



Trabajo fin de ciclo presentado por:	Eduardo Andrés Pop Rad
Tutor/a:	Isidoro Nevares Martín
Fecha:	29 de Mayo de 2025

Memoria del proyecto

Documento técnico

Índice

1. <u>Introducción</u>	1
1.1. <u>Descripción y contexto del proyecto</u>	1
1.2. <u>Motivación del proyecto</u>	1
1.3. <u>Beneficios esperados</u>	1
2. <u>Objetivos</u>	2
2.1. <u>Objetivos generales</u>	2
2.2. <u>Objetivos específicos</u>	2
3. <u>Contexto actual</u>	2
3.1. <u>Estado del arte</u>	2
3.2. <u>Conceptos clave</u>	3
4. <u>Planificación</u>	3
4.1. <u>Metodología de gestión del proyecto</u>	3
4.2. <u>Temporización y secuenciación</u>	3
5. <u>Análisis de requisitos</u>	4
5.1. <u>Diagrama de casos de uso</u>	4
5.2. <u>Requisitos funcionales</u>	4
5.3. <u>Requisitos no funcionales</u>	5
5.4. <u>Descripción de los usuarios y sus necesidades</u>	5
6. <u>Diseño de la aplicación</u>	6
6.1. <u>Mockups o wireframes</u>	6
6.2. <u>Arquitectura del sistema</u>	7
6.3. <u>Diagramas Entidad/Relación</u>	8
6.4. <u>Diagrama de Clases</u>	9
7. <u>Desarrollo de aplicación</u>	9
7.1. <u>Tecnologías y herramientas utilizadas</u>	9
7.2. <u>Descripción de las principales funcionalidades</u>	10
8. <u>Pruebas y validación</u>	11
8.1. <u>Pruebas unitarias</u>	11
8.2. <u>Pruebas de integración</u>	11
9. <u>Conclusiones y trabajo futuro</u>	11
10. <u>Relación del proyecto con los módulos del ciclo</u>	12
11. <u>Bibliografía/Webgrafía</u>	12
12. <u>Anexos</u>	13
<u>Anexo I: Pruebas Unitarias</u>	13
<u>Anexo II: Pruebas Funcionales</u>	14

1. Introducción

1.1. Descripción y contexto del proyecto

El proyecto consiste en el desarrollo de software de una aplicación de escritorio, enfocada en la administración y visualización de distintas obras de arte.

Durante el desarrollo de la aplicación, se ha tenido en cuenta distintos tipos de buenas prácticas de codificación, como la separación de responsabilidades, modularización del código y más, para una mejor escalabilidad y mantenimiento del código y aplicación.

El proyecto pone en práctica los conocimientos adquiridos a lo largo del curso, como la Programación Orientada a Objetos (POO), desarrollo de interfaces con JavaFX y persistencia de datos mediante un ORM como Hibernate que facilita la interacción entre las clases y la base de datos.

La finalidad principal de la aplicación es permitir a los usuarios consultar información sobre distintos tipos de obras de arte, como cuadros, grabados o esculturas.

1.2. Motivación del proyecto

Una de las principales razones por las que decidí hacer este proyecto, es porque no soy especialmente aficionado al arte, pero me rodeo de personas que sí lo son, y muestran mucho interés por ello. Gracias a esta diferencia, me llevó a pensar sobre lo poco que conocía sobre el tema, y al ver la necesidad, tanto personal como general de contar con una herramienta que sirva como punto de partida para quienes, como yo, quieren adentrarse en el mundo del arte sin tener en conocimientos previos.

La segunda motivación está relacionada con el interés por adquirir una mayor cultura general de este sector. Aunque al principio no era un área que me llamara mucho la atención, al investigar y trabajar en el desarrollo de la aplicación, descubrí que se trata de un mundo lleno de historia y un gran trasfondo. Esta curiosidad fue creciendo durante el proyecto y se convirtió en un incentivo para seguir aprendiendo.

Este proyecto también me ha ayudado a nivel académico, permitiéndome consolidar conocimientos técnicos importantes, aplicar habilidades adquiridas durante la formación y enfrentarme al desarrollo completo de una aplicación desde cero.

1.3. Beneficios esperados

Uno de los principales beneficios esperados del proyecto, es ofrecer a los usuarios una forma simple de acceder a información sobre el mundo del arte. La aplicación busca facilitar el aprendizaje sobre obras importantes sin tener que recurrir a varios sitios web, lo que también aporta una mayor comodidad y seguridad de datos personales.

Desde una perspectiva más personal, el proyecto me ha servido como una oportunidad para extender mis conocimientos. Esto no solo me permitirá mantener conversaciones fluidas con personas cercanas, sino que también me aporta una base para seguir explorando este campo.

A nivel académico y profesional, el desarrollo de la aplicación me ha permitido afianzar competencias en el desarrollo de software, como la modularización, programación orientada a objetos (POO) y el uso de herramientas como JavaFX e Hibernate. Estos conocimientos me serán de gran ayuda en futuros proyectos, tanto personales como profesionales.

2. Objetivos

Los objetivos del proyecto representan los logros que deseo alcanzar a través de la aplicación. Estos han sido definidos teniendo en cuenta la intención personal de crear una herramienta útil para personas con interés en iniciarse en el mundo del arte.

2.1. Objetivos generales

- Facilitar el aprendizaje sobre obras de arte a usuarios sin conocimientos previos del tema.
- Reunir en una única aplicación información relevante sobre distintas obras (cuadros, esculturas, grabados).
- Aplicar los conocimientos adquiridos en el ciclo formativo, especialmente Java, Hibernate y JavaFX.

2.2. Objetivos específicos

- Implementar clases y estructuras orientadas a objetos para representar las distintas obras de arte.
- Diseñar una arquitectura modular y ordenada, que respete la separación de responsabilidades.
- Desarrollar una interfaz gráfica con JavaFX que resulte clara y atractiva visualmente.
- Optimizar el rendimiento de la aplicación, evitando cargas innecesarias o duplicación de datos.
- Gestionar posibles errores o excepciones mediante un sistema de manejo de errores controlado.

3. Contexto Actual

3.1. Estado del arte

Tras realizar una búsqueda de diferentes fuentes, he observado que no existe ninguna aplicación que integre información general y accesible sobre distintas obras de arte, sin un nexo temático entre ellas. Lo habitual es encontrar sitios web de museos o autores concretos, como pueden ser las páginas de museos, que presentan colecciones limitadas a sus obras.

Estas plataformas, aunque muy completas en sus ámbitos, no ofrecen una experiencia centralizada ni están orientadas a un público general que solo quiera aprender del mundo del arte. Tampoco están pensadas como aplicaciones de escritorio que funcionen sin conexión, lo que puede ser una limitación en algunos entornos educativos o personales donde el acceso a Internet no está garantizado.

Este vacío en el mercado presenta una oportunidad para el desarrollo de una aplicación como punto de partida para quienes deseen comenzar a aprender sobre arte de forma sencilla y sin necesidad de navegar entre múltiples sitios, se evita la publicidad que suele encontrarse en algunas plataformas web.

3.2. Conceptos clave

Durante el desarrollo de este proyecto se han tratado diversos conceptos clave, los cuales se describen los más relevantes:

- **Aplicación de escritorio:** Software instalado y ejecutado en un ordenador, sin necesidad de conexión a Internet, diseñado para ser accesible y funcional en todo momento.
- **Obras de arte:** Creaciones artísticas que constituyen el contenido principal de la aplicación, organizado para su consulta y visualización.
- **JavaFX:** Biblioteca de Java utilizada para construir interfaces gráficas dinámicas, facilitando la interacción del usuario con la aplicación.
- **Hibernate:** Framework de mapeo objeto-relacional (ORM) que permite comunicar las clases del programa con la base de datos, simplificando las operaciones de persistencia de datos.
- **Base de datos relacional en la nube (AWS):** La aplicación se conecta a una base de datos alojada en un servidor de Amazon Web Services, lo que permite acceder a la información desde cualquier ubicación con conexión a Internet.

