Capa de servicio

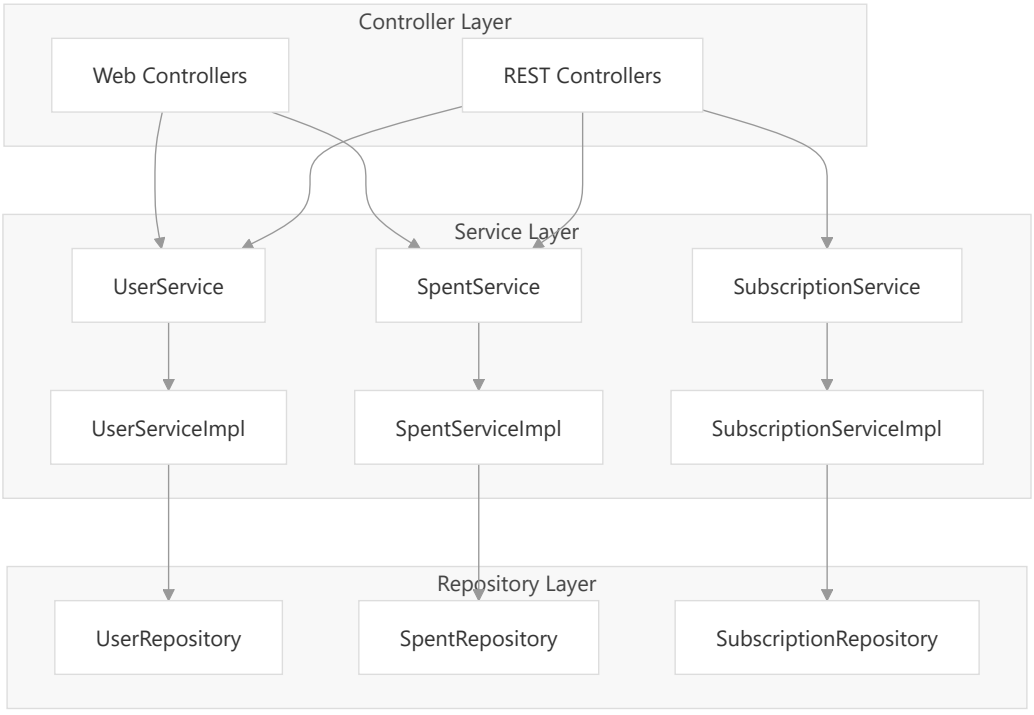
La Capa de Servicio constituye la capa de lógica de negocio de la aplicación ProyectoGestorGastos, implementando las operaciones principales del dominio para la gestión de usuarios, el seguimiento de gastos y la gestión de suscripciones. Esta capa encapsula las reglas de negocio y coordina el acceso a los datos mediante interfaces de repositorio, proporcionando una abstracción clara entre los controladores de la API REST y la capa de persistencia de datos.

Para implementaciones de puntos finales REST que consumen estos servicios, consulte Controladores de API REST . Para operaciones de persistencia de datos, consulte Administración de datos .

Descripción general de la arquitectura de servicios

La capa de servicio sigue el patrón estándar de Spring Service, con diseño basado en interfaces e inyección de dependencias. Cada entidad de dominio tiene su interfaz de servicio correspondiente y una implementación que gestiona las operaciones de lógica de negocio.

