ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

DRAG & DROP

Realizado por
Edgar Pérez Ferrando
Tutorizado por
Eduardo Guzmán de los Riscos
Departamento
Lenguajes y Ciencias de la Computación

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.

Keywords

- Bootstrap.
- Client and service (Restful).
- CSS.
- HTML.
- Input.
- JavaScript.
- jQuery.
- JSON.
- JSP and Servlet.
- Piece.
- Problem.
- MongoDB.
- UML.
- SCRUM.
- Solution.

Contenido

1. Introducción

1.1. Descripción

Este trabajo fin de grado se centra en el desarrollo de un entorno web educativo para la resolución y construcción de problemas. El objetivo de este entorno es, por tanto, facilitar el diseño y evaluación de actividades de tipo procedimental. Se basará en un entorno muy popular, Scratch, desarrollado por el Instituto Tecnológico de Massachusetts (MIT) para enseñar fundamentos de programación a estudiantes no universitarios (alumnado de Primaria y Bachillerato). Scratch permite la construcción de programas a través de un entorno visual a través del cual los alumnos, mediante operaciones de arrastrar y soltar, a partir de una colección de bloques que interconectarán entre sí, elaboran programas.

El entorno desarrollado en este proyecto permitirá a los profesores diseñar actividades de tipo procedimental. Para ello tendrán que definir el conjunto de bloques que cada alumno deberá utilizar para la resolución de un determinado problema. A partir de este conjunto se construirá un enunciado y una solución ideal al problema.

Posteriormente, a través de un entorno específico, el alumnado podrá resolver el problema o problemas propuestos por el profesor. Para ello se le mostrará, por cada problema, el enunciado y la colección de bloques disponibles para construir la solución.

La corrección de las soluciones se realizará a posterior a partir de todas las soluciones que los alumnos elaboren. A partir de ellas, se construirá un grafo que contemple todas las posibles soluciones, así como la solución ideal del profesor. A partir de este grafo y, en base a un conjunto de heurísticos, se calculará la calificación de cada alumno en el problema.

En resumen, dentro del entorno se podrán identificar los siguientes subsistemas:

- **Herramienta del profesor:** A través de ella el profesor podrá especificar problemas (colecciones de bloques, enunciados y solución ideal).
- **Herramienta del alumnado:** Esta herramienta permitirá a cada alumno resolver el problema o problemas que cada profesor proponga. Se tratará de un entorno visual donde la solución se representará a partir de bloques instanciados y conectados entre sí.
- **Módulo de evaluación y construcción de grafo de solución:** Este módulo será el encargado de construir el grafo de solución del problema, a partir de la solución ideal y de todas las formas en las que los diferentes alumnos han resuelto el problema. A partir de este grafo global, y en función de los pasos que cada alumno haya realizado de forma correcta, se calculará su calificación en cada problema.

1.2. División del trabajo

El trabajo quedará dividido en 3 partes entre los componentes del TFG en grupo:

- **Francisco Jesús Domínguez Ruiz**: Módulo de evaluación y construcción de grafo de solución.
- **Edgar Pérez Ferrando:** Herramienta del profesor.
- **Javier Ordóñez Martín:** Herramienta del alumnado.

1.3. Métodos y fases del trabajo

Para el desarrollo del TFG se empleará una metodología ágil, incremental e iterativa. Más concretamente se va a utilizar Scrum.
Roles:

- Product Owner será el tutor del TFG
- Scrum Master: Javier Ordóñez Martín
- Equipo de Desarrollo:
    - Javier Ordóñez Martín.
    - Francisco Jesús Domínguez Ruíz.
    - Edgar Pérez Ferrando.

1.4. Tecnologías y herramientas

Para la realización del TFG expuesto anteriormente se van a utilizar las siguientes herramientas:

- NetBeans: Entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

- Trello: Herramienta de gestión de proyecto aplicada para proyectos con metodologías ágiles y SCRUM.

- MongoDB: Sistema de base de datos NoSQL orientado a documentos. 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, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

- Servlets y JSP: 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. JSP es similar a PHP, pero usa el lenguaje de programación Java.

- JAX-RX: 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).

- HTML: Lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia para la elaboración de páginas web en sus diferentes versiones.

- CSS: Hoja de estilo en cascada o CSS es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML.

- Javascript: Es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

- jQuery: biblioteca de JavaScript, 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: Variante de la librería jQuery centrada en funcionalidades para los interfaces de usuario. En nuestro proyecto será un pilar fundamental, ya que ésta es la librería que se utiliza para el movimiento de piezas mediante los eventos de ratón.

- Bootstrap: Framework de diseño para páginas y aplicaciones web. Se centra en realizar el contenido en modo adaptativo para visualizar todo tipo de contenidos en todo tipo de dispositivos. Contiene estilos para muchos componentes html como formularios, botones, menús de navegación, etc.

- MagicDraw: Software para la elaboración de diagramas CASE, utilizado para realizar los diagramas de UML en las fases de análisis.

1.5. Organización del documento

1 - Introducción

En el primer apartado del documento, encontramos una descripción del trabajo que se propone, así como la división de las tareas del mismo entre los componentes del grupo de trabajo.

Se mencionan también las herramientas y los métodos del trabajo que se utilizarán a lo largo del desarrollo del mismo.

2 - Especificación y análisis

Aquí es donde se realiza el análisis del proyecto planteado (Recolección de los requisitos), una visión inicial del sistema y de lo que se pretende realizar.

Estudiaremos las necesidades, los participantes y la situación en la que se desenvolverá el producto y la competencia que pueda tener el producto final.

3 – Diseño de la aplicación

Una vez planteado el análisis procedemos a realizar un diseño básico del producto. Se mostrará un diagrama sobre el modelo de negocio, diagramas de caso de uso y un diagrama de secuencia por cada caso de uso mencionado.

Se intentará cubrir todos los requisitos funcionales mediante los diagramas de secuencia. Se describirán los pasos que hay que seguir para cumplir el objetivo en el sistema.

Para entender mejor el producto se han realizado dos diagramas añadidos, uno de navegación que describirá la forma en la que podremos movernos por la aplicación, y un diagrama de actividad para entender el procedimiento que utilizaremos para realizar la inserción de un nuevo problema.

Por último se hablará de la estructura de los ficheros que usaremos para la intercomunicación entre el cliente de nuestra aplicación y el servidor de almacenamiento MongoDB.

4 – Implementación

Detallaremos un poco la estructura del código realizado, mostraremos las interfaces gráficas finalizadas y el contenido de los diferentes paquetes que conformas nuestra aplicación.

También se comentará algún fragmento de código que se considere importante.

5 – Pruebas

Hablaremos de las pruebas que se han realizado en el desarrollo y cómo ha respondido la aplicación ante ellas.

6 – Conclusiones y trabajos futuros

Un último repaso al producto, comentando las conclusiones y reflexionando sobre el resultado obtenido. Así mismo también se plantean posibles

extensiones del producto y se mencionan funcionalidades interesantes que para un futuro podrían implementarse.

7 – Referencias

Un pequeño apartado donde podemos encontrar enlaces de interés desde los cuales hemos podido consultar y obtener información necesaria para realizar este documento, así como desarrollar el producto.

2. Especificación y análisis

2.1. Descripción de participantes y usuarios

Nombre	Rol
Profesor	El profesor es el que realiza los problemas para que los solucione el alumno.
Alumno	El alumno plantea una solución a un problema dado y se le evalúa.

2.1.1. Perfiles de usuario

Nombre	Descripción	Responsabilidades	Criterio de Éxito
Profesor	Gestiona problemas, y corrige soluciones.	Creación, lectura, actualización y eliminación de problemas, evaluar soluciones.	Poder evaluar los conocimientos del alumno mediante problemas planteados por él.
Alumno	El alumno plantea soluciones a los problemas.	Realizar soluciones a los problemas para ser evaluados.	Poder evaluar sus conocimientos de una forma constructiva y sencilla.

2.1.2. Alternativas y competencia

Actualmente no existe una alternativa similar, ya que es un proyecto realizado en base a la construcción de código basada en scratch.

2.2. Visión general del producto

2.2.1. Entorno de despliegue

2.2.1.1. Entorno para la implementación del sistema

La aplicación está pensada para desplegarse inicialmente en los mismos servidores de alojamiento de los centros educativos en los que se desee utilizar.

Es un requisito recomendado, por lo tanto se puede utilizar cualquier hosting que permita desplegar aplicaciones web Java con soporte para servicios Restful.

2.2.1.2. Aplicaciones Colaboradoras

En principio nuestra aplicación no cuenta con ninguna aplicación colaboradora directa, pero se podría implementar posteriormente algún ajuste para trabajar con las plataformas educativas como Moodle.

2.3. Requisitos funcionales de la aplicación

RF1 - Iniciar sesión:
Un usuario inicia sesión en el sistema (Profesor).

RF2 - Cerrar sesión:
Un profesor finaliza la sesión en el sistema.

RF3 - Crear un problema:
Un profesor, plantea un ejercicio, mediante el enunciado y una posible solución que se utilizará para corregir la respuesta dada por un alumno.

RF4 - Listar problemas:
Un profesor puede listar todos los problemas propuestos por él mismo.

RF5 - Editar problemas:
Un profesor puede modificar un problema realizado por él.

RF6 - Borrar problemas:
Un profesor puede eliminar un problema propuesto por él.

RF7 - Ver problema:
Un profesor puede ver un problema realizado por él.

RF8 - Listar soluciones de los alumnos:
Un profesor puede ver las soluciones propuestas de los alumnos a un problema propuesto por él.

RF9 - Visualizar una solución:
Un profesor puede visualizar la solución propuesta por un alumno.

RF10 - Corregir soluciones:
Un profesor puede comprobar la solución individual propuesta por un alumno de un problema planteado por él y corregir la calificación si no está conforme con la que se ha generado del algoritmo de corrección automática.

2.4. Requisitos no funcionales

RNF1 - Almacenamiento secundario implementado con MongoDB.

RNF2 - Comunicación entre aplicaciones implementada con servicios Resftul.

RNF3 - Interfaz gráfico realizado con Bootstrap.

RNF4 - Uso de librerías jQuery para arrastrar elementos HTML.

RNF5 - Interfaz responsive para adaptación a todo tipo de dispositivos.

3. Diseño de la aplicación

A continuación se detallará mediante diagramas UML el diseño inicial de la aplicación. UML nos ayudará a entender la estructura del modelo que utilizaremos, así como describir las funcionalidades mediante los casos de uso.

También nos sirve para entender quiénes son los participantes y cómo interactúan con la aplicación.

3.1. Modelo de negocio

A continuación vamos a detallar el modelo inicial que compone nuestra aplicación mediante un diagrama de clases UML. Es posible que la estructura no se entienda en un principio, se explica a continuación.

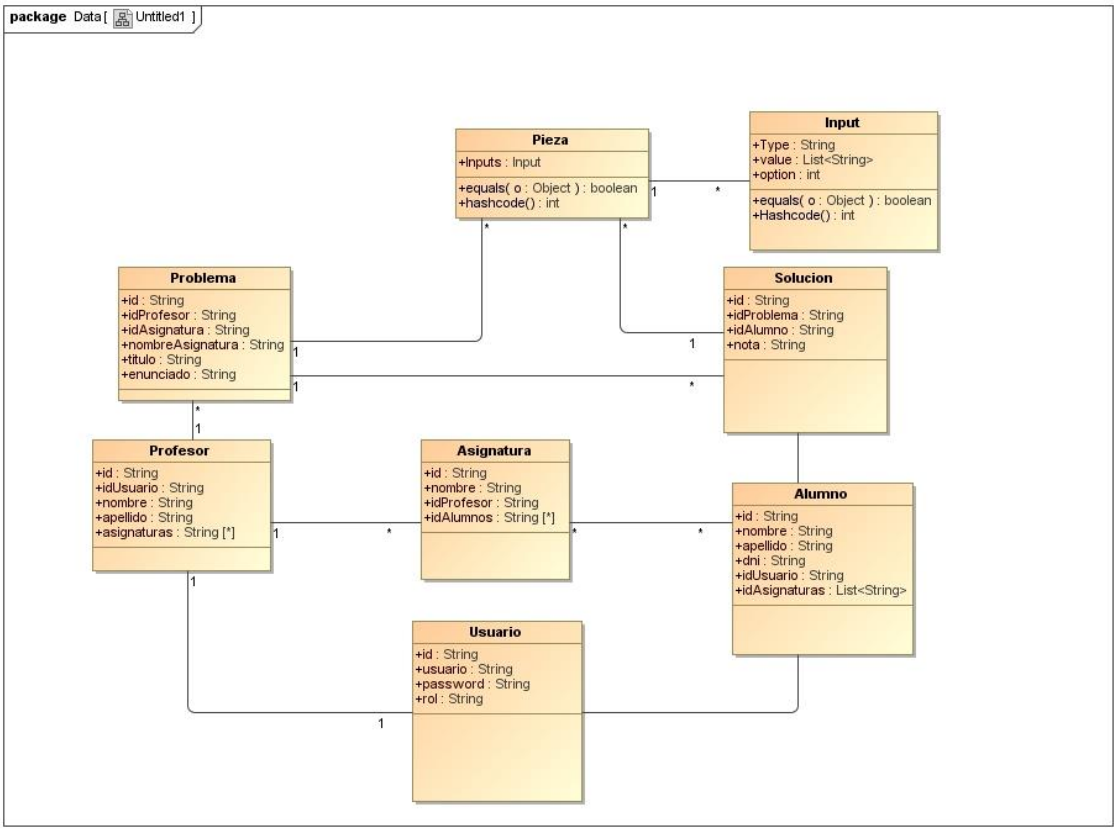


Ilustración 1 - Diagrama de clases UML

- Usuario: Representa a un usuario en el sistema para acceder al mismo.
- Profesor: Representa a un usuario de tipo Profesor, ya que el "rol" del profesor será ligeramente diferente al alumno.
- Alumno: Representa a un usuario de tipo Alumno, ya que el "rol" de alumno tiene unas capacidades limitadas y diferentes a las de un profesor.
- Asignatura: La asignatura se utiliza para identificar la materia a la que está asociada un profesor.
- Problema: Representa a un problema como tal, un enunciado, un título, una asignatura, etc.

- Solución: Es la representación física de la solución de un problema.
- Pieza: Para construir un problema y una solución se utilizan las piezas.
- Input: Una pieza utiliza elementos input, ya sea para poner un texto, número, valor asociado, etc…

3.2. Diagramas de casos de uso

Un caso de uso describe brevemente la lista de pasos que se siguen en un proceso para llevarse a cabo. Por lo tanto, el diagrama de caso de uso nos ayuda a entender las funcionalidades del sistema desde un punto de vista muy abstracto, así como quién es el causante del mismo.

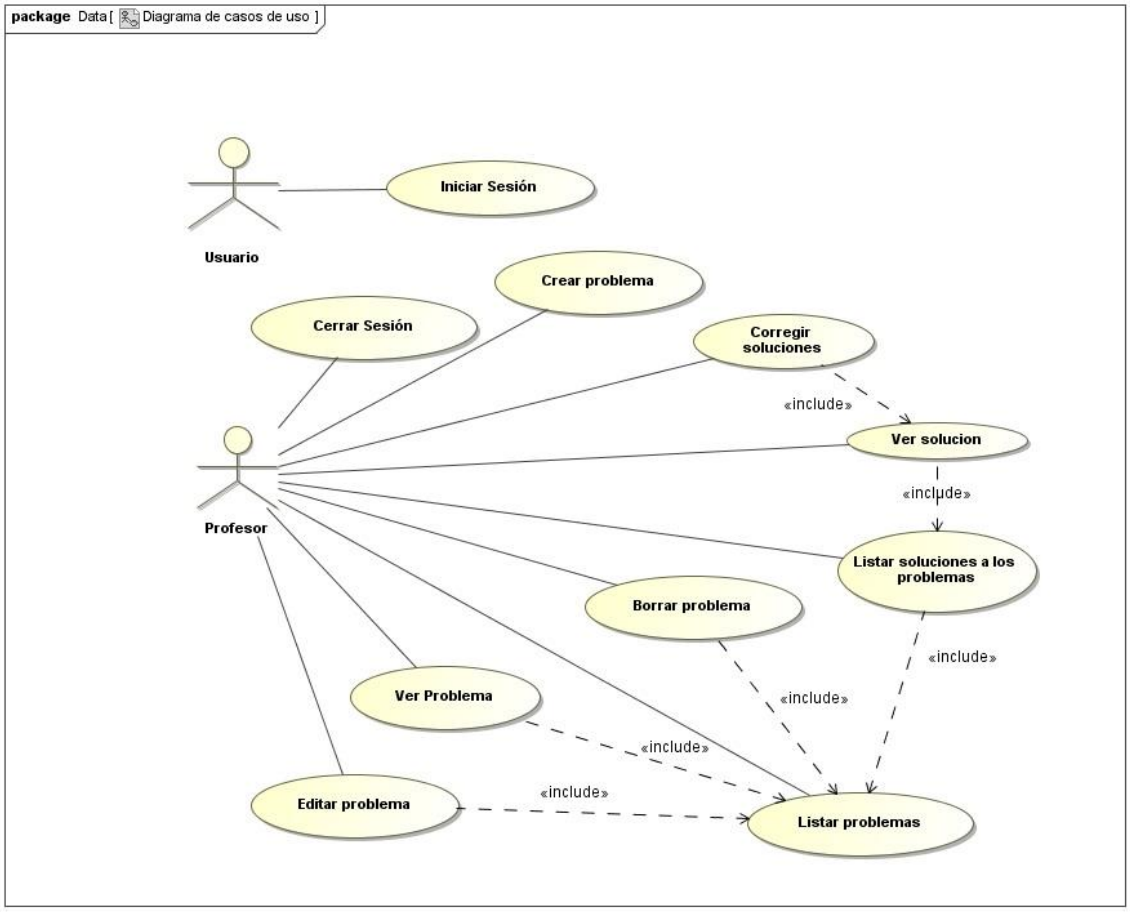


Ilustración 2 - Diagrama de casos de uso UML

3.2.1. RF1 - Iniciar sesión

Título	Iniciar sesión
Descripción	Un usuario inicia sesión en el sistema, a través de unos credenciales.
Pre-condición	No hay ninguna sesión iniciada.

Post-condición	Un usuario, se identifica en el sistema.
Prioridad	Media
Autor(es)	Usuario
Escenario principal	
1. El usuario introduce el nombre de usuario y contraseña en el formulario. 2. El usuario hace clic en el botón de iniciar sesión. 3. El sistema recibe los datos y comprueba que es un usuario válido. 4. El sistema inicia la sesión.	
Escenario alternativo	
3.b El sistema comprueba que es un usuario incorrecto. 4.b El sistema notifica que el usuario no existe.	
Clases de análisis	
A. Clases de entidad	Usuario, Profesor, HttpSession
B. Clases de control	LoginServlet
C. Clases de interfaz	Index.jsp
Maquetas de interfaz	

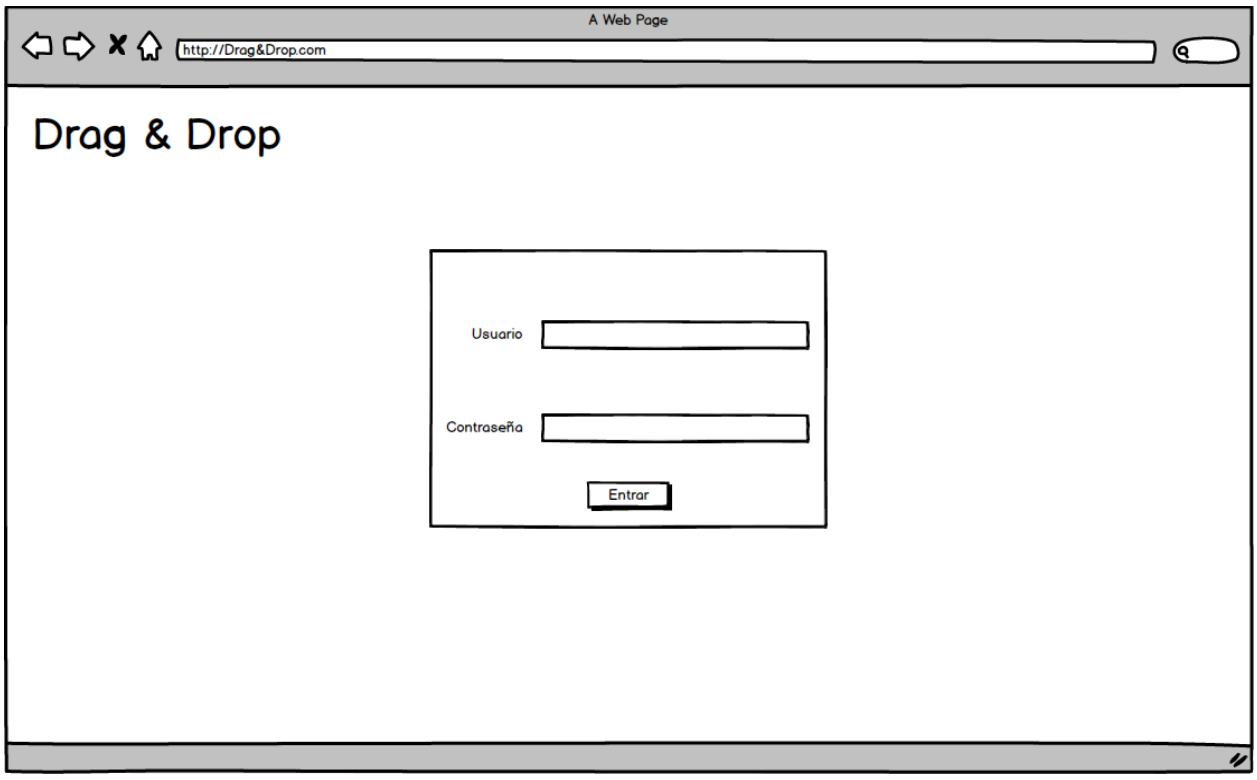


Ilustración 3 - Mockup index.jsp

Diagramas de secuencia

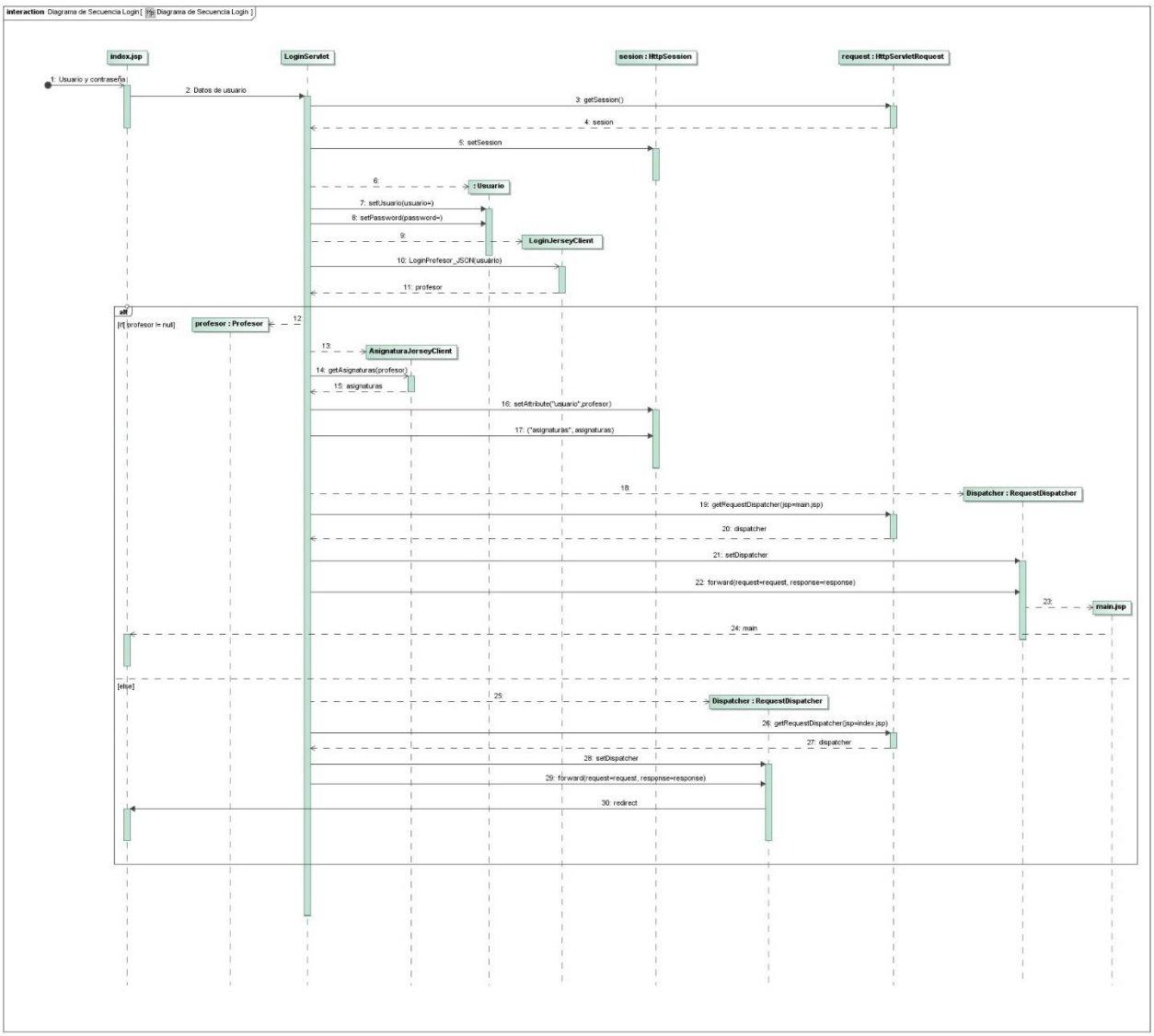


Ilustración 4 - Iniciar sesión (Diagrama de secuencia UML)

3.2.2. RF2 - Cerrar sesión

Título	Cerrar sesión
Descripción	Un profesor finaliza su sesión
Pre-condición	Debe de haber una sesión iniciada por un profesor
Post-condición	No hay ningún profesor en la sesión
Prioridad	baja
Autor(es)	Profesor

Escenario principal

1. El profesor hace clic en el enlace de cerrar sesión.
2. El sistema vacía la sesión y redirige la aplicación a la vista de login.

