Universidad de Valladolid

E.T.S Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería de Informática

Aplicación web de ayuda a la mejora de la pronunciación del español como lengua extranjera

Autor:
David Soler Herrero

Tutor:
Valentín Cardeñoso Payo

Cotutor:
César González Ferreras

Agradecimientos:

A mi familia, por su dedicación y lucha para que tuviera un futuro.

A mi tutor, por su compromiso y confianza en todo este tiempo.

A mi cotutor, por la ayuda y apoyo proporcionado.

RESUMEN

El objetivo de este trabajo de fin de grado es la creación de una aplicación web que ofrezca las suficientes herramientas para realizar – por parte de los estudiantes – diversos ejercicios con el objetivo de mejorar la pronunciación del español como lengua extranjera. También debe facilitar a los investigadores y encargados la manipulación y gestión de las locuciones obtenidas así como la creación de nuevos ejercicios.

Se trata de una iniciativa del grupo de investigación de *Entornos de Computación Avanzada y Sistemas de Interacción Multimodal* (ECA-SIMM) del Departamento de Informática de la Universidad de Valladolid. Surge ante la necesidad de disponer de un sistema informático moderno y multiplataforma que pueda mitigar y facilitar las tareas propias de la gestión de corpus lingüísticos.

El alcance de la construcción de software fue limitado a la generación de dos artefactos claramente diferenciados:

Una aplicación web que proporcione un entorno gráfico y que permita realizar las diversas operaciones disponibles. Dicha aplicación está completamente desarrollada en HTML5 heredando así las ventajas que este nuevo paradigma ofrece.

Un servidor web que proporcione una API con la que brindar al cliente los servicios que solicite. Su cometido principal es realizar operaciones que demanden un cierto poder de cálculo y ofrecer un sistema de persistencia centralizado y genuino.

ABSTRACT

The objective of this final degree work is the creation of a web application that provides the necessary tools in order to make different exercises – by students – with the aim of improving their pronunciation of Spanish as foreign language. It should also provides to researchers and managers an easier way to handler and manage the obtained locutions as well as the creation of new exercises.

It is an initiative of the research group of Advanced Computing Environments and Multimodal Interaction Systems (ECA-SIMM) of Department of Informatics of University of Valladolid. The need arises to have a modern multi-platform system in order to reduce and facilitate the own linguistic corpus management tasks.

The scope of the software building was restricted to the generation of two different applications:

A web application that provides a graphical interface and allow for making the available operations. This application is developed entirely with HTML5 technology, which provides the advantages of this paradigm.

A web server with an specified API for the purpose of affording to customers the services that request. The main purpose is to perform operations that demand a certain computing power and provide a centralized and genuine system of persistence.

Índice

Lista de figuras

Lista de tablas

Capítulo 1

INTRODUCCIÓN

1.1. Visión general

Aunque hace tiempo que el ser humano se ha beneficiado de la tecnología, no ha sido hasta finales del anterior siglo y comienzo del actual cuando su desarrollo ha sufrido una de las mayores aceleraciones. Tal es así que la tecnología nos rodea completamente y nos ha llevado a un estado en el cual nuestros problemas y soluciones provienen principalmente de su uso. La sociedad occidental apenas ha salido de la revolución digital y ha entrado violentamente en la era de la información y las telecomunicaciones, cambio principalmente motivado por uno de los mayores hitos de la historia, Internet.

El concepto de información prácticamente ha calado en todos los ámbitos y áreas del conocimiento así como en la psique humana. La información es ahora considerada un valioso recurso, que debe ser correctamente tratado y almacenado para obtener el mayor beneficio de ella. Lo que provoca, a su vez, que muchas necesidades actuales de las sociedades *occidentales* estén relacionadas directa o indirectamente con el manejo de la información.

La aparición de los ordenadores y sobretodo de los *smartphones* ha hecho que lo virtual se entremezcle con lo tangible, que lo lejano esté próximo y que incluso dichos dispositivos se conviertan casi en extensiones de nuestra mente para interactuar con el mundo.

