ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

Arrastrar y Soltar

Drag'N Drop

Realizado por
Francisco Jesús Domínguez Ruiz
Tutorizado por
Eduardo Guzmán de los Riscos
Departamento
Lenguajes y Ciencias

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Diciembre 2015

Fecha defensa:
El Secretario del Tribunal

RESUMEN: Drag & Drop es una aplicación web diseñada para la creación de problemas a partir de piezas, en la que al profesor se le plantea una nueva posibilidad de evaluar a sus alumnos.

La aplicación web servirá como un entorno dedicado a la elaboración de preguntas y respuestas. Para responder a dichas preguntas, se proporcionan unos elementos llamados “piezas” al alumno que se encargará de utilizar para construir su respuesta.

A su vez, el profesor al elaborar la pregunta establecerá la solución ideal del problema y el conjunto de “piezas” que los alumnos podrán utilizar para crear las suyas propias.

El alumno al terminar la solución de un problema, la enviará al servidor. Este se encargará de evaluarla y comparar la solución del alumno con la solución ideal propuesta por el profesor.

Finalmente el profesor será el encargado de examinar el ejercicio y ajustar la calificación, ya sea aceptando la que propone el sistema o indicando una propia.

Palabras claves: Bootstrap, Cliente y servicio, Restful, CSS, HTML, Input, JavaScript, jQuery, JSON, JSP y Servlet, Pieza, Problema, MongoDB, UML, SCRUM, Solución.

ABSTRACT: Drag’N Drop is based on a web application designed to create problems with pieces, which offers new possibilities to evaluate their students.

The web application would use as an environment used to make problems and solutions. To resolve these solutions, they will be provided some elements called “pieces” to the student which will build a solution.

At the same time, when the teacher are going to develop problems he will have to make an ideal solution for the problem and giving the students the pieces to making their solutions.

When the student would finish his solution, he will send it to the server and the server will correct the solution comparing it with the teacher’s solution.

Finally, the teacher will be in charge to correct the student’s solution and decide if it deserve and upper or lower mark, or let it as it is.

KeyWord: Bootstrap, Client and service, Restful, CSS, HTML, Input, JavaScript, jQuery, JSON, JSP y Servlet, Piece, Problem, MongoDB, UML, SCRUM, Solution.

Contenido

Introducción

Este trabajo de fin de grado es un proyecto dedicado a la creación de problemas cuyo usó servirá a los profesores para crear preguntas con “piezas” las cuales los alumnos podrán resolver con las opciones que les proporcionará el profesor.

Motivación

Hoy en día hay poco tipos de exámenes por ordenador que no sean los clásicos tests y con este nuevo software se abre una nueva puerta a los tipos de exámenes.

Este proyecto tuvo la idea de darle un nuevo enfoque a los tipos de exámenes por ordenador, para que el profesor pueda ser capaz de construir preguntas que sean contestadas por piezas.

Este programa puede ser utilizado para cualquier tipo de problema que requiera un camino o un conjunto de pasos como una solución. Esto puede servir para poder abstraer algoritmos, soluciones de navegación de páginas web, etc.

Objetivos del proyecto

El objetivo principal de este proyecto es poder implementar este tipo de preguntas en exámenes virtuales para poder darle más variedad a estos tipos de prueba.

Esta aplicación está destinada a profesores que quieran realizar cuestiones en formularios web a sus alumnos.
La aplicación web servirá como un entorno dedicado a la elaboración de preguntas y respuestas. Para responder a dichas preguntas, se proporcionan unos elementos llamados “bloques“ al alumno que se encargará de construir para su respuesta.

A su vez el profesor al elaborar la pregunta establecerá la solución ideal del problema y el conjunto de “bloques” que los alumnos podrán utilizar.

Una vez que el profesor realice el cuestionario, se almacenará la respuesta para su posterior comparación con las respuestas del alumnado.

Cuando todos los alumnos han realizado los cuestionarios, se recogen todas las respuestas para su posterior evaluación automática.

Para la corrección de cada ejercicio, se compararán las respuestas de los alumnos con la solución dada por el profesor. La nota se establece mediante un porcentaje de similitud.

Antecedentes

Para esta aplicación nos hemos inspirado en la herramienta “Scratch” la cual se utiliza principalmente para el aprendizaje de los niños y enseñarles las ideas básicas de la programación (bucles, condición, etc.).

La gran semejanza entre Scratch y nuestra aplicación es el uso de unión de Bloques o piezas como método de resolución de problemas. Utilizamos la tecnología arrastrar y soltar (Drap N’ Drog) que nos da la librería JQuery de JavaScript para poder tener la misma similitud con esta aplicación que está hecha con Flash.

Scratch está siendo utilizado en algunos centros para enseñar a programar debido a la facilidad de su interfaz que resulta muy fácil de utilizar y sencilla de entender.

Organización del documento

Punto 1: En este punto de introducción explicamos los objetivos principales del proyecto y definimos un poco el ámbito donde lo queremos implementar. También hablamos de tecnologías de misma características y un poco de la motivación.

Punto 2: En este capítulo hablaremos de las tecnologías que vamos a utilizar para el desarrollo de la aplicación

Punto 3: En este punto hablaremos del análisis y especificación de los requisitos del programa. Hablaremos de los actores que participan, de los “stakeholders” y principalmente de los requisitos funcionales.

Punto 4: En este punto se explicará el diseño de la aplicación. Se mostrarán los diagramas de clases sacados según los requisitos del apartado anterior, diagramas de secuencia que explicara un poco el comportamiento del servidor y finalmente un diagrama de red donde se verá cómo estará desplegada la aplicación.

Punto 5: En este punto se mostrará el resultado de la implementación. Mostraremos imágenes del código y explicando más detalladamente su funcionalidad.

Punto 6: En este último punto para finalizar hablaremos de los objetivos cumplidos y de las posibles ampliaciones que puede tener este proyecto, así como una pequeña bibliografía la cual se podrá mirar todo el contenido en internet que ha sido utilizado para desarrollar este trabajo de fin de grado.

Tecnologías y herramientas utilizadas

Para este proyecto hemos utilizado una gran variedad de tecnologías y API que nos han permitido realizarlo de una manera sencilla y rápida:

NetBeans: Como entorno de programación hemos utilizado NetBeans. Es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de automatización de proyectos basado en Ant, control de versiones y refactoring.

JSP y Servlet : JavaServer Pages (JSP) es una tecnología que ayuda a los desarrolladores de software a crear páginas web dinámicas basadas en HTML, XML, entre otros tipos de documentos.

Para desplegar y ejecutar JavaServer Pages, se requiere un servidor web compatible con contenedores servlet como Apache Tomcat o Jetty.

JAX-RS: JAX-RS: Java API for RESTful Web Services es una API del lenguaje de programación Java que proporciona soporte en la creación de servicios web de acuerdo con el estilo arquitectónico *Representational State Transfer* (REST).1 JAX-RS usa anotaciones, introducidas en Java SE 5, para simplificar el desarrollo y despliegue de los clientes y puntos finales de los servicios web.

MongoDB: MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Morphia: API que utilizaremos para poder trabajar mejor con las clases. Morphia permite crear una instancia de una clase a un objeto document Mongo y viceversa para guardar y leer de la base de datos.

jQuery: jQuery es una biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

jQuery-ui: jQuery UI es una biblioteca de componentes para el framework jQuery que le añaden un conjunto de plug-ins, widgets y efectos visuales para la creación de aplicaciones web. Cada componente o módulo se desarrolla de acuerdo a la filosofía de jQuery

Java: Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es

permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

JavaScript: Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador webpermitiendo mejoras en la interfaz de usuario y páginas web dinámicas4 aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS).

Bootstrap: Es un framework o conjunto de herramientas de software libre para diseño de sitios y aplicaciones web adaptables (responsive). Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript opcionales adicionales.

Especificación y Análisis

Para esta sección vamos a realizar un documento SRS (Software Requirements Specification) en el cual vamos a explicar detalladamente los motivos, objetivos, entorno y alcance, actores y participantes y requisitos del sistema.

Motivos

El motivo principal para realizar este documento es explicar detalladamente una descripción de la aplicación. En este documento serán explicados los requisitos del sistema así como los actores principales, los objetivos y el entorno de este.

Objetivos

Los objetivos de este proyecto se basan en 3 puntos importantes:

- Que los profesores sean capaz de tener una herramienta web la cual puedan gestionar sus problemas y sean capaz de corregirlos.
- Que los alumnos sean capaz de generar sus soluciones según la paleta de piezas que les proporcione el profesor en sus problemas.
- Que el sistema sea capaz de dar una nota previa según un algoritmo de corrección que calcule una nota según el parecido entre la solución del alumno y la del profesor.

Lista de actores

Profesor	Usuario que será el encargado de generar un problema con una solución
Alumno	Usuario que tendrá que resolver los problemas del profesor

Requisitos funcionales

R01. El sistema deberá permitir que el profesor pueda llevar una gestión de los problemas.

R02.El sistema deberá tener un servicio que devuelva todas las soluciones de un problema.

R03. El sistema deberá tener un servicio que devuelva todos los problemas de un profesor.

R04. El sistema deberá tener un servicio que devuelva todos los problemas de una Asignatura.

R05. El sistema deberá corregir la solución del Alumno a través de un algoritmo de comparación.

R06. El sistema deberá permitir que el alumno pueda gestionar sus soluciones.

R07. El sistema deberá tener una gestión de usuarios.

R08. El sistema deberá tener una gestión de asignaturas.

R09. El sistema deberá tener desarrollado un sistema de reconocimiento de entrada a la aplicación (Log in).

Requisitos no funcionales

RN01. Los datos deberán ser guardados regidos por la ley de protección de datos.

RN02. Lenguaje de programación Java.

RN03. Utilizar MongoDB como base de datos.

Diagrama de casos de usos

En el siguiente diagrama podremos observar los casos de usos que corresponderán según al cliente web del Profesor o al cliente web del Alumno.

El cliente web del profesor tiene asignado los casos de usos principalmente referentes a la gestión de problemas (Ejemplo: Gestión de Problemas, listar Problemas de un profesor) y a la corrección manual de la solución de un Alumno.

El cliente web del alumno será el encargado de llevar la gestión de las soluciones del alumno. Principalmente tiene asignado los casos de usos de Eliminar solución, insertar solución y leer solución. Después, tiene asignado todos aquellos casos de usos que tengan que ver con la lectura de problemas.

Estos dos clientes tendrán acceso a un método de autenticación (Login).

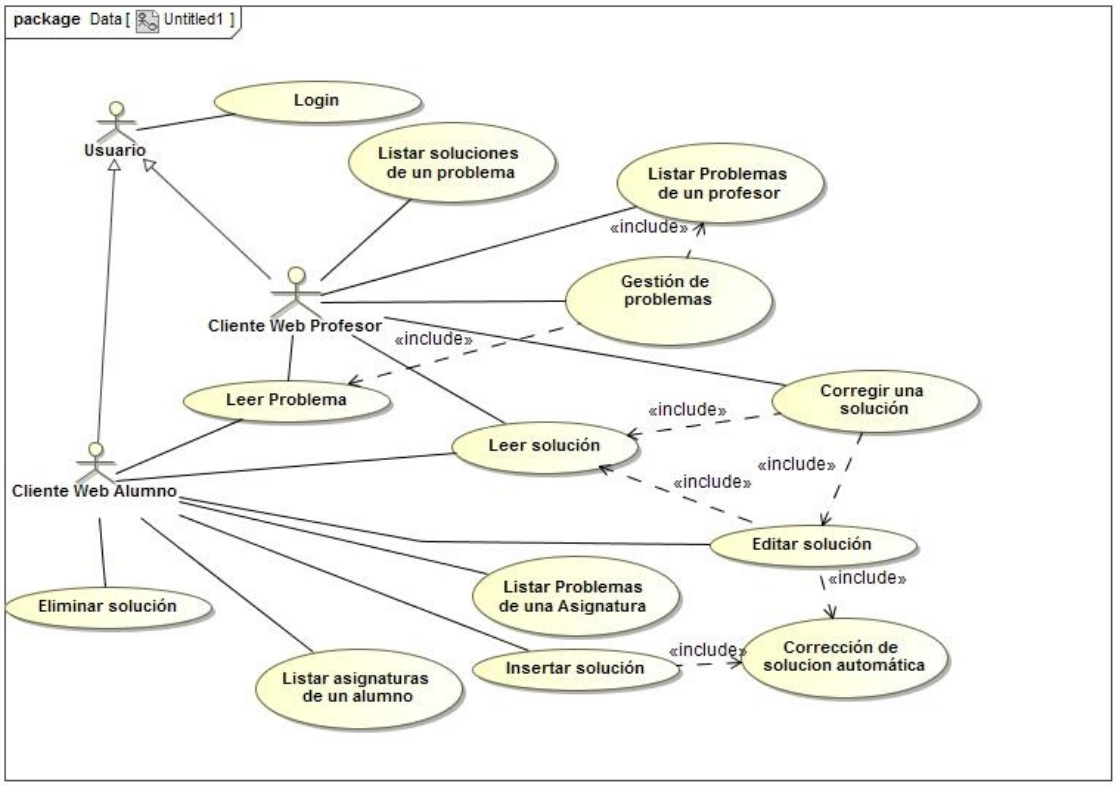

Ilustración 1. Diagrama de casos de usos de la aplicación

Diseño del sistema

Diagramas de clases

Una vez hecho el análisis y la especificación de la aplicación pudimos sacar varias cosas:

1. Según los requisitos del sistema podemos llegar a la conclusión que vamos a tener seis entidades fuertes: Solución, Problema, Profesor, Asignatura, Alumno y Usuario. También podemos observar que tendríamos las entidades Pieza e Input, pero estas sin embargo se utilizarían para las composiciones de problemas y soluciones.
2. No se ha especificado un actor que actúe de administrador para la gestión de Usuarios, Profesores, Alumnos y Asignaturas.