Clases de análisis

A. Clases de entidad	Profesor
B. Clases de control	LogoutServlet
C. Clases de interfaz	navegacion.jsp, index.jsp

Maquetas de interfaz

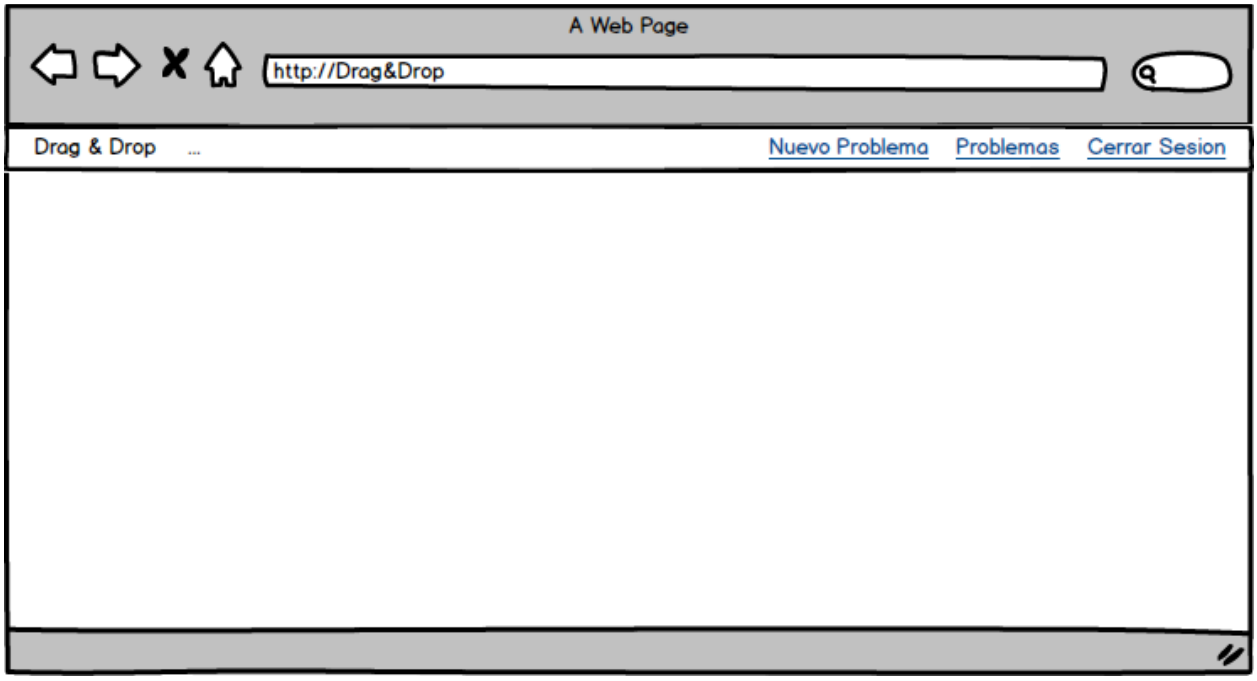


Ilustración 5 - Mockup de la barra de navegación

Diagramas de secuencia

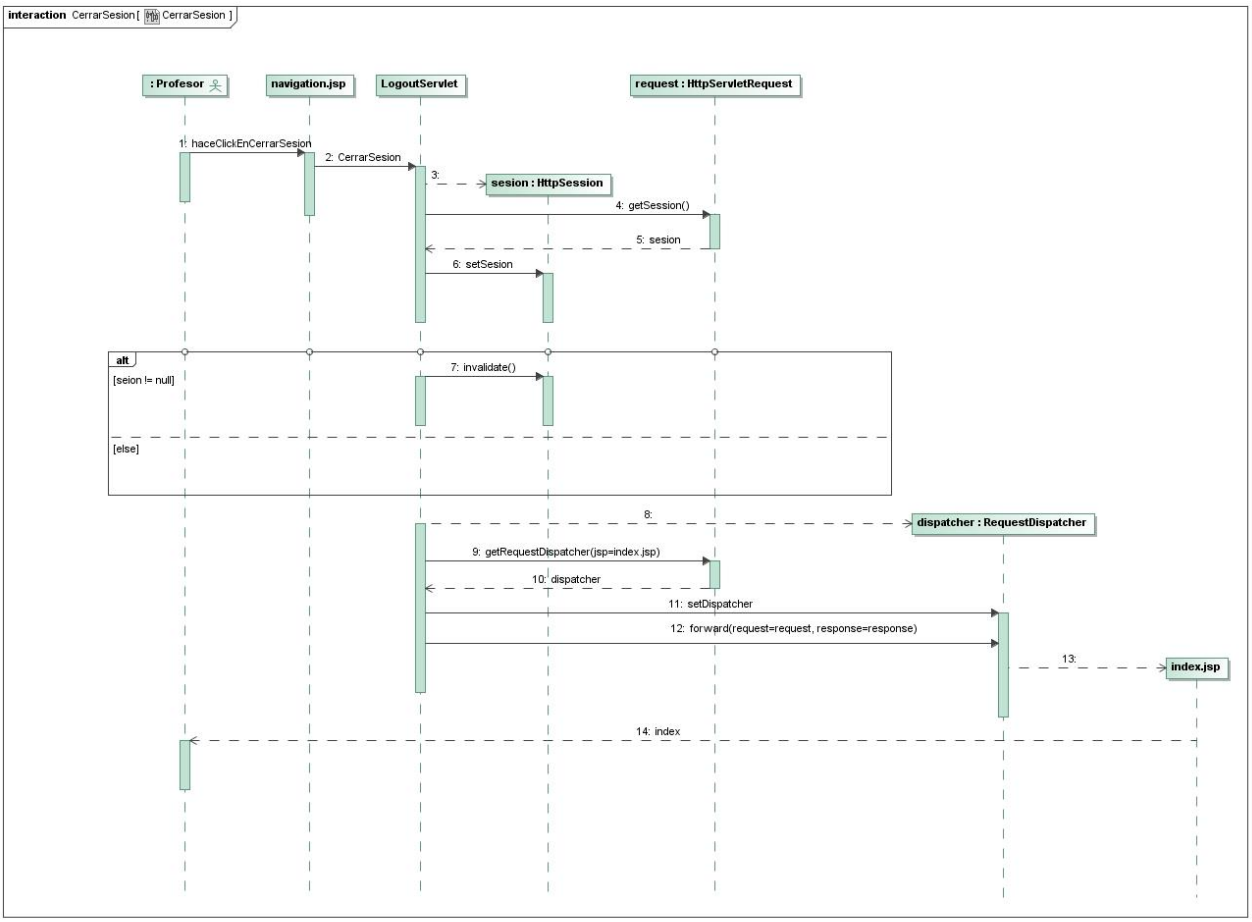


Ilustración 6 - Cerrar sesión (Diagrama de secuencia UML)

3.2.3. RF3 - Crear Problema

Título	Crear Problema
Descripción	Un profesor creará un problema junto con la solución para almacenarlo en la persistencia y que los alumnos puedan realizar sus soluciones.
Pre-condición	El profesor desea realizar un problema a sus alumnos
Post-condición	El problema está almacenado en la persistencia junto con la solución para que los alumnos planteen sus soluciones a dicho problema y se genere una calificación automática.
Prioridad	Alta
Autor(es)	Profesor

Escenario principal	
1. El profesor hace clic en la barra de navegación “Nuevo problema”. 2. El profesor selecciona el archivo de texto para que se generen las piezas. 3. El profesor introduce el título del problema. 4. El profesor introduce el enunciado. 5. El profesor selecciona la asignatura a la que pertenece el problema. 6. El profesor genera la solución ideal. 7. El profesor finaliza la creación haciendo clic en el botón Finalizar. 8. El sistema envía el problema para su almacenamiento. 9. El sistema notifica al profesor que se ha realizado todo sin problemas.	
Escenario alternativo	
3.b El profesor finaliza la creación del problema 4.b El sistema notifica que no se puede almacenar un problema vacío.	
Clases de análisis	
A. Clases de entidad	Profesor
B. Clases de control	
C. Clases de interfaz	Navigation.jsp, main.jsp
Maquetas de interfaz	

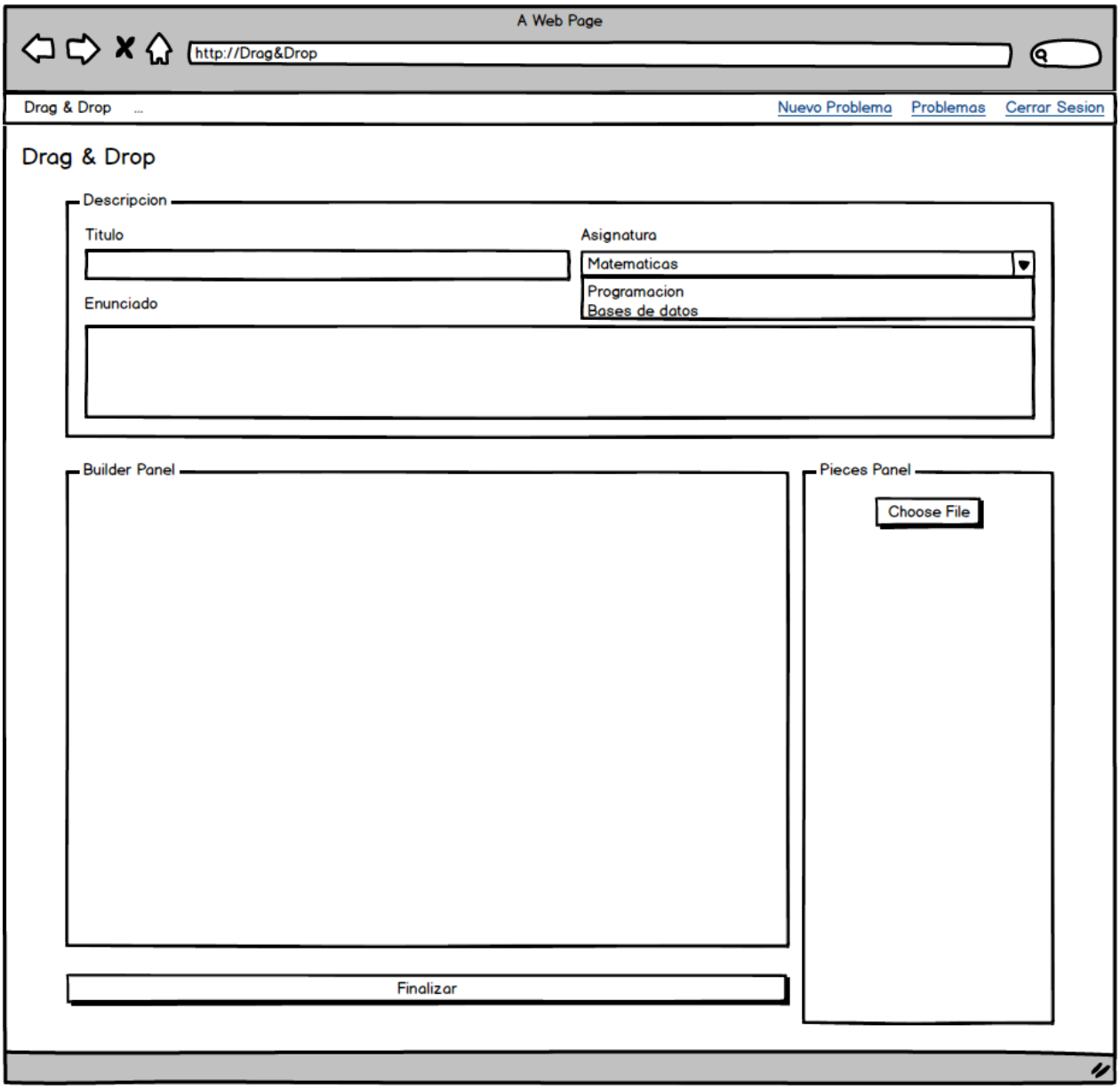


Ilustración 7 - Mockup main.jsp

Diagramas de secuencia

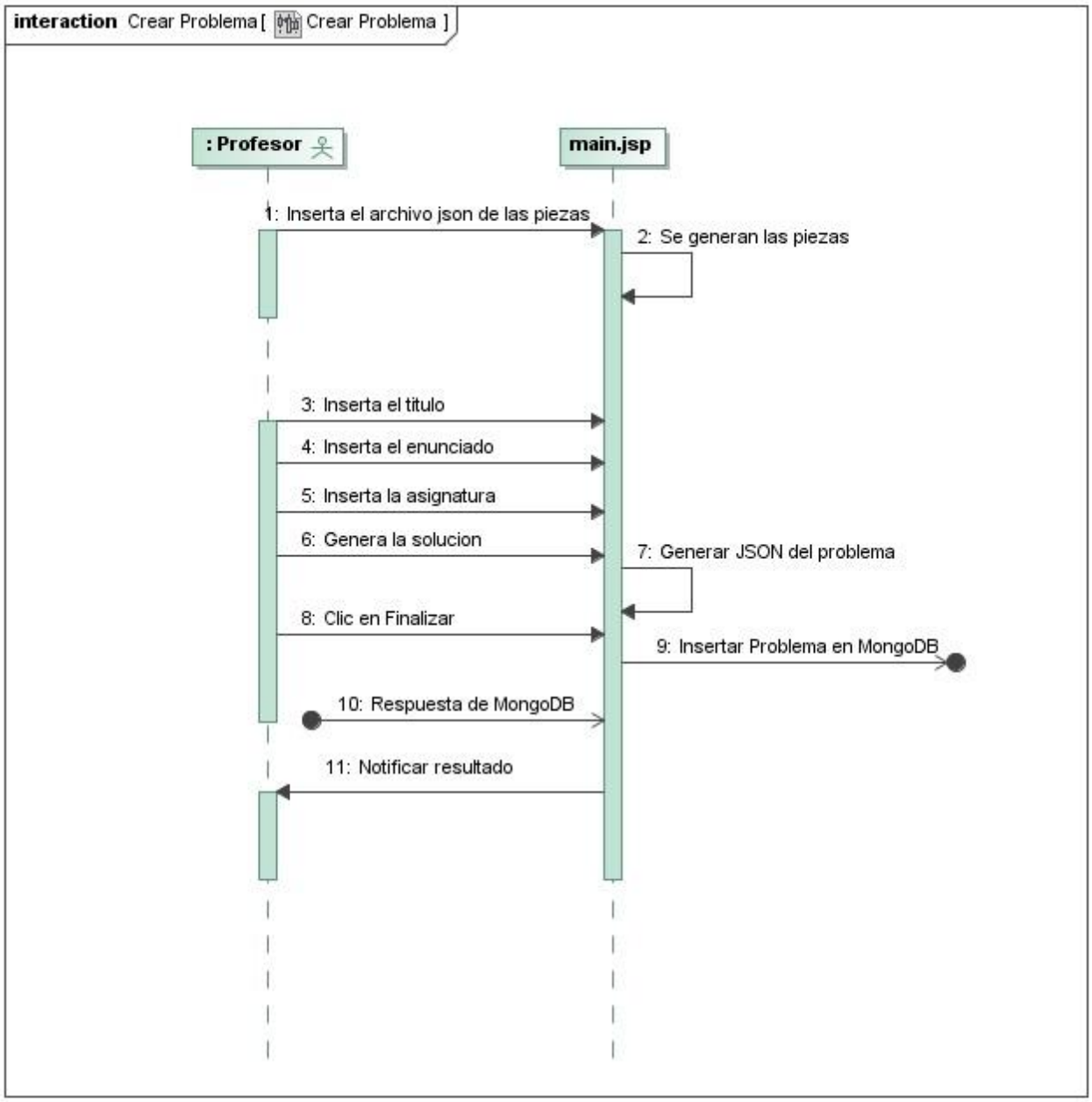


Ilustración 8 - Crear problema (Diagrama de secuencia UML)

3.2.4. RF4 – Listar Problemas

Título	Listar Problemas
Descripción	El profesor va a listar los problemas que ha planteado.
Pre-condición	El profesor desea listar todos los problemas que ha realizado.

Post-condición	El profesor consigue un listado de los problemas.
Prioridad	media
Autor(es)	Profesor

Escenario principal

1. El profesor hace clic en el enlace de “Problemas” en la barra de navegación.
2. Se muestra una nueva ventana en la que aparecen los problemas planteados por el profesor en una tabla.

Escenario alternativo

2.b. No se encuentra ningún problema en la persistencia y se notifica mediante un mensaje.

Clases de análisis

A. Clases de entidad	Profesor, Problema, Asignatura
B. Clases de control	ListarProblemaServlet, ProblemaJerseyClient
C. Clases de interfaz	listadoProblemas.jsp, navigation.jsp

Maquetas de interfaz

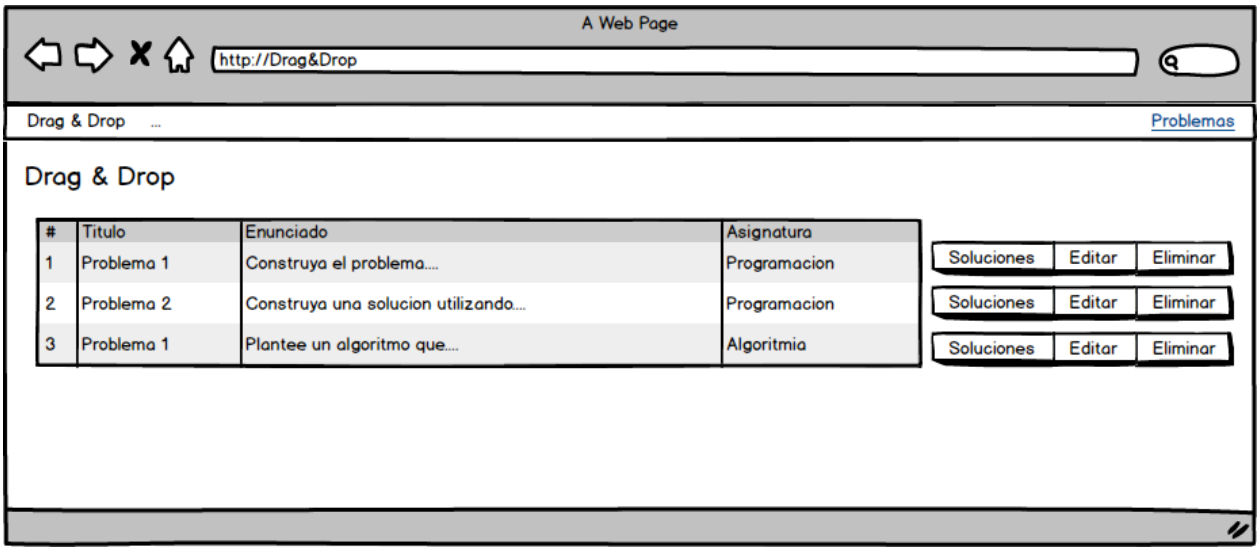


Ilustración 9 - Mockup listadoProblemas.jsp

Diagramas de secuencia

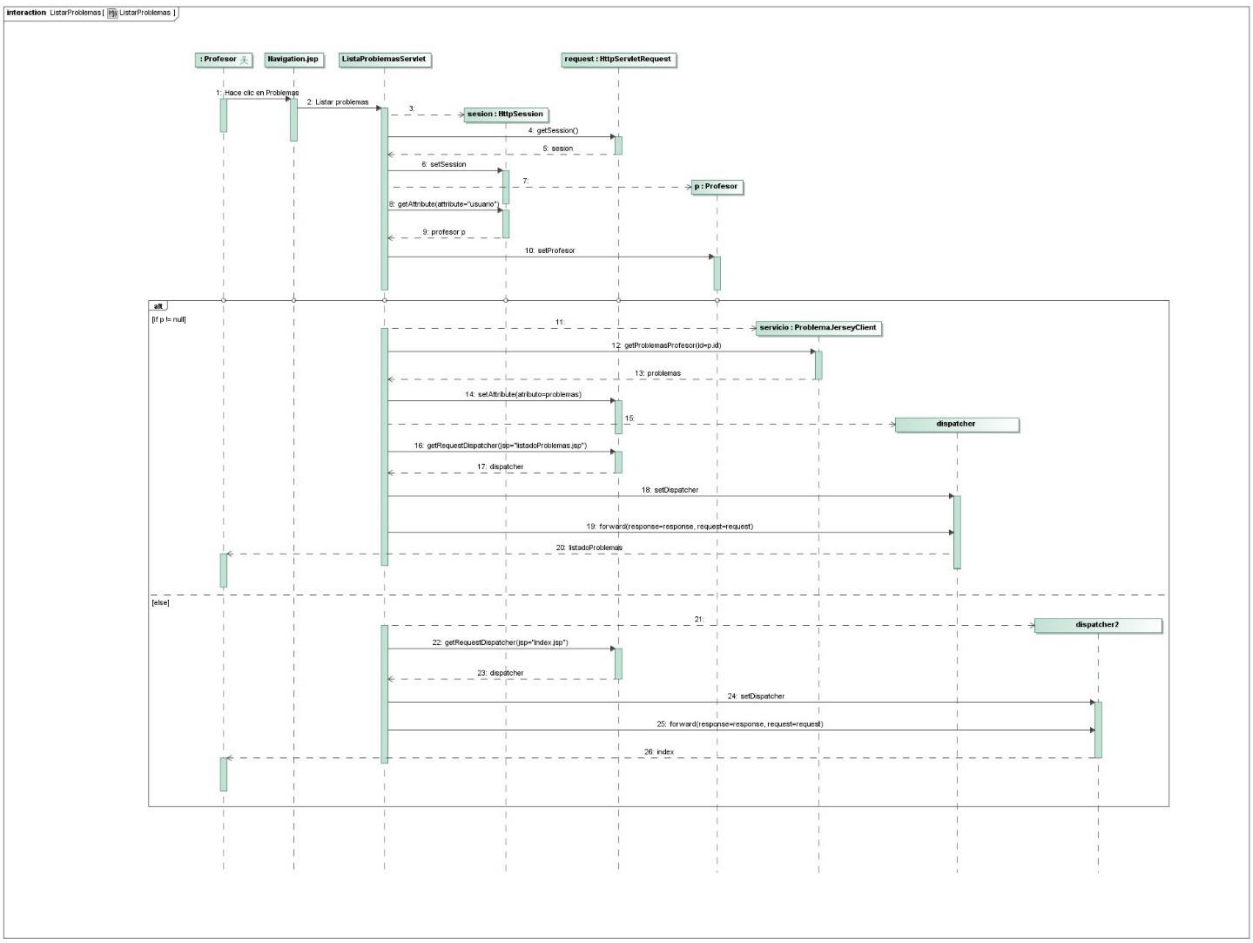


Ilustración 10 - Listar problemas (Diagrama de secuencia UML)

