1º Desarrollo de aplicaciones multiplataforma.

Sistema de Gestión de Calificaciones

Requisitos funcionales, no funcionales y de información.

Leyendo el enunciado del proyecto he llegado a la conclusión de estos Requisitos:

El programa:

El programa en sí tiene unos Requisitos no funcionales para realizar el programa:

- Realizar el programa sobre una base de datos.
- Poder exportar e importar toda la información en JSON.

Sobre los alumnos:

- Requisitos funcionales: Nuestra aplicación debe realizar correctamente un CRUD y modificación sobre la pérdida de la evaluación continua.
- Requisitos no funcionales: Una acotación a la solución podría ser cómo debemos mostrar ordenados a los alumnos cuando mostramos toda la lista de usuarios y como se debe mostrar la fecha de matriculación de los alumnos.
- Requisitos de información: Los atributos de los alumnos serán los siguientes:
    - DNI.
    - Nombre.
    - Apellidos.
    - Email.
    - Teléfono.
    - Pérdida de evaluación continua.
    - Fecha de matriculación.

Para los alumnos hemos añadido un atributo llamado evaluationTest que es un int, con lo que para poder eliminar alumnos, no pueden estar en ninguna prueba de evaluación, así nos encargamos de que sólo eliminemos los alumnos que tengan este atributo a 0.

Sobre las pruebas de evaluación:

- Requisitos funcionales: Debemos poder listarlas, consultarlas, crearlas y eliminarlas, por otro lado, las pruebas de evaluación tienen una categoría las cuales podemos crear y modificar.
- Requisitos no funcionales: A la hora de acotarnos la solución, las pruebas tienen un formato de tiempo específico, no podemos modificar las pruebas de evaluación creadas y las categorías tampoco se pueden modificar y eliminar.
- Requisitos de información: Las pruebas de evaluación tienen los siguientes atributos:
    - Fecha
    - Descripción
    - Categoría
    - Calificaciones(Repositorio)
    - Nota máxima y mínima
    - Porcentaje de aprobados y suspensos

Sobre las calificaciones:

En las calificaciones, tienen en su base de datos dos claves primarias, por qué?

Porque el id de las calificaciones es el id de la prueba de evaluación en la que están añadidos, es decir, que obviamente va a haber varios valores con el mismo id.

La otra clave primaria es el id del alumno, ya que el alumno va a estar en diferentes pruebas de evaluación, pero no puede estar dos veces en la misma calificación.

Sobre la evaluación:

- Requisitos funcionales: Deberemos almacenar una serie de datos sacados de las pruebas de evaluación.
- Requisitos no funcionales: Deberemos exportar la prueba de evaluación elegida por el usuario a un markdown con su formato adecuado y llevar un tiempo en un formato concreto.
- Requisitos de información: Las evaluaciones llevan un repositorio de pruebas de evaluación.

Las fechas, todas han sido puestas en ese momento, ya que al faltar tiempo, preferí invertirlo en avanzar y mejorar cosas que veía más importantes.

