

**Red Social: “VEE”**

**El objetivo principal es ofrecer una alternativa a las redes sociales convencionales, proporcionando una experiencia que complemente la vida de los usuarios en lugar de consumirla.**

**TRABAJO FINAL DE CICLO**

**CFGS DESARROLLO DE APLICACIONES WEB**

Autor/a: Ángel García-Page Rodríguez

Tutor/a: José Enrique Atiénzar Ibáñez

Junio de 2025

Contenido

[Capítulo 1: Introducción y Objetivos 4](#_Toc199537472)

[1.1 Justificación 4](#_Toc199537473)

[Capítulo 2: Especificación de Requisitos 4](#_Toc199537474)

[Requisitos Funcionales 4](#_Toc199537475)

[Requisitos No Funcionales 5](#_Toc199537476)

[Prototipos de la interfaz 5](#_Toc199537477)

[Capítulo 3: Planificación Temporal y Evaluación de Costes 6](#_Toc199537478)

[Diagrama de Gantt 7](#_Toc199537479)

[Evaluación de costes 7](#_Toc199537480)

[Capítulo 4: Tecnologías Utilizadas 7](#_Toc199537481)

[Backend – Spring Boot 7](#_Toc199537482)

[Base de Datos – Oracle 7](#_Toc199537483)

[Frontend – React + Tailwind CSS 7](#_Toc199537484)

[Diseño – Figma 8](#_Toc199537485)

[Herramientas de apoyo 8](#_Toc199537486)

[Capítulo 5: Desarrollo e Implementación 8](#_Toc199537487)

[5.1 Estructura General del Backend 8](#_Toc199537488)

[model 8](#_Toc199537489)

[repository 8](#_Toc199537490)

[dto 8](#_Toc199537491)

[service y service.impl 8](#_Toc199537492)

[controller 8](#_Toc199537493)

[VeeApplication.java 8](#_Toc199537494)

[5.2 Modelo de Datos (model/) 8](#_Toc199537495)

[Usuario.java 8](#_Toc199537496)

[Diario.java 8](#_Toc199537497)

[Comentario.java 8](#_Toc199537498)

[Imagen.java 9](#_Toc199537499)

[EsAmigo.java + EsAmigoId.java 9](#_Toc199537500)

[5.3 Persistencia con JPA (repository/) 9](#_Toc199537501)

[UsuarioRepository 9](#_Toc199537502)

[DiarioRepository 9](#_Toc199537503)

[ComentarioRepository 9](#_Toc199537504)

[ImagenRepository 9](#_Toc199537505)

[EsAmigoRepository 9](#_Toc199537506)

[5.4 Transferencia de Datos (dto/) 9](#_Toc199537507)

[UsuarioDto 9](#_Toc199537508)

[DiarioDto 9](#_Toc199537509)

[ComentarioDto 9](#_Toc199537510)

[ImagenDto 9](#_Toc199537511)

[5.5 Lógica de Negocio (service/ y service.impl/) 10](#_Toc199537512)

[UsuarioServiceImpl 10](#_Toc199537513)

[DiarioServiceImpl 10](#_Toc199537514)

[ComentarioServiceImpl 11](#_Toc199537515)

[ImagenServiceImpl 12](#_Toc199537516)

[5.6 Controlador REST (controller/) 12](#_Toc199537517)

[Endpoints definidos 12](#_Toc199537518)

[Notas técnicas 14](#_Toc199537519)

[5.7 Aplicación Principal (VeeApplication.java) 14](#_Toc199537520)

[5.8 Pruebas y Validaciones 14](#_Toc199537521)

[5.9 Razón por la que no se implementó la funcionalidad de solicitudes de amistad 16](#_Toc199537522)

[Capítulo 6: Conclusiones y Líneas Futuras 17](#_Toc199537523)

[Conclusiones 17](#_Toc199537524)

[Líneas Futuras 17](#_Toc199537525)

[Capítulo 7: Bibliografía 17](#_Toc199537526)

# Capítulo 1: Introducción y Objetivos

En el presente proyecto se desarrolla una aplicación web denominada VEE, una red social centrada en la publicación de diarios personales. El objetivo principal de esta plataforma es ofrecer a los usuarios un espacio digital donde puedan expresar su día a día a través de entradas escritas, acompañadas opcionalmente por imágenes, y recibir comentarios de otros usuarios que también pueden incluir contenido multimedia.