4. Planificación

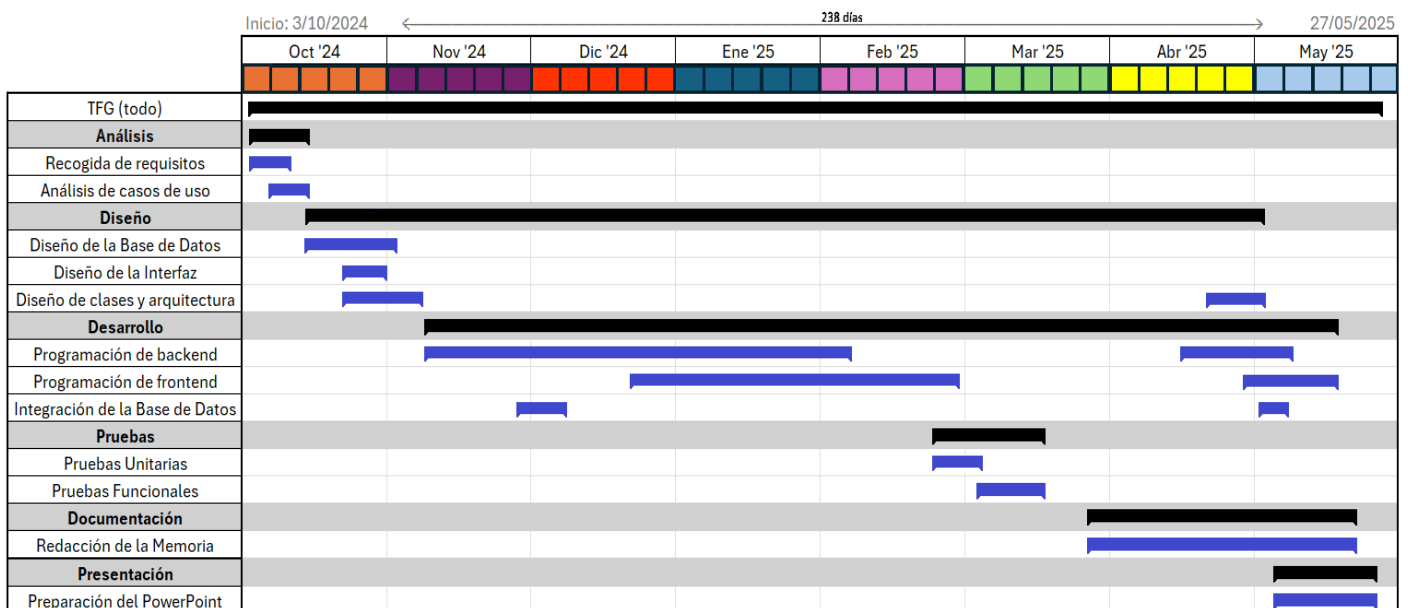
4.1. Metodología de gestión del proyecto

Al ser un proyecto desarrollado de forma individual, la gestión se ha centrado en la planificación personal de tareas.

Se ha seguido una metodología de tipo iterativa, dividiendo el proyecto en etapas claras: análisis, diseño desarrollo y documentación.

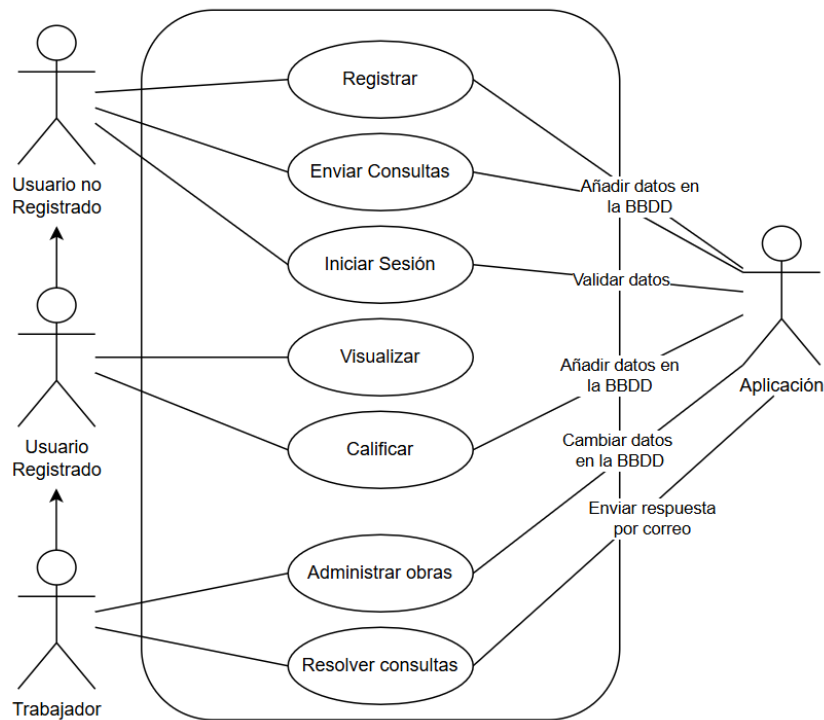
Para llevar un mejor control, se han utilizado herramientas como listas de tareas, lo que ha permitido marcar objetivos semanales y priorizar según la importancia de las tareas.

4.2. Temporización y secuenciación



5. Análisis de requisitos

5.1. Diagrama de casos de uso



5.2. Requisitos funcionales

La aplicación se estructura en varios módulos funcionales, que permiten al usuario interactuar con el sistema de forma sencilla.

Módulo de Autenticación

- Iniciar sesión con un usuario registrado.
- Cerrar sesión de forma segura.
- Registrar nuevos usuarios.

Módulo de Consulta de Obras de Arte

- Filtrar obras por tipo (cuadro, grabado, escultura).
- Buscar obras por título.
- Visualizar los detalles de una obra.
- Ver imágenes asociadas a cada obra de arte.
- Ver imágenes asociadas a cada obra de arte.

Módulo de Gestión de Datos

- Insertar nuevas obras en el sistema.
- Modificar los datos de una obra existente.
- Eliminar las obras del sistema.
- Importar datos desde la base de datos en la nube (AWS).

Módulo de Navegación

- Moverse entre las diferentes secciones de la aplicación mediante botones (Inicio, Lista de obras, Perfil, Consultas, etc...).
- Salir de la aplicación de forma controlada.

5.3. Requisitos no funcionales

A continuación, se describen los principales requisitos no funcionales del proyecto, agrupados por categorías.

Requisitos técnicos

- La aplicación de escritorio está desarrollada en Java utilizando JavaFX para la interfaz gráfica.
- El sistema utiliza Hibernate como ORM para la gestión de la persistencia de datos.
- La base de datos se aloja en un servidor remoto en la nube mediante AWS.

Rendimiento

- La aplicación debe cargar la lista de obras en 2 segundos.
- Las búsquedas y filtrados deben ejecutarse en un tiempo inferior a 2 segundos.
- La carga de imágenes debe realizarse de forma progresiva para evitar bloqueos.

Usabilidad

- La interfaz gráfica debe ser clara, intuitiva y sencilla de usar para cualquier usuario.
- Organización de menús y botones para facilitar la navegación por las distintas secciones.

Mantenibilidad y escalabilidad

- El código se ha desarrollado aplicando principios de modularidad, facilitando posibles ampliaciones y mantenimiento futuro.
- La estructura del proyecto permite agregar nuevas funcionalidades (como filtros más complejos, nuevos tipos de obras).

Disponibilidad

- Se espera que la base de datos remota esté disponible más del 99% del tiempo gracias a los servicios de AWS.

5.4. Descripción de los usuarios y sus necesidades

La aplicación está diseñada para ser utilizada por dos perfiles principales, cada uno con necesidades específicas.

1. Usuarios generales o clientes

Este grupo está compuesto por personas de cualquier edad y género, interesada en conocer y aprender sobre el mundo del arte.

Necesidades principales:

- Acceder a la información de forma sencilla, sin procesos complejos.
- Visualizar obras de arte organizadas por categorías.
- Obtener descripciones claras y breves sobre cada obra, su autor y contexto.
- Realizar búsquedas rápidas y filtradas para encontrar obras específicas.