El desarrollo actual junto con el sistema económico imperante ha generado la globalización, lo que motiva un mayor flujo de masas de población a otros países extranjeros. Es por ello que el conocimiento de lenguas mayoritarias se ha vuelto un requisito prácticamente esencial.

Aunque la lengua inglesa sea el idioma internacional de facto, la lengua española está experimentando una de los mayores tasas de crecimiento y sus previsiones de futuro indican que podría llegar a desbancar al inglés como primera lengua. Esto está motivando a que varios países estén replanteando la idea de añadir el español al sistema educativo estatal lo que conllevaría a un incremento mayor.

Sin embargo, el aprendizaje de una nueva lengua no nativa es un proceso en muchas ocasiones complejo y arduo. Es por ello que actualmente se están desarrollando sistemas educativos basados en las incipientes tecnologías para mejorar el proceso de aprendizaje y así, reducir el tiempo necesario para que una persona no nativa pueda alcanzar un nivel del lenguaje extranjero aceptable.

Para conseguir esto es necesario no sólo centrarse en los aspectos más comunes a la hora de aprender un idioma – gramática, vocabulario… – sino también en su entonación y prosodia. Esto es muy complicado de mejorar sino se tiene un contacto real y continuo con personas de esa misma habla. Es por ello que se necesita diseñar sistemas capaces de ayudar a los estudiantes de lenguas extranjeras y como se ha mencionado anteriormente, la tecnología puede facilitar con creces dicho objetivo.

1.2. Objetivos

El objetivo principal es desarrollar un entorno que brinde las suficientes capacidades para poder desarrollar herramientas relacionadas con la gestión de corpus y toma de muestras.

El segundo objetivo es conseguir el desarrollo de una aplicación web que ayude a personas extranjeras a mejorar su pronunciación de la lengua española. Dicho aprendizaje se realiza mediante la realización de diversos ejercicios definidos en los corpus y en los cuales se captura la pronunciación de diversos fragmentos lingüísticos por parte de los estudiantes. Posteriormente, se evaluarían empleando otros programas externos de análisis (de pronunciación, prosodia...) y se detallarían los aspectos de mejora que debe realizar el estudiante si quiere mejorar su pronunciación de la lengua española.

El último objetivo es proporcionar las herramientas suficientes para la gestión y acceso a los corpus especificados en el sistema, así como permitir gestionar y acceder a las muestras donadas u obtenidas – tras la toma de muestras por parte de donantes o estudiantes – a investigadores y gestores.

El alcance de este proyecto está en la especificación y confección de todo el entorno que sea capaz de proporcionar las capacidades y características necesarias para poder desarrollar las aplicaciones descritas anteriormente. También el desarrollo de una aplicación web que permita la creación, edición y gestión de corpus en el sistema y finalmente otra aplicación web que permita realizar los ejercicios definidos en los corpus y capturar y almacenar las locuciones pertinentes.

1.3. Motivación personal

La principal motivación personal del alumno estuvo relacionada con la descripción del trabajo fin de grado en cuestión. Consideró que era una buena posibilidad para expandir libremente sus conocimientos y experimentar con la tecnología actual; especialmente en el ámbito de las tecnologías de la Web, las cuales están marcando un gran crecimiento y relevancia a nivel mundial.

Por ello, cualquier tipo de inversión de tiempo y esfuerzo en la adquisición de conocimientos en éste área, no hacía más que ayudar y fomentar a sobrepasar las competencias típicas de un graduado de esta disciplina y tener más posibilidades en el mundo laboral y académico.

Otra motivación importante fue la posibilidad de confeccionar un sistema real destinado a la ayuda de la investigación dentro del ámbito universitario. Así como proveer una aplicación para la mejora de ciertas capacidades de una parte de la sociedad.

Finalmente y no menos importante, tener la posibilidad de desarrollar un proyecto de cierta complejidad y aprender a gestionarlo correctamente tras su ejecución. La finalidad última para el alumno en este aspecto es emplearlo como medio para evidenciar sus fortalezas y debilidades y así ser consciente de ellas y actuar en consecuencia.