La idea surge de combinar conceptos tradicionales de blog con dinámicas sociales de plataformas actuales, fomentando una experiencia más íntima, reflexiva y centrada en la escritura y la memoria diaria.

## 1.1 Justificación

En un panorama dominado por redes sociales rápidas y centradas en la imagen o el vídeo, VEE busca recuperar el valor de la escritura personal como medio de expresión, combinándola con una experiencia social mínima pero significativa. La idea es crear una red en la que el texto sea protagonista, pero sin renunciar a la estética ni a las funciones básicas de interacción social.

Desde el punto de vista técnico, este proyecto permite aplicar de forma práctica los conocimientos adquiridos durante el ciclo formativo de Desarrollo de Aplicaciones Web, combinando backend, frontend y diseño de base de datos. Además, se ha buscado emplear buenas prácticas de diseño, como separación de capas, uso de DTOs, control de errores y una estructura preparada para escalar.

# Capítulo 2: Especificación de Requisitos

## Requisitos Funcionales

Registro de usuarios: permite crear cuentas nuevas con email, nombre y contraseña. Se valida que el email no esté repetido.

Inicio de sesión: validación mediante email y contraseña. En futuras versiones se añadirá autenticación JWT.

Publicación de diarios: cada usuario puede crear un diario al día, compuesto por un título, texto y una o varias imágenes.

Visualización de diarios: listado general o filtrado por usuario. Se prioriza mostrar los más recientes.

Comentarios: cualquier usuario puede comentar diarios ajenos, incluyendo texto y una imagen opcional.

Edición del diario propio: permite modificar contenido antes de que finalice el día.

## Requisitos No Funcionales

Interfaz responsive con diseño mobile-first.

Bajo consumo de recursos y tiempos de carga mínimos.

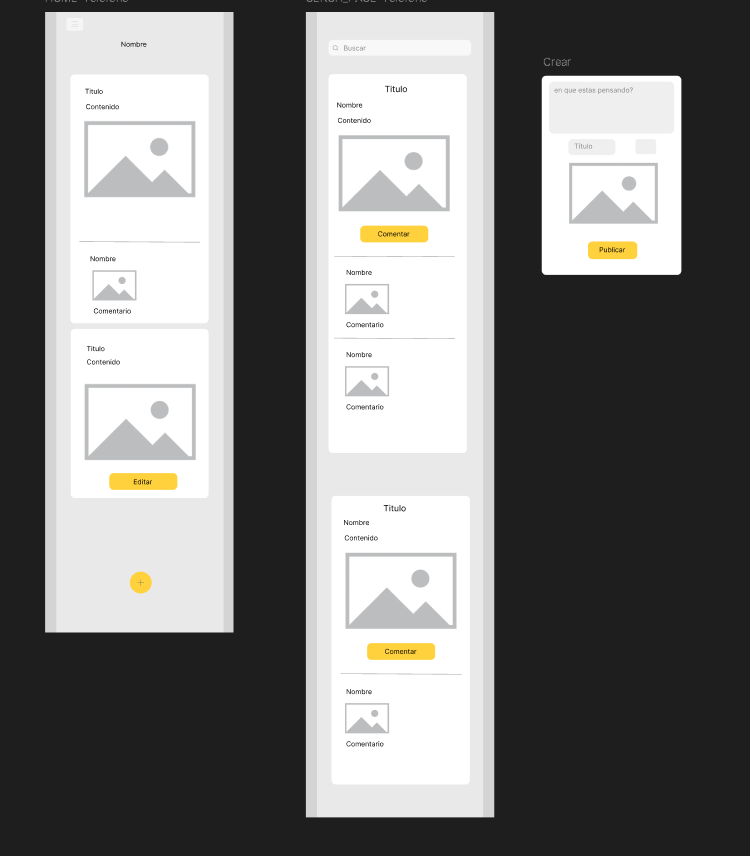
Implementación local sin despliegue obligatorio.

Seguridad básica en la validación de entrada y control de errores.

