DOCUMENTACIÓN DEL SISTEMA SOFTWARE DE GESTIÓN DE ALUMNOS

MEMORIA DE PRÁCTICAS
PARA LA ASIGNATURA INGENIERÍA DEL SOFTWARE
DEL GRADO DE INGENIERÍA INFORMÁTICA
EN LA UNIVERSIDAD DE CÓRDOBA

Grupo 6
Ortega León, Daniel
Pérez Hernández, Julen
Sevilla Molina, Ángel
Diciembre 2018

Índice de figuras

2.1. Diagrama de clases 13
2.2. Diagrama de secuencia del CU-1: Añadir alumno 14
2.3. Diagrama de secuencia del CU-2: Modificar alumno 15
2.4. Diagrama de secuencia del CU-3: Eliminar alumno 15
2.5. Diagrama de secuencia del CU-4: Buscar alumnos 16
2.6. Diagrama de secuencia del CU-5: Mostrar alumnos 17
2.7. Diagrama de secuencia del CU-6: Guardar fichero 18
2.8. Diagrama de secuencia del CU-7: Cargar fichero 18
2.9. Diagrama de secuencia del CU-8: Identificar profesor 19
2.10. Diagrama de secuencia del CU-9: Añadir profesor 19

3.1. Burndown chart 21

Índice de tablas

3.1. Product Backlog 20
3.2. Sprint Backlog 21
3.3. Matriz de correspondencia RF/Casos de Uso 54
3.4. Matriz de correspondencia CU/Clases 54

Índice general

1. Análisis de Requisitos **1**

1.1. Análisis funcional . . . 1

1.1.1. Identificación de actores del Sistema . . . 1

1.1.2. Datos . . . 1

1.1.3. Requisitos . . . 1

1.2. Historias de usuario . . . 4

1.2.1. HU001 Añadir alumno . . . 4

1.2.2. HU002 Modificar alumnos . . . 5

1.2.3. HU003 Eliminar alumnos . . . 5

1.2.4. HU004 Buscar alumnos . . . 5

1.2.5. HU005 Mostrar alumnos . . . 6

1.2.6. HU007 Cargar fichero . . . 6

1.2.7. HU008 Identificar Profesor . . . 6

1.2.8. HU009 Añadir Profesor . . . 7

1.3. Casos de uso . . . 7

1.3.1. CU001 Añadir alumno . . . 7

1.3.2. CU002 Modificar alumno . . . 8

1.3.3. CU003 Eliminar alumno . . . 9

1.3.4. CU004 Buscar alumnos . . . 9

1.3.5. CU005 Mostrar alumnos . . . 10

1.3.6. CU006 Guardar fichero . . . 10

1.3.7. CU007 Cargar fichero . . . 11

1.3.8. CU008 Identificar profesor . . . 11

1.3.9. CU009 Añadir profesor . . . 12

2. Diseño del sistema **13**

2.1. Diagrama de clases . . . 13

2.2. Diagramas de secuencia . . . 14

2.2.1. Diagrama de secuencia. Añadir alumno 14
2.2.2. Diagrama de secuencia. Modificar alumno 15
2.2.3. Diagrama de secuencia. Eliminar alumno 15
2.2.4. Diagrama de secuencia. Buscar alumnos 16
2.2.5. Diagrama de secuencia. Mostrar alumnos 17
2.2.6. Diagrama de secuencia. Guardar fichero 18
2.2.7. Diagrama de secuencia. Cargar fichero 18
2.2.8. Diagrama de secuencia. Identificar profesor 19
2.2.9. Diagrama de secuencia. Añadir profesor 19

3. Implementación y pruebas **20**
3.1. SCRUM 20
3.1.1. Product Backlog 20
3.1.2. Sprint Backlog 21
3.1.3. Burndown Chart 21
3.2. Código 22
3.2.1. alumno 22
3.2.2. baseDatos 26
3.2.3. profesor 34
3.2.4. is 39
3.2.5. ppal-is 40
3.3. Técnicas de validación 54
3.3.1. Matriz de correspondencia RF/Casos de Uso 54
3.3.2. Matriz de correspondencia CU/Clases 54

Capítulo 1

Análisis de Requisitos

1.1. Análisis funcional

En este apartado se incluye la parte funcional, estructural y el comportamiento de la aplicación. Para ello se hace uso de UML (Unified Modeling Language - Lenguaje Unificado de Modelado), un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

1.1.1. Identificación de actores del Sistema

Un actor, en UML, es aquel agente externo (un usuario, un dispositivo hardware o incluso otro sistema) que interactúa con el sistema.

Para este sistema se puede diferenciar el siguiente actor:

- **Profesor:** Profesor: El profesor es el único actor del sistema, y es el que gestiona y opera con los datos de los alumnos. El profesor puede ser coordinador o ayudante.

1.1.2. Datos

El sistema debe permitir un registro histórico de alumnos que gestione la siguiente información: dni, nombre, apellidos, teléfono, e-mail, dirección postal, curso más alto matriculado, fecha de nacimiento, número de equipo, y si es líder o no del equipo.

1.1.3. Requisitos

Requisitos funcionales

Los requisitos funcionales son características que debe cumplir el sistema y que expresan la funcionalidad y/o el comportamiento específico que debe tener el sistema ante determinadas situaciones. Estos se codifican con las siglas *RF*, seguidas de un guión y del número de requisito.

- **RF-1:** El sistema permite que el profesor pueda añadir alumnos en la base de datos.
  - **RF-1.1:** Todos los datos son obligatorios excepto el número de grupo y liderazgo.
- **RF-2:** El sistema permite que el profesor pueda modificar la información de los alumnos ya registrados en la base de datos.
- **RF-3:** El sistema permite que el profesor pueda eliminar alumnos de la base de datos.
- **RF-4:** El sistema permite que el profesor pueda buscar alumnos.
  - **RF-4.1:** La búsqueda de alumnos debe poder filtrarse a partir del dni, de los apellidos o del equipo al que pertenece.
  - **RF-4.2:** La búsqueda debe devolver todas las coincidencias obtenidas.
- **RF-5:** El sistema permite que el profesor pueda acceder a la información de uno o varios alumnos.
  - **RF-5.1:** El profesor puede ordenar el mostrado en orden alfabético, por dni, o por curso más alto matriculado.
  - **RF-5.2:** El orden puede ser ascendente o descendente.
- **RF-6:** El sistema permite que el profesor pueda guardar los datos en un fichero de seguridad.
- **RF-7:** El sistema permite que el profesor pueda cargar los datos a partir de un fichero de seguridad.
- **RF-8:** El sistema debe incorporar un proceso de identificación mediante nombre y contraseña para que los profesores puedan hacer uso de las funcionalidades del sistema.
- **RF-9:** El sistema permite que el profesor coordinador pueda añadir ayudantes en el sistema, aportando nombre de profesor y contraseña.

Requisitos no funcionales

Los requisitos no funcionales son característicos del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que debe cumplir el sistema y que indican las restricciones del mismo. Estos se codifican con las siglas *RNF*, seguidas de un guión y del número de requisito.

- **RNF-1:** El lenguaje de programación es C++.
- **RNF-2:** El lenguaje de documentación será Markdown.
- **RNF-3:** Si se hace uso de un IDE, éste será Eclipse.

- **RNF-4:** El proyecto se realizará haciendo uso del sistema de control de versiones Git y la plataforma GitHub para el almacenamiento del repositorio de forma remota.
- **RNF-5:** El historial de cambios queda guardado en las cuentas de Git por lo que la evaluación será incremental y no servirá subir los ficheros de un día para otro en Git.
- **RNF-6:** El sistema debe funcionar en Linux.
- **RNF-7:** El sistema deberá ser rápido ofreciendo los resultados de consulta en un tiempo inferior a dos segundos. Del mismo modo los procesos de actualización de un registro de la base de datos deben ser realizados en menos de un segundo.
- **RNF-7:** Se utilizará la metodología UML durante la fase de desarrollo.
- **RNF-8:** El sistema debe permitir almacenar información histórica.
- **RNF-9:** Se debe tener un alto control de errores, evitando que el profesor introduzca datos inconsistentes, y permitir al profesor la modificación o posibilidad de subsanar otros errores.
- **RNF-10:** El formato de salida de los listados de alumnos debe ser html o Markdown.
- **RNF-11:** El sistema solo puede admitir como máximo a 150 alumnos.
- **RNF-12:** Debe poderse cargar o guardar los datos del registro histórico de alumnos en ficheros binarios.
- **RNF-13:** Todos los datos de un alumno son necesarios, excepto el número de grupo y el liderazgo.
- **RNF-14:** Un alumno solo puede ser líder de un grupo si forma parte de un grupo.
- **RNF-15:** Como máximo puede haber un solo líder por grupo.
- **RNF-16:** En el mostrado, aquellos alumnos que sean líderes de un grupo deben verse remarcados.
- **RNF-17:** Los atributos identificadores de un alumno son el dni, el e-mail, y el agregado que forma el número de grupo y la condición de líder.
- **RNF-18:** La interfaz del sistema deberá ser a través de línea de comandos.
- **RNF-19:** El sistema debe proporcionar mensajes de error que sean informativos y orientados al usuario final.
- **RNF-20:** El sistema debe ser capaz de manejar caracteres del alfabeto latino.

- **RNF-21:** Solo puede existir un coordinador en el sistema, pero este puede tener varios ayudantes. Todo nuevo usuario que el coordinador cree será de tipo ayudante.
- **RNF-22:** Solo los profesores pueden crear y cargar copias de seguridad.
- **RNF-23:** El sistema debe incorporar un fichero binario con las credenciales de los usuarios para la verificacion de usuarios en el sistema.
- **RNF-24:** Un usuario no identificado no puede hacer uso de las funcionalidades del sistema.

1.2. Historias de usuario

1.2.1. HU-1 Añadir alumno

(ANVERSO)
ID: 001 **Añadir alumno**
Como usuario quiero poder añadir alumnos al listado.
Prioridad: 3
(REVERSO)

- Quiero poder añadir alumnos al listado.
- Como máximo quiero añadir 150 alumnos
- Tengo que añadir todos los datos del alumno necesariamente excepto el numero de grupo y el liderazgo
- Quiero que el programa me reconozca caracteres del alfabeto latino cuando añada un alumno, tales como la "ñ"
- Como máximo puede haber un solo líder por grupo
- Un alumno solo puede ser líder de un grupo si forma parte del mismo.

1.2.2. HU-2 Modificar alumnos

(ANVERSO)

ID: 002 **Modificar alumnos**

Como usuario quiero poder modificar alumnos en el caso de algún fallo o cambio de datos.

