



GESTIÓN 1DAW JAVAFX

Trabajo realizado por:
Nicolás Ortega Fernández

ÍNDICE

1. Especificación de requisitos.
2. Organización Tareas.
3. Tecnologías Utilizadas.
4. Librerías usadas.
5. Principios SOLID.
6. Diagrama de Clases.
7. Diagrama E/R.
8. Diagrama de Casos de Uso.
9. Tablas de casos de Uso
10. Mapa Navegacional.
11. Diagrama de Arquitectura.
12. Elementos de la arquitectura.
13. Ejemplos ejecución.
14. Estimación económica.
15. Video explicativo.

ESPECIFICACIÓN DE REQUISITOS

Requisitos funcionales

- RF1.** El sistema debe permitir registrar nuevos alumnos.
- RF2.** El sistema debe permitir modificar o eliminar los datos de los alumnos.
- RF3.** El sistema debe permitir matricular a los alumnos en los módulos disponibles.
- RF4.** El sistema debe evitar que un alumno se matricule más de una vez.
- RF5.** El sistema debe permitir ingresar y registrar las calificaciones de los alumnos.
- RF6.** El sistema debe permitir consultar el expediente académico de cada alumno (calificaciones, estado de matrícula).
- RF7.** El sistema debe permitir listar todos los alumnos matriculados.

Requisitos no funcionales

- RNF1.** La interfaz debe ser sencilla orientada a los profesores.
- RNF2.** El sistema debe guardar los datos de forma persistente en BBDD.
- RNF3.** El sistema debe incluir validaciones para evitar errores.
- RNF4.** El sistema debe poder gestionar al menos 30 alumnos sin afectar su rendimiento.

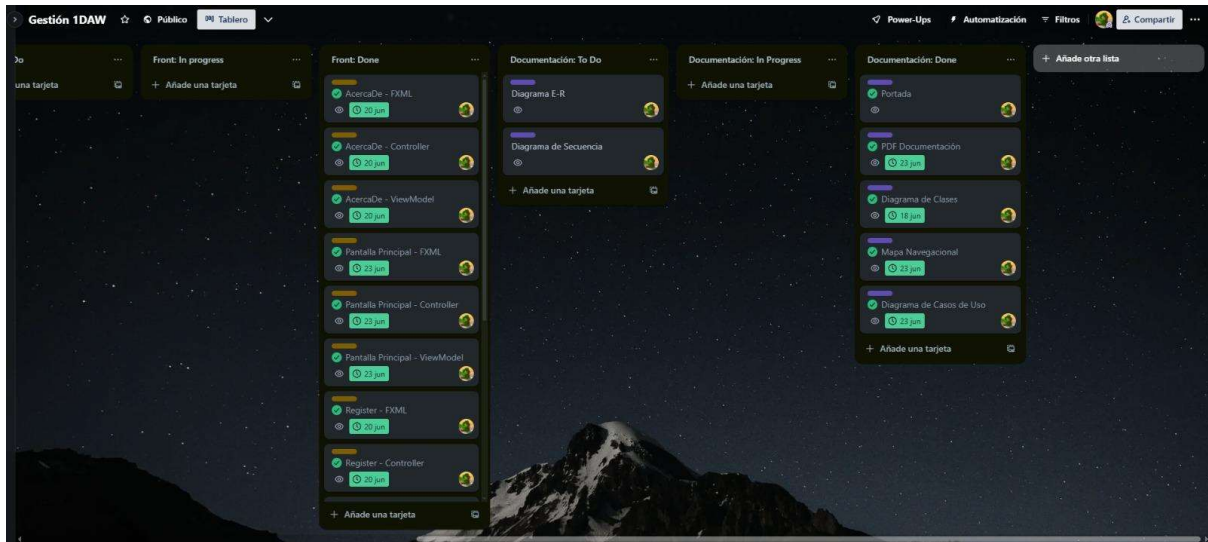
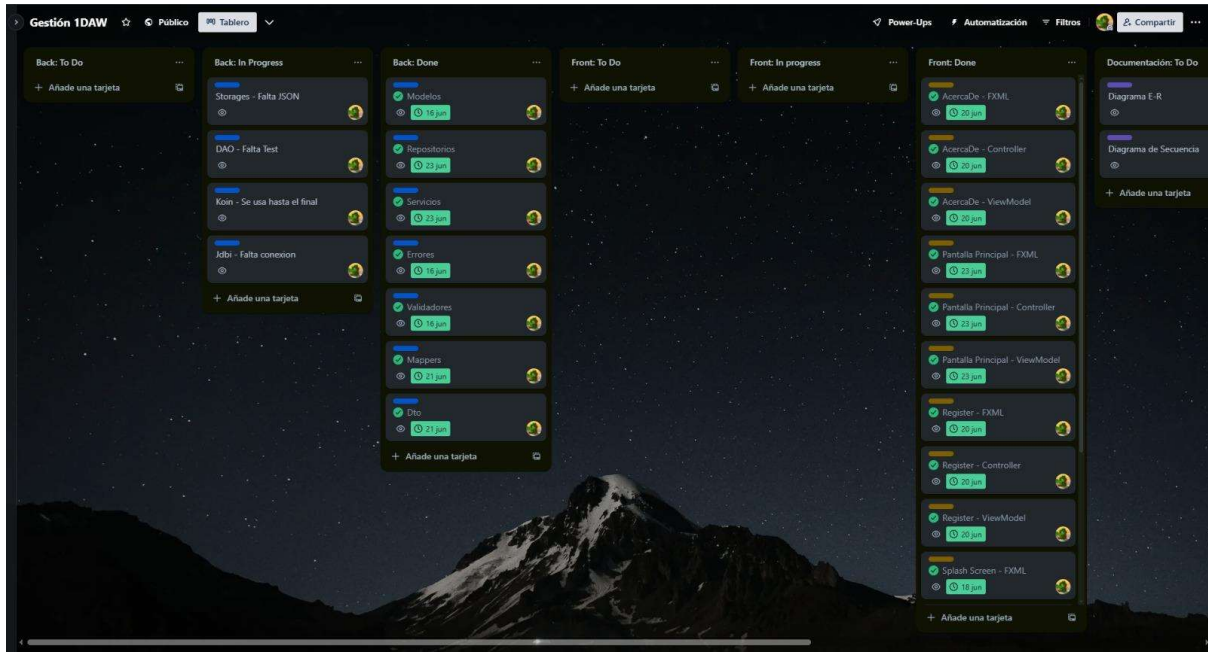
Requisitos informativos

- RI1.** El sistema será utilizado localmente, en un solo ordenador, sin conexión en red.
- RI2.** Cada alumno tiene un identificador único (por ejemplo, DNI o número de matrícula).
- RI3.** El sistema será operado exclusivamente por personal autorizado.

