METACONTI

Desarrollo de Aplicaciones Multiplataforma | Pablo Rey Ramos

ÍNDICE

Introducción

Este documento contiene los diagramas UML técnicos acerca del modelo de la aplicación *MetaConti*, proyecto integrado de Pablo Rey Ramos para el curso de Desarrollo de Aplicaciones Multiplataforma.

La aplicación web *MetaConti* nace fruto de la inspiración obtenida en la observación del entorno comercial. En particular, se nutre de un nicho de mercado muy concreto: la subcontratación. Más allá de una empresa delegar ciertas tareas en otra, esta relación comercial acarrea numerosas pesquisas a nivel administrativo, legal y jurídico; no son pocas las firmas que han sabido ver una tierra fértil en este terreno pantanoso.

Como ejemplo de MVP, *MetaConti* se enfoca en la gestión documental de las subcontratas. Esto implica la creación, modificación y cese de relaciones comerciales, así como la subida, modificación y borrado de documentos. Lo más importante es la validación: administrativos de la empresa contratista validarán la documentación que suba un empleado de una subcontrata.

En este ámbito, se precisa detallar los actores implicados, la interacción entre los componentes y las entidades que sostienen el modelo. Por ello, se emplearán diversos diagramas del lenguaje UML para explicar la casuística de la aplicación al más bajo nivel.

Diagrama de casos de uso

A continuación se muestra el diagrama de casos de uso de la aplicación. Se destacan dos puntos clave que pueden dar pie a confusión:

- El actor *Admin. de la BBDD* hereda TODOS los casos de uso del *Administrativo*, ya que siempre podrá registrarse e iniciar sesión como los demás. Sin embargo, sólo él podrá iniciar un nodo de empresas.
  De forma natural, un administrativo sólo podrá registrar una nueva empresa al iniciar una subcontratación. Ya que todos los usuarios administrativos deben trabajar para alguna empresa, un usuario administrador de la base de datos debe crear las empresas de los usuarios si no están registradas previamente.
- Las relaciones *include* apuntando al inicio de sesión indican que para los casos de uso base, es necesario haber iniciado sesión con anterioridad, es decir, es obligatorio para el flujo real. Del mismo modo, la relación *extend* al crear una empresa subcontratada quiere decir que se puede registrar una nueva empresa, pero no es estrictamente necesario (puede subcontratar una de las ya existentes en la base de datos).