De las entidades sacadas del primer apartado hemos hecho el siguiente diagrama de clases con sus correspondientes relaciones de cada entidad:

- Un alumno tiene un usuario y un usuario le corresponde a un alumno.
- Un profesor tiene un usuario y un usuario le corresponde a un profesor.
- Un alumno puede estar matricula muchas asignaturas y una Asignatura puede tener muchos alumnos.
- Un alumno puede tener muchas soluciones, pero una solución solo puede pertenecer a un alumno.
- Un profesor puede impartir muchas asignaturas, pero una asignatura solamente puede ser impartida por un profesor.
- Un profesor puede crear muchos problemas, pero un problema solo puede ser creado por un profesor.
- Un problema puede tener muchas soluciones, pero una solución solo pertenece a un problema.

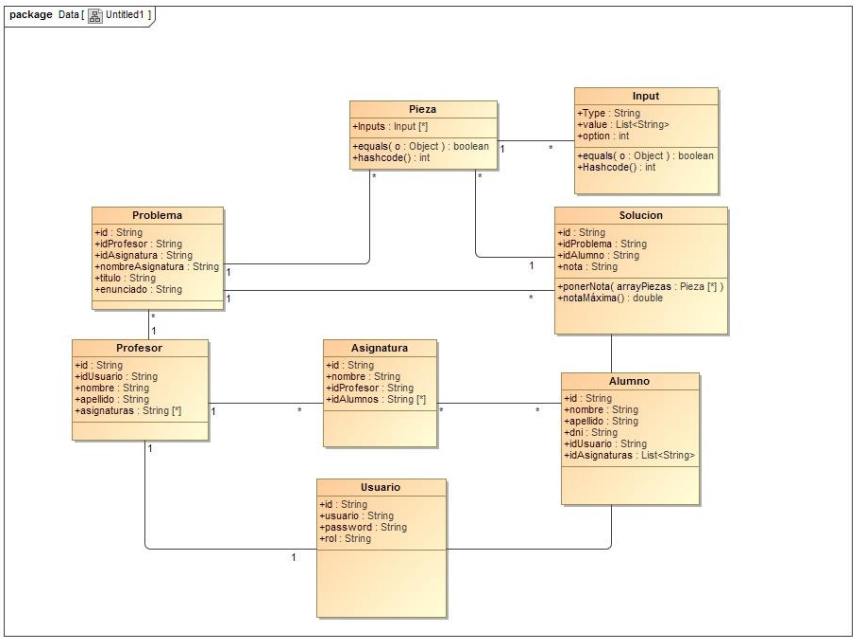

Ilustración 2. Diagrama de clases

Después de hacer el diagrama de clases con sus relaciones, el siguiente paso es diseñar los servicios. Los servicios van a ser aquellos métodos que serán llamados por los clientes web. Lo hemos separado según la entidad que mejor le corresponda.

Todas las entidades van a tener tres métodos de gestión básicos (CRUD). Teniendo estos métodos como bases, se le añadirán aquellos que cumplan con los requisitos anteriormente extraídos en el apartado de especificación y análisis:

- **ServiciosUsuario:** En esta clase tendremos los dos métodos de autenticación, un para el alumno y otro para el profesor.
- **ServiciosAsignatura:** Aquí tendremos un método para buscar las asignaturas que está dando un alumno y otro método para ver las asignaturas que está impartiendo un profesor.
- **ServiciosSolucion:** Aquí tendremos un método para cambiar la nota a una solución y un método para ver las soluciones que tiene un problema.
- **ServiciosProblema:** Por último, en esta clase tendremos tres métodos de lectura; uno para ver los problemas que tiene un alumno en total, otro para ver los problemas que ha creado un profesor y un último método para ver todos los problemas que tiene una asignatura.

Las clases ServiciosAlumno y ServiciosProfesor solamente tendrían los tres métodos de gestión.

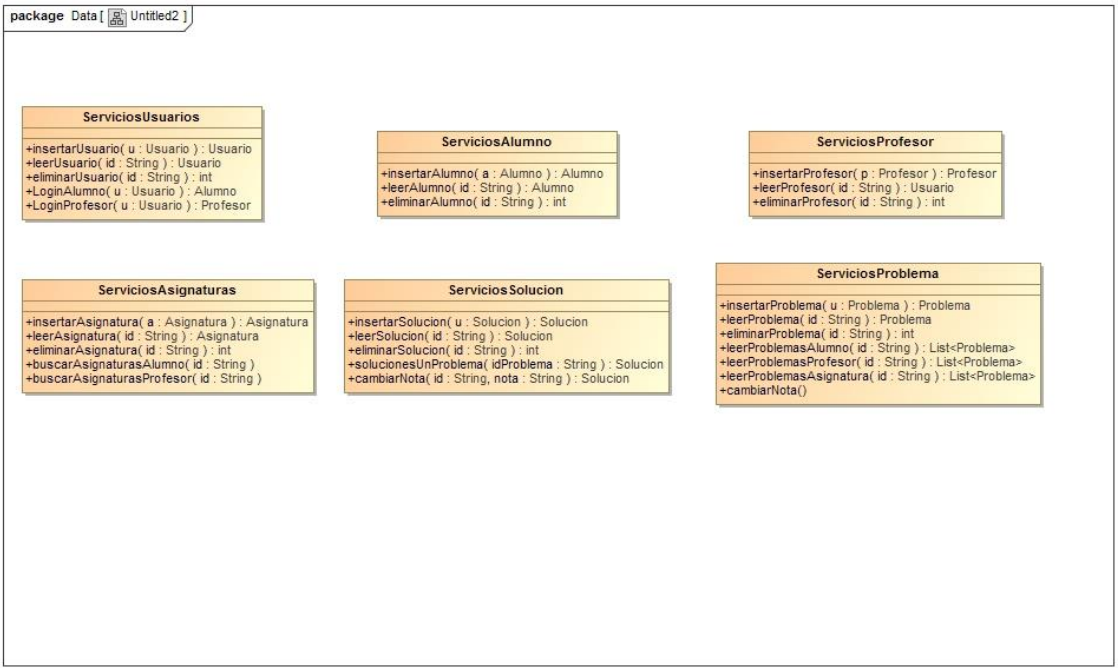

Ilustración 3 Diagrama de clases de los servicios

Con estos dos diagramas tendríamos representado un modelo E/R y los servicios web que tendremos que generar, pero aún falta una clase que haga de mediadora con la base de datos y podamos utilizarla como persistencia de datos.

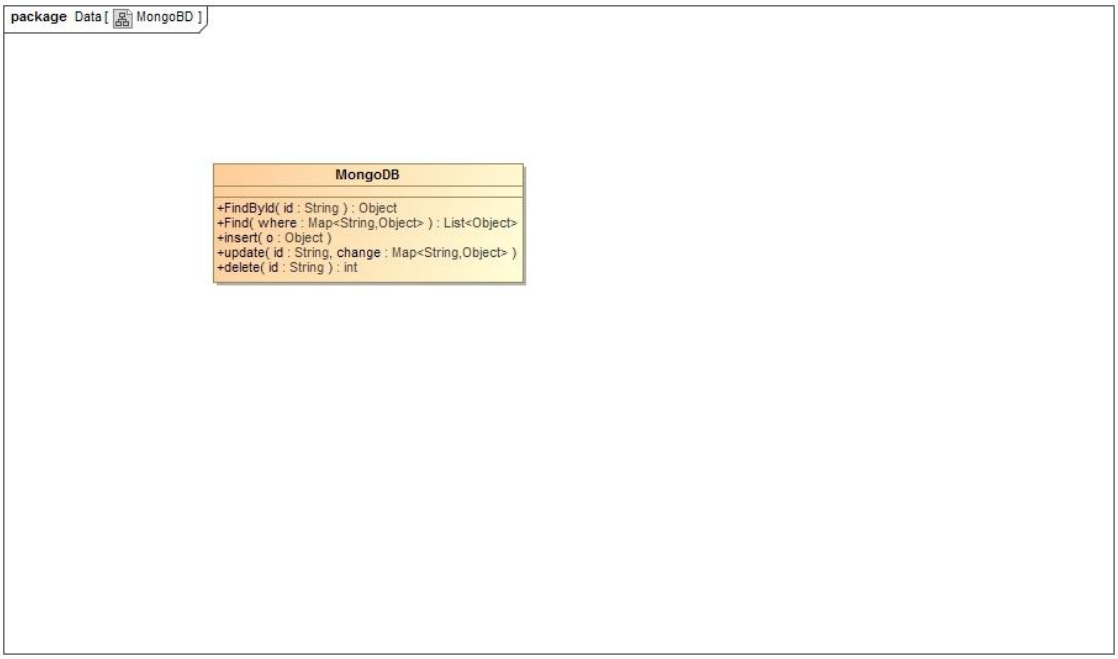

Ilustración 4 Clase MongoDB

Para esto creamos una clase MongoDB que será la encargada de hacer las sentencias básicas de Select, Insert, Update y Delete.

Diagrama de secuencias

Ahora que tenemos diseñado los diagramas de clases, debemos diseñar diagramas de secuencias que expliquen el desarrollo de los casos de usos. Además debemos dar una explicación breve de que hace o que se quiere hacer con este desarrollo, una pre-condición y pos-condición y la prioridad del caso de uso.

Primero vamos a exponer tres diagramas de secuencias que van a representar como ejemplo la gestión de las entidades:

Título	Insertar Problema
Descripción	Insertar un problema en la base de datos
Pre-condición	Estar autenticado en la aplicación
Post-condición	El problema tendrá que estar insertado en la base de datos
Prioridad	Alta
Escenario principal	
1. El cliente web envía una petición al servicio con el problema a insertar. 2. El sistema recibe el problema y lo manda a la clase intermedia MongoDB para que lo inserte. 3. La clase MongoDB inserta el problema en la base de datos.	
Escenario alternativo	
2.b El sistema recibe el problema pero con un id, eso significara que se intentara hacer un update. 3.b La clase MongoDB llamará al mismo método de guardado pero este al ver que el objeto tiene un id lo interpretara	
Clases de análisis	
A. Clases de entidad	Problema
B. Clases de servicios	ServicioProblema
Diagramas de secuencia	

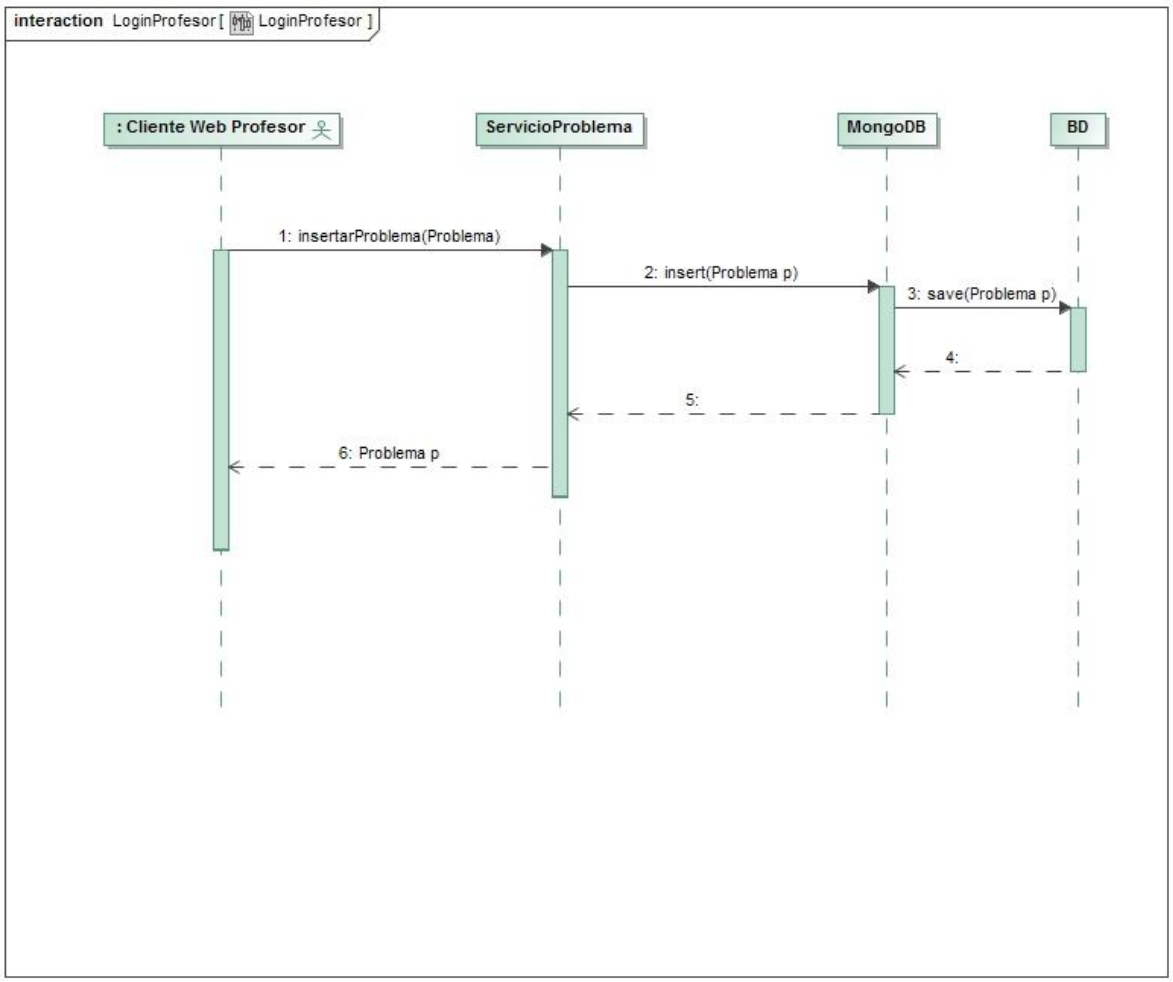

Ilustración 5 Diagrama de secuencia de insertar Problema

Título	Leer Problema
Descripción	Lee un problema de la base de datos.
Pre-condición	Estar autenticado en la aplicación.
Post-condición	El sistema le devolverá al usuario el problema pedido
Prioridad	Alta
Escenario principal	
1. El cliente web envía una petición al servicio con el id del problema que quiere leer. 2. El sistema recibe el id y lo reenvía a la clase MongoDB donde se hará la búsqueda. 3. La clase MongoDB hace la búsqueda en la base de datos y lo envía a la clase ServicioProblema para que esta le devuelva al usuario el problema.	