1.4. Estructura y contenido del documento

A continuación, se va a describir a grandes rasgos la estructura y contenido del presente documento. Consta de diez capítulos en los que se detalla todo el proceso relacionado con el trabajo de fin de grado realizado por el alumno. Además de la estructura principal, se han adjuntado diversos anexos para complementar la información disponible.

- Capítulo 1, Introducción: cuya finalidad es proporcionar una visión general del proyecto realizado; además de exponer el contexto actual en el que se realiza así como las motivaciones personales del alumno.
- Capítulo 2, Conceptos y contexto tecnológico: comprende la explicación del término corpus y un análisis del contexto tecnológico actual.
- Capítulo 3, Plan de desarrollo: se explica la metodología y planificación seguida a lo largo del desarrollo del proyecto. También se realiza un análisis del seguimiento del proyecto una vez ha sido ejecutado.
- Capítulo 4, Análisis: relacionado con la fase de análisis del desarrollo de software. Se realiza un

amplio estudio del dominio del problema a resolver. Va desde la obtención y elicitación de requisitos hasta la primera aproximación, mediante diagramas de secuencia, de los casos de uso propios del sistema.

- Capítulo 5, Diseño: corresponde con la siguiente fase del desarrollo de software. En él se explican todas aquellas decisiones tomadas para adaptar el proceso previo de análisis a las tecnologías elegidas para el desarrollo del sistema.
- Capítulo 6, Implementación: relacionado con la codificación de las especificaciones de la fase de diseño. Se describen las distintas complicaciones surgidas durante dicho proceso además de nuevas funcionalidades añadidas que no fueron especificadas previamente.
- Capítulo 7, Depuración y pruebas: comprende todas comprobaciones y procedimientos realizados para poder determinar la robustez y calidad del software producido en la fase de implementación.
- Capítulo 8, Conclusiones: reúne todas las observaciones realizadas por el alumno sobre el proyecto en su totalidad una vez concluido.
- Capítulo 9, Trabajo futuro: se especifican las líneas para desarrollos posteriores con el objetivo de añadir nuevas funcionalidades no implementadas o mejorar la calidad del producto generado.
- Capítulo 10, Bibliografía: se enumeran las fuentes de información en las cuales se basó el alumno para la realización del proyecto.

Además de la presente memoria, se añadieron tres anexos:

- Anexo A, Manual de Instalación: guía para el proceso de instalación del software desarrollado.
- Anexo B, Manual de Uso: breve manual para el manejo del sistema por parte del usuario final.
- Anexo C, Contenido del CD-ROM: se enumeran todos los recursos que se pueden encontrar en este tipo de dispositivo de almacenamiento.

Capitulo 2

CONCEPTOS Y CONTEXTO TECNOLÓGICO

2.1. Introducción

En el presente capítulo se explica el concepto de corpus y se continúa con un breve análisis del contexto tecnológico existente actualmente.

2.2. Corpus

A continuación se explicará el concepto de Corpus, un término que es pilar fundamental en este trabajo de fin de grado.

Atendiendo a la definición más general del mismo, un corpus (en plural córpora) es un conjunto de datos, textos u otros materiales sobre una determinada materia, y que sirven de base para una investigación o trabajo. Sin embargo, el tipo de corpus que nos acontece es el denominado corpus lingüístico.

Un corpus lingüístico, por tanto, es una colección estructurada de ejemplos o muestras reales del uso de una o varias lenguas. Normalmente su extensión es muy grande y las muestras pueden ser textuales (fragmentos de textos) y/o orales (locuciones). Su tamaño se contabiliza dependiendo del tipo de las muestras, si la muestra es textual entonces se cuenta el número de palabras que la compone, en cambio, si es oral entonces se cuenta los minutos de la locución.

Los córpora pueden clasificarse en cerrados o abiertos. Si es un corpus cerrado significa que la colección de muestras es siempre la misma independientemente del tiempo. En cambio, si es un corpus abierto, el tamaño y la composición de la colección va variando a lo largo del tiempo mediante nuevas incorporaciones o modificaciones. Este último tipo permite adaptarse mejor a las variaciones del léxico de la lengua de estudio.

