UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo
Unidad Iztapalapa

Integración del Servicio de Login al proyecto RIS-INR

Alumno:
Pedro Misael Rodríguez Jiménez

Profesor:
Alfonso Martínez Martínez

26 de agosto de 2025

Índice

1. Visión General del Servicio de Login

El servicio de Login del RIS-INR fue rediseñado con el propósito de superar las limitaciones de la versión anterior, donde predominaba un modelo acoplado y difícil de mantener. El sistema previo centralizaba la lógica en un único orquestador que mezclaba lógica de negocio con la gestión de autenticación, lo que dificultaba su escalabilidad, la incorporación de nuevas funciones y la mantenibilidad del proyecto. Adicionalmente, el proceso de autenticación presentaba ambigüedad en el uso de identificadores de usuario, manejo inadecuado de datos sensibles y problemas en la administración de sesiones, como su creación temprana en escenarios multi-rol, ausencia de cierre por inactividad y registros incompletos al momento del logout.

La integración actual implementa un servicio más seguro y mantenible, que se apega a principios de separación de responsabilidades y aprovecha JPA para consultas más eficientes. Se reorganizó el modelo de datos para separar credenciales de información personal, se fortaleció la gestión de sesiones con registro de eventos y se corrigieron vulnerabilidades que permitían acceder a vistas sin autenticación. Este rediseño mejora la experiencia del usuario al reducir la latencia en el proceso de login, sienta las bases para una evolución escalable del sistema y proporciona a los equipos de desarrollo y gestión un componente más confiable y sostenible en el tiempo.

- **Mantenibilidad:** separación clara de responsabilidades, reducción de dependencias y eliminación de consultas nativas innecesarias.
- **Escalabilidad:** arquitectura que facilita la incorporación futura de funcionalidades.
- **Buenas prácticas:** uso estandarizado de JPA, normalización de datos de sesión, auditoría de eventos y control más estricto del ciclo de vida de las sesiones.

2. Arquitectura del Servicio de Login

La arquitectura del Servicio de Login en el RIS-INR se diseñó con el objetivo de garantizar mantenibilidad, escalabilidad y apego a buenas prácticas. Esta sección describe de manera detallada la estructura interna del servicio, sus componentes principales, las tecnologías empleadas y los flujos de interacción entre ellos. Se incluyen diagramas de clases y de secuencia para ilustrar el comportamiento dinámico del sistema.

2.1. Vista General por Capas

El servicio sigue una arquitectura en capas típica de aplicaciones basadas en Spring Boot:

- **Controlador (Controller):** Exposición de endpoints REST (`AccessController`).
- **Servicio (Service):** Encapsula la lógica de negocio (`AccessService` y `AccessServiceImpl`).
- **Repositorios (Repository):** Acceso a la base de datos usando Spring Data JPA.
- **Persistencia (Entities):** Entidades JPA que reflejan el modelo de datos.

2.2. Paquetes y Organización

El código fuente se organiza en paquetes que reflejan responsabilidades bien delimitadas:

- `model`: Entidades JPA (`Usuario`, `DatosAcceso`, `Rol`, `Perfil`, `Sesion`, `AreaDeServicio`, `Aplicacion`, `RolAplicacion`, `Evento`, `RegistroEvento`).
- `model.dto.access`: DTOs relacionados al proceso de autenticación (`LoginRequestDTO`, `LoginResponseDTO`, `SelectRolRequestDTO`, `LogoutRequestDTO`).
- `model.dto.shared`: DTOs compartidos (`UsuarioDTO`, `RolDTO`, `AreaDTO`).
- `repository`: Interfaces de acceso a datos que extienden `JpaRepository`.
- `service`: Interfaces y clases de implementación de la lógica de negocio.
- `controller`: Controladores REST (`AccessController`).

2.3. Tecnologías Utilizadas

- **Spring Boot** como framework principal.
- **Spring Data JPA** para el acceso a datos.
- **MariaDB** como sistema gestor de base de datos relacional.
- **JWT (JSON Web Tokens)** para el manejo seguro de sesiones y autenticación.
- **IDE NetBeans** como entorno de desarrollo con Maven y Tomcat embebidos.
- **StarUML** para modelado y generación de diagramas.

2.4. Arquitectura de Componentes

Cada componente cumple un rol definido dentro del servicio:

- **AccessController:** Recibe solicitudes del cliente (login, selección de rol, logout).
- **AccessService:** Valida credenciales, aplica reglas de negocio y coordina la creación o cierre de sesiones.
- **Repositorios JPA:** Ejecutan operaciones sobre la base de datos sin necesidad de queries nativos.
- **Entidades JPA:** Reflejan el modelo de datos, garantizando consistencia entre código y base de datos.
- **DTOs:** Aseguran separación entre el modelo interno y los datos expuestos al frontend.

2.5. Diagramas Estructurales

2.5.1. Entidades

Para ilustrar la arquitectura de persistencia del Servicio de Login se incluyen dos perspectivas complementarias:

- **Modelo físico (ERD):** generado en MySQL Workbench, refleja la estructura real de las tablas relacionadas con el servicio de Login; llaves primarias y foráneas, tipos de datos y relaciones.