Arquitectura de la capa de servicio

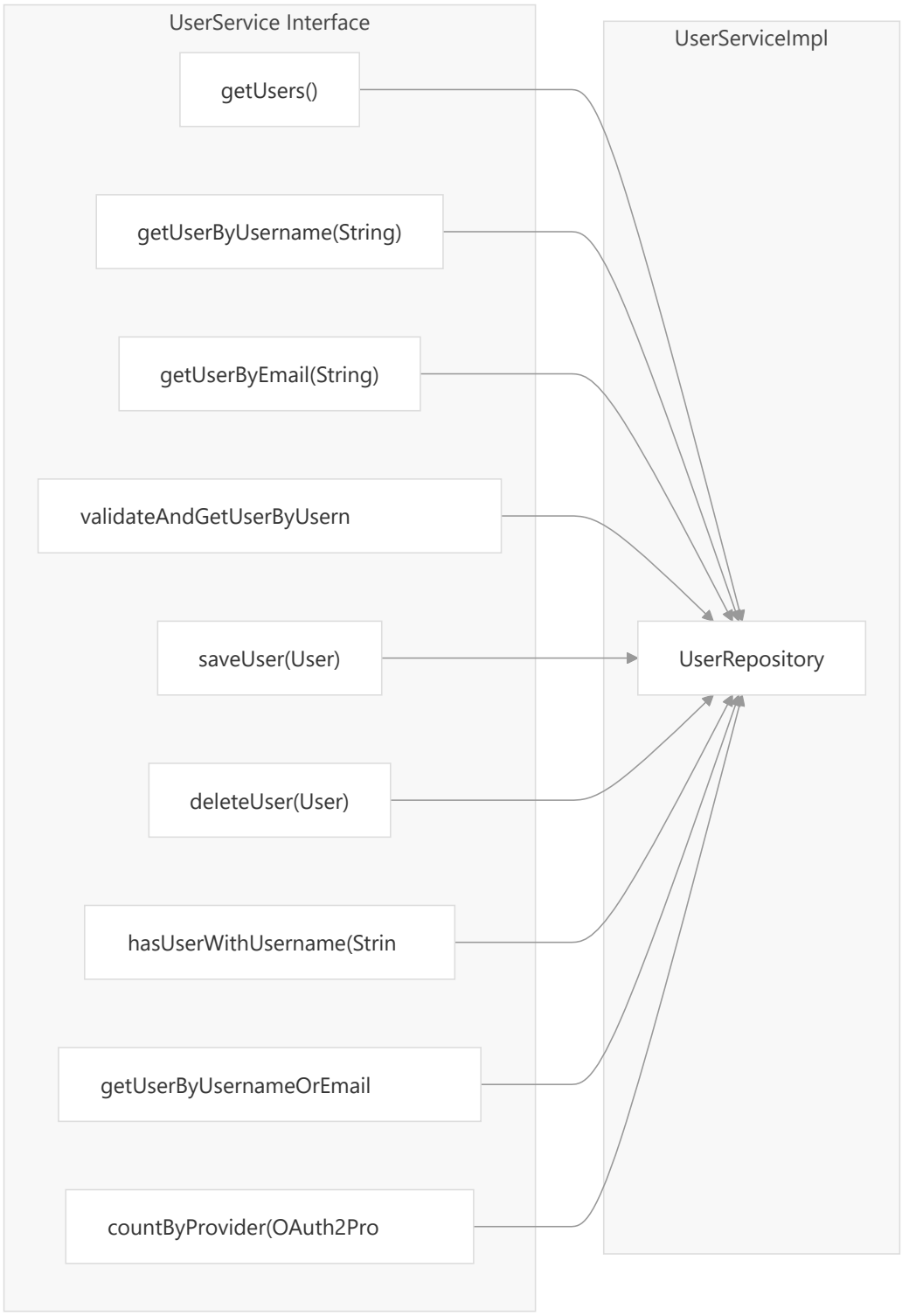


Componentes principales del servicio

Servicio de usuario

La `UserService` interfaz define las operaciones de gestión de usuarios, incluyendo la autenticación, la validación y el seguimiento del proveedor OAuth2. `UserServiceImpl` Proporciona implementaciones concretas utilizando `UserRepository` .

