Luis Imaicela
PROYECTO FINAL DAM
Fase Diseño
09/12/2025

Santiago Roman Viguera
TUTOR DEL PROYECTO

LUIS IMAICELA
Diseño y Comunicación

INDICE

LUIS IMAICELA
Diseño y Comunicación

Proyecto final DAM

FASE DE DISEÑO

Siguiendo el cronograma establecido en la fase de análisis del proyecto, en este documento me centraré en la FASE DE DISEÑO de la aplicación **DesignWorks**. Es importante recalcar que en este punto mi propuesta comienza a tomar forma. Tras superar el "terror vacui" al enfrentarme a la hoja en blanco y después de varias revisiones para jerarquizar la información de la manera más comprensible posible. He organizado el diseño en los siguientes ítems a detallar:

- Arquitectura del sistema
- Diseño de BD (Modelo E-R + relacional)
- Diagramas UML
- ENDPOINTS REST (Defincón Swagger)
- DISEÑO UI/UX (FIGMA + WIREFRAMES)
- MATRIZ DE PERMISOS + SEGURIDAD JWT

Debo recalcar que, al trabajar dentro del ámbito del diseño, es en esta fase donde voy a dar forma a la identidad gráfica de la aplicación, por el momento he priorizado aspectos más técnicos de la programación para para asegurar que todo funcione correctamente antes de avanzar hacia la parte visual.

1. ARQUITECTURA DEL SISTEMA

En mi caso el sistema de la app lo he establecido en tres capas diferentes, esto lo hago pensando en conseguir una arquitectura modular y escalable.

- **PRIMERA CAPA - Aplicación móvil (Frontend – Flutter)**

  Es la responsable e la experiencia de usuario y de todas las interacciones directas con el sistema. Gestiona autenticación mediante JWT, consulta y manipulación de trabajos, comentarios y estados.

  Funciones principales:
  - Autenticación (JWT).
  - Visualización de trabajos ("Mis trabajos" y "Todos los trabajos").
  - Detalle del trabajo.
  - Cambio de estado.
  - Gestión de comentarios.
  - Consulta del historial.

- **SEGUNDA CAPA - API REST implementada en Spring Boot**

  Actúa como núcleo del sistema. Procesa la autenticación, aplica reglas de negocio, controla los roles y ejecuta toda la lógica relacionada con trabajos, requisitos y participantes.

  Se encarga de:
  - Autenticación y generación de tokens.
  - Gestión de trabajos, requisitos y participantes.
  - Persistencia de datos.
  - Validación del flujo de estados.
  - Control de roles (Administrador / Diseñador).

- **TERCERA CAPA - Base de datos MariaDB**

  Es donde planteo almacenar de forma estructurada toda la información del sistema, garantizando integridad, relación entre entidades y trazabilidad mediante el historial de estados.

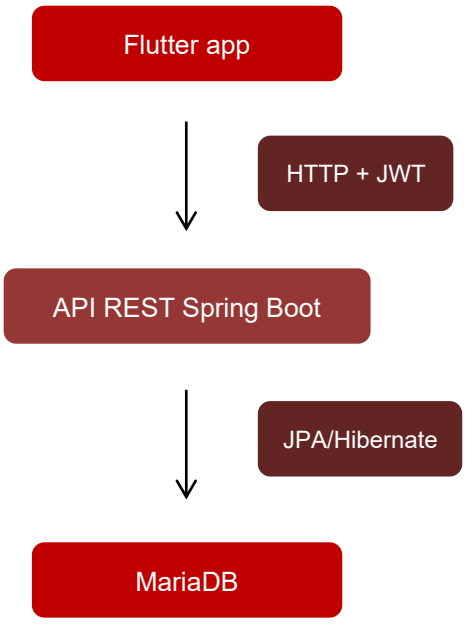
  Contiene información estructurada de:
  - Usuarios
  - Trabajos
  - Participantes
  - Comentarios
  - Requisitos
  - Historial de estados

1.1 DIAGRAMA LÓGICO DE ARQUITECTURA

He intentado que el flujo del sistema siga una comunicación clara y lineal, así evito problemas de funcionamiento y que entorpezcan el desarrollo del proyecto.

La app **Flutter envía solicitudes HTTP acompañadas de tokens JWT hacia la API REST**, que valida los permisos y procesa la operación. **Para persistir o recuperar información, el backend utiliza JPA/Hibernate como capa de acceso a datos, comunicándose finalmente con MariaDB**.

El siguiente diagrama refleja lo descrito:



1.2 ARQUITECTURA INTERNA DEL BACKEND

El backend se organiza en una estructura por capas que favorece la mantenibilidad y el orden del proyecto. Incluye:

- **config:** parámetros globales del sistema.
- **security:** manejo de JWT, filtros y reglas de autenticación.
- **controllers:** puntos de entrada REST para la aplicación.
- **services:** lógica de negocio centralizada.
- **repositories:** acceso a la base de datos mediante JPA.
- **entities:** modelos persistentes del dominio.
- **dto:** objetos optimizados para transporte de datos.
- **exceptions:** manejo estructurado de errores.)

designworks

```
── config
── security (JWT, authentication filters)
── controllers (REST Controllers)
── services (Business logic)
── repositories (JPA Repositories)
── entities (JPA entity models)
── dto (Data Transfer Objects)
── exceptions
```

2. DISEÑO DE BD (MODELO E-R + RELACIONAL)

A partir de los requisitos funcionales que establecí en la fase de análisis, se han identificado las entidades principales, sus relaciones y la posterior transformación al modelo relacional. Esto permite asegurar integridad referencial y una estructura preparada para crecer sin perder orden.