En los últimos años se está empleando activamente medios tecnológicos e informáticos tanto para la gestión como recogida de muestras. Esto ha motivado el desarrollo de una nueva disciplina denominada *Ingeniería lingüística* que, a su vez está compuesta por otros campos interdisciplinarios, como por ejemplo, la *Lingüística computacional*, en la cual participan lingüistas, expertos en lógica e informáticos, normalmente especializados en inteligencia artificial.

2.3. Contexto tecnológico

Es innegable que actualmente la web está sufriendo una de sus mayores revoluciones con la aparición de HTML5 (el cual no es más que las últimas versiones del stack HTML/CSS y JavaScript); esto se debe a que no solo han aportado una mejora en sus funcionalidades sino que han modificado sensiblemente la forma de desarrollo y manejo de la información. Lo que hace que se esté perfilando como el nuevo entorno de programación a nivel mundial para el desarrollo de aplicaciones distribuidas y multiplataforma. Si en los años anteriores fue Java la que abanderaba el desarrollo software ahora comienza a ser eclipsado por esta reinvención.

El éxito de Java se debe a que permite crear un único código y pueda ser compilado independientemente de la arquitectura del sistema anfitrión. La forma de conseguir esta ventaja es

gracias a la máquina virtual que proporciona y que emula una máquina común para todas las arquitecturas soportadas. Aunque tanto el lenguaje y la máquina son desarrollos maduros y poseen grandes mejoras de eficiencia, su rendimiento es menor que si el programa estuviera compilado directamente a código máquina. Además, requiere que se instale un conjunto de componentes para poder ser ejecutado.

La Web, por otra parte, ha ido evolucionando adecuadamente a lo largo del tiempo. Adaptadose poco a poco a las nuevas necesidades que iban surgiendo. Sin embargo, la aparición de HTML5 ha transformado completamente la mentalidad que se tenía de ella. Se demuestra por tanto que aunque la web en su comienzos era muy sencilla y limitada, las ideas y planteamientos que la fundamentaban tienen un gran potencial que en el día de hoy estamos aprendiéndo a explotar mejor.

La web es distribuida, amplia e interconectada. Gracias a su sistema de identificación de recursos URI se puede localizar unívocamente un recurso concreto dentro de la descomunal red de redes. Además no requiere estrictamente de ningún componente adicional, aunque normalmente se usa un navegador web para poder visualizar los contenidos que ésta puede ofrecer.

Por este conjunto de razones, se decidió desarrollar una aplicación web que explotara las ventajas que la nueva web ofrece y así conseguir un producto innovador y versátil.

Aunque se puede desarrollar sin ningún tipo de framework, la experiencia demuestra sus beneficios ya que – además de contar con un soporte y robustez comprobada – se evita el problema de “reinventar la rueda”.

Capítulo 3

PLAN DE DESARROLLO

3.1. Introducción

En el presente capítulo se detallará y explicará la gestión del proyecto y las justificaciones ante diversas cuestiones relacionadas con dicha gestión. Además se explicará la metodología seguida a lo largo del desarrollo de la aplicación así como el software empleado.

3.2. Modelo de desarrollado del software

La ingeniería informática en su afán de ser una ingeniería al nivel de otras menos recientes ha destinado muchos recursos y tiempo en especificar una forma *correcta* de desarrollar software. El proceso unificado (RUP) puede ser una gran solución para un determinado tipo de desarrollos, sin embargo, en un contexto real, su puesta en marcha puede conllevar muchos inconvenientes. Aún así sigue siendo un proceso referente y muy importante.

En el día de hoy hay muchos desarrollos que requieren de una mayor flexibilidad en los mecanismos de gestión y planificación, debido principalmente a la naturaleza real de los sistemas informáticos. La drástica evolución que sufre la tecnología informática no tiene parangón en otras tecnologías. Por ello, es necesario contar con otros modelos más cercanos a la situación en la que se encuentra multitud de desarrolladores de software, ya sean ingenieros o no, para poder alcanzar el mayor número de objetivos con el menor tiempo invertido; intentando en lo posible obtener artefactos de calidad.