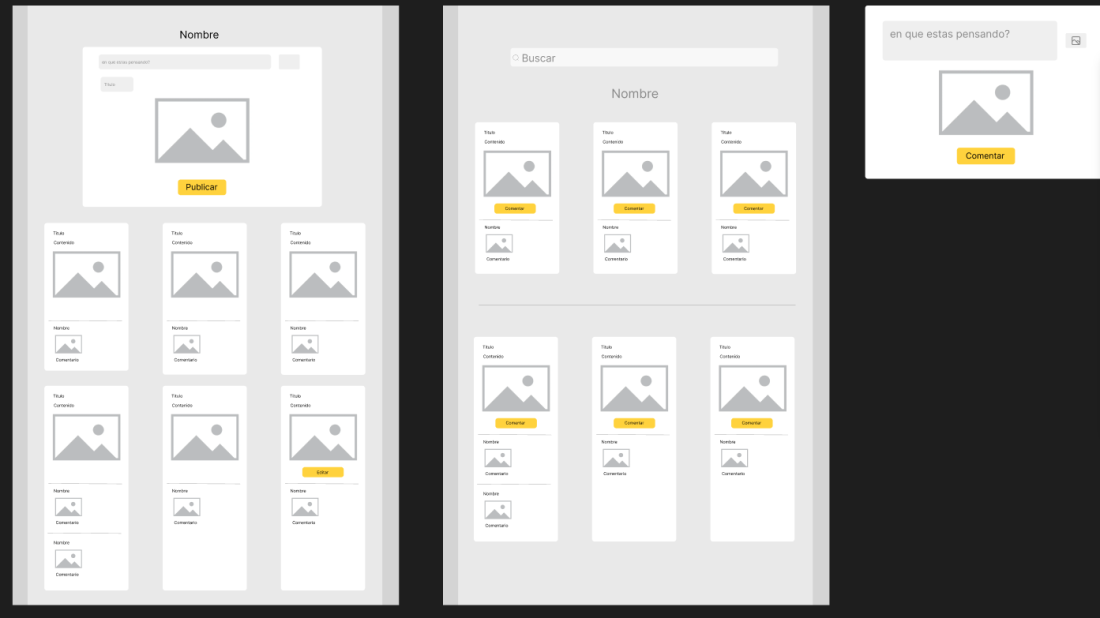
Separación estricta entre lógica de negocio, controladores y vistas.

## Prototipos de la interfaz

Prototipo versión móvil:



Prototipo versión escritorio:



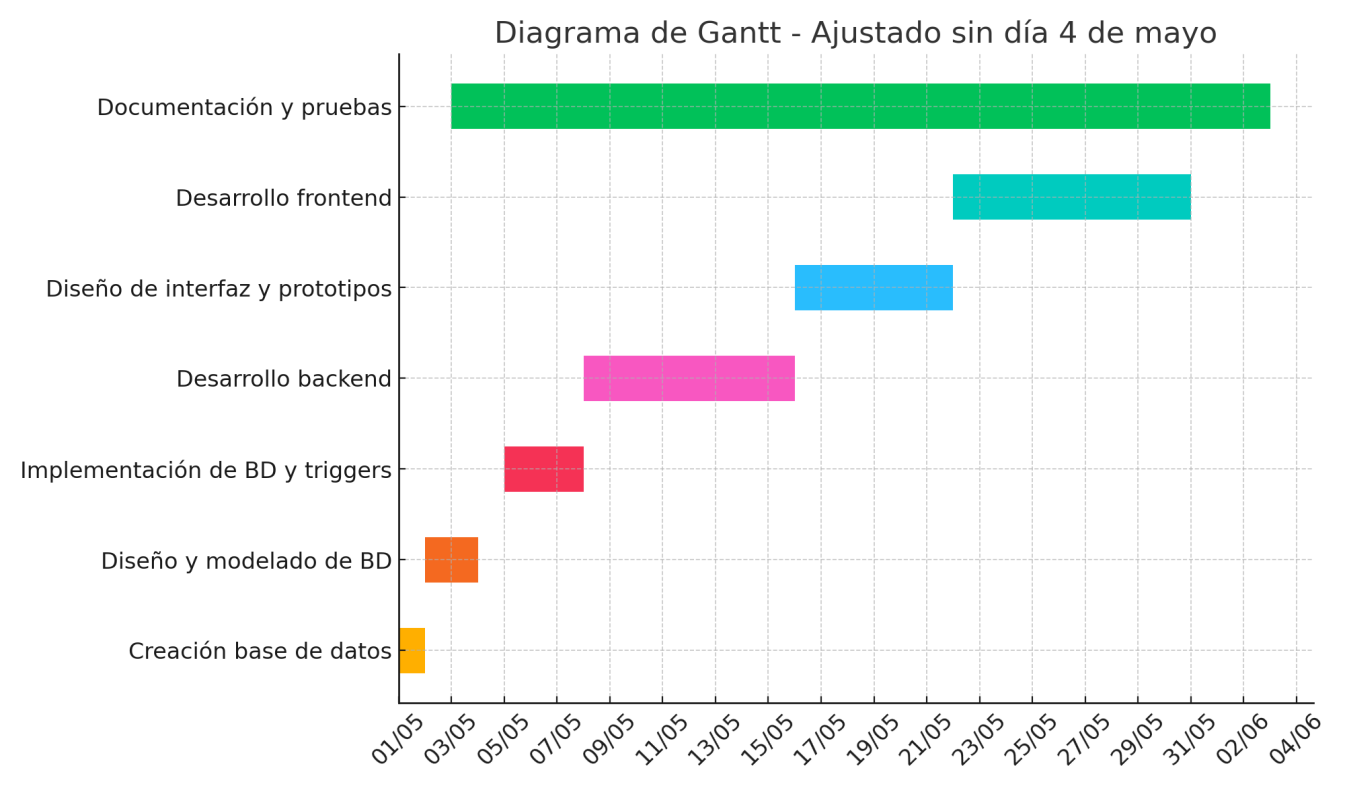
# Capítulo 3: Planificación Temporal y Evaluación de Costes