ORGANIZACIÓN TAREAS

Para el reparto de tareas se ha usado la web llamada Trello:

- Enlace: <https://trello.com/b/VRrZN3BE/gestion-1daw>



TECNOLOGÍAS UTILIZADAS

A continuación, un listado de todas las tecnologías de las que se ha hecho uso a lo largo de la práctica además del porque han sido usadas:

- **IntelliJ:** entorno de desarrollo principal usado para escribir y ejecutar el código. Se ha hecho uso de este entorno de desarrollo debido a que es el que se ha usado a lo largo de todo el curso, en parte por su facilidad a la hora de usar proyectos colaborativos y porque garantiza muy bien la calidad del código.
- **SceneBuilder:** software usado para diseñar la interfaz gráfica (GUI) de la aplicación. No solo es el software usado a lo largo del curso, sino que además es muy intuitivo y útil para perfiles con menos experiencia.
- **Visual Studio Code:** entorno de desarrollo secundario para revisar el código de los compañeros. Es el entorno más rápido si lo que se busca es simplemente visualizar código.
- **Git:** control de versiones para la gestión de cambios en el código. Se ha usado, además de porque es el estándar, ya que permite guardar un historial de cambios muy útil.
- **GitHub:** ha servido para alojar el repositorio del proyecto y poder gestionar los cambios desde ahí. Se ha usado ya que ofrece una integración directa con Git.
- **GitFlow:** para el flujo de trabajo, representado con ramas. Usado ya que muestra claramente la estructura en el desarrollo, lo que ha sido muy útil.
- **Windows Terminal:** para ejecutar los comandos y gestionar el proyecto desde terminal. Usada debido a que es la predeterminada y cubre todas las necesidades del proyecto.
- **Trello:** para el reparto de tareas. Se exigió en la anterior práctica y es muy cómodo en cuanto a gestionar que parte hace cada miembro del equipo
- **Canva:** para el diseño de la portada. Usado por su facilidad para hacer portadas en parte gracias a las plantillas.
- **Word:** para poder redactar la Documentación del proyecto. Usado ya que es el estándar en cuanto a procesadores de texto.

LIBREARÍAS USADAS

```
// Logger
implementation("org.lighthousegames:logging:1.3.0")
implementation("ch.qos.logback:logback-classic:1.5.13")

// BBDD H2
implementation("com.h2database:h2:2.2.224")

// JDBI
// Reflexión
implementation("org.jetbrains.kotlin:kotlin-reflect")
// Core
implementation("org.jdbi:jdbi3-core:3.48.0")
// SQL Object
implementation("org.jdbi:jdbi3-sqlobject:3.48.0")
// Extensión para Kotlin
implementation("org.jdbi:jdbi3-kotlin:3.48.0")
// Extensión de SQL Object para Kotlin
implementation("org.jdbi:jdbi3-kotlin-sqlobject:3.48.0")

// Result
implementation("com.michael-bull.kotlin-result:kotlin-result:2.0.0")

// Vaadin
implementation("com.vaadin:open:8.5.0.4")

// Caché Caffeine
implementation("com.github.ben-manes.caffeine:caffeine:3.2.0")

// Koin
implementation(platform("io.insert-koin:koin-bom:3.5.6"))
implementation("io.insert-koin:koin-core")

// JUnit
testImplementation("org.junit.jupiter:junit-jupiter-
api:${junitVersion}")
testRuntimeOnly("org.junit.jupiter:junit-jupiter-
engine:${junitVersion}")

// Jacoco
implementation("org.jacoco:org.jacoco.core:0.8.12")
```

```
testImplementation "org.jetbrains.kotlin:kotlin-test:1.9.23"

// Reflexión de tipos genéricos, las versiones más recientes de JDBI
no la incluyen por defecto
implementation 'io.leangen.geantyref:geantyref:1.3.0'

testImplementation("org.mockito.kotlin:mockito-kotlin:5.3.1")
testImplementation("org.mockito:mockito-junit-jupiter:5.12.0")
implementation("org.jetbrains.kotlin:kotlin-stdlib:1.9.23")
```

PRINCIPIOS SOLID

- **S — Principio de Responsabilidad Única (SRP)**

La clase `AlumnoServiceImpl` tiene una única responsabilidad: gestionar la lógica de negocio relacionada con los alumnos. No accede a la base de datos, no realiza validaciones, simplemente coordina esas acciones delegándolas a otras clases especializadas.

- **O — Principio de Abierto/Cerrado (OCP)**

`AlumnoServiceImpl` está abierta a extensión, pero cerrada a modificación. Por ejemplo, podemos cambiar la forma de validar o almacenar alumnos sin modificar la implementación

- **L — Principio de Sustitución de Liskov (LSP)**

La clase implementa la interfaz `AlumnoService`, lo cual permite que pueda ser sustituida por cualquier otra implementación de esa interfaz sin afectar al funcionamiento del programa.

- **I — Principio de Segregación de Interfaces (ISP)**

La interfaz `AlumnoService` está bien definida, con métodos específicos y necesarios para la gestión de alumnos. No obliga a las clases a implementar métodos que no usen. Gracias a esto la interfaz no es demasiado grande ni genérica.

- **D — Principio de Inversión de Dependencias (DIP)**

AlumnoServiceImpl depende de abstracciones, no de implementaciones concretas. Aunque en este caso se usa AlumnoRepositoryImpl, el servicio depende de la interfaz que esta implementa, por lo que se podría cambiar por otra implementación fácilmente.

DIAGRAMA DE CLASES

Diagrama de clases de modelo de negocio:

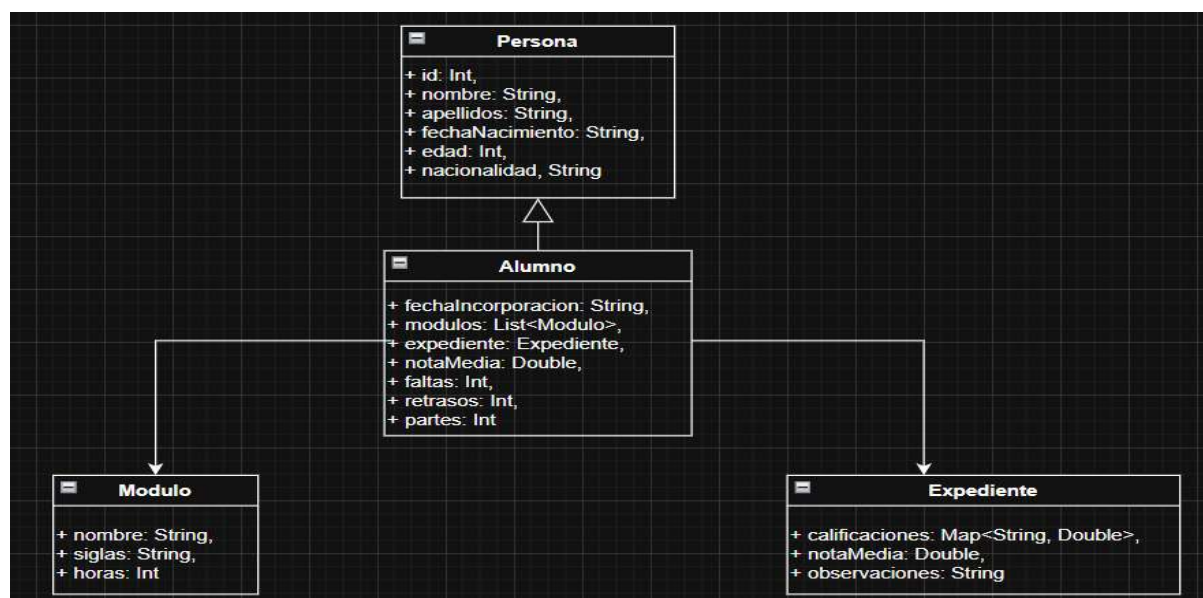


DIAGRAMA ENTIDAD/RELACIÓN

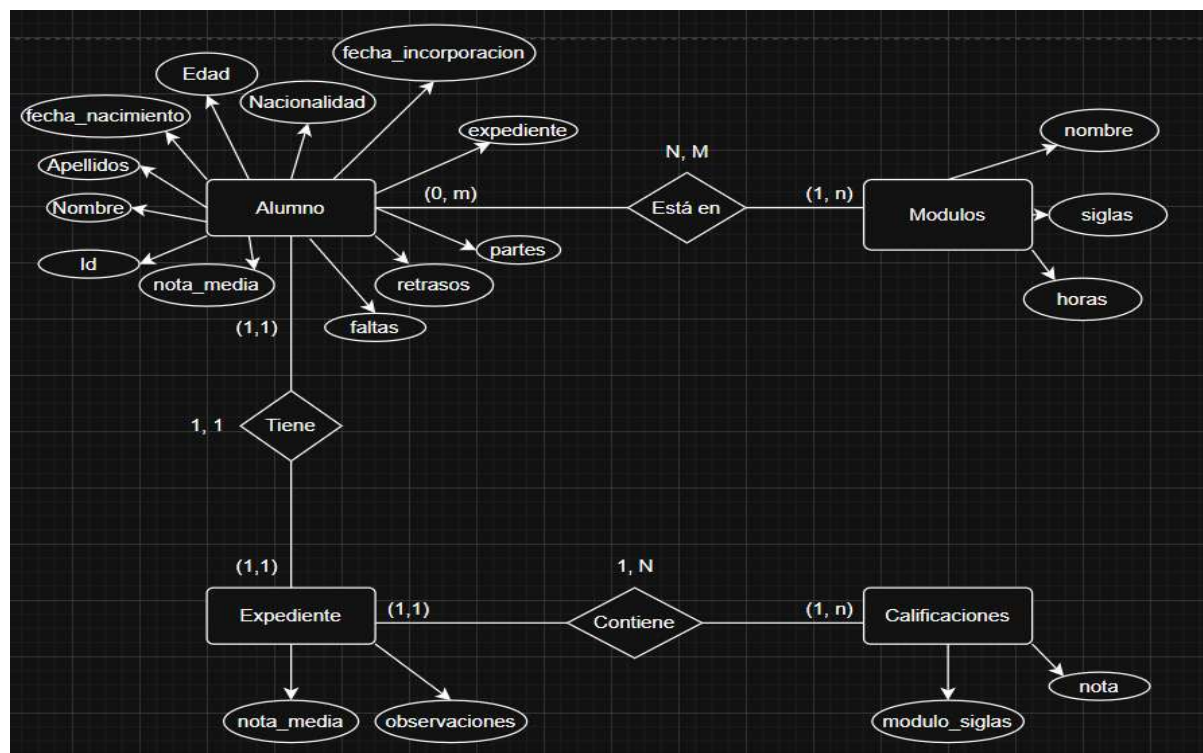
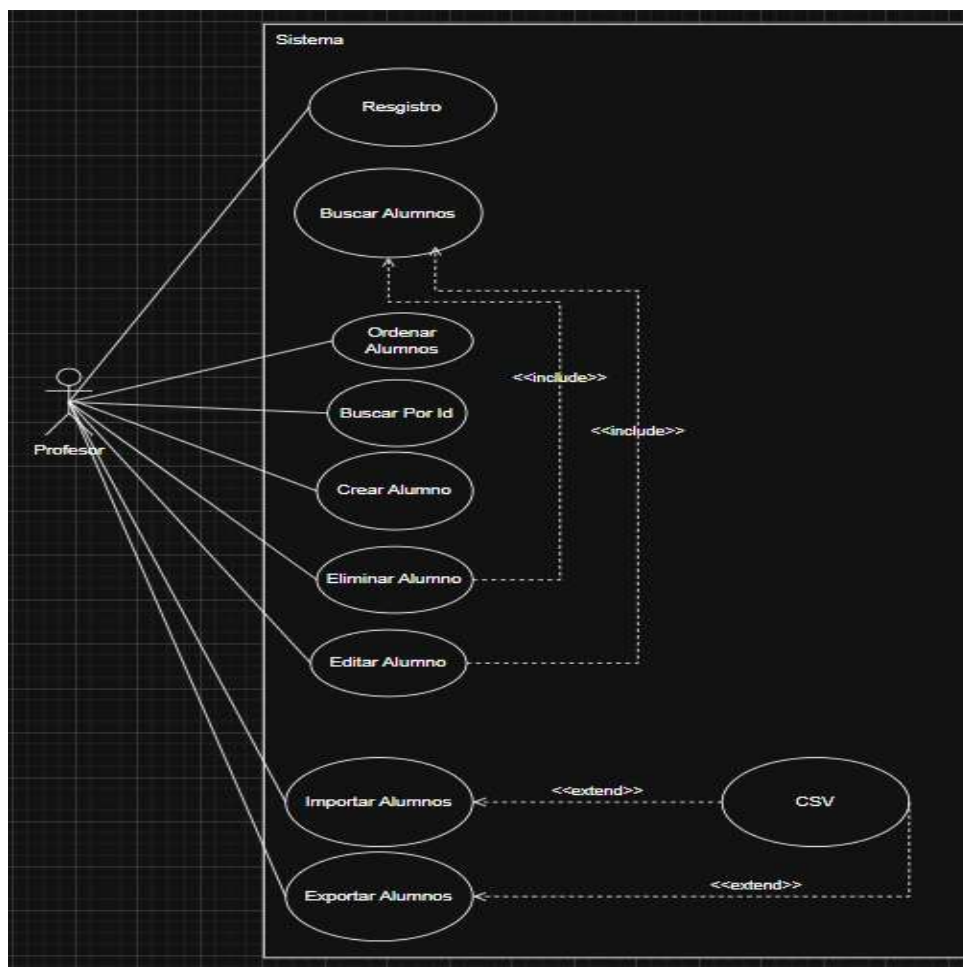


DIAGRAMA DE CASOS DE USO



TABLAS DE CASOS DE USO

Caso de uso	Crear Alumno
Id	CU-1
Descripción	El profesor crea un alumno con datos vacíos y lo inserta en la BBDD
Actor implicado	Profesor
Precondiciones	El profesor debe estar registrado en el sistema
Curso normal	<ol style="list-style-type: none"> 1. El profesor clicka sobre “Crear Alumno” 2. El sistema inserta en la BBDD un Alumno con los datos vacíos. 3. El profesor introduce los datos. 4. El profesor clicka en “Guardar Cambios” 5. El sistema valida la entrada de datos. 6. El sistema guarda al Alumno
Postcondiciones	El alumno debe quedar preparado para aparecer en consultas
Alternativa	El validador da error y no actualizar los datos del alumno o no hay conexión con la BBDD