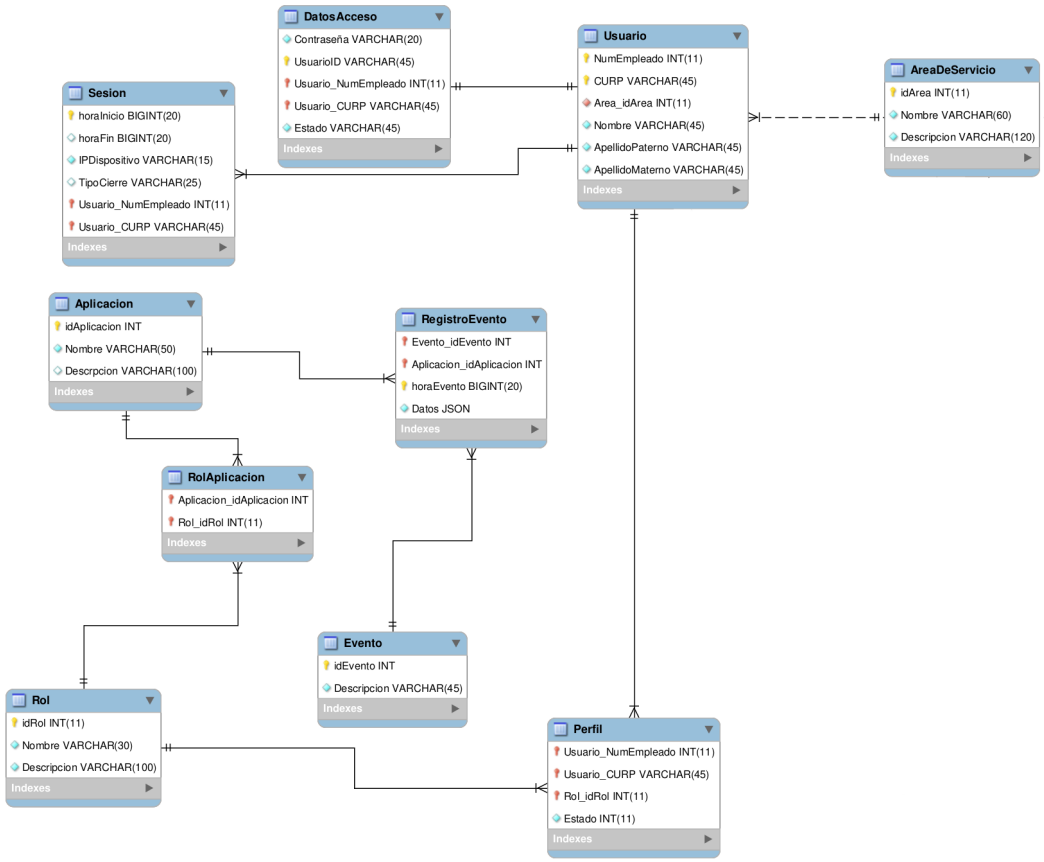

Figura 1: Diagrama de base de datos (Workbench).

- **Modelo lógico (UML):** generado en StarUML, representa las entidades JPA utilizadas en la aplicación. Muestra las clases, atributos, métodos básicos y asociaciones, destacando cómo se implementan las llaves compuestas mediante clases embebidas (`UsuarioPK`, `SesionPK`, etc.).

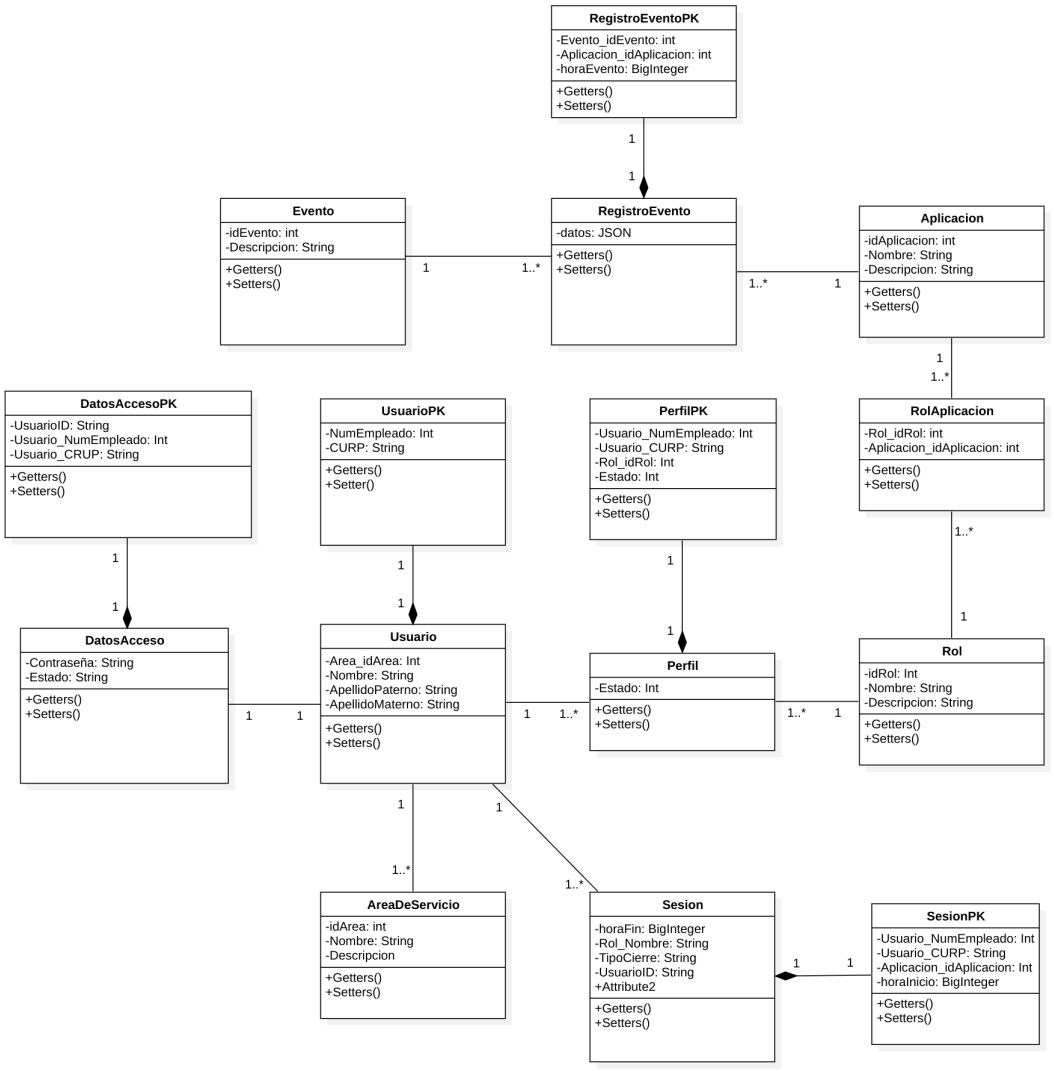

Figura 2: Diagrama de clases UML del modelo de entidades.

A partir de los diagramas anteriores se pueden destacar las siguientes relaciones principales del modelo:

- **Usuario–DatosAcceso (1:1):** cada usuario tiene un único registro de credenciales, lo que asegura separación entre información personal y datos de autenticación.
- **Usuario–Rol (N:M):** la asociación se modela a través de la interrelación `Perfil`, permitiendo que un usuario posea múltiples roles y que un rol sea compartido por distintos usuarios.
- **Usuario–Sesion (1:N):** un usuario puede mantener múltiples sesiones a lo largo del tiempo; cada sesión le pertenece a un solo usuario.
- **Aplicacion–RegistroEvento (1:N):** cada evento registrado se vincula a una aplicación específica, facilitando la auditoría.
- **Rol–Aplicacion (N:M):** gestionada por la interrelación `RolAplicacion`, permite definir qué roles tienen acceso a cada aplicación.