2. Trabajadores

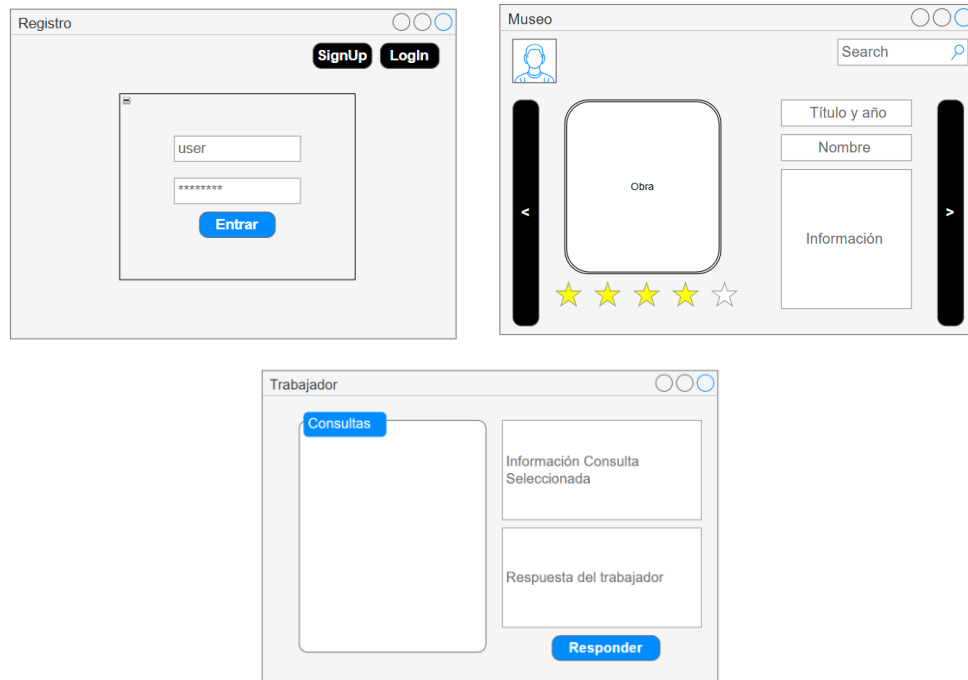
Este perfil está destinado a personas encargadas de la administración del contenido de la aplicación.

Necesidades principales:

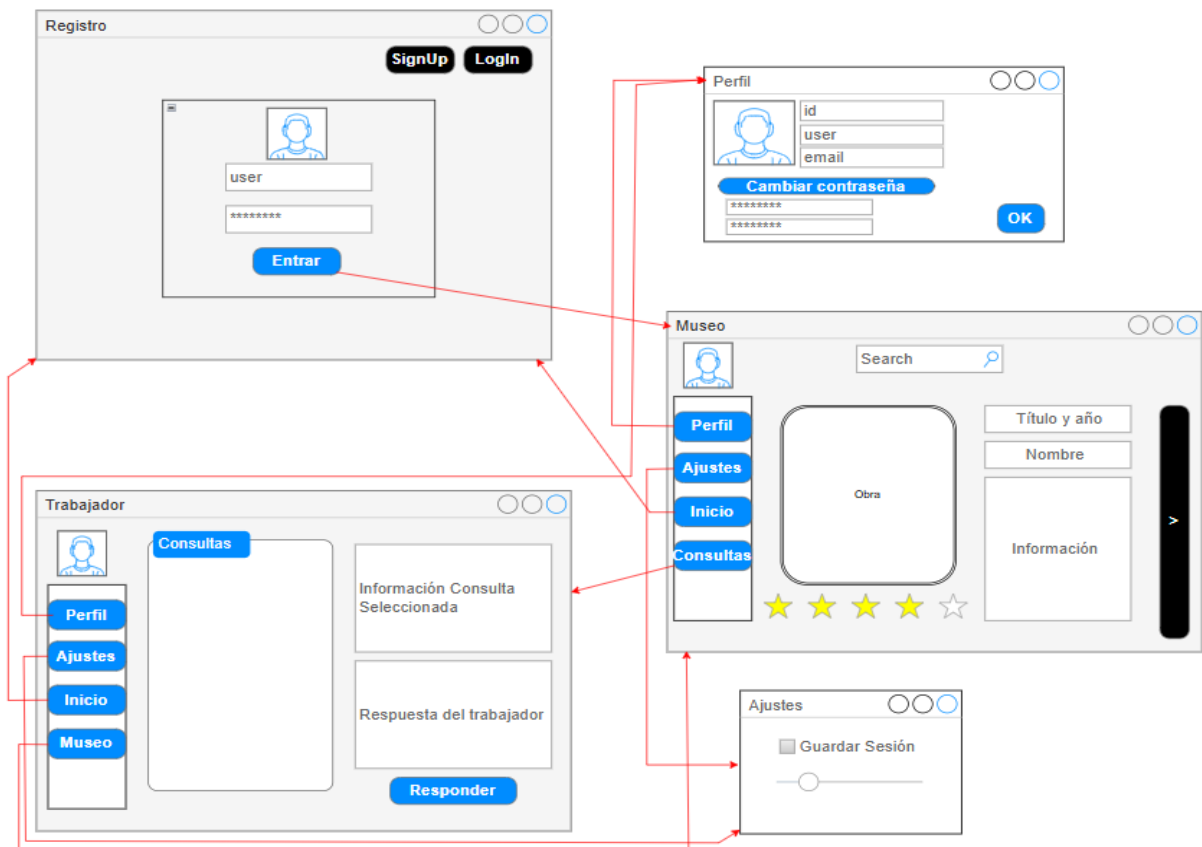
- Acceder al sistema con permisos especiales.
- Administrar las obras en la base de datos.
- Revisar consultas realizadas por los usuarios y responderlas por correo.

6. Diseño de aplicación

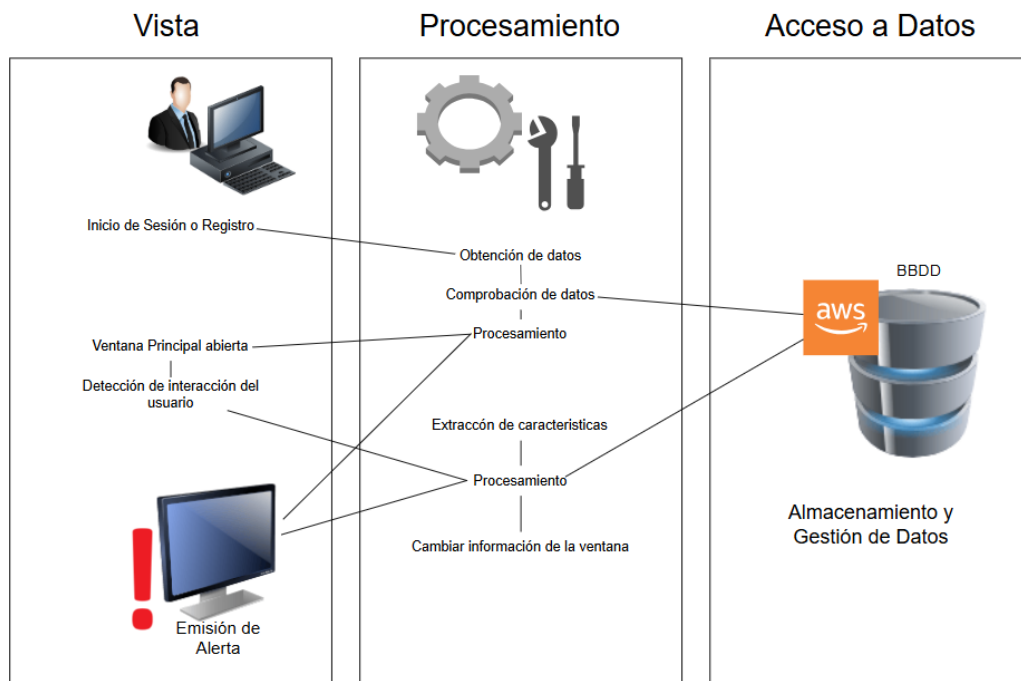
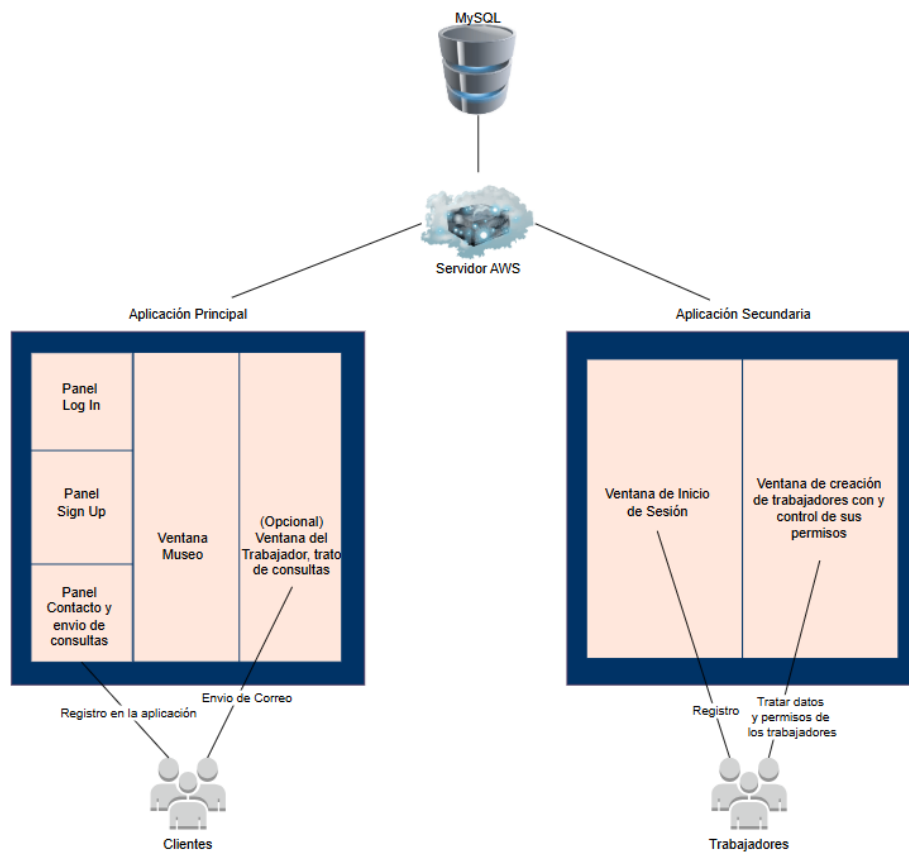
6.1. Mockups o wireframes