Prioridad: 3

(REVERSO)

- Quiero poder modificar alumnos.
- Quiero que la modificación de la base de datos se realice en menos de 1 segundo.
- Quiero que además que el programa tenga un control de errores evitando asi datos inconsistentes.

1.2.3. HU-3 Eliminar alumnos

(ANVERSO)

ID: 003 **Eliminar alumnos**

Como usuario quiero poder eliminar alumnos del listado.

Prioridad: 3

(REVERSO)

- Quiero poder eliminar alumnos del listado.
- La operación de eliminado debe estar realizada en menos de un segundo.

1.2.4. HU-4 Buscar alumnos

(ANVERSO)

ID: 004 **Buscar alumnos**

Como usuario quiero poder buscar y seleccionar alumnos entre el listado para realizar operaciones sobre ellas.

Prioridad: 3

(REVERSO)

- Quiero poder buscar alumnos dentro del listado.
- Se debe poder filtrar a partir de atributos como el DNI, los apellidos, o el equipo al que pertenece.
- La búsqueda debe devolver todas las coincidencias obtenidas.
- En caso de no aplicar ningún filtro se devolverá todo el listado de alumnos.
- En caso de no encontrar ninguna coincidencia, se devolverá un listado vacío.

1.2.5. HU-5 Mostrar alumnos

(ANVERSO)

ID: 005 **Mostrar alumnos**

Como usuario quiero poder visualizar un listado de alumnos del sistema para poder ver su información.

Prioridad: 3

(REVERSO)

- Quiero poder visualizar todos los datos de los alumnos.
- Se debe mostrar claramente si el alumno es líder del grupo o no.
- Se podrá ordenar, de forma ascendente o descendente, el mostrado en orden alfabético, por DNI, o por curso más alto matriculado.
- El mostrado se realizará mediante el volcado de los datos de los alumnos a un fichero de salida.
- La muestra se realizará siempre y para todos los alumnos seleccionados en la búsqueda, sea un listado vacío, unitario o de varios alumnos.

1.2.6. HU-7 Cargar fichero

(ANVERSO)

ID: 007 **Cargar fichero**

Como usuario me interesa poder cargar los datos desde un almacenamiento externo.

Prioridad: 3

(REVERSO)

- Quiero poder cargar un registro histórico de alumnos.
- Quiero que el fichero a cargar sea un fichero binario.

1.2.7. HU-8 Identificar Profesor

(ANVERSO)

ID: 008 **Identificar Profesor**

Como usuario quiero poder acceder al sistema.

Prioridad: 3

(REVERSO)

- Quiero poder acceder al sistema.
- Quiero que el sistema me informe de errores si no coinciden las credenciales introducidas.

1.2.8. HU-9 Añadir Profesor

(ANVERSO)
ID: 009 **Añadir Profesor**
Como usuario quiero poder añadir otros profesores al sistema.
Prioridad: 3
(REVERSO)

- Quiero poder registrar otros usuarios al sistema.
- Quiero que el sistema me informe de errores si existen profesores coincidentes.

1.3. Casos de uso

1.3.1. CU-1 Añadir alumno

Añadir alumno
ID: 001
Breve descripción: El sistema añade un alumno.
Actores principales: Usuario
Actores secundarios: Alumno
Precondiciones:

1. El alumno tiene que existir.

Flujo principal:

1. El caso de uso empieza cuando el usuario necesita añadir un alumno.
2. El usuario introduce los datos del alumno.

Postcondiciones

- La base de datos recibe la información del alumno.

Flujos alternativos

- Si no se introduce toda la información necesaria, el sistema no añadirá al alumno y saldrá un mensaje de error.

1.3.2. CU-2 Modificar alumno

Modificar alumno
ID: 002
Breve descripción: El sistema modifica un alumno.
Actores principales: Usuario
Actores secundarios: Alumno
Precondiciones:

1. El alumno tiene que estar en el sistema.

Flujo principal:

1. El caso de uso empieza cuando el usuario necesita modificar un alumno.
2. El usuario introduce los datos del alumno que quiere modificar mediante previa búsqueda.

Postcondiciones

- La base de datos recibe la nueva información del alumno y la sustituye por la que había anteriormente.

Flujos alternativos

- Ninguno.

1.3.3. CU-3 Eliminar alumno

Eliminar alumno
ID: 003
Breve descripción: El sistema elimina un alumno.
Actores principales: Usuario
Actores secundarios: Alumno
Precondiciones:

1. El alumno tiene que existir.

Flujo principal:

1. El caso de uso empieza cuando el usuario necesita eliminar un alumno.
2. El usuario mediante previa busqueda elimina a un alumno.

Postcondiciones

- El sistema elimina los datos de dicho alumno.

Flujos alternativos

- Si el alumno que se quiere eliminar no existe, el sistema muestra un mensaje de error.

1.3.4. CU-4 Buscar alumnos

Buscar alumnos
ID: 004
Breve descripción: El sistema busca un listado de alumnos.
Actores principales: Usuario
Actores secundarios: Alumno
Precondiciones:
NInguna.
Flujo principal:

1. El caso de uso empieza cuando el sistema necesita buscar un listado de alumnos.
2. El sistema recoge los criterios del filtrado de búsqueda.

Postcondiciones

- El sistema devuelve el listado de alumnos coincidentes.

Flujos alternativos
Ninguno.

1.3.5. CU-5 Mostrar alumnos

Mostrar alumnos
ID: 005
Breve descripción: El sistema muestra un listado de alumnos.
Actores principales: Usuario
Actores secundarios: Alumno
Precondiciones:
NInguna.
Flujo principal:

1. El caso de uso empieza cuando el sistema necesita mostrar un listado de alumnos.
2. El sistema recoge los datos de los alumnos.
3. El sistema recoge los criterios del mostrado.

Postcondiciones

- El sistema genera un fichero con los datos de los alumnos.

Flujos alternativos
Ninguno.

1.3.6. CU-6 Guardar fichero

Guardar fichero
ID: 006
Breve descripción: El sistema guarda una copia de la información de los alumnos.
Actores principales: Usuario
Actores secundarios: Alumno
Precondiciones:
Ninguna.
Flujo principal:

1. El caso de uso empieza cuando el usuario solicita el guardado o antes de cerrar el sistema.
2. El sistema recoge las restricciones para el formato de guardado.

Postcondiciones

- El sistema guarda la información en un fichero externo.

Flujos alternativos
Ninguno.

1.3.7. CU-7 Cargar fichero

Cargar fichero
ID: 007
Breve descripción: El sistema carga la información de los alumnos.
Actores principales: Usuario
Actores secundarios: Alumno
Precondiciones:
Ninguna.
Flujo principal:

1. El caso de uso empieza cuando el usuario solicita la carga de los datos.
2. El sistema recoge las restricciones para el formato de guardado.

Postcondiciones

- El sistema carga la copia de seguridad almacenada.

Flujos alternativos
Ninguno.

1.3.8. CU-8 Identificar profesor

Identificar profesor
ID: 008
Breve descripción: El sistema realiza el inicio de sesión de un profesor.
Actores principales: Usuario
Actores secundarios: Usuario
Precondiciones:

1. Las credenciales de los profesores deben existir.

Flujo principal:

1. El caso de uso empieza cuando el usuario accede al sistema.
2. El usuario introduce sus credenciales de usuario.

Postcondiciones

- El sistema recibe los datos del profesor coordinador.

Flujos alternativos

- Si no se encuentran coincidencias de credenciales, el sistema no identificará al profesor y saldrá un mensaje de error.

1.3.9. CU-9 Añadir profesor

Añadir profesor
ID: 009
Breve descripción: El sistema añade un profesor.
Actores principales: Usuario
Actores secundarios: Usuario
Precondiciones:

1. El profesor no puede existir.

Flujo principal:

1. El caso de uso empieza cuando el profesor coordinador solicita añadir un profesor ayudante.
2. El profesor coordinador introduce los datos del profesor ayudante.

Postcondiciones

- El sistema recibe la información del profesor ayudante.

Flujos alternativos

- Si no se introduce una credencial válida, el sistema no añadirá al profesor ayudante y saldrá un mensaje de error.

Capítulo 2

Diseño del sistema

2.1. Diagrama de clases

Se procede a representar el diagrama de clases indicando nombre de clases y tipos de relaciones. Este diagrama de clases será representado en la figura 2.1.

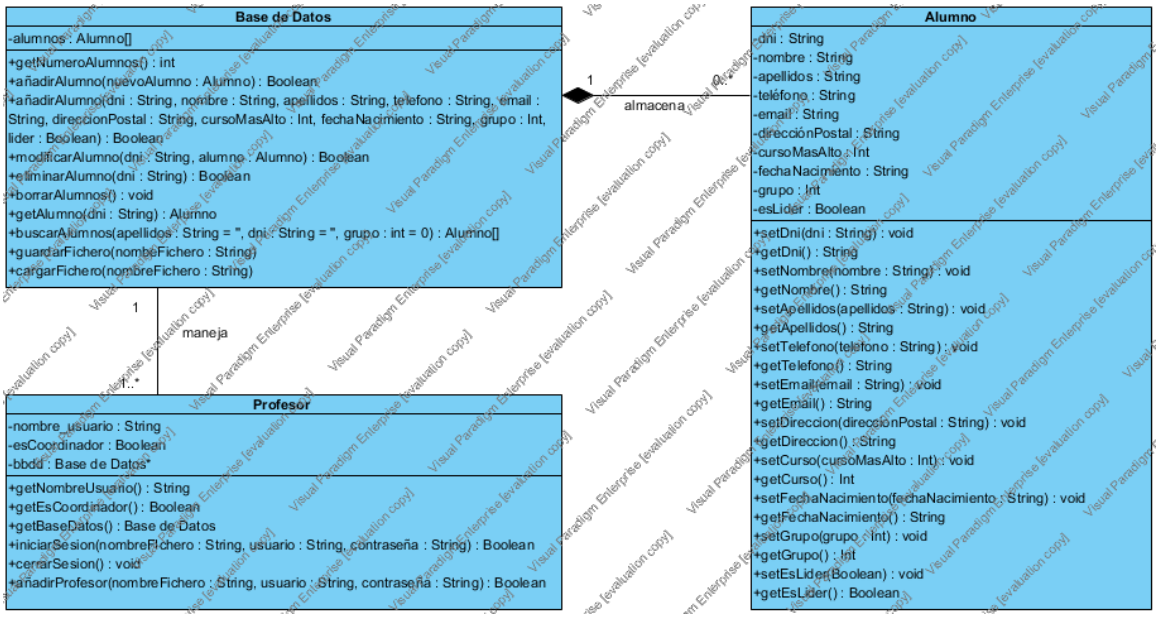

Figura 2.1: Diagrama de clases

2.2. Diagramas de secuencia

El análisis de comportamiento se basará en los casos de uso previamente especificados y para llevarlo a cabo se seguirá la metodología UML utilizando para ello la técnica de diagramas de secuencias.