La planificación del proyecto se basó en bloques de trabajo semanales, priorizando la funcionalidad sobre la optimización. Se adoptó una metodología personal tipo cascada dividida en fases: análisis, diseño, desarrollo backend, frontend, pruebas y documentación.

**Resumen de fases y planificación temporal del proyecto:**

* **1 de mayo:** Creación de la base de datos.
* **2-3 de mayo:** Diseño del modelo entidad-relación y normalización.
* **5-7 de mayo:** Implementación de la base de datos en Oracle, incluyendo creación de tablas, índices y triggers.
* **8-15 de mayo:** Desarrollo del backend utilizando Spring Boot y conexión a la base de datos.
* **16-21 de mayo:** Diseño de la interfaz de usuario y creación de prototipos en Figma.
* **22-30 de mayo:** Desarrollo del frontend con React y Tailwind CSS.
* **3 de mayo - 2 de junio:** Documentación del proyecto y pruebas de la aplicación.

## Diagrama de Gantt



## Evaluación de costes

Costes estimados del proyecto:

|  |  |
| --- | --- |
| Concepto | Coste estimado |
| Hosting (uso local) | 0 € |
| Base de datos (Oracle gratuita) | 0 € |
| Frontend (React + Tailwind) | 0 € |
| Diseño (Figma gratuito) | 0 € |
| Suscripción ChatGPT Plus (1 mes) | 23 € |

# Capítulo 4: Tecnologías Utilizadas

## Backend – Spring Boot

Se utilizó Spring Boot por su arquitectura modular, facilidad de configuración y compatibilidad directa con JPA/Hibernate. Permite crear APIs RESTful de forma ágil y segura. Se aplicó inyección de dependencias, separación de capas y validaciones.

## Base de Datos – Oracle

Se eligió Oracle Database por su potencia, estabilidad y recursos avanzados como triggers y secuencias. El modelo entidad-relación se implementó mediante anotaciones JPA.

## Frontend – React + Tailwind CSS

React facilita la creación de componentes reutilizables, y su enfoque declarativo simplifica la gestión del estado. Tailwind CSS permite construir interfaces modernas sin necesidad de escribir CSS personalizado.

## Diseño – Figma

Figma se usó para crear prototipos responsivos. Se diseñaron las vistas en formato móvil y escritorio, basadas en prácticas modernas de UX.

## Herramientas de apoyo

**Postman**: para testear la API REST.

**ChatGPT**: para soporte técnico y revisión de redacción.

# Capítulo 5: Desarrollo e Implementación

## 5.1 Estructura General del Backend

El backend de la red social VEE se ha desarrollado utilizando Spring Boot, siguiendo el patrón arquitectónico MVC (Modelo - Vista - Controlador). Se ha dividido el código fuente en paquetes lógicamente organizados:

model: contiene las entidades JPA que representan las tablas de la base de datos.

repository: interfaces que permiten realizar operaciones CRUD mediante Spring Data JPA.

dto: objetos que encapsulan datos para la comunicación con el frontend.

service y service.impl: lógica de negocio y validaciones.

controller: controlador REST que expone los endpoints de la API.

VeeApplication.java: clase principal que lanza la aplicación.