3.2.5. RF5 – Editar Problema

Título	Editar Problema
Descripción	Un profesor desea modificar un problema realizado previamente.
Pre-condición	Existe un problema que un profesor desea cambiar, ya sea porque se ha equivocado realizándolo o que desee modificar la solución.
Post-condición	El problema quedará almacenado correctamente.
Prioridad	Alta
Autor(es)	Profesor
Escenario principal	

1. <<Include>> ListarProblemas 2. El profesor hace clic en el botón editar correspondiente al problema. 3. El profesor cambia el título, asignatura, enunciado o la solución planteada. 4. El profesor hace clic en el botón Actualizar. 5. El sistema notifica diciendo que el problema se ha actualizado.	
Escenario alternativo	
3.b. El profesor no realiza ningún cambio. 4.b. El profesor hace clic en el botón actualizar. 5.b. El sistema notifica diciendo que el problema se ha actualizado.	
Clases de análisis	
A. Clases de entidad	Profesor
B. Clases de control	
C. Clases de interfaz	editarProblema.jsp
Maquetas de interfaz	

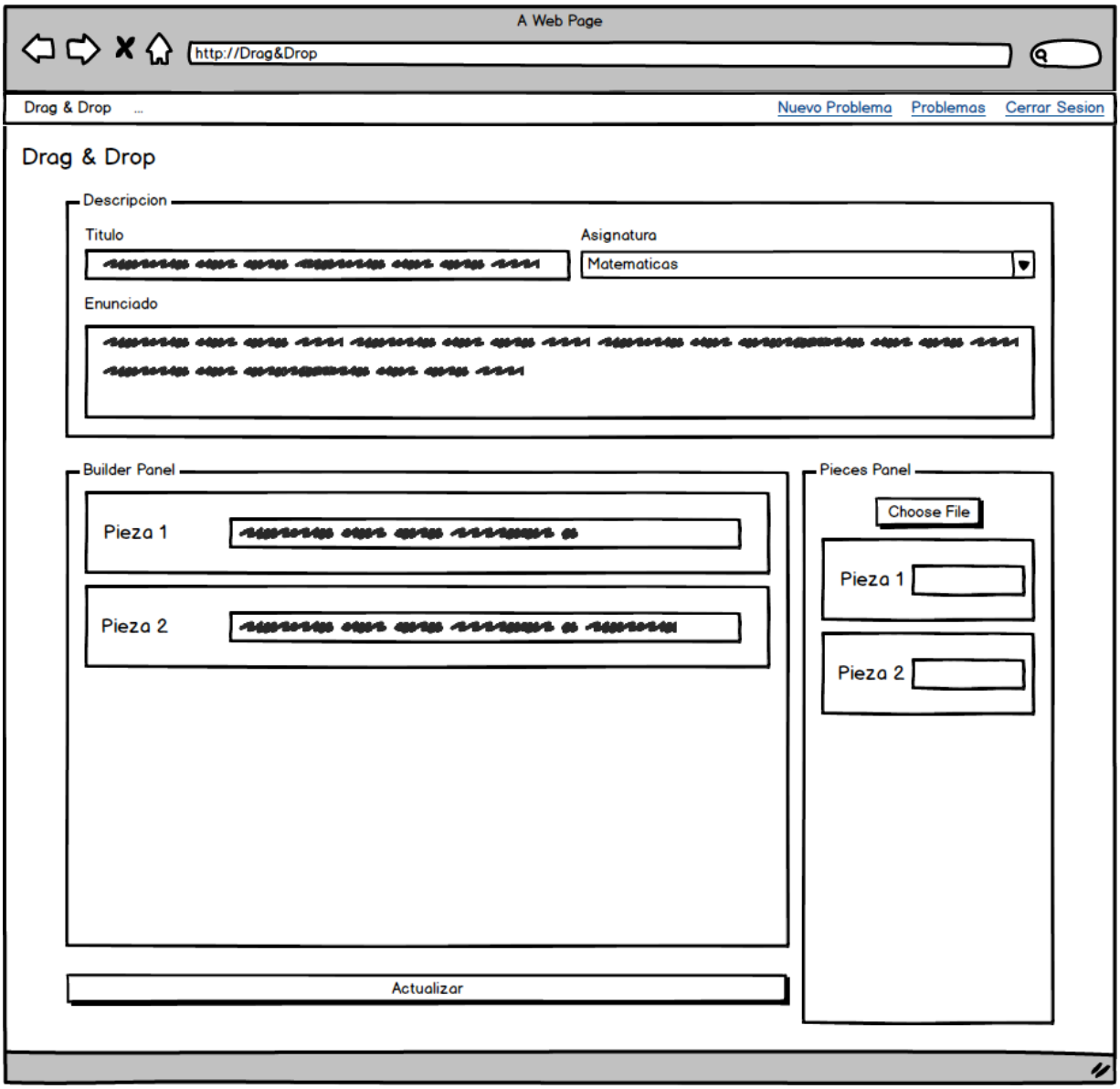


Ilustración 11 - Mockup editarProblema.jsp

Diagramas de secuencia

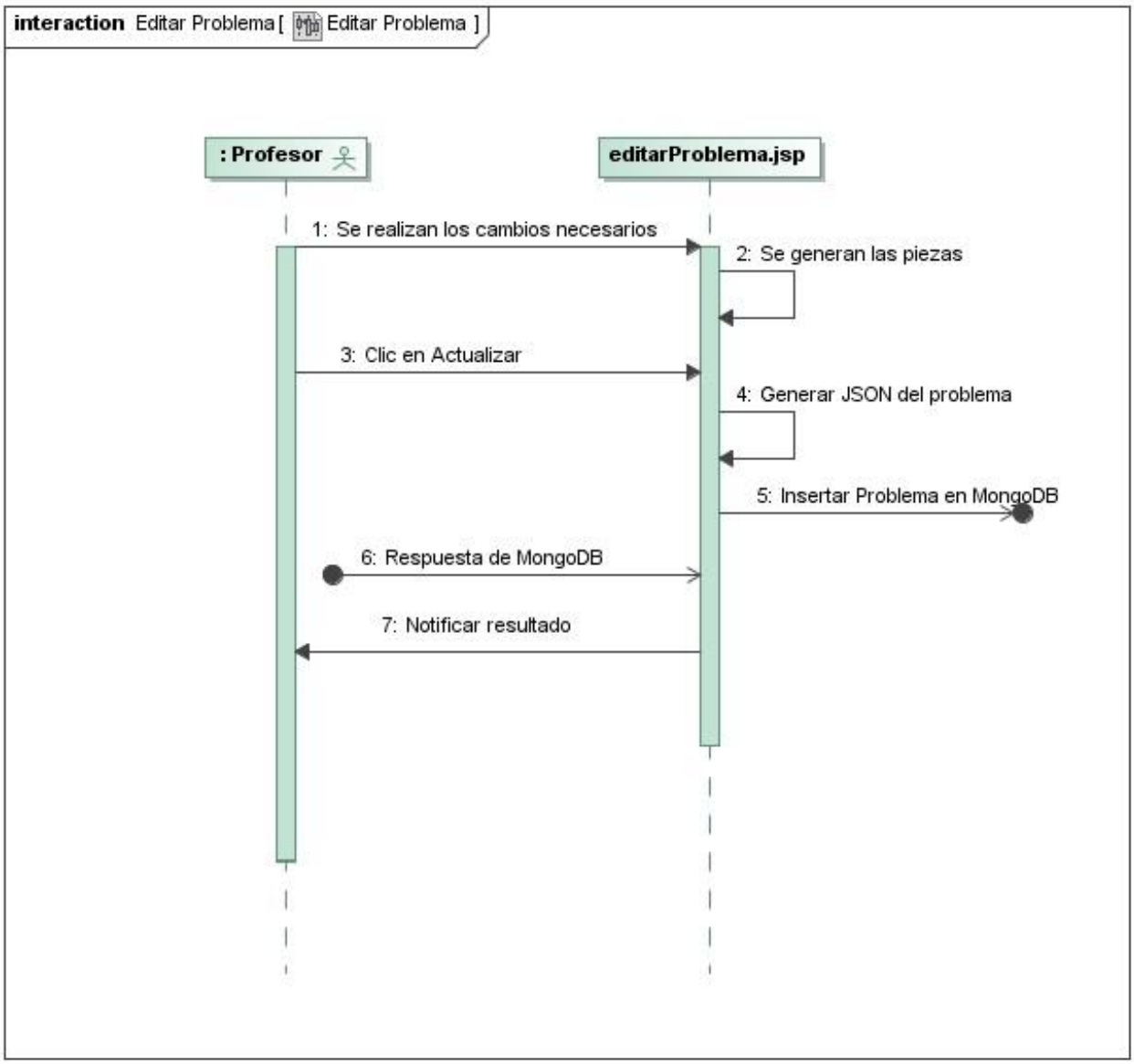


Ilustración 12 - Editar problema (Diagrama de secuencia UML)

3.2.6. RF6 – Borrar Problema

Título	Borrar Problema
Descripción	Un profesor desea eliminar un problema realizado.
Pre-condición	Existe un problema que un profesor desea eliminar.
Post-condición	El problema se eliminará del almacenamiento secundario correctamente.
Prioridad	Baja

Autor(es)	Profesor
Escenario principal	
1. <<Include>> ListarProblemas 2. El profesor hace clic en el botón borrar correspondiente al problema. 3. El sistema notifica diciendo que el problema se ha eliminado correctamente. 4. El sistema actualiza el listado de problemas.	
Escenario alternativo	
2.b. No existen problemas que eliminar.	
Clases de análisis	
A. Clases de entidad	Profesor, Problema
B. Clases de control	EliminarProblemaServlet, ProblemaJerseyClient
C. Clases de interfaz	listadoProblemas.jsp
Maquetas de interfaz	

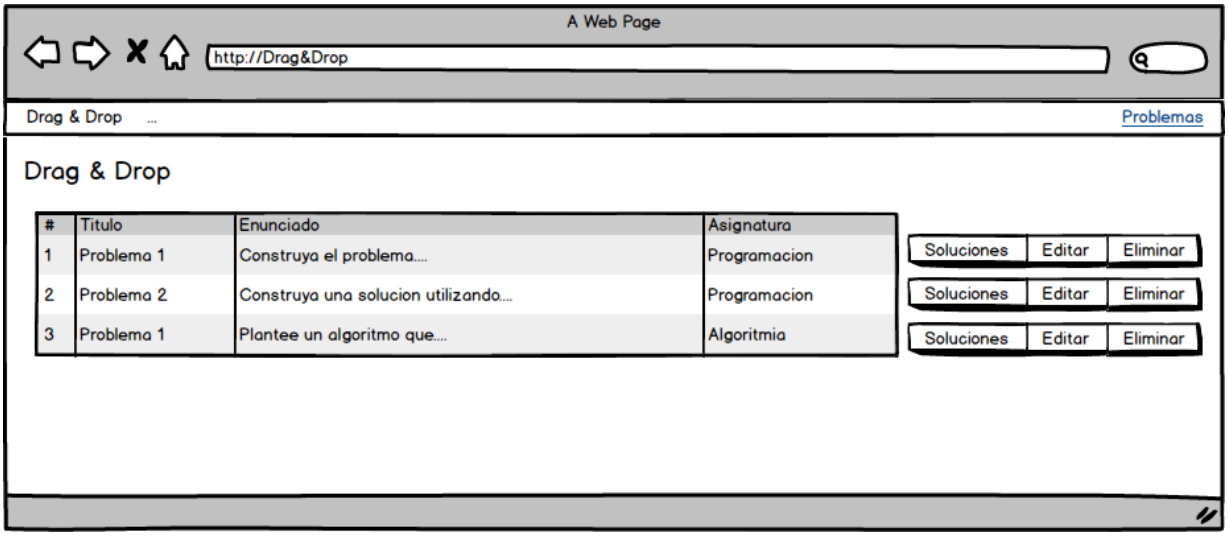


Ilustración 13 - Mockup listadoProblemas.jsp

Diagramas de secuencia

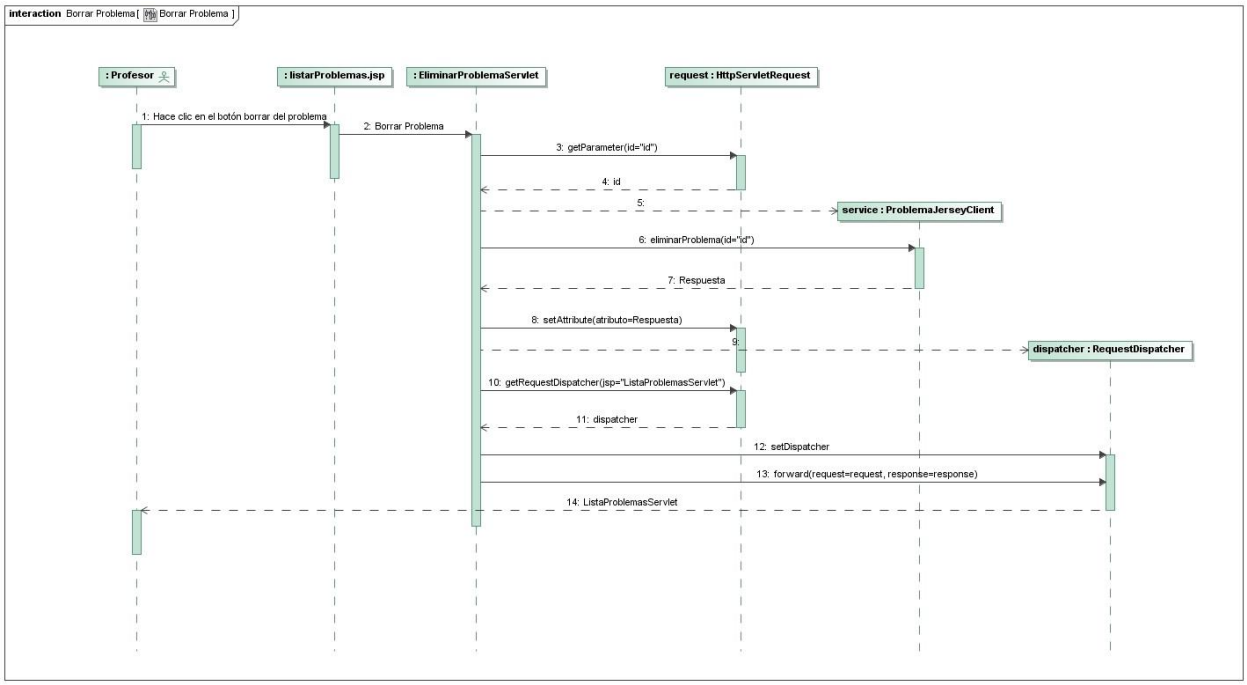


Ilustración 14 - Borrar Problema (Diagrama de secuencia UML)

3.2.7. RF7 – Ver Problema

Título	Ver Problema
Descripción	Un profesor desea ver un problema realizado previamente.
Pre-condición	Existe un problema que un profesor desea visualizar.
Post-condición	El problema se visualiza en pantalla.
Prioridad	Alta
Autor(es)	Profesor
Escenario principal	
1. <<Include>> ListarProblemas 2. El profesor hace clic en el botón editar correspondiente al problema. 3. El sistema busca en la persistencia el problema solicitado. 4. El sistema muestra el problema en pantalla.	
Escenario alternativo	
1.b. No existen problemas realizados.	

Clases de análisis	
A. Clases de entidad	Profesor, Problema
B. Clases de control	EditarProblemaServlet, ProblemaJerseyClient
C. Clases de interfaz	listarProblemas.jsp, editarProblema.jsp

Maquetas de interfaz

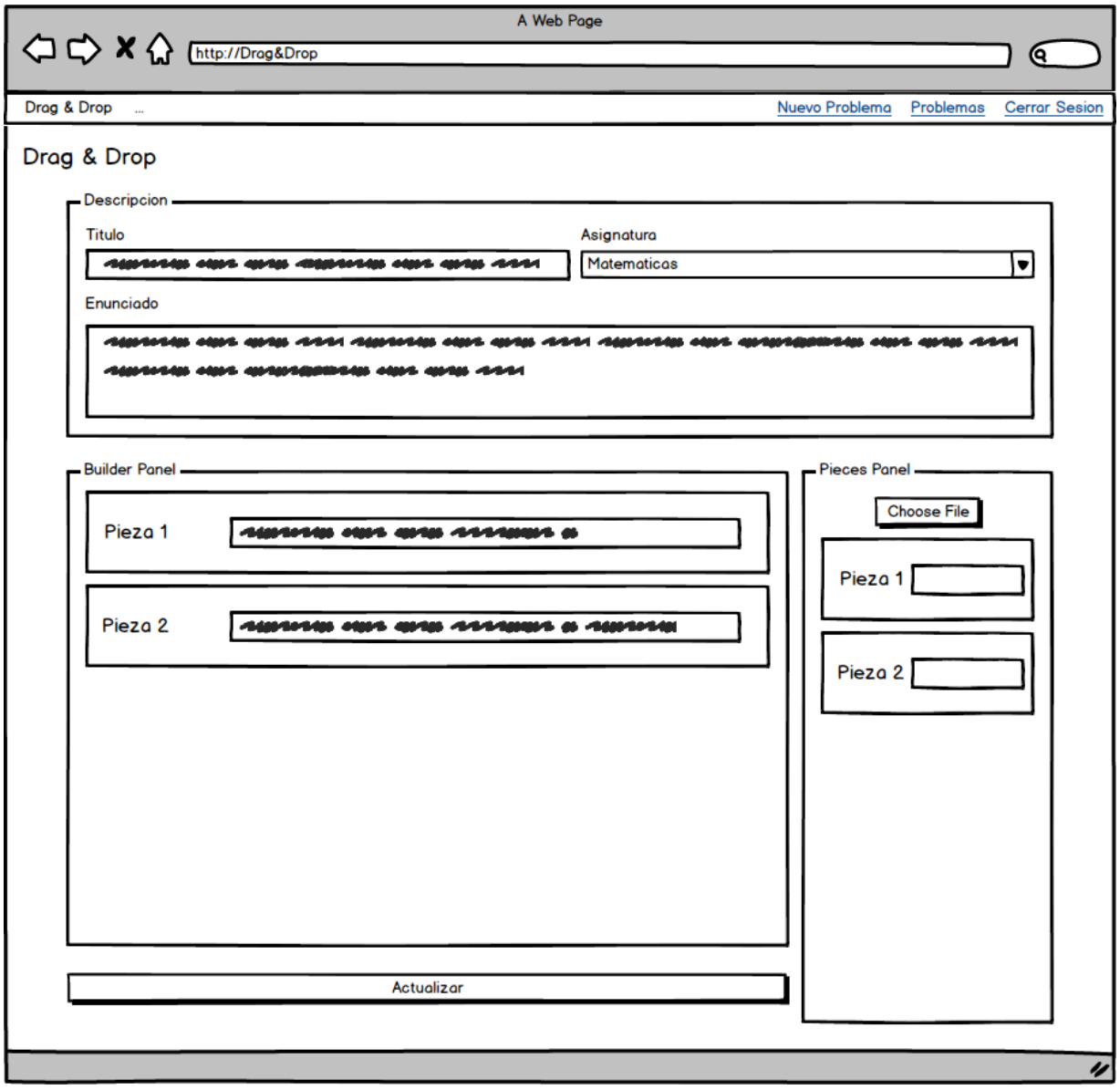


Ilustración 15 - Mockup editarProblema.jsp

Diagramas de secuencia

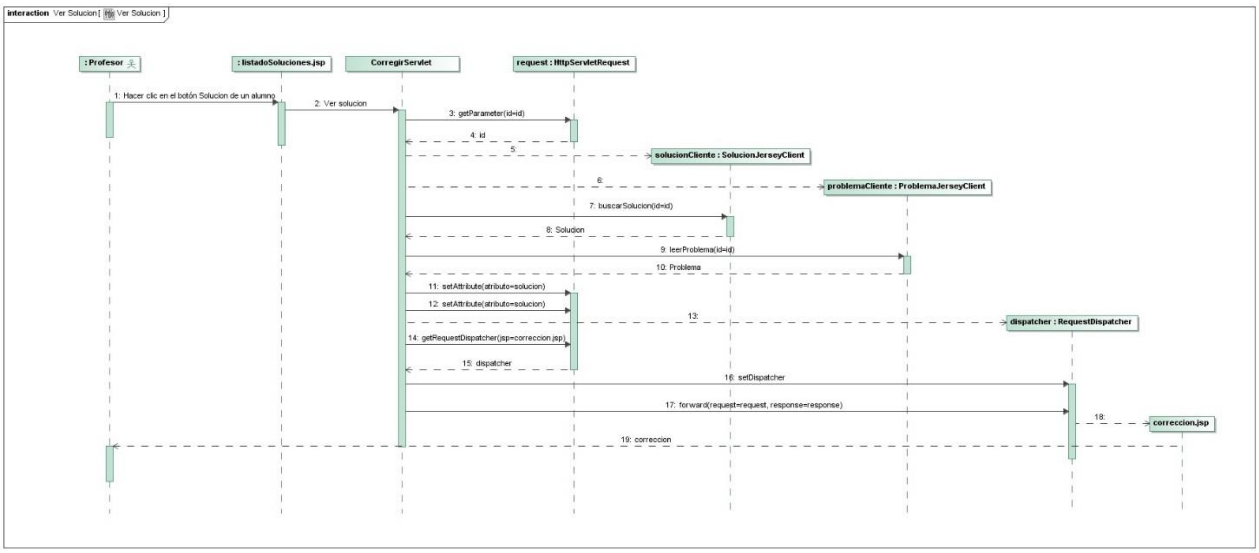


Ilustración 16 - Ver problema (Diagrama de secuencia UML)