Caso de uso	Eliminar Alumno
Id	CU-2
Descripción	El profesor elimina a un alumno y lo borra de la BBDD
Actor implicado	Profesor
Precondiciones	El profesor debe estar registrado en el sistema. Debe existir al menos 1 alumno.
Curso normal	<ol style="list-style-type: none"> 1. El profesor seleccionar un alumno. 2. El profesor clicka "Eliminar Alumno". 3. El sistema borra el alumno de la BBDD. 4. La tabla de alumnos se actualizar y se bajan todos los ID 1, pero solo los superiores al alumno borrado.
Postcondiciones	La BBDD ya no muestra al alumno
Alternativa	No hay conexión con la BBDD

Caso de uso	Actualizar Alumno
Id	CU-3
Descripción	El profesor actualiza los datos de un alumno y se guardan en la BBDD
Actor implicado	Profesor
Precondiciones	El profesor debe estar registrado en el sistema. Debe existir al menos 1 alumno.
Curso normal	<ol style="list-style-type: none"> 1. El profesor selecciona un alumno. 2. El sistema muestra los datos del alumno. 3. El profesor cambia campos del alumnos. 4. El profesor clicka "Guardar Cambios" 5. El sistema valida la entrada de datos. 6. El sistema actualiza los datos en la BBDD
Postcondiciones	Los datos cambiados se cambian en tiempo real en la BBDD
Alternativa	No hay conexión con la BBDD o el validador falla.

Caso de uso	Buscar por ID
Id	CU-4
Descripción	El profesor busca el alumno de un ID determinado
Actor implicado	Profesor
Precondiciones	El profesor debe estar registrado en el sistema. Debe existir al menos 1 alumno.
Curso normal	<ol style="list-style-type: none"> 1. El profesor introduce el ID en el buscador. 2. El sistema busca al alumno con ese ID. 3. El sistema muestra al alumno
Postcondiciones	En la tabla solo se ve ese alumno
Alternativa	No hay conexión con la BBDD.

MAPA NAVEGACIONAL

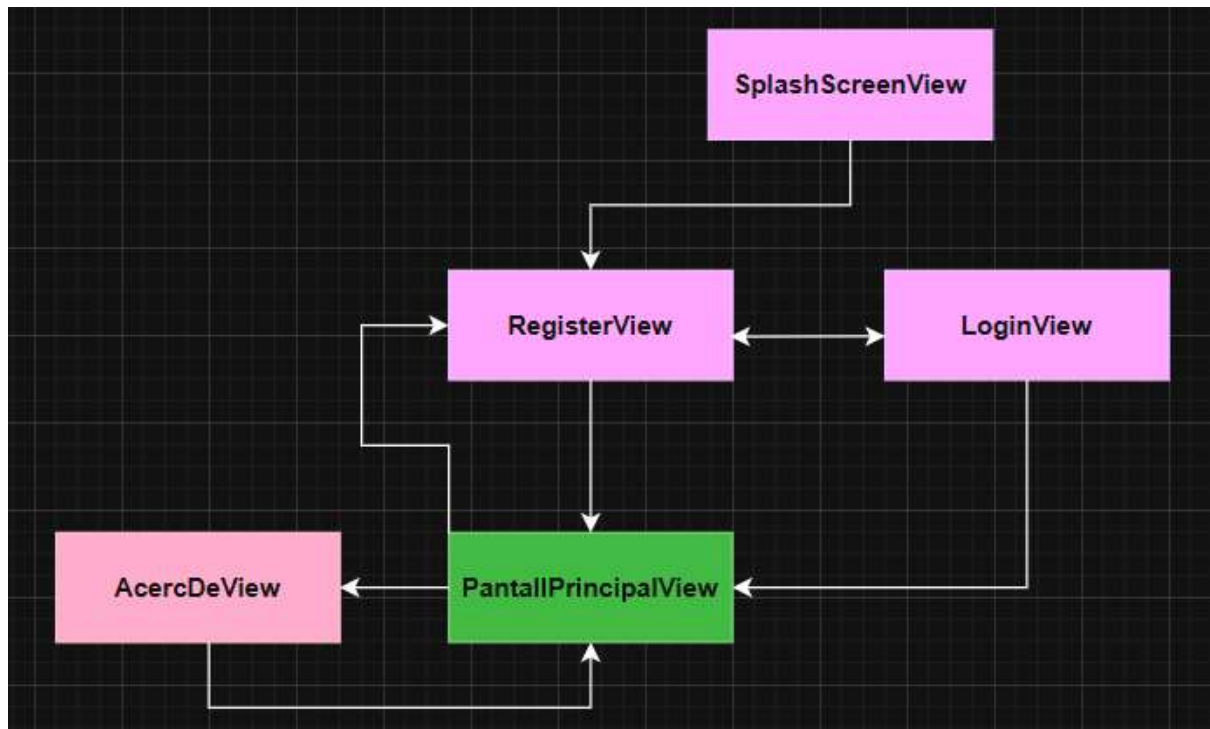
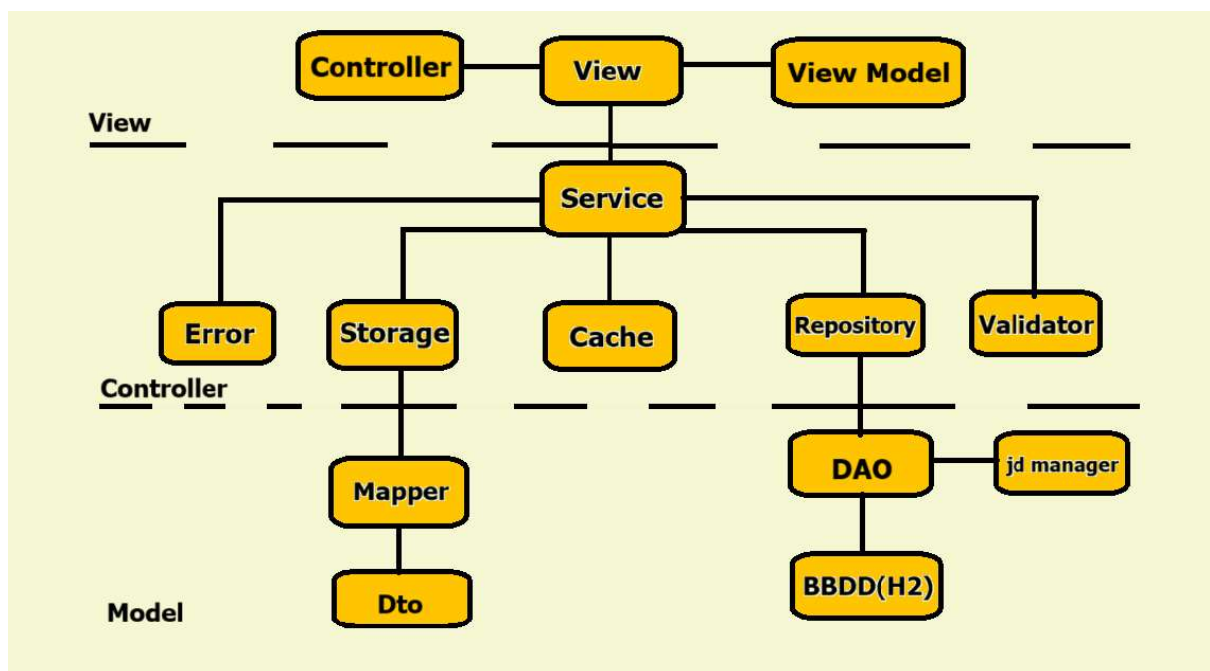
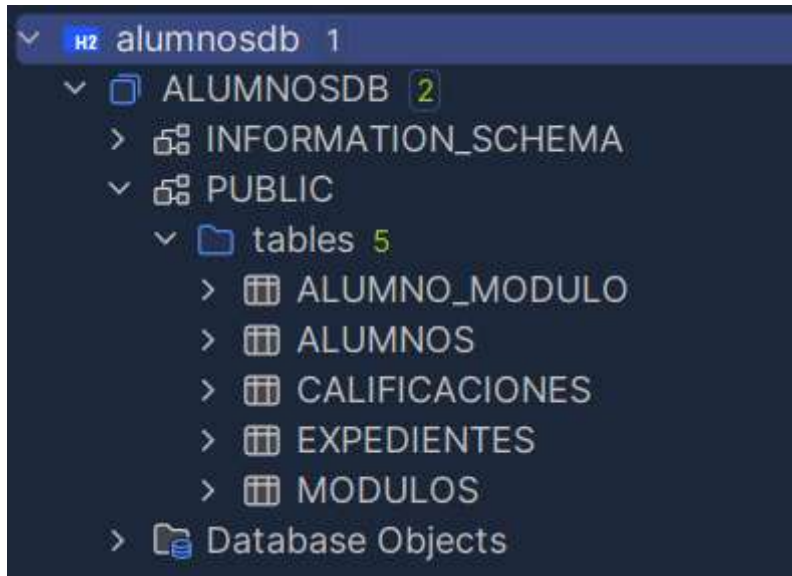