## 5.2 Modelo de Datos (model/)

### Usuario.java

Define al usuario registrado. Campos clave: id, nombre, email, pass

Relaciones: un usuario puede tener múltiples diarios (OneToMany), comentarios e imágenes.

### Diario.java

Representa la publicación diaria de un usuario. Campos: título, texto, fecha.

Relación ManyToOne con Usuario.

Relación OneToMany con Comentario e Imagen.

### Comentario.java

Permite a los usuarios comentar diarios. Campos: texto, usuarioId, diarioId.

Puede tener una imagen (OneToOne opcional).

### Imagen.java

Modelo reutilizable para imágenes relacionadas con comentarios o diarios.

Campos: url, título.

Relaciones ManyToOne opcionales con Comentario, Diario y Usuario.

### EsAmigo.java + EsAmigoId.java

Modelo para representar relaciones de amistad entre usuarios con clave compuesta (EmbeddedId).

Campos: usuario1Id, usuario2Id, estado.

Relaciones bidireccionales mediante MapsId.

## 5.3 Persistencia con JPA (repository/)

Todos los repositorios extienden JpaRepository, lo que permite acceder a las entidades sin escribir SQL.

Repositorios definidos:

UsuarioRepository: búsqueda por email, nombre, ID.

DiarioRepository: diarios por usuario.

ComentarioRepository: comentarios por diario.

ImagenRepository: imágenes asociadas.

EsAmigoRepository: estado de amistad entre dos usuarios.

## 5.4 Transferencia de Datos (dto/)

Los DTOs se usan para encapsular la información expuesta por la API y evitar mostrar entidades directamente.

UsuarioDto: contiene id, nombre, email.

DiarioDto: id, título, texto, fecha, usuarioId, listas de imágenes y comentarios.

ComentarioDto: id, texto, usuarioId, diarioId, imagen asociada.

ImagenDto: id, url, título, relaciones a diario o comentario.

Esto facilita la validación, protege el modelo de datos y mejora la comunicación con React.

## 5.5 Lógica de Negocio (service/ y service.impl/)

Cada entidad cuenta con una interfaz Service y una clase ServiceImpl que encapsulan la lógica de negocio del sistema. Estas clases actúan como capa intermedia entre los controladores (API REST) y los repositorios (acceso a base de datos). Su función principal es aplicar reglas de validación, garantizar la integridad de los datos y preparar respuestas adecuadas (DTOs).

### UsuarioServiceImpl

#### List<UsuarioDto> findAll()

Devuelve una lista de todos los usuarios registrados en el sistema. Se obtienen las entidades desde la base de datos y se transforman en DTOs.

#### UsuarioDto findById(Long id)

Busca un usuario por su ID. Si existe, devuelve su representación como UsuarioDto. Si no, lanza una excepción o retorna null según la implementación.

#### List<UsuarioDto> findByUsername(String username)

Devuelve todos los usuarios cuyo nombre coincida (o contenga) el string proporcionado. Útil para búsquedas o filtrado.

#### void save(UsuarioDto usuarioDto)

Guarda un nuevo usuario a partir del DTO recibido. Convierte el DTO en entidad Usuario, lo valida y lo persiste usando usuarioRepository.

#### void save(UsuarioDto usuarioDto, Long id)

Versión sobrecargada del método anterior. Permite actualizar los datos de un usuario ya existente identificado por su id.

#### void delete(Long id)

Elimina el usuario cuyo ID coincide. Antes de borrar, se podría validar si existen relaciones (diarios, comentarios…).

### DiarioServiceImpl

#### 1.List<DiarioDto> findAll()

Devuelve todos los diarios de la plataforma. Transforma cada entidad Diario en un DiarioDto.

#### 2.void save(DiarioDto diarioDto)

Guarda un nuevo diario con los datos recibidos desde el frontend. Asocia el diario a un usuario si el ID es válido.

#### 3. void save(DiarioDto diarioDto, Long id)

Actualiza un diario ya existente. Usa el id para localizarlo y luego aplica los cambios desde el DTO.

#### 4.void delete(Long id)

Elimina un diario por su ID. Se encarga de borrar también sus comentarios e imágenes asociadas si procede.

## ComentarioServiceImpl

#### 1.List<ComentarioDto> findAll()

Recupera todos los comentarios del sistema. Convierte cada entidad Comentario en su correspondiente DTO.

#### 2.void save(ComentarioDto comentarioDto)

Crea un nuevo comentario a partir del DTO.