2.1 ENTIDADES PRINCIPALES

Las entidades definen los elementos clave de la app, entre ellos se encuentran:

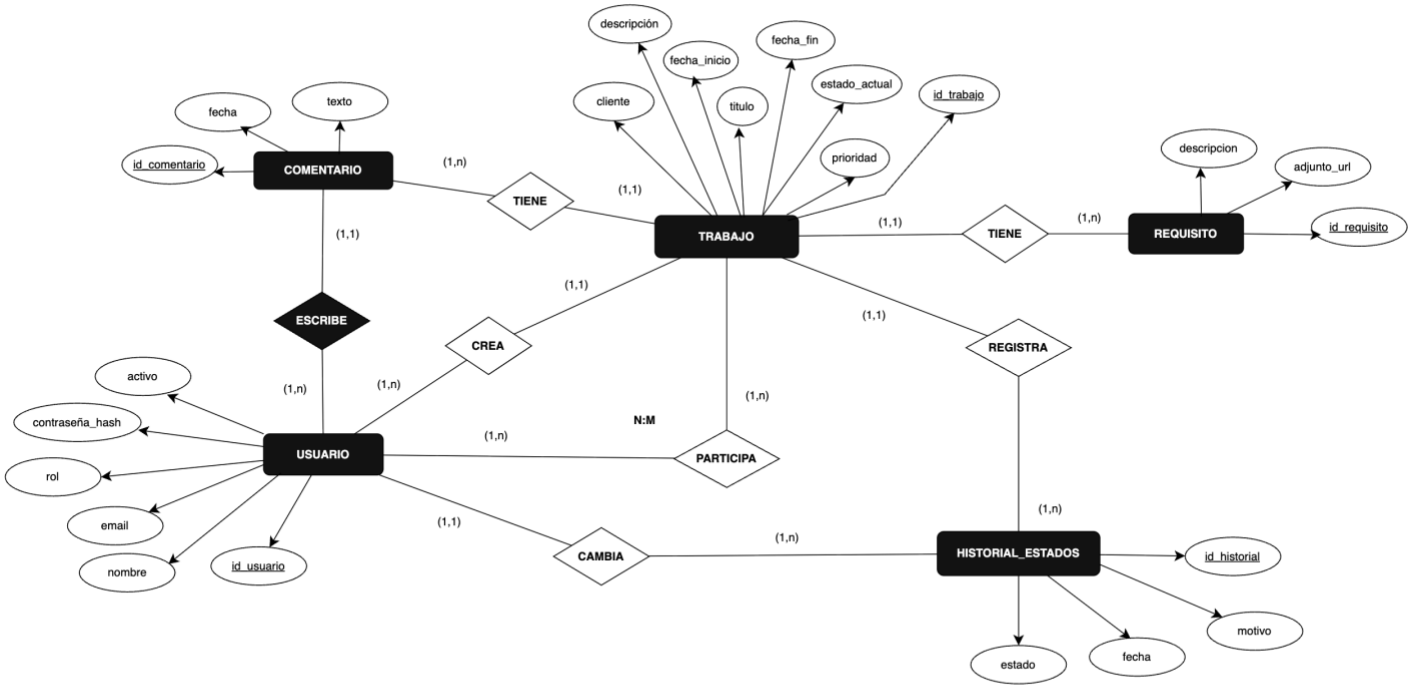
- **Usuario:** representa a los actores del sistema (administradores y diseñadores). Incluye información de identificación, credenciales (almacenadas como “contraseña_hash”) y el estado de actividad.
- **Trabajo:** núcleo del sistema, agrupa toda la información de un encargo de diseño (cliente, prioridad, fechas, estado actual, descripción y el usuario que lo creó).
- **Participantes:** modela la relación entre trabajos y usuarios, indicando qué usuarios participan en cada trabajo y su “rol_en_trabajo”.
- **Comentario:** permite registrar las aportaciones, revisiones y comunicación entre participantes sobre un trabajo.
- **Requisito:** recoge las necesidades específicas del trabajo, con posibilidad de asociar adjuntos (adjunto_url).
- **Historial de Estados:** almacena todos los cambios de estado de un trabajo, quién los realizó y el motivo, proporcionando una trazabilidad completa del ciclo de vida.

Estas entidades cubren tanto la gestión operativa (usuarios, trabajos, participantes) como el seguimiento detallado (comentarios, requisitos, historial).