DIAGRAMA DE ARQUITECTURA



BASE DE DATOS



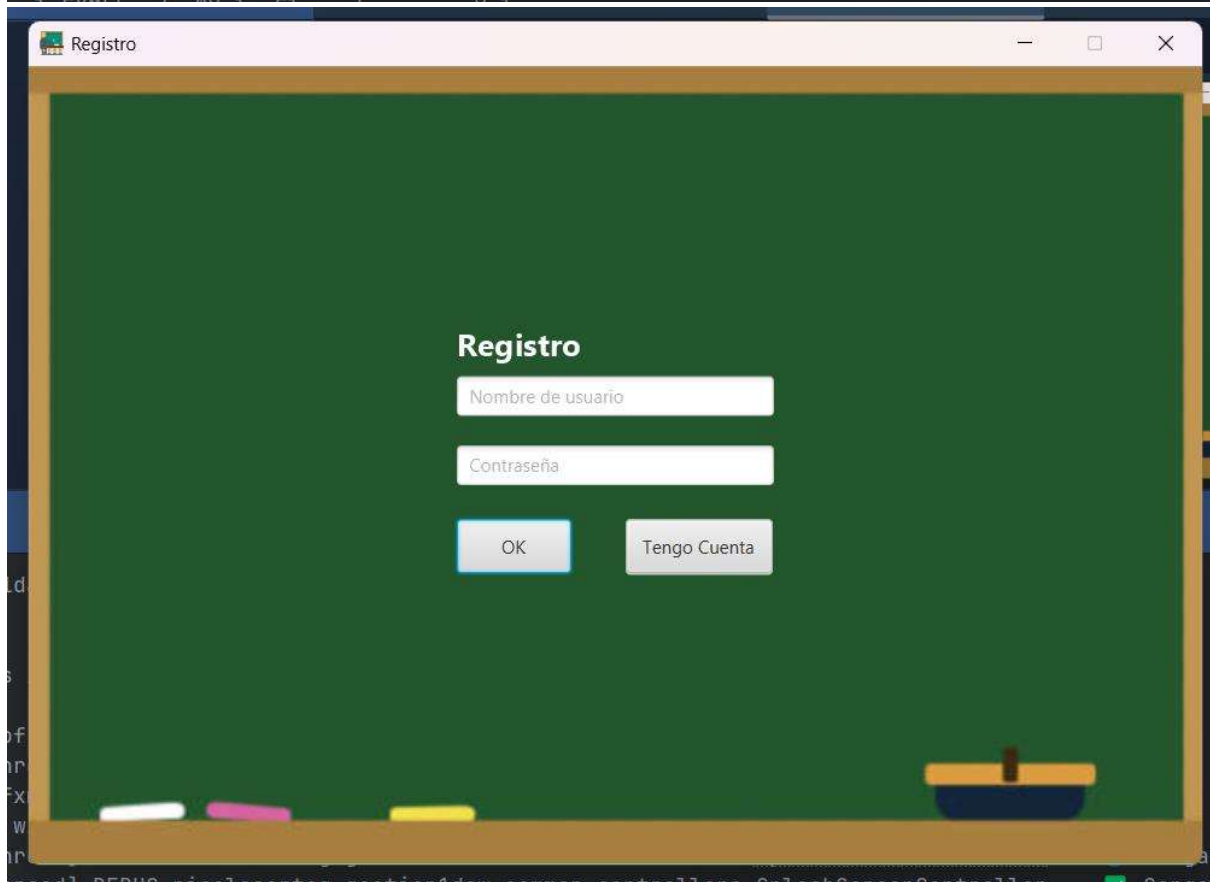
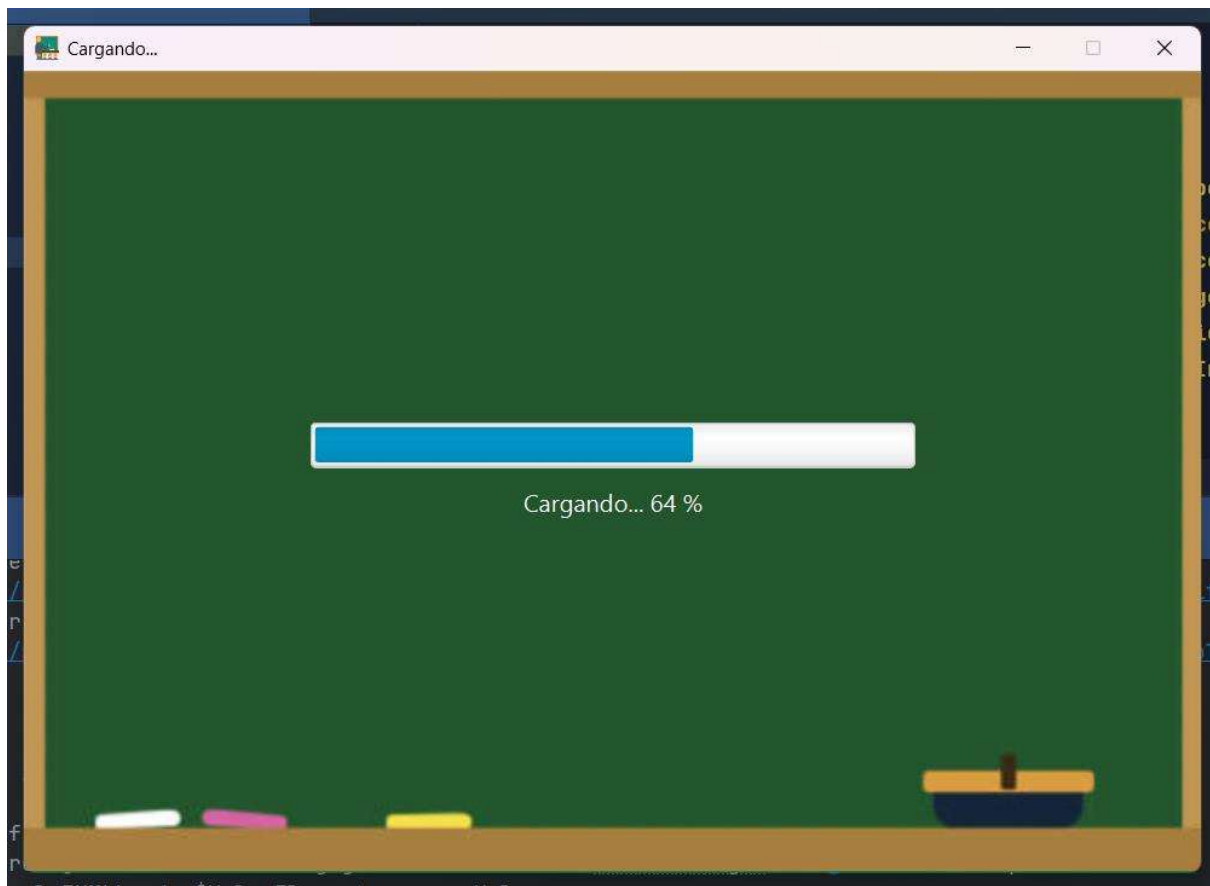
ELEMENTOS DE LA ARQUITECTURA