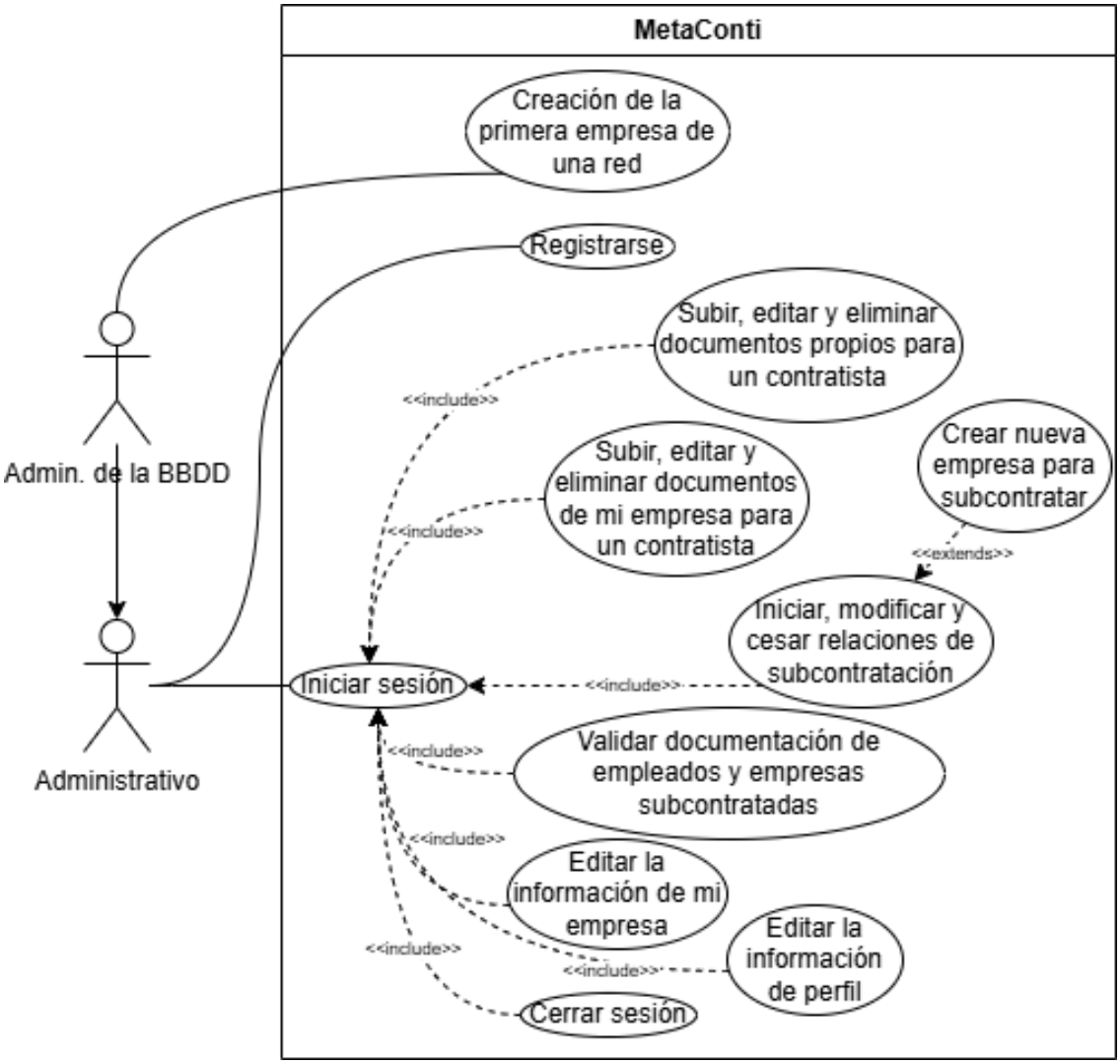


Diagrama de clases

Ya hemos detallado las posibles acciones de cada actor que interactúe con nuestro sistema: ahora especificaremos las clases que definirán las entidades fuertes para nuestra capa de persistencia en base de datos.

Tal como queda representado, la entidad *Company* es el eje central de todo el esquema, sin la cual no existirían contratos, empleados ni documentos.