Diagrama de Clases

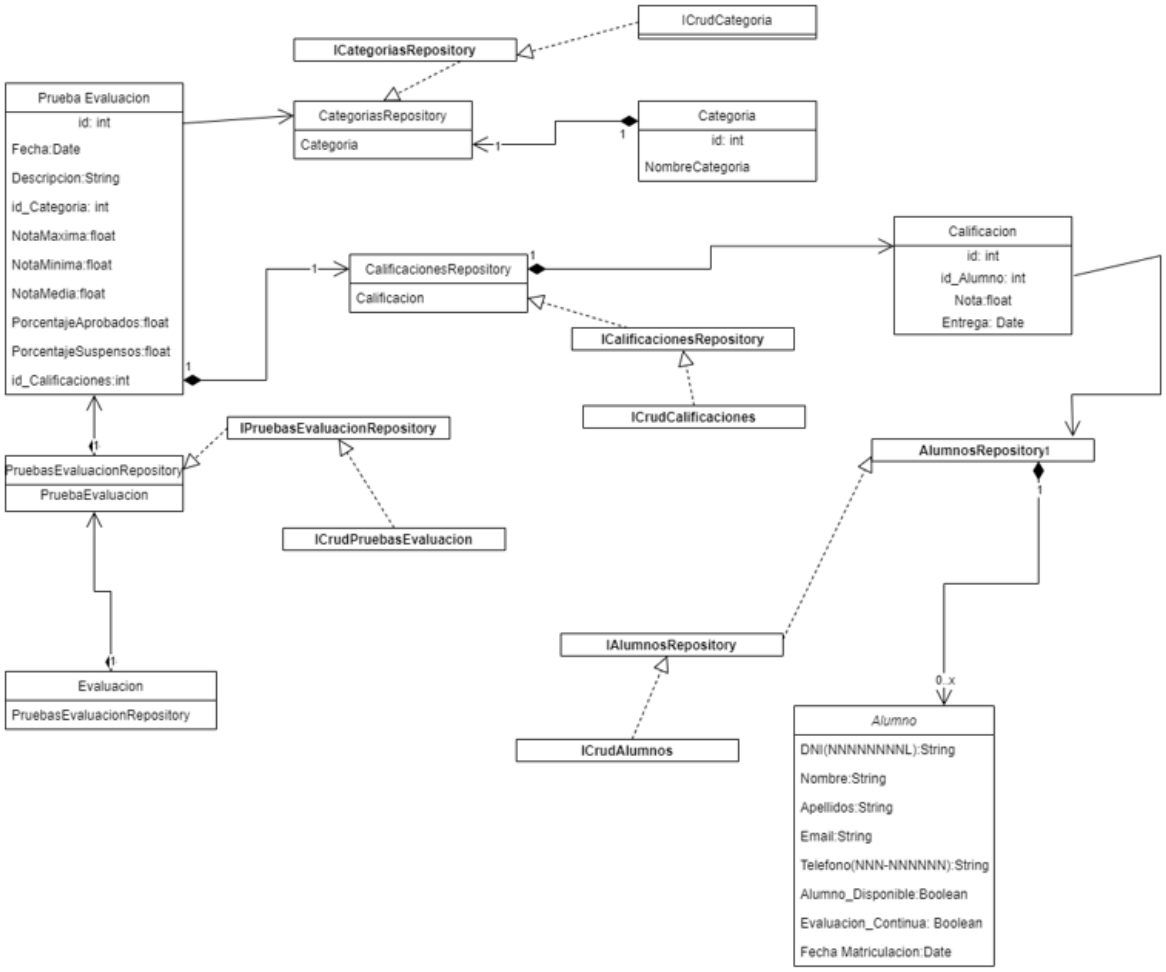