2.2.1. Diagrama de secuencia. Añadir alumno

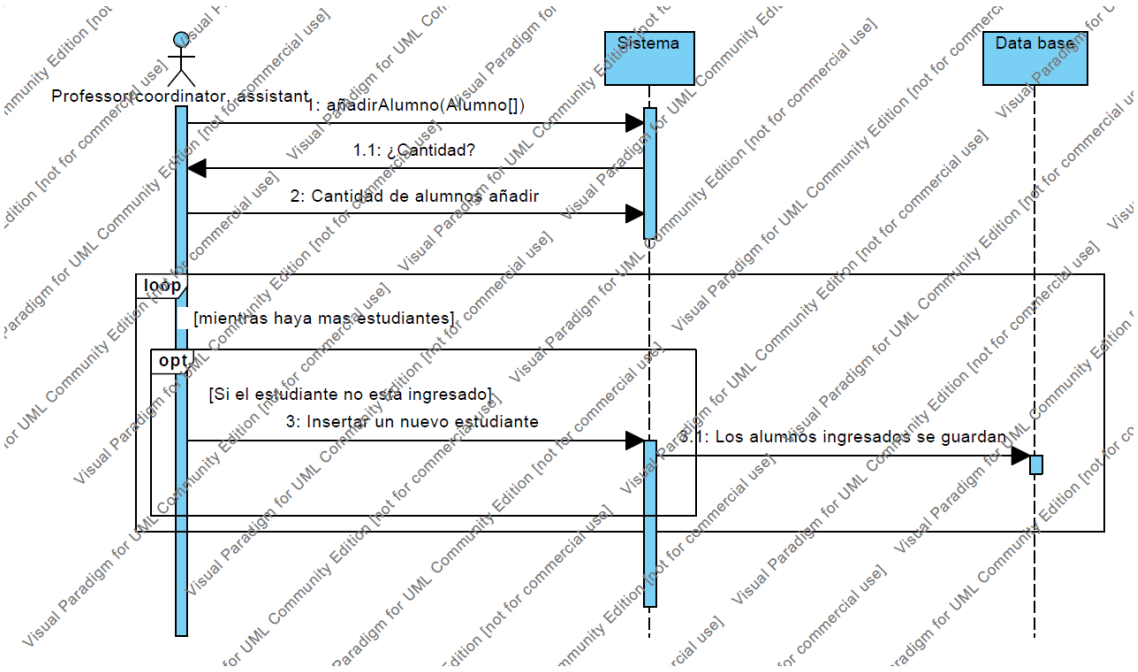

Figura 2.2: Diagrama de secuencia del CU-1: Añadir alumno

2.2.2. Diagrama de secuencia. Modificar alumno

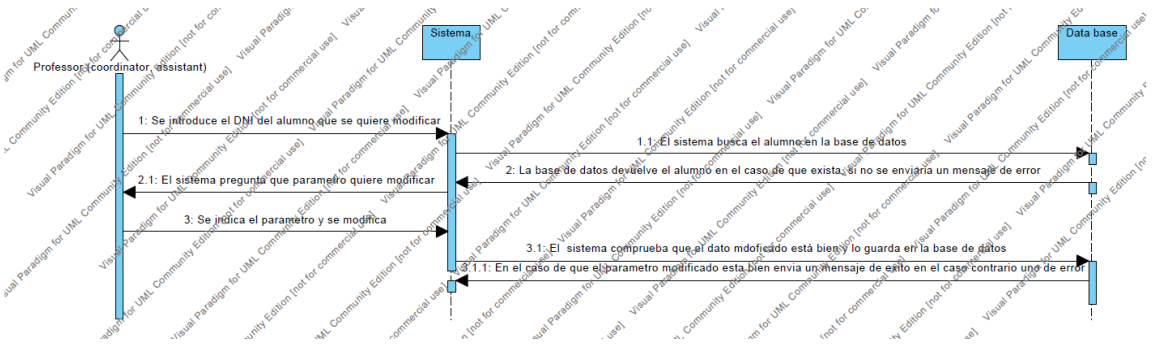

Figura 2.3: Diagrama de secuencia del CU-2: Modificar alumno

2.2.3. Diagrama de secuencia. Eliminar alumno

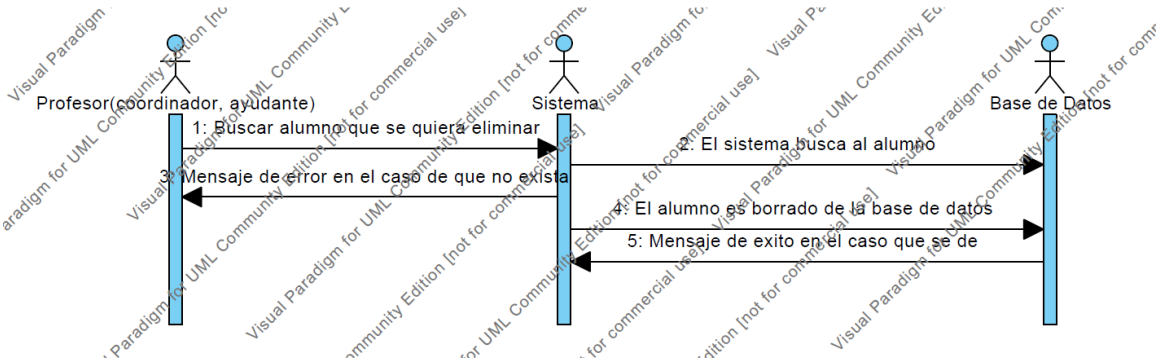

Figura 2.4: Diagrama de secuencia del CU-3: Eliminar alumno

2.2.4. Diagrama de secuencia. Buscar alumnos

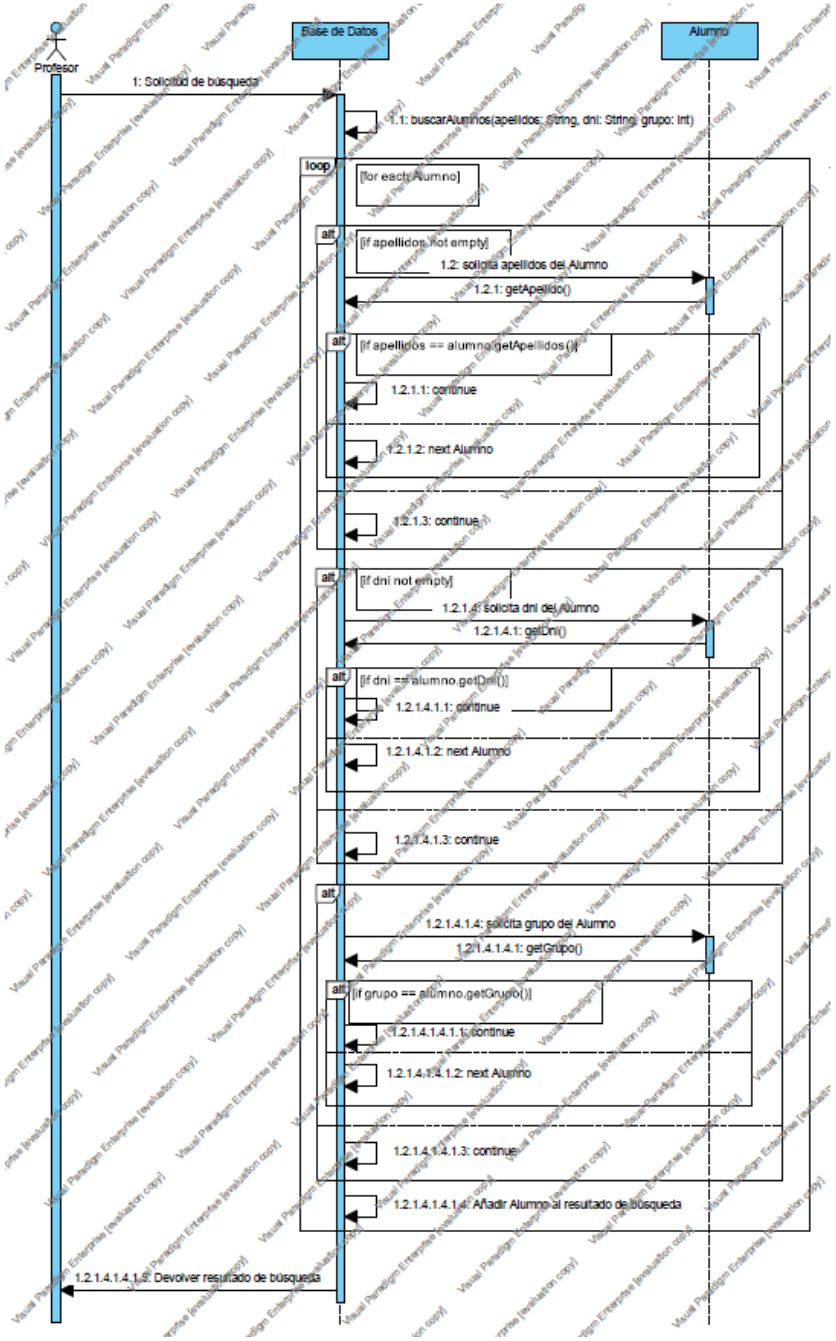

Figura 2.5: Diagrama de secuencia del CU-4: Buscar alumnos

2.2.5. Diagrama de secuencia. Mostrar alumnos

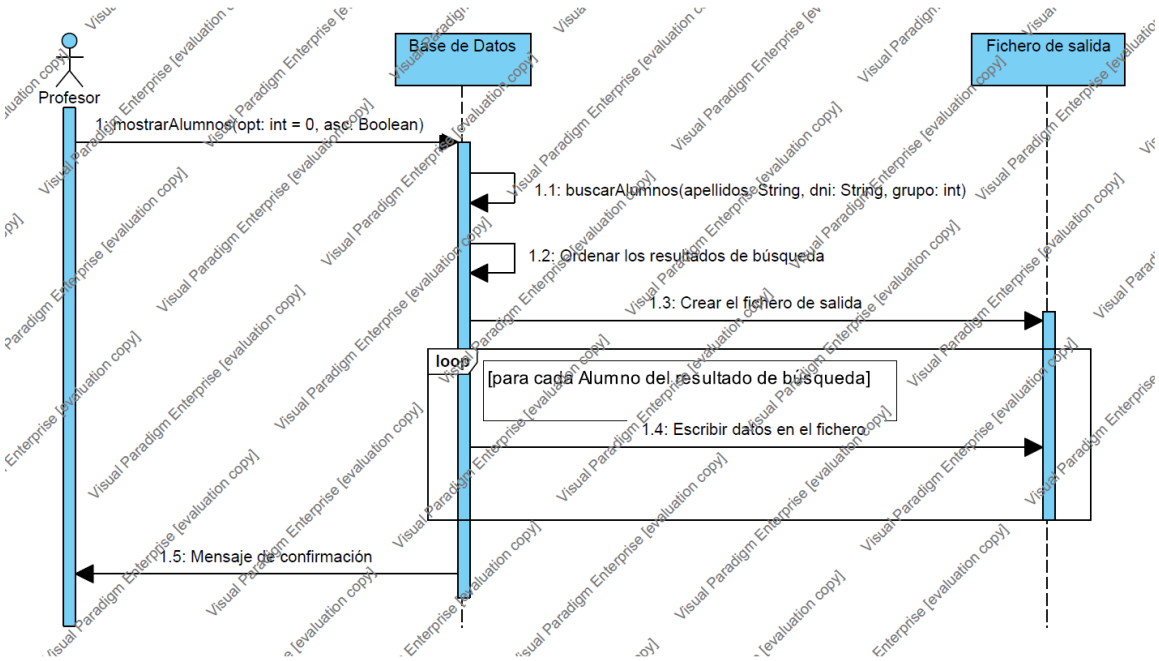

Figura 2.6: Diagrama de secuencia del CU-5: Mostrar alumnos

2.2.6. Diagrama de secuencia. Guardar fichero

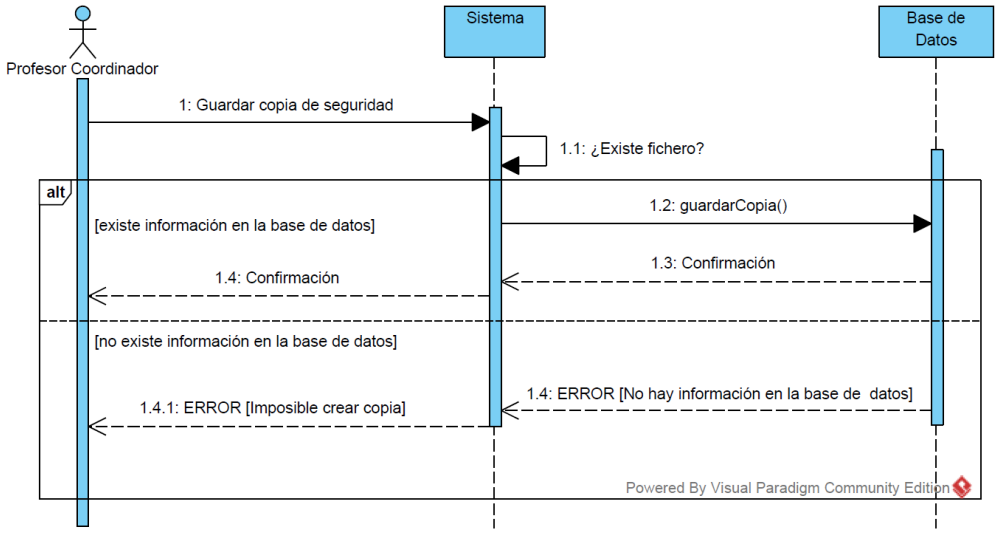

Figura 2.7: Diagrama de secuencia del CU-6: Guardar fichero

2.2.7. Diagrama de secuencia. Cargar fichero

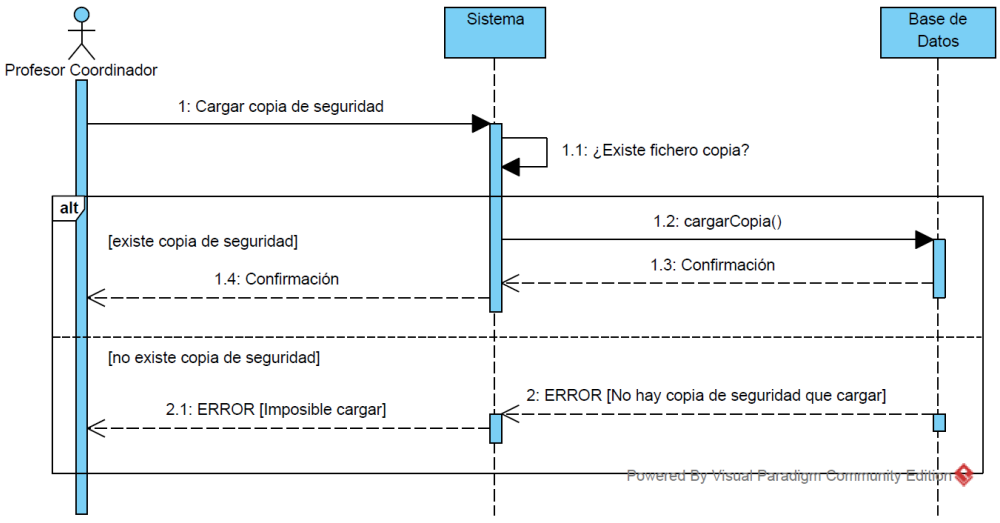

Figura 2.8: Diagrama de secuencia del CU-7: Cargar fichero

2.2.8. Diagrama de secuencia. Identificar profesor

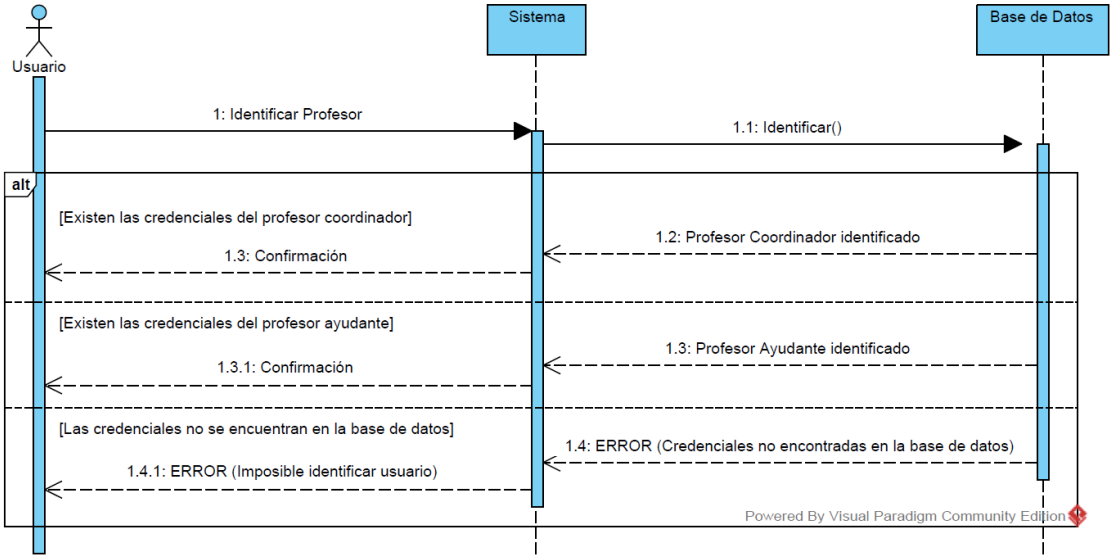

Figura 2.9: Diagrama de secuencia del CU-8: Identificar profesor

2.2.9. Diagrama de secuencia. Añadir profesor