2.5.2. *DTOs*

Además de las entidades de persistencia, el Servicio de Login utiliza Objetos de Transferencia de Datos (DTOs) para estructurar la comunicación entre el backend y el frontend. Estos DTOs permiten exponer únicamente la información necesaria, evitando filtrar atributos sensibles de las entidades JPA y facilitando la serialización a JSON en las respuestas de la API.

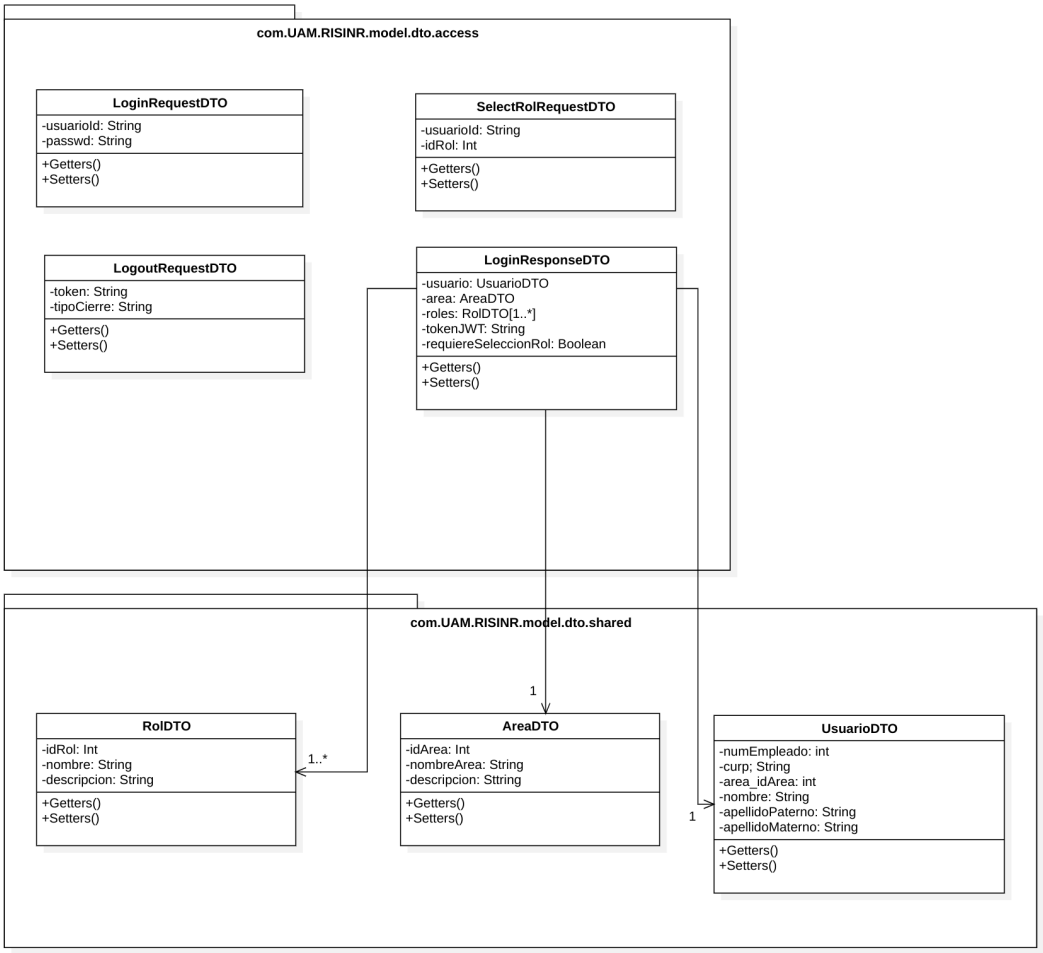

Figura 3: Diagrama UML de los DTOs del Servicio de Login.

Los DTOs se organizan en dos paquetes principales:

- `dto.access`: contiene los objetos específicos para la autenticación.
  - **LoginRequestDTO**: credenciales de acceso enviadas por el usuario.
  - **LoginResponseDTO**: información devuelta al cliente tras la autenticación, incluyendo datos del usuario, área, roles, token JWT y si se requiere selección de rol.
  - **SelectRolRequestDTO**: utilizado cuando un usuario posee múltiples roles y debe seleccionar uno para crear la sesión.
  - **LogoutRequestDTO**: información necesaria para cerrar la sesión (token y tipo de cierre).
- `dto.shared`: agrupa los DTOs reutilizables en distintos servicios del sistema.
  - **UsuarioDTO**: contiene datos identificativos y personales del usuario (sin incluir credenciales).
  - **AreaDTO**: representa las distintas áreas de servicio.
  - **RolDTO**: define los roles asociados a un usuario y sus descripciones.

En conjunto, estos DTOs separan la capa de persistencia de la capa de presentación, reducen el acoplamiento entre entidades y controladores, y aseguran que solo se expongan al cliente los datos necesarios para la autenticación y gestión de sesiones.

2.5.3. Repositorios

El paquete `repository` contiene las interfaces que gestionan la persistencia de las entidades mediante Spring Data JPA. Cada repositorio extiende de `JpaRepository`, lo que permite disponer de operaciones CRUD básicas sin necesidad de implementar consultas manualmente. Además, cuando es necesario, se definen métodos adicionales siguiendo la convención de nombres de Spring Data para generar consultas dinámicas.