Valida que el usuario y el diario existen.

Si hay imagen adjunta, la guarda y la enlaza.

#### 3. void save(ComentarioDto comentarioDto, Long id)

Permite actualizar un comentario existente. Se localiza por id y se actualizan sus campos.

#### 4. void deleteById(Long id)

Elimina un comentario específico usando su identificador.

### ImagenServiceImpl

#### 1. List<ImagenDto> findAll()

Devuelve todas las imágenes de la base de datos, transformadas en ImagenDto. Incluye imágenes asociadas a diarios o comentarios.

#### 2. void save(ImagenDto imagenDto)

Guarda una nueva imagen en la base de datos.

Puede asociarse a un diario o un comentario, según los IDs que lleguen.

Se valida que las entidades asociadas existan.

#### 3. void delete(Long id)

Elimina una imagen por su ID.

# 5.6 Controlador REST (controller/)

El controlador principal de la aplicación es VeeController.java. Esta clase está anotada con @RestController y expone todos los endpoints REST utilizados por el frontend para interactuar con el backend. Implementa operaciones CRUD completas (crear, leer, actualizar y borrar) para las entidades Usuario, Diario, Comentario e Imagen.

### Endpoints definidos

#### Usuarios

* GET /usuarios  
  Devuelve la lista completa de usuarios registrados.  
  → List<UsuarioDto> usuarios()
* GET /usuarios/{id}  
  Devuelve los datos de un usuario específico según su ID.  
  → UsuarioDto usuario(Long id)
* GET /usuarios/{username}  
  Devuelve todos los usuarios cuyo nombre coincide con el username.  
  → List<UsuarioDto> usuarioByUsername(String username)
* POST /usuarios  
  Registra un nuevo usuario en el sistema.  
  → void saveUsuario(UsuarioDto usuarioDto)
* PUT /usuarios/{id}  
  Actualiza los datos de un usuario existente.  
  → void updateUsuario(Long id, UsuarioDto usuarioDto)
* DELETE /usuarios/{id}  
  Elimina un usuario por su ID.  
  → void deleteUsuario(Long id)

#### Diarios

* GET /diarios  
  Devuelve todos los diarios disponibles en la plataforma.  
  → List<DiarioDto> diarios()
* POST /diarios  
  Crea un nuevo diario.  
  → void saveDiario(DiarioDto diarioDto)
* PUT /diarios/{id}  
  Actualiza un diario existente.  
  → void updateDiario(Long id, DiarioDto diarioDto)
* DELETE /diarios/{id}  
  Elimina un diario por su identificador.  
  → void deleteDiario(Long id)

#### Comentarios

* GET /comentarios  
  Devuelve todos los comentarios registrados.  
  → List<ComentarioDto> comentarios()
* POST /comentarios  
  Crea un nuevo comentario vinculado a un diario.  
  → void saveComentario(ComentarioDto comentarioDto)
* PUT /comentarios/{id}  
  Modifica un comentario ya existente.  
  → void updateComentario(Long id, ComentarioDto comentarioDto)
* DELETE /comentarios/{id}  
  Elimina un comentario por su ID.  
  → void deleteComentario(Long id)

#### Imágenes

* GET /imagenes  
  Devuelve todas las imágenes almacenadas en el sistema.  
  → List<ImagenDto> imagenes()
* POST /imagenes  
  Sube una nueva imagen asociada a un diario o comentario.  
  → void saveImagen(ImagenDto imagenDto)
* DELETE /imagenes/{id}  
  Elimina una imagen por su identificador.  
  → void deleteImagen(Long id)

### Notas técnicas

* Todos los métodos usan @RequestBody para recibir DTOs del frontend y @PathVariable para parámetros de ruta.
* Las operaciones GET devuelven listas o elementos individuales, mientras que POST, PUT y DELETE son void.
* La lógica interna está delegada completamente a los servicios correspondientes, respetando el patrón de capas.

## 5.7 Aplicación Principal (VeeApplication.java)

Contiene el punto de entrada de la aplicación:

@SpringBootApplication

public class VeeApplication {

public static void main(String[] args) {

SpringApplication.run(VeeApplication.class, args);

}

}

Esto configura automáticamente los beans, la base de datos, controladores y servicios.

