UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Ingeniería del Software 2

EJERCIO DE LA AGENDA MONOLÍTICA

PARA LA ASIGNATURA INGENIERÍA DEL SOFTWARE 2

3º GRADO EN INGENIERÍA INFORMÁTICA

(AGENDA MONOLÍTICA)

Autor: Eusebio Guijarro Collado

Índice de contenido:

1. INTRODUCCIÓN.

Este proyecto consiste en la realización del ejercicio de clase, **E.01.02.01.01. Agenda Monolítica**, de la asignatura Ingeniería del software 2, del curso 3º de Grado en Ingeniería Informática.

El ejercicio consiste en la completa realización de una aplicación con arquitectura monolítica que debe satisfacer los requisitos funcionales propuestos en el enunciado, basándose en el proceso unificado de desarrollo y siguiendo el patrón MVC:

Se tiene un equipo de desarrollo formado por cuatro personas, todas con similares habilidades en el desarrollo y todas con la misma tasa por hombre (25 €/hora). Se quiere desarrollar una agenda que tiene los requisitos funcionales mostrados en la Tabla:

Req#	Objetivo
RF.1	Añadir un contacto
RF.2	Borrar un contacto
RF.3.	Modificar un contacto
RF.4.	Buscar un contacto

Tabla 2. Requisitos funcionales de la Agenda

2. MARCO TECNOLÓGICO DEL TRABAJO.

A continuación se describen las herramientas y tecnologías que se requieren para la realización del ejercicio.

Medios Hardware

El único medio hardware para realizar este ejercicio es un ordenador.

Medios Software

Visual Paradigm: Es una herramienta muy avanzada para el modelado de UML. Se usará para modelar los casos de uso y crear los diagramas que se consideren necesarios.

Github: Es una plataforma de trabajo colaborativo que utiliza el control de versiones git. (https://github.com/Ejercicio-Agenda)

Git: Software de control de versiones. Cliente de github donde se alojará el proyecto.

Net Beans: Entorno de desarrollo de software. Esta herramienta ha sido seleccionada porque es muy versátil dada la cantidad de herramientas que provee y su capacidad para integrarla con Visual Paradigm y Github. También es la preferida por el equipo de desarrollo debido a que, por ser usuarios con experiencia en su uso, pueden generar código de forma tremendamente eficiente por conocer los atajos de teclado que dotan al código de gran maleabilidad sobre el documento y otras características que aumentan el grado de usabilidad del editor.

Microsoft Project: Esta herramienta se utilizará para gestionar la planificación temporal del proyecto. Concretamente la realización de los diagramas de Gantt.

3. EQUIPO Y MÉTODO DE TRABAJO.

El proyecto cuenta con un equipo de 4 personas, para su desarrollo, que cobran 25€/h. El método de trabajo escogido para este proyecto es el proceso unificado de desarrollo porque se ha decidido que es el que mejor se adapta a las circunstancias.

4. FASE DE INICIO

En la fase de inicio se realiza la captura e identificación de requisitos, el estudio de la viabilidad del sistema y la planificación temporal del proyecto.

También se procede a instalar y configurar todas las herramientas necesarias para la creación del proyecto.

4.1 CAPTURA E IDENTIFICACIÓN DE REQUISITOS

Los requisitos a tener en cuenta son los siguientes, durante esta fase, se ha valorado el añadir el requisito de autenticarse en el sistema **RF1**.

Requisito	**Objetivo**
RF.1	Autenticarse
RF.2	Añadir contacto
RF.3	Borrar contacto
RF.4	Modificar contacto
RF.5	Buscar contacto

4.2 REQUISITOS FUNCIONALES

Para analizar los requisitos propuestos se tendrá en cuenta los roles de los usuarios de la aplicación. En esta aplicación el único rol identificado es el rol de usuario. A continuación se detallan los requisitos a tener en cuenta:

- **RF.1 Autenticarse:** El usuario debe poder demostrar que es él quien accede a su cuenta y poder protegerla con una contraseña.

- **RF.1 Añadir un contacto:** El usuario debe poder añadir contactos en su agenda.

- **RF.2 Borrar un contacto:** El usuario elimina un contacto de su agenda.

- **RF.3 Modificar un contacto:** El usuario modifica ciertas características de un

contacto existente en su agenda.

- **RF.4 Buscar un contacto:** El usuario busca un contacto filtrando alguno de los campos que identifican unívocamente al contacto.

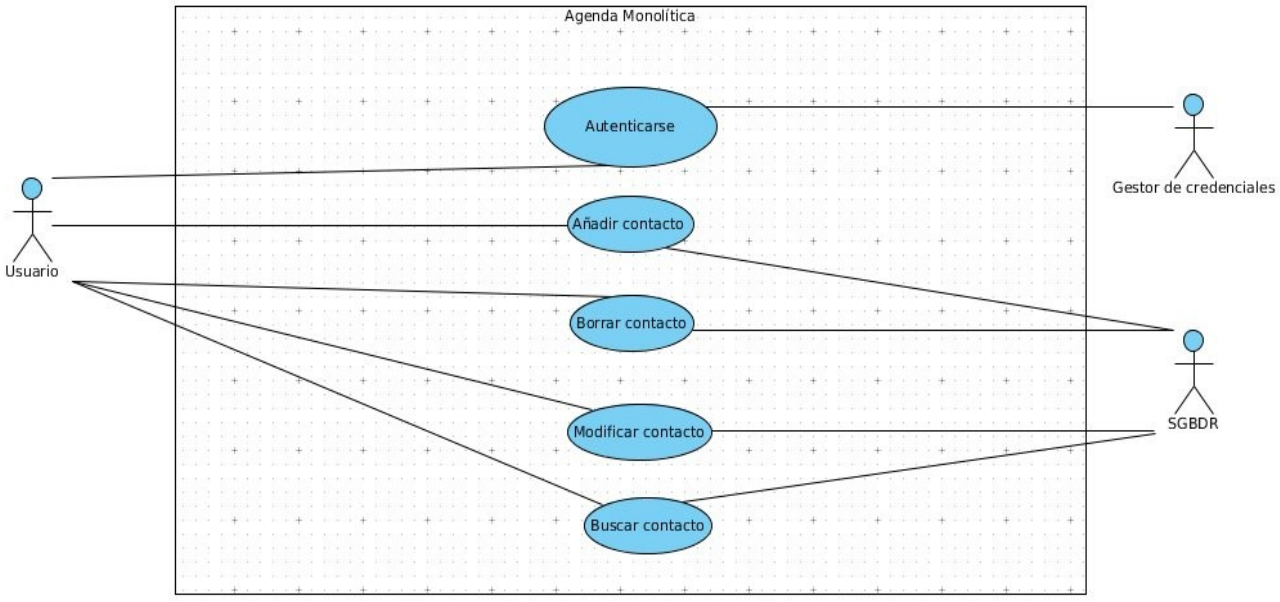
4.3 REQUISITOS NO FUNCIONALES

- **RNF.1** : Será necesario crear una base de datos con las tablas que se utilicen para la persistencia.

- **RNF.2 :** La interfaz de usuario debe ser sencilla e intuitiva.

4.4 MODELADO DE CASOS DE USO

En este apartado se procede a modelar, los requisitos identificados en la etapa anterior, en un diagrama de casos de uso.



4.5 GRUPOS FUNCIONALES

Se han identificado los siguientes grupos funcionales en el análisis de los casos de uso. El objetivo es centrar el desarrollo en cada uno de ellos independientemente.

Grupo funcional	**F1** Autenticarse
Casos de uso	Autenticarse
Requisitos funcionales	**RF.1** Autenticarse

Grupo funcional	**F2** Añadir contacto
Casos de uso	Añadir contacto
Requisitos funcionales	**RF.2** Añadir contacto

Grupo funcional	**F3** Borrar contacto
Casos de uso	Borrar contacto
Requisitos funcionales	**RF.3** Borrar contacto

Grupo funcional	**F4** Modificar contacto
Casos de uso	Modificar contacto
Requisitos funcionales	**RF.4** Modificar contacto

Grupo funcional	**F5** Buscar contacto
Casos de uso	Buscar contacto
Requisitos funcionales	**RF.5** Buscar contacto

4.6 PRIORIDADES

Se realiza la evaluación de las prioridades con que deben desarrollarse los distintos grupos funcionales y se muestra en la siguiente tabla.

Prioridad	**Grupo funcional**	**ENUMERACIÓN**
5	Autenticarse	F1
1	Añadir contacto	F2
3	Borrar contacto	F3
3	Modificar contacto	F4
2	Buscar contacto	F5

4.7 PLANIFICACIÓN TEMPORAL

4.7.1 PLAN DE ITERACIONES

Iteración	0
Fase	Inicio
Objetivos	- Capturar los requisitos funcionales y no funcionales del sistema. - Realizar el modelado de casos de uso. - Realizar la planificación temporal. - Planificación económica.
Artefactos	- Modelo general de casos de uso del sistema. - Plan de iteraciones. - Diagrama de Gantt. - Planificación económica.