Escenario alternativo	
3.b En el caso que no exista ese problema se devuelve un objeto nulo.	
Clases de análisis	
A. Clases de entidad	Problema
B. Clases de servicios	ServicioProblema
Diagramas de secuencia	

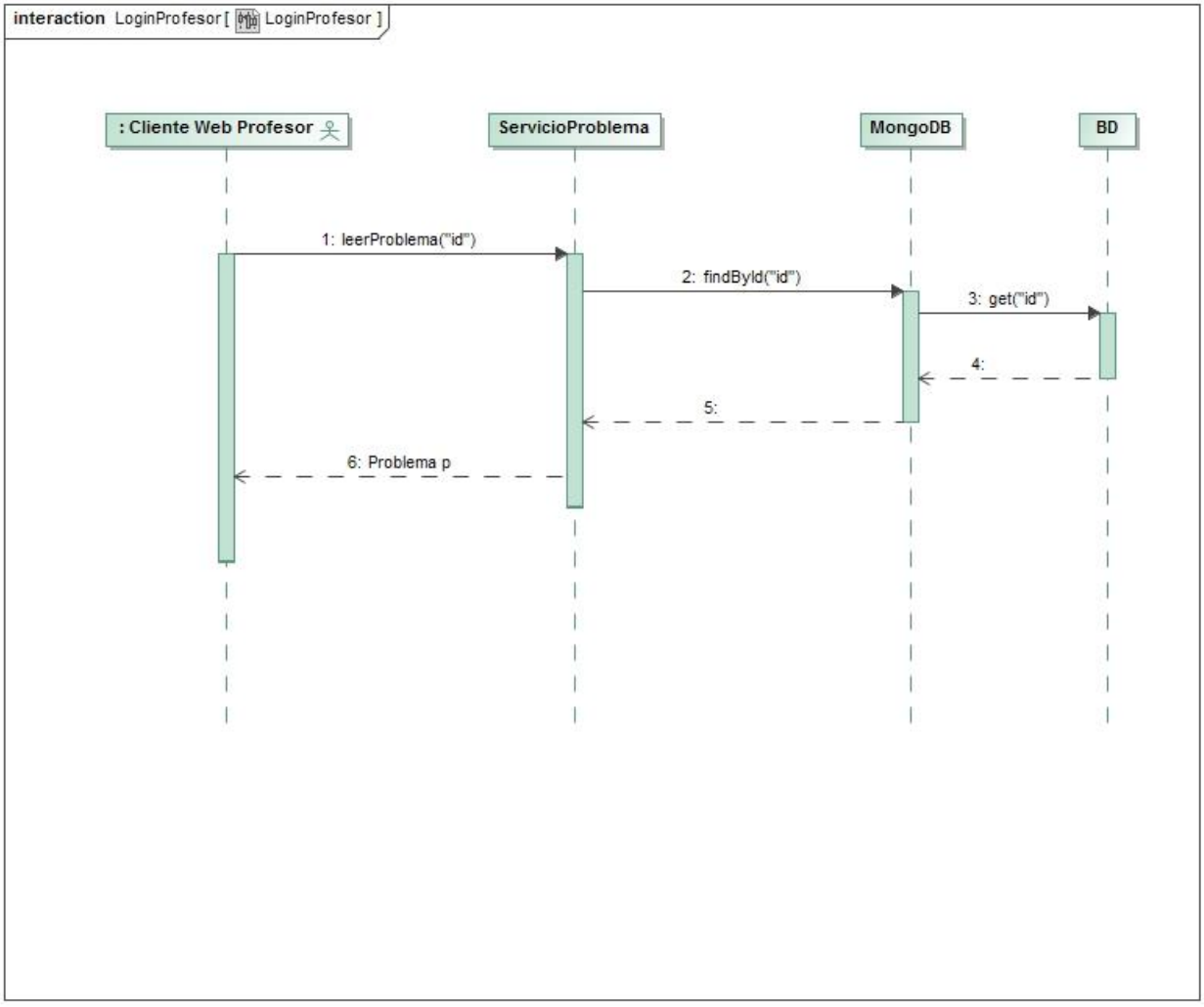

Ilustración 6 Diagrama de secuencia de leer Problema

Título	Eliminar un Problema
Descripción	Elimina un problema de la base de datos.
Pre-condición	Estar autenticado en la aplicación y que el problema que se quiera borrar este en la base de datos.
Post-condición	El problema tendrá que estar eliminado de la base de datos
Prioridad	Alta
Escenario principal	
1. El cliente manda una petición donde lleva el id del problema que se quiere eliminar. 2. El servicio lo recibe y lo envía a la clase MongoDB donde se eliminará el problema. 3. La clase MongoDB se encargará de eliminar el problema que tenga ese id. 4. El servicio envía al cliente un número donde indica el número de líneas que han sido borradas.	
Escenario alternativo	
3.b No existe ningún problema con ese id.	
Clases de análisis	
A. Clases de entidad	Problema
B. Clases de servicios	ServicioProblema
Diagramas de secuencia	

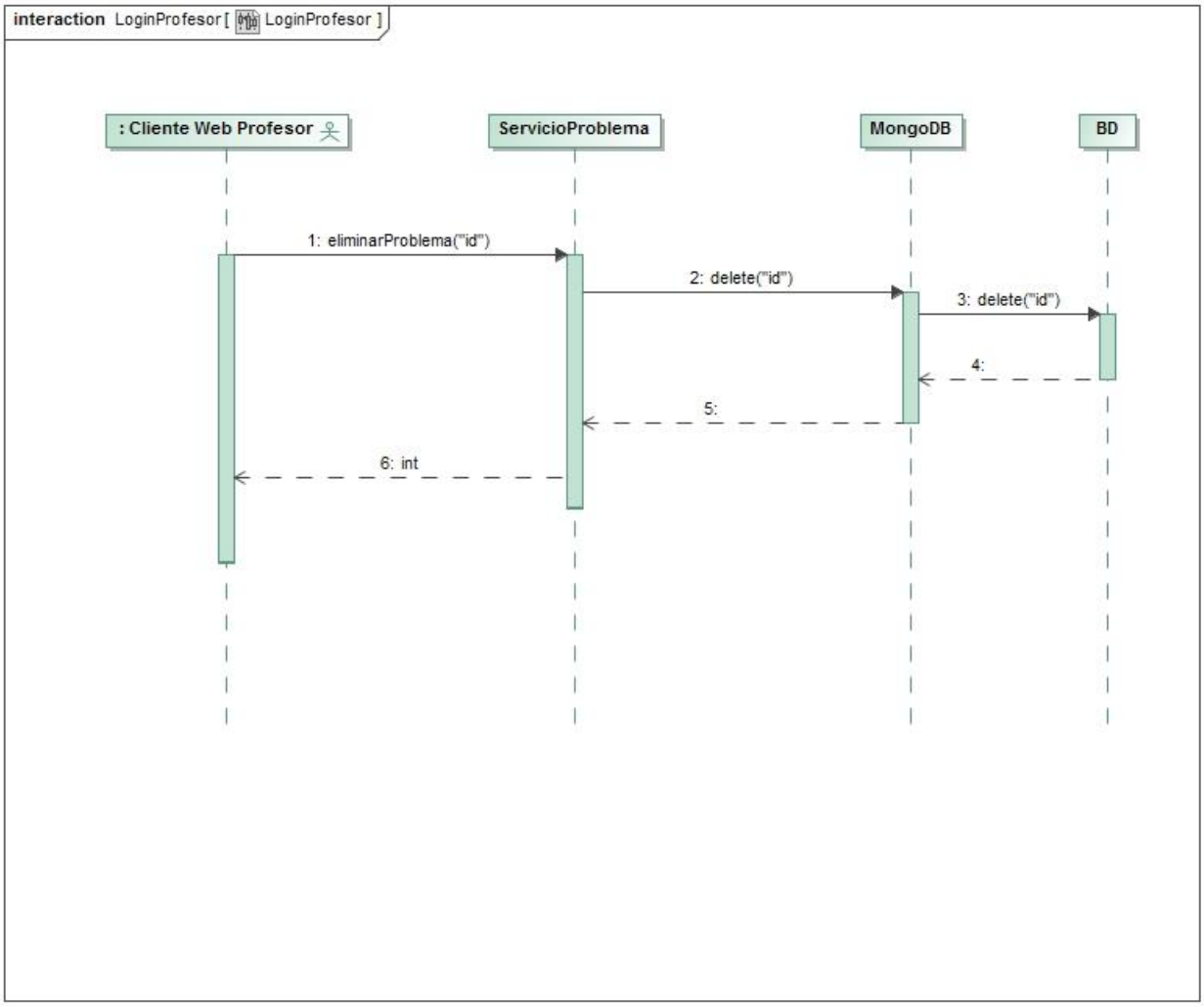

Ilustración 7 Diagrama de secuencia de eliminar Problema

Así es como actúan los métodos de gestión de cada entidad. Como se puede ver son secuencias muy simples. A continuación vamos a detallar los demás diagramas de secuencias y clasificarlos según a los servicios que pertenecen.

ServicioUsuario

Título	Login Profesor
Descripción	Autenticación de un profesor dentro del sistema
Pre-condición	El usuario debe existir y estar relacionado con un profesor. No debe estar autenticado.

Post-condición	El usuario estará autenticado
Prioridad	Alta
Escenario principal	
1. El cliente manda una petición donde lleva el nombre de usuario y la contraseña del profesor. 2. El servicio lo recibe y encripta la contraseña para buscar en la base de datos si existe un usuario con ese nombre de usuario y contraseña. 3. La clase MongoDB comprueba si existe ese usuario. 4. Una vez comprobado la validez del usuario, se busca el profesor que este relacionado con ese id. 5. La clase MongoDB busca ese profesor con ese id de Usuario. 6. El servicio devuelve al cliente el objeto del profesor.	
Escenario alternativo	
3.b no existe ningún usuario con ese nombre de usuario y contraseña. 5.b no existe ningún profesor con ese id de usuario	
Clases de análisis	
A. Clases de entidad	Usuario, Profesor
B. Clases de servicios	ServicioUsuario
Diagramas de secuencia	

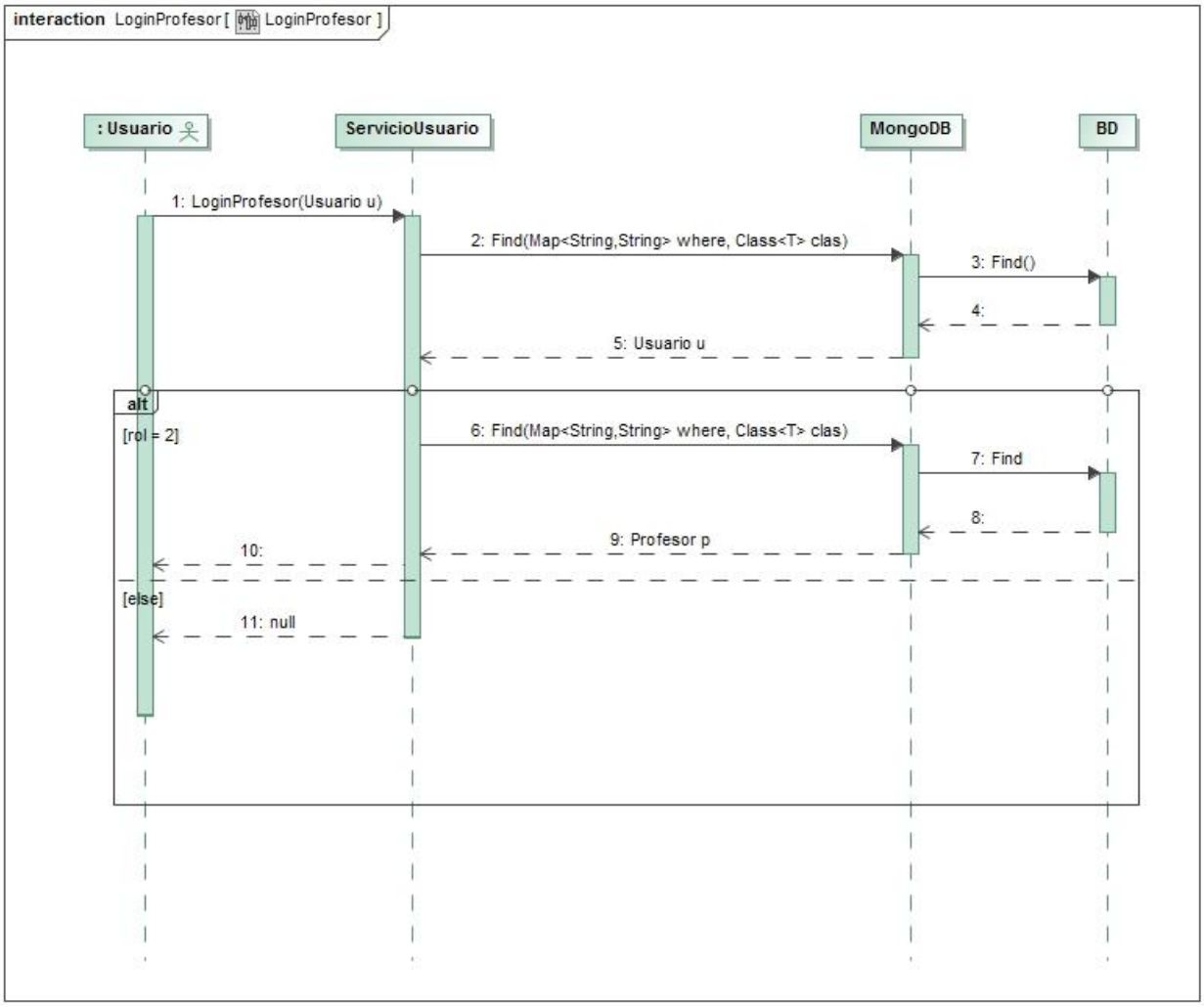

Ilustración 8 Diagrama de secuencia de Login Profesor

Título	Login Alumno
Descripción	Autenticación de un alumno dentro del sistema
Pre-condición	El usuario debe existir y estar relacionado con un alumno. No debe estar autenticado.
Post-condición	El usuario estará autenticado
Prioridad	Alta
Escenario principal	