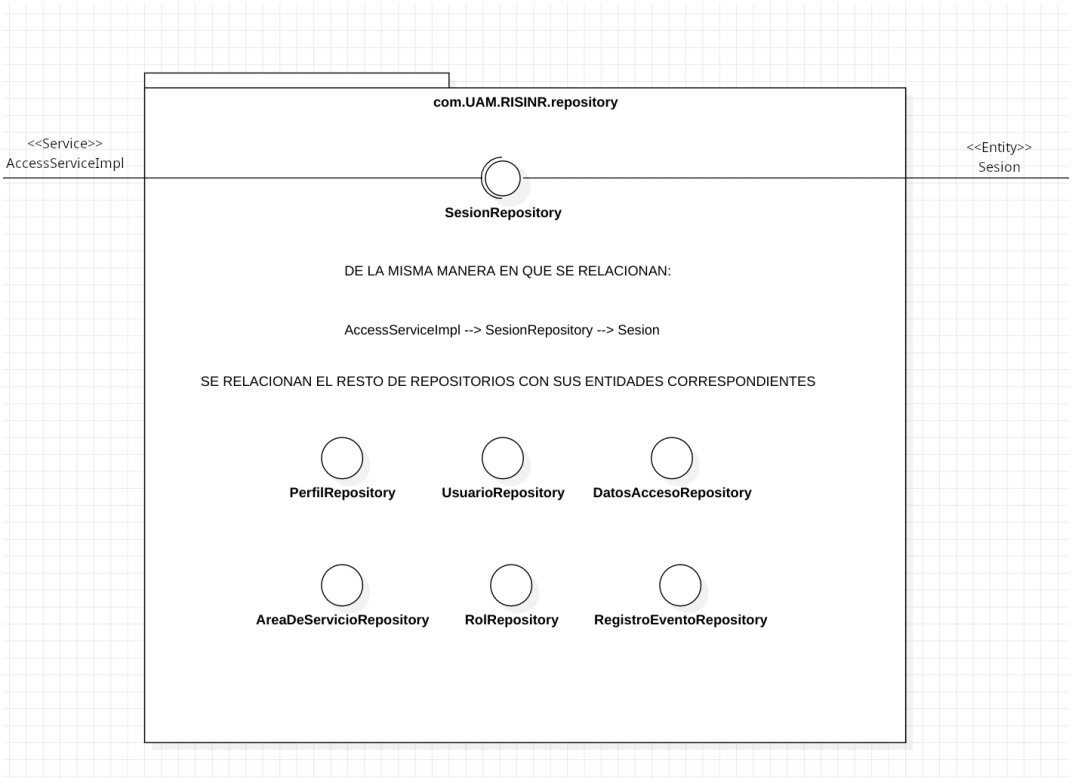

Figura 4: Diagrama del paquete `repository` y su relación con las entidades.

Los repositorios se relacionan de forma directa con sus entidades correspondientes:

- **SesionRepository** → `Sesion`
- **UsuarioRepository** → `Usuario`
- **DatosAccesoRepository** → `DatosAcceso`
- **PerfilRepository** → `Perfil`
- **RolRepository** → `Rol`
- **AreaDeServicioRepository** → `AreaDeServicio`
- **RegistroEventoRepository** → `RegistroEvento`

El patrón de uso es uniforme: el `AccessServiceImpl` interactúa con el repositorio correspondiente para obtener o persistir entidades. Por ejemplo:

`AccessServiceImpl` → `SesionRepository` → `Sesion`

Este mismo patrón se aplica al resto de repositorios, garantizando una capa de persistencia consistente y desacoplada de la lógica de negocio. Gracias a Spring Data JPA, se reduce al mínimo el uso de consultas SQL nativas, mejorando la mantenibilidad y facilitando la evolución futura del sistema.

2.5.4. *Services*

La capa de servicios coordina la autenticación, la auditoría y el ciclo de vida de las sesiones. A continuación se presenta el diagrama, donde se aprecian dependencias explícitas e incorporación de métodos privados de apoyo para mantener la cohesión y reducir complejidad en `AccessServiceImpl`.

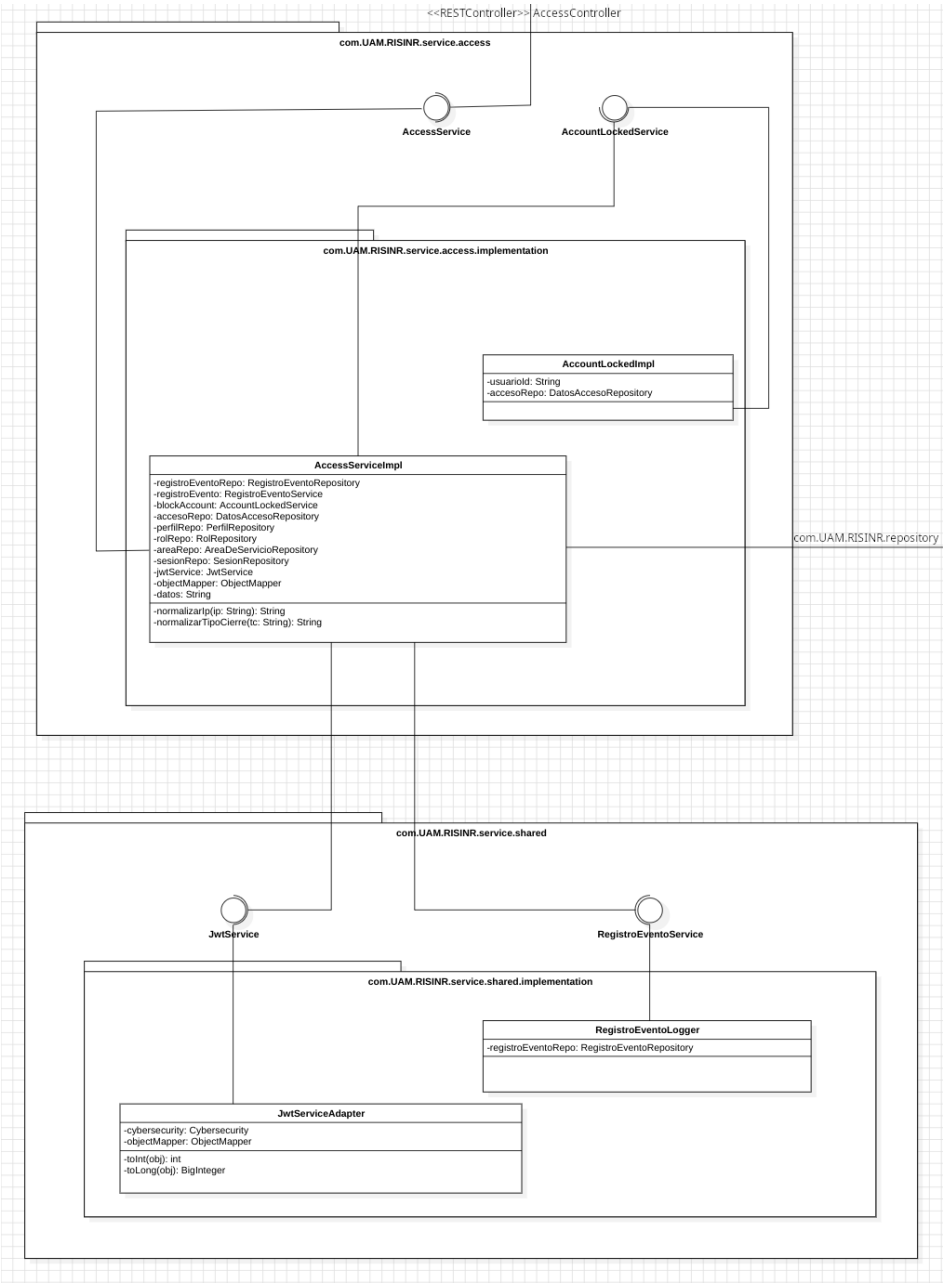

Figura 5: Paquetes `service.access` y `service.shared` con sus implementaciones.