Operaciones de UserService



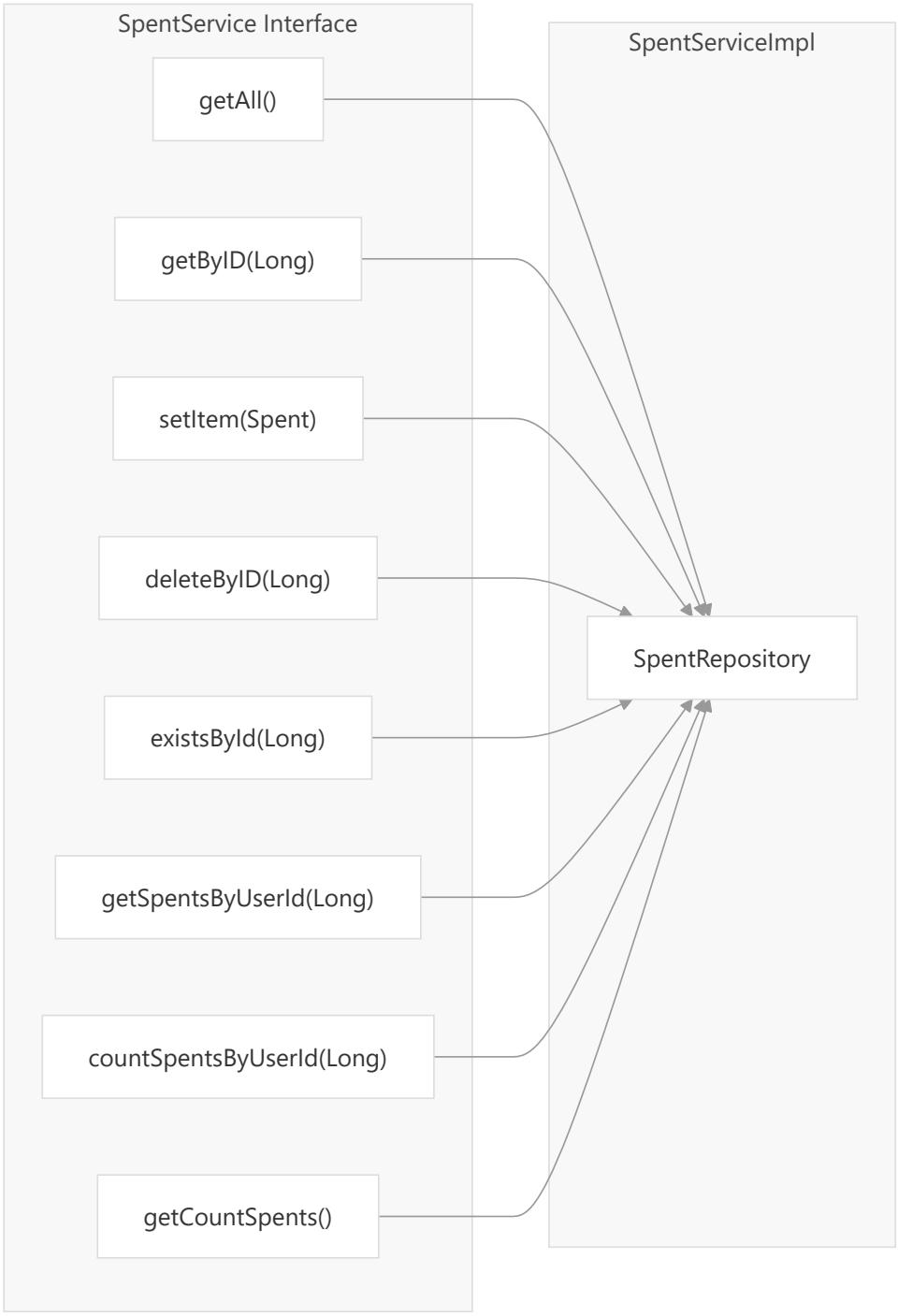
Método	Objetivo	Método de repositorio utilizado
`getUsers()`	Recuperar todos los usuarios	`findAll()`
`getUserByUsername(String)`	Buscar usuario por nombre de usuario	`findByUsername(String)`
`getUserByEmail(String)`	Buscar usuario por correo electrónico	`findByEmail(String)`