1. El cliente manda una petición donde lleva el nombre de usuario y la contraseña del alumno.
2. El servicio lo recibe y encripta la contraseña para buscar en la base de datos si existe un usuario con ese nombre de usuario y contraseña.
3. La clase MongoDB comprueba si existe ese usuario.
4. Una vez comprobado la validez del usuario, se busca el alumno que esté relacionado con ese id.
5. La clase MongoDB busca ese alumno con ese id de Usuario.
6. El servicio devuelve al cliente el objeto del alumno.

Escenario alternativo

3.b no existe ningún usuario con ese nombre de usuario y contraseña.
5.b no existe ningún alumno con ese id de usuario

Clases de análisis

A. Clases de entidad	Usuario, Alumno
B. Clases de servicios	ServicioUsuario

Diagramas de secuencia

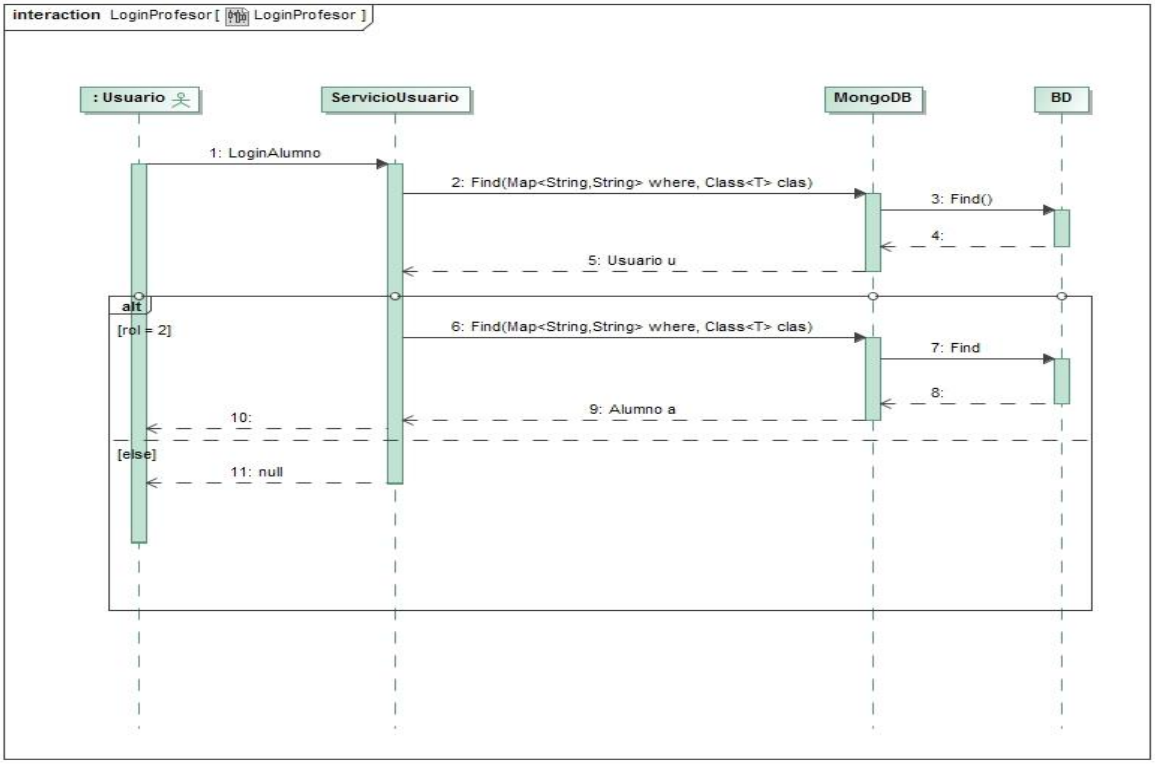

Ilustración 9 Diagrama de secuencia de Login Alumno

ServicioProblema

Título	Leer Problemas de un profesor
Descripción	Mostrar una lista con todos los problemas creados por un profesor
Pre-condición	El profesor tiene que estar autenticado en la aplicación
Post-condición	El cliente tendrá la lista de problemas de dicho profesor
Prioridad	Alta
Escenario principal	
1. El cliente manda una petición donde lleva el id del profesor. 2. El servicio lo recibe y llama al método find de la clase MongoDB donde se hará la búsqueda en la base de datos. 3. Se hace la búsqueda de todos los problemas que hayan sido creados por el profesor que tenga ese id y se devuelve al servicio. 4. Una vez el servicio tiene la lista de problemas lo devuelve al cliente.	
Escenario alternativo	
3.b no existe ningún problema hecho por ese profesor	
Clases de análisis	
A. Clases de entidad	Problema
B. Clases de control	ServicioProblema
Diagramas de secuencia	

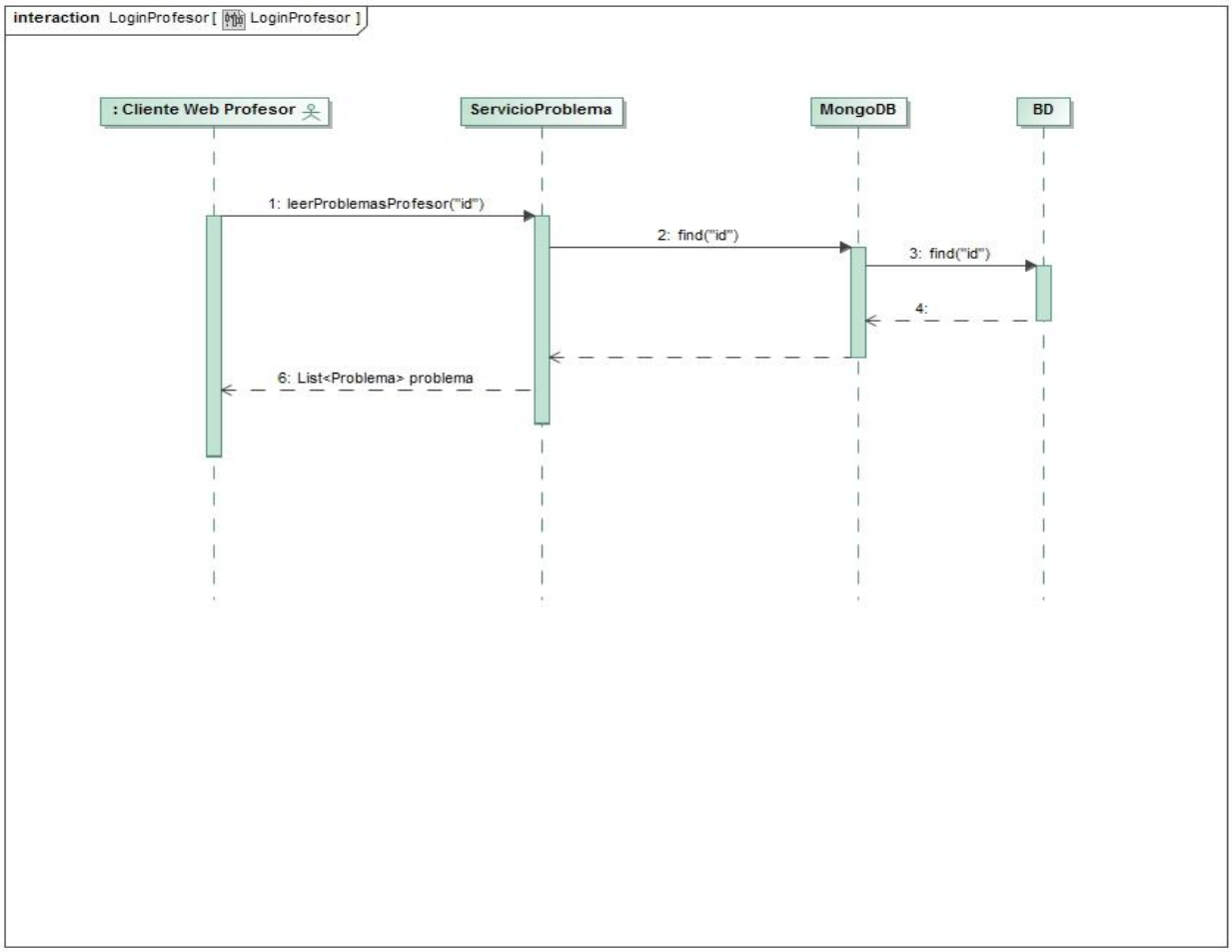

Ilustración 10 Diagrama de secuencia de leer problemas profesor

Título	Leer Problemas de una asignatura
Descripción	Mostrar una lista con todos los problemas que tiene una asignatura
Pre-condición	El alumno tiene que estar autenticado en la aplicación
Post-condición	El cliente tendrá la lista de problemas de dicha asignatura
Prioridad	Alta
Escenario principal	

1. El cliente manda una petición donde lleva el id de la asignatura.
2. El servicio lo recibe y llama al método find de la clase MongoDB donde se hará la búsqueda en la base de datos.
3. Se hace la búsqueda de todos los problemas que hechos para tal asignatura.
4. Una vez el servicio tiene la lista de problemas lo devuelve al cliente.

Escenario alternativo

3.b no existe ningún problema

Clases de análisis

A. Clases de entidad	Problema
B. Clases de control	ServicioProblema

Diagramas de secuencia

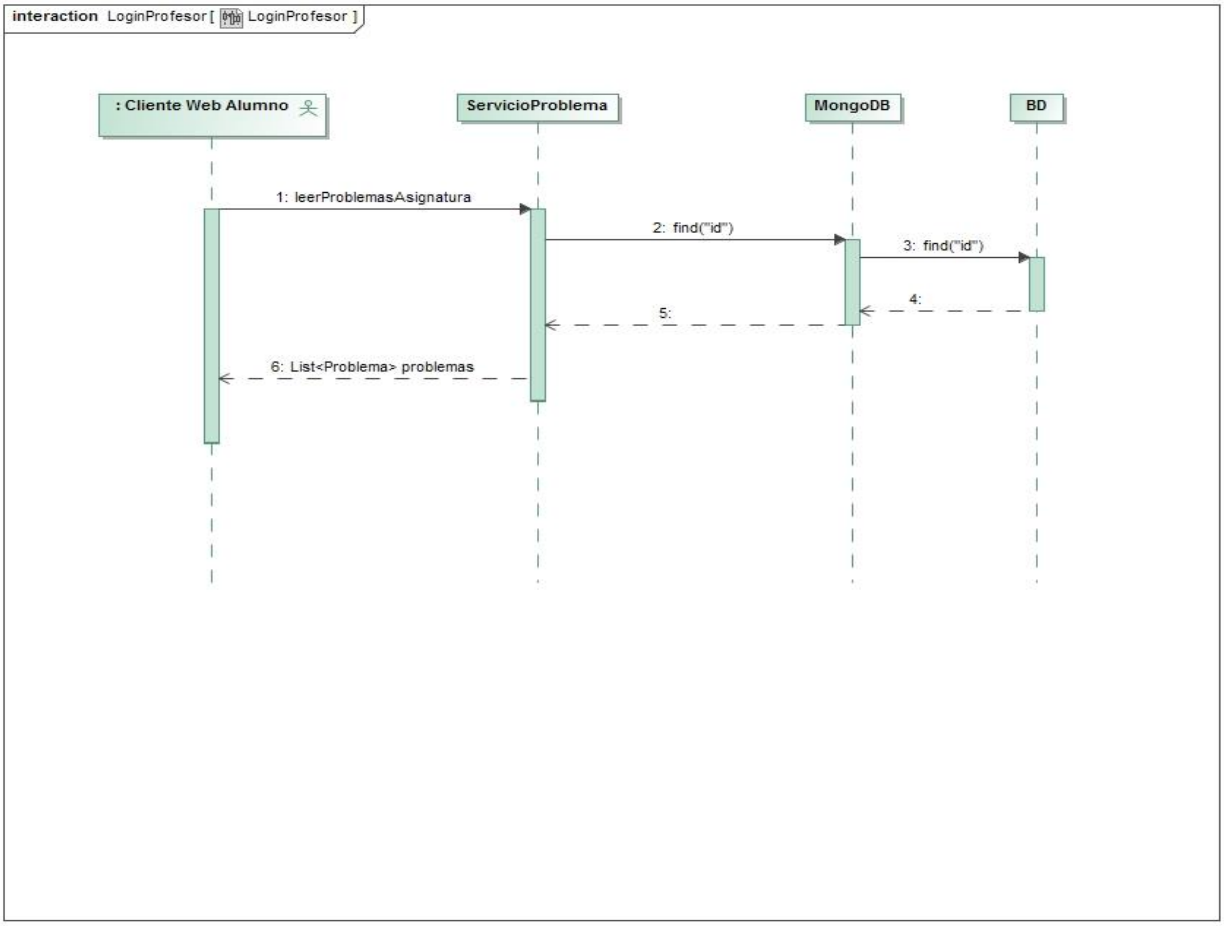

Ilustración 11 Diagrama de secuencia de leer Problemas Asignatura

Título	Leer problemas de un alumno
Descripción	Mostrar una lista con todos los problemas que tiene un alumno
Pre-condición	El alumno tiene que estar autenticado en la aplicación
Post-condición	El cliente tendrá la lista de problemas de un alumno
Prioridad	Alta
Escenario principal	
1. El cliente manda una petición donde lleva el id del alumno. 2. El servicio lo recibe y llama al método find de la clase MongoDB donde se hara la búsqueda de las asignaturas que tenga en la base de datos. 3. Se hace la búsqueda de todas las asignaturas que tenga ese alumno con ese id y se devuelve la lista al servicio. 4. Una vez que tenemos las asignaturas del alumno, por cada asignatura buscamos todos sus problemas a través de la clase MongoDB. 5. Una vez que tengamos todos los problemas de todas las asignaturas devolvemos la lista.	
Escenario alternativo	
3.b El alumno no está matriculado en ninguna asignatura. 4.b Las asignaturas que está matriculado el alumno no tienen ningún problema aún.	
Clases de análisis	
A. Clases de entidad	Problema
B. Clases de control	ServicioProblema
Diagramas de secuencia	