Plano de Navegación entre ventanas

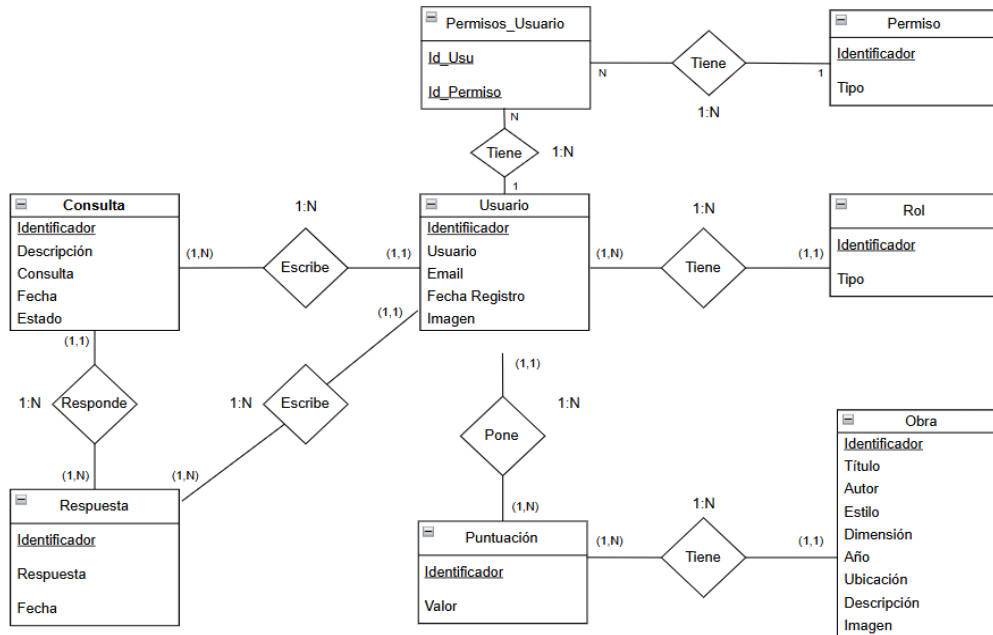


6.2. Arquitectura del sistema

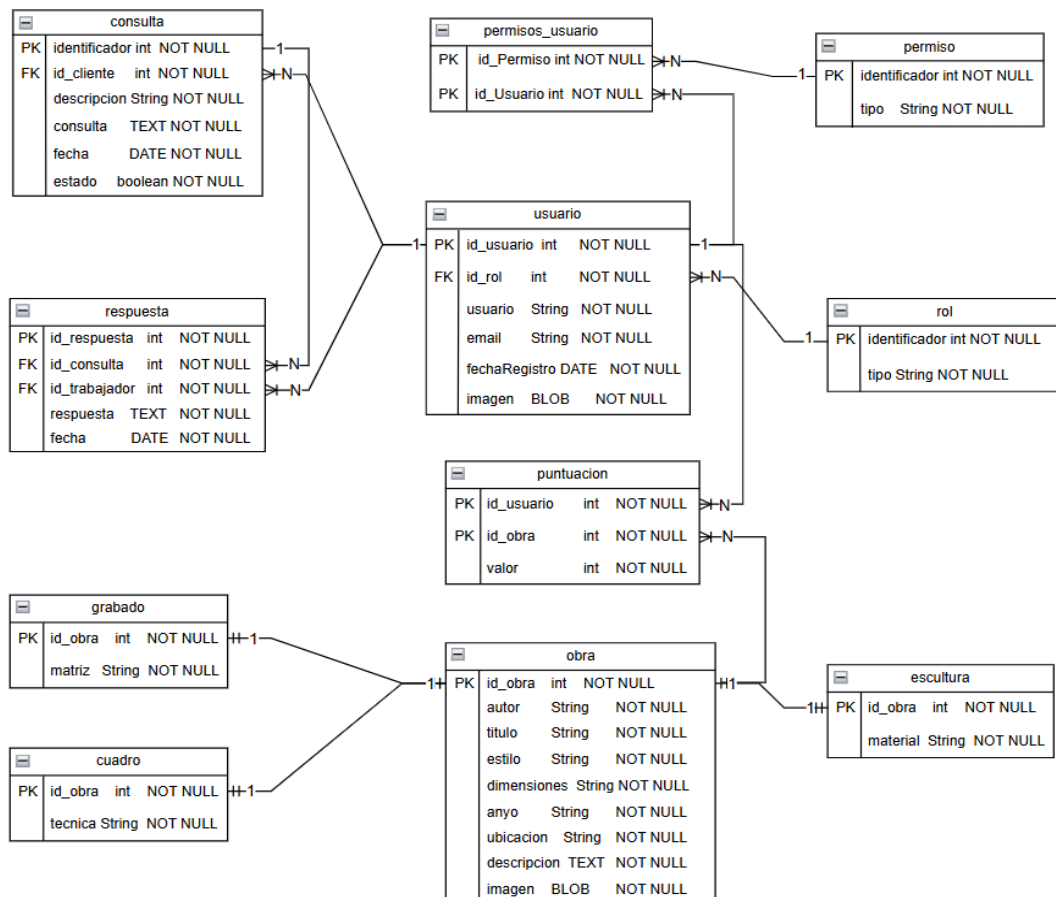


6.3. Diagramas Entidad/Relación

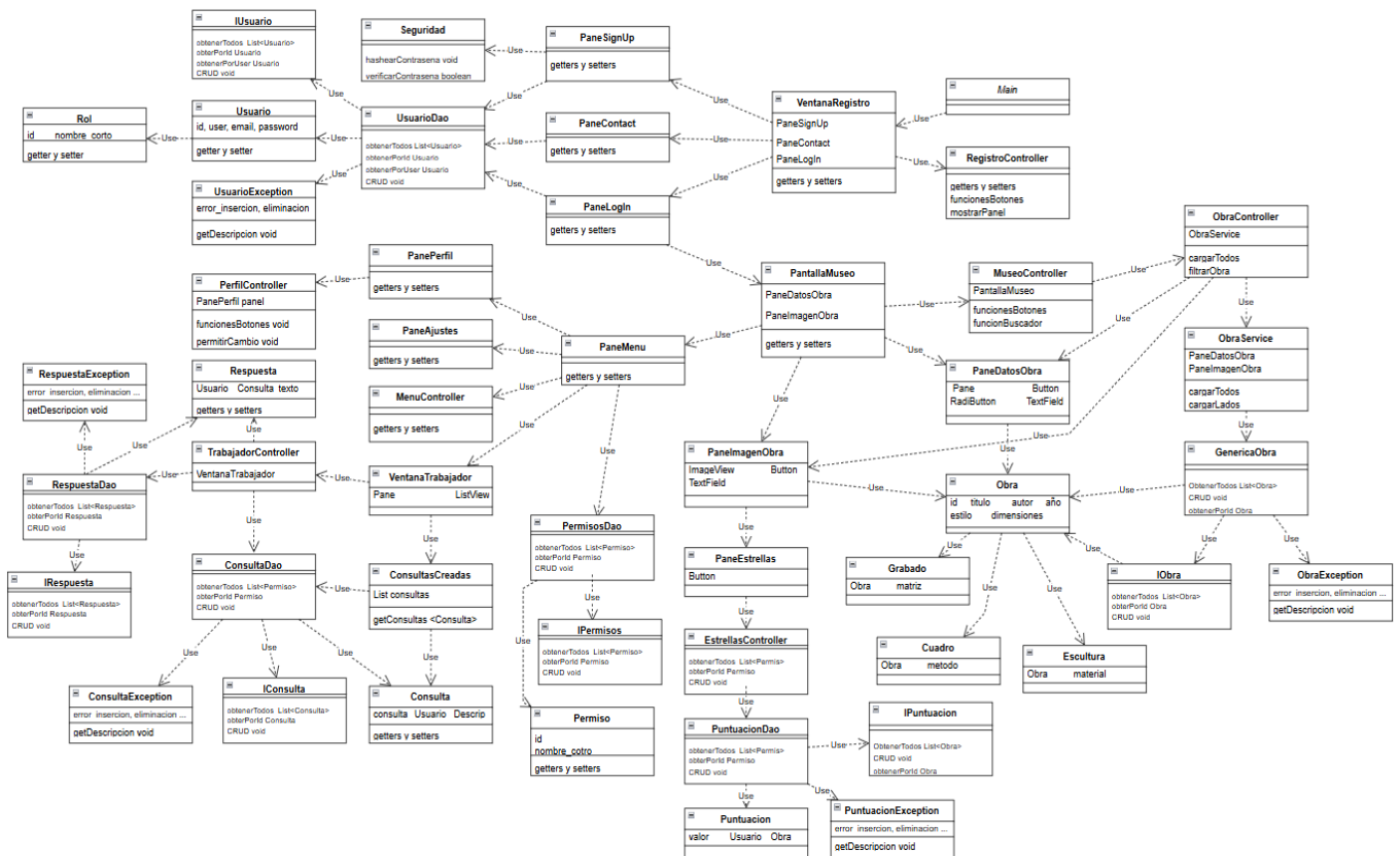
Modelo Entidad Relación



Modelo Relacional



6.4. Diagrama de clases



7. Desarrollo de aplicación

7.1. Tecnologías y herramientas utilizadas

Durante el desarrollo del proyecto se ha empleado diversas herramientas:

- Eclipse IDE: Entorno de desarrollo utilizado para escribir y depurar código Java.
- Java 23: Lenguaje de programación principal del proyecto.
- JavaFX: Framework utilizado para el desarrollo de la interfaz gráfica de usuario.
- DBeaver: Herramienta de gestión de bases de datos que ha facilitado la visualización, control de la base de datos MySQL, tanto en entorno local como en el servidor remoto.
- MySQL: Sistema de gestión de bases de datos relacional utilizado para almacenar la información relacionada con la aplicación.
- Hibernate: Framework de mapeo objeto-relacional que permite la interacción entre las clases Java y la base de datos.
- Patrón DAO: Patrón de diseño empleado para separar la lógica de acceso a datos del resto de la aplicación.
- Modelo Vista Controlador (MVC): Arquitectura empleada para organizar la aplicación en componentes principales: el modelo (datos), la vista (interfaz gráfica), y el controlador (lógica). Esto permite un desarrollo modular y facilita futuras ampliaciones.
- BCrypt: Algoritmo utilizado para asegurar contraseñas, garantizando que los datos sensibles no se almacenen en texto plano.
- Jakarta.mail: librería que permite la creación, envío, recepción y manejo de correos electrónicos a través de protocolos como SMTP.

7.2. Descripción de las principales funcionalidades

Función de Registro

```
IUsuario usu = new UsuarioDao();
// Poner no visible TextField de error, ya que al pulsar no debería aparecer ya
// que seguramente se ha solucionado el error
signup.getError().setVisible(false);
// Comprobar si hay algún campo vacío
if (signup.getEmail().getText().strip().isEmpty() || signup.getPsw().getText().strip().isEmpty() ||
    signup.getTxtUsu().getText().strip().isEmpty() || signup.getPsw2().getText().strip().isEmpty()) {
    signup.getError().setText("Debes rellenar todos los campos");
    signup.getError().setVisible(true);
    return;
}
// Comprobar si condiciones están aceptadas
if (signup.getCheck().isSelected()) {
    signup.getError().setText("No ha aceptado los términos y condiciones");
    signup.getError().setVisible(true);
    return;
}
// Comprobar las contraseñas son iguales
if (signup.getPsw().getText().strip().equals(signup.getPsw2().getText().strip())) {
    signup.getError().setText("Las contraseñas no coinciden");
    signup.getError().setVisible(true);
    return;
}
// Las contraseñas no son iguales por lo que no debería ejecutar más código
return;

String user = signup.getTxtUsu().getText().strip();
