Servicios de backend

Propósito y alcance

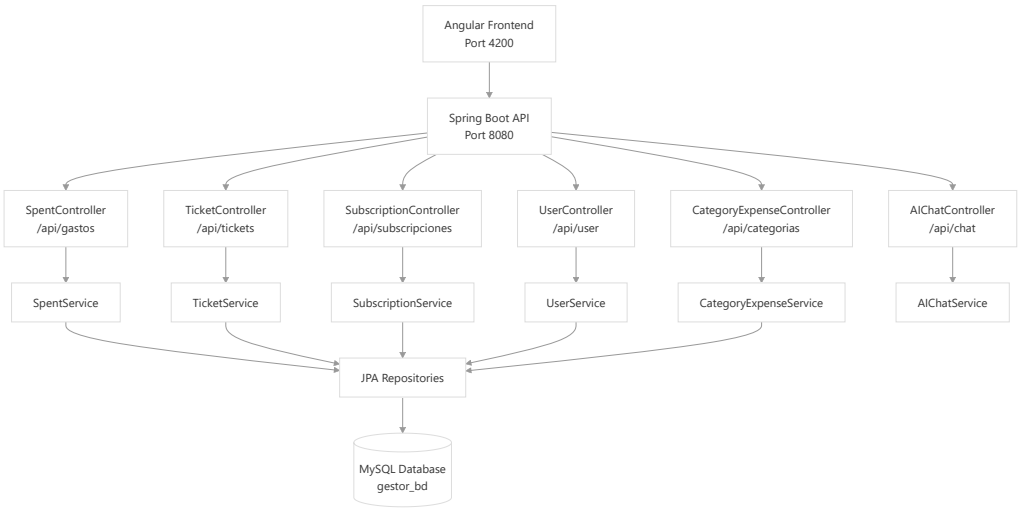
Este documento describe los servicios de la API backend de Spring Boot que proporcionan la lógica de negocio básica y la gestión de datos para el sistema de gestión de gastos. El backend expone puntos finales REST para el seguimiento de gastos, la gestión de usuarios, la integración con OCR y las operaciones administrativas.

Para obtener más información sobre la integración del servicio OCR, consulte Integración de OCR con el backend . Para la integración del cliente frontend, consulte Aplicación frontend . Para una implementación de seguridad completa, consulte Seguridad y autenticación . Para obtener más información sobre la capa de datos, consulte Gestión de datos .

Descripción general de la arquitectura de backend

Los servicios de backend siguen una arquitectura en capas: los controladores gestionan las solicitudes HTTP, los servicios gestionan la lógica de negocio y los repositorios gestionan la persistencia de los datos. El sistema admite múltiples tipos de gastos mediante herencia y proporciona control de acceso basado en roles.