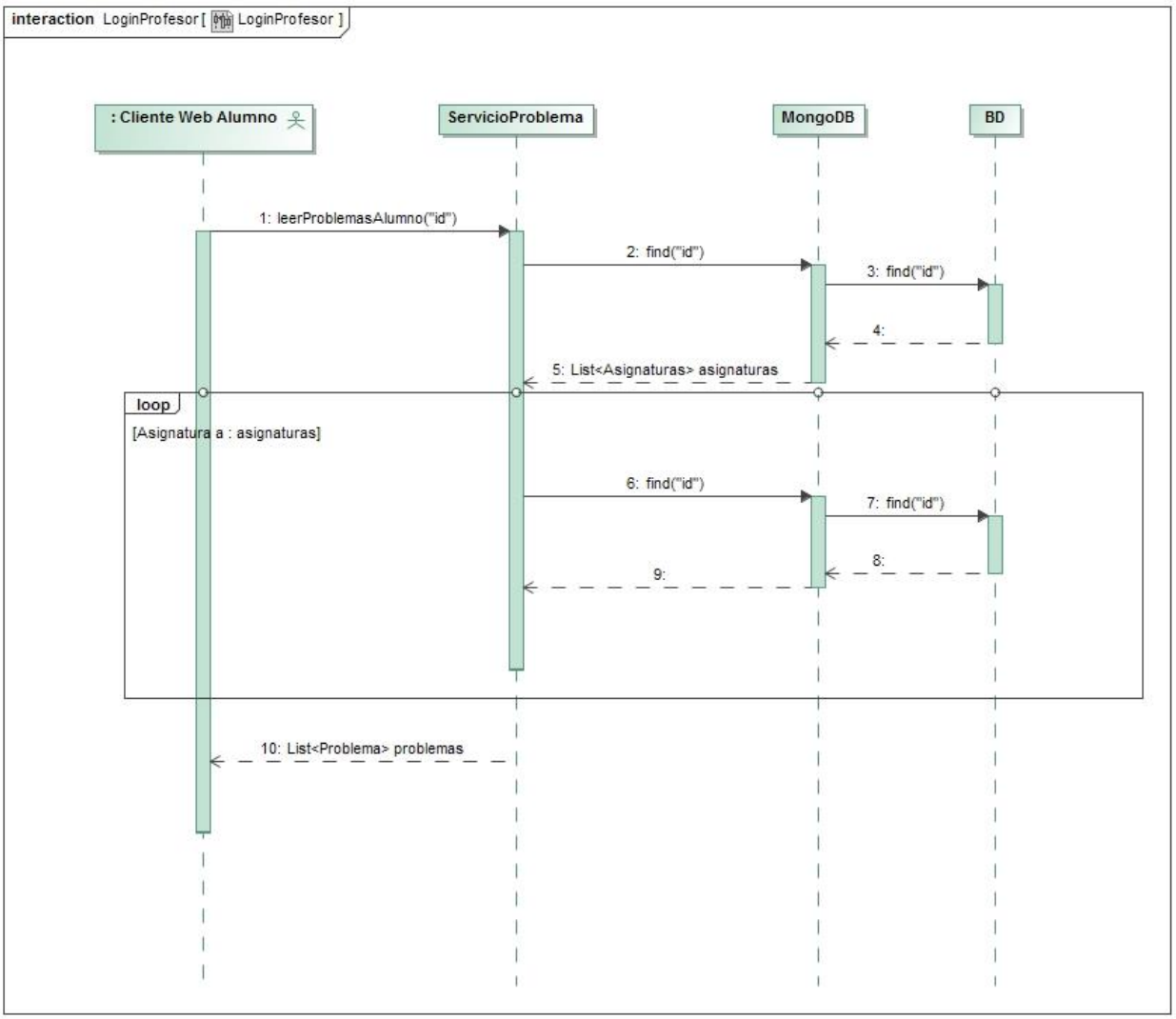

Ilustración 12 Diagrama de secuencia de leer Problemas Alumno

ServicioSolucion

Título	SolucionesUnProblema
Descripción	Mostrar una lista con todas las soluciones de un problema
Pre-condición	El profesor tiene que estar autenticado en la aplicación
Post-condición	El cliente tendrá la lista de soluciones
Prioridad	Alta

Escenario principal	
1. El cliente manda una petición donde lleva el id del problema. 2. El servicio lo recibe y llama al método find de la clase MongoDB donde se hará la búsqueda en la base de datos. 3. Se hace la búsqueda de todas las soluciones que tenga el problema concreto. 4. Una vez el servicio tiene la lista de soluciones lo devuelve al cliente.	
Escenario alternativo	
3.b no existe ningúna solución hecho por ese profesor	
Clases de análisis	
A. Clases de entidad	Solucion
B. Clases de control	ServicioSolucion
Diagramas de secuencia	

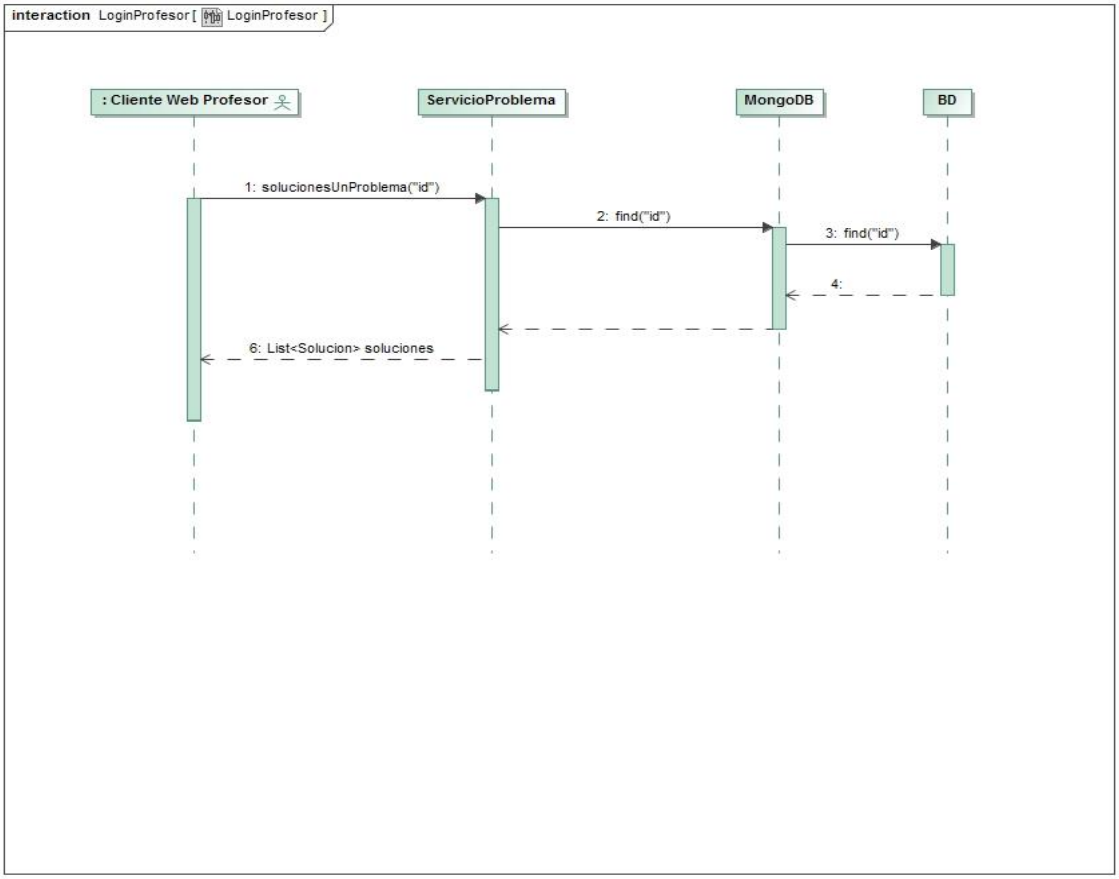

Ilustración 13 Diagrama de secuencia de soluciones de un problema

Título	cambiarNota
Descripción	Cambiar la nota de una solución
Pre-condición	El profesor tiene que estar autenticado en la aplicación
Post-condición	La nota estará cambiada en la base de datos
Prioridad	Alta
Escenario principal	
1. El cliente manda una petición donde lleva el id de la solución y la nota. 2. El servicio lo recibe y llama al método update de la clase MongoDB donde se cambiará la nota de la solución. 3. Se hace el update y se devuelve el número de filas afectadas. 4. Al cliente se le devuelve el este número.	
Escenario alternativo	
3.b no existe ninguna solución con ese id.	
Clases de análisis	
A. Clases de entidad	Solucion
B. Clases de control	ServicioSolucion
Diagramas de secuencia	

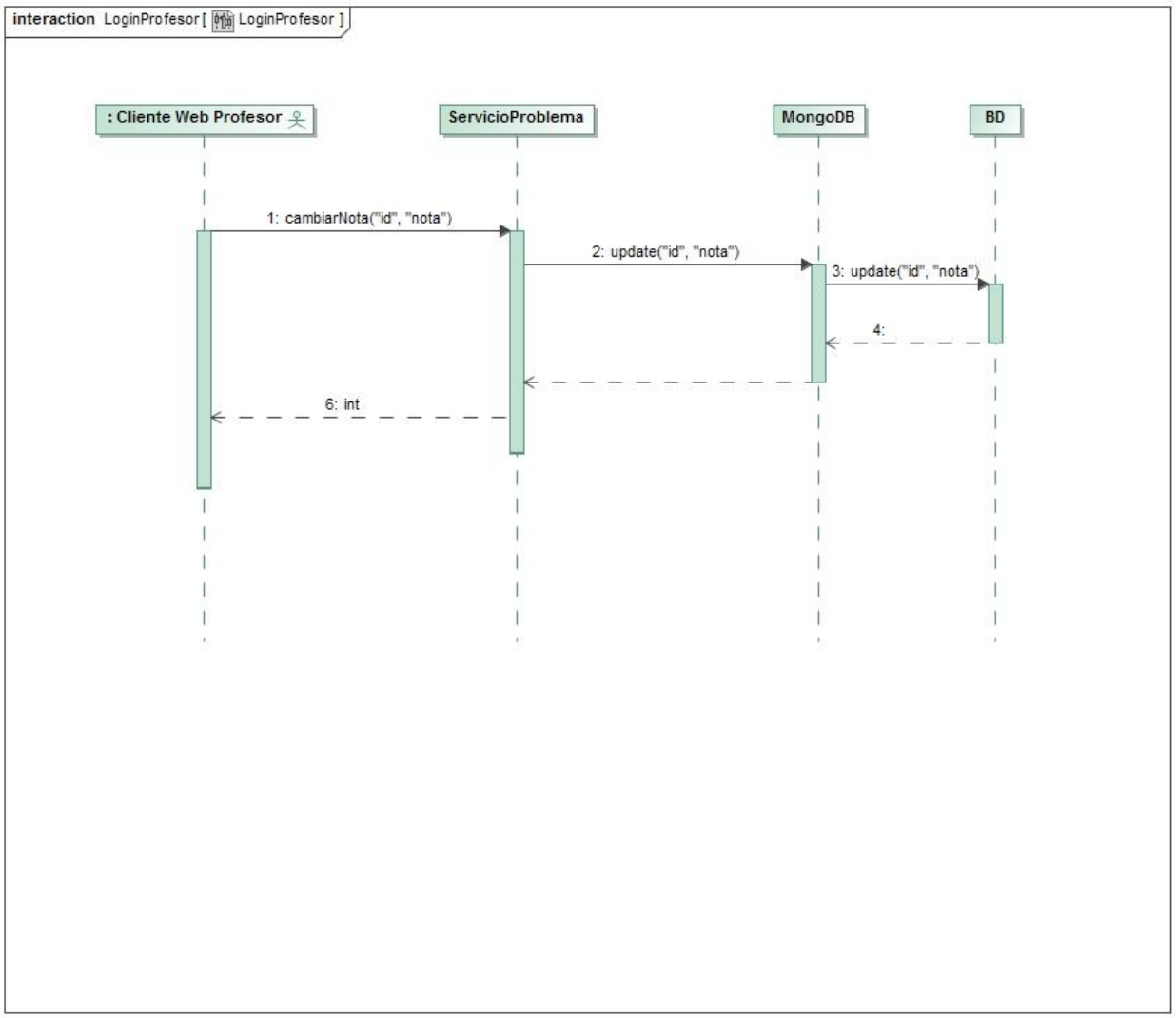

Ilustración 14 Diagrama de secuencia de cambiar nota

Estos serían los diagramas de secuencia de los servicios tanto para el cliente profesor como para el cliente alumno. Hay algunos servicios que están implementados pero están enfocado más a la administración, pero aún no se ha definido un cliente con ese rol.