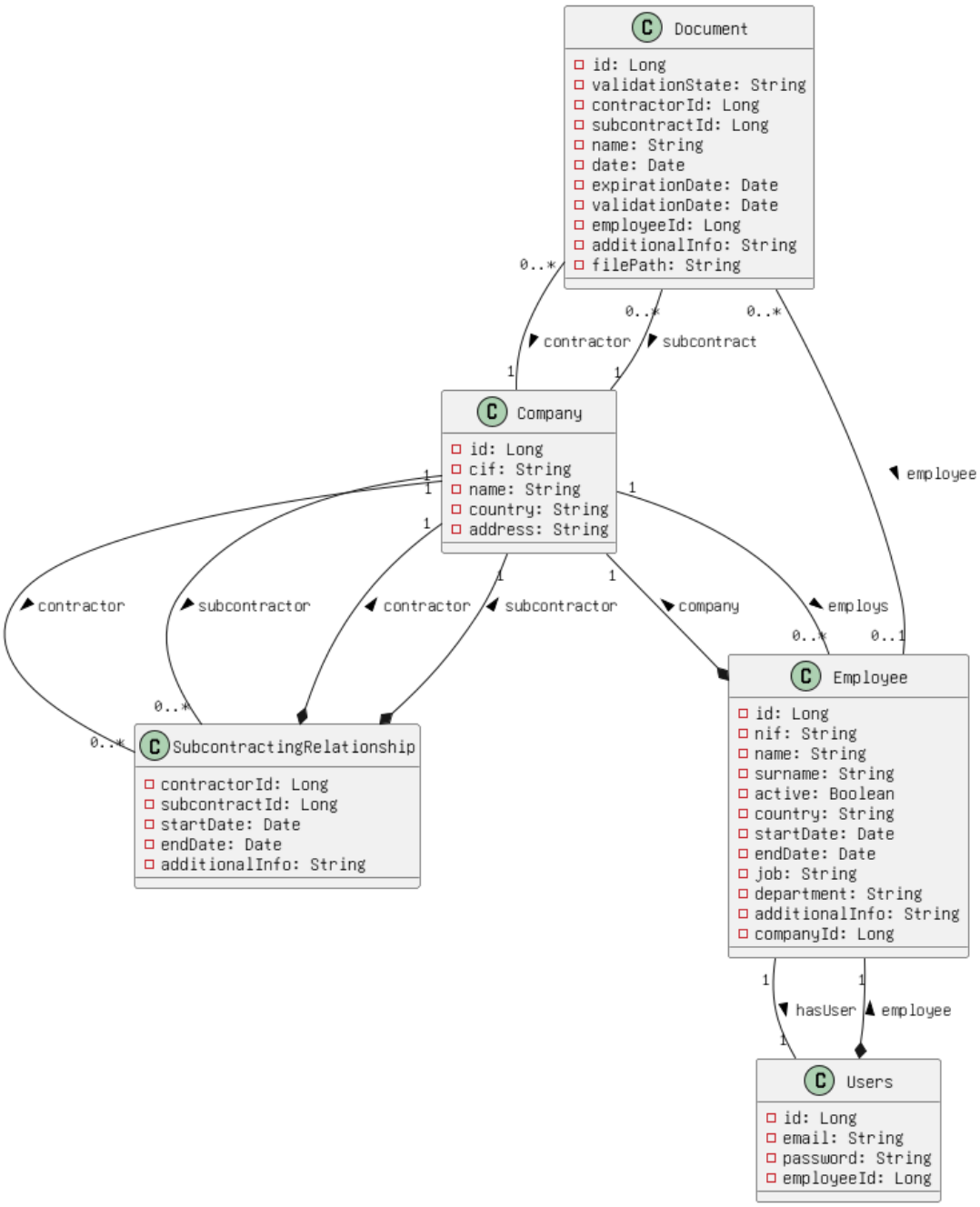


Diagrama entidad-relación

Con este diagrama, profundizaremos en las tablas que debe tener nuestra base de datos; en especial, este tipo de esquema UML hace hincapié en la forma en que se relaciona una tabla con otra. De esta forma, por medio de expresiones verbales y de lenguaje humano, podemos entender no tanto la lógica de la relación, sino la naturaleza intrínseca de la misma.