Contratos e implementaciones

- **AccessService** / **AccessServiceImpl**: login, selección de rol, logout; orquesta repositorios, auditoría y emisión de JWT.
- **AccountLockedService** / **AccountLockedImpl**: aplica política de bloqueo sobre `DatosAcceso`.
- **JwtService** / **JwtServiceAdapter**: emisión/validación de tokens, utilidades de conversión.
- **RegistroEventoService** / **RegistroEventoLogger**: Registro de eventos con PK compuesta.

Métodos privados en `AccessServiceImpl` Para mantener la clase centrada en el flujo de autenticación y evitar lógica incidental repetida, se agregaron métodos privados:

- `normalizarIp(ip: String): String`
  Limpia y uniformiza la IP de dispositivo (trimming, límites de longitud, IPv4/IPv6 normalizado). Se usa en login y en la creación de `Sesion`.
- `normalizarTipoCierre(tc: String): String`
  Toma valores de entrada (ej. `normal`, `timeout`, elimina espacios en blanco al final y recorta la cadena en caso de ser de una extensión mayor a la permitida. Se usa en `logout`.
- `datos: String` (campo auxiliar)
  Construcción compacta del JSON con datos del formulario (Usuario y Contraseña), y dirección IP. Se apoya de `ObjectMapper` para serialización segura.

Utilidades en `JwtServiceAdapter` El adaptador de JWT encapsula detalles del paquete de seguridad y ofrece utilidades de conversión usadas en `claims`/tiempos:

- `toInt(obj): int` y `toLong(obj): BigInteger`
  Convierten con validaciones y fallos controlados para evitar *ClassCastException* en lectura/escritura de `claims` (p.ej., `exp`, `iat`, `rolId`).
- Dependencias: `cybersecurity` (proveedor de firma/algoritmo) y `ObjectMapper` (serialización JSON).

2.5.5. *Controlador*

El paquete `com.UAM.RISINR.controller` contiene el controlador REST responsable de exponer los endpoints de autenticación al cliente. Esta capa no contiene lógica de negocio, sino que delega en los servicios de la capa `service`. De esta forma se asegura una clara separación de responsabilidades y se facilita la mantenibilidad.

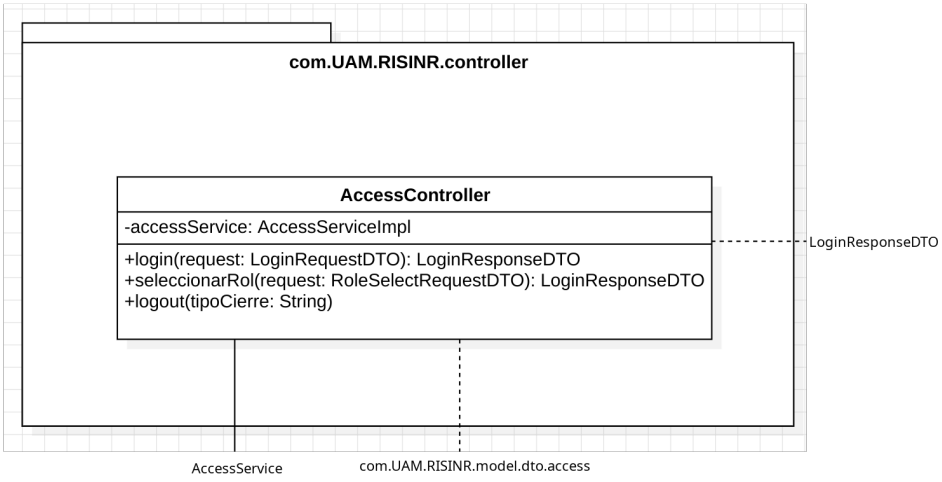

Figura 6: Controlador `AccessController` y sus operaciones principales.

Clase `AccessController`

- **Dependencia:** mantiene una referencia a `AccessServiceImpl`, inyectada mediante Spring.
- **Métodos públicos:**
  - `login(request: LoginRequestDTO): LoginResponseDTO`
    Endpoint `POST /login`. Recibe credenciales, delega autenticación al servicio y retorna datos del usuario, roles disponibles y token JWT.
  - `seleccionarRol(request: RoleSelectRequestDTO): LoginResponseDTO`
    Endpoint `POST /seleccionarRol`. Utilizado cuando un usuario posee múltiples roles; delega en el servicio la creación de la sesión definitiva y el retorno del token JWT.
  - `logout(tipoCierre: String)`
    Endpoint `POST /logout`. Cierra la sesión activa actualizando `horaFin` y `tipoCierre` en la entidad `Sesion`, además de registrar el evento correspondiente en la auditoría.

Notas importantes

- `login` y `seleccionarRol` retornan objetos DTO, evitando exponer directamente entidades JPA.
- Los códigos de error devueltos siguen un conjunto estandarizado (`USER_INVALID`, `PWD_INVALID`, `USER_BLOCKED`, `ROLE_REQUIRED`), lo cual facilita el manejo en frontend.
- La responsabilidad del controlador es mínima: validación básica de parámetros, delegación al servicio, y empaquetado de la respuesta HTTP.

2.5.6. Diagrama de clases completo

Para tener una vista integral del Servicio de Login se incluye un diagrama de clases completo que reúne todos los paquetes descritos anteriormente: entidades de base de datos, DTOs, repositorios, servicios y el controlador. Este diagrama facilita entender la relación entre las diferentes capas antes de analizar el comportamiento dinámico en los diagramas de secuencia.