Método	Objetivo	Método de repositorio utilizado
`hasUserWithUsername(String)`	Comprobar la existencia del nombre de usuario	`existsByUsername(String)`
`validateAndGetUserByUsername(String)`	Obtener el usuario o lanzar una excepción	`findByUsername(String)` + validación
`getUserByUsernameOrEmail(String)`	Búsqueda flexible de usuarios	Utiliza `isEmail()` el ayudante + el método de búsqueda apropiado
`countByProvider(OAuth2Provider)`	Contar usuarios por proveedor OAuth2	`countByProvider(OAuth2Provider)`

Servicio gastado

Gestiona las `SpentService` operaciones relacionadas con los gastos, ofreciendo funcionalidades CRUD y consultas de gastos específicas para cada usuario. Este servicio gestiona tanto los gastos generales como los específicos de cada ticket.

Operaciones de SpentService

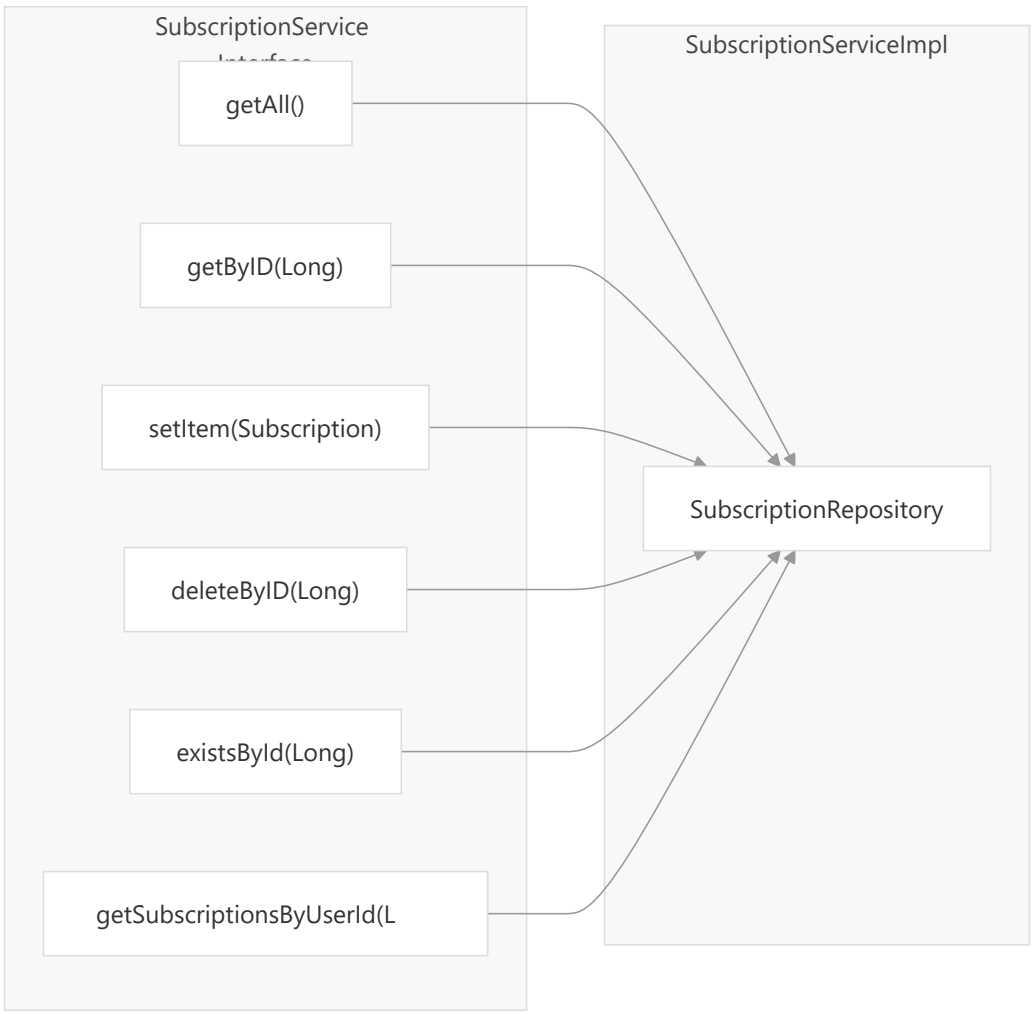


Método	Objetivo	Método de repositorio utilizado
`getAll()`	Recuperar todos los gastos	`findAll()`
`getByID(Long)`	Buscar gastos por ID	`findById(Long)`