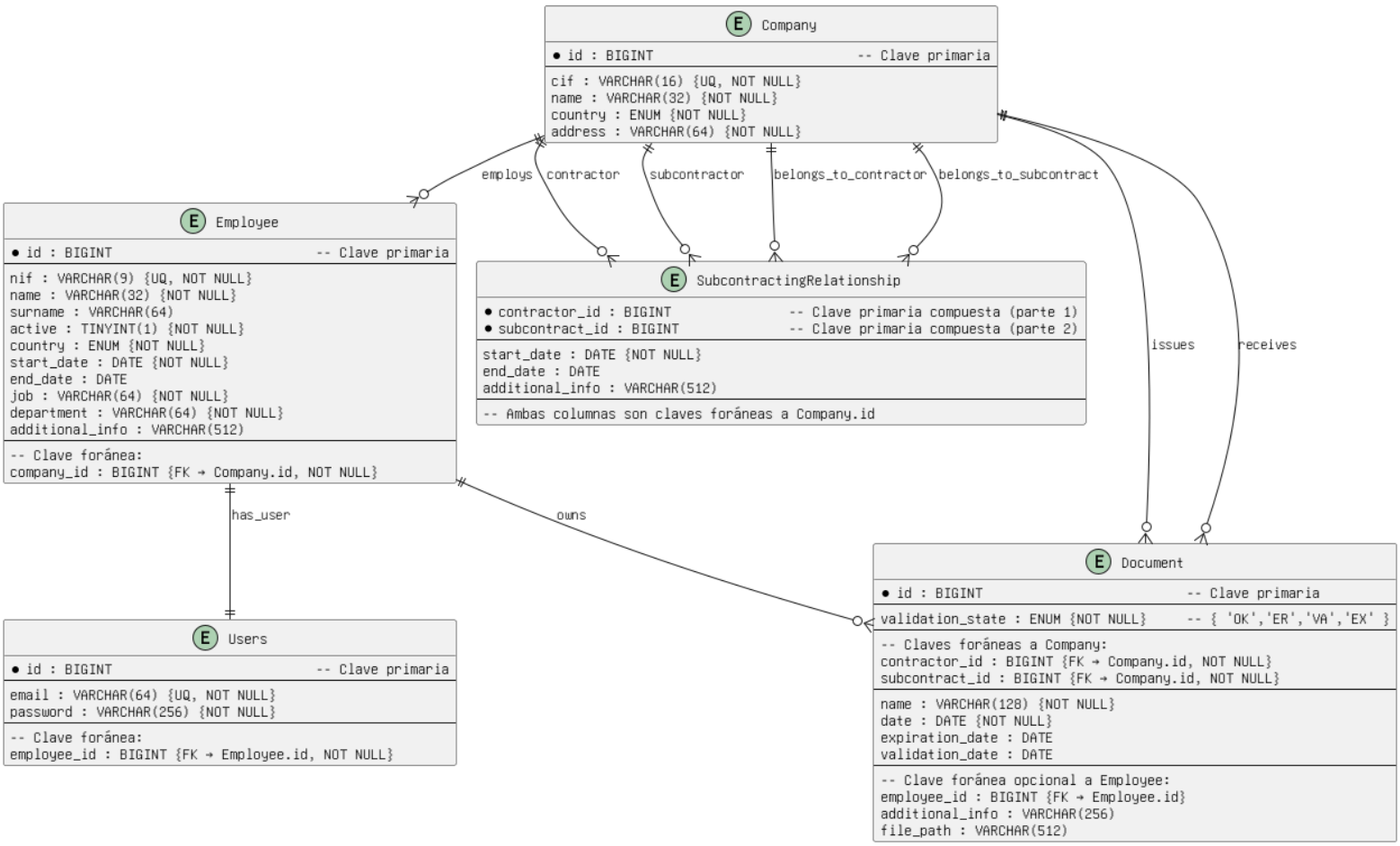
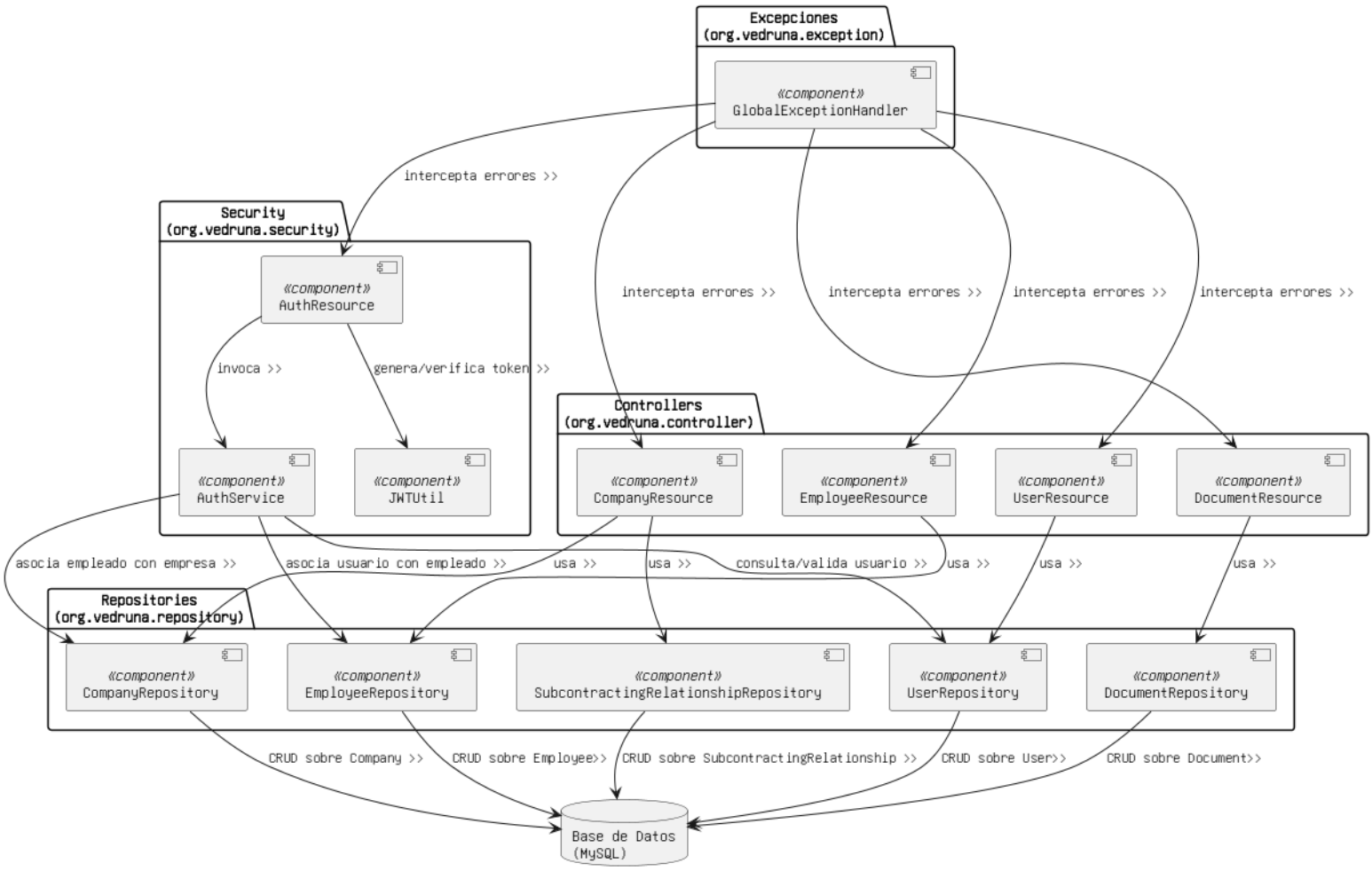