Iteración	1
Fase	Elaboración
Objetivos	- Revisar los requisitos funcionales. -Revisar el plan de iteraciones teniendo en cuenta los nuevos posibles requisitos funcionales. - Definir y diseñar la arquitectura.
Artefactos	- Diagrama de casos de uso del sistema. - Plan de iteraciones. - Diseño de la línea base de la arquitectura.

Iteración	2
Fase	Elaboración
Objetivos	- Analizar e identificar los objetos del dominio del sistema. - Diseñar la base de datos. - Diseñar la arquitectura
Artefactos	- Diseño de los objetos del dominio. - Implementación de la línea base de la arquitectura.

Iteración	3
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Añadir contacto” F2.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F2.**

Iteración	4
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Buscar contacto” F5.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F5.**

Iteración	5
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Borrar contacto” F3.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F3.**

Iteración	6
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Modificar contacto” F4.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F4.**

Iteración	7
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Autenticarse” F1.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F1.**

Iteración	8
Fase	Transición
Objetivos	- Probar la aplicación
Artefactos	- Resultados de las pruebas**.**

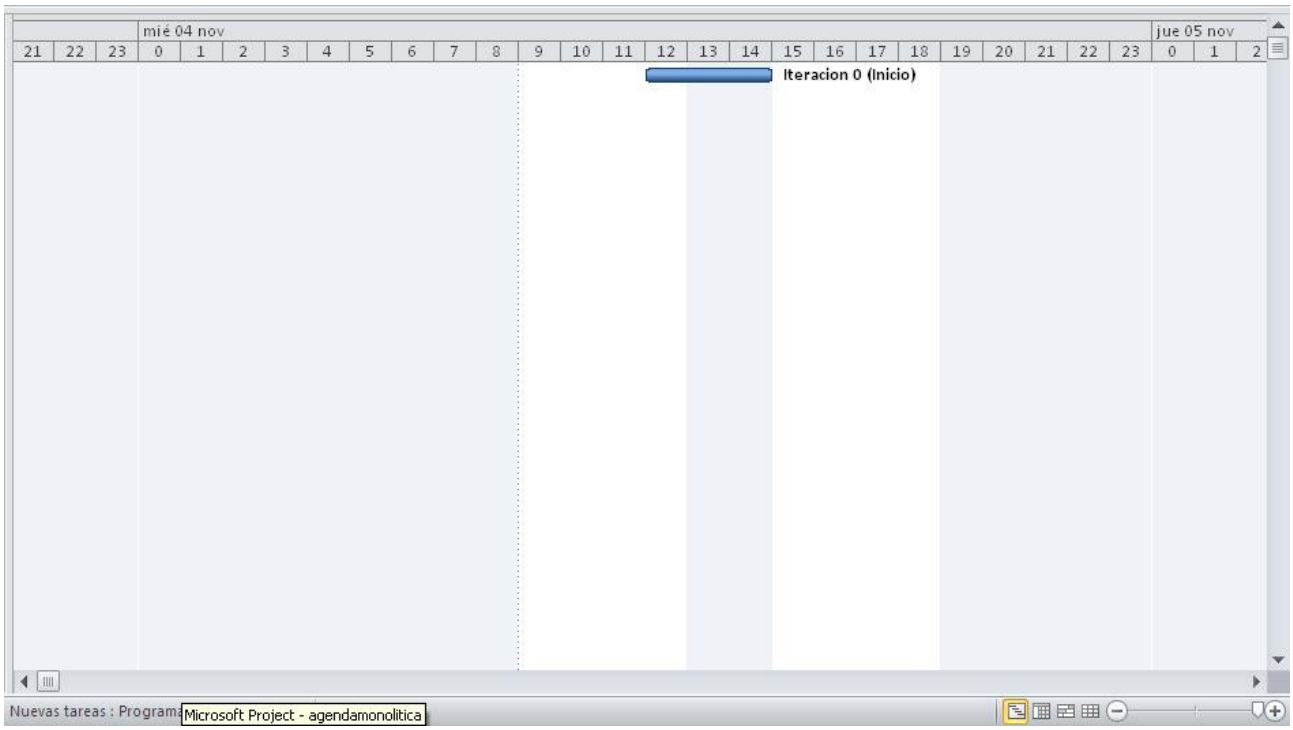
4.7.2 ESTIMACIÓN TEMPORAL Y ASIGNACIÓN DE RECURSOS

Iteración	**Estimación temporal**	**Asignación de recursos**
0	3 horas	4 Desarrolladores
1	3 horas	4 Desarrolladores
2	3 horas	4 Desarrolladores
3	1 horas	2 Desarrolladores
4	1 hora	2 Desarrolladores
5	1 hora	2 Desarrolladores
6	1 hora	2 Desarrolladores
7	1 hora	4 Desarrolladores

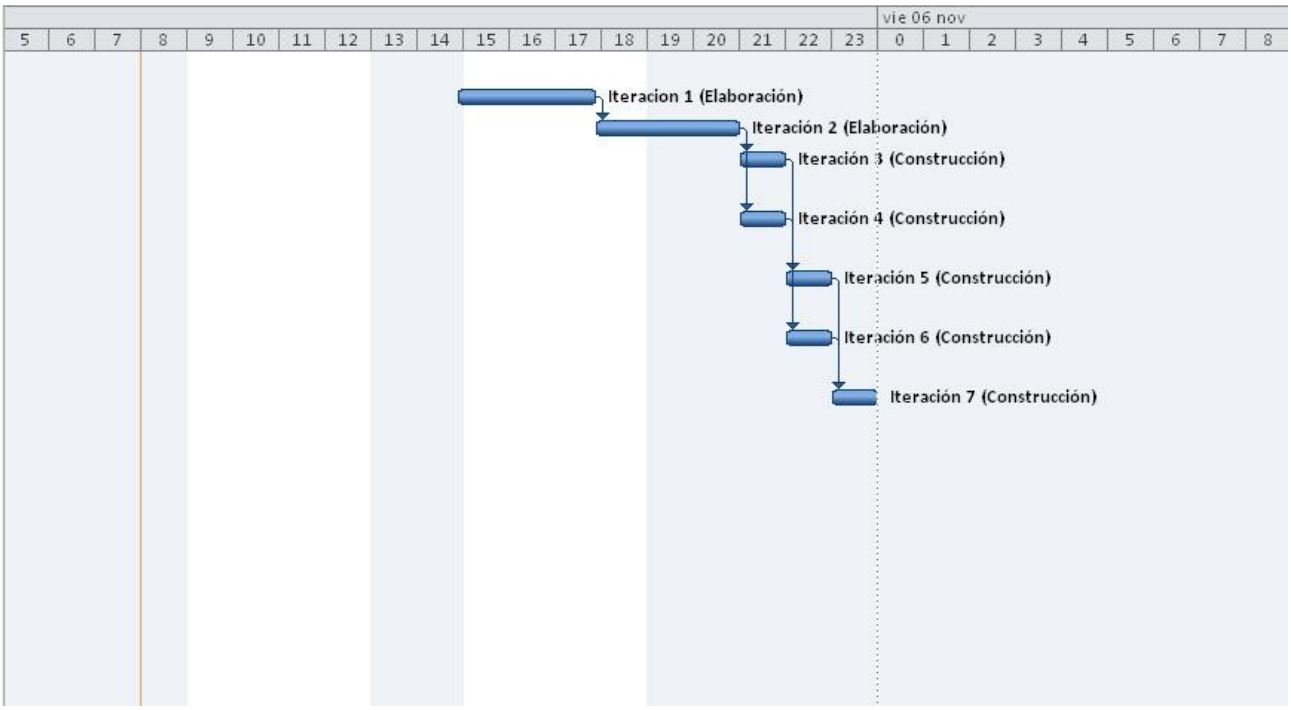
4.7.2 DIAGRAMA DE GANTT

Teniendo en cuenta todo lo anterior, se estructura la planificación temporal en dos diagramas de Gantt:

Día 4 de noviembre 2015:



Día 5 noviembre de 2015:



La iteración 0, de la fase de inicio, será llevada a cabo por los 4 miembros del equipo durante el primer día de trabajo y tendrá una duración estimada de 3 horas comenzando la jornada a las 12 de la mañana.

El día 5 de noviembre, la jornada laboral comienza a las 3 de la tarde cuando los 4 desarrolladores trabajarán en la iteración 1, que tendrá una duración de 3 horas y será predecesora de la iteración 2 de otras tres horas de duración. Esto concluirá con la fase de elaboración.

A partir de las 9 de la noche comienza la fase de construcción, en ella, 2 equipos de 2 desarrolladores trabajarán paralelamente en la iteración 3 y 4. Cuando acabe la iteración 3, el equipo de dos desarrolladores encargados de ella, comenzarán a ejecutar la iteración 6, y cuando a cabe la iteración 4 los 2 desarrolladores que han terminado comenzarán con la iteración 5.

Finalizando la iteración 5 y 6, los cuatro desarrolladores se pondrán a trabajar en la iteración 7, para la cual se estima una hora de duración dado el cansancio acumulado durante toda la jornada laboral.