3.2.8. RF8 – Listar Soluciones

Título	Listar Soluciones
Descripción	Un profesor desea listar todas las soluciones propuestas por los alumnos.
Pre-condición	Un profesor desea ver el listado de soluciones propuestas por los alumnos a un problema.
Post-condición	El profesor puede ver todas las soluciones en un listado.
Prioridad	Media
Autor(es)	Profesor
Escenario principal	
1. <<Include>> ListarProblemas. 2. El profesor hace clic en el botón “Soluciones” correspondiente al problema. 3. El sistema busca las soluciones correspondientes. 4. El sistema visualiza el listado de soluciones al problema seleccionado.	
Escenario alternativo	
4.b. No existen soluciones que listar.	
Clases de análisis	

A. Clases de entidad	Profesor, Problema
B. Clases de control	SolucionServlet, SolucionJerseyClient, ProblemaJerseyClient
C. Clases de interfaz	listadoProblemas.jsp, listadoSoluciones.jsp
Maquetas de interfaz	

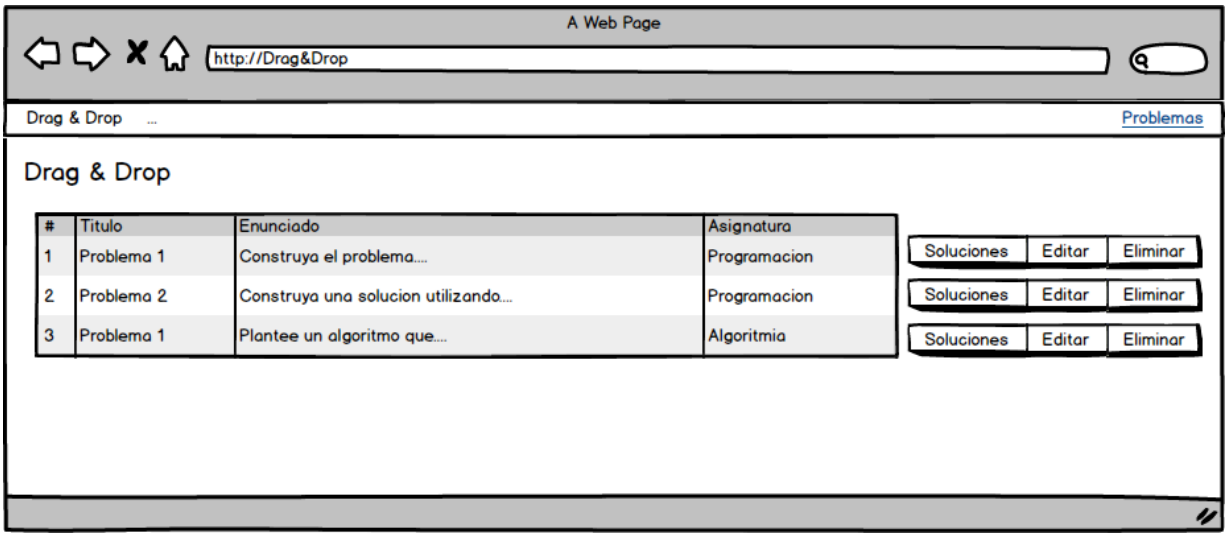


Ilustración 17 - Mockup listadoProblemas.jsp

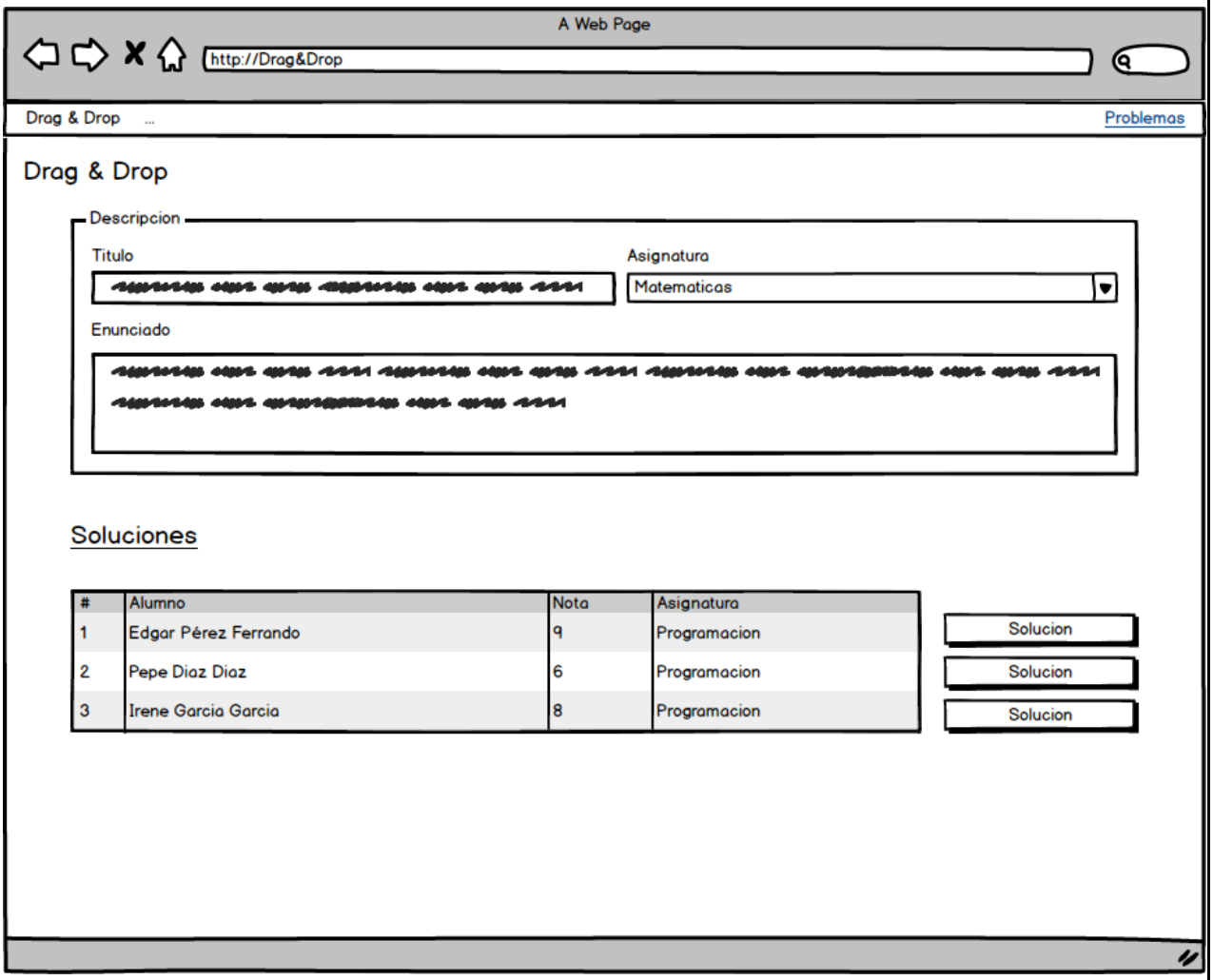


Ilustración 18 - Mockup listadoSoluciones.jsp

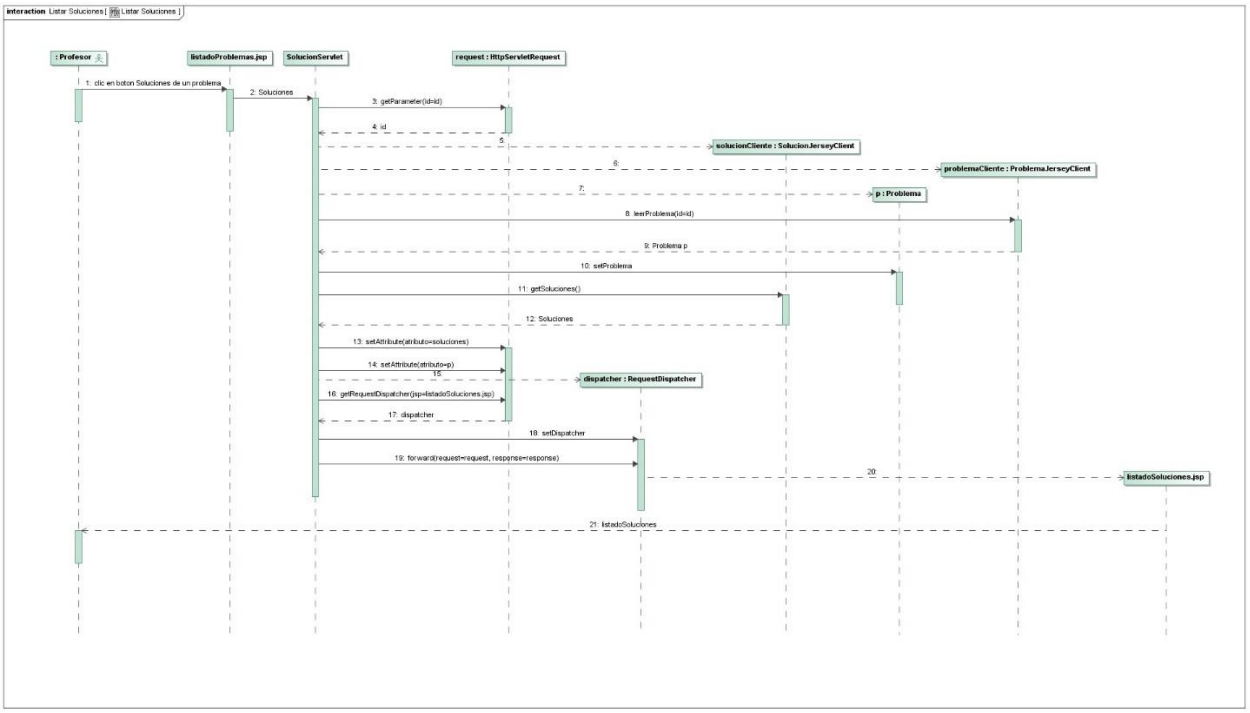
Diagramas de secuencia


Ilustración 19 - Listar soluciones (Diagrama de secuencia UML)

3.2.9. RF9 – Visualizar una solución

Título	Ver una solución
Descripción	Un profesor desea ver la solución planteada por un alumno a un problema planteado.
Pre-condición	El profesor quiere ver la solución propuesta por un alumno.
Post-condición	El profesor ha contemplado la solución.
Prioridad	Media
Autor(es)	Profesor
Escenario principal	

1. <<Include>> ListarProblemas. 2. <<Include>> ListarSoluciones. 3. El profesor hace clic en el botón “Solución” asociado a un alumno. 4. El sistema carga la vista con la solución planteada y la nota generada automáticamente.	
Escenario alternativo	
3.b. No existen soluciones propuestas.	
Clases de análisis	
A. Clases de entidad	Profesor, Solucion, Problema
B. Clases de control	CorregirServlet, SolucionJerseyClient, ProblemaJerseyClient
C. Clases de interfaz	listadoSoluciones.jsp, corrección.jsp
Maquetas de interfaz	

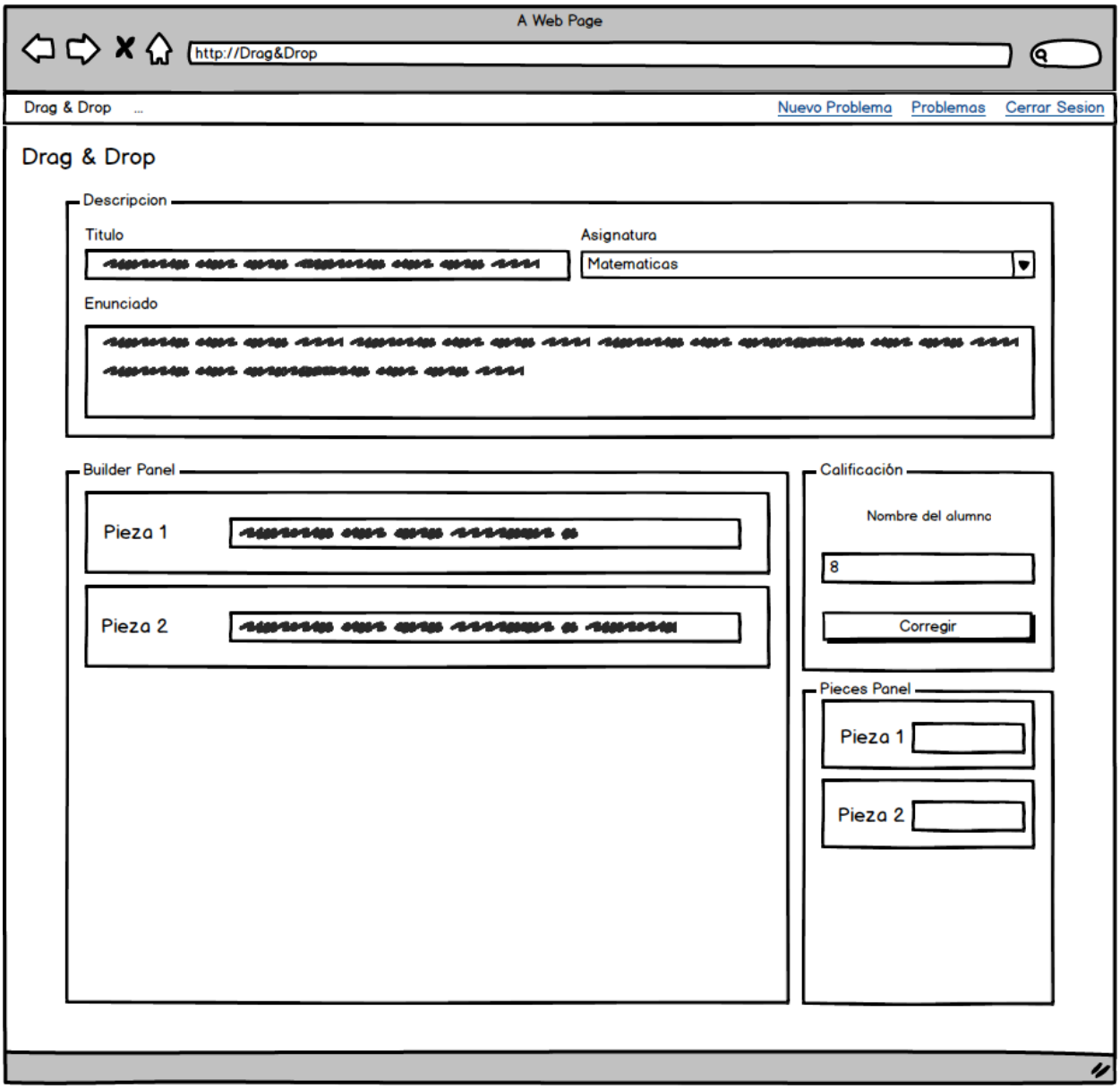


Ilustración 20 - Mockup correccion.jsp

Diagramas de secuencia

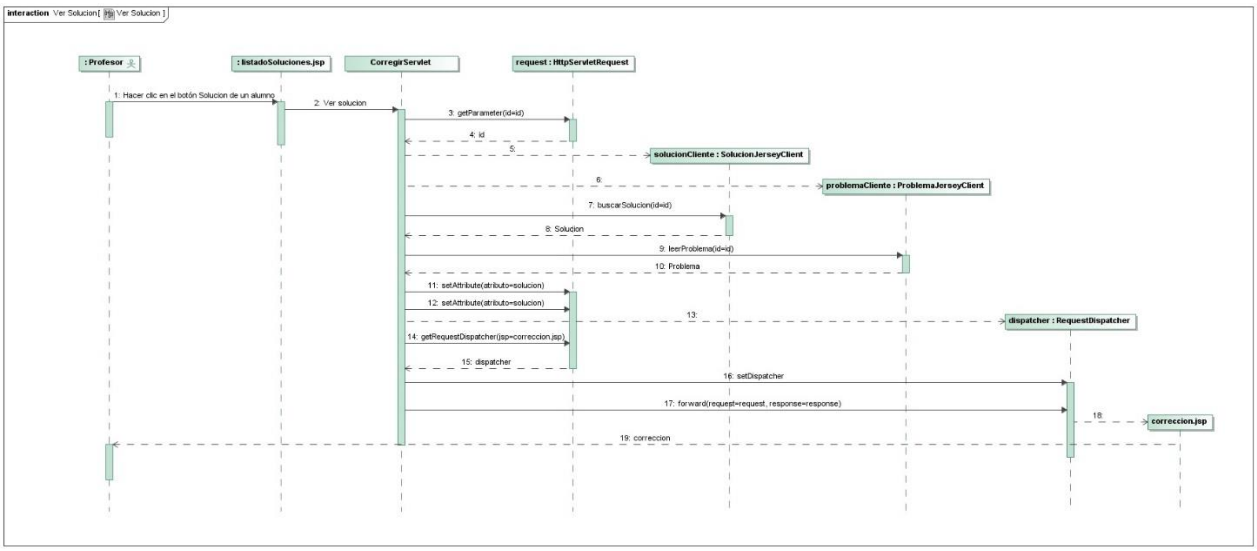


Ilustración 21 - Ver Solucion (Diagrama de secuencia UML)

3.2.10. RF10 – Corregir soluciones

Título	Corregir Soluciones
Descripción	Un profesor desea ver (y/o evaluar) la solución planteada por un alumno a un problema planteado.
Pre-condición	El profesor quiere ver (y/o evaluar) la solución propuesta por un alumno.
Post-condición	El profesor ha contemplado la solución y evaluado en caso de que no esté de acuerdo con la evaluación automática.
Prioridad	Media
Autor(es)	Profesor
Escenario principal	
1. <<Include>> ListarProblemas. 2. <<Include>> ListarSoluciones. 3. <<Include>> Ver Solucion. 4. El profesor corrige la nota generada en el campo de texto donde se visualiza. 5. El profesor hace clic en el botón Corregir. 6. El sistema actualiza la calificación en la persistencia y recarga la vista.	
Escenario alternativo	

3.b. El profesor no cambia la nota generada por el sistema.	
Clases de análisis	
A. Clases de entidad	Profesor
B. Clases de control	CorregirCalificacionServlet
C. Clases de interfaz	corrección.jsp
Maquetas de interfaz	

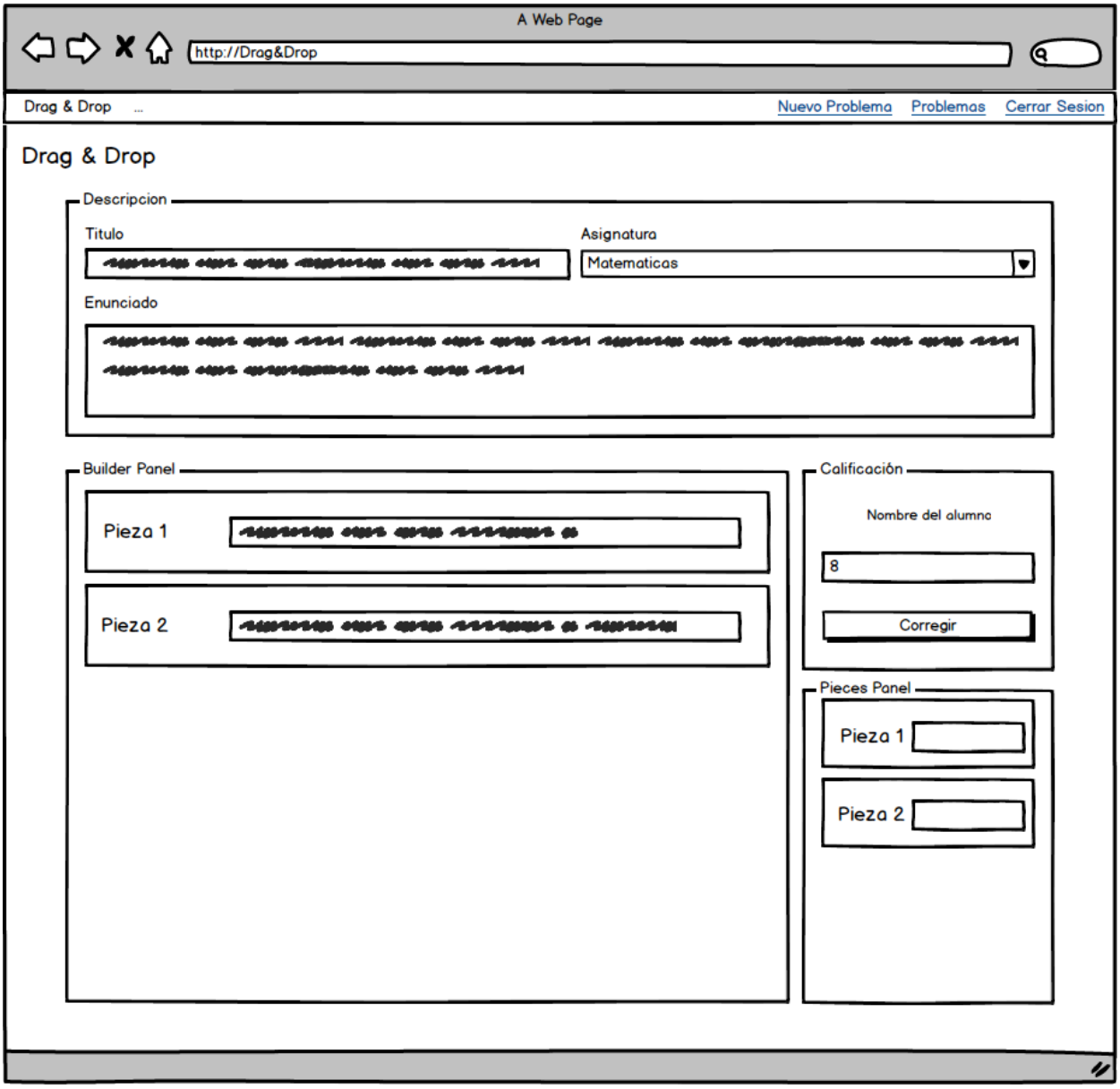


Ilustración 22 - Mockup correccion.jsp

Diagramas de secuencia

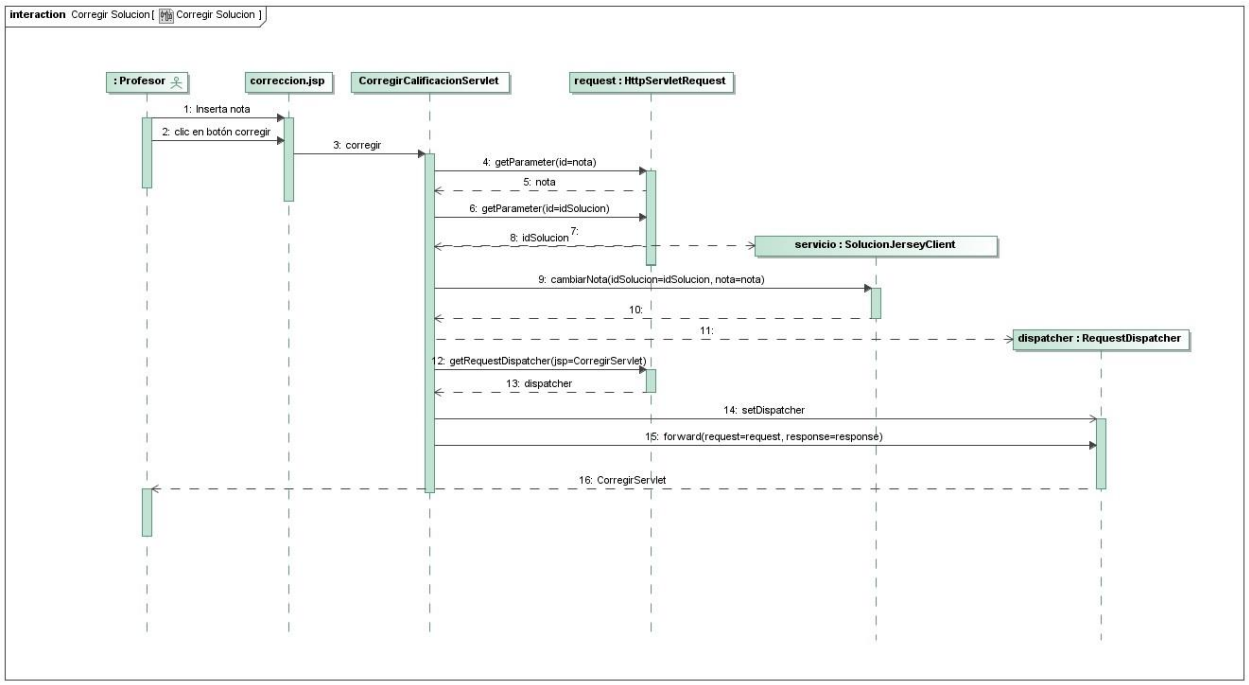


Ilustración 23 - Corregir Solucion (Diagrama de secuencia UML)

3.3. Diagrama de navegación

Con un diagrama de navegación podemos entender un poco cómo ir de una parte de la aplicación a otra con facilidad, así como aclarar los puntos de acceso a las diferentes funcionalidades del sistema.