Diagrama de componentes

En este tipo de esquema UML, se detalla la estructura de nuestro *backend*, especificando los paquetes en los que se organiza el proyecto y la funcionalidad que cubren los distintos elementos al relacionarse unos con otros. Es útil para, a la hora de analizar el código, saber identificar a simple vista dónde encontrar determinada implementación, sin ir buscando a ciegas.



Casos de prueba

A continuación se detallan algunos casos de prueba para los puntos de validación más importantes en el flujo de nuestro modelo de persistencia, como son la unicidad de identificadores, fechas coherentes, existencia de entidades cuya clave es foránea, etc.

TC-REGISTER-01: Registro exitoso de nuevo empleado

- **Precondiciones:**
    - No existe en BD ningún usuario con email = "nuevo@ejemplo.com".
    - No existe en BD ningún empleado con nif = "11111111A".
    - start_date = LocalDate.now().minusDays(1) (o cualquier fecha de inicio anterior a hoy).
    - Existe una company con id = 5.
- **Pasos de Ejecución:**
    1. Construir un objeto RegisterDTO.
    2. Realizar una petición POST a /auth/register con el registerDTO como cuerpo.
- **Datos de Entrada (JSON):**

```
{
  "email": "nuevo@ejemplo.com",
  "nif": "11111111A",
  "name": "Ana",
  "surname": "López",
  "active": true,
  "country": "ES",
  "start_date": "2023-08-15",
  "job": "Developer",
  "department": "IT",
  "password": "Secreto123",
  "company_id": 5
}
```

- **Resultado Esperado:**
    - No lanza excepción.

- Se persiste un nuevo Employee en la BD con nif = "11111111A".
- Se persiste un nuevo User asociado a ese empleado, con email = "nuevo@ejemplo.com" y contraseña hasheada (BCrypt).
- El método retorna el objeto Employee recién guardado (con su id generado).

- **Estado:** PENDIENTE
- **Notas:**
  - Verificar en la base de datos que el nuevo empleado tenga company_id = 5.
  - Comprobar que en la tabla user, el campo password no almacena el texto en claro, sino el hash (debe empezar con $2a$...).

TC-REGISTER-02: Intento de registro con email duplicado

- **Precondiciones:**
  - Existe en BD un User con email = "dup@ejemplo.com".
  - No existe en BD ningún empleado con nif = "22222222B".
  - Existe una company con id = 5.
- **Pasos de Ejecución:**
  1. Construir un objeto RegisterDTO.
  2. Realizar una petición POST a /auth/register con el registerDTO como cuerpo.
- **Datos de Entrada (JSON):**

```
{
  "email": "dup@ejemplo.com",
  "nif": "22222222B",
  "name": "Luis",
  "surname": "Gómez",
  "active": true,
  "country": "ES",
  "start_date": "2023-08-15",
  "job": "Analista",
  "department": "IT",
  "password": "OtraClave123",
  "company_id": 5
```

}

- **Resultado Esperado:**
    - Se lanza IllegalArgumentException con el mensaje exacto:

“Este email ya está registrado”

    - No se persiste ningún Employee ni User en la BD.
- **Estado:** PENDIENTE
- **Notas:**
    - Comprobar que el texto de la excepción coincide literalmente con el definido en el código.
    - Este caso de prueba es idéntico para un NIF duplicado, sólo que el mensaje será "Este NIF ya está registrado".