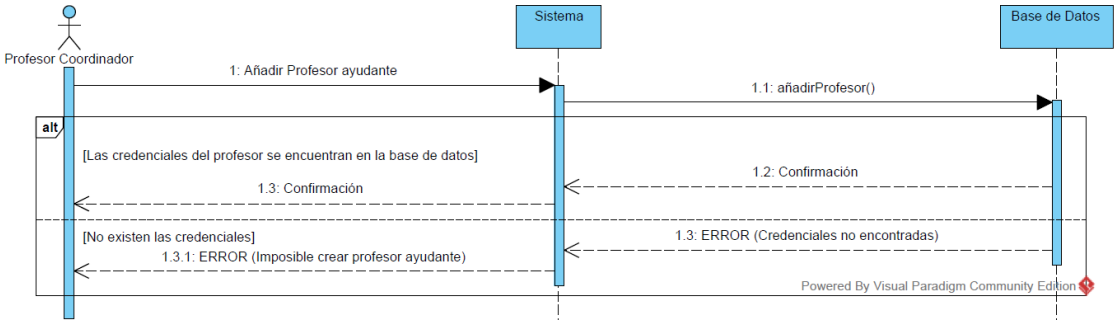

Figura 2.10: Diagrama de secuencia del CU-9: Añadir profesor

Capítulo 3

Implementación y pruebas

3.1. SCRUM

3.1.1. Product Backlog

Historia de Usuario	Estimación	Prioridad
Como usuario quiero poder identificarme en el sistema.	2	1
Como usuario quiero poder añadir otros usuarios en el sistema.	2	2
Como usuario quiero poder añadir alumnos al listado.	1	3
Como usuario quiero poder buscar y seleccionar alumnos entre el listado.	1	4
Como usuario quiero poder visualizar un listado de alumnos.	3	5
Como usuario me interesa poder guardar los datos almacenados externamente.	2	6
Como usuario me interesa poder cargar los datos almacenados externamente.	2	7
Como usuario quiero poder modificar alumnos del listado.	1	8
Como usuario quiero poder eliminar alumnos del listado.	1	9

Tabla 3.1: Product Backlog

3.1.2. Sprint Backlog

Tarea	Quién	Estado	Semana 1	Semana 2	Semana 3
HU008.	i72pehej	Terminada	0.5	0.5	0
HU009.	i72pehej	Terminada	0.5	0.5	0
HU001.	i72orled	Terminada	0.5	0	0
HU004.	i42semoa	Terminada	0.5	0	0
HU005.	i42semoa	Terminada	4	2	0
HU006.	i72pehej	Terminada	2	1	0
HU007.	i72pehej	Terminada	2	1	0
HU002.	i72orled	Terminada	1	0.5	0
HU003.	i72orled	Terminada	1	0	0