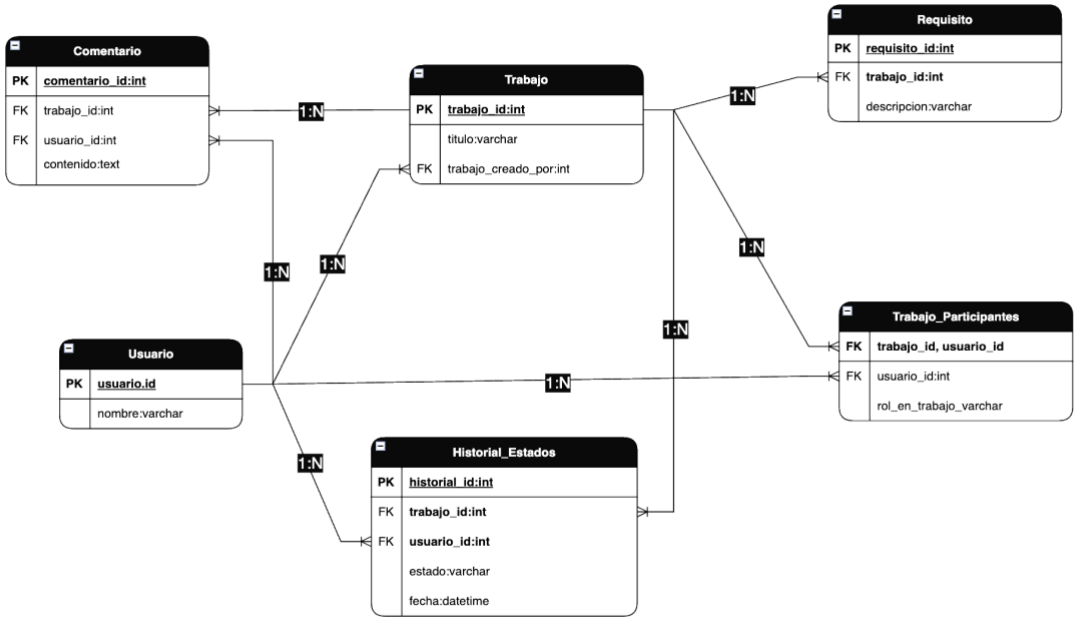
2.2 MODELO E-R

En el modelo Entidad–Relación se representan las asociaciones entre las entidades definidas, es aquí donde se reflejan todos los conocimientos adquiridos en la asignatura de Base de Datos:

- Un Usuario puede crear muchos “Trabajos” → “relación 1:N” entre “Usuario” y “Trabajo” (mediante el campo “trabajo_creado_por”).
- La participación en trabajos se representa como una “relación N:M” entre “Usuario y Trabajo”, resuelta mediante la entidad intermedia “Trabajo_Participantes” (trabajo_id, usuario_id, rol_en_trabajo).
- Cada Trabajo se relaciona con múltiples Requisitos, Comentarios y entradas en el Historial de Estados → “relaciones 1:N” desde Trabajo hacia estas entidades.
- Cada registro del Historial de Estados está asociado al Usuario que realiza el cambio, reforzando la responsabilidad y trazabilidad (FK usuario_id).



En el siguiente diagrama lógico se representan únicamente las claves primarias y foráneas; el detalle completo de los atributos que va a tener cada tabla lo hago en el apartado “Modelo Relacional (tablas)”.



2.3 MODELO RELACIONAL (TABLAS)

A partir del modelo E-R se obtiene el modelo relacional, que es lo que voy a implementar en MariaDB:

- **USUARIOS:** Contiene la información de los usuarios del sistema (id, nombre, email único, rol, contraseña_hash, activo).
- **TRABAJOS:** Registra cada trabajo con sus atributos clave (titulo, cliente, prioridad, fechas, estado_actual, descripcion) y una FK creado_por hacia USUARIOS.
- **TRABAJO_PARTICIPANTES:** Tabla de relación para la participación de usuarios en trabajos. Usa una PK compuesta (trabajo_id, usuario_id) y almacena el rol_en_trabajo.
- **COMENTARIOS:** Guarda los comentarios asociados a un trabajo y a un usuario (trabajo_id, usuario_id, fecha, texto).
- **REQUISITOS:** Define los requisitos de cada trabajo (descripcion, adjunto_url) vinculados mediante trabajo_id.
- **HISTORIAL_ESTADOS:** Registra cada cambio de estado (estado, fecha, motivo) asociado a un trabajo (trabajo_id) y al usuario que lo ejecuta (usuario_id).

Me he parecido interesante plantear los tipos de datos a elegir para cada uno de los datos de las tablas, así mi tutor puede indicar si lo ve viable u optaría por otra opción.

USUARIOS		
Campo	Tipo	
id	INT AUTO_INCREMENT PK	Identificador numérico como clave primaria
nombre	VARCHAR(100)	Suficiente para nombres completos.
email	VARCHAR(150) UNIQUE	Email con índice, único
rol	VARCHAR(20)	VARCHAR por si en algún momento incluyo más roles
contraseña_hash	VARCHAR(255)	Necesario para hashes Bcrypt/Argon2
activo	VARCHAR(255)	Representa boolean (0/1)

TRABAJOS		
Campo	Tipo	
id	INT AUTO_INCREMENT PK	Identificador estándar
titulo	VARCHAR(100)	Suficiente para nombres completos
cliente	VARCHAR(150)	Nombre del cliente
prioridad	ENUM('baja','media','alta','urgente')	Valores fijos
fecha_inicio	DATETIME	Para registrar momento exacto
fecha_fin	DATETIME NULL	Puede no existir fecha límite
estado_actual	ENUM('pendiente','en_progreso','revision', 'completado','cancelado')	Normalmente estado fijo
descripcion	TEXT	Textos largos
creado_por	INT FK	Relación con usuario

TRABAJOS_PARTICIPANTES		
Campo	Tipo	
trabajo_id	INT FK	Enlace al trabajo
usuario_id	INT FK	Enlace al usuario
rol_en_trabajo	VARCHAR(50)	Flexible para roles
PK compuesta	(trabajo_id, usuario_id)	Evita duplicidades

COMENTARIOS		
Campo	Tipo	
id	INT AUTO_INCREMENT PK	Identificador único
usuario_id	INT FK	Relación
rol_en_trabajo	INT FK	Relación
fecha	DATETIME	Momento de creación
texto	TEXT	Contenido del comentario

REQUISITOS		
Campo	Tipo	
id	INT AUTO_INCREMENT PK	Identificador único
trabajo_id	INT FK	Relación
descripcion	TEXT	Puede ser larga
adjunto_url	VARCHAR(255)	URLs estándar

HISTORIAL_ESTADOS		
Campo	Tipo	
id	INT AUTO_INCREMENT PK	Identificador único
trabajo_id	INT FK	Relación
estado	ENUM(...)	Debe coincidir con “estado_actual”
fecha	DATETIME	Fecha del cambio
usuario_id	INT FK	Usuario que realizó el cambio
motivo	TEXT	Explicación del cambio

3. DIAGRAMAS UML

Luego de definir la estructura de la BD, he intentado plantear la estructura principal del sistema de **DesignWorks** en términos de clases de dominio, servicio y controladores, así como las relaciones entre estos componentes. Ha sido una tarea que me ha costado bastante tiempo debido a que acotar todo para que funcione correctamente y tenga sentido es un poco tedioso y en principio difícil de entender.