Sería por otra parte un grave error olvidarse de la razón principal por la cual se está desarrollando este software. El desarrollo pertenece a una asignatura cursada por un alumno en un grado universitario. Es obvio que dicho desarrollo está delimitado y condicionado por la guía docente de dicha asignatura y a los plazos de presentación del centro educativo. También es importante remarcar que el tiempo que puede disponer un alumno no puede ser pleno y más aún si éste lo compagina con otras actividades o trabajos ajenos al grado cursado.

La naturaleza del trabajo de fin de grado además hace que haya muy pocos actores en su desarrollo. Únicamente los tutores involucrados en éste y el alumno. Esto lleva a que la necesidad de gestión de clientes, requisitos, reuniones... sea reducida.

El alumno siguió la metodología UPEDU pero sin aplicarla estrictamente y obviando ciertos procesos especificados en ella. Además de las razones anteriormente expuestas, se enumeran otras adicionales:

- Aunque uno de los clientes son los tutores, el alumno también es otro. Esto hace que el propio desarrollador conozca los requisitos y alcance del proyecto de primera mano.
- El alumno debe desarrollar todo el proceso de ingeniería del software, es decir, debe hacer la planificación y la ejecución del proyecto, así como el análisis, diseño, implementación y documentación. Es por ello que tendrá un gran conocimiento tanto del problema como de la solución proporcionada.
- Es un pequeño proyecto más o menos fácil de manejar por una única persona.

El resultado fue una gestión más ágil y menos burocrática en cierta medida, al prescindir de varios artefactos que en el contexto actual no suponía ninguna ventaja realizarlos.

3.3. Recursos

Antes de cualquier estimación temporal se realizó un estudio de los distintos recursos necesarios y recomendables para poder realizar correctamente las tareas propias del proyecto en cuestión; dado que el tipo de proyecto únicamente está relacionado con la informática, se han desglosado atendiendo a si son recursos de tipo hardware o software.

Respecto a los recursos hardware empleados solamente es necesario indicar que para el desarrollo fue necesario un ordenador con unas prestaciones de gama media.

Ordenador portátil	
CPU	Intel Core i5
RAM	4 GB DDR3
HDD	500 GB
OS	GNU/Linux Debian 7 para arquitectura de 64 bits.

Tabla 1: Recurso hardware

Por otra parte – la relacionada con los recursos software – se intentó en lo posible emplear herramientas y tecnologías de *software* libre para el desarrollo de la aplicación. Este hecho fue completamente deliberado por el alumno por su cercanía a este movimiento.

A continuación se listan y se proporciona una breve explicación de cada una de ellas:

- **Libreoffice**

Es un conjunto de aplicaciones de ofimática gratuitas y de código libre. Fue desarrollado por *The Document Foundation* después de su escisión de *OpenOffice* en 2010. Los programas de la suite permiten hacer procesamiento de texto, hojas de cálculo, dispositivas y esquemas.

Además emplea el estándar internacional de ISO *OpenDocument* como formato nativo de la aplicación, permitiendo así una gran compatibilidad de sus ficheros generados por otros programas de ofimática que tengan soporte para este tipo de formato.

Esta herramienta se empleó para realizar los documentos de texto de forma ágil y sencilla con el objetivo de compartirlos con el tutor en un repositorio común para su revisión y corrección.

La presente memoria está realizada íntegramente por dicha aplicación.

- **Texmaker**

Es un editor multiplataforma de documentos de tipo LaTeX. Licenciado como GNU GPL ofrece unas características muy apropiadas para este tipo de edición. Cuenta con un visualizador integrado de PDF en el cuál se compila el código existente. Además ofrece soporte unicode y asistente en la sintaxis.

Aunque se empezó a emplear este programa, finalmente el alumno decidió realizar los documentos finales empleando LibreOffice por el tiempo que demandaba.

- **Astah Community**

Es una herramienta para modelado de UML que cuenta con buena reputación dentro del mundo académico y profesional. La versión Community, aunque no es código libre si que se ofrece de forma gratuita.