TC-REGISTER-03: Intento de registro con fecha de contratación futura

- **Precondiciones:**
    - No existe en BD ningún User con email = "futuro@ejemplo.com".
    - No existe en BD ningún Employee con nif = "44444444D".
    - registerDTO.start_date = (fecha futura).
    - Existe una company con id = 5.
- **Pasos de Ejecución:**
    1. Construir un objeto RegisterDTO.
    2. Realizar una petición POST a /auth/register con el registerDTO como cuerpo.
- **Datos de Entrada (JSON):**

{

"email": "futuro@ejemplo.com",

"nif": "44444444D",

"name": "Elena",

"surname": "Ruiz",

"active": true,

"country": "ES",

"start_date": "2029-01-01",

"job": "Manager",

"department": "RRHH",

"password": "Password123",

"company_id": 5

}

- **Resultado Esperado:**
    - Se lanza IllegalArgumentException con el mensaje exacto:

"La fecha de contratación no puede ser futura"

    - No se persiste ningún Employee ni User en la BD.
- **Estado:** PENDIENTE
- **Notas:**
    - Asegurarse de que la comparación isAfter(LocalDate.now()) se realice con la zona horaria correcta del sistema.

TC-REGISTER-05: Intento de registro con company_id inválido

- **Precondiciones:**
    - No existe en BD ningún User con email = "noempresa@ejemplo.com".
    - No existe en BD ningún Employee con nif = "55555555E".
    - No existe ninguna company con id = 9999 (ID inválido).
- **Pasos de Ejecución:**
    1. Construir un objeto RegisterDTO.
    2. Realizar una petición POST a /auth/register con el registerDTO como cuerpo.
- **Datos de Entrada (JSON):**

{

"email": "noempresa@ejemplo.com",

"nif": "55555555E",

"name": "Lucas",

"surname": "Méndez",

"active": true,

"country": "ES",

"start_date": "2023-08-15",

"job": "Técnico",

"department": "Mantenimiento",

"password": "ClaveValida789",

"company_id": 9999

}

- **Resultado Esperado:**
    - Se lanza NotFoundException con el mensaje exacto:

"Empresa no encontrada."

    - No se persiste ningún Employee ni User en la BD.
- **Estado:** PENDIENTE
- **Notas:**
    - Verificar que la llamada a companyRepository.findByIdOptional(...) sea la que arroja la excepción con ese mensaje.

TC-COMP-GET-01: Obtener empresa existente por ID

- **Precondiciones:**
    - En la base de datos existe un registro Company con id = 10.
- **Pasos de Ejecución:**
    1. Petición GET a company/10.
- **Resultado Esperado:**
    - Devuelve código 200 con un Company no vacío.
    - El objeto Company retornado tiene id = 10 y coincide con los demás campos guardados en BD (por ejemplo, cif, name, country, address).
- **Notas:**
    - Verificar que ningún campo sale como null.

TC-COMP-UPDATE-03: Intento de actualizar empresa inexistente

- **Precondiciones:**
    - No existe en BD ningún Company con id = 7777.

- **Pasos de Ejecución:**
    1. Construir un NewCompanyDTO cualquiera (cualquiera de sus campos puede tener un valor válido).
    2. Hacer petición PUT a /company/7777 con cualquierDto en el cuerpo).
- **Resultado Esperado:**
    - Se lanza NotFoundException con el mensaje exacto:

"No hay empresas con el id 7777"

    - No se realiza ningún cambio en la base de datos.
- **Notas:**
    - Comprueba que concatena correctamente el id en el mensaje ("No hay empresas con el id " + id).