// Comprobar si el usuario existe en la bd ya que es único
try {
    if (usu.existeUsuario(new Usuario(user))) {
        signup.getError().setText("El usuario ya existe");
        signup.getError().setVisible(true);
        return;
    }
} catch (UsuarioException e) {
    System.out.println(e);
}

Rol rol = new Rol();
// Comprobar si el email existe
String email = signup.getEmail().getText().strip();
if (email.endsWith("@gmail.com") || email.endsWith("@hotmail.com")) {
    rol.setId(1);
} else {
    signup.getError().setText("El email no es válido");
    signup.getError().setVisible(true);
    return;
}
// Obtener la imagen del círculo
Image img = (signup.getImage().getFill() instanceof ImagePattern)
    ? ((ImagePattern) signup.getImage().getFill()).getImage(0, null);

byte[] imagen = UtilsData.redimensionarImagenAByte(img);
String contra = Seguridad.hashearContraseña(signup.getPsw().getText().strip());

Usuario usu = new Usuario(user, contra, email, rol, LocalDate.now(), imagen);

try {
    usu.insertar(usu);
} catch (UsuarioException e) {
    System.out.println(e);
}

signup.getSalir().fire();
```

Función de Conectarse

```
// Botón para registrarse
login.getLogin().setOnAction(event -> {

    IUsuario usu = new UsuarioDao();
    login.getError().setVisible(false);

    if (login.getPsw().getText().strip().isEmpty() || login.getTxtUsu().getText().strip().isEmpty()) {
        login.getError().setText("Debes rellenar todos los campos");
        login.getError().setVisible(true);
        return;
    }

    if (login.getRecordar().isSelected()) {
        GestorFicheroConfiguracion.actualizarValor("usuario", login.getTxtUsu().getText().strip());
        GestorFicheroConfiguracion.actualizarValor("password", login.getPsw().getText().strip());
        GestorFicheroConfiguracion.actualizarValor("recordar", "true");
    } else {
        GestorFicheroConfiguracion.actualizarValor("recordar", "false");
        GestorFicheroConfiguracion.actualizarValor("usuario", "");
        GestorFicheroConfiguracion.actualizarValor("password", "");
    }

    // Busca el cliente en la base de datos con:
    Usuario u = null;
    try {
        u = usu.obtenerPorUser(new Usuario(login.getTxtUsu().getText().strip()));
    } catch (UsuarioException e) {
        System.out.println(e);
    }

    if (u == null) {
        login.getError().setText("Alguna de las credenciales no concuerdan");
        login.getError().setVisible(true);
        return;
    }

    // Comprobar contraseña hasheada
    if (!Seguridad.verificarContraseña(login.getPsw().getText().strip(), u.getPassword()) {
        login.getError().setText("Alguna de las credenciales no concuerdan");
        login.getError().setVisible(true);
        return;
    }

    primaryStage.setResizable(true);
    primaryStage.setScene(new PantallaMuseo(primaryStage, u).getScene());
    primaryStage.setMaximized(true);
    vaciarPaneLogin();
});
```