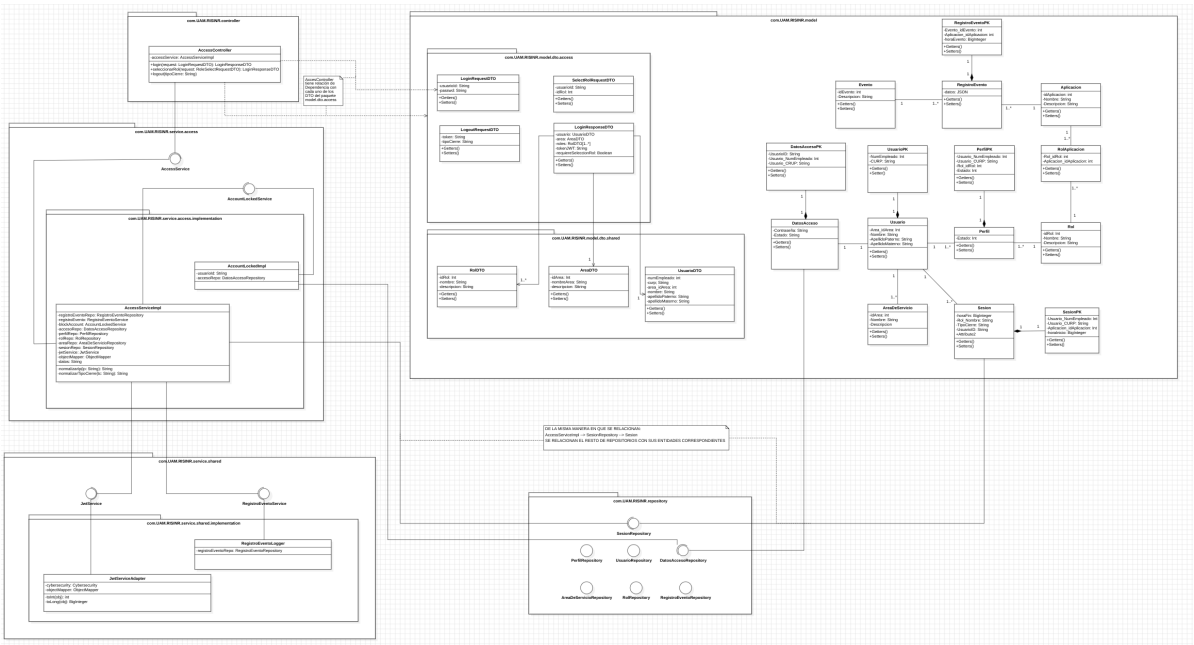

Figura 7: Diagrama de clases del Servicio de Login.

2.6. Comprobación de secuencia de ejecución del Servicio de Login

En esta sección se presentan los diagramas de secuencia que validan el comportamiento dinámico del Servicio de Login. Dado que los diagramas completos resultan extensos, se muestran de manera recortada en fragmentos representativos, acompañados de su explicación. De esta forma se facilita la lectura sin perder detalle en el análisis de cada flujo.

2.6.1. Login fallido

La siguiente figura muestra el flujo de ejecución cuando la autenticación no es válida. Se contemplan tres escenarios distintos:

Caso 1: Usuario inexistente

- El `AccessController` recibe el `LoginRequestDTO` y lo envía al `AccessServiceImpl`.
- El repositorio de `DatosAcceso` no encuentra coincidencias para el `usuarioId`.
- Se registra un evento en `RegistroEvento` indicando intento de acceso con usuario inexistente.
- La respuesta al cliente es un error `401 Unauthorized`.

Caso 2: Cuenta bloqueada

- El `AccessServiceImpl` encuentra un usuario con estado `Bloqueado`.
- Se detiene el flujo de autenticación y se registra el evento correspondiente en `RegistroEvento`.
- El controlador retorna un error `423 Locked`.

Caso 3: Contraseña incorrecta

- El usuario existe y está activo, pero la contraseña no coincide con la almacenada en `DatosAcceso`.
- El intento fallido incrementa el contador de errores de acceso con el mismo `UsuarioID` y misma `ipAddress`.
- Si se supera el umbral configurado, el servicio `AccountLockedService` bloquea al usuario automáticamente.

- Si se bloquea el Usuario se registra el evento de `Bloqueo de usuario`.
- Se registra un evento en `RegistroEvento` indicando intento de acceso con contraseña incorrecta.
- La respuesta inmediata es un error `401 Unauthorized`.

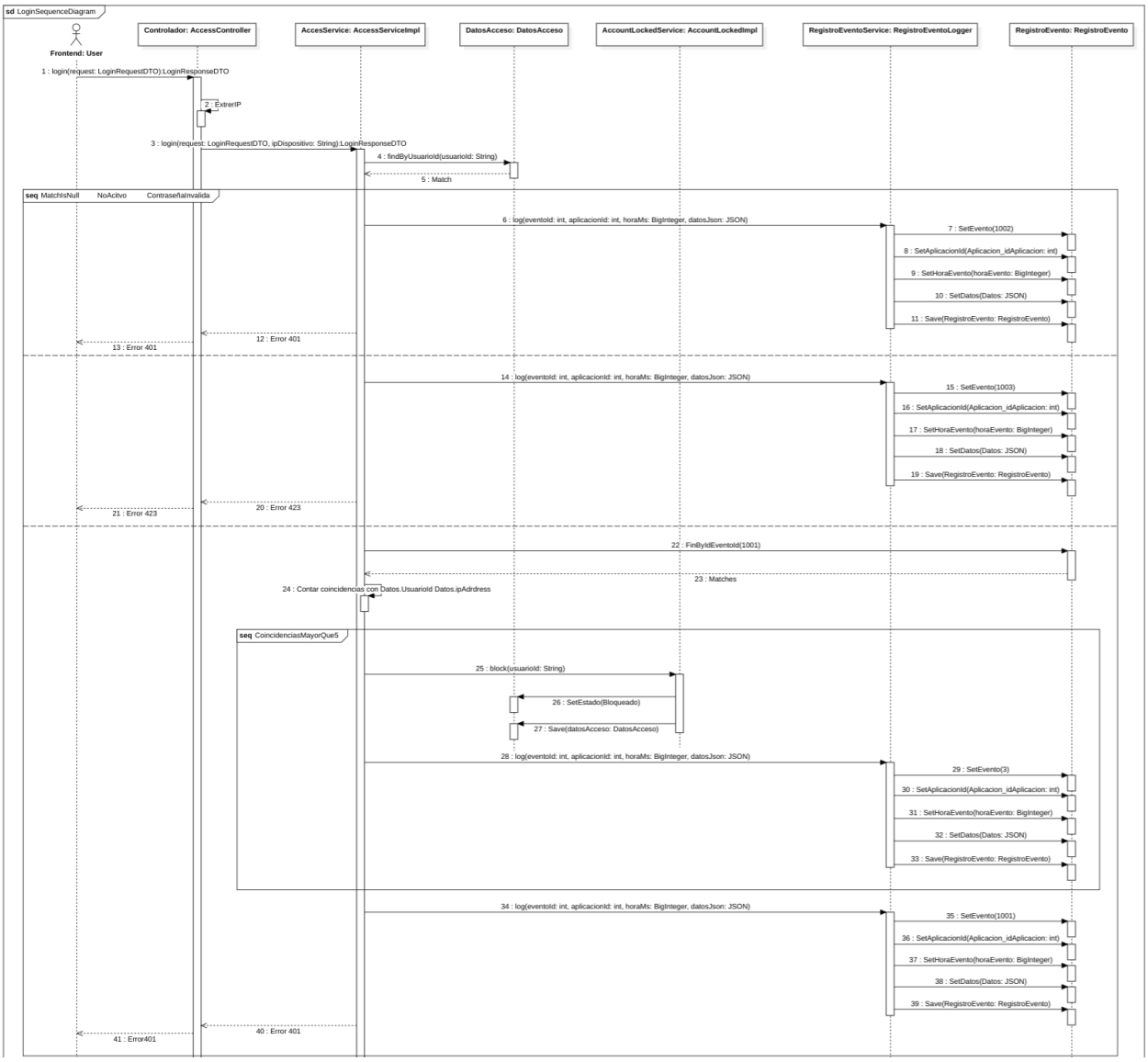

Figura 8: Diagrama de secuencia de los casos de login fallido.