4.8 PLANIFICACIÓN ECONÓMICA

El equipo de desarrollo lo forman cuatro personas y su coste es de 25€/h. Según la planificación temporal, el proyecto tiene una duración de 12 horas de trabajo activo y, por tanto, el coste estimado para la realización del proyecto es de 1200 euros.

5. FASE DE ELABORACIÓN

La fase de elaboración cuenta con dos iteraciones, durante la iteración 1 se repasan los requisitos y se actualizan los casos de uso y el plan de iteraciones con las conclusiones obtenidas. También se debe formular la línea base de la arquitectura.

La iteración 2 concluirá con el diseño de los diagramas de clases de dominio.

5.1 ITERACIÓN 1

En esta iteración se realiza un análisis en profundidad de los requisitos y se actualiza el modelo de casos de uso y la planificación temporal teniendo en cuenta las conclusiones. También se elabora la línea base de la arquitectura del sistema.

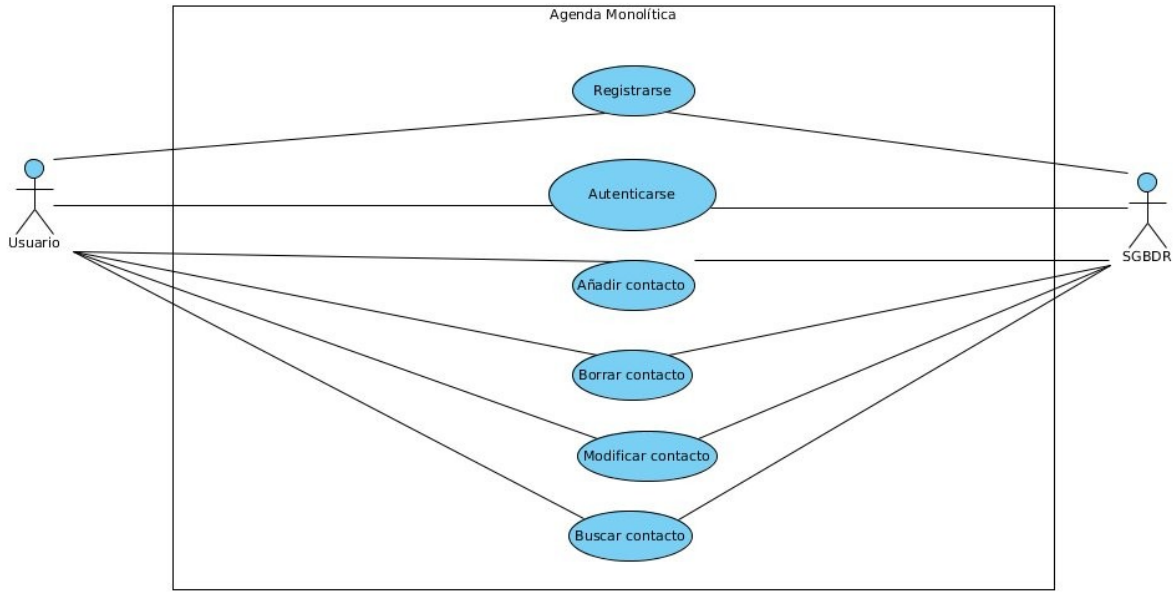
5.1.2 IDENTIFICACIÓN DE REQUISITOS

Tras revisar exhaustivamente el modelo de casos de uso, se llega a la conclusión de que es necesario añadir un nuevo requisito funcional que permita al usuario registrarse en la aplicación eliminando el rol de gestor de credenciales por no considerarlo necesario una vez se añadida esta funcionalidad:

- **RF5 Registrar usuario:** El usuario podrá registrarse en la aplicación, es decir, grabar sus datos de usuario en la base de datos para poder iniciar sesión a partir de ese momento.

5.1.3 MODELADO DE CASOS DE USO

Identificado un nuevo requisito el modelado de casos de uso queda como se aprecia en la figura:



5.1.4 GRUPOS FUNCIONALES

Se incorpora entonces, un nuevo grupo funcional en donde se incluirá únicamente ese nuevo requisito:

Grupo funcional	**F6** Registrar contacto
Casos de uso	Registrar contacto
Requisitos funcionales	**RF.6** Registrar contacto

5.1.5 PRIORIDADES

La tabla de prioridades se actualiza incluyendo el nuevo grupo funcional a tratar.

Prioridad	**Grupo funcional**	**ENUMERACIÓN**
5	Autenticarse	F1
1	Añadir contacto	F2
3	Borrar contacto	F3
3	Modificar contacto	F4
2	Buscar contacto	F5
4	Registrar usuario	F6

5.1.4 PLAN DE ITERACIONES

Tras realizar los anteriores cambios el plan de iteraciones se actualiza a continuación:

Iteración	1
Fase	Elaboración
Objetivos	- Revisar los requisitos funcionales. -Revisar el plan de iteraciones teniendo en cuenta los nuevos posibles requisitos funcionales. - Definir y diseñar la arquitectura.
Artefactos	- Diagrama de casos de uso del sistema. - Plan de iteraciones. - Diseño de la línea base de la arquitectura.

Iteración	2
Fase	Elaboración
Objetivos	- Analizar e identificar los objetos del dominio del sistema. - Diseñar la base de datos. - Diseñar la arquitectura
Artefactos	- Diseño de los objetos del dominio. - Implementación de la línea base de la arquitectura.

Iteración	3
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Añadir contacto” F2.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F2.**

Iteración	4
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Buscar contacto” F5.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F5.**

Iteración	5
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Borrar contacto” F3.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F3.**

Iteración	6
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Modificar contacto” F4.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F4.**

Iteración	7
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Registrar usuario” F6.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F6.**

Iteración	8
Fase	Construcción
Objetivos	- Analizar, diseñar, implementar y realizar las pruebas correspondientes al grupo funcional **“Autenticarse” F1.**
Artefactos	- Modelo de análisis y diseño del grupo funcional **F1.**

Iteración	9
Fase	Transición
Objetivos	- Probar la aplicación
Artefactos	- Resultados de las pruebas**.**

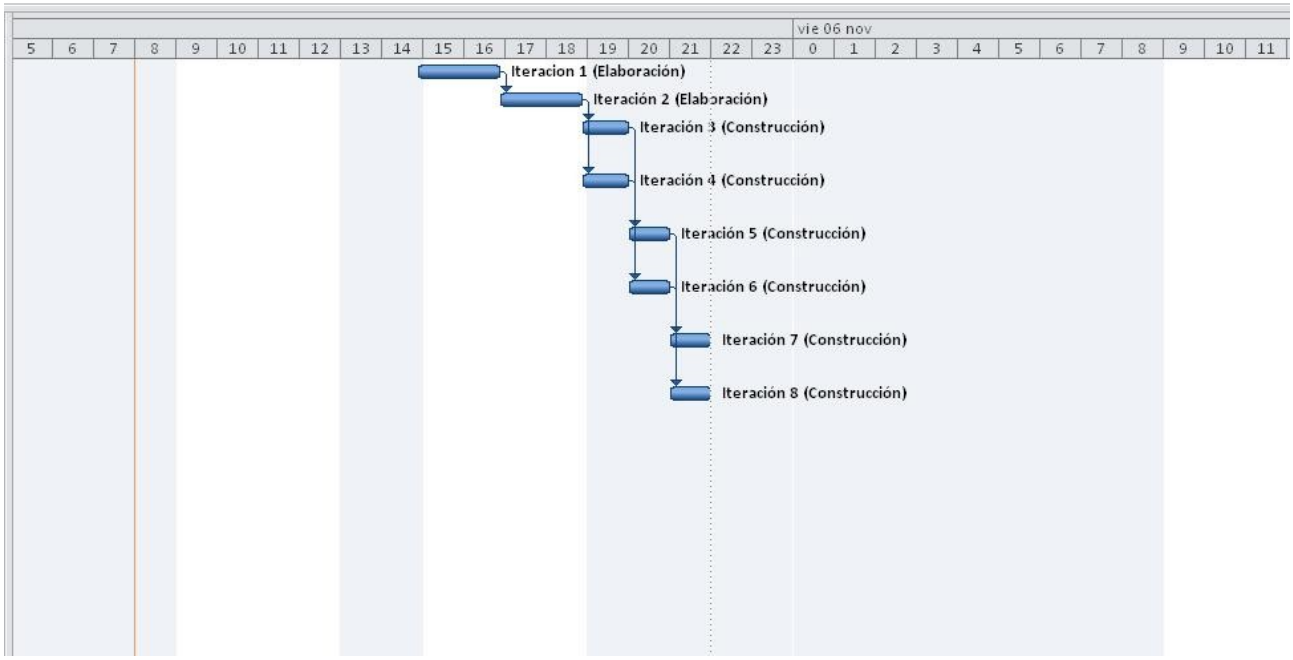
5.1.5 ESTIMACIÓN TEMPORAL Y ASIGNACIÓN DE RECURSOS

Al añadir un nuevo grupo funcional la planificación temporal se ve afectada, pero también se ha considerado que la estimación de la iteración 1 y 2 es demasiado pesimista y se le ha reducido el tiempo de ejecución. Los cambios se muestran a continuación:

Iteración	**Estimación temporal**	**Asignación de recursos**
1	2 horas	4 Desarrolladores
2	2 horas	4 Desarrolladores
3	1 horas	2 Desarrolladores
4	1 hora	2 Desarrolladores
5	1 hora	2 Desarrolladores
6	1 hora	2 Desarrolladores
7	1 hora	2 Desarrolladores
8	1 hora	4 Desarrolladores

5.1.6 DIAGRAMA DE GANTT

Día 5 de noviembre de 2015



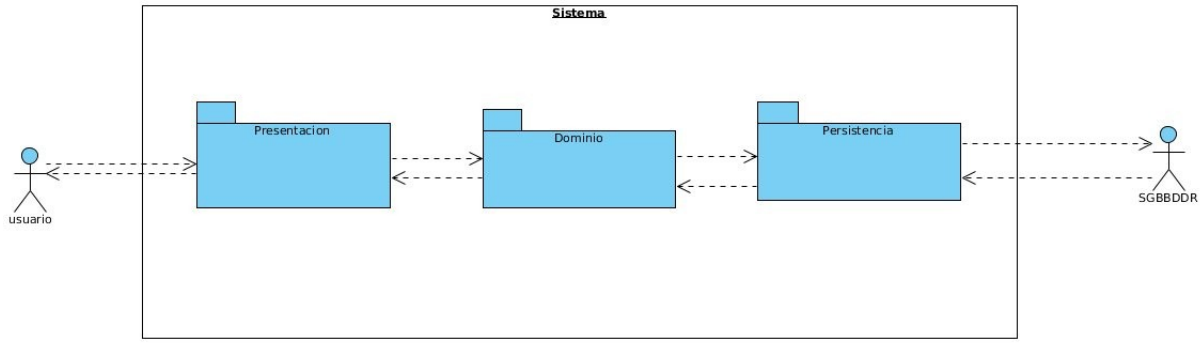
Como se muestra en el diagrama los nuevos cambios consisten en dedicar 2 horas a la iteración 1 y a la iteración 2 con el trabajo de los 4 desarrolladores. Seguidamente se establecerán 2 equipos de trabajo de 2 desarrolladores cada uno. El equipo A realizará en cascada, las iteraciones 3, 5 y 7. El equipo B realizará, paralelamente, las iteraciones 4, 6 y 8.

5.1.7 PLANIFICACIÓN ECONÓMICA

Los cambios considerados en subapartados anteriores implican añadir una hora más de trabajo (13 horas) al desarrollo y por tanto el coste total del proyecto asciende a 1300 euros.

5.1.8 ARQUITECTURA DEL SISTEMA

Se ha decidido que el sistema a desarrollar, tendrá una arquitectura monolítica, todo el sistema se ejecutará en un único computador. Se realizará siguiendo el patrón MVC, modelo, vista, controlador.



El patrón modelo vista controlador se compone de 3 paquetes:

- **Presentación** que incluye las clases dedicadas a mostrar al usuario la metáfora compuesta por toda la información y el procesamiento de la información de las capas subyacentes. Es decir, le ofrece la superficie de contacto a partir de la cual interactuar con el sistema.

- **Dominio** compuesto por todas las clases cuyas responsabilidades consisten en manipular la información y ofrecer al paquete de presentación una abstracción o interpretación de los datos almacenados en la base de datos.

- **Persistencia** este paquete sirve como interfaz entre la base de datos y la capa de dominio. Es decir, se encarga de mapear los datos almacenados en la base de datos para ofrecérselos al dominio de forma que pueda manipularlos con las operaciones propias del lenguaje de programación del sistema, en este caso java. Por otro lado, también realiza el proceso inverso, cuando se encarga de grabar la información, debe interpretar los objetos del dominio y determinar el formato apropiado para guardar esa información según el medio de soporte seleccionado.

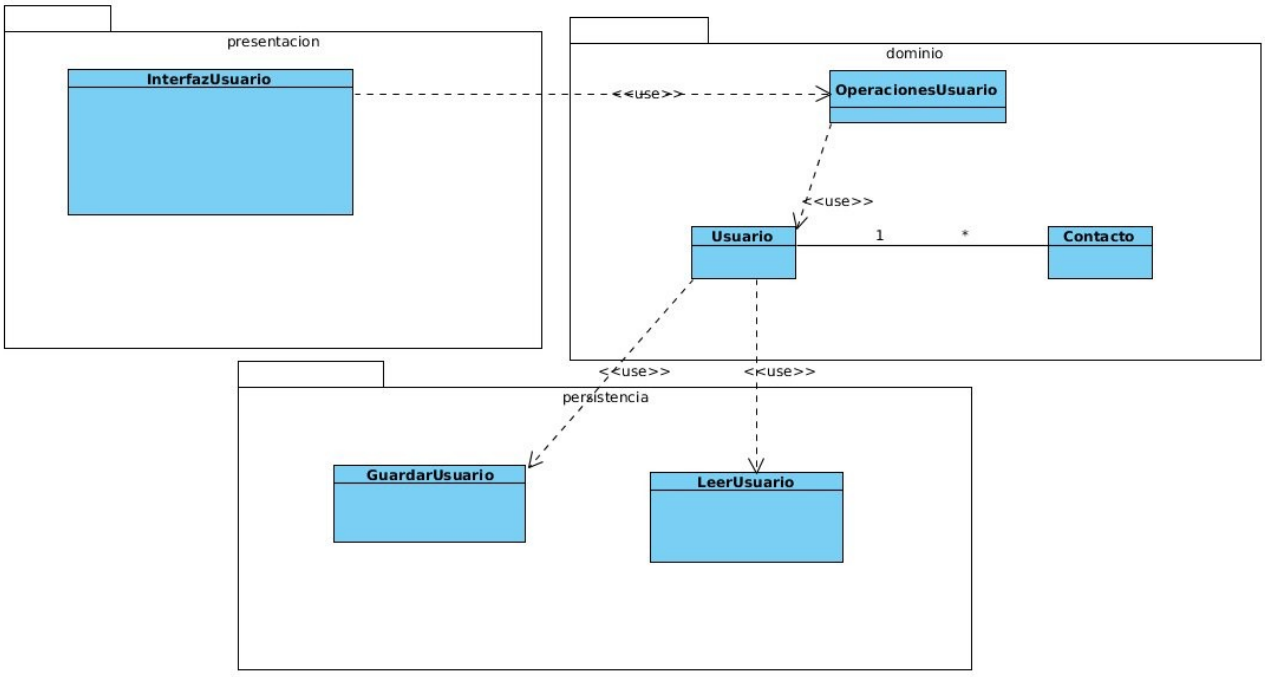
5.2 ITERACIÓN 2

En esta iteración tenemos una idea básica de como debe ser la arquitectura y se lleva a cabo un modelado más preciso de la misma.

Seguidamente se discute sobre el diseño de la base de datos.

Después se otorgan las responsabilidades a las clases de los distintos paquetes identificando qué operaciones deben llevar a cabo y qué atributos representarán su estado.

5.2.1 MODELADO DE LA ARQUITECTURA MEDIANTE DIAGRAMA DE CLASES



Presentación

Este paquete se compone de la clase InterfazUsuario que mostrará un menú con el que el usuario podrá interactuar con la aplicación.

Dominio

En este paquete se crean las clases Usuario y Contacto las cuales representan ambos objetos en la lógica de dominio y las operaciones que se pueden hacer con ambos. La relación de asociación comunica que un usuario puede tener 0, 1 ó más contactos y que todos los contactos siempre pertenecen a un único

usuario.

La clase OperacionesUsuario sirve de interfaz al paquete presentación para interactuar con los objetos de dominio, restringiendo las operaciones que puede hacer con ellos según los requisitos funcionales de la aplicación.

Persistencia

En este paquete coexisten las clases GuardarUsuario y LeerUsuario, ambas permiten a la clase Usuario realizar las operaciones obvias sobre la base de datos.