Función Filtrar Obras

```
public void hacerFiltrado(String tipo) {
    try {
        switch (tipo) {
            case "Cuadro":
                IObra<Cuadro> cua = new GenericaObra<Cuadro>(Cuadro.class);
                obras.clear();
                obras.addAll(cua.obtenerTodos());
                Collections.shuffle(obras);
                break;
            case "Grabado":
                IObra<Grabado> gra = new GenericaObra<Grabado>(Grabado.class);
                obras.clear();
                obras.addAll(gra.obtenerTodos());
                Collections.shuffle(obras);
                break;
            case "Escultura":
                IObra<Escultura> esc = new GenericaObra<Escultura>(Escultura.class);
                obras.clear();
                obras.addAll(esc.obtenerTodos());
                Collections.shuffle(obras);
                break;
            default:
                obras.clear();
                cargarTodos();
        }
        pos = 0;
    } catch (ObraException e) {
        System.out.println(e);
    }
}

}
```

Función Obtener Obras

```
public void cargarTodos() {
    IObra<Cuadro> cua = new GenericaObra<Cuadro>(Cuadro.class);
    IObra<Grabado> gra = new GenericaObra<Grabado>(Grabado.class);
    IObra<Escultura> esc = new GenericaObra<Escultura>(Escultura.class);

    try {
        obras.addAll(cua.obtenerTodos());
        obras.addAll(gra.obtenerTodos());
        obras.addAll(esc.obtenerTodos());
    } catch (ObraException e) {
        System.out.println(e);
    }

    Collections.shuffle(obras);
}
```

Función Enviar Correo

```
public static void enviarCorreo(String cuerpo, String correo) {
    final String smtpHost = "smtp.gmail.com"; // Solo funciona por gmail
    final int puerto = 587;
    final String usuario = "eduardoandrespop2004@gmail.com";
    final String password = GestorFicheroConfiguracion.obtenerRuta("conEmail");

    Properties props = new Properties();
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.host", smtpHost);
    props.put("mail.smtp.port", String.valueOf(puerto));

    Session session = Session.getInstance(props, new Authenticator() {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(usuario, password);
        }
    });

    try {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress("service@miempresa.com")); // Gmail no permite q
        message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(correo));
        message.setSubject("Respuesta a consulta");
        message.setText(cuerpo);

        Transport.send(message);

        System.out.println("Correo enviado a " + correo);
    } catch (MessagingException e) {
        e.printStackTrace();
    }
}
```

Función Buscador

```
// Permite filtrar las coincidencias cuando se escribe en el buscador
pantallaMuseo.getBuscador().textProperty().addListener((observable, oldValue, newValue) -> {
    if (newValue.isEmpty()) {
        pantallaMuseo.getListView().setVisible(false); // Ocultar si no hay texto
    } else {
        // Buscamos las coincidencias
        List<Obra> filteredData = new ArrayList<Obra>();
        for (Obra item : obraService.obtenerTodos()) {
            if (item.getTitulo().toLowerCase().startsWith(newValue.toLowerCase())) {
                filteredData.add(item);
            }
        }

        pantallaMuseo.getListView().getItems().setAll(filteredData);
        if (pantallaMuseo.getListView().getItems().isEmpty()) { // Si está vacío no es visible
            pantallaMuseo.getListView().setVisible(false);
        } else { // Hacemos visible y ajustamos el tamaño del contenedor según los posibles
            // resultados
            pantallaMuseo.getListView().setVisible(true);
            switch (pantallaMuseo.getListView().getItems().size()) {
                case 1:
                    pantallaMuseo.getListView().setPrefHeight(25);
                    break;
                case 2:
                    pantallaMuseo.getListView().setPrefHeight(50);
                    break;
                default:
                    pantallaMuseo.getListView().setPrefHeight(75);
                    break;
            }
        }
    }
});
```

8. Pruebas y validación

Durante el proceso de desarrollo de la aplicación se han realizado diversas pruebas para garantizar el correcto funcionamiento del código.

8.1. Pruebas unitarias

Estas son pruebas que se hacen a partes pequeñas de código (como métodos) para asegurarse de que cada una funciona correctamente por sí sola.

En este tipo de pruebas he verificado los siguientes apartados:

- Cambio de la variable que permite editar el perfil del usuario.
- Hashear la contraseña del usuario al registrarse.
- Desplazamiento lateral de las obras.
- Métodos específicos como el envío de correos, animación de botones y conversión de *String* a *Date*.

8.2. Pruebas de integración

Estas son pruebas que verifican si una funcionalidad completa del sistema (como un proceso o una tarea que hace el usuario) funciona como se espera.

En este tipo de pruebas he verificado los siguientes apartados:

- Obtención de todas las obras de arte de la Base de Datos.
- Obtención de las consultas almacenadas en la Base de Datos.
- Acceso a la información de los usuarios registrados.
- Tratar de iniciar sesión.
- Funcionamiento del filtrado de la lista de obras que se van a mostrar.

9. Conclusiones y trabajo futuro

Una vez finalizado el desarrollo del proyecto, se puede concluir que, se han cumplido los objetivos planteados inicialmente. La aplicación permite un a los usuarios consultar información acerca de distintas obras de arte, de forma rápida y sencilla, cumpliendo con su propósito principal de acercar el arte a personas con poco conocimiento.

Durante el proceso, se han puesto en práctica diversos conocimientos adquiridos a lo largo del ciclo formativo. Esto ha supuesto un aprendizaje muy valioso tanto a nivel técnico como personal.

En cuanto al trabajo futuro, se tienen en cuenta varios puntos que podrían mejorar la aplicación.

- Mejora en el mantenimiento de los datos, permitiendo la inserción de nuevas obras de forma más sencilla, así como la inclusión de nuevos tipos de obras o campos.
- Reducción del peso de las imágenes mediante redimensionado o compresión, con el fin de optimizar el rendimiento de la base de datos.
- Incorporación de más ajustes y elementos interactivos, como música de fondo o efectos visuales para mejorar la experiencia del usuario.

En definitiva, este proyecto ha sido una oportunidad excelente para consolidar conocimientos y a enfrentarse a problemas reales de desarrollo de aplicaciones.

10. Relación del proyecto con los módulos del ciclo

El proyecto integra conocimientos adquiridos durante el ciclo formativo de Desarrollo de Aplicaciones Multiplataforma (DAM). A continuación, se detalla cómo se relaciona cada módulo con las distintas partes del desarrollo del proyecto.

Programación (1º curso)

Este módulo asentó las bases para la lógica del proyecto. La aplicación está desarrollada en Java, haciendo uso de la Programación Orientada a Objetos (POO), incluyendo conceptos como la herencia, polimorfismo y encapsulamiento.

Bases de Datos (1º curso)

De este módulo se aplicaron conceptos fundamentales como el diseño de la base de datos relacional (mediante diagramas entidad/relación), la normalización de datos y el uso de MySQL.

Entornos de Desarrollo (1º curso)

A lo largo del desarrollo del proyecto se aplicaron buenas prácticas como la organización de código, el uso de control de versiones (por ejemplo, Git) y la creación de pruebas unitarias. También se abordó el diseño inicial mediante modelos entidad/relación.