2.6.2. *Login exitoso*

La siguiente figura muestra el flujo de ejecución cuando el usuario es válido, sus credenciales son correctas y el estado de la cuenta está activo. En este escenario se crea una sesión, se emite un token JWT y se registra el evento de login exitoso.

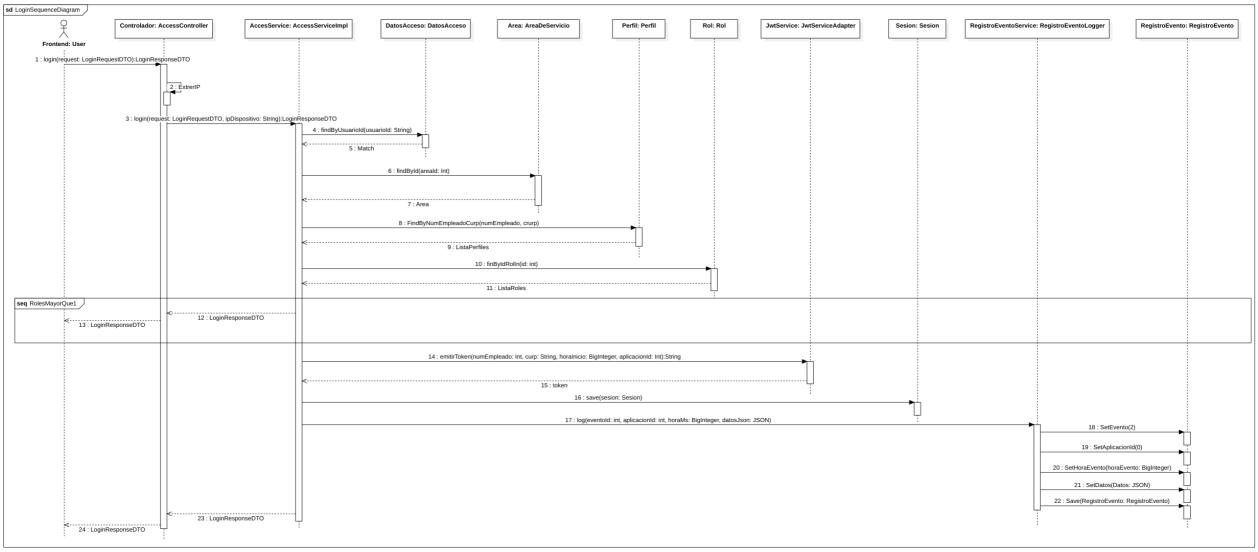

Figura 9: Diagrama de secuencia del caso de login exitoso.

El flujo principal es el siguiente:

- El `AccessController` recibe el `LoginRequestDTO` y lo envía al `AccessServiceImpl`.
- El servicio valida las credenciales contra `DatosAcceso`.
- Si las credenciales son correctas y el usuario está activo:
  - Se consulta el área correspondiente en `AreaDeServicioRepository`.
  - Se obtienen los roles del usuario a través de `PerfilRepository` y `RolRepository`.
- Si el usuario tiene un solo Rol:
  - Se genera directamente un token JWT con la información de sesión.
  - Se crea la entidad `Sesion`, almacenando hora de inicio, IP del dispositivo y rol.
  - Se registra un evento en `RegistroEvento` con evento *login exitoso*.
- Si el usuario tiene mas de un Rol, se responde con un `LoginResponseDTO` indicando la necesidad de seleccionar un rol.

2.6.3. *Selección de Rol*

Cuando un usuario posee múltiples roles, el backend requiere la confirmación explícita del rol con el que iniciará la sesión. El siguiente diagrama muestra el flujo completo de `/seleccionarRol`: validación del rol elegido, generación del JWT con dicho rol, creación de la `Sesion` y registro de evento.

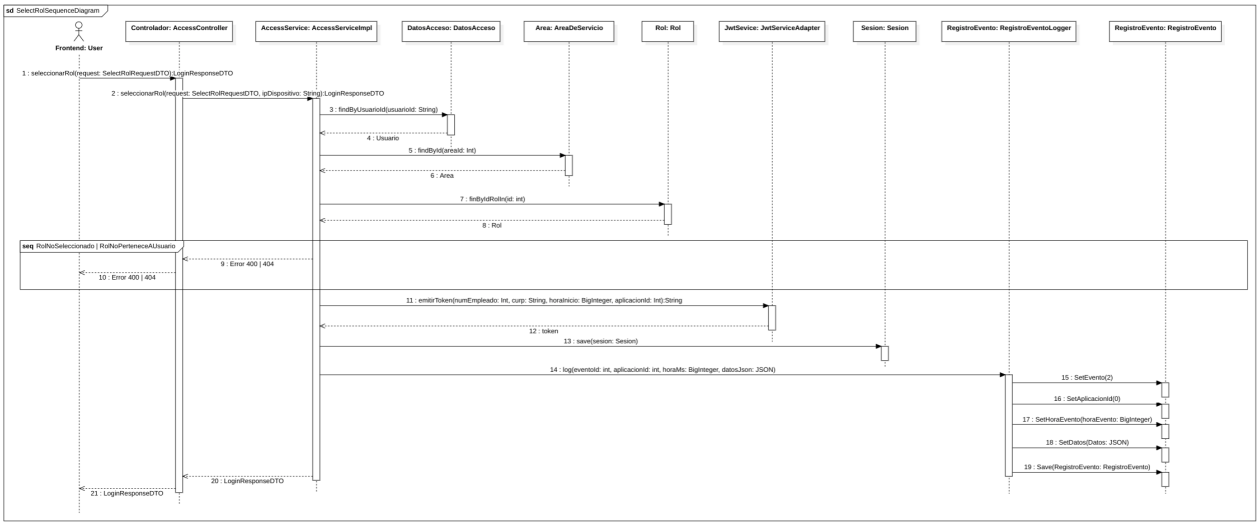

Figura 10: Diagrama de secuencia del endpoint `/SeleccionarRol`.