La aplicación es una aplicación web, por lo tanto, siempre se debe de tener un menú de navegación rápido y fácilmente manejable. Para ello listaremos las principales opciones de navegación “Nuevo problema”, ”Problemas” y cerrar sesión.

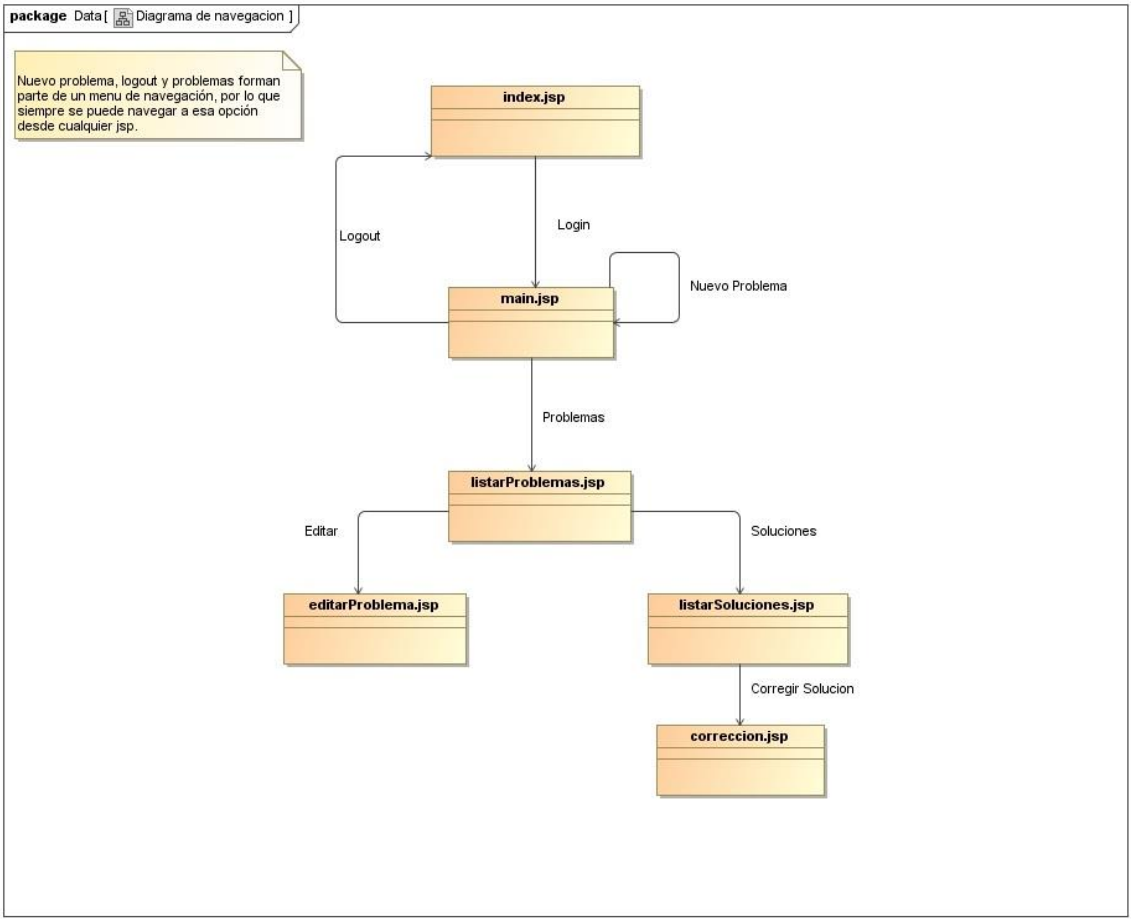


Ilustración 24 - Navegación entre ficheros JSP (Diagrama de navegación UML)

El diagrama muestra el flujo de navegación entre las vistas (jsp) normal, sin embargo, si la sesión del profesor expira o se encuentra algún error de navegación, la aplicación volverá a la vista “index.jsp” para que el profesor vuelva a iniciar la sesión.

3.4. Diagrama de Actividad (Insertar Problema)

Viendo el diagrama de secuencia, puede que quede un poco difuso el procedimiento para insertar un nuevo problema en la persistencia. Es por ello que a través de un diagrama de actividad veremos cómo se realiza el proceso a más bajo nivel.

Cabe destacar que para éste caso en particular, hemos preferido utilizar AJAX. De esta forma no nos hace falta utilizar ningún servlet, así como un objeto cliente que conecte con un servicio.

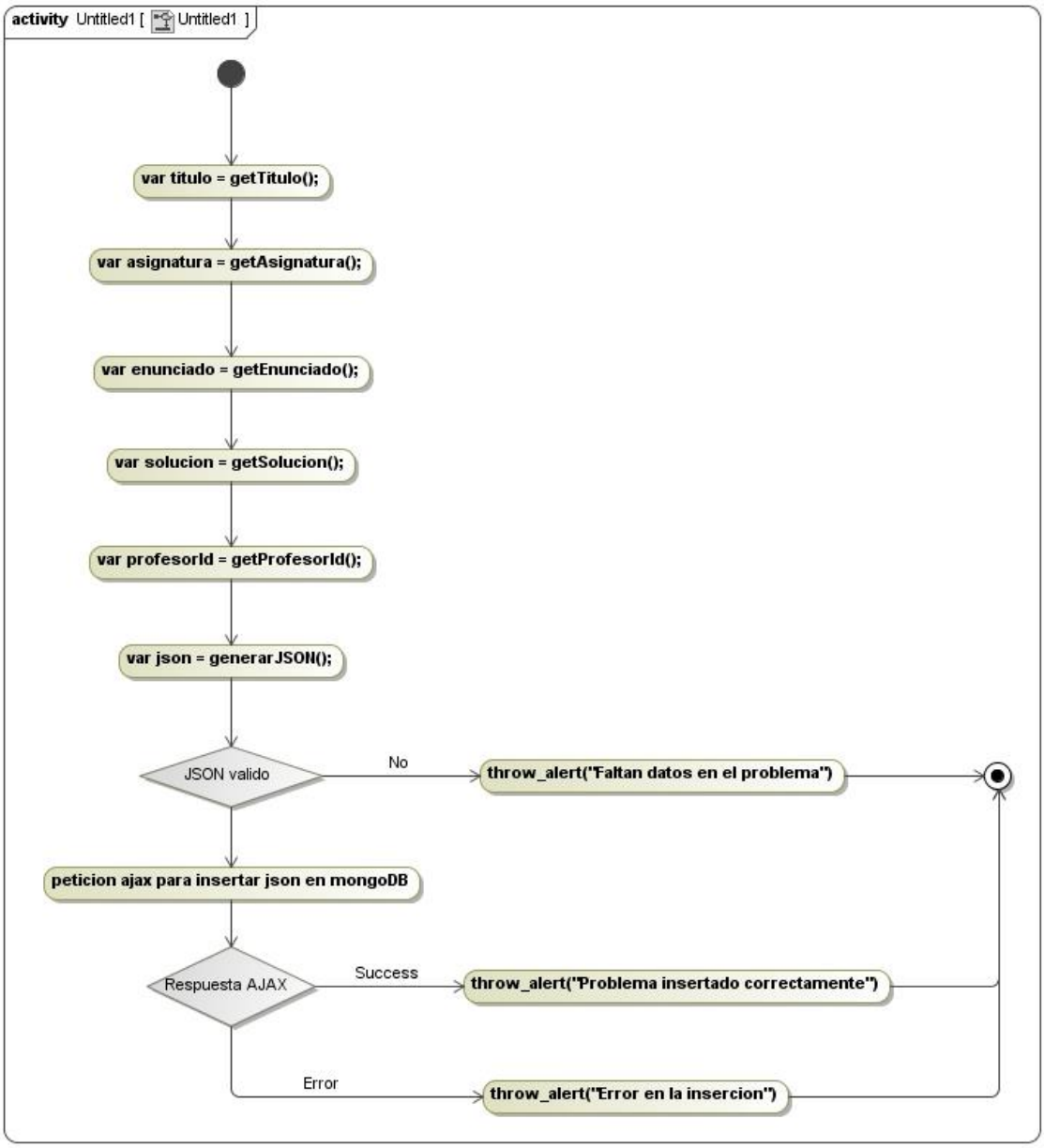


Ilustración 25 - Insertar problema (Diagrama de actividad UML)

3.5. Diagrama de componentes (Servicios)

A continuación mostramos un diagrama de componentes UML. El objetivo es principalmente entender cuáles son los clientes java que utilizamos en nuestra aplicación y ver con qué servicios del servidor se conectan.

Al lado izquierdo tenemos nuestra aplicación del lado del profesor y al lado derecho el servidor mongo.

Existen más servicios que ofrece el servidor, sin embargo, para nuestra aplicación solo utilizamos los cuatro que se ven en la ilustración 26.

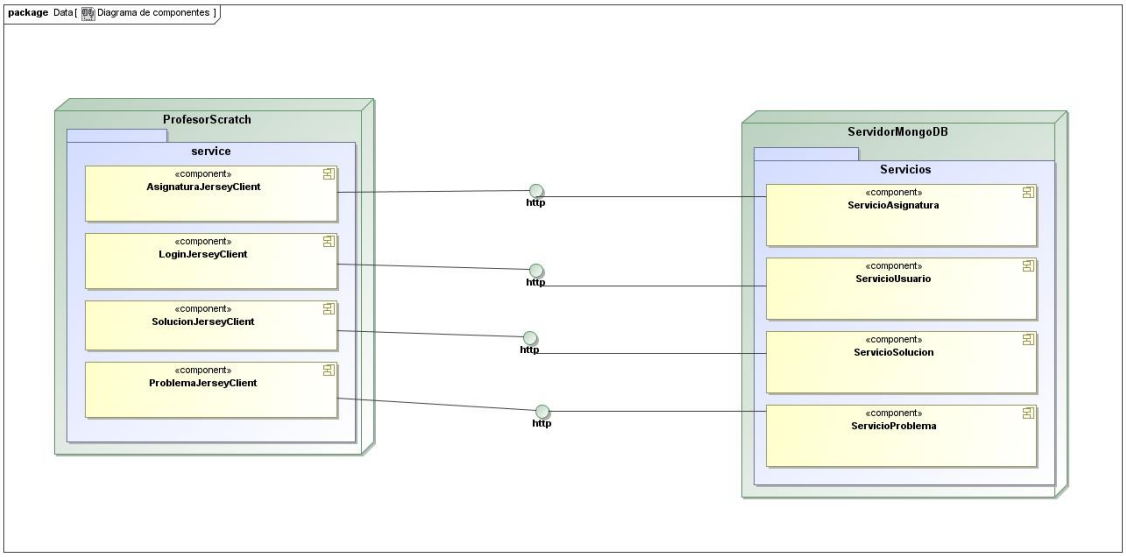


Ilustración 26 - Conexión entre clientes y servicios (Diagrama de componentes UML)

Para entender mejor las tareas que ocupan a cada servicio, se ha realizado el siguiente diagrama de clases que muestra las operaciones principales que realiza.

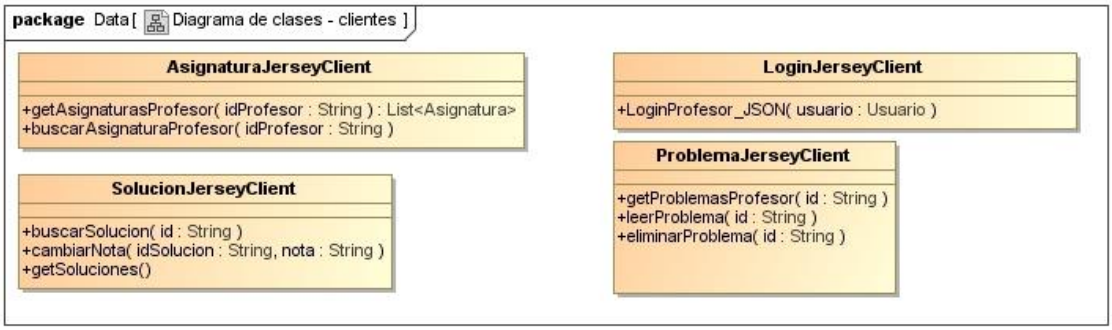


Ilustración 27 - Clases de los clientes web (Diagrama de clases UML)

3.6. Estructuras de los ficheros JSON

Para realizar y utilizar la aplicación se van a utilizar dos ficheros JSON, uno de ellos es el que introducirá el profesor en la aplicación para generar las piezas y el otro es el que se genera automáticamente para enviar la aplicación de MongoDB.

3.6.1. JSON de construcción de piezas

Este JSON es el que tiene que importar el profesor en la aplicación para generar las piezas y poder construir la solución a un problema. Éste mismo es el que se almacenará en otro JSON posterior en el proceso de inserción de un problema (el campo “piezas”) y almacenarlo en el servidor de MongoDB. De este modo, se podrán regenerar las piezas utilizadas para que el alumno plantee su solución.

A continuación se detalla un ejemplo de la estructura deseada de un JSON para la construcción de las piezas:

```
[{
"inputs": [
        { "type": "label","value": "Pieza 1"},
        {"type": "text", "value": 1}
    ] },
     { "inputs": [
        { "type": "label", "value": "Pieza 1" },
        {"type": "text","value": 1}
    ]},{
"inputs": [
        {"type": "select","value": ["op1","op2","op3"]}
    ]}, {
"inputs": [
        {"type": "label","value": "Pieza 1"},
        {"type": "text","value": 1}
    ]}, {
"inputs": [
        {"type": "label","value": "Pieza 1"},
        {"type": "text","value": 1},
        {"type": "label","value": " ok"}
 ] }]
```

Se compone de una lista de elementos denominados “inputs”, en realidad determina una pieza visual en la aplicación. A su vez, un elemento input se compone los elementos html. Esto quiere da lugar a que puedan existir tres tipos en nuestra aplicación, “label”, “text” y “select”.

Un campo de tipo “label” hace referencia a una etiqueta de texto en html y el campo value nos da el valor en caracteres del texto en sí.

Por ejemplo, si tenemos el siguiente json:

```
{"type": "label","value": "Pieza 1"}
```

Equivaldrá al siguiente código html:

Pieza 1

Es importante que el archivo esté correctamente estructurado y balanceado, ya que si hay algún error la aplicación lo detectaría y no se generarían las piezas o generaría las que están correctamente.

3.6.2. JSON de envío a Mongo DB para crear un problema o editarlo.

Cuando insertamos un nuevo problema en nuestra aplicación, enviaremos un JSON con todos los datos correspondientes. Sin embargo, este no es un fichero físico como el anterior, este se genera de forma dinámica recolectando los datos introducidos en el formulario del problema (Ya sea en el caso de un nuevo problema o edición de uno existente).

El JSON tiene la siguiente estructura:

{“enunciado”:“enunciado”,

“titulo”:“titulo”,

“nombreAsignatura”:”asignatura”,

“piezas”:”jsonPiezas”,

“solución”:”solución”,

“idProfesor”:”idProfesor”};

El campo enunciado, será una cadena de texto con la pregunta que se desea realizar al alumno.

El título sirve para identificar un problema planteado y también se solicita al profesor que lo introduzca.

El nombre de la asignatura para asociarlo al propio problema, se ha planteado así para optimizar el funcionamiento de los servicios y aprovechar el almacenamiento NoSQL que ofrece MongoDB.

El campo piezas es el que representa las piezas de construcción que se tienen que utilizar para realizar la solución al problema. Es importante destacar que tiene la misma estructura que el JSON anterior.

El campo solución, es otro JSON con la misma estructura que el campo piezas, sin embargo, esta vez es importante destacar que representa la solución planteada al problema que se almacenará y utilizará para evaluar a los alumno mediante el algoritmo de corrección automático.

El campo idProfesor se utiliza para identificar al profesor que está utilizando la aplicación y asociar el problema a él mismo.

4. Implementación

La fase de implementación ha dado lugar a la estructura de proyecto que comentaremos brevemente:

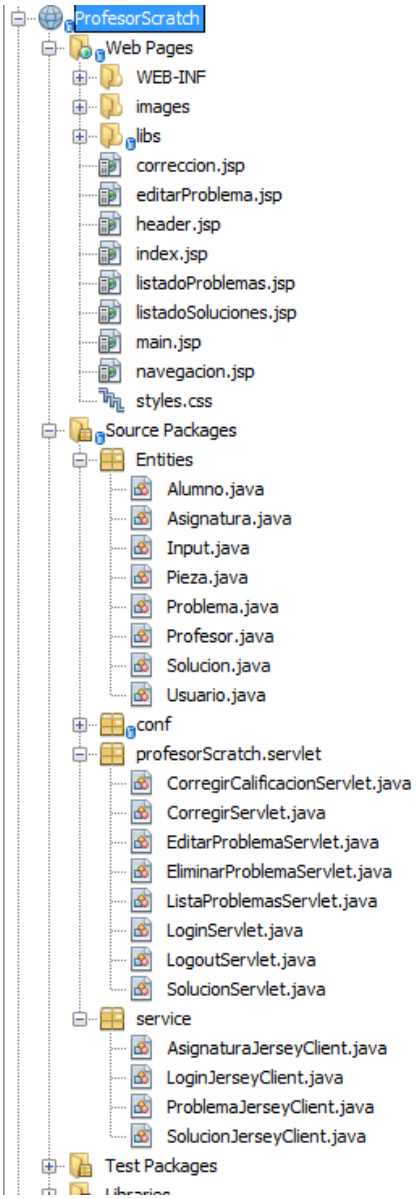


Ilustración 28 - Estructura del proyecto

4.1. Web Pages

El interfaz gráfico ha sido desarrollado completamente utilizando HTML5 en los ficheros .jsp para las páginas web. Los estilos de la aplicación se han aplicado mediante un framework llamado Bootstrap, que facilita la tarea de diseño y además utilizando CSS3.

A continuación mostramos las principales vistas totalmente desarrolladas de nuestra aplicación utilizando lo ya mencionado.

No se va a comentar sobre el desarrollo de las mismas, ya que es código HTML, JSP y JS. Posteriormente hablaremos sobre las funciones más relevantes.

4.1.1. Index.jsp

Esta es la página principal de nuestra aplicación, desde aquí un profesor debe de introducir el nombre de usuario y contraseña que se le ha facilitado para poder acceder a la aplicación.

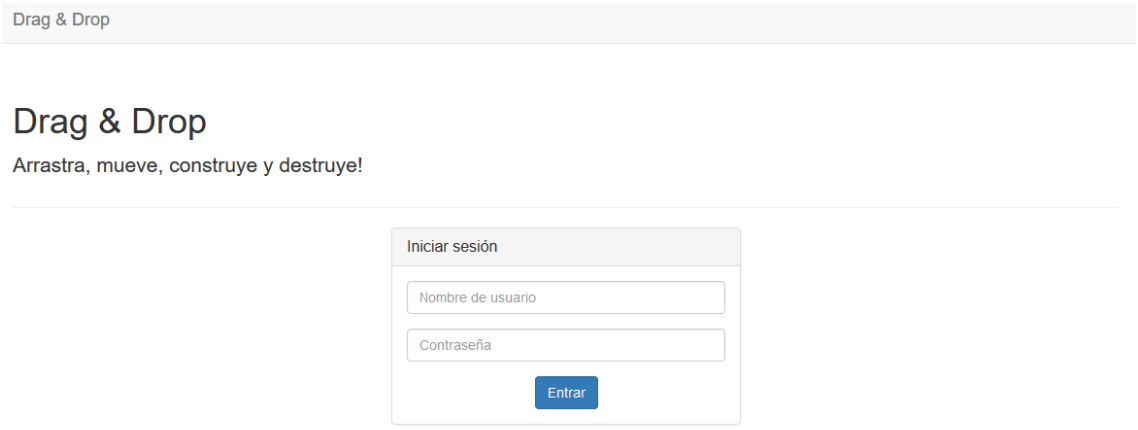


Ilustración 29 - index.jsp

4.1.2. Main.jsp

Por defecto, una vez que el profesor inicia sesión correctamente, se carga la siguiente vista. Desde la cual puede realizar un nuevo problema.