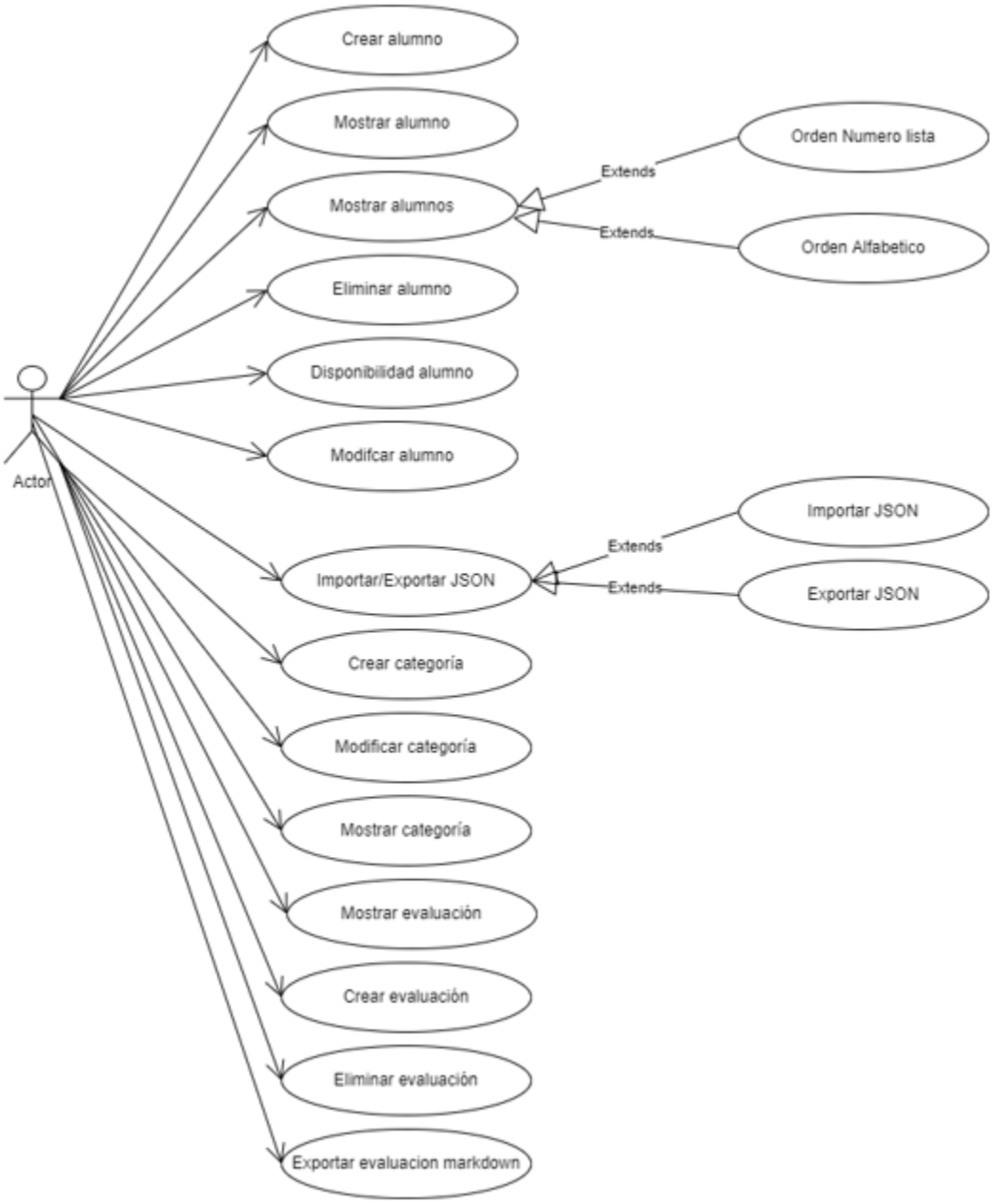