Arquitectura del controlador y flujo de solicitudes



Controladores de API REST

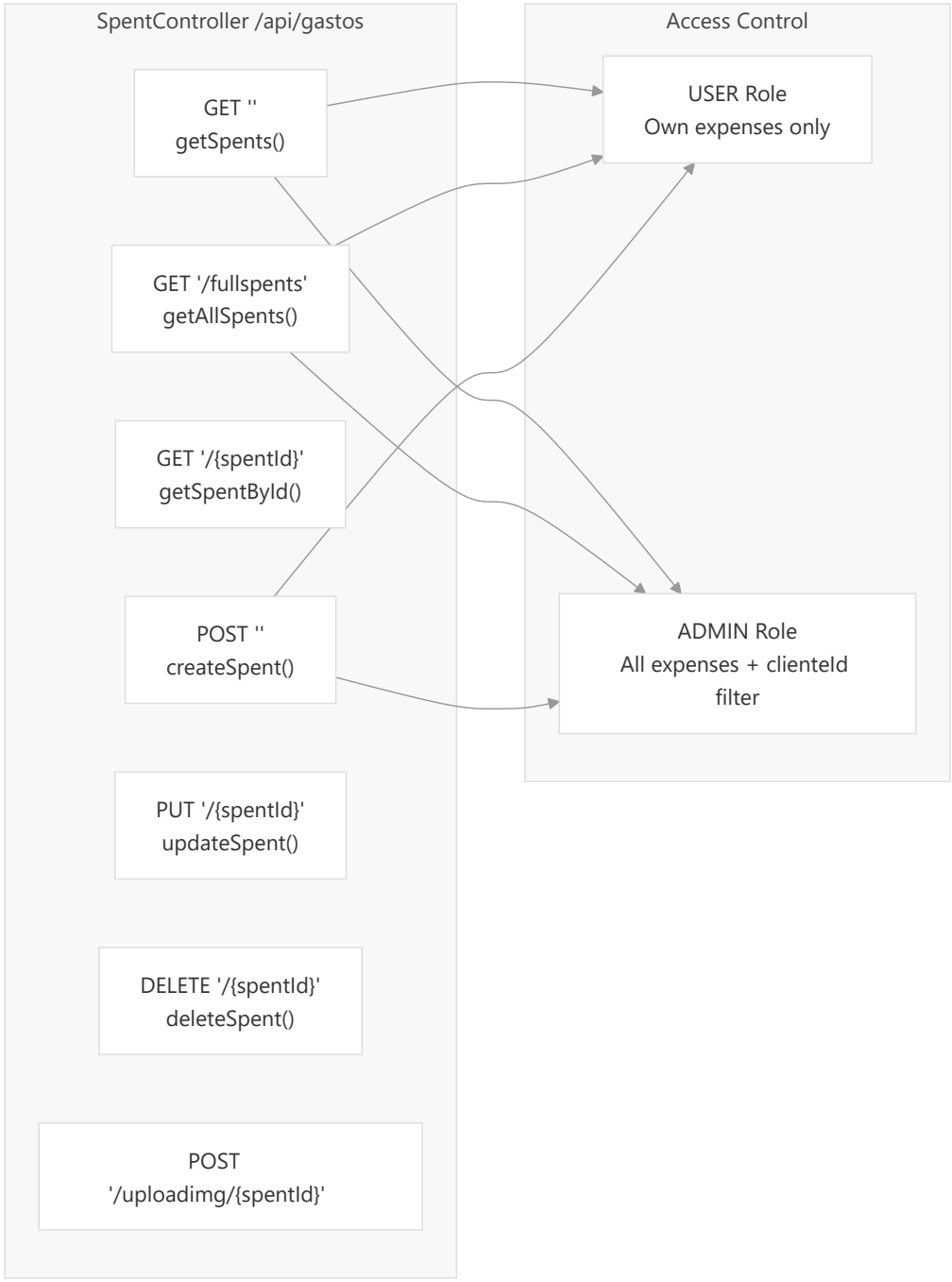
Controladores de gestión de gastos

La funcionalidad principal de gestión de gastos se divide en tres controladores especializados que manejan diferentes tipos de gastos: gastos generales, tickets (recibos procesados por OCR) y suscripciones recurrentes.

Controlador gastado

Gestiona `SpentController` las operaciones generales de gastos y actúa como controlador base para la funcionalidad relacionada con los gastos. Ofrece operaciones CRUD con filtrado basado en roles y funciones de carga de archivos.

Puntos finales clave y control de acceso



El controlador implementa varios patrones importantes:

- **Filtrado basado en roles** : los usuarios que no son administradores solo pueden acceder a sus propios gastos, mientras que los administradores pueden filtrar por `clienteId` parámetro.

  `ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/SpentController.java` 89-97

- **Mapeo DTO polimórfico** : El `mappingSpentFullDto` método maneja diferentes subtipos de gastos (Ticket, Suscripción)

  `ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/SpentController.java` 169-198

- **Integración de carga de archivos** : funcionalidad de carga de imágenes con limpieza automática de archivos antiguos

ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/SpentController.java 393-433

Controlador de tickets

Se `TicketController` especializa en la gestión de tickets, que representan recibos procesados mediante OCR con datos de productos estructurados almacenados como JSON.

Punto final	Método	Control de acceso	Validación de claves
`/api/tickets`	CONSEGUIR	Propietario o administrador	Filtrado basado en roles
`/api/tickets/{ticketId}`	CONSEGUIR	Propietario o administrador	Verificación de propiedad
`/api/tickets`	CORREO	Cualquier autenticado	Productos JSON requeridos
`/api/tickets/{ticketId}`	PONER	Propietario o administrador	Comprobación de consistencia de spentId
`/api/tickets/{ticketId}`	BORRAR	Propietario o administrador	Verificación de propiedad

El punto final de creación de tickets incluye una validación específica para el `productsJSON` campo, que almacena los datos estructurados extraídos por el servicio de OCR.

ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/TicketController.java 154-156

Controlador de suscripción

Gestiona `SubscriptionController` gastos recurrentes con lógica de negocio compleja para cálculos de acumulación y estado de activación.

Las características principales incluyen:

- **Cálculo automático de acumulación** : `accumulate = (total * (iva/100)) + total`

  ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/SubscriptionController.java 169

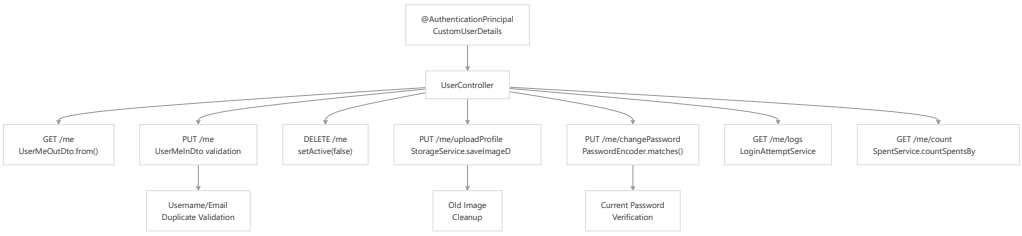
- **Parámetros de facturación recurrente** : los campos `restartDay` , `intervalTime` , y `activa` controlan el comportamiento de la suscripción
- **Herencia de Spent** : Las suscripciones amplían la entidad base Spent con campos adicionales

Gestión de usuarios

Controlador de usuario

Proporciona `UserController` operaciones de autoservicio para usuarios autenticados, centrándose en la gestión de perfiles y la seguridad de la cuenta.