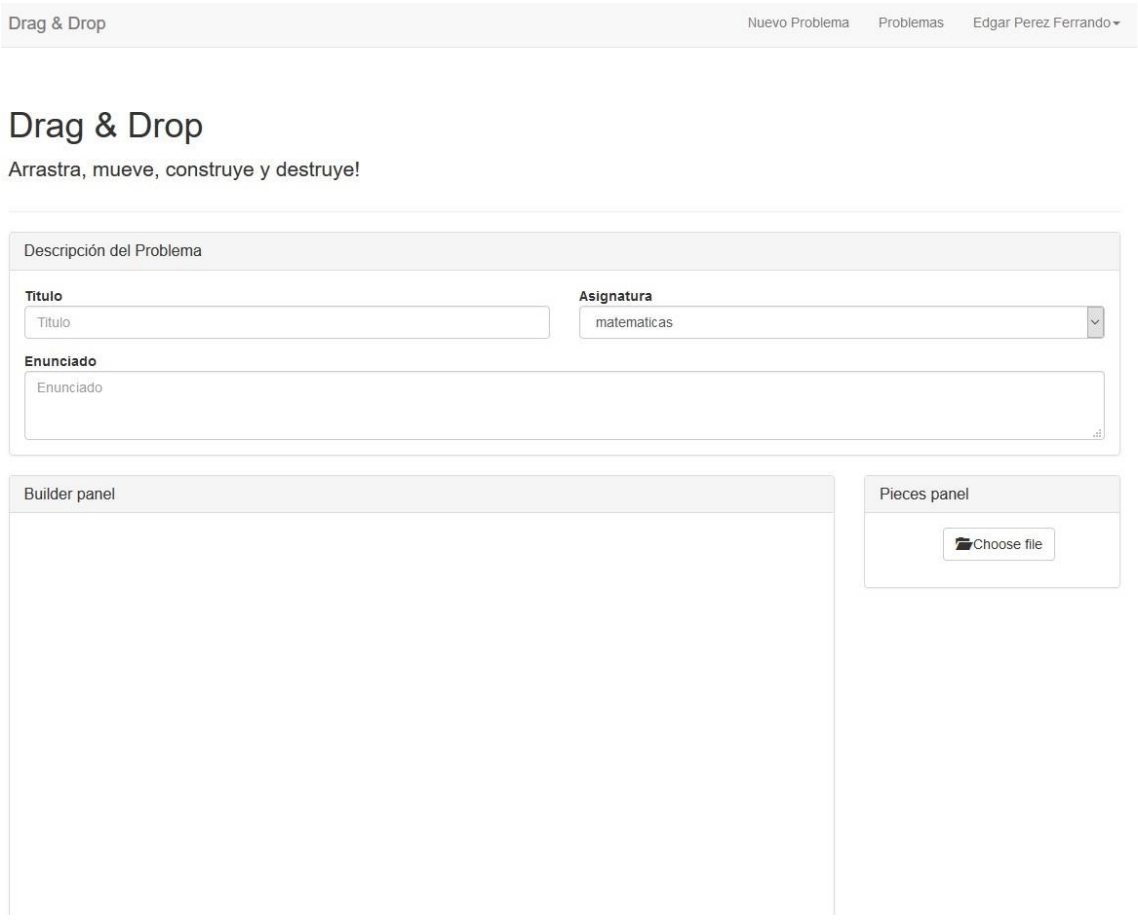


Ilustración 30 - main.jsp

4.1.3. navegacion.jsp

En realidad no es una vista como tal, se trata de la barra de navegación que se utiliza para acceder a diferentes partes de nuestra aplicación.

Esta barra de navegación se incluye en todos los jsp de nuestra aplicación.

Ilustración 31 - navegacion.jsp

4.1.4. header.jsp

Como en el caso anterior, este jsp no es una vista completa como tal. Se trata de la cabecera de la página web con el título del sitio que se incluye en el resto de jsp.

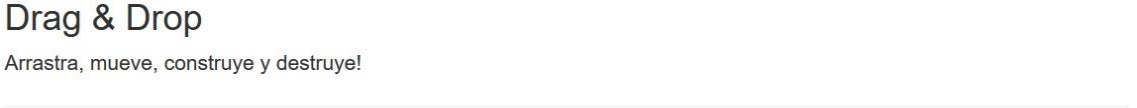


Ilustración 32 - header.jsp

4.1.5. editarProblema.jsp

La siguiente interfaz es la que se utiliza cuando se desea modificar algún problema ya realizado. Es la misma que utilizamos para visualizar el problema.

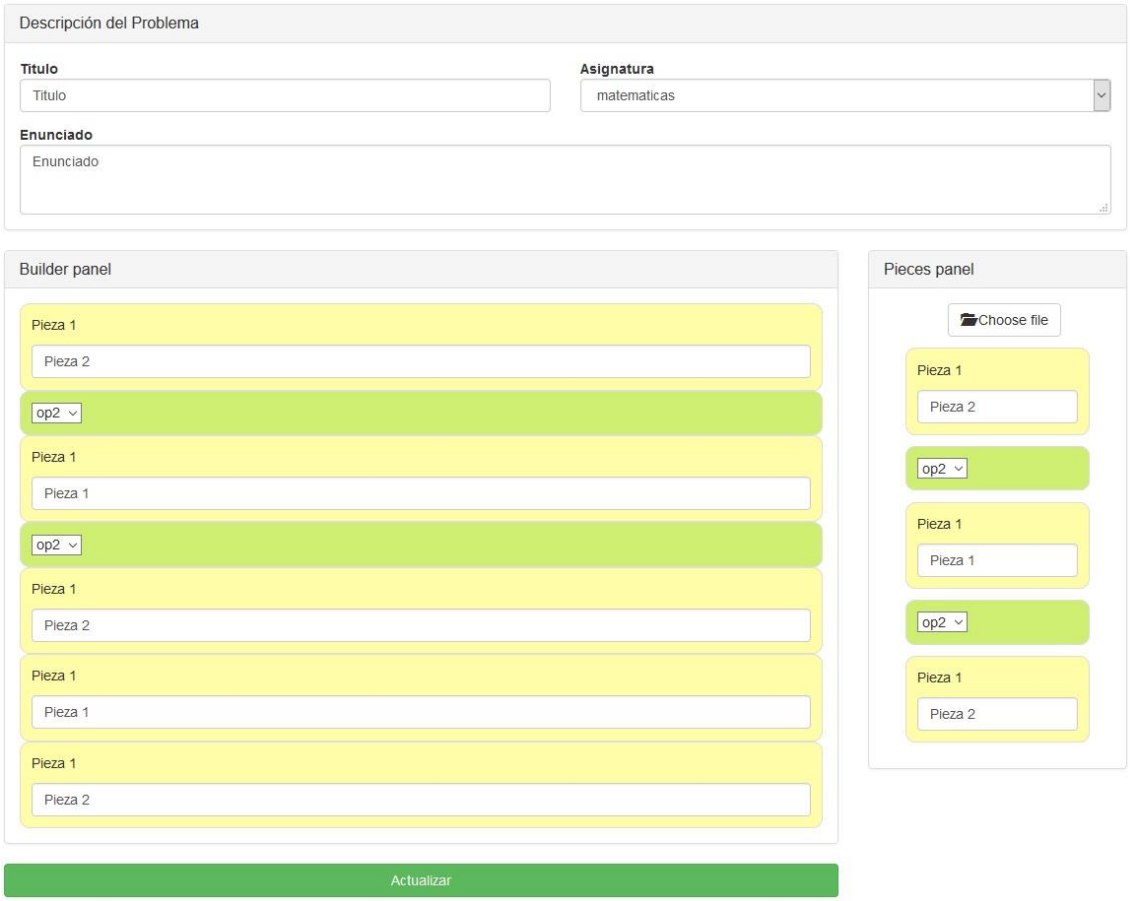


Ilustración 33 - editarProblema.jsp

4.1.6. listadoProblemas.jsp

Con esta interfaz seremos capaces de visualizar todos los problemas que hemos realizado previamente, así como la asignatura a la que pertenece y las acciones que podemos realizar sobre cada uno de ellos.

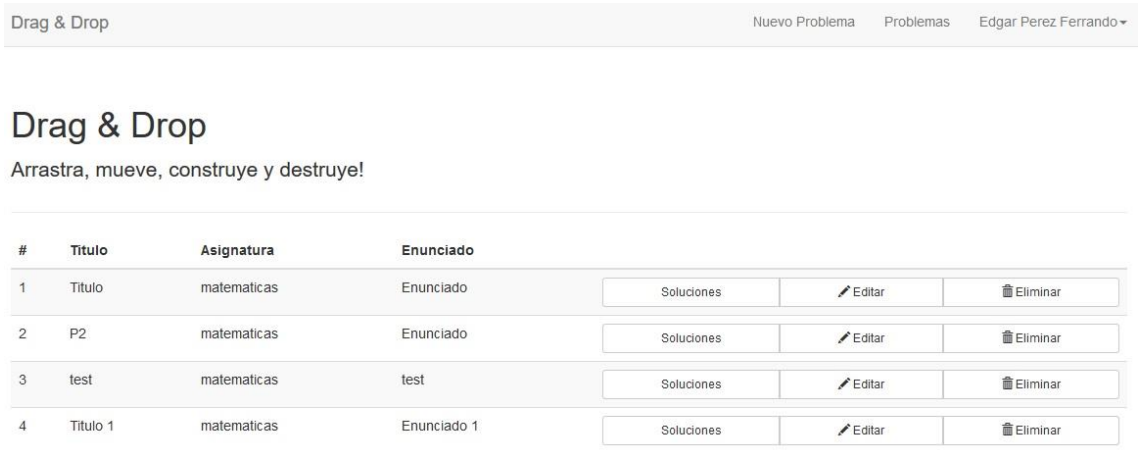


Ilustración 34 - listadoProblemas.jsp

4.1.7. listadoSoluciones.jsp

Similar a la vista anterior, podremos ver todas las soluciones propuestas a un problema listadas siempre que accedamos a este interfaz. Para ello, basta con hacer clic en el botón soluciones del problema asociado en la vista "listadoProblemas.jsp".

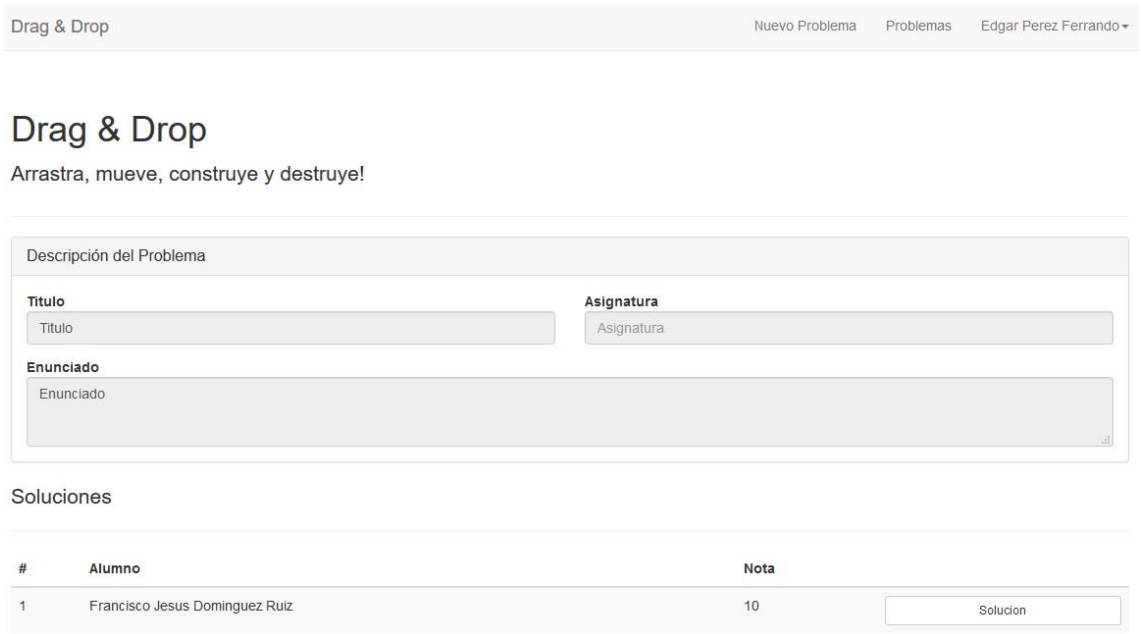


Ilustración 35 - listadoSoluciones.jsp

4.1.8. corrección.jsp

Por último la vista que empleamos para ver una solución individual de un alumno y también para corregir la calificación en caso de que el profesor lo desee.

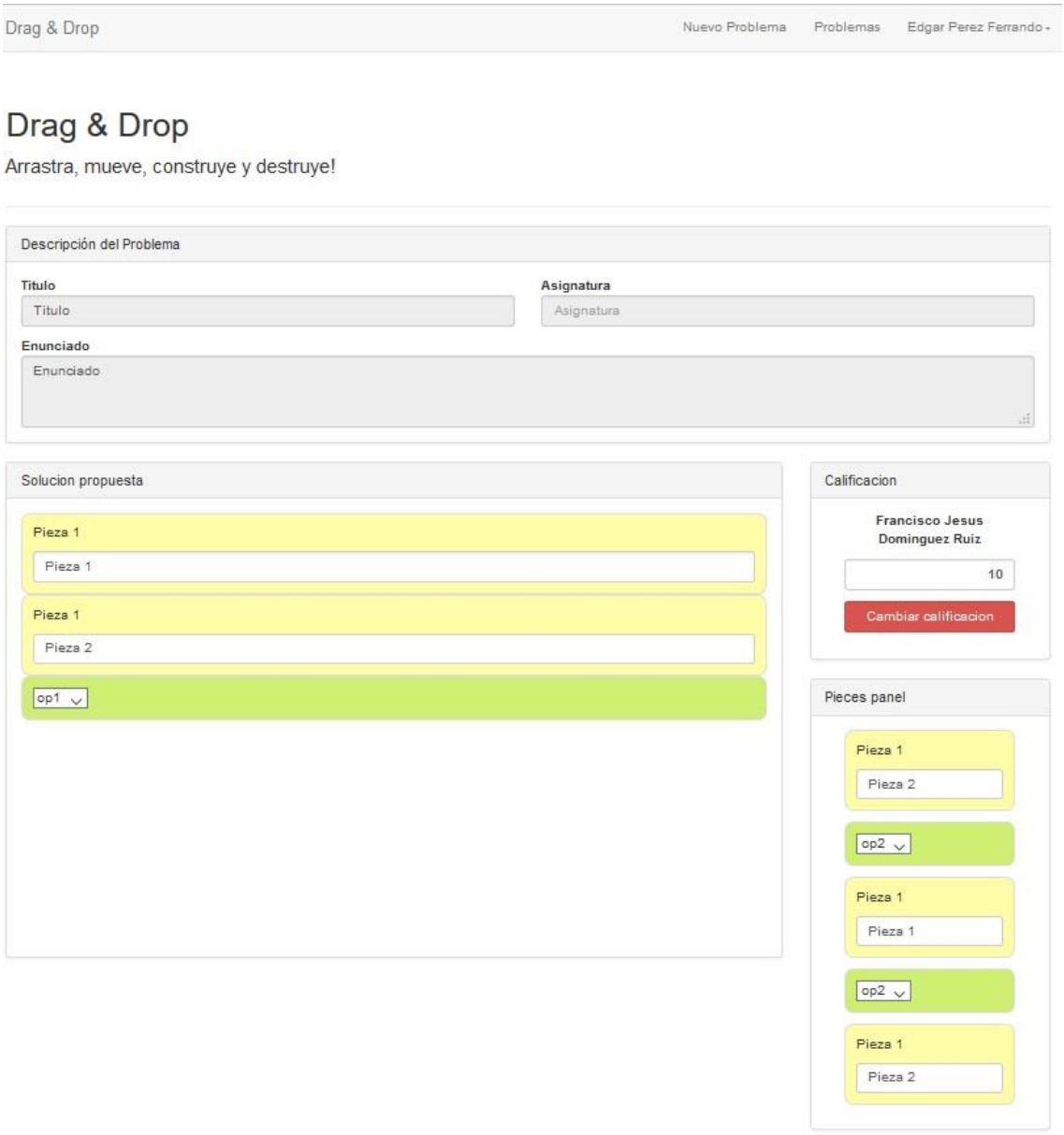


Ilustración 36 - corregir.jsp

4.1.9. Styles.css

Éste es nuestro fichero css particular, como se ha dicho anteriormente, hemos utilizado el framework Bootstrap para aplicar los estilos en el diseño del interfaz gráfico. A pesar de ello, no es suficiente para ajustar el diseño a nuestro gusto por lo tanto hemos aplicado los estilos restantes en ese archivo, el cual se incluye en todos los JSP de nuestra aplicación.

4.1.10. Directorio libs

En éste directorio hemos alojado los ficheros que hemos desarrollado en javascript, así como las librerías de bootstrap, jQuery y jQuery ui utilizadas. Podemos encontrar los siguientes directorios:

- jquery-ui-contextmenu: Directorio donde encontramos este plugin particular de jQuery, lo utilizamos para desplegar el menú contextual de una pieza al hacer clic con el botón derecho.
- jquery-ui-1.11.4: Directorio donde encontramos la versión 1.11.4 de jQuery-ui. Como ya se comentó, la diferencia entre jQuery-ui y jQuery, reside en que ésta versión, se centra en las interfaces de usuario.
- jquery-2.1.4: Directorio donde se encuentra el framework de jQuery en su versión 2.1.4.
- bootstrap: Directorio donde se encuentra el framework de bootstrap.
- bootstrap-filestyle: Directorio del plugin de bootstrap para los estilos de botones de selección de ficheros. Lo utilizamos para el botón que permite leer el fichero txt y generar las piezas.

También podemos encontrar los siguientes ficheros javascript:

4.1.10.1. Main.js

Éste es el fichero principal, aquí podemos encontrar las funciones que se emplean para generar las piezas, las peticiones Ajax e incluso las funciones necesarias que hemos desarrollado para arrastrar las piezas html visualmente.

4.1.10.2. readFile.js

En este script, encontramos la función que se encarga de leer un fichero txt y convertir el contenido a un string, el cual podremos procesar y manipular para generar las piezas en html.

4.1.10.3. alerts.js

Este script contiene la función de alertas que se utiliza en todo el sitio, ya sea en los scripts anteriores o las páginas jsp.

4.1.11. Entities

Aquí es donde encontramos las clases de entidad en Java del modelo de negocio. Es decir, aquí encontramos las clases Alumno, Asignatura, Input, Pieza, Problema, Profesor, Solucion y Usuario.

En este lado de la aplicación, las clases no tienen más que atributos, ya que no realizamos ninguna operación con ellos más que manipular sus propiedades.

4.1.12. Servlet

El directorio donde encontraremos todos los servlets que hacen posible el funcionamiento y navegación correcta entre las páginas jsp. Se listan los siguientes:

- CorregirCalificacionServlet
- CorregirServlet

- EditarProblemaServlet
- EliminarProblemaServlet
- ListaProblemasServlet
- LoginServlet
- LogoutServlet
- SolucionServlet

En cada uno de los servlets se realizan de forma genérica las siguientes funciones, evaluación de la sesión. Recogida de los atributos necesarios y llamada a los servicios correspondientes para recoger los datos solicitados por el profesor y mostrar en el siguiente jsp al cual redirigimos, la vista solicitada.

Para más detalle del funcionamiento de cada uno de los servlets, se disponen los diagramas de secuencia en el apartado de Análisis, una vista global de cómo interactúan en el sistema.

4.1.13. service

Este directorio tiene como finalidad listar los clientes java utilizados en nuestra aplicación. Son los cuatro ya mencionados en el diagrama de componentes UML (ilustración 26) en el cual mostramos a qué servicio del servidor mongo se conecta.

Las clases java que encontramos aquí son:

- AsignaturaJerseyClient: Destinado a recuperar las asignaturas pertenecientes a un profesor.
- LoginJerseyClient: Se encarga del proceso de login en el sistema.
- ProblemaJerseyClient: Recupera los listados de problemas asociados a un profesor, así como la inserción de nuevos problemas.
- SolucionJerseyClient: Recupera el listado de soluciones asociado a un problema.

4.2. Funciones Javascript

Vamos a comentar algunos fragmentos importantes de código javascript que se utiliza en la aplicación.

4.2.1. Make_draggable

Esta función se encarga de permitir que el elemento html al que se pasa como argumento, pueda ser arrastrado por la pantalla mediante los eventos de ratón.

Para esta función es necesaria la API de jQuery-ui que importamos en el inicio del html.

```
function make_draggable(element){
    element.draggable({
        helper:'clone'
    });
}
```

Ilustración 37 - funcion make_draggable

4.2.2. buildPieceField

Esta función es de las más importantes, ya que se encarga de generar mediante jQuery, el código html que compone un elemento de una pieza. En base al tipo de elemento que se desee construir, la función devuelve un elemento html predeterminado. Las opciones inicialmente son, un string, un campo para introducir texto o un selector con varias opciones.

```
function buildPieceField(f){

    $.content = "";

    // Comprobamos que tenemos un tipo de pieza para poder identificarlo
    if(f.type === undefined ){
        throw_alert("warning","No se encuentra el tipo de componente en una pieza");
        return ;
    }

    switch(f.type){

        case "label":
            $.content = $('<p/>');
            $.content.text(f.value);
            break;

        case "text":
            $.content = $('<input/>').attr({ type: 'text'});
            $.content.addClass("form-control");
            $.content.addClass("input-text-piece");
            break;

        case "select":
            $.content = $('<select/>');
            f.value.forEach(function(option) {
                $.content.append("<option>" + option + "</option>");
            });
            break;

        default:
            $.content = "";
            break;
    }
    return $.content;
}
```

Ilustración 38 - funcion buildPieceField

4.2.3. leerFichero

Función principal, que recibe un fichero, lo evalúa para comprobar que es un archivo válido y lo procesa en tiempo de ejecución. El objetivo es obtener el contenido almacenado en tipo texto para después convertir su contenido en contenido HTML.

La estructura necesaria para esta aplicación viene detallada en el apartado correspondiente. (Formato del JSON necesario).

```
function leerFichero(evt) {
    var f = evt.target.files[0];

    if (f.type.match('text.*')) {
        var reader = new FileReader();

        reader.onload = (function(theFile) {
            return function(e) {
                var json = e.target.result;

                if( isValidJson(json)) buildPieces(json);   //Construir piezas
                else throw_alert("danger","El fichero <strong>" + f.name + "</strong> introducido no tiene un formato válido.");
            };
        })(f);
        reader.readAsText(f);
    }else throw_alert("danger","El fichero <strong>" + f.name + "</strong> introducido no es un fichero válido");
}

document.getElementById('files').addEventListener('change', leerFichero, false);
```

Ilustración 39 - función leerFichero

4.2.4. getSoulcion

La función que se realiza cuando el profesor ha concluido su labor de plantear la pregunta junto con su solución y con las piezas utilizadas en el ejercicio.

Se encarga de recopilar la solución propuesta por el profesor y de convertirla en una cadena de texto para enviarla al servidor Mongo. El formato utilizado es el JSON, con el convertimos el html a la estructura propuesta y se envía mediante el servicio Restful a la aplicación que controla la persistencia.

```
function getSolucion() {
    var list = $("#sortable").find(".piece ");
    var piezas = "[";

    if (list != null) {
        for (var i = 0, len = list.length; i < len; i++) {
            piezas += "{\"inputs\":[";

            for (var r = 0, tam = list[i].children.length; r < tam; r++) {

                if (list[i].children[r].nodeName == "P") {
                    piezas += "{\"type\":\"label\",\"value\":\"" + list[i].children[r].innerHTML + "\"}";
                } else if (list[i].children[r].nodeName == "INPUT") {
                    piezas += "{\"type\":\"text\",\"value\": \"" + list[i].children[r].value + "\"}";
                } else if (list[i].children[r].nodeName == "SELECT") {
                    piezas += "{\"type\":\"select\",\"value\":\"" + list[i].children[r].value + "\"}";
                }
                if (r + 1 < tam) piezas += ",";
            }
            piezas += "]}";
            if (i + 1 < len) piezas += ",";

        }
    }
   piezas = piezas + "]";
    return piezas;
}
```

Ilustración 40 - función getSolucion

4.2.5. #Content-panel

Con el fragmento anterior, hacemos que el div con id “content-panel” permita que se dejen caer elementos html cuyas clases sean “dragIn” y “dragOut”. También se encarga de realizar la función anónima cuando sobre él se realiza un evento de tipo drop, es decir, cuando se deja algún elemento html caer sobre él. En este caso, clonamos el elemento y le hacemos las manipulaciones necesarias para duplicar el elemento y añadirlo en la página html.