TC-DOC-CREATE-01: Creación válida de documento sin fechas conflictivas

- **Precondiciones:**
    - Existe en BD una Company con id = 10 (contratista).
    - Existe en BD una Company con id = 20 (subcontratista).
    - Existe en BD un Employee con id = 5 (opcional para asociación).
    - El enum ValidationState.valueOf("VA") es válido.
    - dto.getDate() = fecha no futura.
    - dto.getExpirationDate() = null.
    - dto.getValidationDate() = null.
- **Pasos de Ejecución:**
    1. Construir un objeto NewDocumentDTO con:
        - validationState = "VA"
        - contractorId = 10
        - subcontractId = 20
        - name = "Contrato.pdf"
        - date = LocalDate.now().minusDays(1)
        - expirationDate = LocalDate.now().plusDays(10)
        - validationDate = LocalDate.now()
        - employeeId = 5
        - additionalInfo = "Documento de prueba"
    2. Realizar una petición POST a /document con el DTO en el cuerpo.

- **Resultado Esperado:**
    - No se lanza excepción.
    - Se persiste un nuevo Document con los valores proporcionados y file_path = null.
    - El objeto retornado tiene un id distinto de null y referencias a las entidades Company y Employee correctas.
- **Notas:**
    - Verificar en BD que la fila existe con contractor_id = 10, subcontract_id = 20 y employee_id = 5.
    - Confirmar que validation_state = ValidationState.VA en el registro.

TC-DOC-ADDFILE-04: Agregar fichero a documento con archivo previo

- **Precondiciones:**
    - Existe en BD un Document con id = 17 y file_path = "uploads/antiguo-17-12345-antiguo.pdf".
    - El archivo "uploads/antiguo-17-12345-antiguo.pdf" existe físicamente en el sistema de archivos.
    - Se dispone de un InputPart válido con filename="nuevo.pdf".
- **Pasos de Ejecución:**

1. Realizar petición POST a document/17 pasando parteNueva como cuerpo.

- **Resultado Esperado:**
    - No se lanza excepción.
    - Se elimine el archivo "uploads/antiguo-17-12345-antiguo.pdf" sin errores.
    - Se cree un nuevo archivo, por ejemplo "uploads/doc-17-<timestamp>-nuevo.pdf".
    - El campo file_path en BD se actualice con la ruta del nuevo archivo.
    - El método retorna el Document actualizado.
- **Notas:**
    - Verificar que Files.deleteIfExists(oldFile) borra correctamente el archivo previo.

TC-SR-FIND-CONTR-01: Buscar relaciones con contratista existente con relaciones asociadas

- **Precondiciones:**
    - Existe en BD una Company con id = 10.
    - Existen en BD al menos 2 SubcontractingRelationship cuyos contractor.id = 10.
- **Pasos de Ejecución:**
    1. Realizar petición GET a /company/10/hires.
- **Resultado Esperado:**
    - Retorna una List<SubcontractingRelationship> cuyo tamaño corresponde al número de relaciones asociadas (≥ 2).
    - Cada elemento de la lista tiene contractor.id = 10.
- **Notas:**
    - Confirmar que la consulta find("contractor.id", contractorId).list() solo devuelve relaciones válidas.

TC-SR-CREATE-02: Intento de crear relación ya existente

- **Precondiciones:**
    - Existe en BD una SubcontractingRelationship con contractor.id = 100 y subcontract.id = 110.
    - srDTO.getStartDate() y srDTO.getEndDate() válidas.
- **Pasos de Ejecución:**
    1. Realizar petición POST a /company/100/hires/110 con el DTO en el cuerpo.
- **Resultado Esperado:**
    - Se lanza IllegalArgumentException con mensaje:

“La relación de contratación ya existe entra las empresas”

    - No se persiste una nueva relación idéntica.
- **Notas:**
    - Verificar que el flujo interno findByIds(...) encuentra la relación y no entra en las validaciones de fechas o IDs iguales.

Conclusión

Esta documentación integra de manera coherente todos los aspectos clave de *MetaConti*: a través del diagrama de clases se ha mostrado la estructura de entidades y sus relaciones, el diagrama E-R ha detallado el modelo relacional sobre la base de datos, y el diagrama de componentes ha ilustrado la organización modular del *backend* y sus dependencias. Además, los casos de prueba propuestos garantizan la verificación de la lógica de negocio, la persistencia y el manejo de errores en cada repositorio y controlador.

En conjunto, estos elementos proporcionan una visión global y detallada del diseño, la implementación y la validación del sistema, facilitando tanto la comprensión para nuevos desarrolladores como la mantenibilidad y la escalabilidad futura de la aplicación.