Tabla 3.2: Sprint Backlog

3.1.3. Burndown Chart

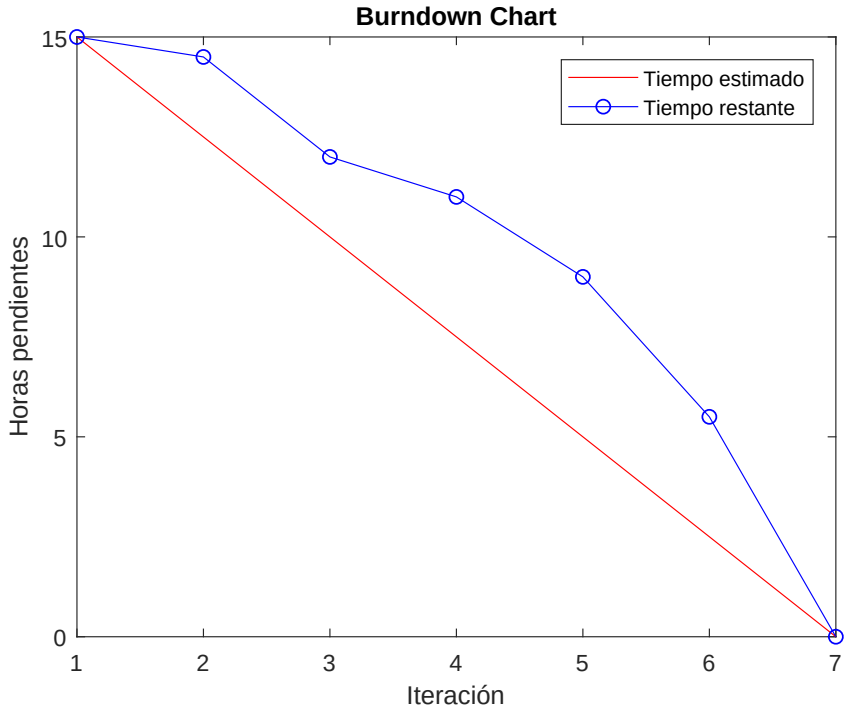

Figura 3.1: Burndown chart

3.2. Código

3.2.1. alumno

```
/**
 * @file alumno.hpp
 * @brief Declaración de la clase alumno.
 * @date 27-11-2018
 *
 * Este fichero forma parte del proyecto realizado para la gestión de alumnos,
 * para la asignatura Ingeniería del Software de la Universidad de Córdoba.
 */

#ifndef ALUMNO_HPP
#define ALUMNO_HPP

#include <string>
#include <assert.h>
#include <ctype.h>

/**
 * @brief Almacena la información de un alumno de la Universidad de Córdoba.
 */
class alumno {
  public:
    alumno();
    alumno(std::string dni, std::string nombre, std::string apellidos,
           std::string telefono, std::string email, std::string direccion,
           std::string fecha_nacimiento, unsigned curso, unsigned grupo = 0,
           bool esLider = false);
    std::string getDni() const { return dni_; }
    std::string getNombre() const { return nombre_; }
    std::string getApellidos() const { return apellidos_; }
    std::string getTelefono() const { return telefono_; }
    std::string getEmail() const { return email_; }
    std::string getDireccion() const { return direccion_; }
    std::string getFechaNacimiento() const { return fechaNacimiento_; }
    unsigned getCurso() const { return curso_; }
    unsigned getGrupo() const { return grupo_; }
    bool getEsLider() const { return esLider_; }
    void setDni(std::string dni);
```

```
    void setNombre(std::string nombre);
    void setApellidos(std::string apellidos);
    void setTelefono(std::string telefono);
    void setEmail(std::string email) { email_ = email; }
    void setDireccion(std::string direccion) { direccion_ = direccion; }
    void setFechaNacimiento(std::string fecha_nacimiento);
    void setCurso(unsigned curso) { curso_ = curso;
                                    assert(curso_<=4);}
    void setGrupo(unsigned grupo) { grupo_ = grupo; }
    void setEsLider(bool esLider) { esLider_ = esLider; }

  private:
    std::string dni_; //!< DNI del alumno.
    std::string nombre_; //!< Nombre del alumno.
    std::string apellidos_; //!< Ambos apellidos del alumno.
    std::string telefono_; //!< Teléfono de contacto.
    std::string email_; //!< Dirección de correo electrónico.
    std::string direccion_; //!< Dirección postal.
    std::string fechaNacimiento_; //!< Fecha de nacimiento.
    unsigned curso_; //!< Curso mas alto matriculado.
    unsigned grupo_; //!< Numero de grupo de'practicas.
    bool esLider_; //!< Determina si es lider o no del grupo.
}; // class alumno.

#endif // ALUMNO_HPP
```

```
/**
 * @file alumno.cpp
 * @brief Implementación de los métodos de la clase alumno.
 * @date 27-11-2018
 *
 * Este fichero forma parte del proyecto realizado para la gestión de alumnos,
 * para la asignatura Ingeniería del Software de la Universidad de Córdoba.
 */

#include <string>
#include "alumno.hpp"

// Constructor por defecto.
alumno::alumno()
```

```
{
    dni_ = "";
    nombre_ = "";
    apellidos_ = "";
    telefono_ = "";
    email_ = "";
    direccion_ = "";
    fechaNacimiento_ = "";
    curso_ = 0;
    grupo_ = 0;
    esLider_ = false;
}

// Constructor de la clase alumno.
alumno::alumno(std::string dni, std::string nombre, std::string apellidos,
               std::string telefono, std::string email, std::string direccion,
               std::string fechaNacimiento, unsigned curso, unsigned grupo,
               bool esLider)
{
    setDni(dni);
    setNombre(nombre);
    setApellidos(apellidos);
    setTelefono(telefono);
    setEmail(email);
    setDireccion(direccion);
    setFechaNacimiento(fechaNacimiento);
    setCurso(curso);
    setGrupo(grupo);
    setEsLider(esLider);
}

void alumno::setDni(std::string dni)
{
    dni_ = dni;
    assert(dni_.length() == 9);
    for (unsigned i = 0; i < 8; i++) {
        assert(std::isdigit(dni_[i]));
    }
    assert(std::isalpha(dni_[8]));
}
```

```
void alumno::setNombre(std::string nombre)
{
    nombre_ = nombre;
    for (unsigned i=0 ; i < nombre_.length() ; i++) {
        assert(std::isalpha(nombre_[i]));
    }
}

void alumno::setApellidos(std::string apellidos)
{
    apellidos_ = apellidos;
    for (unsigned i = 0; i < apellidos_.length(); i++) {
        assert(std::isalpha(apellidos_[i]) or apellidos_[i] == ' ');
    }
}

void alumno::setTelefono(std::string telefono)
{
    telefono_ = telefono;
    assert(telefono_.length() == 9);
    for (unsigned i = 0 ;i < 9; i++) {
        assert(std::isdigit(telefono_[i]));
    }
}

void alumno::setFechaNacimiento(std::string fecha_nacimiento)
{
    fechaNacimiento_ = fecha_nacimiento;
    assert(isdigit(fechaNacimiento_[0]) and isdigit(fechaNacimiento_[1]) and
           isdigit(fechaNacimiento_[3]) and isdigit(fechaNacimiento_[4]) and
           isdigit(fechaNacimiento_[6]) and isdigit(fechaNacimiento_[7]) and
           isdigit(fechaNacimiento_[8]) and isdigit(fechaNacimiento_[9]));
    assert(fechaNacimiento_[2]=='/');
    assert(fechaNacimiento_[5]=='/');
}
```