Diagrama de Casos de Uso



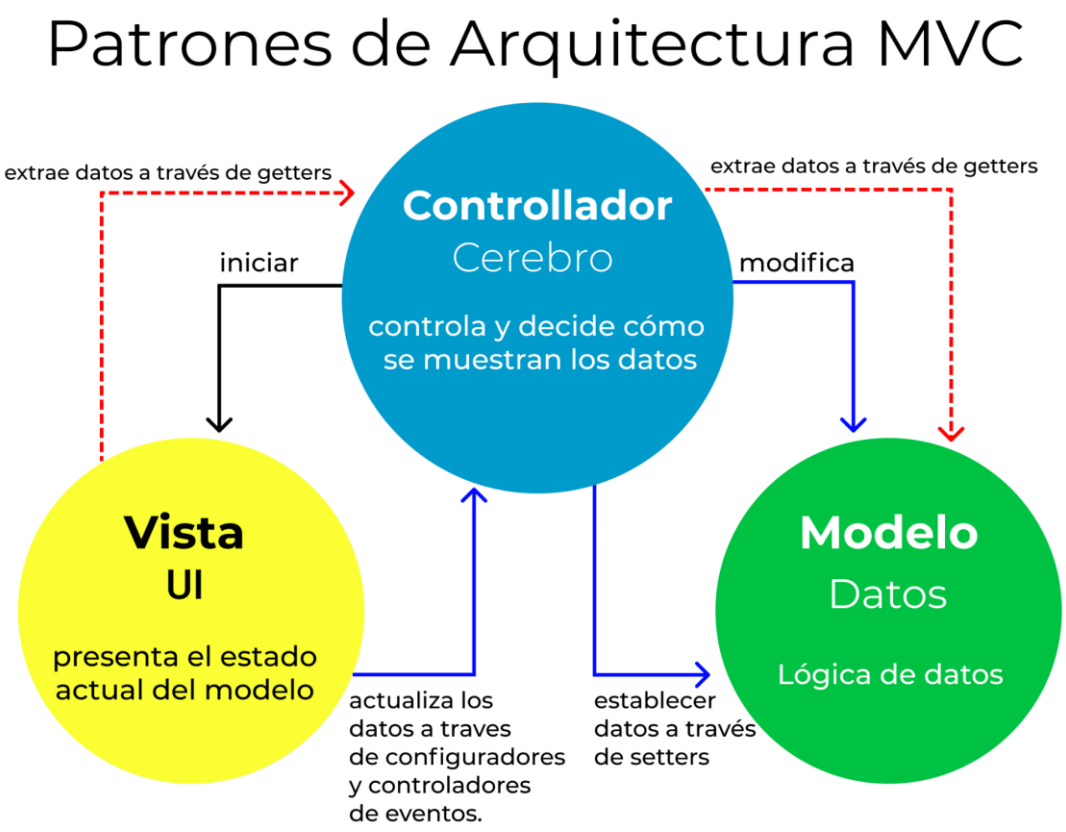
He añadido todos los casos de uso, que he podido sacar del enunciado, aparte he añadido otros como por ejemplo mostrar categoría.

Ya que para poder crear y eliminar deberá de ver qué tiene.

Arquitectura de software aplicada.

La arquitectura software aplicada ha sido el modelo vista controlador.

Es muy práctica para separar la vista, la lógica del programa y sus datos.



Patrones de diseño y ejemplo de cómo se han aplicado.

He utilizado los patrones → Fachada y Singleton.

- Singleton → Lo he aplicado en las clases que quería que siempre me devolviese la misma instancia de la clase, aunque lo he utilizado en muy pocas clases ya que, el patrón Singleton puede llegar a ser un anti patrón.
- Fachada → Lo he aplicado creando una vista, la que ve el usuario.

  El usuario sólo ve la vista, ve lo que nosotros queremos mostrarle, no ve el interior de nuestro programa.

Principios SOLID aplicados y ejemplos.

- Principio de responsabilidad única. Cada método sólo tiene una responsabilidad y va delegando.

  Ejemplo: En el repositorio de alumnos en el método para añadir alumnos, sólo los añade, no los crea y los añade. De crearlo se encarga otra clase, delegamos funciones.

- Principio de abierto/cerrado. Los repositorios implementan interfaces, que están en este principio → abiertas a extensión pero cerradas a modificación.

  Ejemplo, hay unas interfaces “básicas para cada tipo de CRUD”, están son extendidas en otras interfaces a las que luego ya se le pasa a la clase. Por si hubiese otra clase que también necesitase la interfaz CRUD.

- Principio de Segregación de Interfaces, todas las interfaces del programa tienen los métodos justos y necesarios, ya que así cumplimos este principio y no tenemos más métodos de los que necesitamos.

- Principio de inyección de dependencias. En casi todas las clases que se necesitaban dependencias, por no decir todas, están puestas como inyección de dependencias por constructor. Es decir que todas las clases que necesitan otras clases para trabajar, las reciben por su constructor.

Diseño de Pruebas unitarias

En las pruebas, las fechas han sido sacadas ya que da problemas por milisegundos, aparte de esto, comparaciones de la clase en las que aparezca la fecha también han sido quitadas por el mismo problema.

Están hechos y testeados con todo ✓ → Los repositorios

Ejemplo de uso y capturas del mismo.

El ejemplo de uso y sus capturas están en el vídeo.

https://drive.google.com/file/d/1pI-KE_VmjkVGb5vD2DFeR1NVHnaFrpld/view?usp=sharing