Diagrama de flujo: Algoritmo evaluación

Para diseñar el algoritmo de evaluación vamos a utilizar un diagrama de actividad donde veremos la lógica de este y su funcionamiento.

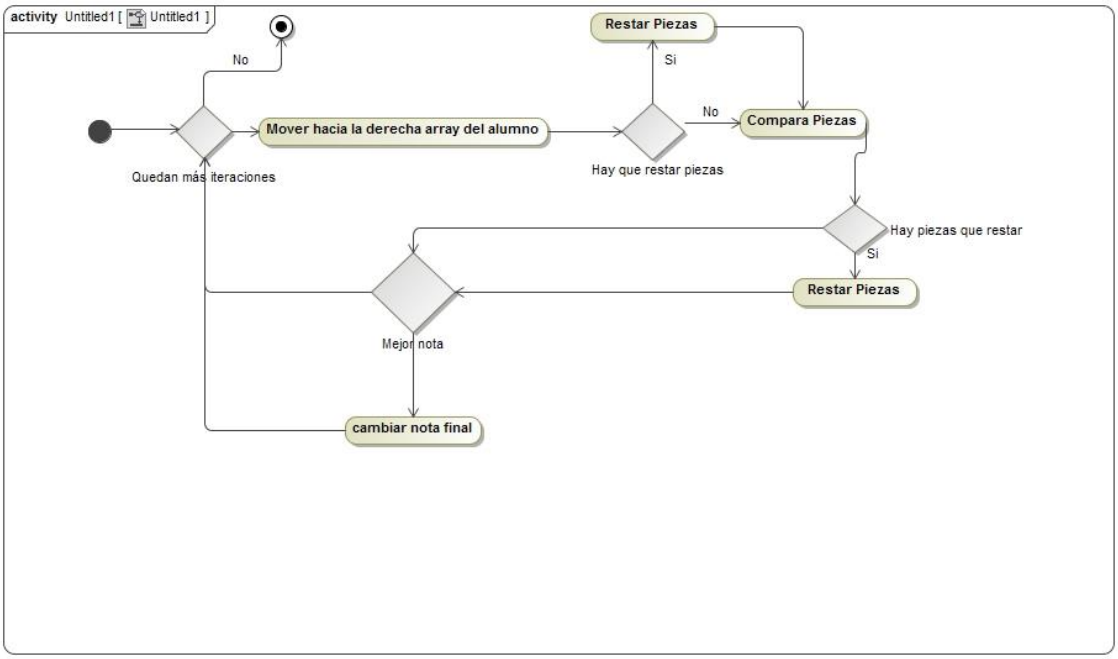

Ilustración 15 Diagrama de actividad del algoritmo de evaluación

Como podemos observar el algoritmo se va a basar en un bucle el cual parará cuando la última iteración sea aquella en la que el array de solución no se pueda mover más hacia la derecha. Este caso sería cuando la última pieza del array del profesor se va a comparar con la primera pieza de la solución del alumno.

Mientras, en cada iteración se hará lo siguiente:

1. Observamos si hay piezas del alumno que no vayan a ser comparadas y se restan los puntos que equivaldrían estas.
2. Comparamos tantas piezas de alumno como de profesor haya o hasta las que queden del alumno.
3. Si han sobrado piezas del alumno, se hace igual que en primer punto.
4. Se comprueba si la nota obtenida es mejor que la anterior. Si es así esta nueva pasa a ser la nota máxima del ejercicio.

Una vez hecho estos cuatro pasos, se vuelve al inicio del bucle y se moverá en un paso hacia la derecha el array de piezas del alumno.

Diagrama de despliegue

En este diagrama vamos a ver como los servicios estarían conectados a los clientes web.

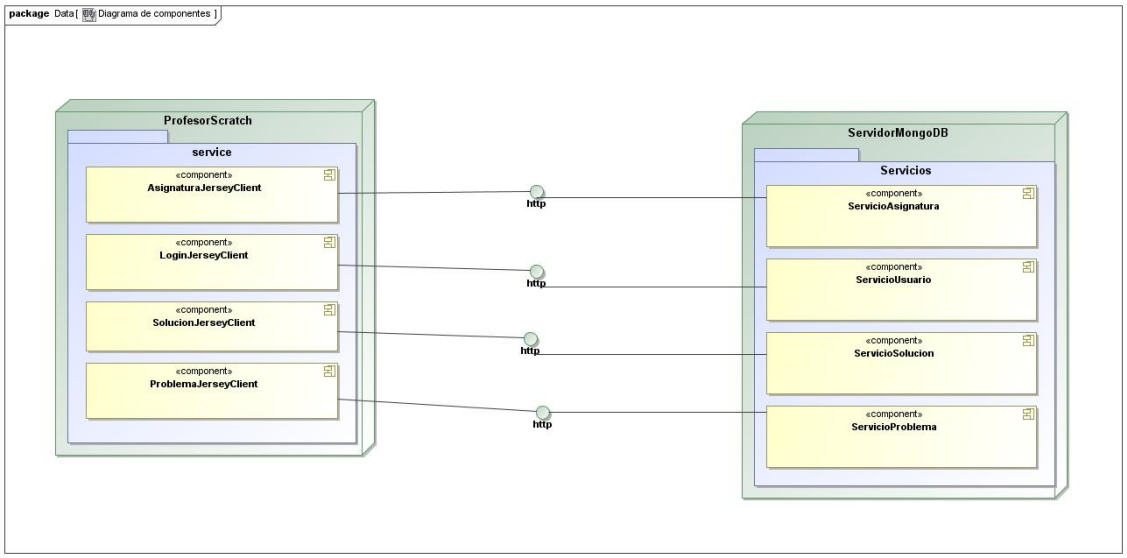

Ilustración 16 Diagrama de despliegue de la aplicación entre el cliente web profesor y el servidor mongo

Los clientes jersey son los encargados en el cliente web de hacer las peticiones a los servicios. Estas peticiones se hacen por el protocolo Http a través de cual se envían los datos.

Para el cliente web del Alumno sería igual que para el del profesor cambiando únicamente los servicios que necesitase el cliente.

Implementación y pruebas

En este apartado del documento veremos cómo hemos implementado la aplicación y cómo hemos utilizado las herramientas explicadas anteriormente para la creación de los servicios web y el uso de un programa SoapUi, el cual hemos utilizado para la realización de pruebas de los servicios una vez implementados.

Implementación

Para la implementación de los servicios, hemos creado tres paquetes: un paquete llamado Entity, donde se guardan las entidades con las que trabajaremos los datos; el paquete servicio, donde se desarrollaran los servicios web a los que el cliente web accederá después; y un paquete BD, donde se guardara la clase MongoDB que se encargará de hacer las operaciones CRUD del servidor de base de datos.

Paquete Entity

El paquete Entity va a albergar las clases que servirán después para la manipulación de los datos. Todas estas entidades tienen anotaciones @Entity debido al uso de la API Morphia. Estas anotaciones son las que después nos servirán para guardar las entidades de forma más sencilla en la base de datos Mongo, debido a que la API se encarga de convertir una instancia de un objeto normal a un objeto Document.

La funcionalidad de la anotación @Id sirve para la hora de devolver un objeto de la base de datos, almacenar el campo ObjectId en ese String.

Para atributos que no fuesen tipos simples, hemos utilizado la anotación @Embedded que se encarga de transformar ese atributo a una instancia de objeto Document.

La implementación de las entidades fue la siguiente:

SOLUCION:

```
@Entity("Soluciones")
@XmlRootElement
public class Solucion implements EntityMongo {

    @Id
    public String id;
    public String nota;
    public String idAlumno;
    public String idProblema;
    public String nombre;
    @Embedded
    public ArrayList<Pieza> piezas = new ArrayList<>();
```

Ilustración 17 Clase solución

ASIGNATURA

```
 */
@Entity("Asignaturas")
public class Asignatura {

    @Id
    public String id;
    public String nombre;
    public String idProfesor;
    public List<String> idAlumnos = new ArrayList<>();

    public Asignatura() {

    }

}
```

Ilustración 18 Clase Asignatura

INPUT

```
public class Input {

    public String type;
    public ArrayList<String> value;
    public int opcion;

    public Input() {

    }
```

Ilustración 19 Clase input

PIEZA

```
public class Pieza {

    public ArrayList<Input> inputs = new ArrayList<>();
```

Ilustración 20 Clase pieza

USUARIO

```java
public class Usuario{

    @Id
    public String id;
    public String usuario;
    public String password;
    public int rol;
```

Ilustración 21 Clase usuario

PROBLEMA

```java
public class Problema {

    @Id
    public String id;
    public String idProfesor;
    public String idAsignatura;
    public String nombreAsignatura;
    public String titulo;
    public String enunciado;
    @Embedded
    public ArrayList<Pieza> piezas = new ArrayList<>();
    @Embedded
    public ArrayList<Pieza> solucion = new ArrayList<>();
```

Ilustración 22 Clase problema

PROFESOR:

```java
public class Profesor {

    @Id
    public String id;
    public String nombre;
    public String apellido;
    public String idUsuario;
    public List<String> asignaturas = new ArrayList<>();

    public Profesor() {

    }
```

Ilustración 23 Clase profesor

Algoritmo de corrección:

En la clase solución se incluye el algoritmo de corrección. Este algoritmo está desarrollado de tal manera que lo que la solución del alumno sea tolerante y que un simple fallo no signifique que haya hecho mal el ejercicio entero.

Este algoritmo solo es llamado cuando el alumno ha insertado su solución o quiere tener modificarla.

El parámetro que recibirá este método será la solución ideal del profesor. Una vez recibido el array de piezas del profesor se calcula la nota máxima, que es simplemente contar todos los inputs que tiene la solución. Si una pieza tiene más inputs que otra está va a tener más peso que las demás.

Para contar que una pieza este bien esta pieza debe de tener todos los campos iguales.

Una vez dicho esto el algoritmo actuará de la siguiente manera como dijimos antes en el apartado de diseño:

Vamos a suponer que el array de la solución del profesor se queda fijo mientras que la solución del alumno se irá moviendo de izquierda a derecha. Al principio la solución del alumno será comparada con la solución del profesor suponiendo que la última pieza de la solución del alumno es igual a la primera pieza del array del profesor.

El array de piezas del alumno se irá moviendo hacia la derecha y comparándose de nuevo por cada iteración hasta acabar suponiendo que la primera pieza del alumno es igual a la última del profesor.

Para explicar estos dos últimos párrafos vamos a poner un ejemplo. Imaginad que la solución de un profesor consta de tres piezas y la del alumno de otras tres. El algoritmo actuaria de la siguiente forma:

1. Empezara restando las primeras dos piezas del alumno y comparando la última con la primera pieza del profesor.
2. En la siguiente iteración la primera pieza de la solución del alumno será restada, la segunda pieza comparada con la primera del profesor y la tercera con la segunda.
3. En la tercera iteración se van a comparar las piezas ordenadamente, la primera pieza del alumno con la primera del profesor, la segunda pieza del alumno con la segunda del profesor y la tercera pieza del alumno con la tercera del profesor. Si el alumno tiene el ejercicio bien esta sería la iteración con la cual nos quedaríamos y su nota sería la máxima, un diez.
4. En la cuarta iteración la primera pieza del alumno se comparará con la segunda del profesor, la segunda pieza con la tercera y la tercera pieza será restada a la nota.
5. En la quinta iteración la primera pieza del alumno será comparada con la última pieza del array de del profesor y las otras dos restarán.

Es obvio que en estos dos extremos la nota será muy baja y que probablemente la mejor nota del alumno se encuentre en una iteración media ya que todas las piezas que no son comparadas con las del profesor restan puntos.

Por cada iteración comprobamos si la nota obtenida es mejor que la iteración anterior y si es así se reemplaza y se guarda como máxima nota esta nueva.

Hay que recordar que nos quedaremos con la mejor iteración de todas debido a que como hemos dicho anteriormente vamos a tener cierta tolerancia con la solución del alumno y si no sabe que pieza va primera o última no puntuar todo el ejercicio a 0.