3.2.2. baseDatos

```
/**
 * @file baseDatos.hpp
 * @brief Declaración de la clase baseDatos.
 * @date 27-11-2018
 *
 * Este fichero forma parte del proyecto realizado para la gestión de alumnos,
 * para la asignatura Ingeniería del Software de la Universidad de Córdoba.
 */

#ifndef BASEDATOS_HPP
#define BASEDATOS_HPP

#include <string>
#include <list>
#include "alumno.hpp"

class baseDatos
{
private:
  std::list<alumno> alumnos_;

public:
  unsigned getNumeroAlumnos() const {return alumnos_.size(); }
  bool anadirAlumno(const alumno &nuevoAlumno);
  bool anadirAlumno(std::string dni, std::string nombre, std::string apellidos,
      std::string telefono, std::string email, std::string direccion,
      std::string fechaNacimiento, unsigned curso, unsigned grupo, bool esLider);
  bool modificarAlumno(std::string dni, alumno alumno);
  bool eliminarAlumno(std::string dni);
  void borrarAlumnos() { alumnos_.clear(); }
  bool getAlumno(std::string dni, alumno &a) const;
  void buscarAlumnos(std::list<alumno> &resultado, std::string apellidos = "",
       std::string dni = "", unsigned grupo = 0) const;
  void guardarFichero(std::string nombreFichero) const;
  void cargarFichero(std::string nombreFichero);
};

#endif // BASEDATOS_HPP
```

```
/**
 * @file basedatos.hpp
 * @brief Implementación de los métodos de la clase baseDatos.
 * @date 27-11-2018
 *
 * Este fichero forma parte del proyecto realizado para la gestión de alumnos,
 * para la asignatura Ingeniería del Software de la Universidad de Córdoba.
 */

#include <list>
#include <string>
#include <iostream>
#include <fstream>
#include "alumno.hpp"
#include "basedatos.hpp"


bool baseDatos::anadirAlumno(const alumno &nuevoAlumno) {
    if (getNumeroAlumnos() >= 150) {
        return false;
    }

    alumnos_.push_back(nuevoAlumno);
    return true;
}

bool baseDatos::anadirAlumno(std::string dni, std::string nombre,
    std::string apellidos, std::string telefono, std::string email,
    std::string direccion, std::string fechaNacimiento, unsigned curso,
    unsigned grupo, bool esLider) {

    if (getNumeroAlumnos() >= 150) {
        return false;
    }

    alumno nuevoAlumno(dni, nombre, apellidos, telefono, email, direccion, fechaNacimiento,
        curso, grupo, esLider);

    alumnos_.push_back(nuevoAlumno);
    return true;
```

```
}

bool baseDatos::modificarAlumno(std::string dni, alumno nuevoAlumno) {
    std::list<alumno>::iterator iter;

   if (getNumeroAlumnos() == 0) {
       return false;
   }

   for (iter = alumnos_.begin(); iter != alumnos_.end(); iter++) {
      if (iter->getDni() == dni) {
         iter->setDni(nuevoAlumno.getDni());
            iter->setNombre(nuevoAlumno.getNombre());
            iter->setApellidos(nuevoAlumno.getApellidos());
            iter->setTelefono(nuevoAlumno.getTelefono());
            iter->setEmail(nuevoAlumno.getEmail());
            iter->setDireccion(nuevoAlumno.getDireccion());
            iter->setFechaNacimiento(nuevoAlumno.getFechaNacimiento());
            iter->setCurso(nuevoAlumno.getCurso());
            iter->setGrupo(nuevoAlumno.getGrupo());
            iter->setEsLider(nuevoAlumno.getEsLider());
         return true;
      }

   }

    return false;
}

bool baseDatos::eliminarAlumno(std::string dni)
{
    std::list<alumno>::iterator iter;

   if (getNumeroAlumnos() == 0) {
       return false;
   }

   for (iter = alumnos_.begin(); iter != alumnos_.end(); iter++) {
      if (iter->getDni() == dni){
         alumnos_.erase(iter);
         return true;
```

```
      }

   }

   return false;
}

bool baseDatos::getAlumno(std::string dni, alumno &a) const
{
    std::list<alumno>::const_iterator iter;

    for (iter = alumnos_.begin(); iter != alumnos_.end(); iter++) {
        if (dni == iter->getDni()) {
            a.setDni(iter->getDni());
            a.setNombre(iter->getNombre());
            a.setApellidos(iter->getApellidos());
            a.setTelefono(iter->getTelefono());
            a.setEmail(iter->getEmail());
            a.setDireccion(iter->getDireccion());
            a.setFechaNacimiento(iter->getFechaNacimiento());
            a.setCurso(iter->getCurso());
            a.setGrupo(iter->getGrupo());
            a.setEsLider(iter->getEsLider());
            return true;
        }
    }

    return false;
}

void baseDatos::buscarAlumnos(std::list<alumno> &resultado,
    std::string apellidos, std::string dni, unsigned grupo) const
{
    std::list<alumno>::const_iterator iter;
    alumno nuevoAlumno;

    resultado.clear();
    for (iter = alumnos_.begin(); iter != alumnos_.end(); iter++) {
        if (apellidos == "" or apellidos == iter->getApellidos()) {
            if (dni == "" or dni == iter->getDni()) {
                if (grupo == 0 or grupo == iter->getGrupo()) {
```

```
                nuevoAlumno.setDni(iter->getDni());
                nuevoAlumno.setNombre(iter->getNombre());
                nuevoAlumno.setApellidos(iter->getApellidos());
                nuevoAlumno.setTelefono(iter->getTelefono());
                nuevoAlumno.setEmail(iter->getEmail());
                nuevoAlumno.setDireccion(iter->getDireccion());
                nuevoAlumno.setFechaNacimiento(iter->getFechaNacimiento());
                nuevoAlumno.setCurso(iter->getCurso());
                nuevoAlumno.setGrupo(iter->getGrupo());
                nuevoAlumno.setEsLider(iter->getEsLider());

                resultado.push_back(nuevoAlumno);
            }
        }
    }
}

void baseDatos::guardarFichero(std::string nombreFichero) const
{
    std::ofstream outfile(nombreFichero, std::ios::out | std::ios::binary);
    std::list<alumno>::const_iterator iter;
    unsigned len;

    if (outfile.is_open()) {
        for (iter = alumnos_.begin(); iter != alumnos_.end(); iter++) {
            // Escritura del dni.
            len = iter->getDni().length();
            outfile.write(reinterpret_cast<char *>(&len), sizeof(unsigned));
            outfile.write(iter->getDni().c_str(), len);

            // Escritura del nombre.
            len = iter->getNombre().length();
            outfile.write(reinterpret_cast<char *>(&len), sizeof(unsigned));
            outfile.write(iter->getNombre().c_str(), len);

            // Escritura de los apellidos.
            len = iter->getApellidos().length();
            outfile.write(reinterpret_cast<char *>(&len), sizeof(unsigned));
            outfile.write(iter->getApellidos().c_str(), len);
```

```
            // Escritura del telefono.
            len = iter->getTelefono().length();
            outfile.write(reinterpret_cast<char *>(&len), sizeof(unsigned));
            outfile.write(iter->getTelefono().c_str(), len);

            // Escritura del e-mail.
            len = iter->getEmail().length();
            outfile.write(reinterpret_cast<char *>(&len), sizeof(unsigned));
            outfile.write(iter->getEmail().c_str(), len);

            // Escritura de la direccion postal.
            len = iter->getDireccion().length();
            outfile.write(reinterpret_cast<char *>(&len), sizeof(unsigned));
            outfile.write(iter->getDireccion().c_str(), len);

            // Escritura del curso mas alto matriculado.
            unsigned curso = iter->getCurso();
            outfile.write(reinterpret_cast<char *>(&curso), sizeof(unsigned));

            // Escritura de la fecha de nacimiento.
            len = iter->getFechaNacimiento().length();
            outfile.write(reinterpret_cast<char *>(&len), sizeof(unsigned));
            outfile.write(iter->getFechaNacimiento().c_str(), len);

            // Escritura del grupo.
            unsigned grupo = iter->getGrupo();
            outfile.write(reinterpret_cast<char *>(&grupo), sizeof(unsigned));

            // Escritura de la condicion de lider.
            unsigned esLider = iter->getEsLider();
            outfile.write(reinterpret_cast<char *>(&esLider), sizeof(unsigned));
        }

        outfile.close();
    }
}

void baseDatos::cargarFichero(std::string nombreFichero) {
    std::ifstream infile(nombreFichero, std::ios::in | std::ios::binary);
    std::string dni;
    std::string nombre;
```

```
std::string apellidos;
std::string telefono;
std::string email;
std::string direccion;
std::string fechaNacimiento;
unsigned curso;
unsigned grupo;
unsigned aux;
bool esLider;
unsigned len;

if (infile.is_open()) {
    // Se eliminan la lista de alumnos previa carga.
    alumnos_.clear();

    while (infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned))) {
        dni.resize(len);
        infile.read(&dni[0], dni.size());

        // Lectura del nombre.
        infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned));
        nombre.resize(len);
        infile.read(&nombre[0], nombre.size());

        // Lectura de los apellidos.
        infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned));
        apellidos.resize(len);
        infile.read(&apellidos[0], apellidos.size());

        // Lectura del telefono.
        infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned));
        telefono.resize(len);
        infile.read(&telefono[0], telefono.size());

        // Lectura del e-mail.
        infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned));
        email.resize(len);
        infile.read(&email[0], email.size());

        // Lectura de la direccion postal.
        infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned));
```

```
            direccion.resize(len);
            infile.read(&direccion[0], direccion.size());

            // Lectura del curso mas alto matriculado.
            infile.read(reinterpret_cast<char *>(&curso), sizeof(unsigned));

            // Lectura de la fecha de nacimiento.
            infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned));
            fechaNacimiento.resize(len);
            infile.read(&fechaNacimiento[0], fechaNacimiento.size());

            // Lectura del grupo.
            infile.read(reinterpret_cast<char *>(&grupo), sizeof(unsigned));

            // Lectura de la condicion de lider.
            infile.read(reinterpret_cast<char *>(&aux), sizeof(unsigned));
            esLider = aux;

            // Se añade el alumno a la lista.
            anadirAlumno(dni, nombre, apellidos, telefono, email,
                direccion, fechaNacimiento, curso, grupo, esLider);
        }

        infile.close();
    }
}
```

3.2.3. profesor

```
/**
 * @file profesor.hpp
 * @brief Declaración de la clase profesor.
 * @date 27-11-2018
 *
 * Este fichero forma parte del proyecto realizado para la gestión de alumnos,
 * para la asignatura Ingeniería del Software de la Universidad de Córdoba.
 */

#ifndef PROFESOR_HPP
#define PROFESOR_HPP

#include <string>
#include <list>
#include "alumno.hpp"
#include "basedatos.hpp"

class profesor
{
  private:
    std::string nombreUsuario_;
    bool esCoordinador_;
    baseDatos bbdd_;
  public:
    profesor();
    std::string getNombreUsuario() const { return nombreUsuario_; }
    bool getEsCoordinador() const { return esCoordinador_; } //Devuelve si el profesor es
        coordinador o no.
    baseDatos& getBaseDatos() { return bbdd_; }
    const baseDatos& getBaseDatos() const { return bbdd_; }
    bool iniciarSesion(std::string nombreFichero, std::string usuario, std::string
        password); //Devuelve si el usuario introducido existe.
    void cerrarSesion();
    bool anadirProfesor(std::string nombreFichero, std::string usuario, std::string
        password) const; //Añade un profesor ayudante.
};

#endif // PROFESOR_HPP
```

```
/**
 * @file profesor.hpp
 * @brief Implementación de los métodos de la clase profesor.
 * @date 27-11-2018
 *
 * Este fichero forma parte del proyecto realizado para la gestión de alumnos,
 * para la asignatura Ingeniería del Software de la Universidad de Córdoba.
 */

#include <fstream>
#include <string>
#include "alumno.hpp"
#include "basedatos.hpp"
#include "profesor.hpp"

bool existeProfesor(std::string nombreFichero, std::string usuario)
{
    std::ifstream infile(nombreFichero, std::ios::in | std::ios::binary);
    std::string aux;
    unsigned len;
    bool existe = false;

    if (infile.is_open()) {
        // Lee el fichero hasta el final o hasta encontrar coincidencia.
        while (infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned)) and not
             existe) {
            aux.resize(len);
            infile.read(&aux[0], aux.size());

            // Si el profesor existe, lo señala para terminar la ejecución.
            existe = (aux == usuario);

            // Lee la contraseña para continuar por el siguiente profesor.
            infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned));
            aux.resize(len);
            infile.read(&aux[0], aux.size());
        }

        infile.close();
    }
```

```
    return existe;
}

profesor::profesor()
{
    nombreUsuario_ = "";
    esCoordinador_ = false;
}

bool profesor::iniciarSesion(std::string nombreFichero, std::string usuario,
    std::string password)
{
    bool esCoordinador = true;
    bool encontrado = false;
    std::ifstream infile(nombreFichero, std::ios::in | std::ios::binary);
    std::string nombreProfesor, contrProfesor;
    unsigned len;

    if (infile.is_open()) {
        while (infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned)) and not
            encontrado) {
            nombreProfesor.resize(len);
            infile.read(&nombreProfesor[0], nombreProfesor.size());

            // Lectura del nombre.
            infile.read(reinterpret_cast<char *>(&len), sizeof(unsigned));
            contrProfesor.resize(len);
            infile.read(&contrProfesor[0], contrProfesor.size());

            // Si el profesor se ha encontrado, lo señala para terminar la ejecución.
            if ((usuario == nombreProfesor) and (password == contrProfesor)) {
                encontrado = true;
                nombreUsuario_ = nombreProfesor;
                esCoordinador_ = esCoordinador;
            }

            // El primer profesor del registro es el coordinador.
            esCoordinador = false;
        }
```

```
        infile.close();
    }

    return encontrado;
}

void profesor::cerrarSesion()
{
    nombreUsuario_ = "";
    esCoordinador_ = "";
    bbdd_.borrarAlumnos();
}

bool profesor::anadirProfesor(std::string nombreFichero, std::string usuario,
    std::string password) const
{
    // Si el profesor no es coordinador no puede registrar otros profesores.
    // También impide crear dos profesores con el mismo nombre de usuario.
    if (not getEsCoordinador() or existeProfesor(nombreFichero, usuario)) {
        return false;
    }

    std::ofstream outfile(nombreFichero, std::ios::app | std::ios::binary);
    unsigned len;

    if (outfile.is_open()) {
        // Escritura del nombre.
        len = usuario.length();
        outfile.write(reinterpret_cast<char *>(&len), sizeof(unsigned));
        outfile.write(usuario.c_str(), len);

        // Escritura de la contraseña.
        len = password.length();
        outfile.write(reinterpret_cast<char *>(&len), sizeof(unsigned));
        outfile.write(password.c_str(), len);

        outfile.close();
        return true;
    }
    else {
        return false;
```

```
    }
}
```