Éste código es el que nos permite que una pieza se pueda arrastrar hasta el panel donde se construye la solución.

```
$("#content-panel").droppable({
    accept: ".dragIn, .dragOut",
    drop: function(ev, ui) {
        if (!ui.draggable.hasClass("dragOut")) {
            var droppedItem = $(ui.draggable).clone();
            droppedItem.addClass("item-" + counts[0]);
            droppedItem.addClass("dragOut");
            droppedItem.removeClass("dragIn");
            $("#sortable").append(droppedItem);
        }
    }
});
```

Ilustración 41 - Código jQuery sobre el panel de construcción de soluciones

4.2.6. #finalizar

Una vez que el profesor ha terminado de escribir el enunciado y elaborar la respuesta "ideal", el profesor hará click en el boton de finalizar. Es en este momento cuando este fragmento de código se ejecuta, recopilando el enunciado y la solución convirtiéndolo en un JSON para enviarlo a la aplicación de Mongo.

```
$("#finalizar").click(function(ev, ui) {

    var json;
    var enunciado = getEnunciado(); //Obtenemos el enunciado
    var solucion = getSolucion();   //Obtemenos la solución planteada

    /* @todo - Control de errores */
    json = '{\"enunciado\":\"' + enunciado + '", \"piezas\":' + jsonPiezas + ", \"solucion\":" + solucion + "}";
    //var ob = JSON.parse(json);
    console.log(json);

    $.ajax({
        type: 'POST',
        url: "http://localhost:8080/ServidorMongo/API/Problema/insertarProblema",
        data: json,
        contentType: "application/json",
        dataType: 'jsonp',
        success: function(data, textStatus, jqXHR){
            console.log(data);
        }
    });

});
```

Ilustración 42 - Código jQuery lanzado para insertar nuevo problema

5. Pruebas

Para comprobar que el producto cumple las funcionalidades solicitadas, hemos realizado algunas pruebas básicas. Por motivos de tiempo, no se ha podido realizar todas las pruebas necesarias para asegurar la estabilidad del producto y depurar todos aquellos fallos que puedan derivar en un comportamiento indebido del software.

Se han realizado pruebas en el inicio de sesión y el cierre de la misma, evaluando los siguientes casos:

- Usuario y contraseñas correctos.
- Usuario y contraseñas incorrectos.
- Usuario y contraseñas en blanco.
- Usuario correcto y contraseña incorrecta.
- Usuario incorrecto y contraseña correcta.

Se han realizado algunas pruebas de navegación, es decir, se ha intentado entrar sin una sesión válida previamente iniciada, dando lugar al comportamiento esperado, es decir, redireccionando la aplicación al inicio de sesión, prohibiendo así el uso desautorizado de cualquier usuario.

También se han realizado pruebas de unidad en la manipulación de los formularios en jQuery, evaluando que se tienen todos los valores necesarios, y válidos para la inserción o edición del problema.

Es importante también destacar que se han realizado pruebas de integración sobre los clientes y servicios para asegurar que la comunicación entre ambos era correcta y funcionando como se esperaba.

Por último, hay que mencionar que se han realizado pruebas sobre todos los casos de uso establecidos en la documentación, siendo el caso ideal la primera guía para asegurar la funcionalidad solicitada.

6. Conclusiones y trabajos futuros

6.1. Objetivos cumplidos

Podemos estar contentos, ya que todas las funcionalidades negociadas inicialmente se han cumplido. Cada uno de los requisitos funcionales establecidos se han implementado satisfactoriamente.

Obviamente, se trata de una aplicación web que siempre puede ser actualizada, mejorada y optimizada. Dado el plazo de tiempo invertido, las funciones están operativas con un interfaz adaptativo (responsive).

Se han utilizado herramientas actuales, tanto en temas de diseño (Bootstrap) como en APIs de desarrollo web (jQuery o los servicios Restful). De este modo, se ha permitido trabajar más eficientemente, pero a cambio se ha tenido que invertir cierto tiempo en la pequeña curva de aprendizaje que ha supuesto.

6.2. Dificultades encontradas

Como se ha mencionado en el punto anterior, se ha tenido que invertir una pequeña porción del tiempo de desarrollo a investigar las tecnologías utilizadas y nuevas para el propio equipo.

No es una dificultad como tal, pero ha supuesto trabajar con algo completamente nuevo, probando cada ejemplo hasta entender cómo funcionaba para llevar a cabo nuestro objetivo concreto.

Una de las partes claramente dificultosas, puede decirse que ha sido los pequeños scripts en jQuery utilizados para el movimiento de piezas, ya que es algo nuevo que he utilizado, al igual que Bootstrap, API que no he utilizado nunca pero que me ha servido para entender cómo funciona y hacer la aplicación de forma más rápida que si la hiciera completamente desde cero.

Aunque el concepto de mover piezas visualmente parece algo sencillo, no lo resulta en un primer paso ya que lo que se hace en realidad es cambiar el código html en tiempo de ejecución por bloques de código. Un concepto un poco chocante pero que si se realiza un buen código no tiene por qué dar mucho problema.

Podría decirse que también algún que otro diagrama UML ha presentado dificultad, como por ejemplo el diagrama de secuencia que hace referencia al procedimiento de crear un problema. La dificultad viene dada por el propio proceso que se utiliza para insertarlo en el servidor, en lugar de utilizar los servlets, se utiliza directamente una petición AJAX, descuadrando lógicamente el esquema.

Por último indicar que la última dificultad ha residido en las pruebas, ya que se ha tenido que seleccionar bien las pruebas que realizar. El tiempo es un recurso muy limitado y no ha dado tiempo a realizar todo el control de errores que se deseaba.

6.3. Posibles ampliaciones

Un aspecto del que podemos presumir es de las tantas ampliaciones que puede tener la aplicación en un futuro:

- Zona de administración.
  Actualmente la aplicación tiene pensada una zona de trabajo para el profesor, otra para el alumno y el algoritmo de corrección junto con los servicios.

  Sin embargo, no se ha elaborado una zona de administración interna para dar de alta, modificar o eliminar a los alumnos ni profesores que utilizarán nuestra aplicación. Se podría elaborar un pequeño backend que proporcione dichas funcionalidades.

- Añadir funcionalidad para todo tipo de dispositivos.
  Actualmente se utiliza un interfaz gráfico responsive, permitiendo que tenga una visualización clara y agradable hasta para los dispositivos móviles. Sin embargo, pierde la funcionalidad más característica de nuestra aplicación.

Las librerías javascript utilizadas para arrastrar las piezas, actualmente sólo funcionan para eventos de ratón. Habría que buscar e implementar una librería que trabaje con los eventos táctiles de los dispositivos.

- Integración con otros sistemas actuales
  Se podría elaborar algún componente, plugin o incluso versión del software que pueda integrarse con plataformas educativas como Moodle. Reflejando los resultados en pequeños expedientes o archivos que podrían enviarse por correo al profesorado y que sirvan para evaluar al alumnado.

- Listado de ejercicios o enlazado de problemas
  Una curiosa ampliación que podría desembocar en los que se conocen como exámenes o cuestionarios de evaluación. Se podrían enlazar varios problemas dando lugar a un examen.

Referencias

https://netbeans.org/
https://jquery.com/
https://jqueryui.com/
http://getbootstrap.com/
http://jsonlint.com/
https://scratch.mit.edu/
http://es.gizmodo.com/como-organizar-toda-tu-vida-utilizando-trello-1684529913
http://www.mongodb.org/
http://es.wikipedia.org/wiki/Scrum
Leonard Richardson (2007). *RESTful Web Services*. O'Reilly Media.
Kristina Chodorow (2013). *MongoDB: The Definitive Guide*. O'Reilly Media. 2nd edición.

ANEXO I – Diagramas de secuencia

Se han agregado en un anexo para que se puedan apreciar a mejor tamaño los diagramas de secuencia mostrados en las ilustraciones del apartado 3.2 Diagramas de casos de uso.

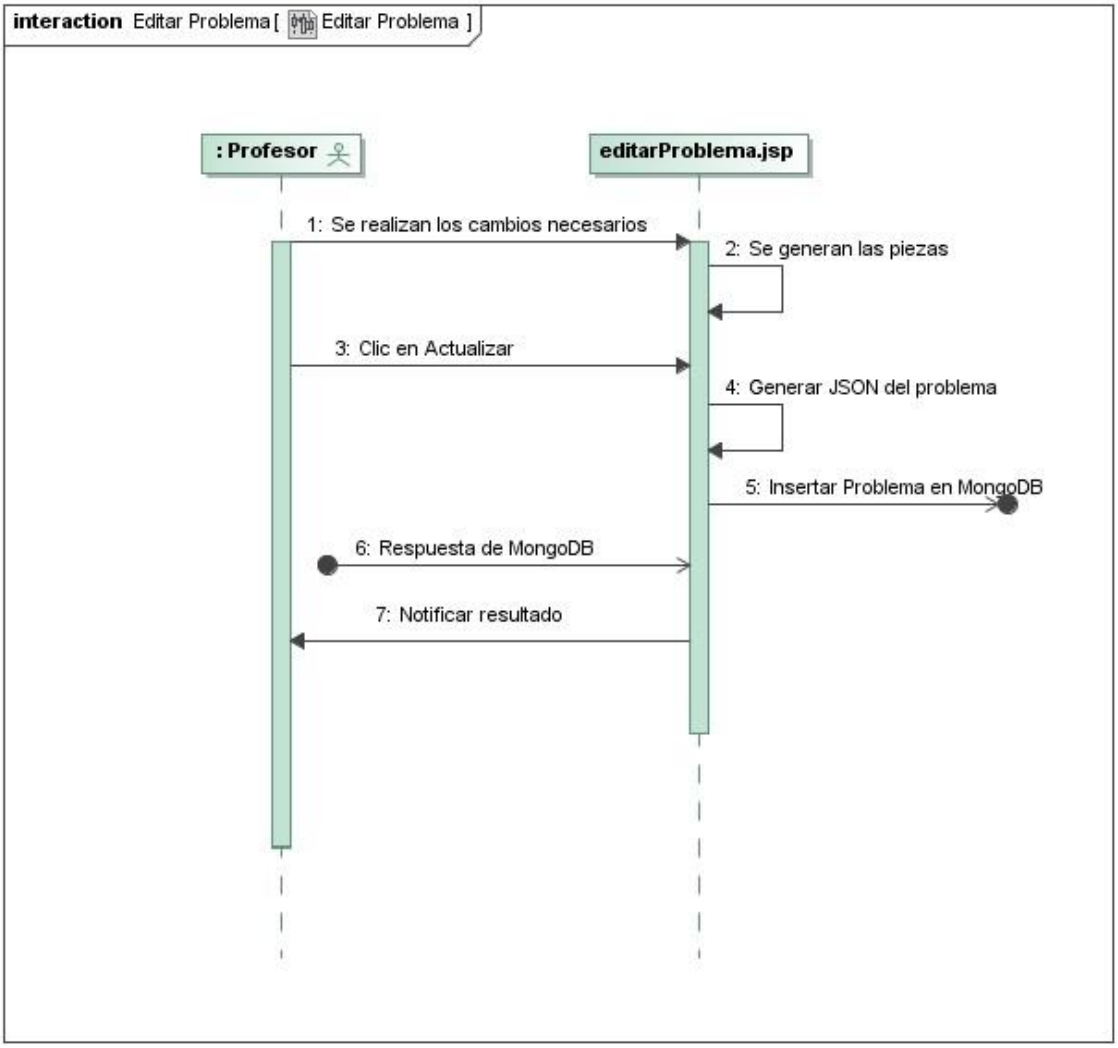


Ilustración 43 - Editar problema (Diagrama de secuencia)

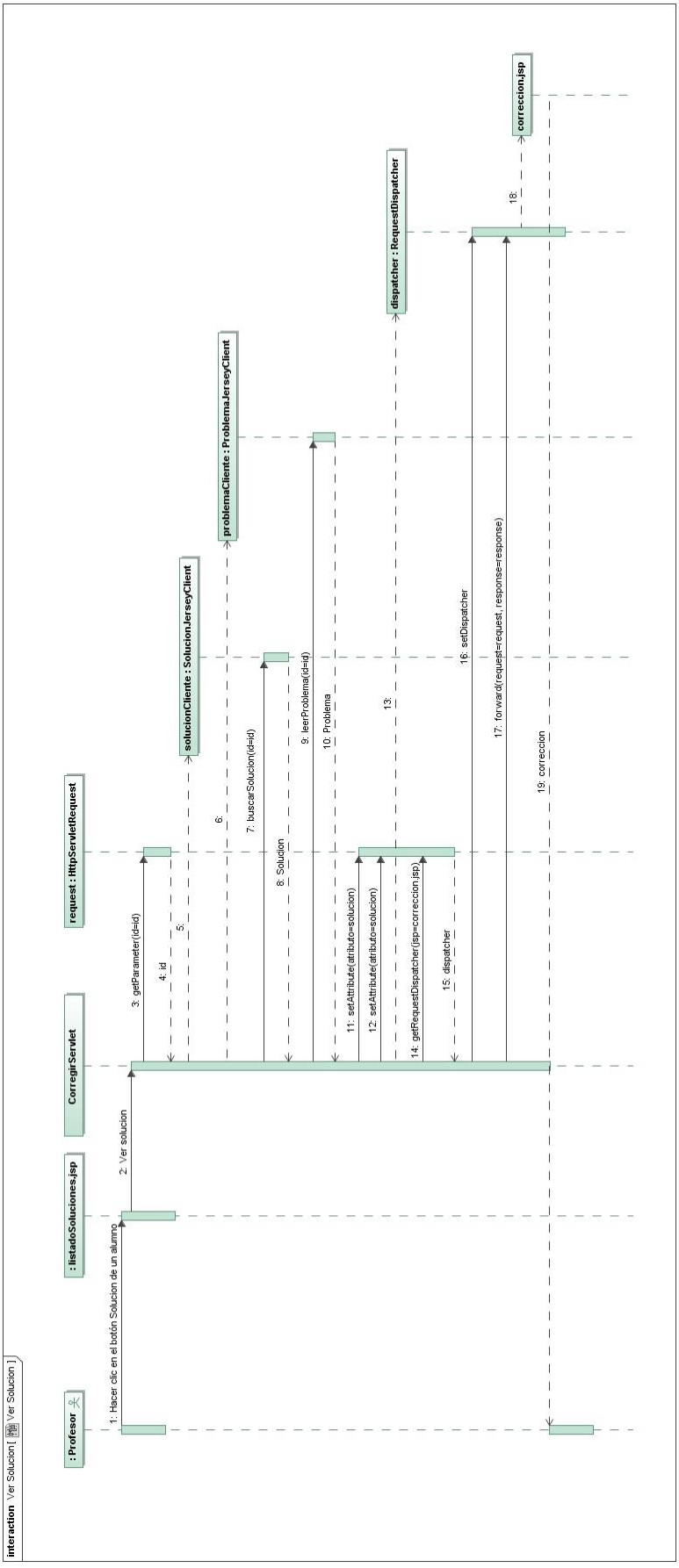


Ilustración 44 - Ver solución (Diagrama de secuencia)

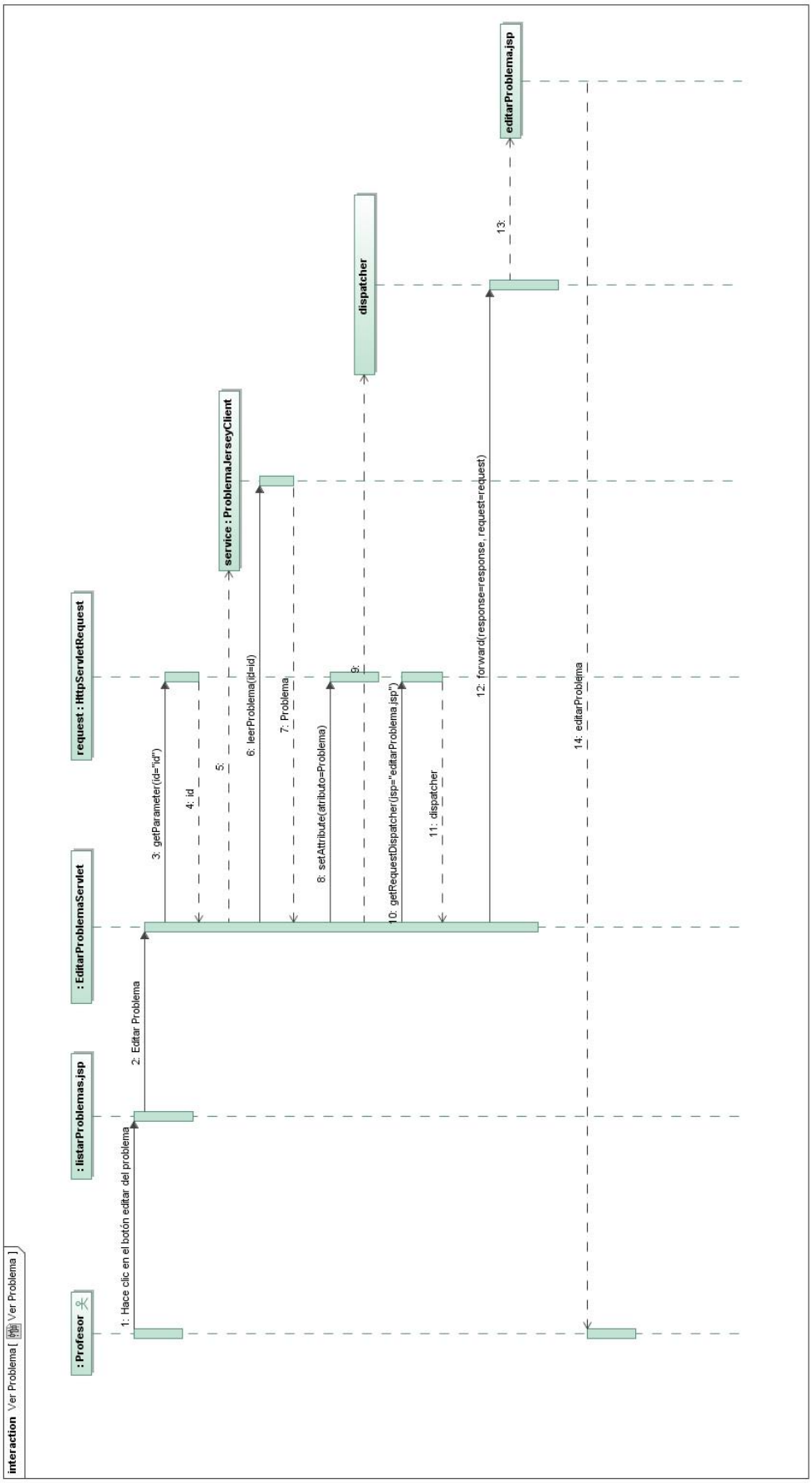


Ilustración 45 - Ver problema (Diagrama de secuencia)

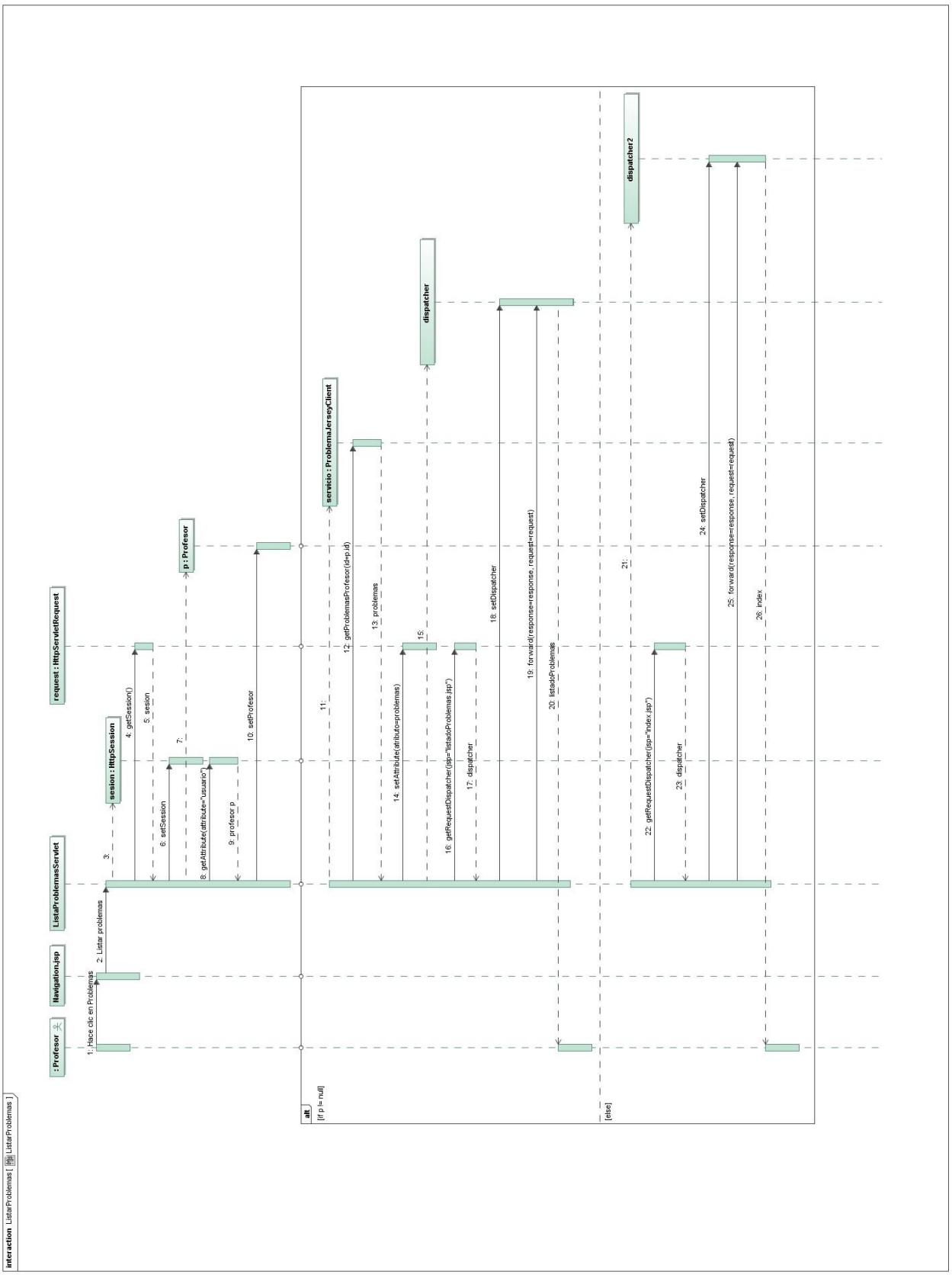


Ilustración 46 - Listar problemas (Diagrama de secuencia)