Acceso a Datos (2º curso)

Ha sido uno de los pilares técnicos del proyecto. Se ha trabajado con el ORM Hibernate para gestionar la persistencia de objetos en la base de datos, también se utilizaron herramientas como MySQL y DBeaver para administrar y visualizar la base de datos. Se aplicó en Patrón DAO y servidores AWS.

Servicios y Procesos (2º curso)

Este módulo se ha visto reflejado en la gestión de operaciones que requieren hilos, como la música de fondo.

Desarrollo de Interfaces (2º curso)

Este módulo ha sido esencial en la creación de la interfaz gráfica de usuario, desarrollada con JavaFX. Aunque en clase se trabajó con Swing, el paso a JavaFX ha sido sencillo ya que comparten conceptos similares.

Inglés Técnico (2º curso)

Se ha utilizado vocabulario técnico en inglés en la interfaz, fomentando así la familiarización con términos del ámbito profesional.

11. Bibliografía/Webgrafía

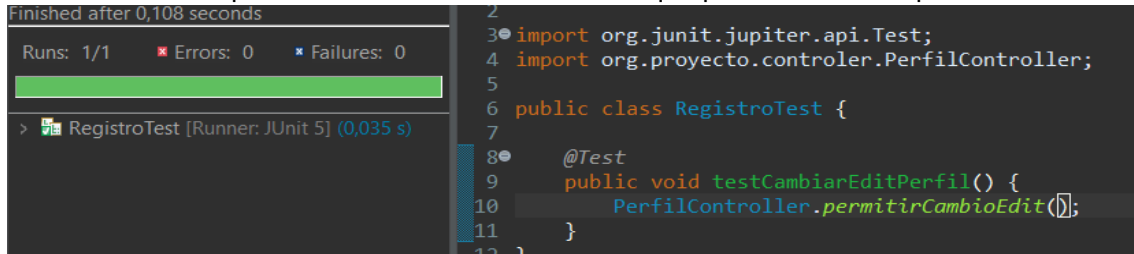
- chatopenai.com
- manual-informatica.com
- github.com
- www.bcrypt.io/
- mailtrap.io/blog/jakarta-mail-tutorial/
- www.baeldung.com/
- jenkov.com/tutorials/javafx/index.html

12. Anexos

Anexo I: Pruebas Unitarias

En este anexo se detallan las pruebas unitarias realizadas durante el desarrollo de la aplicación, las cuales ya se detallaron previamente.

- Esta es la prueba del cambio de la variable que permite editar el perfil del usuario.



The screenshot shows the IDE interface with the test results on the left and the source code on the right. The test results pane indicates 'Finished after 0,108 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. The test 'RegistroTest [Runner: JUnit 5] (0,035 s)' is listed. The source code on the right shows the test method:

```
2
3 import org.junit.jupiter.api.Test;
4 import org.proyecto.controller.PerfilController;
5
6 public class RegistroTest {
7
8     @Test
9     public void testCambiarEditPerfil() {
10         PerfilController.permitirCambioEdit();
11     }
12 }
```

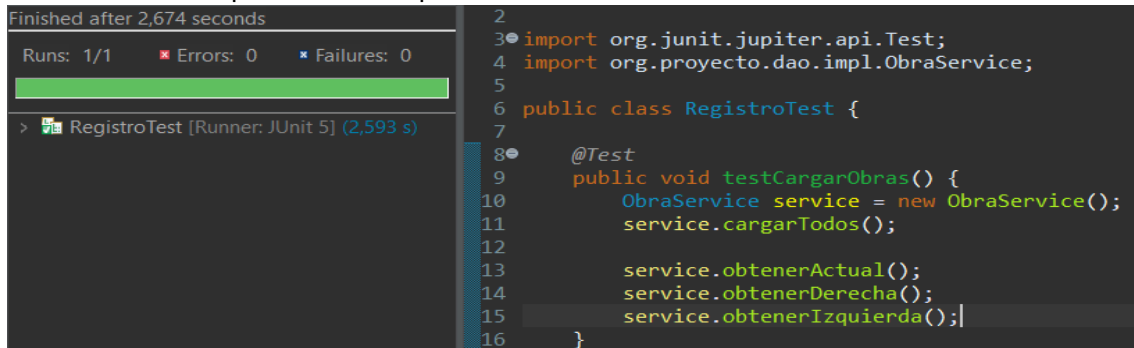
- Esta es la prueba de generación del hash de la contraseña del usuario al registrarse.



The screenshot shows the IDE interface with the test results on the left and the source code on the right. The test results pane indicates 'Finished after 0,512 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. The test 'RegistroTest [Runner: JUnit 5] (0,397 s)' is listed. The source code on the right shows the test method:

```
2
3 import org.junit.jupiter.api.Test;
4 import org.proyecto.util.Seguridad;
5
6 public class RegistroTest {
7
8     @Test
9     public void testHasearContrasena() {
10         Seguridad.hasearContrasena("contrasena");
11         Seguridad.verificarContrasena("Contraseña", "contra");
12     }
13 }
14 }
```

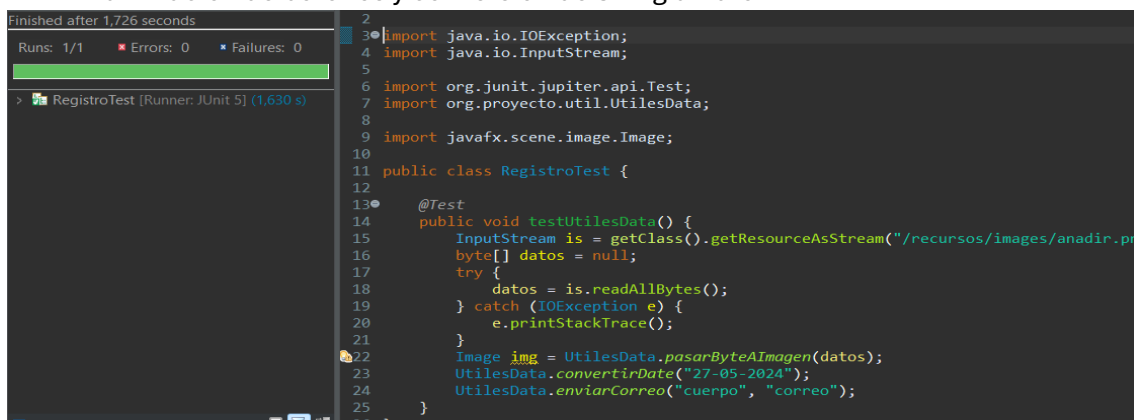
- Esta es la prueba del desplazamiento lateral de las obras.



The screenshot shows the IDE interface with the test results on the left and the source code on the right. The test results pane indicates 'Finished after 2,674 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. The test 'RegistroTest [Runner: JUnit 5] (2,593 s)' is listed. The source code on the right shows the test method:

```
2
3 import org.junit.jupiter.api.Test;
4 import org.proyecto.dao.impl.ObraService;
5
6 public class RegistroTest {
7
8     @Test
9     public void testCargarObras() {
10         ObraService service = new ObraService();
11         service.cargarTodos();
12
13         service.obtenerActual();
14         service.obtenerDerecha();
15         service.obtenerIzquierda();
16     }
17 }
```

- Esta son las pruebas de los métodos específicos como el envío de correos, animación de botones y conversión de *String* a *Date*.