3.1 DIAGRAMA DE CLASES

- **Clases de dominio**

  Las clases de dominio representan los elementos principales del modelo de negocio. Contienen los atributos y comportamientos propios de las entidades que forman parte del sistema, proporcionando una estructura lógica para los datos que se gestionan.

  - **Usuario**: Representa a los usuarios del sistema (administradores y diseñadores). Contiene los datos básicos de identificación y autenticación (nombre, email, rol, contraseña cifrada, estado activo).
  - **Trabajo:** Es la entidad central del sistema. Modela los encargos de diseño con información como título, cliente, prioridad, fechas, estado actual, descripción y el usuario que crea el trabajo (administrador de ese trabajo).
  - **Requisito:** Describe las necesidades específicas asociadas a un trabajo (por ejemplo, entregables, formatos o indicaciones del cliente), pudiendo incluir ficheros adjuntos mediante una URL.
  - **Comentario:** Registra las revisiones y la comunicación entre los participantes de un trabajo. Cada comentario está vinculado a un trabajo y a un usuario, e incluye la fecha y el texto del mensaje.
  - **Participante**: Modela la relación entre usuarios y trabajos. Permite indicar qué usuarios participan en cada trabajo y con qué rol (por ejemplo, diseñador asignado), resolviendo así la relación N..M entre Usuario y Trabajo.
  - **HistorialEstado:** Guarda todos los cambios de estado que sufre un trabajo a lo largo de su ciclo de vida. Cada registro indica el nuevo estado, la fecha del cambio, el usuario que lo realiza y el motivo.

- **Clases de servicio**

  Estas clases encapsulan la lógica de negocio y las operaciones que se realizan sobre las entidades del dominio. Actúan como intermediarias entre los controladores y la capa de acceso a datos, garantizando reglas, validaciones y procesos internos que deben cumplirse antes de modificar o consultar la información.

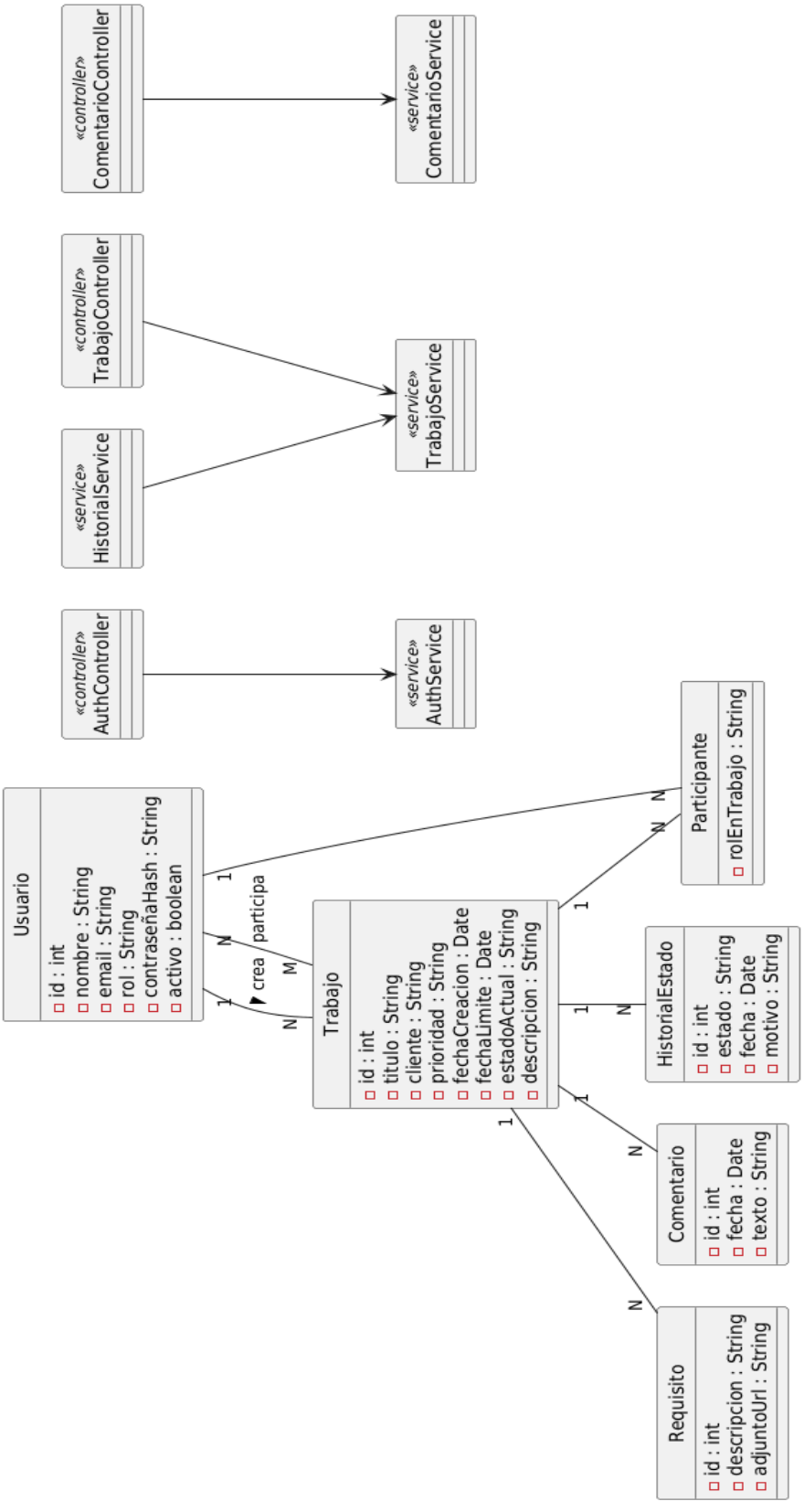
  - **AuthService**: Encapsula la lógica de autenticación y autorización: validación de credenciales, generación y verificación de tokens JWT y gestión de roles.
  - **TrabajoService:** Contiene la lógica de negocio relacionada con los trabajos: creación, edición, cancelación, cambio de estado, asignación de participantes y validación de reglas (por ejemplo, un único administrador por trabajo).
  - **ComentarioService:** Gestiona la creación y consulta de comentarios asociados a los trabajos, asegurando que el usuario tenga permisos sobre el trabajo.
  - **HistorialService:** Se encarga de registrar y consultar el historial de estados de un trabajo, garantizando la trazabilidad de todos los cambios realizados.