Se usa una arquitectura basada en MVC, la **MVVM**:

- **Model**: representa los datos de la aplicación, por ejemplo, los modelos de los alumnos
- **View**: interfaz con la que el usuario interactúa. En nuestro caso hemos usado JavaFx para esto.
- **ViewMdel**: lleva a cabo la interacción entre modelo vista y servicio.

EJEMPLOS EJECUCIÓN

Todas estas imágenes están en el repositorio:



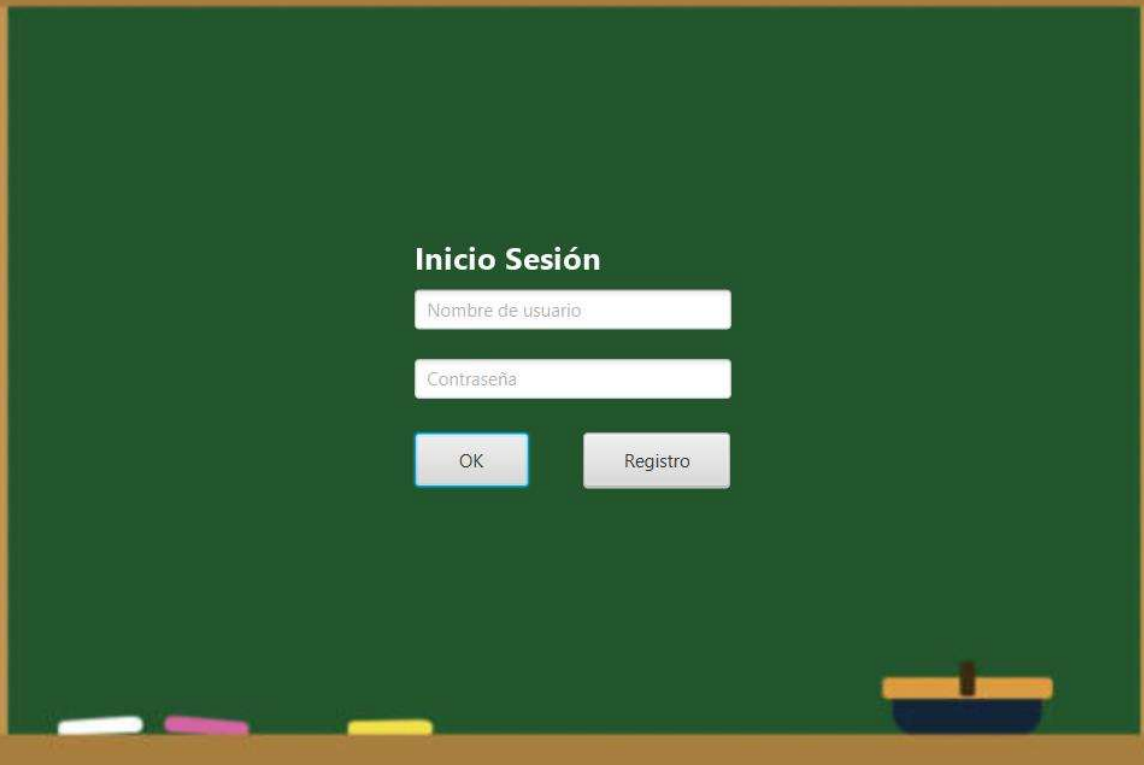
Inicio de Sesión

Inicio Sesión

Nombre de usuario

Contraseña

OK Registro



Gestión Alumnos

Archivo Acerca De

Listado de alumnos:

ID:

ID	Nombre	Apellidos	Edad	Nota Media	Fecha Nacimiento	Nacionalidad
Tabla sin contenido						

Crear Alumno

Detalles del alumno:

Nombre:

Apellidos:

Edad:

Fecha Nacimiento:

Nacionalidad:

Fecha Incorporación:

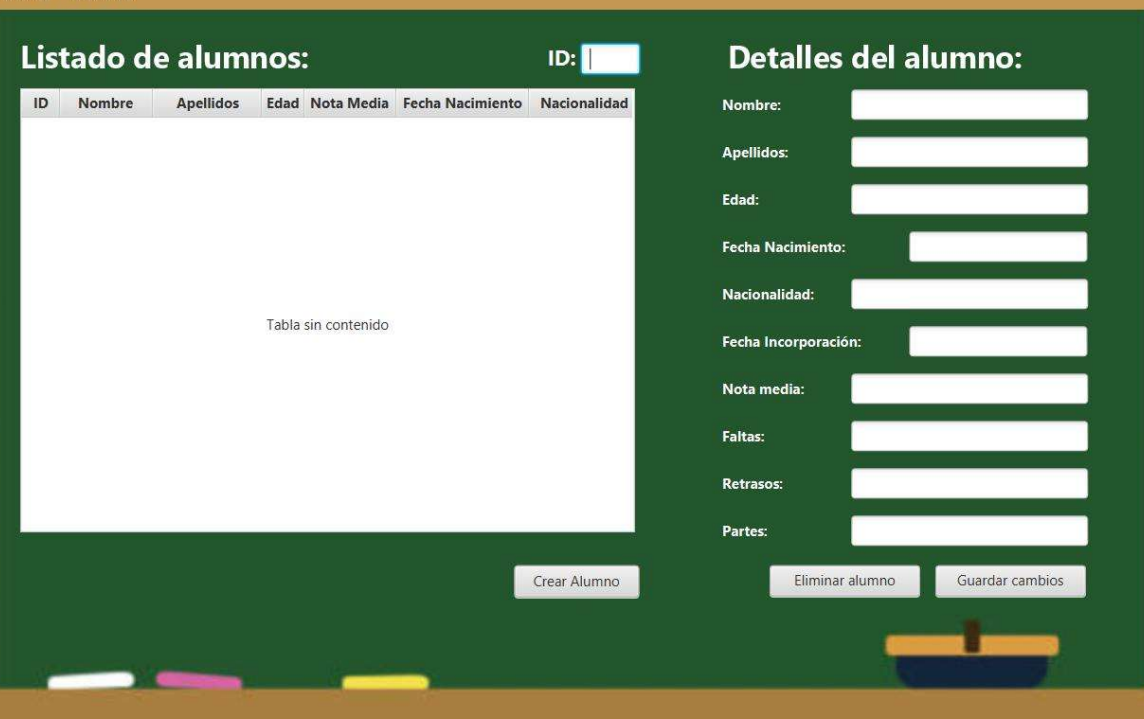
Nota media:

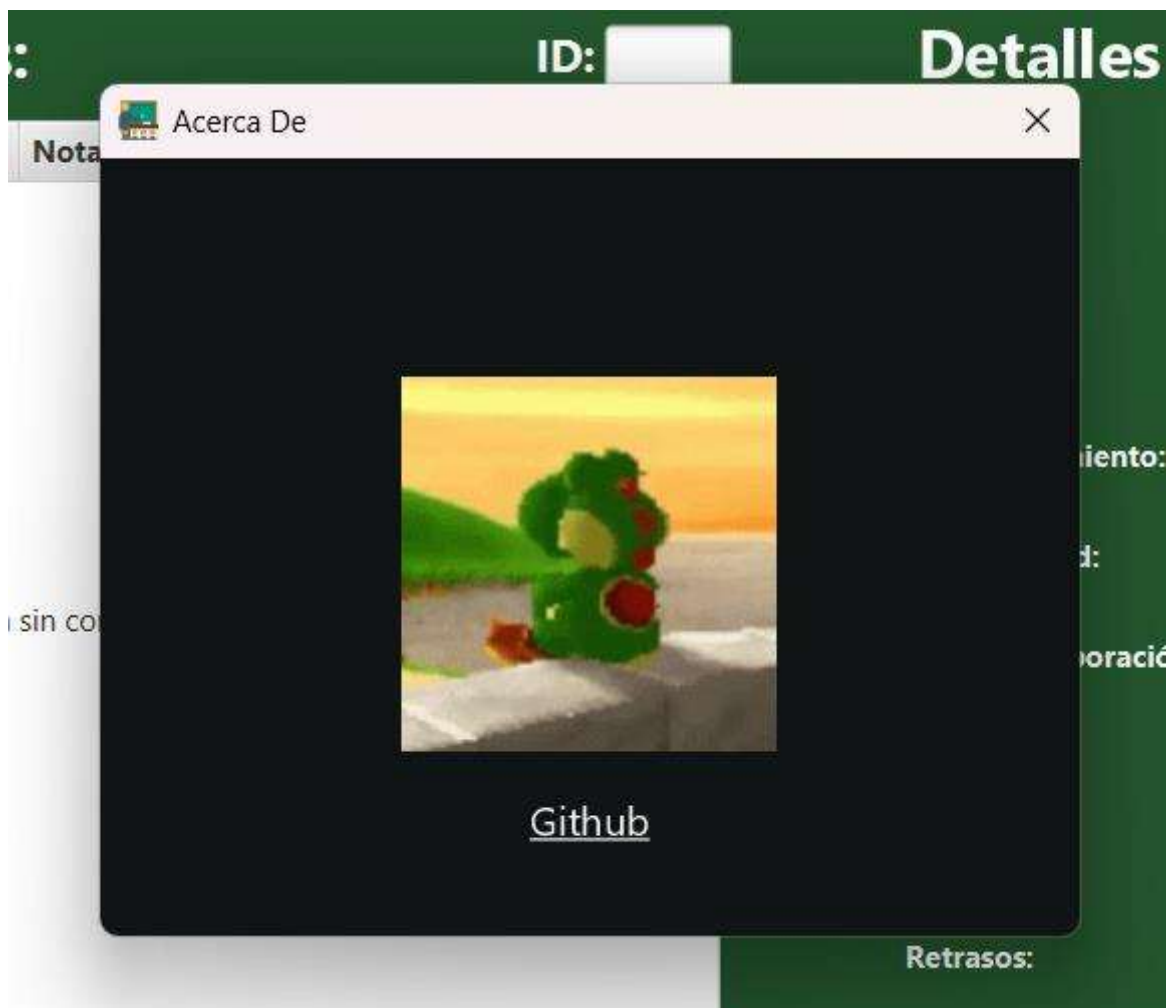
Faltas:

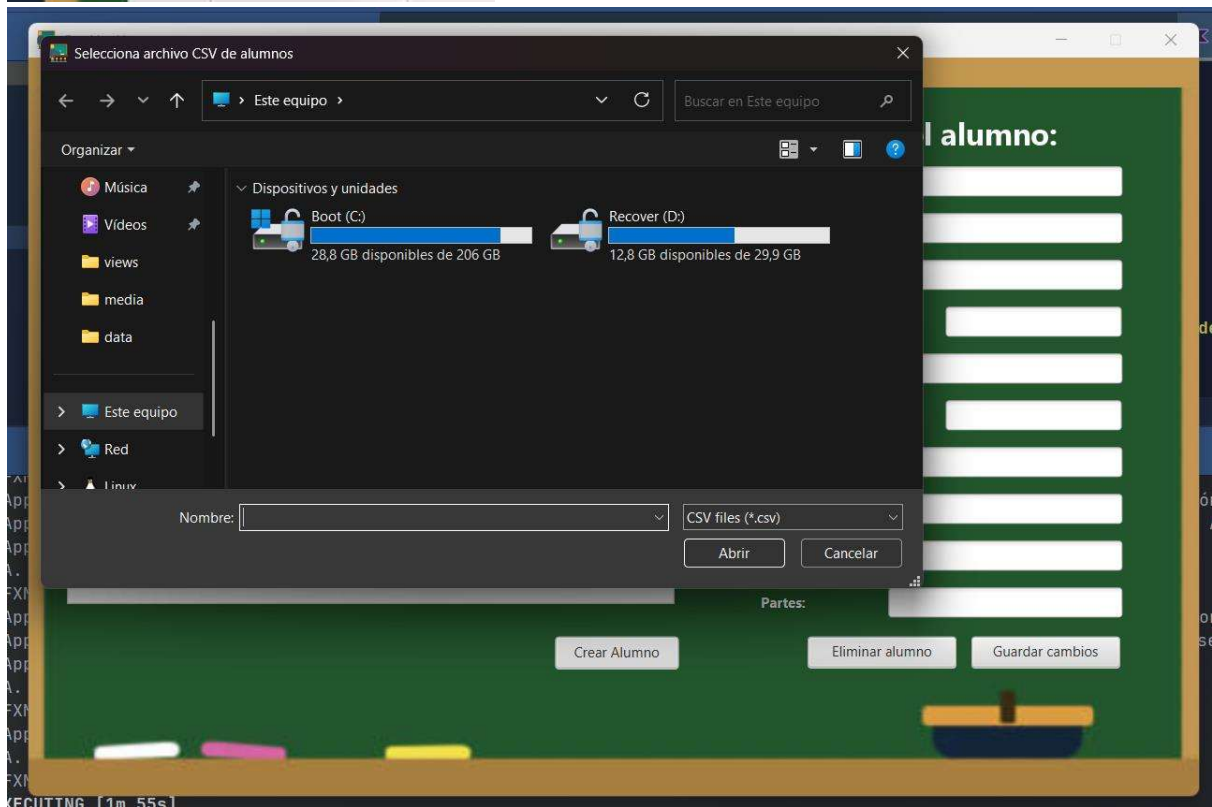
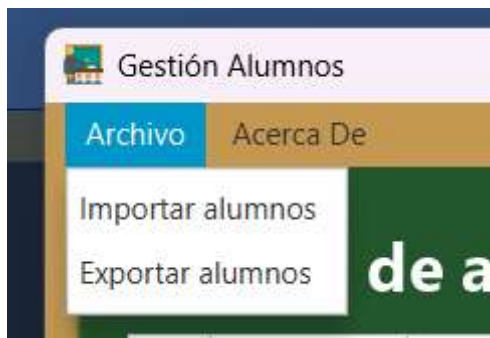
Retrasos:

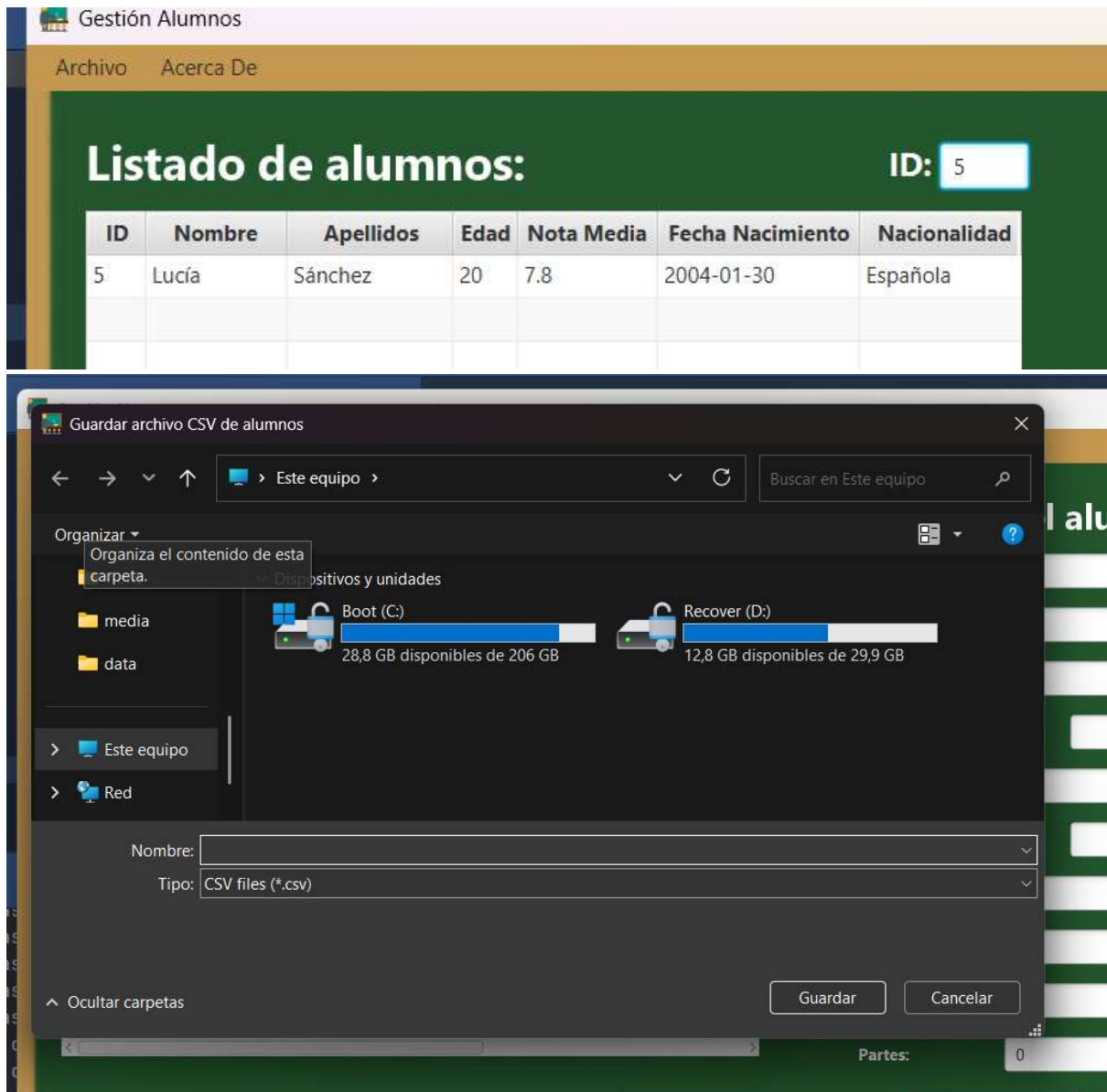
Partes:

Eliminar alumno Guardar cambios









ESTIMACIÓN ECONÓMICA

Para llevar a cabo el cálculo de lo que se estima que podría costar este proyecto hay que tener los siguientes factores en cuenta:

- **Duración del proyecto:** 2-3 semanas.
- **Integrantes:** SOLO 1.
- **Entrega:** además de la aplicación funcional, para la entrega era necesaria esta documentación, un vídeo explicando el funcionamiento de la aplicación y una presentación presencial.
- **Horas de trabajo:** durante la duración del proyecto se han invertido a lo largo de 50h.

En base a esto, y poniendo un salario de **12€/h:**

50x 12€/h = **600€**

A estos 600€ se le podrían haber sumado extras como:

- Software de pago. No añadido ya que el centro nos lo proporciona.
- Costes de edición y producción del vídeo. No añadido ya que se han usado herramientas gratuitas como OBS para la grabación.
- Transporte.

VÍDEO EXPLICATIVO

<https://youtu.be/13kD-ySnhlE>