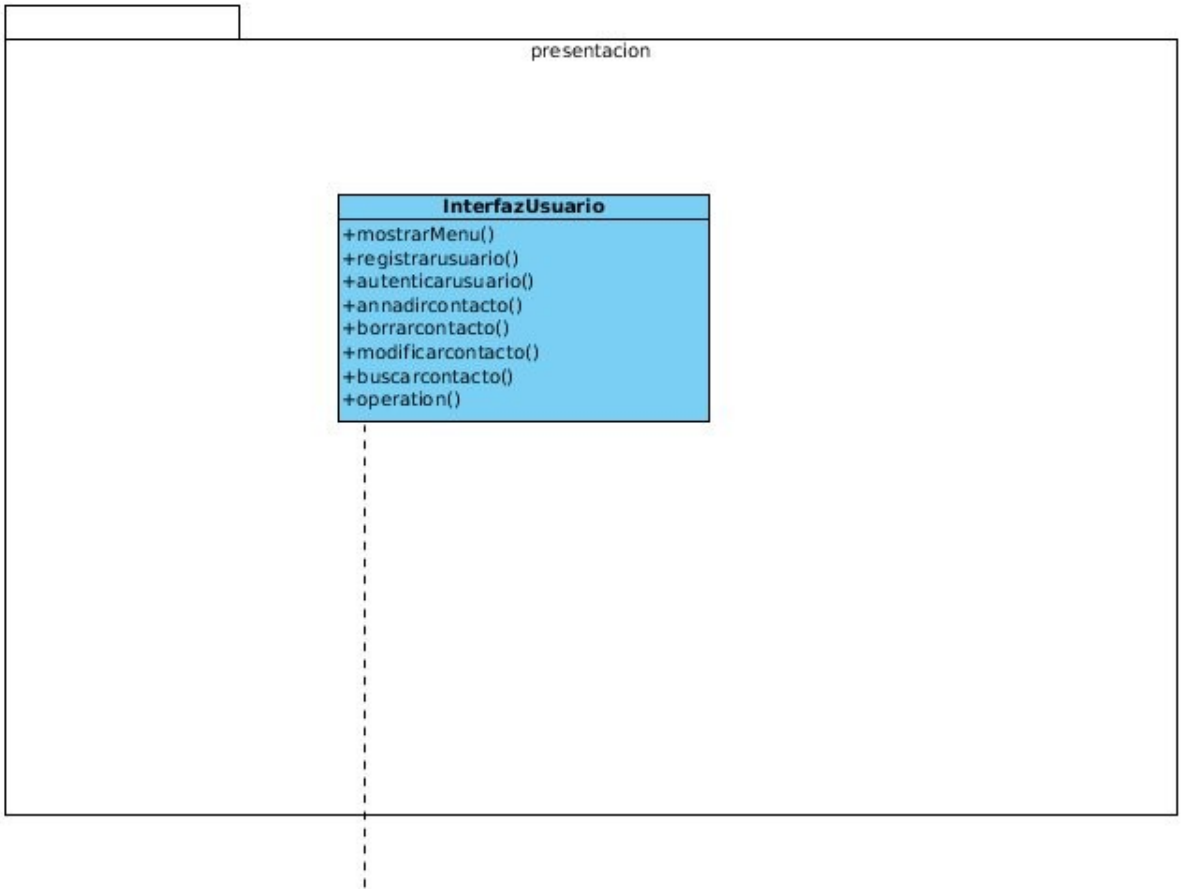
5.2.2 DISEÑO DE LA BASE DE DATOS

Se ha decidido realizar la persistencia de información mediante el uso de ficheros. Por tanto, no se utilizará una base de datos sino que las operaciones de las clases LeerUsuario y GuardarUsuario interactuarán directamente con el sistema de ficheros.

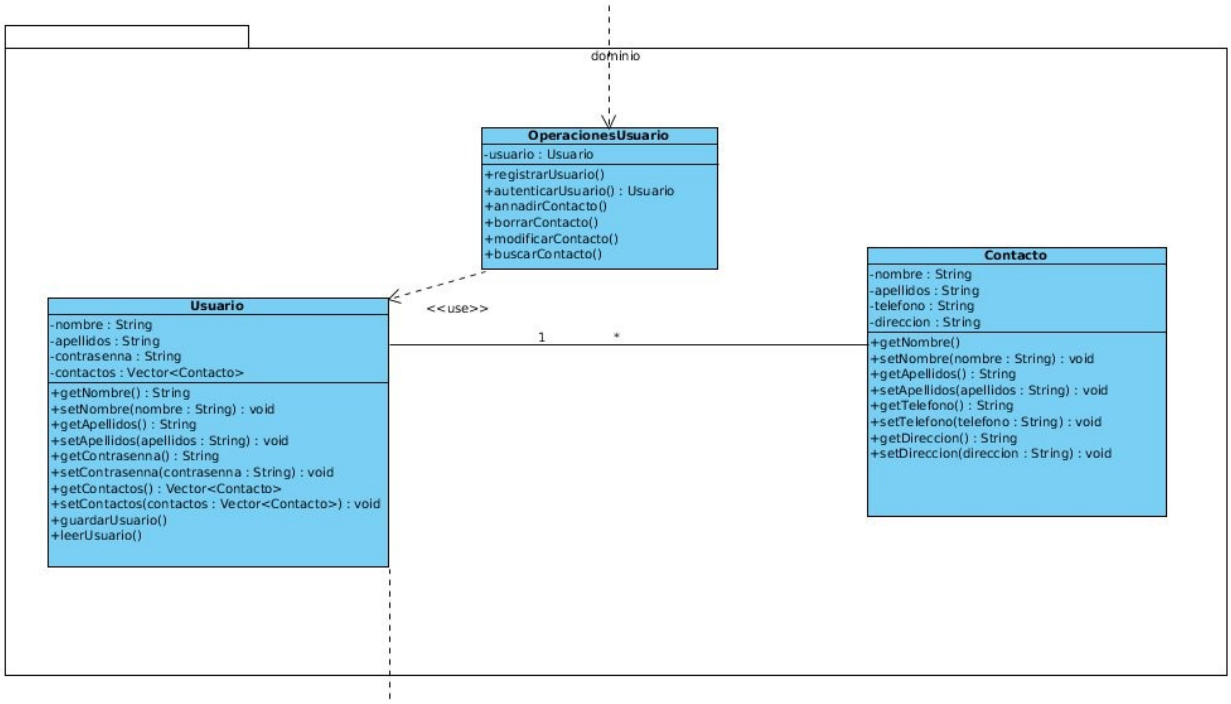
Cada objeto usuario, se guardará íntegramente en un fichero con nombre el nombre de usuario. Ese objeto tendrá toda la información relacionada con ese usuario.

5.2.3 DISEÑO DE LOS DIAGRAMAS DE CLASES DE LOS DISTINTOS PAQUETES

Presentación

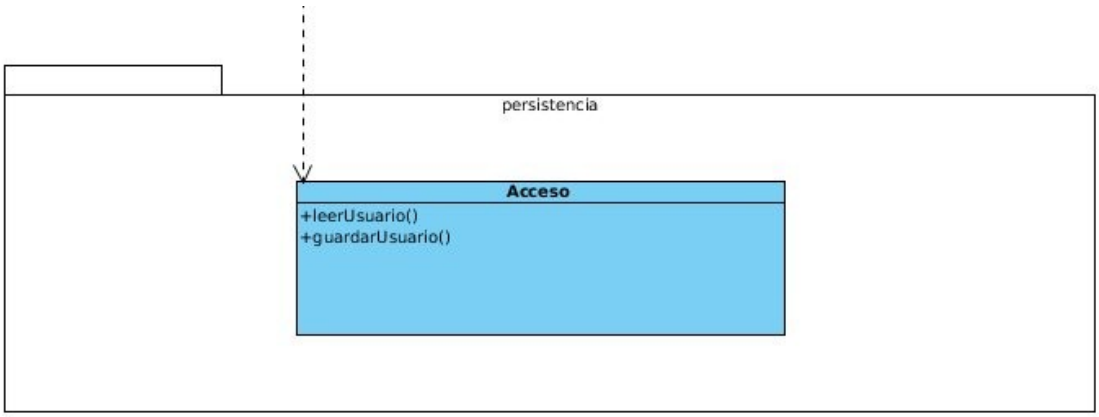


Dominio



Persistencia

Al discutir sobre el diseño de las clases del paquete persistencia, se ha decidido que solo habrá una clase encargada de ambas operaciones.



6. FASE DE CONSTRUCCIÓN

La fase de construcción comienza una vez se ha generado el código correspondiente a la arquitectura de la aplicación, utilizando el plugin de visual paradigm “Visual_Paradigm_NetBeans_Integration”.

A continuación se procede a ejecutar las iteraciones de esta fase que se abordan como mini proyectos de grupos funcionales independientes.

6.1 ITERACIÓN 3

Esta iteración aborda la construcción del grupo funcional “**añadir contacto”**.

6.1.1 ANÁLISIS DE CASO DE USO

El grupo funcional solo contiene un caso de uso:

- “Añadir contacto”

Descripción: El usuario podrá almacenar datos significativos de otra persona.

Precondición: El usuario debe estar logueado y por tanto dado de alta en la aplicación.

Postcondición: Los datos son almacenados en el espacio de datos del usuario y en el medio de almacenamiento, en nuestro caso en un fichero.