También está hecho para perjudicar aquellas soluciones que se pasan de piezas y que resten todas esas piezas de más que han puesto. Esto sirve para si en un problema no sabes que pieza poner, mejor no poner nada.

```
public void ponerNota(List<Pieza> arrayPiezas) {

    double notaMaxima = notaMaxima(arrayPiezas);
    double notaAlumno = 0.0;
    double notaAlumnoMejor = 0.0;
    double notaARestar = 0;
    int i;

    for (int k = -piezas.size() + 1; k < arrayPiezas.size(); k++) {
        i = 0;
        notaARestar = 0;

        for (; k + i < 0; i++) {
            for (Input inp : piezas.get(i).inputs) {
                notaARestar = notaARestar + 1 / notaMaxima;
            }
        }

        for (; ((i + k) < arrayPiezas.size()) && (i < piezas.size()); i++) {
            Pieza Alumno = piezas.get(i);
            if (arrayPiezas.get(i + k).equals(Alumno)) {
                for (Input inp : Alumno.inputs) {
                    notaAlumno = notaAlumno + 1 / notaMaxima;
                }
            }
        }
```

Ilustración 24 Algoritmo de corrección 1ª foto

```
            for (; i < piezas.size(); i++) {
                for (Input inp : piezas.get(i).inputs) {
                    notaARestar = notaARestar + 1 / notaMaxima;
                }
            }
            notaAlumno = notaAlumno - notaARestar;
            System.out.println("nota alumno :"+notaAlumno+" ; K :"+k+"/n");
            if (notaAlumno > notaAlumnoMejor) {
                notaAlumnoMejor = notaAlumno;
            }
            notaAlumno = 0.0;
        }

        nota = String.valueOf(notaAlumnoMejor * 10);

    }
```

Ilustración 25 Algoritmo de corrección 2ª foto

Paquete BD

En este paquete solo albergamos la clase MongoBD que como hemos dicho anteriormente va a ser la encargada de interactuar con la base de datos.

Como atributos vamos a tener:

- Un objeto Datastore que se encargará de hacer las operaciones con la base de datos.
- Un objeto mongoClient que dará la conexión al Datastore.
- Un objeto Morphia el cual necesitaremos para que nos devuelva una instancia de un objeto Datastore.

```java
public class MongoDB {

    public final static Morphia morphia = new Morphia();

    public static Datastore ds;

    public static MongoClient mongoClient;

    /**
     * Método que se utiliza para abrir conexión con la base de datos
     */
    public static void abrirConexion() {
        mongoClient = new MongoClient("localhost");

        ds = morphia.createDatastore(mongoClient, "Prueba");
    }

    /**
     * Método que se utiliza para cerrar la conexión con la base de datos
     */
    public static void cerrarConexion() {
        mongoClient.close();
    }
```

Ilustración 26 Atributos y métodos de abrir y cerrar conexión de la clase MongoDB

Esta clase consta de 5 métodos:

-**FindById:**

```java
public static <T> T findById(String id, Class<T> object) {

    try {
        //Abrimos conexion
        abrirConexion();

        T res;
        ObjectId oid = new ObjectId(id);
        res = ds.get(object, oid);

        cerrarConexion();

        return res;

    } catch (Exception e) {
        return null;
    }
}
```

Ilustración 27 Método findById de la clase MongoDB

Este método pasándole como parámetro un id y la clase del objeto obtiene el objeto especifico que queremos. El objeto Class<T> actúa como si le estuviésemos indicando al datastore la clase que queremos buscar el objeto.

-**Find:**

```
public static <T> List<T> find(Map<String, String> wheres, Class<T> clas) {

    abrirConexion();

    List<T> array;

    Query<T> query = ds.createQuery(clas);
    for (Entry<String, String> s : wheres.entrySet()) {
        query = query.filter(s.getKey(), s.getValue());
    }

    array = query.asList();

    MongoDB.cerrarConexion();

    return array;
}
```

Ilustración 28 Método find de la clase MongoDB

El método Find actúa como un select. El parámetro wheres serían las cláusulas de la sentencia y como hemos dicho anterior mente el objeto Class<T> actuaría como tabla.

Primero lo que hacemos en el método es crear la query y una vez hayamos insertado en la sentencia todas las restricciones de búsqueda, el método “asList” actuaría como si fuese un Execute y devuelve una lista de objetos según las restricciones dadas anteriormente.

-**Update:**

```
public static <T> void update(String id, Class<T> object,
        Map<String, String> change) {

    abrirConexion();
    //Accedemos a la tabla
    ObjectId oid = new ObjectId(id);
    UpdateOperations<T> ops = ds.createUpdateOperations(object);
    Query<T> elem = ds.createQuery(object).field("_id").equal(oid);

    for (Entry<String, String> s : change.entrySet()) {
        ops = ops.set(s.getKey(), s.getValue());
    }

    ds.update(elem, ops);

    //cerramos conexión
    cerrarConexion();
}
```

Ilustración 29 Método update de la clase MongoDB

Para hacer un método que simulase el Update necesitamos 3 parametros:

- El id el cual nos dice el objeto que queremos cambiar

- la clase la cual queremos acceder

- y los campos que vamos a cambiar con sus respectivos valores

-Insert

```
public static <T> void insert(T object) {

    abrirConexion();
    //Accedemos a la tabla
    ds.save(object);
    //cerramos conexión
    cerrarConexion();

}
```

Ilustración 30 Método insert de la clase MongoDB

El método insert como se puede observar es muy simple. Abrimos la conexión, guardamos la entidad y cerramos la conexión.

El método “save” nos proporciona un método de guardado en la base de datos. Reconoce la entidad y guarda el objeto en la colección que se le indique según el nombre que le indicamos en la etiqueta “Entity” como explicamos en el apartado anterior.

Este método sirve también como un update. Si al objeto se le introduce el id de un objeto asociado, este nuevo reemplaza al antiguo manteniendo el mismo id.

Delete

```
public static <T> int delete(String id, Class<T> entity) {

    try{
        int res;
        abrirConexion();
        //eliminamos el elemento de la tabla entity
        Query<T> elem = ds.createQuery(entity).field("_id").equal(id);
        WriteResult d = ds.delete(elem);

        res = d.getN();
        //cerramos conexión
        cerrarConexion();

        return res;
    }catch(Exception e){
        return -1;
    }
}
```

Ilustración 31 Método delete de la clase MongoDB

El método delete tiene como parámetros el id y la clase de la entidad que se quiere borrar. Se crea la sentencia y después es ejecutada.

Como resultado se devuelve el número de filas eliminadas.

Paquete Servicios

En este paquete se encontraran los servicios divididos en clases según a la colección que pertenezcan.

Para desarrollar los servicios web vamos a utilizar la tecnología JAX-RS, la cual nos va a permitir las anotaciones que nos indicaran la ruta del servicio, el verbo a utilizar, que tipo va a consumir y que tipo va a producir principalmente, pero antes de empezar debemos crear una clase que extienda de Application, la cual le vamos a indicar la dirección en la que se va a alojar todos los servicios.

```
@ApplicationPath("API")
public class ApplicationConfig extends Application{

}
```

Ilustración 32 Clase ApplicationConfig la cual nos indica en que url se van a hospedar los servicios

En nuestro caso sería la siguiente http://localhost:8080/ServidorMongo/API/...

A partir de esta dirección se almacenarán todos los servicios, los cuales en su clase correspondiente antes de definir la clase se le añadirá una etiqueta @Path que indicá la ruta del servicio

```
*/
@Path("Usuario")
public class ServicioUsuario {
```

Ilustración 33 Anotación Path que indica la ruta de un servicio

(En este ejemplo para acceder a los métodos del servicio Usuario sería introduciendo la dirección: http://localhost:8080/ServidorMongo/API/Usuario).

Todas las clases tendrán mínimo tres métodos: uno para guardar y hacer el update (según si el objeto tiene id o no), otro para buscar un objeto por su id y otro para eliminar un objeto por su id. Estos tres métodos llamaran a las funciones desarrolladas en la clase MongoDB explicada en el apartado anterior donde se llamará a los métodos insert, FindById y delete respectivamente.

Como ejemplo vamos a ilustrar la clase Usuario:

```
public Usuario insertarUsuario(Usuario u) {

    try {
        u.password = u.Encriptar();
        MongoDB.insert(u);
    } catch (Exception e) {
        return null;
    }
    return u;
}

@GET
@Path("buscarUsuario")
@Consumes({"application/xml", "application/json"})
@Produces("application/json")
public Usuario buscarUsuario(@QueryParam("id") String id) {

    Usuario u = MongoDB.findById(id, Usuario.class);

    return u == null ? new Usuario() : u;
}

@GET
@Path("eliminarUsuario")
public int eliminarUsuario(@QueryParam("id") String id) {

    return MongoDB.delete(id, Usuario.class);
}
```

Ilustración 34 Métodos de insertarUsuario, buscarUsuario y eliminarUsuario ServicioUsuario

Esta clase, aparte de los métodos de la página anterior que sirven como CRUD de Usuario, tiene principalmente los dos métodos de Login: uno para el alumno y otro para el profesor. Para acceder a los métodos de este servicio se deberá llamar a la siguiente dirección http://localhost:8080/ServidorMongo/API/Usuario

-LoginAlumno

```
@POST
@Path("LoginAlumno")
@Consumes({"application/xml", "application/json"})
@Produces("application/json")
public Alumno LoginAlumno(Usuario u) {
    Alumno a = null;
    try {
        Map<String, String> where = new TreeMap<>();
        where.put("password", u.Encriptar());
        where.put("nombreUsuario", u.nombreUsuario);

        u = MongoDB.find(where, Usuario.class).get(0);

        if (u.rol == 1) {
            where.clear();
            where.put("idUsuario", u.id);
            a = MongoDB.find(where, Alumno.class).get(0);
        }

        return a;
    } catch (Exception e) {

        return null;
    }
}
```

Ilustración 35 Servicio Login Alumno

Como se puede observar, para acceder a este método se deberá llamar con el verbo Post y a la siguiente dirección: localhost:8080/ServidorMongo/API/Usuario/LoginAlumno. El método consumirá un objeto json o xml el cual debe tener la estructura de un tipo Usuario.

Para empezar crearemos un Treemap el cual almacenará de “key” los atributos password y nombreUsuario y de values los valores de los atributos anteriores.

Debido a que tenemos las contraseñas guardadas en SHA1 encriptamos la contraseña que nos han pasado y la guardamos en el Treemap como value de la clave password.

Después buscamos en la base de datos si existe un usuario con ese nombre y contraseña. Sí existe comprobamos después si su rol se corresponde con el de un Alumno, en caso contrario se devuelve nulo.

Sí resulta que existe y es un alumno, buscamos en la colección de alumnos el alumno con ese id de Usuario y lo devolvemos.

-LoginProfesor

```
@POST
@Path("LoginProfesor")
@Consumes({"application/xml", "application/json"})
@Produces("application/json")
public Profesor LoginProfesor(Usuario u) {
    Profesor p = new Profesor();
    try {
        Map<String, String> where = new TreeMap<>();
        where.put("password", u.Encriptar());
        where.put("nombreUsuario", u.nombreUsuario);

        u = MongoDB.find(where, Usuario.class).get(0);

        if (u.rol == 2) {
            where.clear();
            where.put("idUsuario", u.id);
            p = MongoDB.find(where, Profesor.class).get(0);
        }

        return p;
    } catch (Exception e) {

        return null;
    }
}
```

Ilustración 36 Servicio Login Profesor

Para acceder al Login del profesor se deberá llamar con el Verbo Post y a la siguiente dirección: localhost:8080/ServidorMongo/API/Usuario/LoginProfesor. El método consumirá un objeto json o xml el cual debe tener la estructura de un tipo Usuario.

Como hicimos en el método anterior, empezaremos creando un Treemap el cual almacenará de “key” los atributos password y nombreUsuario y de values los valores de los atributos anteriores.

También encriptamos la contraseña que nos han pasado y la guardamos en el Treemap como value de la clave password.

Después buscamos en la base de datos si existe un usuario con ese nombre y contraseña. Si existe comprobamos después si su rol se corresponde con el de un Profesor, en caso contrario se devuelve nulo.

Sí resulta que existe y su rol corresponde al de un profesor, buscamos en la colección de Profesores el profesor con ese id de Usuario y lo devolvemos.

ServicioProfesor y ServicioAlumno

ServicioProfesor

```
@POST
@Path("insertarProfesor")
@Consumes({"application/xml", "application/json"})
@Produces("application/json")
public Profesor insertarProfesor(Profesor p) {

    try {
        MongoDB.insert(p);
    } catch (Exception e) {
        return null;
    }
    return p;
}

@GET
@Path("buscarProfesor")
@Produces("application/json")
public Profesor leerProfesor(@QueryParam("id") String id) {

    return MongoDB.findById(id, Profesor.class);
}

@GET
@Path("eliminarProfesor")
public int eliminarProfesor(@QueryParam("id") String id) {

    return MongoDB.delete(id, Profesor.class);
}
```

Ilustración 37 Clase servicioProfesor

ServiciosAlumno:

```
@POST
@Path("insertarAlumno")
@Consumes({"application/xml", "application/json"})
@Produces("application/json")
public Alumno insertarUsuario(Alumno a) {

    try {
        MongoDB.insert(a);
    } catch (Exception e) {
        return null;
    }
    return a;
}

@GET
@Path("buscarAlumno")
@Produces("application/json")
public Alumno buscarAlumno(@QueryParam("id") String id) {

    return MongoDB.findById(id, Alumno.class);

}

@GET
@Path("eliminarAlumno")
public int eliminarAlumno(@QueryParam("id") String id) {

    return MongoDB.delete(id, Alumno.class);
}
```

Ilustración 38 Clase ServicioAlumno

Como se puede observar, los únicos servicios creados para el profesor y para el alumno son los necesarios para llevar la gestión de ellos.