Operaciones de usuario y flujo de datos



El controlador implementa varios patrones de seguridad:

- **Restricción de autoservicio** : los usuarios solo pueden modificar sus propios perfiles

  `ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/UserController.java` `125-127`

- **Validación de duplicados** : evita conflictos entre nombres de usuario y correo electrónico durante las actualizaciones

  `ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/UserController.java` `130-141`

- **Verificación de contraseña** : requiere contraseña actual para realizar cambios

  `ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/UserController.java` `171-173`

Operaciones administrativas

Controlador de gastos de categoría

Gestiona `CategoryExpenseController` categorías de gastos con operaciones de escritura solo para administradores y acceso de lectura universal.

Operación	Nivel de acceso	Validación
CONSEGUIR `/api/categorias`	Todos los usuarios autenticados	Ninguno
CORREO `/api/categorias/`	Solo administrador	@Categoría válidaExpenseDto
PONER `/api/categorias/{categoryId}`	Solo administrador	Existencia de categoría + @Valid DTO
BORRAR `/api/categorias/{categoryId}`	Solo administrador	Existencia de la categoría

Controlador de chat de AIC

Proporciona `AIChatController` una funcionalidad de chat impulsada por inteligencia artificial para brindar asistencia relacionada con los gastos.

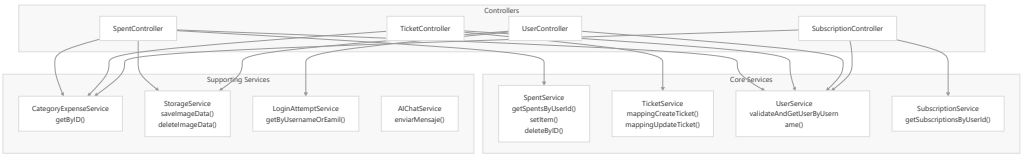
El controlador expone un único punto final `/api/chat/message` que acepta mensajes de texto sin formato y devuelve respuestas generadas por IA a través del `AIChatService.enviarMensaje()` método

`ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/AIChatController.java` `70`

Integración de la capa de servicio

Los controladores dependen de varias interfaces de servicio para la implementación de la lógica empresarial:

Dependencias de servicios e inyección



Todos los controladores siguen el `@Autowired` patrón de inyección de dependencia para el acceso al servicio

`ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/SpentController.java` 54-64

Patrones de autenticación y autorización

Anotaciones de seguridad y manejo de tokens

Todos los controladores utilizan patrones de seguridad consistentes:

- **Autenticación de token de portador** : `@SecurityRequirement(name = BEARER_KEY_SECURITY_SCHEME)` en todos los puntos finales
- **Inyección principal** : `@AuthenticationPrincipal CustomUserDetails currentUser` para el contexto del usuario
- **Documentación de Swagger** : Anotaciones completas de OpenAPI para requisitos de seguridad

Control de acceso basado en roles

El sistema implementa un control de acceso de dos niveles:

Role	Patrón de acceso	Implementación
`RoleServer.USER`	Sólo recursos propios	`user.getId().equals(resource.getUser().getId())`
`RoleServer.ADMIN`	Todos los recursos + filtrado	`user.getRole() == RoleServer.ADMIN`

Patrones de autorización comunes

1. **Verificación de propiedad** :

   `ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/SpentController.java` 226-228

2. **Operaciones solo para administradores** :

   `ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/CategoryExpenseController.java` 88-90

3. **Restricciones de autoservicio** :

   `ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/UserController.java` 125-127

Objetos de transferencia de datos y validación

Patrones de mapeo de DTO

Los controladores utilizan métodos de fábrica estáticos para la conversión de DTO:

- `SpentDto.from(spent)` para la representación de gastos básicos
- `SpentFullDto` con mapeo polimórfico para vistas detalladas

  `ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/controllers/SpentController.java` `169-198`

- `UserMeOutDto.from(user)` para datos de perfil de usuario

Validación de solicitud

Todas las operaciones de creación/actualización utilizan `@Valid` anotaciones con clases DTO personalizadas:

- `CreateSpentRequest` , `UpdateSpentRequest`
- `CreateTicketRequest` , `UpdateTicketRequest`
- `CreateSubscriptionRequest` , `UpdateSubscriptionRequest`
- `UserMeInDto` para actualizaciones de perfil

Inicialización de la base de datos

La `DatabaseInitializer` clase proporciona datos de prueba completos que incluyen usuarios, categorías, gastos, tickets, suscripciones e intentos de inicio de sesión.

`ServidorServicioAPI/GestorAPI/src/main/java/Proyecto/GestorAPI/config/DatabaseInitializer.java` `90-119`

Estos datos solo se insertan cuando la tabla de usuario está vacía, lo que evita la inicialización