Escenario principal:

1. El usuario elige la opción del menú añadir contacto.

2. La aplicación le pide el nombre del contacto, el usuario lo introduce y pulsa entrar.

3. La aplicación le pide los apellidos del contacto, el usuario lo introduce y pulsa entrar.

4. La aplicación le pide el teléfono del contacto, el usuario lo introduce y pulsa entrar.

5. La aplicación le pide la dirección del contacto, el usuario lo introduce y pulsa entrar.

Escenario alternativo

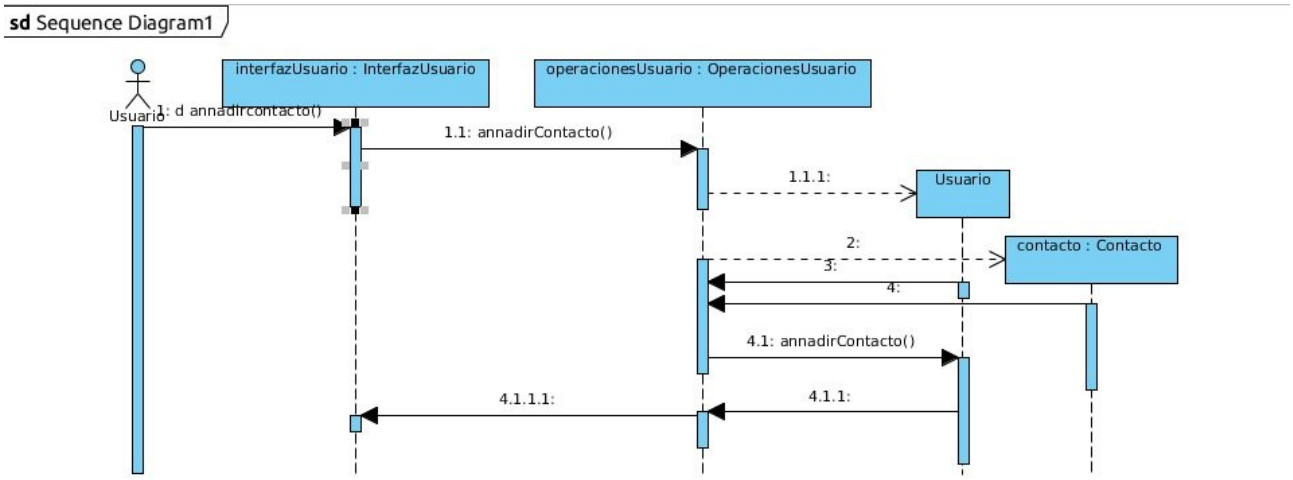
1. El usuario elige la opción añadir contacto sin estar logueado en la aplicación.

2. La aplicación devuelve un mensaje de error informando de la situación.

Para que el grupo funcional sea totalmente funcional también debe abordarse en esta iteración la persistencia de los datos.

6.1.2 DISEÑO

A partir del análisis previo se elaboran los diagramas de secuencia del caso de uso analizado.



6.1.3 IMPLEMENTACIÓN

En la interfaz de usuario

```
case 2:
                        if(u!=null){
                            annadircontacto();
                            System.out.println("Annadido");
                        }
                        else{
                            System.out.println("No se ha logueado "
                                    + "en el sistema");
                        }

                        break;
```

OperacionesUsuario.java

```
	public static void annadirContacto(Usuario u,Contacto c) {
		u.annadirContacto(c);
		u.guardarUsuario();
	}
```

Usuario.java

```
    public void annadirContacto(Contacto c){
		contactos.add(c);
	  }


	public void guardarUsuario() {
		Acceso.guardarUsuario(this);
	}
```

La persistencia de los datos

Este es el primer grupo funcional que hace uso de la persistencia de los datos y por tanto se aborda en esta iteración pero será utilizado también en las siguientes.

La clase **Usuario.java** tendrá los siguientes métodos para guardar o leer un usuario.

```
	public void guardarUsuario() {
		Acceso.guardarUsuario(this);
	}

	public static Usuario leerUsuario(String nombre,String contrasenna) {
		Usuario u=Acceso.leerUsuario(nombre);
		return contrasenna.equals(u.getContrasenna())?u:null;
	}
```

Estos métodos hacen de métodos estáticos de la clase acceso.

La clase acceso se muestra a continuación.

Acceso.java

```
    public class Acceso {

      public static Usuario leerUsuario(String nombre) {
            Usuario r=new Usuario("","","");
             try{
                ObjectInputStream ois = new ObjectInputStream(
                        new FileInputStream("usuarios/"+nombre));

                r =(Usuario) ois.readObject();

                ois.close();
            }
            catch (Exception e){
            }
            return r;
      }

      public static void guardarUsuario(Usuario u) {
             try{
                ObjectOutputStream oos = new ObjectOutputStream(
                        new FileOutputStream("usuarios/"+u.getNombre()));
                    oos.writeObject(u);
                oos.close();
            } catch (Exception e){
                e.printStackTrace();
            }
      }

}
```

6.1.4 PRUEBAS

El paquete de pruebas debe comprobar:

- El usuario intenta añadir un contacto sin estar logueado y obtiene un mensaje de error.

- El usuario puede añadir un contacto en la agenda.

6.2 ITERACIÓN 4

Esta iteración aborda la construcción del grupo funcional **“buscar contacto”**.

6.2.1 ANÁLISIS DE CASO DE USO

El grupo funcional solo contiene un caso de uso:

- “Buscar contacto”

Descripción: El usuario podrá visualizar los datos previamente almacenados en la aplicación correspondientes a un contacto en cuestión.

Precondiciones:

1. El usuario debe estar logueado y por tanto dado de alta en la aplicación.

Postcondición: Los datos correspondientes al contacto consultado, son mostrados en pantalla.

Escenario principal:

1. El usuario elige la opción del menú buscar contacto.

2. La aplicación le pide el nombre del contacto, el usuario lo introduce y pulsa entrar.

3. La aplicación muestra al usuario los datos del contacto buscado.

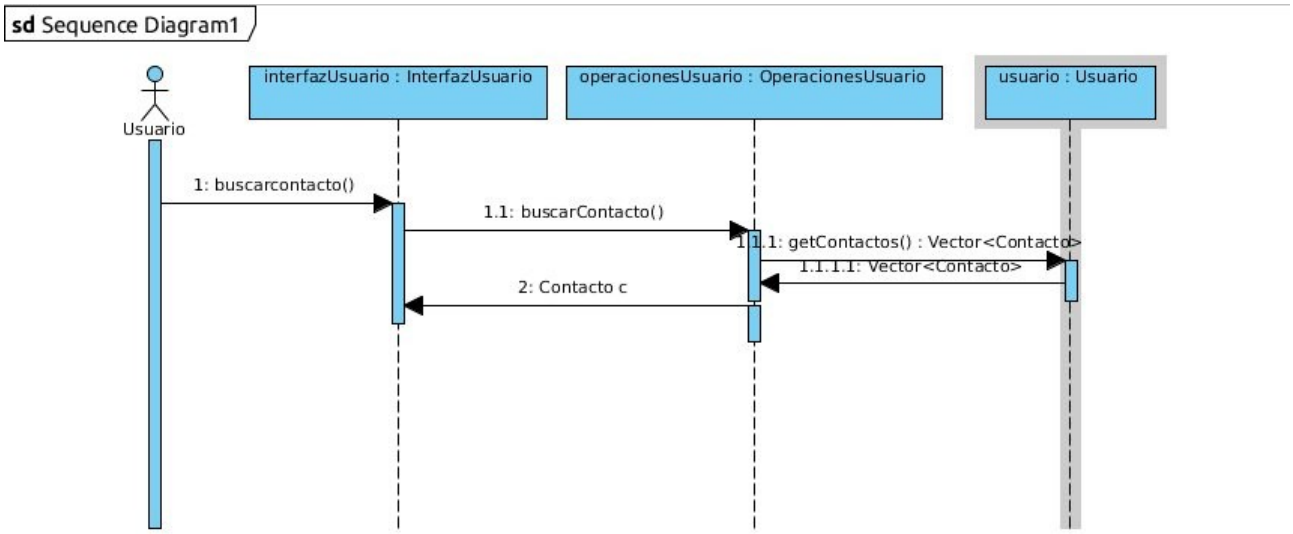
Escenario alternativo

1. El usuario elige la opción buscar contacto sin estar logueado en la aplicación.

2. La aplicación devuelve un mensaje de error informando de la situación.

6.2.2 DISEÑO

A partir del análisis previo se elaboran los diagramas de secuencia del caso de uso analizado.



6.2.3 IMPLEMENTACIÓN

En la interfaz de usuario

```
    case 3:
                        if(u!=null){
                           System.out.println(buscarcontacto());
                        }
                        else{
                            System.out.println("No se ha logueado "
                                    + "en el sistema");
                        }
                        break;
```

OperacionesUsuario.java

```
public static Contacto buscarContacto(Usuario u,String nombre) {
            Vector<Contacto> contactos=u.getContactos();
            Contacto r=null;
            for(Contacto c:contactos ){
                if(c.getNombre().equals(nombre)){
                    r=c;
                }
            }
            return r;
      }
```

Usuario.java

```
      public Vector<Contacto> getContactos() {
            return this.contactos;
      }
```

6.2.4 PRUEBAS

El paquete de pruebas debe comprobar:

- El usuario intenta buscar un contacto sin estar logueado y obtiene un mensaje de error.

- El usuario puede buscar un contacto en la agenda satisfactoriamente.

6.3 ITERACIÓN 5

Esta iteración aborda la construcción del grupo funcional **“borrar contacto”**.

6.3.1 ANÁLISIS DE CASO DE USO

El grupo funcional solo contiene un caso de uso:

- “Borrar contacto”

Descripción: El usuario podrá eliminar los datos previamente almacenados en la aplicación correspondientes a un contacto en cuestión.