3.2.4. is

```
#ifndef IS_HPP
#define IS_HPP

// Fichero de cabecera que incluye la base del proyecto.

#include "alumno.hpp"
#include "basedatos.hpp"
#include "profesor.hpp"

#endif // IS_HPP
```

3.2.5. ppal-is

```
/**
 * @file main.cpp
 * @brief Programa que permite la gestión de prof.getBaseDatos() de una asignatura.
 * @date 11-12-2018
 *
 * Este fichero forma parte del proyecto realizado para la gestión de prof.getBaseDatos(),
 * para la asignatura Ingeniería del Software de la Universidad de Córdoba.
 */

#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include "is.hpp"

#define MAX_ALUMNOS 150
#define FILE_CREDENCIALES "../data/credenciales.bin"
#define FILE_BBDD_LOCAL "../data/localdata.bin"

enum MENU_OPTS
{
    ADD_ALUMNO = 1,
    MODIFICAR = 2,
    BORRAR = 3,
    MOSTRAR = 4,
    GUARDAR = 5,
    CARGAR = 6,
    GUARDAR_COPIA = 7,
    CARGAR_COPIA = 8,
    ADD_PROFESOR = 9,
    SALIR = 0
};


// Funciones auxiliares de comparación para la ordenación.
bool compararDni(const alumno &a, const alumno &b) { return a.getDni() < b.getDni(); }
bool compararApellidos(const alumno &a, const alumno &b) { return a.getApellidos() <
    b.getApellidos(); }
```

```
bool compararCurso(const alumno &a, const alumno &b) { return a.getCurso() < b.getCurso();
    }

void pulseEnter();
bool acceder(profesor &prof);
bool registrar(const profesor &prof);
unsigned escribirMenu(bool esCoordinador);
void obtenerAlumno(alumno &a);
void mostrarAlumnoTerminal(const alumno &a);
unsigned getCriteriosBusqueda(unsigned numeroAlumnos, std::string dni,
    std::string apellidos, unsigned grupo, bool asc, unsigned criterio,
    bool formato_md);
void setCriteriosBusqueda(unsigned option, std::string &dni, std::string &apellidos,
    unsigned &grupo, bool &asc, unsigned &criterio, bool &formato_md);
void ordenarAlumnos(std::list<alumno> &alumnos, unsigned criterio, bool asc);
void mostrarAlumnosMarkdown(const std::list<alumno> &alumnos);
void mostrarAlumnosHTML(const std::list<alumno> &alumnos);
void listarAlumnos(const profesor &prof);

int main(void)
{
    profesor prof;
    bool b;
    alumno alumnoAux;
    std::string stringAux;
    std::list<alumno> resultado;
    unsigned option;

    // Inicio de sesión del usuario.
    do {
        b = acceder(prof);
        if (not b) {
            std::cout << "Datos erróneos. Por favor, inténtelo otra vez. ";
            pulseEnter();
        }
    } while (not b);

    do {
        switch(option = escribirMenu(prof.getEsCoordinador())) {
            case ADD_ALUMNO:
                if (prof.getBaseDatos().getNumeroAlumnos() >= MAX_ALUMNOS) {
```

```
            std::cout << "El número máximo de alumnos ha sido alcanzado. ";
        }
        else {
            obtenerAlumno(alumnoAux);
            if (prof.getBaseDatos().anadirAlumno(alumnoAux) == true) {
                std::cout << "Alumno añadido correctamente. ";
            }
            else {
                std::cout << "Ha ocurrido algún problema durante la inserción. ";
            }
        }
        pulseEnter();
        break;

    case MODIFICAR:
        if (prof.getBaseDatos().getNumeroAlumnos() == 0) {
            std::cout << "La lista de alumnos está vacía. ";
            pulseEnter();
        }
        break;

    case BORRAR:
        if (prof.getBaseDatos().getNumeroAlumnos() == 0) {
            std::cout << "La lista de alumnos está vacía. ";
        }
        else {
            std::system("clear");
            std::cout << "BORRAR ALUMNO: " << std::endl;
            std::cout << "\tDNI del alumno a eliminar: ";
            std::cin >> stringAux;

            b = prof.getBaseDatos().getAlumno(stringAux, alumnoAux);
            if (b == true) {
                std::cout << "El alumno seleccionado es el siguiente:" << std::endl;
                mostrarAlumnoTerminal(alumnoAux);
                std::cout << "¿Desea eliminarlo de la base de datos? (S/N): ";
                std::cin >> stringAux;
                if (stringAux == "S" or stringAux == "s") {
                    prof.getBaseDatos().eliminarAlumno(alumnoAux.getDni());
                    std::cout << "Alumno eliminado correctamente. ";
                }
```

```
            }
            else {
                std::cout << "No se ha encontrado al alumno. ";
            }
        }
        pulseEnter();
        break;

    case MOSTRAR:
        listarAlumnos(prof);
        pulseEnter();
        break;

    case GUARDAR:
        prof.getBaseDatos().guardarFichero(FILE_BBDD_LOCAL);
        std::cout << "Cambios guardados correctamente. ";
        pulseEnter();
        break;

    case CARGAR:
        prof.getBaseDatos().cargarFichero(FILE_BBDD_LOCAL);
        std::cout << "Datos cargados correctamente. ";
        pulseEnter();
        break;

    case GUARDAR_COPIA:
        if (prof.getEsCoordinador()) {
            std::cout << "Indique el nombre de la copia de seguridad: ";
            std::cin >> stringAux;
            prof.getBaseDatos().guardarFichero("../data/" + stringAux + ".bin");
            std::cout << "Copia de seguridad guardada correctamente. ";
            pulseEnter();
        }
        break;

    case CARGAR_COPIA:
        if (prof.getEsCoordinador()) {
            std::cout << "Indique el nombre de la copia de seguridad: ";
            std::cin >> stringAux;
            prof.getBaseDatos().cargarFichero("../data/" + stringAux + ".bin");
            std::cout << "Datos cargados correctamente. ";
```

```
                pulseEnter();
            }
            break;

        case ADD_PROFESOR:
            if (prof.getEsCoordinador()) {
                b = registrar(prof);
                if (not b) {
                    std::cout << "Error al añadir el nuevo usuario. Puede que ya exista.
                        ";
                }
                else {
                    std::cout << "Usuario añadido correctamente. ";
                }
                pulseEnter();
            }
            break;

        case SALIR:
            break;
      }
   } while (option != SALIR);

  exit(EXIT_SUCCESS);
}

void pulseEnter()
{
    std::cout << "Pulse ENTER para continuar. ";
    std::cin.ignore();
    std::cin.ignore();
}

bool acceder(profesor &prof)
{
    std::string usuario;
    std::string password;

    std::system("clear");
    std::cout << "ACCEDER:" << std::endl;
```

```
    std::cout << "\tNombre de usuario: ";
    std::cin >> usuario;

    std::cout << "\tContraseña: ";
    std::cin >> password;

    return prof.iniciarSesion(FILE_CREDENCIALES, usuario, password);
}

bool registrar(const profesor &prof)
{
    std::string usuario;
    std::string password;

    std::system("clear");
    std::cout << "REGISTRAR USUARIO:" << std::endl;

    std::cout << "\tNombre de usuario: ";
    std::cin >> usuario;

    std::cout << "\tContraseña: ";
    std::cin >> password;

    return prof.anadirProfesor(FILE_CREDENCIALES, usuario, password);
}

// Imprime por pantalla el menu de opciones.
unsigned escribirMenu(bool esCoordinador)
{
    unsigned option;

    std::system("clear");
    std::cout << "MENÚ DE OPCIONES:" << std::endl;
    std::cout << "\t1. Añadir alumno." << std::endl;
    std::cout << "\t2. Modificar alumno." << std::endl;
    std::cout << "\t3. Eliminar alumno." << std::endl;
    std::cout << "\t4. Mostrar alumno." << std::endl;
    std::cout << "\t5. Guardar cambios." << std::endl;
    std::cout << "\t6. Cargar cambios." << std::endl;

    // Opciones reservadas a los profesores coordinadores.
```

```
    if (esCoordinador) {
        std::cout << "\t7. Guardar copia de seguridad." << std::endl;
        std::cout << "\t8. Cargar copia de seguridad." << std::endl;
        std::cout << "\t9. Añadir profesor colaborador." << std::endl;
    }
    std::cout << "\t0. Salir del programa." << std::endl;

    // Se pide la opción al usuario.
    std::cout << "Seleccione opción: ";
    std::cin >> option;

    return option;
}

void obtenerAlumno(alumno &a)
{
    std::string aux;

    std::system("clear");
    std::cout << "AÑADIR ALUMNO: " << std::endl;

    std::cout << "\tDNI: ";
    std::cin >> aux;
    a.setDni(aux);

    std::cout << "\tNombre: ";
    std::cin >> aux;
    a.setNombre(aux);

    std::cout << "\tApellidos: ";
    std::cin.ignore();
    std::getline(std::cin, aux);
    a.setApellidos(aux);

    std::cout << "\tTelefono: ";
    std::cin >> aux;
    a.setTelefono(aux);

    std::cout << "\te-mail: ";
    std::cin >> aux;
    a.setEmail(aux);
```

```
    std::cout << "\tDirección postal: ";
    std::cin >> aux;
    a.setDireccion(aux);

    std::cout << "\tFecha de nacimiento (dd/mm/aaaa): ";
    std::cin >> aux;
    a.setFechaNacimiento(aux);

    std::cout << "\tCurso más alto matriculado: ";
    std::cin >> aux;
    a.setCurso(atoi(aux.c_str()));

    std::cout << "\tGrupo: ";
    std::cin >> aux;
    a.setGrupo(atoi(aux.c_str()));

    if (a.getGrupo() != 0) {
        std::cout << "\t¿Es líder del grupo? (S/N): ";
        std::cin >> aux;
        a.setEsLider(aux == "S" or aux == "s"? true : false);
    }
}

void mostrarAlumnoTerminal(const alumno &a)
{
    std::cout << "\tDNI: " << a.getDni() << std::endl;
    std::cout << "\tNombre: " << a.getNombre() << std::endl;
    std::cout << "\tApellidos: " << a.getApellidos() << std::endl;
