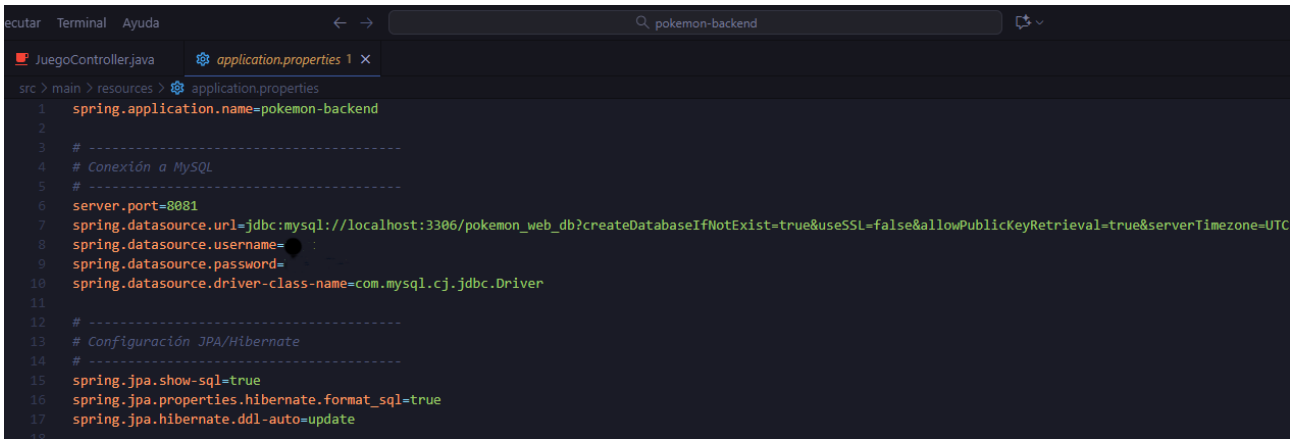
La combinación de Java 21, SpringBoot y JPA/Hibernate dan solidez y rendimiento, mientras que JWT permite escalabilidad horizontal, es decir, los servidores no necesitan compartir el estado de sesión.

2. Configuración del Entorno y Persistencia

2.1. Configuración del Servidor y Base de Datos (BD)

2.1.1. Configuración del driver y conexión a MySQL (Archivo application.properties)

El archivo application.properties actúa como centro de configuración del entorno. En la configuración inicial se incluye la especificación del driver com.mysql.cj.jdbc.Driver y la agrupación de conexiones “HikariCP”, que es gestionado por SpringBoot. Se utiliza el parámetro spring.jpa.hibernate.ddl-auto=update para la creación de tablas, acelerando así el desarrollo.



2.1.2. Justificación del puerto 8081 y parámetros de conexión (allowPublicKeyRetrieval)

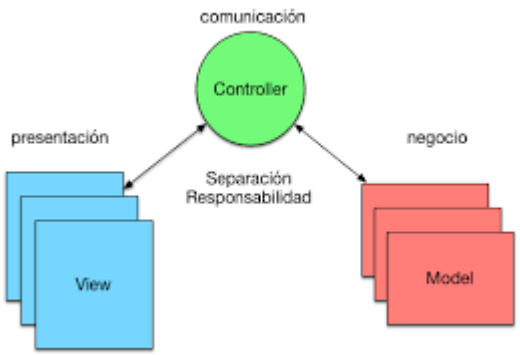
Se hace uso del puerto 8081 con el fin de evitar conflictos con el puerto por defecto de Tomcat (8080) o con el frontend (React) , asegurando la disponibilidad del servicio.

El parámetro allowPublicKeyRetrieval se uso para resolver un conflicto de cifrado SSL entre el conector J de Java y las versiones mas recientes de MySQL(Error 1045/Acceso denegado), para que la capa de lógica pudiera establecer la conexión de datos sin excepción.

2.2 Diseño de la entidad Usuario

2.2.1 Mapeo Objeto-Relacional (ORM) y ddl-auto

La clase Usuario.java actua como el modelo en MVC (patrón de diseño que divide una aplicación en 3 componentes interconectados, para separar responsabilidades) y como la entidad mapeada (@Entity, @Table) por Hibernate/JPA. Permite que Java represente la fila de la tabla USUARIOS. Se utiliza un campo passwordHash con el fin de almacenar un hash cifrado.



2.2.2 Implementación de la interfaz UserDetails

Interfaz: org.springframework.security.core.userdetails.UserDetails

La función de esta interfaz es transformar la entidad Usuario en un "adaptador de seguridad", permitiendo a Spring Security acceder a los datos esenciales para la autenticación.

Se implementan dos métodos clave:

- **getPassword()** para devolver el passwordHash que verifica Bcrypt
- **getUsername()** proporciona la clave de identificación del usuario

2.3 Creación y mapeo del repositorio (UserRepository)

La clase UserRepository.java es una interfaz que extiende JpaRepository<Usuario, Long>. Su función es constituir la capa de abstracción de datos. Hereda varios método CRUD sin requerir codificación SQL.