Precondiciones:

1. El usuario debe estar logueado y por tanto dado de alta en la aplicación.

Postcondición: Los datos correspondientes al contacto consultado, son borrados del espacio de datos del usuario.

Escenario principal:

1. El usuario elige la opción del menú borrar contacto.

2. La aplicación le pide el nombre del contacto, el usuario lo introduce y pulsa entrar.

3. La aplicación muestra al usuario un mensaje informando al usuario del resultado de la operación.

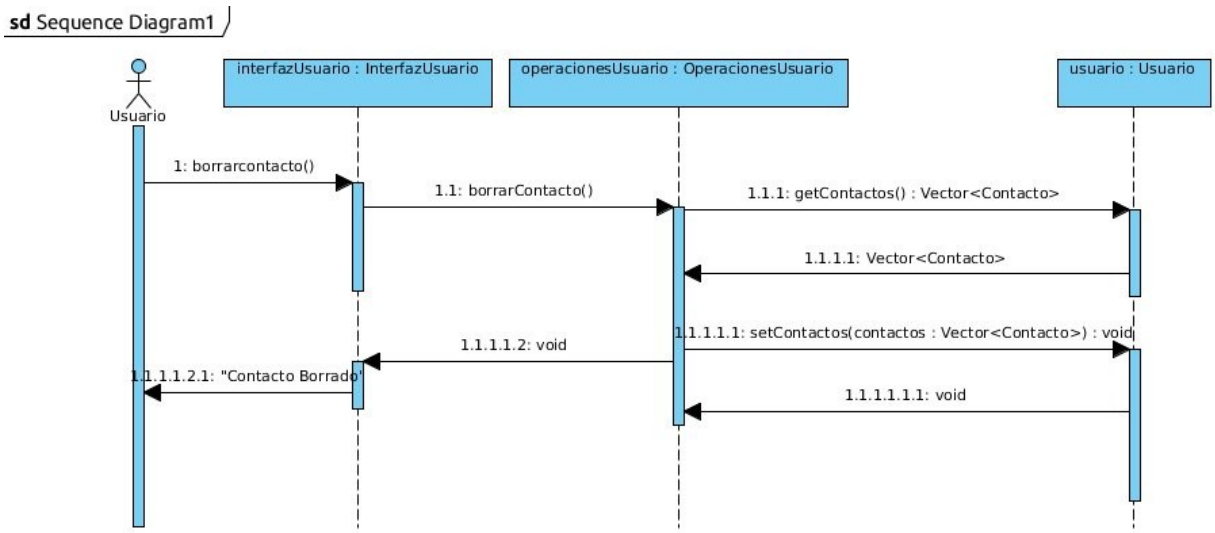
Escenario alternativo

1. El usuario elige la opción borrar contacto sin estar logueado en la aplicación.

2. La aplicación devuelve un mensaje de error informando de la situación.

6.3.2 DISEÑO

A partir del análisis previo se elaboran los diagramas de secuencia del caso de uso analizado.



6.3.3 IMPLEMENTACIÓN

En la interfaz de usuario

```
                        case 4:
                         if(u!=null){
                            borrarcontacto();
                             System.out.println("Contacto borrado ");
                         }
                         else{
                             System.out.println("No se ha logueado "
                                     + "en el sistema");
                         }

                         break;
```

OperacionesUsuario.java

```
    public static void borrarContacto(Contacto c,Usuario u) {
          Vector<Contacto> contactos=u.getContactos();
          Vector<Contacto> r=new Vector();
          for(Contacto vc:contactos ){
              if(!c.equals(vc)){
                  r.add(vc);
              }
          }
          u.setContactos(r);
          u.guardarUsuario();
    }
```

6.3.4 PRUEBAS

El paquete de pruebas debe comprobar:

- El usuario intenta borrar un contacto sin estar logueado y obtiene un mensaje de error.

- El usuario puede borrar un contacto en la agenda satisfactoriamente.

6.4 ITERACIÓN 6

Esta iteración aborda la construcción del grupo funcional **“modificar contacto”**.

6.4.1 ANÁLISIS DE CASO DE USO

El grupo funcional solo contiene un caso de uso:

- “Modificar contacto”

Descripción: El usuario podrá modificar los datos previamente almacenados en la aplicación correspondientes a un contacto en cuestión.

Precondiciones:

1. El usuario debe estar logueado y por tanto dado de alta en la aplicación.

Postcondición: Los datos correspondientes al contacto consultado, son modificados del espacio de datos del usuario.

Escenario principal:

1. El usuario elige la opción del menú modificar contacto.

2. La aplicación le pide el nombre del contacto, el usuario lo introduce y pulsa entrar.

3. La aplicación le pide los nuevos apellidos del contacto, el usuario lo introduce y pulsa entrar.

4. La aplicación le pide el nuevo teléfono del contacto, el usuario lo introduce y pulsa entrar.

5. La aplicación le pide la nueva dirección del contacto, el usuario lo introduce y pulsa entrar.

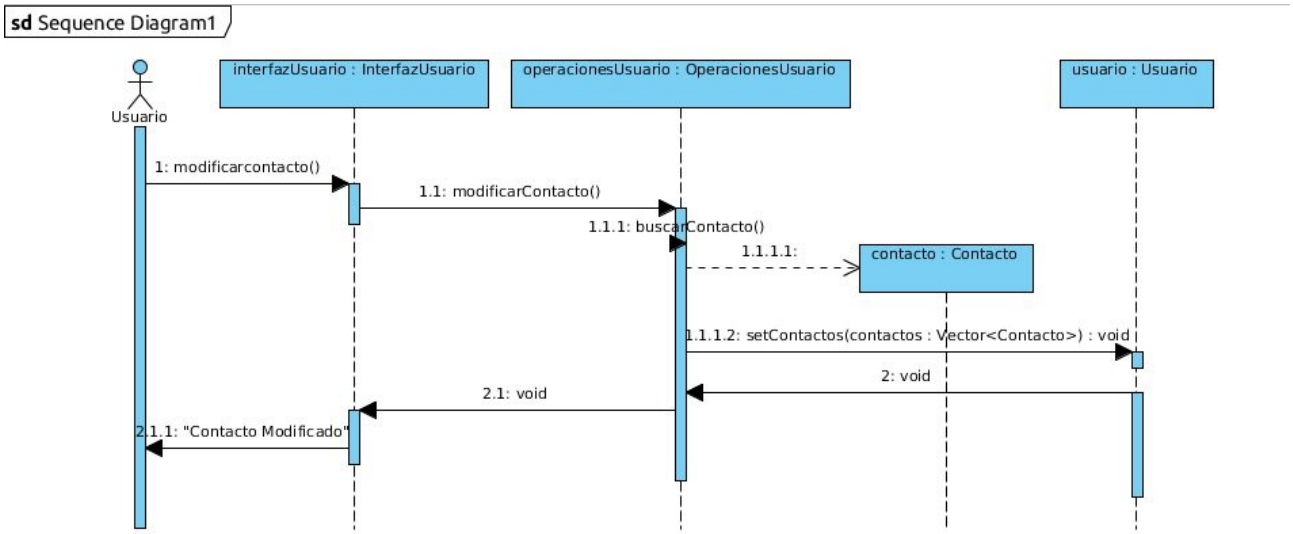
Escenario alternativo

1. El usuario elige la opción modificar contacto sin estar logueado en la aplicación.

2. La aplicación devuelve un mensaje de error informando de la situación.

6.4.2 DISEÑO

A partir del análisis previo se elaboran los diagramas de secuencia del caso de uso analizado.



6.4.3 IMPLEMENTACIÓN

<u>En la interfaz de usuario</u>

```
                                case 5:
                    if(u!=null){
                        modificarcontacto();

                    }
                    else{
                        System.out.println("No se ha logueado "
                                + "en el sistema");
                    }
                    break;
```

```
                    public void modificarcontacto() {
            System.out.println("Nombre nuevo del contacto ");
            String nombre=Leer.cadena();
            System.out.println("Apellidos nuevo del contacto ");
            String apellidos=Leer.cadena();
            System.out.println("Telefono nuevo del contacto ");
            String telefono=Leer.cadena();
```

```
            System.out.println("Direccion nueva del contacto");
            String direccion=Leer.cadena();
            Contacto nuevo=new Contacto(nombre,apellidos,telefono,direccion);
            OperacionesUsuario.modificarContacto(u, nuevo);
      }
```

OperacionesUsuario.java

```
      public static void modificarContacto(Usuario u,Contacto nuevo) {
            Contacto c=OperacionesUsuario.buscarContacto(u, nuevo.getNombre());
            c=nuevo;
            u.guardarUsuario();
      }
```

6.4.4 PRUEBAS

El paquete de pruebas debe comprobar:

- El usuario intenta modificar un contacto sin estar logueado y obtiene un mensaje de error.

- El usuario puede modificar un contacto en la agenda satisfactoriamente.

6.5 ITERACIÓN 7

Esta iteración aborda la construcción del grupo funcional **“registrar usuario”**.