- **Clases de controladores**

  Gestionan la comunicación con el exterior. Reciben peticiones del cliente (API o interfaz gráfica), coordinan las acciones necesarias y delegan la lógica en los servicios correspondientes. Además, son las responsables de preparar y devolver las respuestas adecuadas al usuario o al sistema que realiza la solicitud.

  - **AuthController**: Expone los endpoints REST relacionados con el inicio de sesión y la obtención de tokens (por ejemplo, /auth/login).
  - **TrabajoController:** Ofrece los endpoints para gestionar trabajos y estados (crear, editar, listar, cambiar estado, asignar participantes, etc.).
  - **ComentarioController:** Proporciona los endpoints para añadir y consultar comentarios vinculados a un trabajo concreto.

- **Relaciones principales**
  - Un Trabajo se relaciona con Requisito, Comentario y HistorialEstado mediante asociaciones 1..N, ya que un trabajo puede tener múltiples requisitos, comentarios y registros de historial.
  - La relación entre Usuario y Trabajo es N..M a través de la clase Participante, que actúa como entidad intermedia y añade el rol del usuario en el trabajo.
  - Las clases de servicio se relacionan con las clases de dominio para aplicar la lógica de negocio, y las clases de controlador utilizan los servicios para atender las peticiones de la aplicación móvil.


«controller»
ComentarioController
«service»
ComentarioService
«controller»
TrabajoController
«service»
HistorialService
«service»
TrabajoService
«controller»
AuthController
«service»
AuthService
Usuario
id : int
nombre : String
email : String
rol : String
contraseñaHash : String
activo : boolean
Trabajo
id : int
titulo : String
cliente : String
prioridad : String
fechaCreacion : Date
fechaLimite : Date
estadoActual : String
descripcion : String
Participante
rolEnTrabajo : String
HistorialEstado
id : int
estado : String
fecha : Date
motivo : String
Comentario
id : int
fecha : Date
texto : String
Requisito
id : int
descripcion : String
adjuntoUrl : String
crea
participa
1
N
M

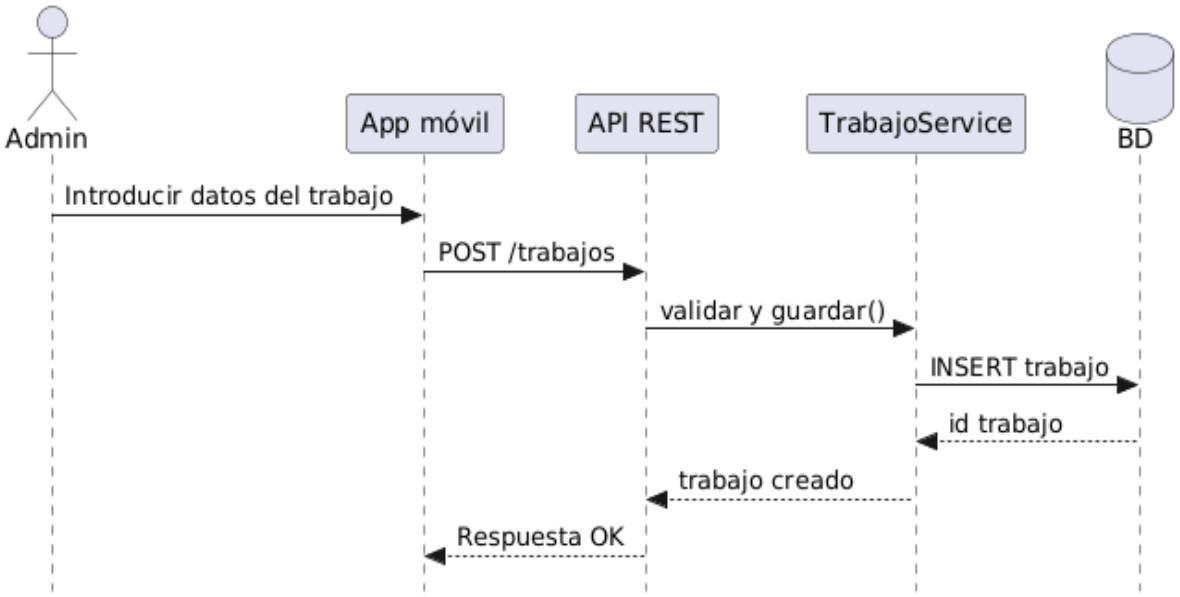
3.2 DIAGRAMA DE SECUENCIA

Los diagramas de secuencia muestran el flujo de mensajes entre la App, la API y las capas internas del backend para cada caso de uso relevante. A continuación, he reflejado los diferentes flujos de mensajes existentes en la app.