ServicioProblemas

Esta clase se va a utilizar para la gestión de Problemas. Tiene principalmente 3 importantes métodos, **LeerProblemasProfesor, LeerProblemasAsignaturas, LeerProblemasAlumno**. Para acceder a los métodos de este servicio se deberá llamar a la siguiente dirección http://localhost:8080/ServidorMongo/API/Usuario

```java
@GET
@Path("buscarProblemasProfesor")
@Produces("application/json")
public List<Problema> leerProblemasProfesor(@QueryParam("id") String id) {

    Map<String, String> where = new TreeMap<>();
    where.put("idProfesor", id);
    List<Problema> res = MongoDB.find(where, Problema.class);

    return res;
}

@GET
@Path("buscarProblemasAlumno")
@Produces("application/json")
public List<Problema> leerProblemasAlumno(@QueryParam("id") String id) {

    List<Problema> res = new ArrayList<>();
    Map<String, String> where = new TreeMap<>();
    where.put("idAlumnos", id);
    List<Asignatura> resAsignatura = MongoDB.find(where, Asignatura.class);

    for (Asignatura a : resAsignatura) {
        where.clear();
        where.put("idAsignatura", a.id);
        res.addAll(MongoDB.find(where, Problema.class));
    }

    return res;
```

Ilustración 39 Servicios leerProblemasProfesor y leerProblemasAlumno

LeerProblemasProfesor:

Para este método tenemos que acceder a la URL: localhost:8080/ServidorMongo/API/Problema/buscarProblemasProfesor utilizando el verbo GET.

De parámetro se le va a pasar el id del profesor.

El método se encargara de crear una búsqueda dentro de la colección de problemas trayendo aquellos cuyos campos “idProfesor” pertenezcan al que nos han enviado.

LeerProblemasAlumno:

Para este método tenemos que acceder a la URL: localhost:8080/ServidorMongo/API/Problema/buscarProblemasAlumno utilizando el verbo GET.

De parámetro se le va a pasar el id del alumno.

Como Alumnos y Problemas no están relacionadas directamente, necesitaremos pasar por la clase intermedia Asignatura. Esto quiere decir que para conseguir todos los problemas del alumno tendremos que sacar anterior mente todas las asignaturas que tenga el alumno.

Como vemos al principio del método buscamos todas las asignaturas que el alumno está cursando. Una vez que tenemos las asignaturas en un bucle, por cada asignatura que está cursando el alumno buscamos todos los problemas y lo añadimos a una lista de problemas.

leerProblemasAsignatura:

```
@GET
@Path("buscarProblemasAsignatura")
@Produces("application/json")
public List<Problema> leerProblemasAsignatura(@QueryParam("id") String idAsignatura) {

    Map<String, String> where = new TreeMap<>();
    where.put("idAsignatura", idAsignatura);
    List<Problema> res = MongoDB.find(where, Problema.class);

    return res;
```

Ilustración 40 Servicio leerProblemasAsignatura

Para este método tenemos que acceder a la URL: localhost:8080/ServidorMongo/API/Problema/buscarProblemasAsignatura utilizando el verbo GET.

De parámetro se le va a pasar el id de la Asignatura.

El método se encargara de crear una búsqueda dentro de la colección “Problemas” trayendo aquellos cuyos campos “idAsignatura” sean iguales al id pasado por parámetro.

ServicioSoluciones

Esta clase va a ser la encargada de llevar la gestión de las Soluciones del Alumno.

SolucionesUnProblema:

```java
@GET
@Path("SolucionesDeProblema")
@Produces("application/json")
public List<Solucion> SolucionesUnProblema(@QueryParam("id") String idProblema) {

    Map<String, String> where = new TreeMap<>();
    where.put("idProblema", idProblema);

    return MongoDB.find(where, Solucion.class);
}
```

Ilustración 41 Servicio solucionesUnProblema

Para este método tenemos que acceder a la URL: localhost:8080/ServidorMongo/API/Solucion/SolucionesDeProblema utilizando el verbo GET.

De parámetro se le va a pasar el id del problema.

El método se encargara de crear una búsqueda dentro de la colección “Soluciones” y buscará todas las soluciones del problema que le hemos indicado.

```java
@POST
@Path("insertarSolucion")
@Consumes({"application/json", "application/xml"})
@Produces("application/json")
public Solucion insertarSolucion(Solucion sol) {
    try {

        if (sol.id == null || MongoDB.findById(sol.id, Solucion.class) == null) {
            Problema p = MongoDB.findById(sol.idProblema, Problema.class);
            sol.ponerNota(p.solucion);
            MongoDB.insert(sol);
        } else {
            MongoDB.insert(sol);
        }

        return sol;

    } catch (Exception e) {
        return null;
    }
}
```

Ilustración 42 Servicio insertarSolucion

insertarSolucion:

Para este método tenemos que acceder a la URL: localhost:8080/ServidorMongo/API/Solucion/insertarSolucion utilizando el verbo POST.

De parámetro se le va a pasar un objeto “Solucion”. Para insertar una Solución primero observamos si el objeto que nos pasa tiene id o no. Si no hay id significa que la solución que nos llega es nueva y debemos de insertarla en la base de datos. En este caso también debemos buscar el problema que le corresponde a la solución y

comparar la solución ideal que tiene el problema con la que vamos a insertar mediante el algoritmo de corrección. Una vez corregida la solución se inserta en la base de datos.

En caso contrario significa que se va a hacer un simple update en la base de datos.

ServicioAsignatura

```
@GET
@Path("buscarAsignaturaProfesor")
@Produces("application/json")
public List<Asignatura> buscarAsignaturaProfesor(@QueryParam("id") String id) {

    Map<String, String> where = new TreeMap<>();
    where.put("idProfesor", id);
    List<Asignatura> res = MongoDB.find(where, Asignatura.class);

    return res;
}

@GET
@Path("buscarAsignaturaAlumno")
@Produces("applicaction/json")
public List<Asignatura> buscarAsignaturaAlumno(@QueryParam("id") String idAlumno){

    Map<String, String> where = new TreeMap<>();
    where.put("idAlumnos in", idAlumno);
    List<Asignatura> res = MongoDB.find(where, Asignatura.class);

    return res;
}
```

Ilustración 43 Servicios buscarAsignaturaProfesor y buscarAsignaturaAlumno

buscarAsignaturaProfesor:

Para este método tenemos que acceder a la URL: localhost:8080/ServidorMongo/API/Asignatura/buscarAsignaturaProfesor utilizando el verbo GET.

De parámetro se le va a pasar el id del profesor.

El método se encarga de buscar dentro de la colección de Asignaturas todas aquellas cuyo “idProfesor” sea igual al id pasado por parámetro.

buscarAsignaturaAlumno:

Para este método tenemos que acceder a la URL: localhost:8080/ServidorMongo/API/Asignatura/ buscarAsignaturaAlumno utilizando el verbo GET.

De parámetro se le va a pasar el id del alumno.

El método se encarga de buscar dentro de la colección de Asignaturas todas aquellas cuyo array “idAlumnos” contenga el id pasado por parámetro.

Pruebas

Para probar los servicios hemos utilizado el programa SoapUi. Este programa sirve para hacer peticiones a los servicios web en el cual en la llamada podemos elegir el verbo http, tipo de formato, dirección del servicio, etc.

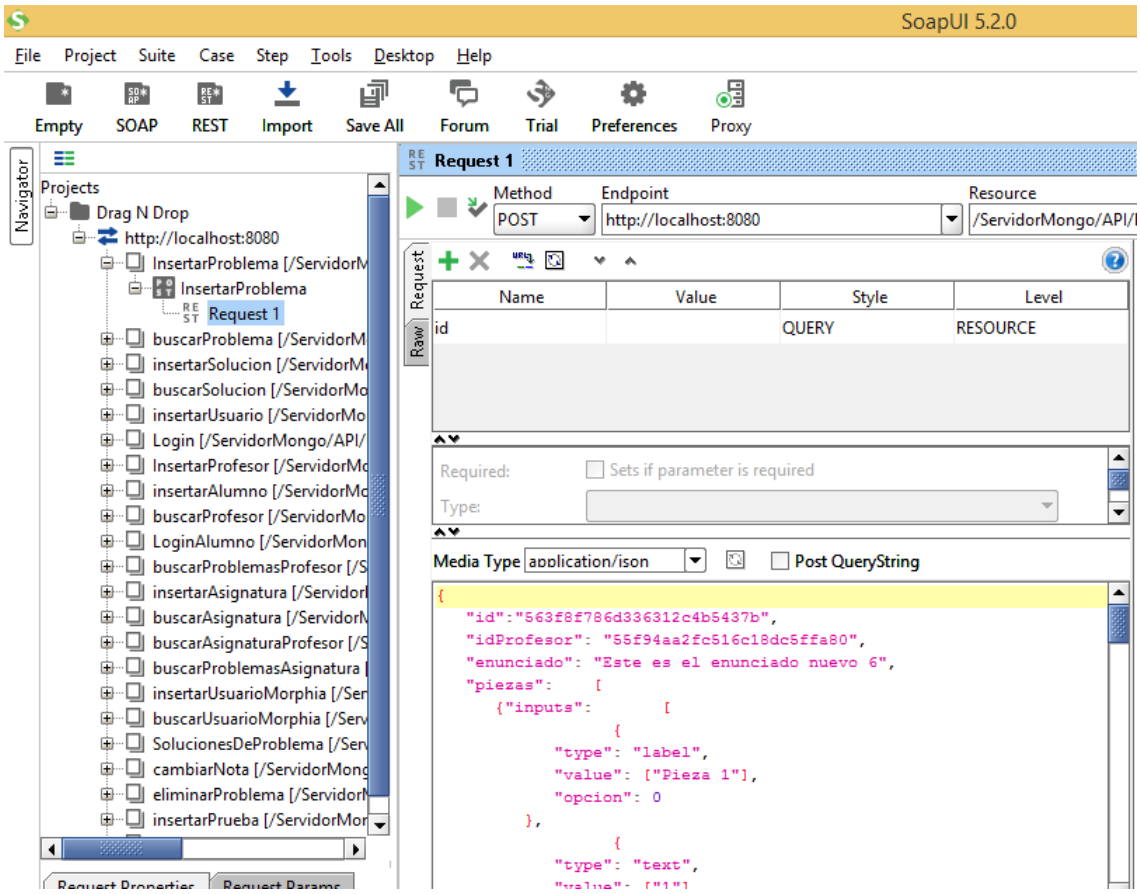

Ilustración 44 Imagen del programa SoapUi

Con este programa también podemos ver la respuesta que da el servidor y así asegurarnos si se había producido un error. El procedimiento es el siguiente:

1. Se crea un proyecto donde se especifica que van a ser llamadas a servicios RESTFul.
2. Después creamos un recurso por cada servicio.
3. A continuación le indicamos a la aplicación el método a utilizar, POST o GET según el que le especificamos en el apartado anterior.
4. Sí el servicio necesita un objeto json, se le indica el “Media Type” y después se escribe el objeto. En caso de que necesitase mandarle parámetros por la url, se hace un clic en el botón “+” y se le indica los parámetros.

5. Una vez que tenemos todo listo enviamos la petición y en la ventana de la derecha se mostrará el resultado. El resultado puede ser un objeto XML, JSON o HTML.

Conclusiones y Trabajos Futuros

Objetivos cumplidos

Los objetivos principales han sido cumplidos. Los servicios son capaces de darles a los clientes web todos los métodos imprescindibles para la ejecución de la aplicación y con detección de errores.

Un profesor ya puede crear un problema y revisar las soluciones de un alumno y un alumno ya puede ver los problemas que tiene y poder resolverlos sin ningún problema.

También hemos hecho posible hacer una conversión de una pieza “html” a un objeto json para que pueda ser almacenado en la base de datos.

Hemos creado un algoritmo de evaluación que sea capaz de evaluar una solución de un alumno.

Se han creado todos los servicios posibles para la gestión de los alumnos, profesores, usuarios, asignaturas, problemas y soluciones.

Varios métodos de autenticación de usuario que nos permiten acceder a la aplicación.

Dificultades encontradas

Realmente la única dificultad encontrada importante ha sido el uso de una nueva tecnología, MongoDB. Aunque al final la adaptación haya sido rápida al uso de esta tecnología, no ha sido fácil realizar el modelado, puesto que difiere considerablemente con respecto clásico E/R (Entidad/Relación).

Con MongoDB se rompe este esquema y en una Entidad pueden albergar arrays de otras entidades menos importante, como por ejemplo, ha sucedido con las entidades Solución y Problema las cuales albergaban un array de Piezas y que estas aparte incluían un array de Inputs.

Otra dificultad menor ha sido la serialización de un objeto input html a un objeto json. Aunque inicialmente parecía complejo, finalmente fue muy fácil de realizar poniendo únicamente los atributos del objeto input como atributos de una clase java.

Para acabar, la última complicación ha sido la elaboración del algoritmo de corrección automático. Se he intentado que no sea simplemente una comparación de dos arrays de piezas; se ha dado también un cierto margen al alumno para que, si no se acuerda de cómo empezar o se equivoca poniendo una pieza de más al principio o al final, no pierda todo el punto del ejercicio, aunque cabe recordar que al final la última palabra la tendrá el profesor.

Posibles Ampliaciones

Como posibles ampliación para este proyecto se podría añadir un nuevo usuario con el rol de administrador. Ya en varios apartados anteriores se ha mencionado que hay funcionalidades como la gestión de usuarios o la gestión de profesor que están implementadas, pero no hay un cliente web que las pueda modificar, de hecho la gestión se ha hecho mediante el programa SoapUi explicado en el apartado de Pruebas.

Otra posible ampliación podría hacer la aplicación más profesional y adaptarla al campus virtual para poder aplicarla en algunos exámenes o para simples ejercicios.

Podríamos también hacer una aplicación móvil para reutilizar los servicios en Android o modificar los clientes web para que los eventos de JQuery también detecten los eventos táctiles del móvil.

Bibliografia

https://docs.mongodb.org/manual/?_ga=1.263137099.610464664.1448211269

http://www.soapui.org/

http://mongodb.github.io/morphia/