6.5.1 ANÁLISIS DE CASO DE USO

El grupo funcional solo contiene un caso de uso:

- “Registrar usuario”

Descripción: El usuario podrá registrarse en la aplicación.

Precondiciones:

Postcondición: Se crea un nuevo fichero en el sistema de archivos que almacena los datos del usuario registrado.

Escenario principal:

1. El usuario elige la opción del menú registrase.

2. La aplicación le pide el nombre de usuario, el usuario lo introduce y pulsa entrar.

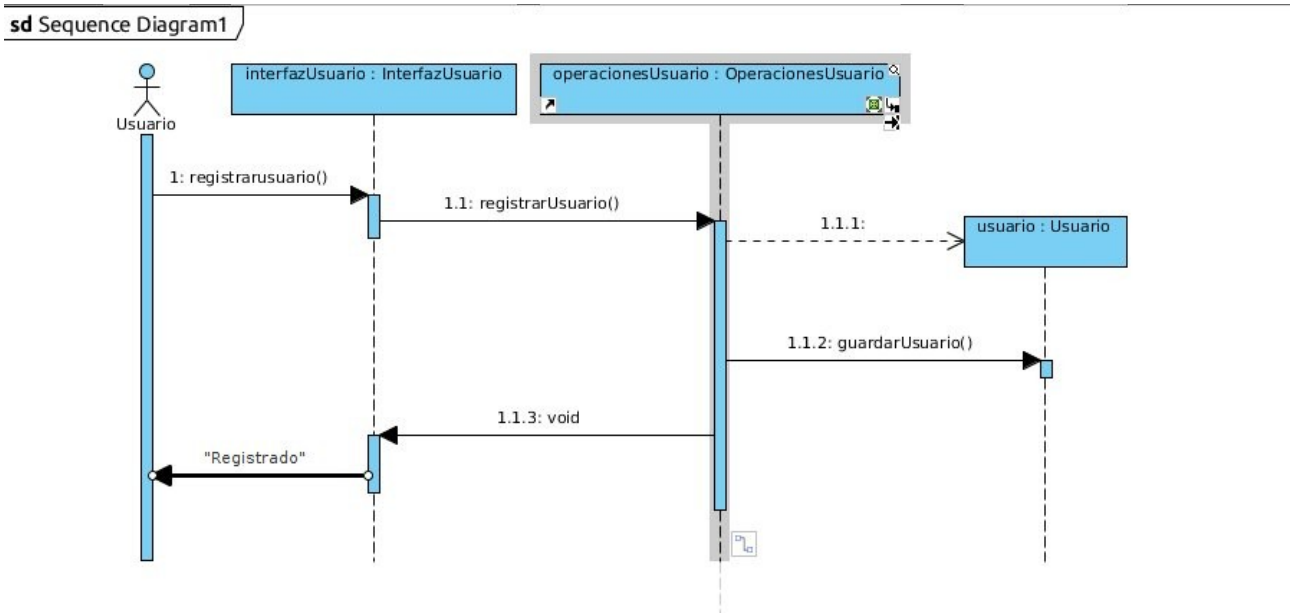
3. La aplicación le pide los apellidos de usuario, el usuario lo introduce y pulsa entrar.

3. La aplicación le pide la contraseña, el usuario la introduce y pulsa entrar.

4. La aplicación le devuelve un mensaje informándole del resultado de la operación.

6.5.2 DISEÑO

A partir del análisis previo se elaboran los diagramas de secuencia del caso de uso analizado.



6.5.3 IMPLEMENTACIÓN

En la interfaz de usuario

```
                    case 6:
                        registrarusuario();
                        System.out.println("Usuario registrado con exito");
                        break;
```

```
    public void registrarusuario() {
          System.out.println("==Usuario nuevo==");
          System.out.println("Nombre de usuario");
          String nombre=Leer.cadena();
          System.out.println("Apellidos del usuario");
          String apellidos=Leer.cadena();
          System.out.println("contrasenna");
          String contrasenna=Leer.cadena();
          System.out.println(contrasenna);
          OperacionesUsuario.registrarUsuario(nombre,apellidos,contrasenna);

    }
```

OperacionesUsuario.java

```
        public static void registrarUsuario(String nombre,
            String apellidos,String contrasenna) {
        Usuario u=new Usuario(nombre, apellidos,contrasenna);
        u.guardarUsuario();
    }
```

6.6 ITERACIÓN 8

Esta iteración aborda la construcción del grupo funcional **“autenticar usuario”**.

6.6.1 ANÁLISIS DE CASO DE USO

El grupo funcional solo contiene un caso de uso:

- “Autenticar usuario”

Descripción: El usuario podrá autenticarse en la aplicación.

Precondiciones:

1. El usuario debe estar registrado en la aplicación.

Postcondición: Los datos de usuario pasan a formar parte del modelo de dominio y este puede tener acceso al resto de funcionalidades de la agenda.

Escenario principal:

1. El usuario elige la opción del menú **autenticarse**.

2. La aplicación le pide el nombre de usuario, el usuario lo introduce y pulsa entrar.

3. La aplicación le pide la contraseña, el usuario la introduce y pulsa entrar.

4. La aplicación devuelve el mensaje **Autenticado** informando de que el usuario se encuentra logueado en la aplicación.

Escenario alternativo

1. El usuario elige la opción del menú **autenticarse**.

2. La aplicación le pide el nombre de usuario, el usuario lo introduce y pulsa

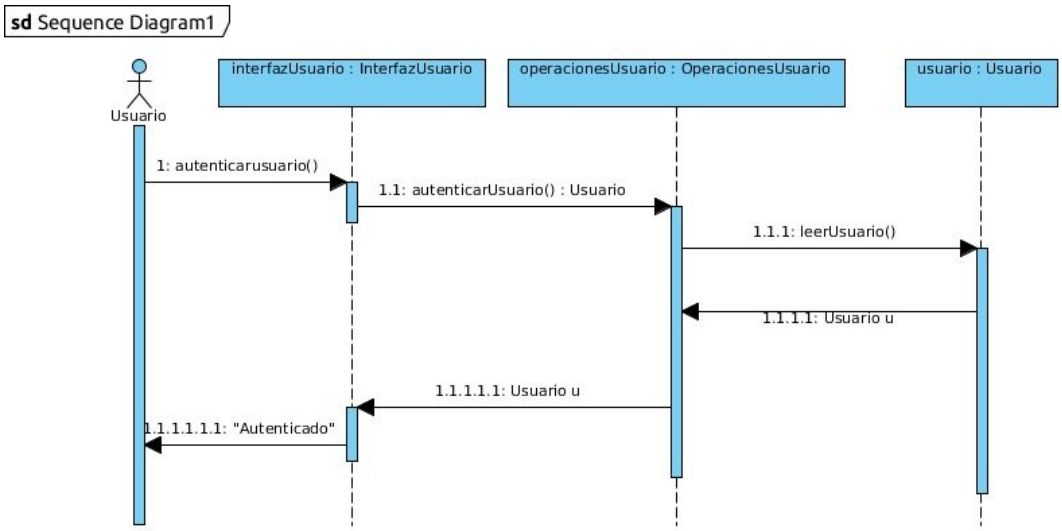
entrar.

3. La aplicación le pide la contraseña, el usuario la introduce y pulsa entrar.

6. La aplicación devuelve el mensaje **Nombre de usuario o contrasenna erroneos** informando de que el usuario ha cometido un error introduciendo sus datos o no se encuentra registrado en la aplicación.

6.6.2 DISEÑO

A partir del análisis previo se elaboran los diagramas de secuencia del caso de uso analizado.



6.6.3 IMPLEMENTACIÓN

En la interfaz de usuario

```
                    case 1:
                        u=autenticarusuario();
                        if(u==null){
                            System.out.println("Nombre de usuario o contrasenna
"
                                    + "erroneos");
                        }
                        else{
                            System.out.println("Autenticado");
                        }
                        break;
```

```
                              public Usuario autenticarusuario() {

             System.out.println("Nombre de usuario");
            String nombre=Leer.cadena();
            System.out.println("Contrasenna");
            String contrasenna=Leer.cadena();
            Usuario usuario=OperacionesUsuario.autenticarUsuario(nombre,
contrasenna);
            return usuario;
      }
```

OperacionesUsuario.java

```
      public static Usuario autenticarUsuario(String nombre,String contrasenna)
{
            Usuario r= Usuario.leerUsuario(nombre,contrasenna);
            return r;
      }
```

6.6.4 PRUEBAS

El paquete de pruebas debe comprobar:

- El usuario intenta autenticarse sin estar registrado y obtiene un mensaje de error.

- El usuario puede registrarse en la agenda satisfactoriamente.

7 REFERENCIAS

- *PFC_LauraDelRío.pdf*

- *caps_4_y_5_del_pfc_almudena_buitrago.pdf*

- *caps4_y5_-_memoria_pfc_isabel_bermejo.pdf*

- *El proceso unificado de desarrollo de software,*

    Ivar Jacobson, Grady Booch, James Rumbaugh