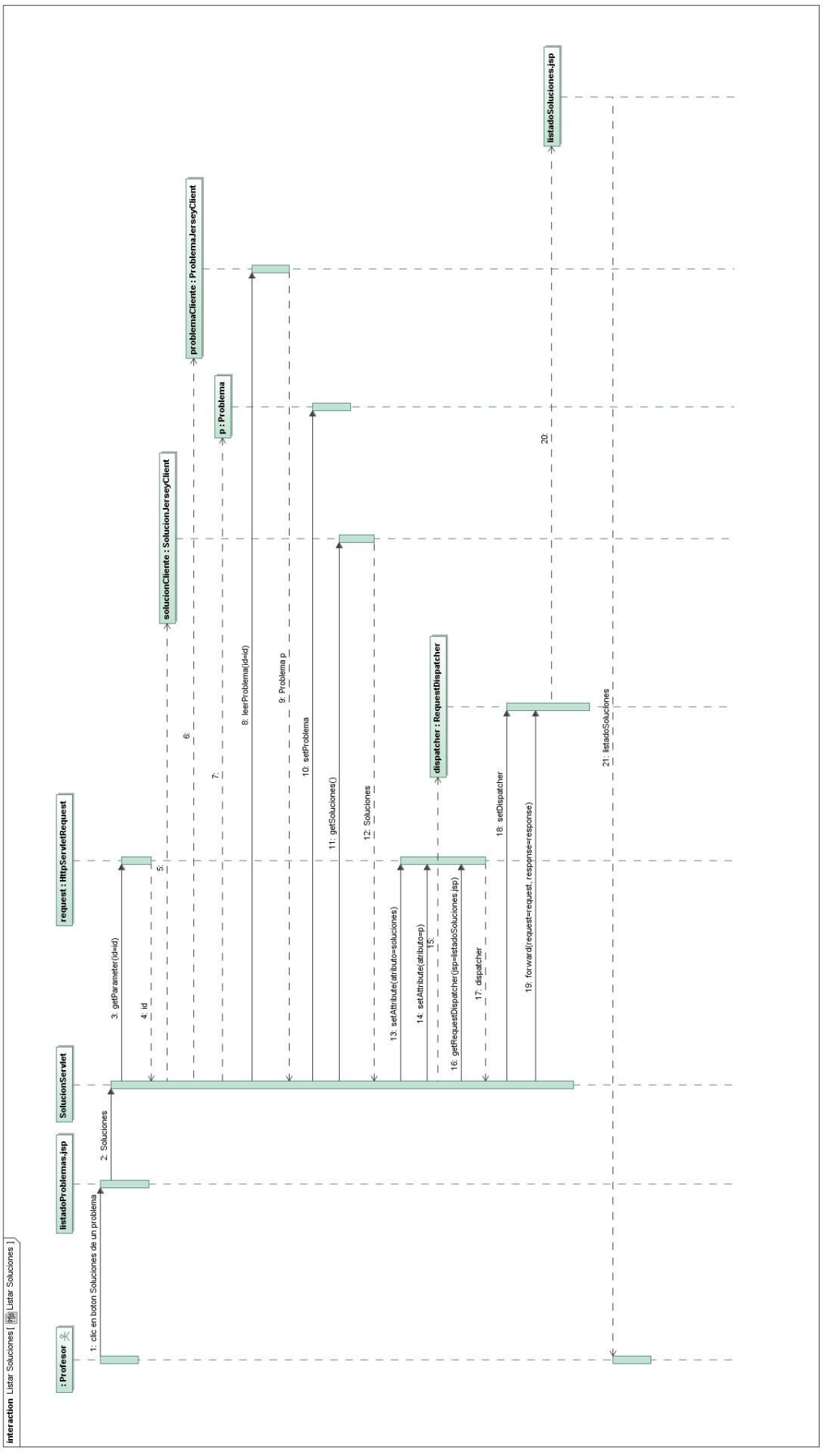


Ilustración 47 - Listar soluciones (Diagrama de secuencia)

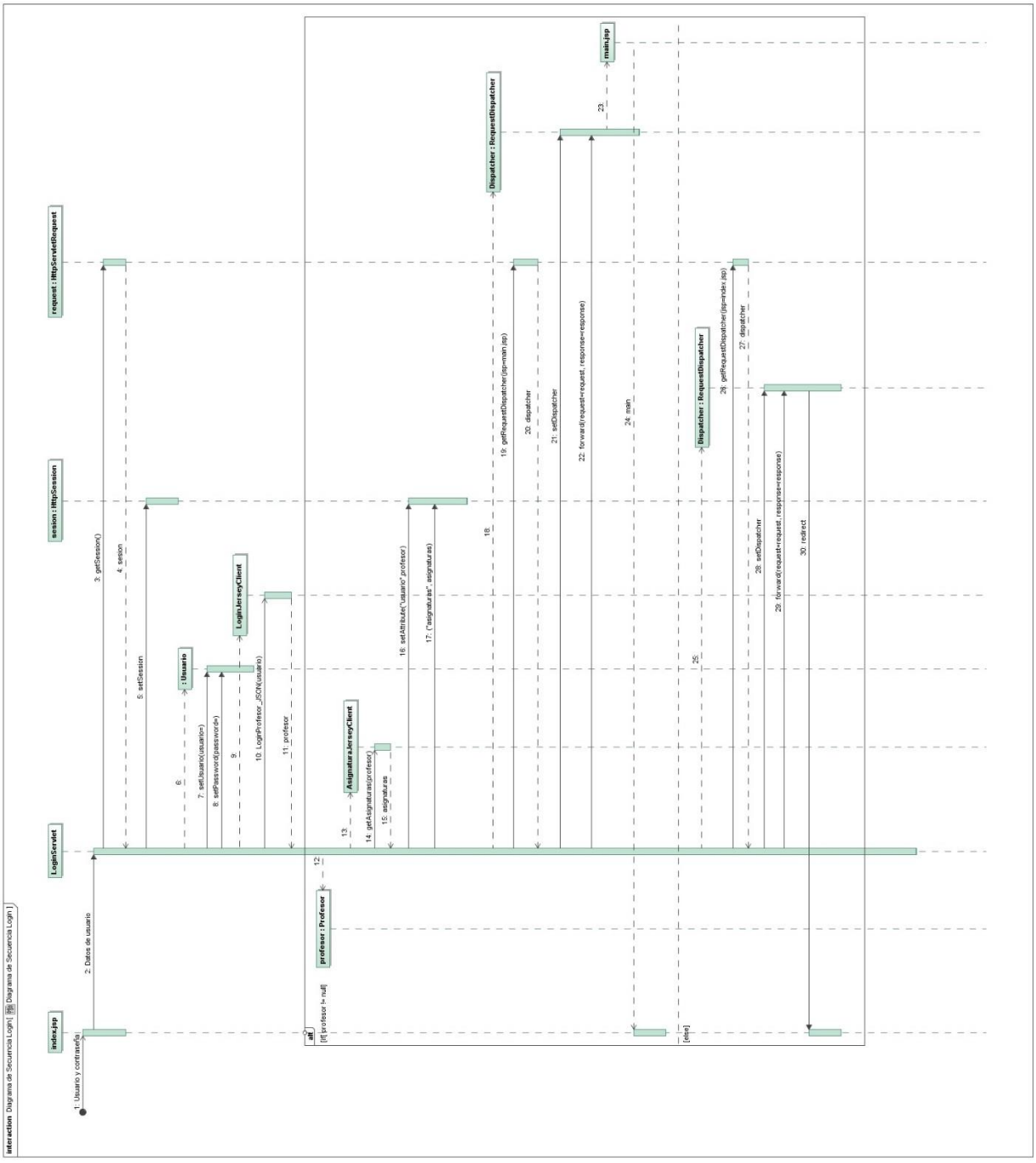


Ilustración 48 - Iniciar sesión (Diagrama de secuencia)

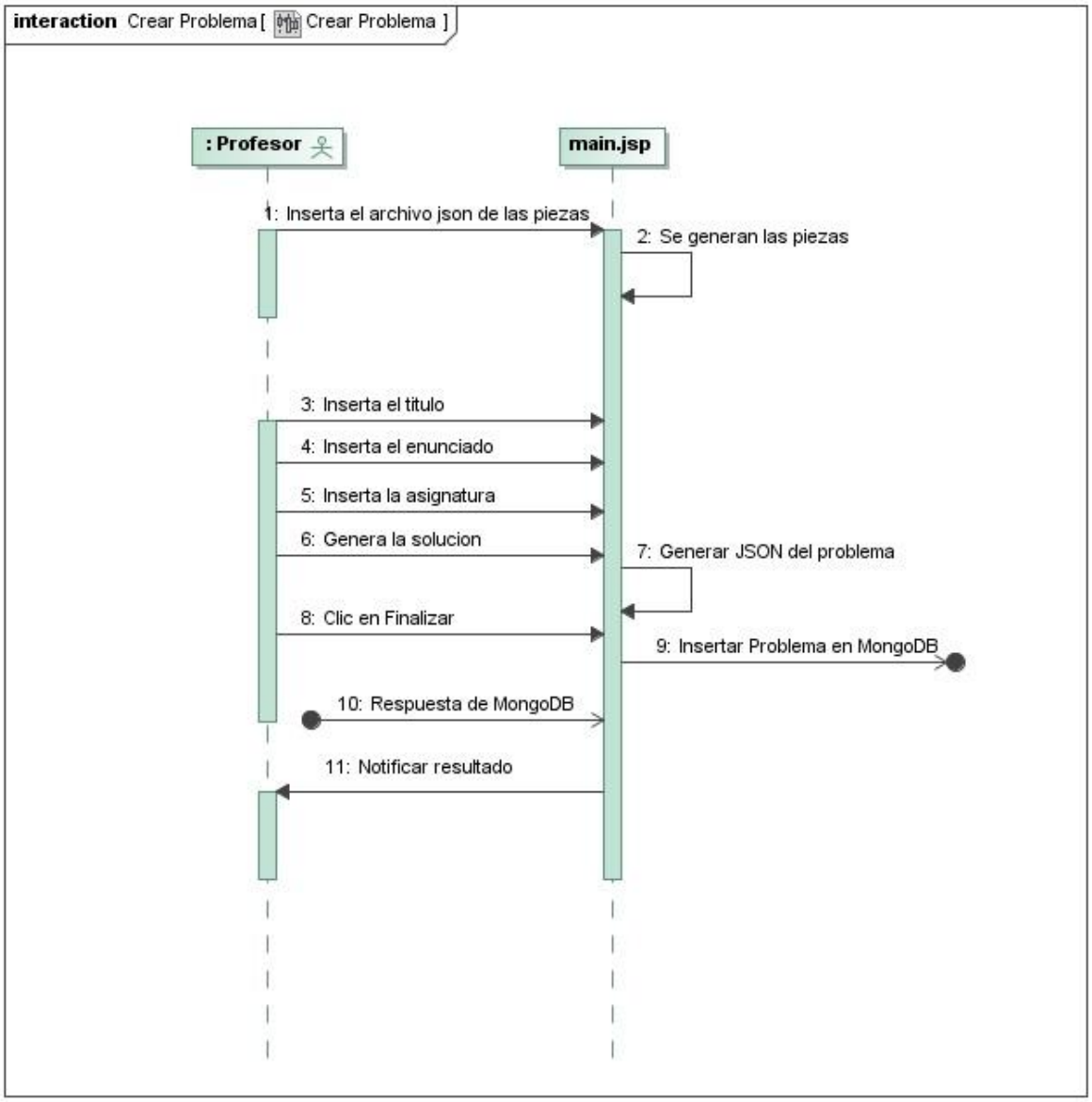


Ilustración 49 - Crear problema (Diagrama de secuencia)

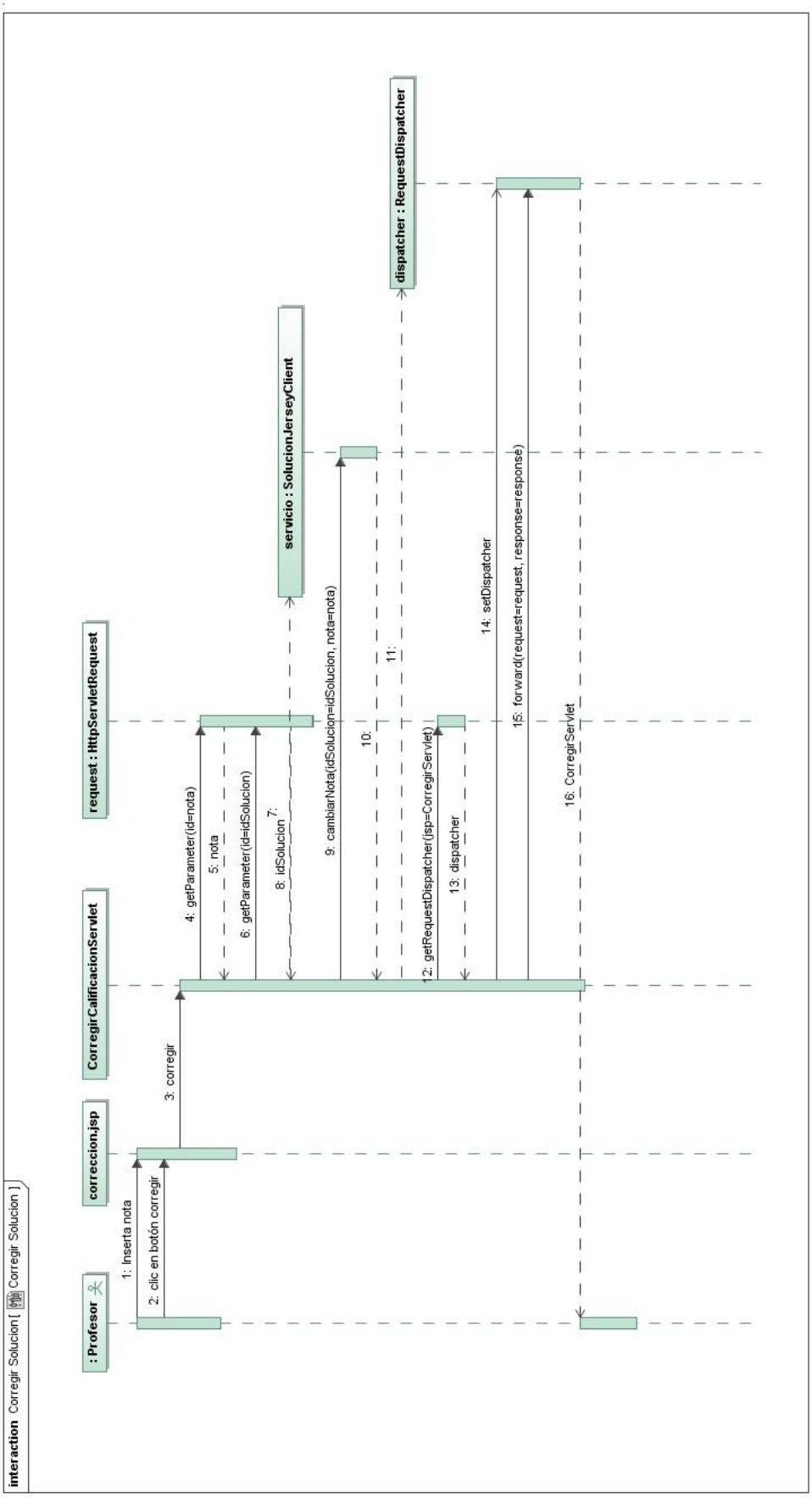


Ilustración 50 - Corregir solución (Diagrama de secuencia)

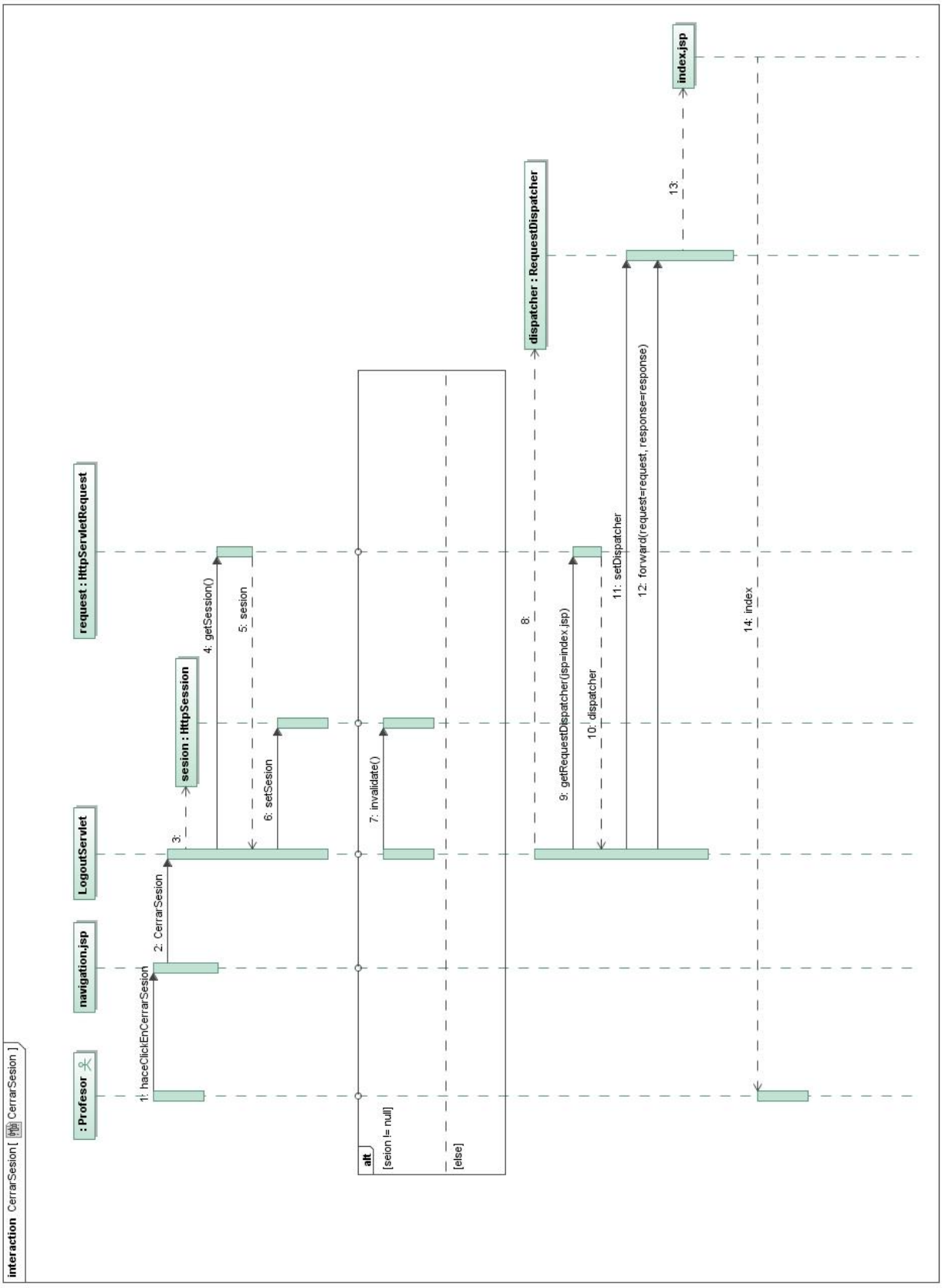


Ilustración 51 - Cerrar sesión (Diagrama de secuencia)

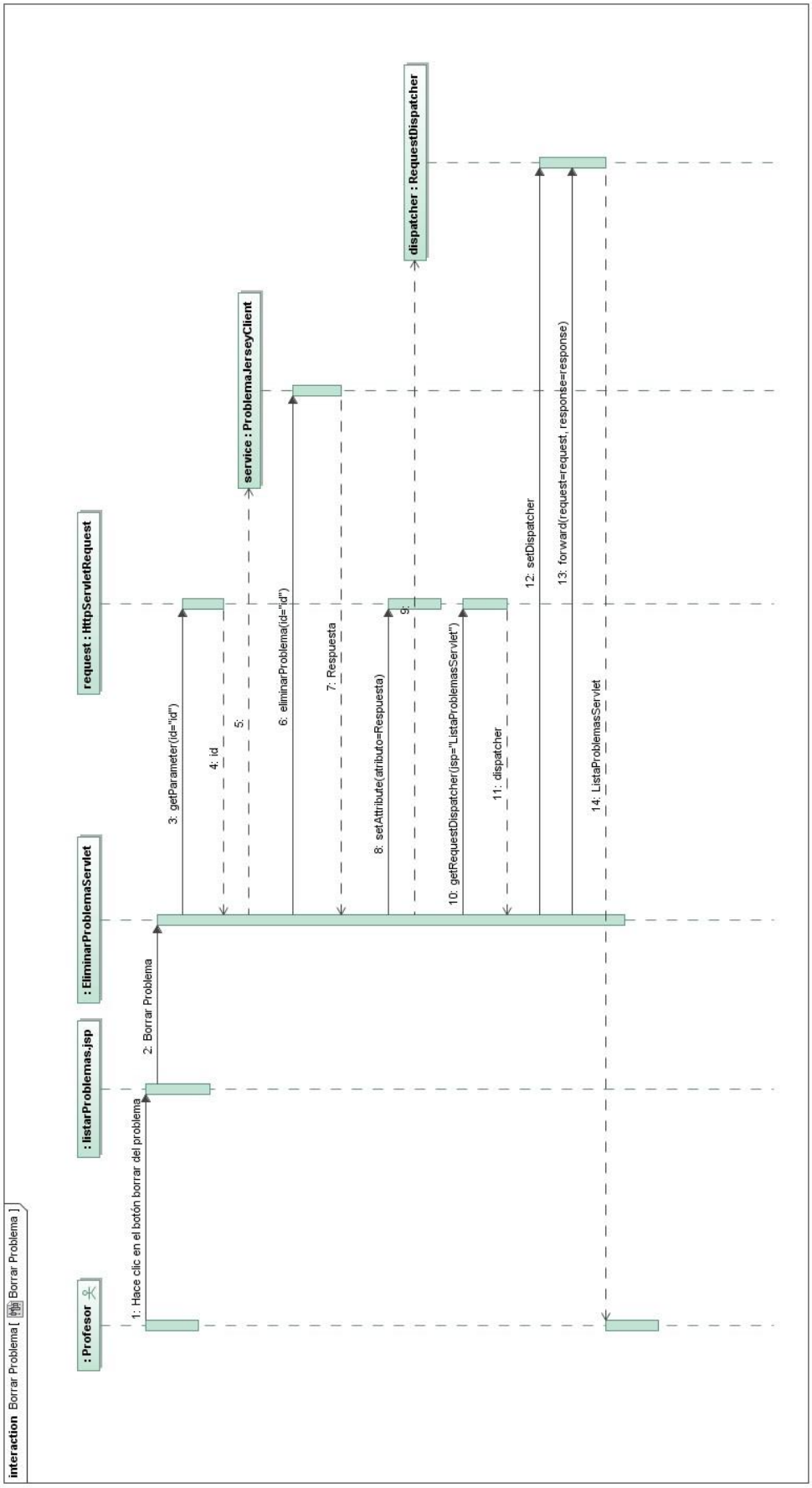


Ilustración 52 - Borrar problema (Diagrama de secuencia)