- **CU3 – Crear trabajo**

  En este caso de uso, un administrador crea un nuevo trabajo desde la aplicación:

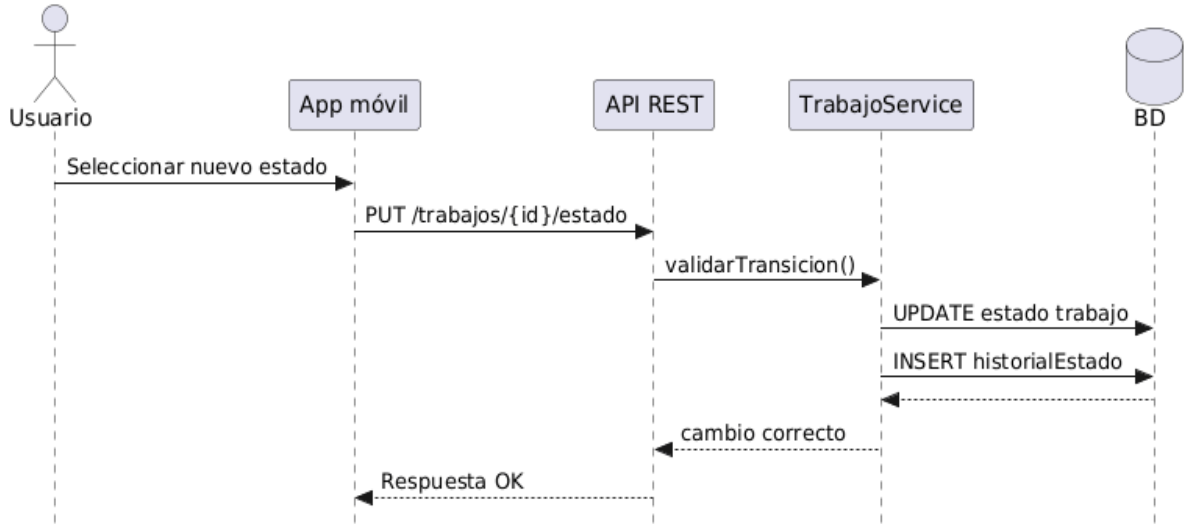
  - El **Administrador** introduce los datos del trabajo en la App móvil y envía el formulario.
  - La **App** realiza una petición POST /trabajos a la **API REST**, incluyendo el token JWT.
  - La API delega en "TrabajoService" la validación de los datos y de los permisos del usuario.
  - "TrabajoService" persiste el nuevo trabajo en la base de datos (mediante el repositorio/JPA).
  - La **BD** devuelve el identificador generado; el servicio responde a la **API**, y esta a su vez devuelve a la **App** la confirmación de creación del trabajo.



- **CU10 – Cambiar estado de trabajo**

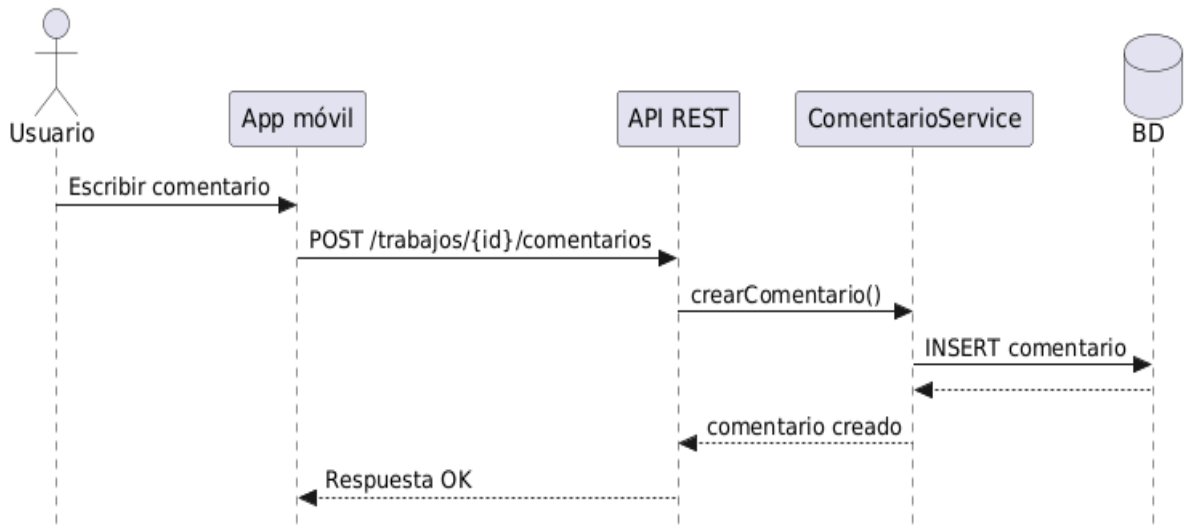
  Este caso de uso describe la transición de estado de un trabajo:

  1. El **Usuario** (normalmente el administrador del trabajo o un rol autorizado) selecciona un nuevo estado en la App.
  2. La **App** envía una petición PUT /trabajos/{id}/estado a la **API REST** con el nuevo estado.
  3. La **API** llama a "TrabajoService" para validar la transición de estado según las reglas definidas (por ejemplo, de En progreso solo puede pasar a "En revisión" o "Cancelado").
  4. Si la transición es válida, "TrabajoService" actualiza el estado del trabajo en la BD.
  5. A continuación, el servicio registra el cambio en la tabla de "HistorialEstado" (insertando un nuevo registro con estado, fecha, usuario y motivo).
  6. Finalmente, la **BD** confirma las operaciones, el servicio responde a la **API** y esta devuelve el resultado a la App.



- **CU11 – Añadir comentario**
  Este caso de uso refleja la creación de un comentario sobre un trabajo:

  1. El **Usuario** escribe un comentario desde la **App**.
  2. La **App** envía una petición “POST /trabajos/{id}/comentarios” a la **API REST** con el contenido del comentario.
  3. La **API** deriva la operación a “ComentarioService”, que valida que el usuario pueda comentar ese trabajo.
  4. “ComentarioService” inserta el comentario en la **BD**, asociado al trabajo y al usuario.
  5. La **BD** confirma la inserción y el servicio devuelve la información a la **API**, que responde a la App con el comentario creado.