Flujo principal (éxito):

- El `AccessController` recibe `SelectRolRequestDTO` y delega en `AccessServiceImpl`.
- El servicio revalida al usuario con `DatosAccesoRepository` y obtiene su `Area`.
- Verifica el rol solicitado consultando `RolRepository` (p. ej., `findByIdRol(id)`).
- Si el rol es válido para ese usuario, emite el `JWT` con *claims* que incluyen: `numEmpleado`, `curp`, `idRol`, `horaInicio`, `idAplicacion`.
- Crea `Sesion` con hora de inicio, IP del dispositivo y rol seleccionado.
- Registra el evento de *login exitoso* en `RegistroEvento`.
- Devuelve `LoginResponseDTO` con el `tokenJWT` y los datos de usuario.

Subcasos de error (manejados en el diagrama):

- *Rol no seleccionado* (**400 Bad Request**): el request no incluye un `idRol`.
- *Rol no pertenece al usuario* (**404 Not Found**): el `idRol` existe pero no está asociado al usuario autenticado.

2.6.4. *Logout*

En este flujo se cierra la sesión activa del usuario. El backend parsea el token JWT para identificar la sesión, actualiza su `horaFin` y el `tipoCierre`, y registra la operación. Además, elimina la cookie/token local.

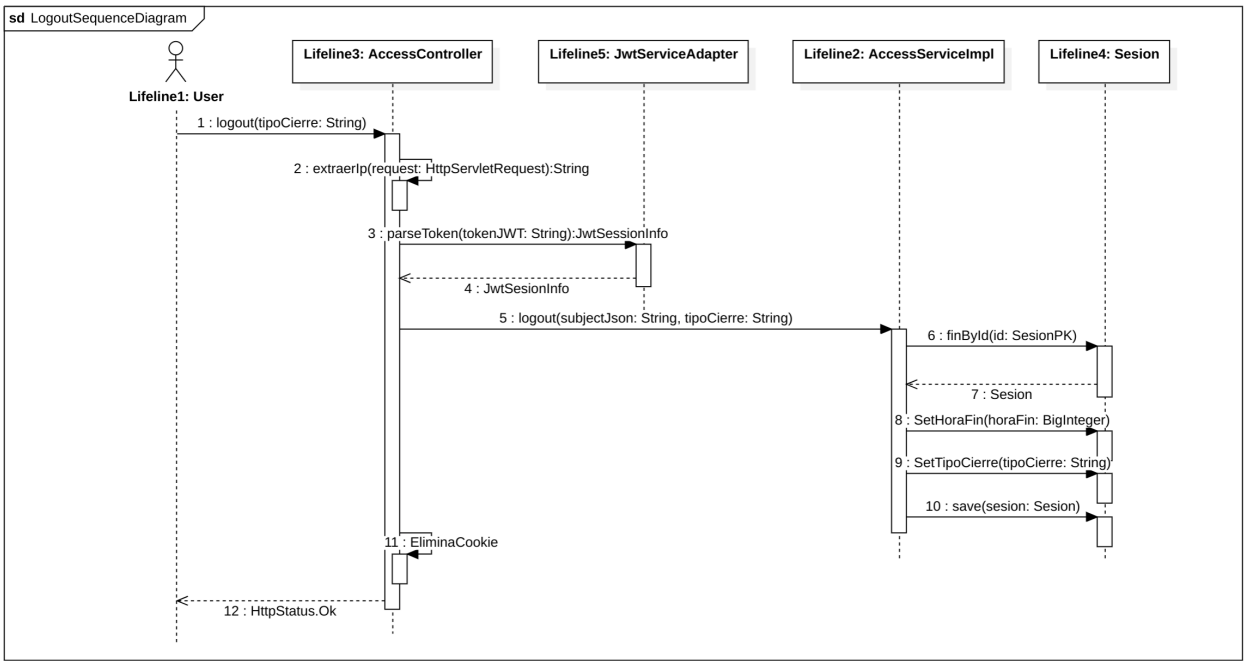

Figura 11: Diagrama de secuencia del endpoint `/logout`.

Flujo principal:

- El `AccessController` recibe la petición con el `tipoCierre` (*natural*, *inactividad*) y extrae la IP del request.
- Se delega a `AccessServiceImpl`, que solicita a `JwtServiceAdapter` el parseo del `tokenJWT` para obtener la información de sesión (`JwtSessionInfo`).
- Con la `SesionPK` obtenida del token, el servicio busca la sesión en `SesionRepository` (`findById`).
- Se actualizan los campos de la entidad `Sesion`: `horaFin` y `tipoCierre` (previamente normalizado).
- Se persiste la sesión actualizada (`save`)
- El controlador borra la cookie y responde `HTTP 200 OK`.

2.7. Mecanismos de Seguridad

El Servicio de Login cuenta con un mecanismo de seguridad integral que combina medidas en el **backend** y en el **frontend** para garantizar la protección de las sesiones, la correcta validación de tokens y el cierre oportuno en caso de inactividad.

Backend (Spring Security)

- **Filtro `OncePerRequestFilter`:** Se implementaron filtros personalizados que interceptan cada petición HTTP para:
  - Validar la firma y vigencia del `JWT`.
  - Renovar el token cuando aún es válido pero está próximo a expirar, reduciendo la necesidad de reautenticaciones.
  - Rechazar peticiones cuando el token es inválido o ha caducado.
- **Filtro `filterChain`:** Define las rutas que requieren autenticación. Evita accesos directos a vistas sin pasar por el proceso de login.
- **Bloqueo de usuarios:** Tras exceder un número configurable de intentos fallidos, el estado del usuario en `DatosAcceso` cambia a *Bloqueado*, hasta su desbloqueo administrativo.

Frontend (JavaScript)

- **`sessionWatch.js`:** Script ejecutado en el cliente que valida cada 5 minutos el estado del token en almacenamiento de sesión:
  - Si el token sigue siendo válido, continúa la sesión de forma transparente para el usuario.
  - Si el token ha caducado, o el backend responde como no válido, la sesión se cierra automáticamente.
  - El cierre de sesión por inactividad redirige al usuario a la pantalla de login con un mensaje de "*Sesión caducada por inactividad*".

Por implementar

- Encriptación de datos en tránsito (UsuarioID y Contraseña) al integrar Gestión de Usuarios.