## 5.8 Pruebas y Validaciones

Durante el desarrollo se utilizaron herramientas como Postman para probar los endpoints manualmente. Se insertaron datos de prueba directamente en Oracle, y se validaron:

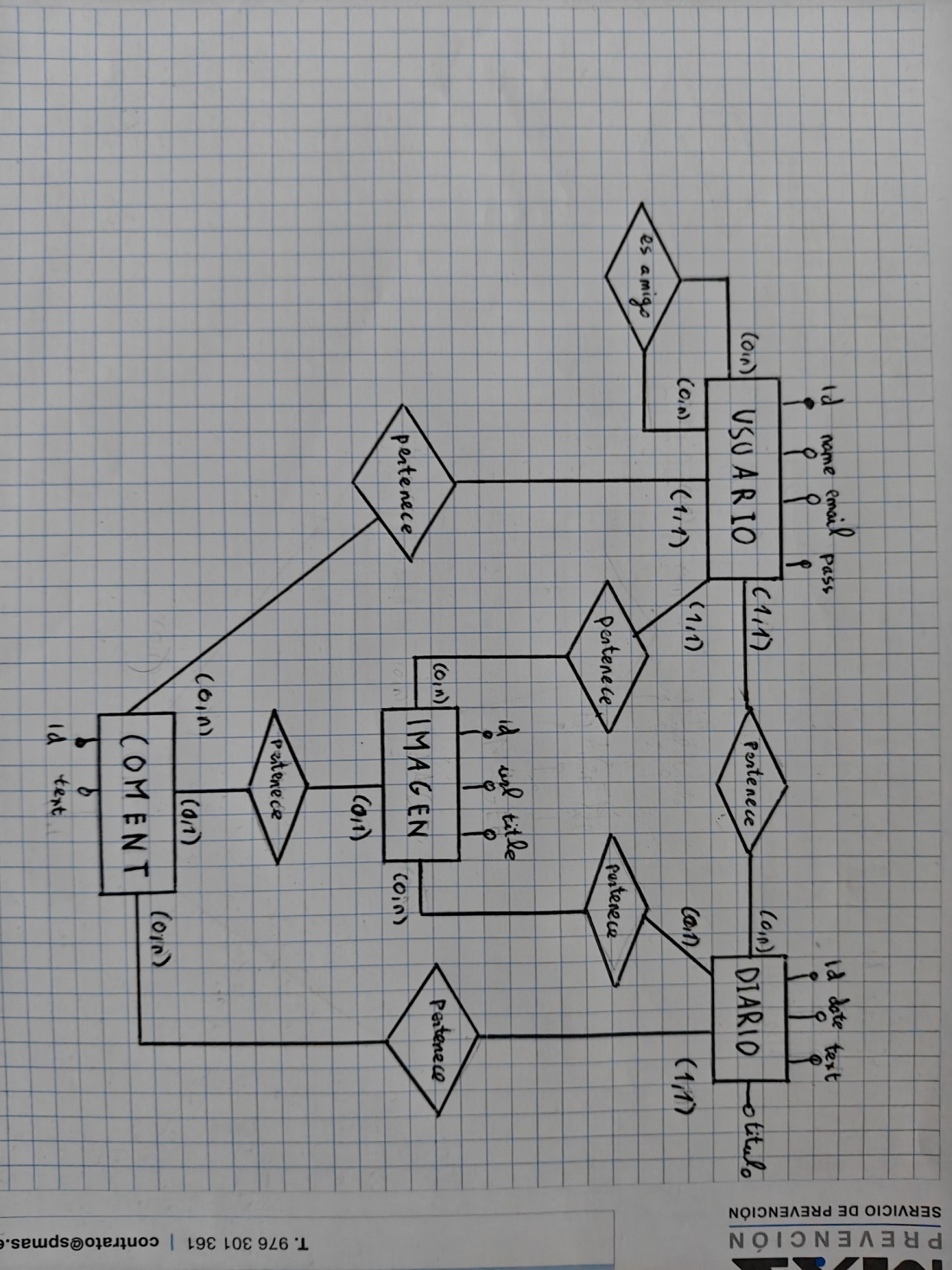
Inserciones de usuarios, diarios y comentarios.

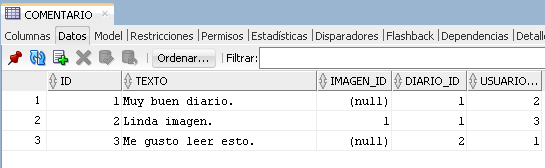
Correcta relación entre entidades.

Serialización y estructura de respuesta de los DTOs.