El método Optional<Usuario> findByUsername(String username) es crucial para la verificación de unicidad en el registro y carga del usuario durante un posible login.

3. Implementación del modulo de Autenticación

3.1 Configuración de Spring Security (SecurityConfig.java)

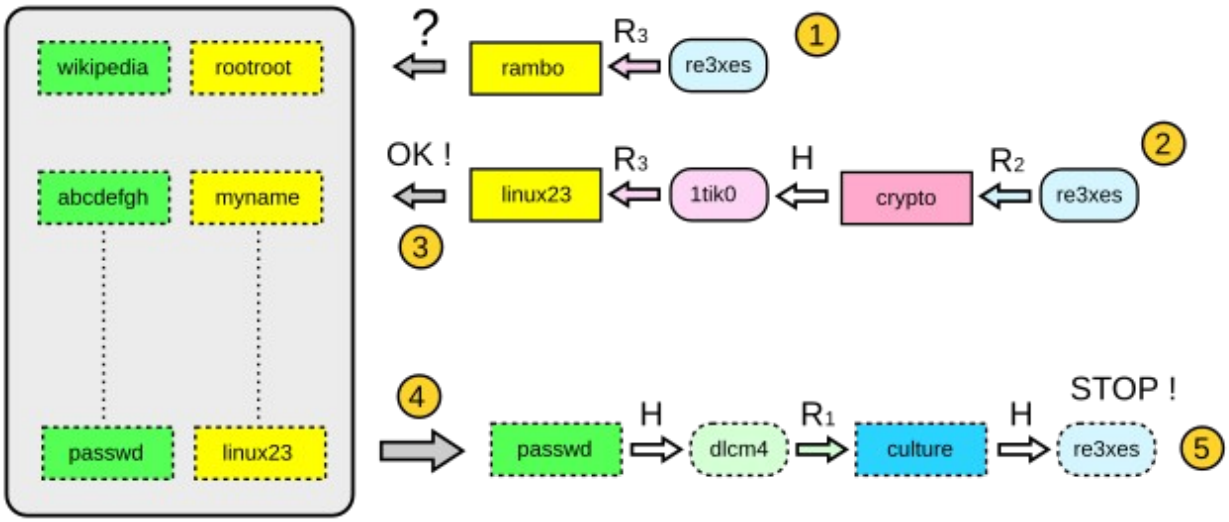
La clase SecurityConfig.java define los Beans de seguridad, hashing y la cadena de filtros HTTP que actúa como un firewall principal para la API

3.1.1 Desactivación de filtros web por defecto (csrf().disable(), sessionManagement(STATELESS))

- **.csrf(csrf → csrf.disable()):** Desactiva la protección contra el Cross-Site Request Forgery. Esta desactivación es obligatoria para cualquier API REST que haga uso de Tokens JWT, ya que la protección CSRF esta diseñada para sesiones basadas en cookies y no es compatible con el modelo Stateless. Su persistencia estaba generando el error "403 Forbidden".
- **.sessionManagement(STATELESS):** Configura el servidor para operara sin estado. Esto significa que el servidor no creara ni almacenara JSESSIONID´s, indicando que la autenticación debe de basarse por completo en el Token JWT enviado por el cliente.

3.1.2 Implementación del PasswordEncoder (BcryptPasswordEnconder)

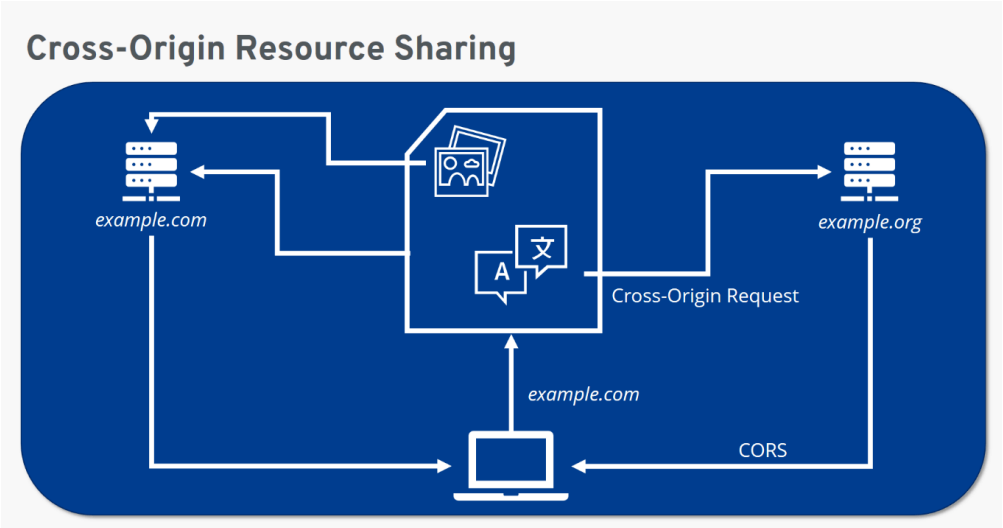
El método @Bean public PasswordEncoder() expone el objeto BcryptPasswordEncoder, generando un valor aleatorio diferente para hash, lo que permite prevenir ataques "Rainbow Table"(técnica utilizada para descifrar contraseñas hasheadas). El uso de este método permite garantizar una robustez criptográfica de las contraseñas en MySQL.