- **REM**

Es un programa gratuito para la gestión de requisitos desarrollado en la universidad de Sevilla y que sigue los principios y metodologías propuestas por Amador Durán.

Aunque el alumno no empleó directamente el programa, si que se basó en sus plantillas a la hora de realizar las tablas relacionadas con los requisitos del sistema y los casos de uso de la etapa de análisis.

- **Eclipse**

Se trata de un entorno de desarrollo integrado (del inglés IDE) que ofrece numerosos servicios y herramientas para agilizar la codificación de programas. Es multiplataforma y aunque inicialmente fue pensado para editar programas en lenguaje Java su aceptación fue tan grande que soporta numerosos lenguajes de programación.

Una de sus principales características, y por la cuál se debe su éxito, es su capacidad incorporar nuevas funcionales mediante el uso de *plug-ins*.

- **Eclipse Aptana Plugin**

Es un plug-in *open source* desarrollado por la compañía Aptana para el IDE Eclipse. Gracias a él se aumentan considerablemente las capacidades del IDE, como por ejemplo soporte de JavaScript, HTML, CSS y DOM; junto con un asistente de sintaxis de dichos lenguajes. Además soporta otros lenguajes y frameworks como Ruby on Rails, PHP, Perl y Python.

- **Git**

Es un gestor de código fuente distribuido que fue creado por Linus Torvalds y distribuido como *software* libre. Desde su creación ha sido el SCM más empleado a nivel mundial para el desarrollo de *software*.

El alumno lo empleó con el objetivo principal de gestionar las versiones de los artefactos producidos y mantenerlos en un repositorio remoto a modo de *backup*. Dicho repositorio remoto está alojado en la plataforma BitBucket.

- **GanttProject**

Aplicación licenciada bajo GPLv3 y que permite realizar planificaciones y gestiones de proyectos mediante diagramas gantt. Es multiplataforma y una de las mejores soluciones para entornos GNU/Linux.

3.4. Entregables del proyecto

En el siguiente apartado se enumeran y describen los distintos artefactos que se han planificado elaborar. A continuación se enumeran sólo aquellos que son obligatorios de ser entregados como parte de las especificaciones de la asignatura en la que se está realizando este proyecto.

- **Memoria del proyecto**: documento que contiene toda la información del proyecto – desde su concepción hasta su ejecución final – detallando los procesos y la forma que se ha seguido para finalizarlo. Corresponde con el presente documento.
- **Software desarrollado**: se debe proporcionar todo el código fuente del software resultante de la ejecución del proyecto.

- **Manual de instalación**: documentación relacionada con el proceso de instalación del sistema generado así como sus posibles ficheros. Será incluido como anexo en la propia memoria del proyecto.
- **Manual de usuario**: documentación informativa sobre el uso correcto del sistema. En él se explica la forma de proceder para poder emplear la aplicación por parte del usuario final. Será incluido como anexo en la propia memoria del proyecto.

El resto de entregables que no son de carácter obligatorio están descritos en el anexo “Contenido del CD-ROM” del presente documento.

3.5. Organización del proyecto

En la actual sección se hablará de la estructura interna de la organización, así como las distintas interfaces externas, roles y responsabilidades existentes.

3.5.1. Estructura organizativa

El desarrollo íntegro del sistema así como la planificación y ejecución del proyecto será realizado por una única persona, concretamente por el alumno David Soler Herrero.

Sin embargo, y dado la asignatura que engloba este proyecto, el alumno es tutelado por un conjunto de profesores que actúan como supervisores y asesores. En este caso fue necesario la asistencia de dos profesores, Valentín Cardeñoso Payo como tutor principal del proyecto y César González Ferreras como cotutor.