`setItem(Spent)`	Guardar o actualizar gastos	`save(Spent)`
`deleteByID(Long)`	Eliminar gasto por ID	`deleteById(Long)`
`getSpentsByUserId(Long)`	Obtener los gastos del usuario	`getByUserId(Long)`
`countSpentsByUserId(Long)`	Contabilizar los gastos del usuario	`countByUserId(Long)`
`getCountSpents()`	Recuento total de gastos	`countGastos()`

Servicio de suscripción

Gestiona `SubscriptionService` suscripciones recurrentes y extiende la funcionalidad de gastos básicos con características específicas de la suscripción, como pagos recurrentes y estado de activación.

Operaciones del servicio de suscripción

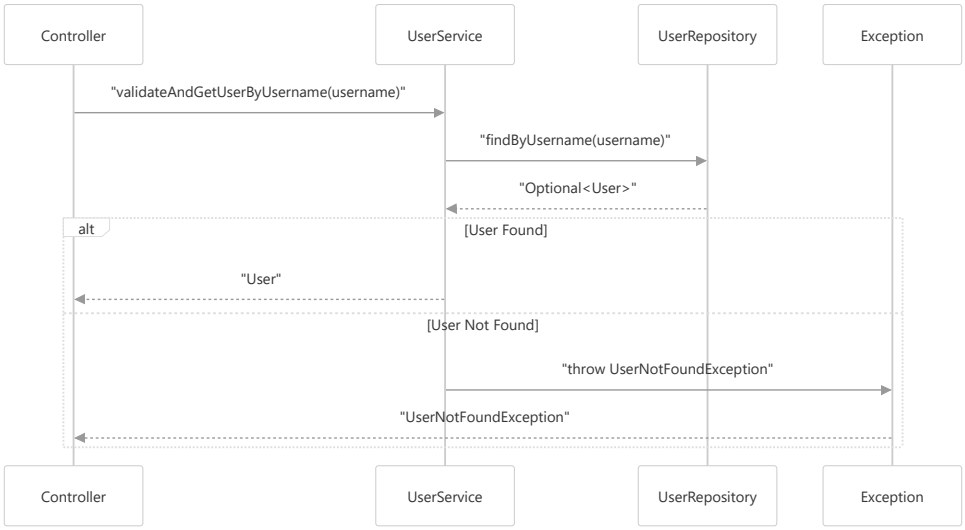


Patrones de lógica empresarial

Validación y manejo de excepciones

La capa de servicio implementa patrones de validación, particularmente `UserServiceImpl` donde métodos como `validateAndGetUserByUsername()` proporcionan un comportamiento a prueba de fallos lanzando `UserNotFoundException` una excepción cuando no se encuentran las entidades.