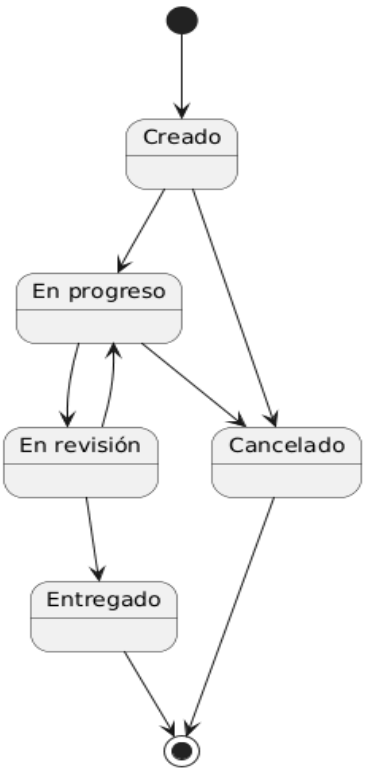


3.3 DIAGRAMA DE ESTADOS DEL TRABAJO

El diagrama de estados describe el ciclo de vida de un trabajo dentro del sistema **DesignWorks**.

- **Estados del trabajo**
  - **Creado:** estado inicial cuando se da de alta el trabajo, pero aún no se ha empezado a realizar.
  - **En progreso**: el trabajo está siendo ejecutado activamente por los diseñadores asignados.
  - **En revisión:** el trabajo está pendiente de revisión por parte del cliente o del administrador antes de considerarse finalizado.
  - **Entregado:** el trabajo ha sido finalizado y entregado; no se esperan más modificaciones (estado final "positivo").
  - **Cancelado:** el trabajo se ha detenido y no continuará (estado final "negativo").

- **Transiciones permitidas**
  - **Creado → En progreso / Cancelado**
    Desde un trabajo recién creado se puede comenzar la ejecución o cancelarlo directamente.
  - **En progreso → En revisión / Cancelado**
    Un trabajo en marcha puede pasar a revisión cuando se completa una versión, o puede cancelarse.
  - **En revisión → En progreso / Entregado**
    Tras revisar el trabajo, puede requerirse volver a "En progreso" para aplicar cambios, o bien marcarlo como "Entregado" si está aprobado.



4. ENDPOINTS REST (DEFINICIÓN SWAGGER)

La API REST de **DesignWorks** define los endpoints necesarios para que la aplicación móvil pueda autenticarse, gestionar trabajos y registrar toda la actividad relacionada. Los endpoints siguen un diseño coherente, orientado a recursos, y se documentan mediante Swagger/OpenAPI (a definir) para facilitar su uso y mantenimiento.

A continuación, se describen los principales módulos de la API y la funcionalidad que ofrece cada uno.

4.1 AUTENTICACIÓN

La autenticación se realiza mediante JSON Web Tokens (JWT). El usuario inicia sesión con su email y contraseña, y el sistema genera un token firmado que se usa posteriormente en todas las peticiones.

- **POST /auth/login**
  Este endpoint permite que un usuario registrado acceda al sistema.
  - **Request**: contiene las credenciales del usuario (email y password).

    { "email": "luis@mail.com", "password": "Diseno2710" }
  - **Response:** devuelve un token JWT y el rol del usuario autenticado, permitiendo controlar permisos en el frontend y en el backend.

    { "token": "jwt_here", "rol": "ADMIN" }

El token devuelto debe enviarse en los encabezados “Authorization: Bearer <token>” en todas las peticiones posteriores.

4.2 TRABAJOS

El módulo de trabajos expone los endpoints necesarios para consultar, crear, editar y gestionar el ciclo de vida de los trabajos. Algunos endpoints están restringidos a administradores.

- **POST /trabajos**
  Permite a un administrador crear un nuevo trabajo. Se validan los datos y se asigna automáticamente como administrador del trabajo al usuario que lo crea.
- **GET /trabajos**
  Devuelve el listado completo de trabajos disponibles.
  Admite filtros opcionales por estado, prioridad, cliente o rango de fechas, facilitando consultas más específicas en el entorno de administración.
- **GET /trabajos/mis**
  Muestra los trabajos asignados al diseñador autenticado, es decir, aquellos en los que participa como miembro del equipo.
- **GET /trabajos/{id}**
  Proporciona el detalle completo de un trabajo concreto: información general, requisitos, comentarios, participantes y estado actual.
- **PUT /trabajos/{id}**
  Permite editar los datos de un trabajo. El acceso está restringido al administrador del trabajo o a perfiles autorizados.

- **PUT /trabajos/{id}/estado**
  Gestiona el cambio de estado de un trabajo, verificando que la transición sea válida según las reglas definidas.
  El body incluye el nuevo estado y el motivo del cambio, que se registra en el historial.

  { "nuevoEstado": "EN_REVISION", "motivo": "Diseño terminado" }

4.3 REQUISITOS

- **POST /trabajos/{id}/requisitos**
  Permite añadir un requisito asociado al trabajo indicado. Se registran la descripción y, opcionalmente, la URL del archivo adjunto.

4.4 COMENTARIOS

La funcionalidad de comentarios permite la comunicación entre los participantes de un trabajo.

- **POST /trabajos/{id}/comentarios**
  Crea un nuevo comentario asociado al trabajo y al usuario autenticado.
- **GET /trabajos/{id}/comentarios**
  Devuelve todos los comentarios del trabajo, ordenados generalmente por fecha de creación.