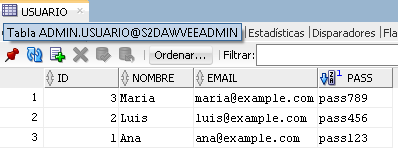
Validaciones de existencia antes de guardar o vincular datos.

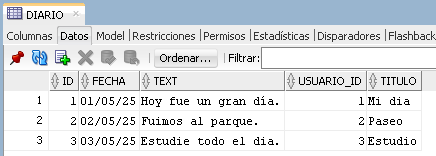
Las siguientes capturas muestran el modelo entidad-relación diseñado, incluyendo la definición de relaciones entre entidades como 'Usuario', 'Diario', 'Comentario' e 'Imagen'.

  
  
En el modelo se observa que cada 'Usuario' puede tener múltiples 'Diarios', y cada 'Diario' múltiples 'Comentarios' e 'Imágenes'. Los 'Comentarios' a su vez pueden incluir imágenes, y todos los elementos están correctamente enlazados mediante claves foráneas.  
  
Además, se incluyen capturas de las tablas generadas con datos de prueba. Esto evidencia que la estructura diseñada es funcional, y que el sistema de inserción mediante triggers funciona correctamente. Los IDs se autogeneran utilizando secuencias, lo cual mejora la integridad y eficiencia del sistema.









### 5.9 Razón por la que no se implementó la funcionalidad de solicitudes de amistad

Durante el desarrollo del backend, se diseñó una entidad llamada EsAmigo para modelar las relaciones de amistad entre usuarios. Esta entidad permite, mediante una clave compuesta (usuario1Id y usuario2Id), representar relaciones bidireccionales y almacenar un posible estado de la relación (pendiente, aceptada, rechazada, etc.).

Sin embargo, **la funcionalidad completa de solicitudes de amistad no se llegó a implementar** por las siguientes razones:

* **Limitación de tiempo del proyecto**: El desarrollo de un sistema de amistad con lógica de solicitudes, aceptaciones, notificaciones y comprobaciones bidireccionales requiere una lógica adicional compleja que no era prioritaria frente a otras partes más esenciales del sistema, como los diarios, los comentarios o la subida de imágenes.
* **Dificultad añadida en la validación de operaciones**: La implementación de esta función habría requerido mecanismos adicionales para validar acciones según el estado de amistad entre usuarios (por ejemplo, restringir ver diarios solo si hay amistad aceptada), lo cual habría incrementado significativamente la complejidad del backend y del frontend.

A pesar de no haberse incluido en esta versión, **la estructura de datos y la entidad EsAmigo ya están preparadas para una futura expansión del sistema**, por lo que la funcionalidad de amistad podría incorporarse fácilmente más adelante con control de estados, endpoints REST dedicados y validaciones de acceso basadas en la relación entre usuarios.

# Capítulo 6: Conclusiones y Líneas Futuras

### Conclusiones

La realización de este proyecto ha supuesto un reto importante tanto técnico como personal. Se han aplicado los conocimientos adquiridos durante el ciclo DAW y se han superado dificultades propias del desarrollo de una aplicación real.

Entre los principales logros destacan:

* La correcta modelización de entidades y relaciones.
* La implementación de una API REST organizada y funcional.
* La validación de datos mediante DTOs.
* La separación de capas que facilita mantenimiento futuro.

### Líneas Futuras

* Implementar autenticación JWT.
* Añadir solicitudes de amistad (pendientes, rechazadas, aceptadas).
* Despliegue en Vercel (frontend) y Oracle Cloud o Railway (backend).
* Sistema de notificaciones y reacciones a publicaciones.
* Mejora de experiencia de usuario con animaciones y retroalimentación visual.

# Capítulo 7: Bibliografía

* Fernando Herrera (Udemy). React: De cero a experto (Hooks y MERN)
* https://www.udemy.com/course/react-cero-experto/
* Documentación de Spring Boot: https://spring.io/projects/spring-boot
* Documentación de React: https://react.dev/
* Oracle Database Documentation: https://docs.oracle.com/en/database/
* Tailwind CSS Docs: https://tailwindcss.com/docs
* Figma Learn: https://help.figma.com
* Stack Overflow: https://stackoverflow.com/
* ChatGPT (OpenAI): asistencia técnica durante el desarrollo.
* Postman Docs: https://learning.postman.com/
* Materiales del profesor Julio Efrén: teoría de bases de datos y JPA.