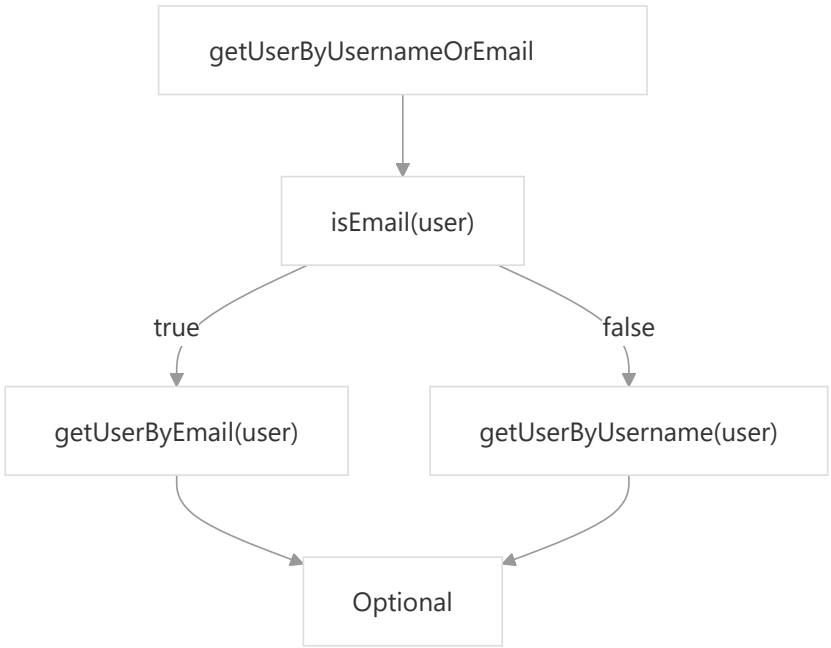
Flujo de validación



Búsqueda flexible de usuarios

Implementa `UserServiceImpl` una lógica de búsqueda de usuarios sofisticada `getUserByUsernameOrEmail()` que determina si el parámetro de entrada es un correo electrónico o un nombre de usuario utilizando el `isEmail()` método auxiliar.

Flujo de decisión de búsqueda de usuarios



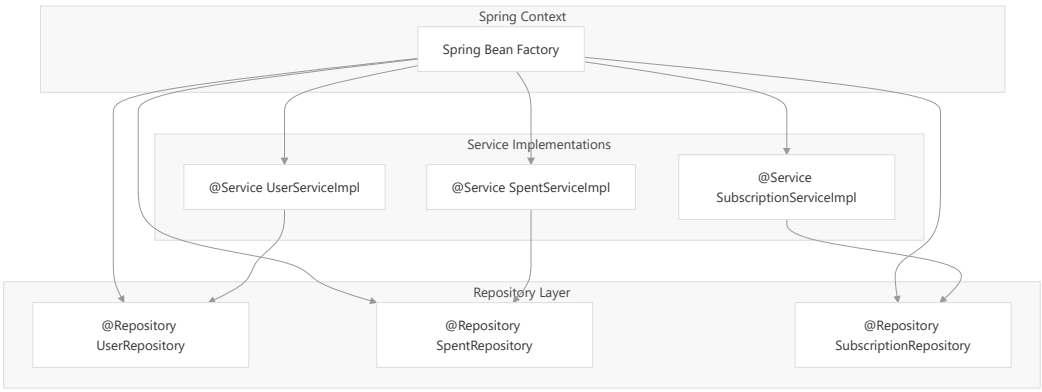
Patrones de integración de servicios

Inyección de dependencia del repositorio

Todas las implementaciones de servicios utilizan inyección de dependencia basada en constructor (a través `@RequiredArgsConstructor` de in `UserServiceImpl`) o inyección de campo (a través de

`@Autowired` in `SpentServiceImpl` y `SubscriptionServiceImpl`) para acceder a las instancias del repositorio.

Relaciones entre servicios y repositorios



Soporte estadístico y analítico

Los servicios proporcionan métodos de agregación para el análisis del sistema, como `countByProvider()` las estadísticas de uso de OAuth2, `countSpentsByUserId()` las métricas de gastos de los usuarios y `getCountSpents()` el recuento total de gastos del sistema. Estos métodos respaldan la funcionalidad del panel administrativo.

Servicio	Método de análisis	Objetivo
Servicio de usuario	`countByProvider(OAuth2Provider)`	Seguimiento del uso del proveedor OAuth2
Servicio de usuario	`getCountUsers()`	Número total de usuarios activos
Servicio gastado	`countSpentsByUserId(Long)`	Estadísticas de gastos de usuario
Servicio gastado	`getCountSpents()`	Recuento de gastos de todo el sistema