4.5 HISTORIAL

El historial permite visualizar todos los cambios de estado que ha tenido un trabajo, garantizando trazabilidad y transparencia.

- **GET /historial/{trabajoId}**
  Devuelve la secuencia completa de cambios de estado del trabajo indicado: estado previo, estado nuevo, fecha y responsable del cambio.

5. DISEÑO UI/UX

El diseño de la interfaz de usuario constituye un elemento esencial para garantizar una experiencia clara, fluida y accesible dentro de la aplicación móvil. En esta fase se definen los wireframes iniciales que servirán como base para el posterior diseño visual en Figma, además me hace especial ilusión al poder poner en práctica mis conocimientos como Diseñador Gráfico.

Actualmente, esta fase se encuentra en proceso, por lo que se han establecido las pantallas principales y su funcionalidad prevista, pendientes de completar con prototipos de alta fidelidad. Posteriormente enviaré una propuesta gráfica al tutor para comentar las posibles mejoras y obtener su retroalimentación.

5.1 WIREFRAMES A ENTREGAR

En los wireframes voy a representar la estructura básica de cada pantalla, la organización de los elementos y la navegación principal de la aplicación. Estas vistas guiarán el diseño final en Figma y asegurarán consistencia en la interacción.

- **Pantalla de Login**
  Permite al usuario autenticarse mediante email y contraseña. Incluye validación de campos y acceso directo al flujo principal de la aplicación.
  Contendrá: formulario, botón de acceso, mensajes de error.
- **Lista de trabajos**
  Muestra todos los trabajos visibles para el usuario administrador. Incluirá filtros por estado, prioridad, cliente y rango de fechas.
  Es la vista principal de gestión.
- **Mis trabajos**
  Lista los trabajos asignados al diseñador autenticado. Está enfocada en la ejecución diaria del trabajo, permitiendo un acceso rápido al detalle y cambios permitidos.
- **Detalle del trabajo**
  Pantalla central de la aplicación. Presenta toda la información relevante del trabajo: cliente, prioridad, estado, participantes, requisitos, comentarios y fechas. Desde aquí se accederá a acciones como cambiar estado o añadir comentarios.
- **Añadir comentario**
  Interfaz para redactar y enviar un comentario asociado al trabajo. Incluye campo de texto, fecha automática y validación básica.
- **Cambiar estado**
  Pantalla destinada a seleccionar un nuevo estado válido y justificar el cambio. Integra las reglas de transición definidas en el backend.
- **Perfil**
  Permite al usuario ver su información básica (nombre, email, rol) y cerrar sesión. Opcionalmente podrá incluir funcionalidades futuras como cambio de contraseña o personalización de la cuenta.

6. MATRIZ DE PERMISOS + SEGURIDAD JWT

La gestión de permisos y la implementación de seguridad mediante JWT garantizan que cada usuario acceda únicamente a las funcionalidades que le corresponden según su rol. El sistema **DesignWorks** define dos perfiles principales —Administrador y Diseñador—, y cada uno dispone de acciones específicas dentro del flujo de trabajo. Además, la autenticación y autorización se realizan mediante tokens JWT, asegurando un acceso seguro desde la aplicación móvil.

6.1 ROLES Y MATRIZ DE PERMISOS

El sistema diferencia dos roles operativos:

- **Administrador (ADMIN)**
  Responsable de la creación, edición y supervisión global de los trabajos. Puede gestionar participantes, definir requisitos y acceder a todos los trabajos del sistema.
- **Diseñador (DISEÑADOR)**
  Participa en los trabajos que le han sido asignados. Puede añadir comentarios, consultar detalles, cambiar estados permitidos y ver únicamente sus trabajos asignados.

La siguiente matriz resume de forma clara las acciones permitidas por cada rol:

Acción	ADMIN	DISEÑADOR
Crear trabajo	✓ Sí	✕ No
Editar trabajo	✓ Sí	✕ No
Cancelar trabajo	✓ Sí	✕ No
Ver todos los trabajos	✓ Sí	✕ No
Ver trabajos asignados	✓ Sí	✓ Sí
Añadir comentario	✓ Sí	✓ Sí
Cambiar estado	✓ Sí	✓ Sí (si participa)
Ver historial	✓ Sí	✓ Sí

Esta estructura permite un control estricto de las operaciones sensibles del sistema, evitando que los diseñadores puedan modificar o crear trabajos sin autorización, pero manteniendo su capacidad de colaborar dentro de los proyectos asignados.

6.2 SEGURIDAD JWT (FLUJO DE AUTENTICACIÓN Y AUTORIZACIÓN)

El sistema utiliza JSON Web Tokens (JWT) como mecanismo de autenticación. Este enfoque permite un acceso seguro desde la app móvil sin necesidad de mantener sesiones en el servidor y garantiza que cada petición incluya la información necesaria para validar permisos. El flujo de seguridad funciona de la siguiente manera:

- El **usuario envía su email y contraseña** desde la aplicación.
- La **API valida las credenciales** consultando la base de datos.
- Si son correctas, la API **genera un token JWT** que contiene:
    - identificador del usuario.
    - Email.
    - rol (ADMIN o DISEÑADOR).
    - fecha de expiración.
- La App guarda el token de forma segura en “flutter_secure_storage”.
- En cada petición posterior, la App envía el token en la cabecera:
    - Authorization: Bearer {token}.
- Un **filtro de Spring Security** intercepta la petición, valida la firma, el contenido y la expiración del token.
- Finalmente, el rol incluido en el **JWT determina el acceso** a cada endpoint, aplicando la matriz de permisos definida.

Este mecanismo asegura que solo usuarios autenticados y con permisos adecuados puedan ejecutar operaciones sensibles, manteniendo así la integridad del sistema y la protección de los datos.