Ejemplo de funcionamiento de "Rainbow Table"

3.1.3 Configuración de CORS para el frontend (React)

Los CORS (Cross-Origin Resource Sharing) son un mecanismo de seguridad implementado por los navegadores web para permitir o bloquear peticiones de recursos entre diferentes orígenes.



El método addCorsMappings() permite que la API (localhosts:8081) acepte solicitudes POST y cabeceras de Authorization provenientes del dominio React que se implementará (localhost:3000), resolviendo problemas de políticas del mismo origen impuestas por los navegadores.

3.2 Flujo de autenticación de la base de datos

3.2.1 Implementación de CustomUserDetailsService (Adaptador de base de datos)

La clase CustomUserDetailsService actúa como un adaptador para la interfaz UserDetailsService. Su método loadUserByUsername() es el puente de datos que hace uso UserRepository para obtener la entidad Usuario de MySQL para verificar credenciales.

3.2.2 Definicion del DaoAuthetication Provider (Bcrypt+MySQL)

El método @Bean public DaoAuthenticationProvider() es el motor principal para la verificación de contraseñas. Su función es encapsular la lógica de comparación de credenciales, combinando CustomUserDetailsService con PasswordEncoder. Lo que hace es tomar la contraseña de texto plano del usuario, la cifra internamente con Bcrypt, y compara el resultado con el hash almacenado en la tabla USUARIOS de MySQL.

3.2.3 Exposición del AuthenticationManager

El método @Bean public AuthenticationManager(AuthenticationConfiguration) expone la interfaz central de Spring Security, siendo el objeto invocado por el AuthController para iniciar el proceso de verificación de credenciales. Garantiza que el controlador pueda delegar la seguridad a Spring con una única llamada, liberando al controlador de la lógica de comparación.

4. Flujo JWT y proteccion de la API

4.1 Servicio de JSON Web Token (JwtService.java)

JwtService.java es el core criptográfico, el responsable de la creación y validación de los tokens.

4.1.1 Justificación de la version 0.12.5 (Resolución de conflicto parserBuilder())

La actualización a JJWT 0.12.5 fue una decisión para la compatibilidad con el entorno Java 21/SpringBoot (Jakarta). El error “parserBuilder() is undefined” se resolvió al asegurar una sintaxis mas moderna en el código: “.parserBuilder().build().parseSignedClaims()”

Esta sintaxis fue crucial para resolver un error durante la ejecución , “500 Internal Server Error”.

4.1.2 Lógica de generación del Token (Configuración de Claims tiempo de expiración)

El método generateToken() construye el Token JWT, haciendo uso de HS56 (algoritmo de firma simétrico) con una clave inyectada.

El Token incluye claims esenciales: sub(subject-nombre del usuario), iat(Issued At) y exp(Expiration). La expiración a 24 horas define la ventana de validez del Token.

4.2 AuthService y AutController (API REST)

4.2.1 POST/register: Lógica de negocio (unidad y cifrado de contraseñas)

El AuthController gestiona la entrada HTTP. El AuthService ejecuta la lógica de negocio pre-persistencia, verificando la unicidad del username con el repositorio y aplica el cifrado Bcrypt. Devuelve “409 Conflict” si la regla de negocio se rompe, aplicando un control de flujo de errores.

4.2.2 POST/login: Flujo de autenticación y devolución del Token JWT

El controlador invoca al AuthenticationManager para la verificación. Si el proceso ocurre con éxito, llama al JwtService para generar el Token, devolviendo la cadena JWT en la respuesta “200 Ok”. En caso de fallo en la autenticación, se traduce en “401 Unauthorized”.

4.3 Implementación y verificacion del filtro JWT.

4.3.1 Estructura del JwtAuthenticationFilter.java

La clase JwtAuthenticationFilter.java implementa OncePerRequestFilter. El método doFilterInternal() es el punto de entrada para la seguridad de cada petición: extrae el Token del header Authorization y lo valida antes de que la petición avance al controlador.

4.3.2 Cableado del filtro en SecurityFilterChain (addFilterBefore)

La línea .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class) tiene como propósito insertar el filtro JWT para que la validación del Token sea el primer mecanismo de autenticación aplicado. Esto asegura que el acceso a rutas protegidas se resuelva de manera inmediata con el Token sin recurrir a una autenticación “usuario/contraseña”.

4.3.3 Pruebas funcionales: Verificacion de las rutas protegidas(/api/v1/juego/estado)

Las pruebas realizadas con “Postman” confirmaron el funcionamiento correcto de la seguridad JWT:

- **Acceso bloqueado (401 Unuathorizaded):** Respuesta obtenida al enviar una petición sin un token valido
- **Acceso concedido (200 OK):** Respuesta obtenida al enviar la petición con un token valido en el header “Authorization: Bearer [Token].

Esto valida que el sistema de seguridad protege de forma funcional y robusta las rutas criticas del juego.