    std::cout << "\tTelefono: " << a.getTelefono() << std::endl;
    std::cout << "\te-mail: " << a.getEmail() << std::endl;
    std::cout << "\tDirección postal: " << a.getDireccion() << std::endl;
    std::cout << "\tFecha de nacimiento: " << a.getFechaNacimiento() << std::endl;
    std::cout << "\tCurso más alto matriculado: " << a.getCurso() << std::endl;
    std::cout << "\tGrupo: " << a.getGrupo() << std::endl;
    std::cout << "\tLider del Grupo: " << (a.getEsLider()? "Si" : "No") << std::endl;
}

unsigned getCriteriosBusqueda(unsigned numeroAlumnos, std::string dni,
    std::string apellidos, unsigned grupo, bool asc, unsigned criterio,
    bool formato_md)
```

```
{
    unsigned option;

    std::system("clear");
    std::cout << "MOSTRAR ALUMNOS: " << std::endl;
    std::cout << numeroAlumnos << " resultados coincidentes." << std::endl;
    std::cout << "\t1. DNI: " << dni << std::endl;
    std::cout << "\t2. Apellidos: " << apellidos << std::endl;
    std::cout << "\t3. Grupo: " << grupo << std::endl;
    std::cout << "\t4. Orden: " << (asc? "ascendente" : "descendente") << std::endl;

    std::cout << "\t5. Criterio de orden: ";
    switch (criterio) {
        case 0:
            std::cout << "por orden de inserción." << std::endl;
            break;

        case 1:
            std::cout << "por dni." << std::endl;
            break;

        case 2:
            std::cout << "por apellidos." << std::endl;
            break;

        case 3:
            std::cout << "por curso." << std::endl;
            break;

        default:
            std::cout << "error." << std::endl;
            break;
    }

    std::cout << "\t6. Formato: " << (formato_md? ".md" : ".html") << std::endl;
    std::cout << "\t0. Salir." << std::endl;

    std::cout << "Seleccione opción: ";
    std::cin >> option;

    return option;
```

```
}


void setCriteriosBusqueda(unsigned option, std::string &dni, std::string &apellidos,
    unsigned &grupo, bool &asc, unsigned &criterio, bool &formato_md)
{
    unsigned aux;

    switch (option) {
        case 1:
            std::cout << "DNI: ";
            std::cin >> dni;
            std::cout << "Campo modificado. ";
            pulseEnter();
            break;

        case 2:
            std::cout << "Apellidos: ";
            std::getline(std::cin, apellidos);
            std::cout << "Campo modificado. ";
            pulseEnter();
            break;

        case 3:
            std::cout << "Grupo: ";
            std::cin >> grupo;
            std::cout << "Campo modificado. ";
            pulseEnter();
            break;

        case 4:
            asc = not asc;
            std::cout << "Campo modificado. ";
            pulseEnter();
            break;

        case 5:
            std::cout << "Posibles criterios: " << std::endl;
            std::cout << "\t1. por orden de inserción." << std::endl;
            std::cout << "\t2. por dni." << std::endl;
            std::cout << "\t3. por apellidos." << std::endl;
```

```
            std::cout << "\t4. por curso." << std::endl;
            std::cout << "Criterio de orden: ";
            std::cin >> aux;
            if (aux > 0 and aux < 5) {
                std::cout << "Campo modificado. ";
                criterio = aux;
            }
            else {
                std::cout << "Valor incorrecto. ";
            }
            pulseEnter();
            break;

         case 6:
            formato_md = not formato_md;
            std::cout << "Campo modificado. ";
            pulseEnter();
            break;
    }
}

void ordenarAlumnos(std::list<alumno> &alumnos, unsigned criterio, bool asc)
{
    switch (criterio) {
        case 2:
            alumnos.sort(compararDni);
            break;

        case 3:
            alumnos.sort(compararApellidos);
            break;

        case 4:
            alumnos.sort(compararCurso);
            break;
    }

    if (not asc) {
        alumnos.reverse();
    }
}
```

```
void mostrarAlumnosMarkdown(const std::list<alumno> &alumnos)
{
    std::list<alumno>::const_iterator iter;
    std::ofstream outfile("../output.md");

    if (outfile.is_open()) {

        outfile << "**LISTA DE ALUMNOS**" << std::endl << std::endl;
        outfile << "---" << std::endl << std::endl;

        for (iter = alumnos.begin(); iter != alumnos.end(); iter++) {
            if (iter->getEsLider()) {
                outfile << "* **Nombre: " << iter->getApellidos() << ", " <<
                    iter->getNombre() << "**" << std::endl;
            } else {
                outfile << "* Nombre: " << iter->getApellidos() << ", " << iter->getNombre()
                    << std::endl;
            }
            outfile << "* DNI: " << iter->getDni() << std::endl;
            outfile << "* Telefono: " << iter->getTelefono() << std::endl;
            outfile << "* e-mail: " << iter->getEmail() << std::endl;
            outfile << "* Direccion: " << iter->getDireccion() << std::endl;
            outfile << "* Fecha de nacimiento: " << iter->getFechaNacimiento() << std::endl;
            outfile << "* Curso mas alto matriculado: " << iter->getCurso() << std::endl;
            outfile << "* Grupo: " << iter->getGrupo() << std::endl;
            outfile << "* Lider del grupo: " << (iter->getEsLider()? "Si" : "No") <<
                std::endl;
            outfile << "---" << std::endl;
        }

        outfile.close();
    }
}

void mostrarAlumnosHTML(const std::list<alumno> &alumnos)
{
    std::list<alumno>::const_iterator iter;
    std::ofstream outfile("../output.html");

    if (outfile.is_open()) {
```

```
        // Se imprime la cabecera.
        outfile << "<!DOCTYPE html>" << std::endl;
        outfile << "<html>" << std::endl;
        outfile << "<body>" << std::endl << std::endl;
        outfile << "<h1>LISTA DE ALUMNOS</h1>" << std::endl << std::endl;

        for (iter = alumnos.begin(); iter != alumnos.end(); iter++) {
            outfile << "<ul>" << std::endl;
            if (iter->getEsLider()) {
                outfile << "<li><b>Nombre: " << iter->getApellidos() << ", " <<
                    iter->getNombre() << "</b></li>" << std::endl;
            } else {
                outfile << "<li>Nombre: " << iter->getApellidos() << ", " <<
                    iter->getNombre() << "</li>" << std::endl;
            }
            outfile << "<li>DNI: " << iter->getDni() << "</li>" << std::endl;
            outfile << "<li>Telefono: " << iter->getTelefono() << "</li>" << std::endl;
            outfile << "<li>e-mail: " << iter->getEmail() << "</li>" << std::endl;
            outfile << "<li>Direccion: " << iter->getDireccion() << "</li>" << std::endl;
            outfile << "<li>Fecha de nacimiento: " << iter->getFechaNacimiento() << "</li>"
                << std::endl;
            outfile << "<li>Curso mas alto matriculado: " << iter->getCurso() << "</li>" <<
                std::endl;
            outfile << "<li>Grupo: " << iter->getGrupo() << "</li>" << std::endl;
            outfile << "<li>Lider del grupo: " << (iter->getEsLider()? "Si" : "No") <<
                "</li>" << std::endl;
            outfile << "</ul>" << std::endl << std::endl;
        }

        outfile << "</body>" << std::endl;
        outfile << "</html>" << std::endl;

        outfile.close();
    }
}

void listarAlumnos(const profesor &prof)
{
    std::string dni;
    std::string apellidos;
    unsigned grupo = 0;
```

```
    unsigned criterio = 0;
    bool asc = true;
    bool formato_md = true;
    unsigned option;
    std::list<alumno> resultado;

    do {
        // Se selecciona el criterio de búsqueda a modificar.
        option = getCriteriosBusqueda(prof.getBaseDatos().getNumeroAlumnos(),
            dni, apellidos, grupo, asc, criterio, formato_md);

        // Se modifica dicho criterio de búsqueda.
        setCriteriosBusqueda(option, dni, apellidos, grupo, asc, criterio, formato_md);

        // Se buscan los alumnos con los criterios actualizados.
        prof.getBaseDatos().buscarAlumnos(resultado, apellidos, dni, grupo);

        // Se ordena el resultado.
        ordenarAlumnos(resultado, criterio, asc);

        // Se muestran en el formato seleccionado.
        if (formato_md) {
            mostrarAlumnosMarkdown(resultado);
        }
        else {
            mostrarAlumnosHTML(resultado);
        }

    } while (option != 0);
}
```

3.3. Técnicas de validación

En este capítulo se llevarán a cabo una serie de técnicas de validación para comprobar la corrección y completitud de la información, realizando para ello distintas matrices de validación.

3.3.1. Matriz de correspondencia RF/Casos de Uso

En la tabla 3.3 se muestra la matriz de correspondencia entre los requisitos funcionales y los casos de uso.

	CU-1	CU-2	CU-3	CU-4	CU-5	CU-6	CU-7	CU-8	CU-9
RU-1	x								
RU-2		x							
RU-3			x						
RU-4				x					
RF-5					x				
RF 6						x			
RU-7							x		
RU-8								x	
RU-9									x

Tabla 3.3: Matriz de correspondencia RF/Casos de Uso

3.3.2. Matriz de correspondencia CU/Clases

En la tabla 3.4 se muestra la matriz de correspondencia entre los casos de uso y las clases.

	Alumno	BaseDatos	Profesor
CU-1	x	x	
CU-2	x	x	
CU-3	x	x	
CU-4	x	x	
CU-5	x	x	
CU-6	x	x	
CU-7	x	x	
CU-8			x
CU-9			x

Tabla 3.4: Matriz de correspondencia CU/Clases