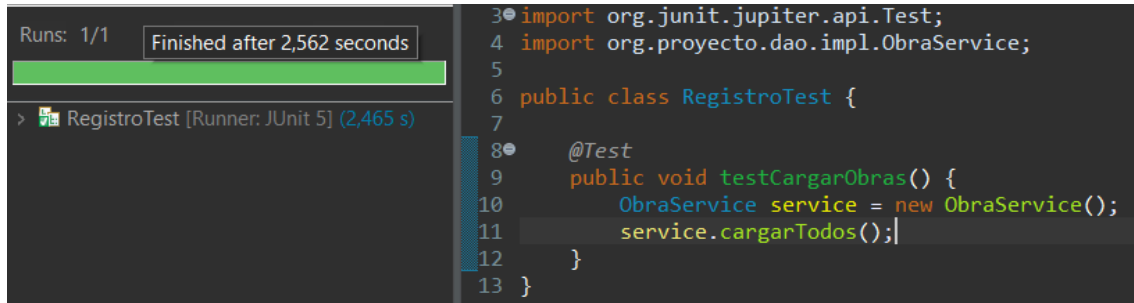
The screenshot shows the IDE interface with the test results on the left and the source code on the right. The test results pane indicates 'Finished after 1,726 seconds', 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. The test 'RegistroTest [Runner: JUnit 5] (1,630 s)' is listed. The source code on the right shows the test method:

```
2
3 import java.io.IOException;
4 import java.io.InputStream;
5
6 import org.junit.jupiter.api.Test;
7 import org.proyecto.util.UtilesData;
8
9 import javafx.scene.image.Image;
10
11 public class RegistroTest {
12
13     @Test
14     public void testUtilesData() {
15         InputStream is = getClass().getResourceAsStream("/recursos/images/anadir.png");
16         byte[] datos = null;
17         try {
18             datos = is.readAllBytes();
19         } catch (IOException e) {
20             e.printStackTrace();
21         }
22         Image img = UtilesData.pasarByteAImagen(datos);
23         UtilesData.convertirDate("27-05-2024");
24         UtilesData.enviarCorreo("cuerpo", "correo");
25     }
26 }
```

Anexo II: Pruebas Funcionales

En este anexo se detallan las pruebas funcionales realizadas durante el desarrollo de la aplicación, las cuales ya se detallaron previamente.

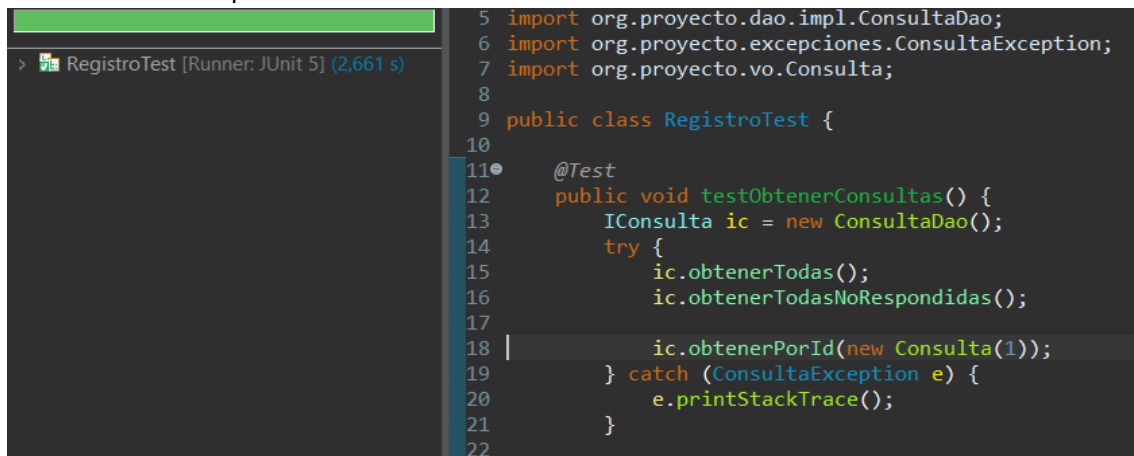
- Esta es la prueba de cargar todas las obras a la lista que se mostrará al usuario.



```
3 import org.junit.jupiter.api.Test;
4 import org.proyecto.dao.impl.ObraService;
5
6 public class RegistroTest {
7
8     @Test
9     public void testCargarObras() {
10         ObraService service = new ObraService();
11         service.cargarTodos();
12     }
13 }
```

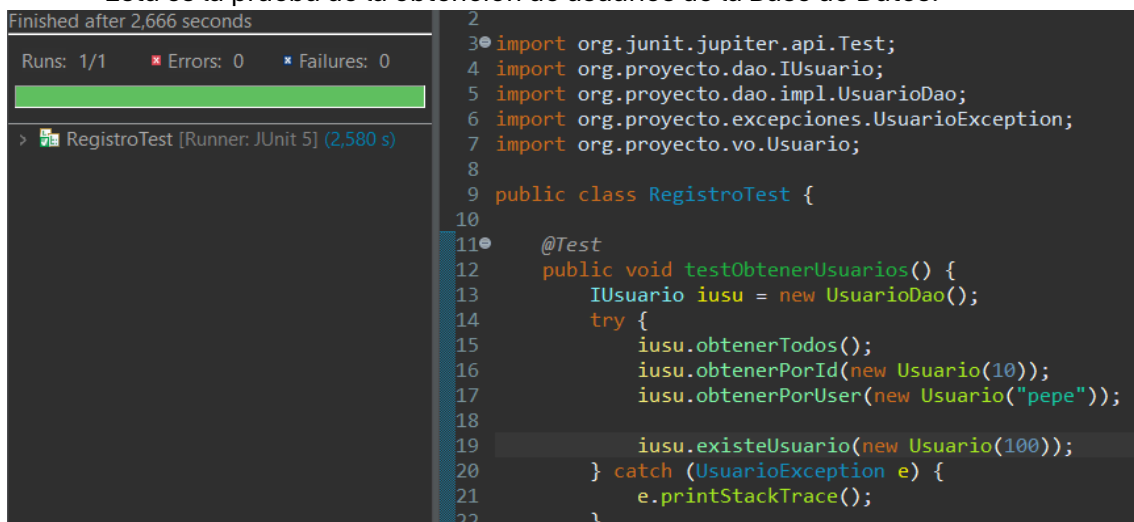
Como se pedía en los requisitos no funcionales en el apartado de rendimiento, se tarda alrededor de 2 segundos para obtener la lista de todas las obras de arte.

- Esta es la prueba de la obtención de las consultas de la Base de Datos.



```
5 import org.proyecto.dao.impl.ConsultaDao;
6 import org.proyecto.excepciones.ConsultaException;
7 import org.proyecto.vo.Consulta;
8
9 public class RegistroTest {
10
11     @Test
12     public void testObtenerConsultas() {
13         IConsulta ic = new ConsultaDao();
14         try {
15             ic.obtenerTodas();
16             ic.obtenerTodasNoRespondidas();
17             ic.obtenerPorId(new Consulta(1));
18         } catch (ConsultaException e) {
19             e.printStackTrace();
20         }
21     }
22 }
```

- Esta es la prueba de la obtención de usuarios de la Base de Datos.



```
2
3 import org.junit.jupiter.api.Test;
4 import org.proyecto.dao.IUsuario;
5 import org.proyecto.dao.impl.UsuarioDao;
6 import org.proyecto.excepciones.UsuarioException;
7 import org.proyecto.vo.Usuario;
8
9 public class RegistroTest {
10
11     @Test
12     public void testObtenerUsuarios() {
13         IUsuario iusu = new UsuarioDao();
14         try {
15             iusu.obtenerTodos();
16             iusu.obtenerPorId(new Usuario(10));
17             iusu.obtenerPorUser(new Usuario("pepe"));
18             iusu.existeUsuario(new Usuario(100));
19         } catch (UsuarioException e) {
20             e.printStackTrace();
21         }
22     }
23 }
```


- Esta es la prueba del intento de inicio de sesión de un usuario.

```

1 package test;
2
3 import org.junit.jupiter.api.Test;
4 import org.proyecto.controler.RegistroController;
5
6 public class RegistroTest {
7
8     @Test
9     public void testRegistro() {
10         RegistroController.registro("georumano", "pop");
11     }
12 }

```

Se ha tratado de hacer un intento de sesión con unas credenciales indorrectas.

- Esta es la prueba del funcionamiento del filtrado de las obras de arte.

```

1 package test;
2
3 import org.junit.jupiter.api.Test;
4 import org.proyecto.dao.impl.ObraService;
5
6 public class RegistroTest {
7
8     @Test
9     public void testCargarObras() {
10         ObraService service = new ObraService();
11         service.cargarTodos();
12         service.hacerFiltrado("Cuadro");
13         service.hacerFiltrado("Escultura");
14         service.hacerFiltrado("Grabado");
15         service.hacerFiltrado("Todo");
16     }
17 }

```

Según lo indicado en los requisitos no funcionales, en el apartado de rendimiento, la aplicación tarda unos 2 segundos en cargar todas las obras de arte, y alrededor de 1 segundo cuando se filtran por tipo.