La siguiente tabla asocia los distintos roles y responsabilidades con las personas involucradas:

Rol	Responsabilidades	Persona encargada
Gestor del proyecto	• Realizar una planificación del proyecto a seguir. • Realizar un seguimiento de la planificación durante su ejecución. • Corregir desviaciones surgidas tras su puesta en marcha. • Dar solución a posibles complicaciones durante el desarrollo del proyecto.	• David Soler Herrero
Analista	• Extracción de los requisitos de usuario de los clientes implicados en el proyecto. • Elicitación de los anteriores requisitos obtenidos en requisitos del sistema. • Definición y concreción del dominio del problema y posterior formalización. • Detallar los distintos casos de uso.	• David Soler Herrero
Diseñador	• Concretar una arquitectura lógica y física que mejor se adapte a los requisitos. • Evolucionar el análisis previo a una solución que esté contemplada la tecnología a emplear. • Refinar los diagramas de análisis mediante la adición y resolución de entidades que permitan su futura implementación. • Detallar el diseño de los datos.	• David Soler Herrero
Desarrollador	• Implementar las guías y especificaciones descritas en la fase de diseño. • Resolver adecuadamente diversas dificultades encontradas en esta fase de desarrollo. • Documentación del código generado.	• David Soler Herrero
Supervisor	• Controlar los contenidos y forma de la documentación entregada. • Asesorar y tutelar al alumno ante posibles complicaciones. • Avalar la calidad del proyecto. • Seguimiento del trabajo realizado.	• Valentín Cardeñoso Payo • César González Ferreras

Table 1: Roles y responsabilidades

3.5.2. Interfaces externas

En los trabajos de fin de grado normalmente los tutores encargados son también los clientes ya que son ellos los encargados de definir y concretar el proyecto que se quiere desarrollar. Por tanto se expondrá las interfaces resultantes al contemplarlos como clientes.

Valentín Cardeñoso Payo		
Datos de contacto	Teléfono	+34 983 185 601
	Email	valen@infor.uva.es
Tutorías	Tardes de días hábiles (necesidad de concertar cita previa).	

César González Ferreras		
Datos de contacto	Teléfono	+34 983 185 623
	Email	cesargf@infor.uva.es
Tutorías	Días hábiles (necesidad de concertar cita previa).	

3.6. Proceso de gestión

En esta sección se detalla toda la información relacionada con la planificación del proyecto. Desde las estimaciones realizadas y fases necesarias para completar correctamente el proyecto en cuestión, hasta las distintas dificultades que motivaron rectificaciones y desviaciones en la planificación realizada.

Las estimaciones temporales se concretaron siguiendo los criterios descritos en la asignatura de Planificación y Gestión de Proyectos junto con la experiencia adquirida por el alumno tras el desarrollo y entrega de trabajo durante todo su ciclo formativo superior. Además, las estimaciones se realizaron empleando únicamente días hábiles (Lunes a Viernes).

3.6.1. Enumeración de tareas

A continuación se enumeraran las tareas que fueron necesarias para concluir correctamente con el desarrollo del sistema propuesto.

- **Planificación**: en ella se realiza un análisis inicial del conjunto de tareas necesarias y se estima su coste tanto temporal como de recursos. También se describen los distintos riesgos existentes del proyecto.
- **Experimentación inicial**: relacionada con el estudio y adquisición de conocimientos de las distintas tecnologías que se van a emplear en el sistema. Gracias a ella se permite al estudiante tener un conocimiento real de las ventajas y limitaciones de cada una de ellas y así tenerlo en cuenta en fases posteriores.
- **Análisis**: engloba todas las tareas relacionadas con la obtención y elicitación de requisitos del sistema, la definición del modelo de dominio así como la descripción de los casos de uso resultantes.
- **Diseño**: relacionado con la especificación de la interfaz de usuario y la adaptación de los artefactos elaborados en la fase previa a la tecnología que se va a emplear en el sistema final.
- **Implementación**: codificación de las especificaciones descritas en la fase anterior y cuyo objetivo principal es el desarrollo real de la aplicación y módulos dependientes.
- **Pruebas**: definición y ejecución de una batería de pruebas capaces de verificar la consistencia del código y el comportamiento real del sistema ante diversos escenarios contemplados.
- **Revisión**: focalizada en la comprobación de los distintos artefactos que se van generando a lo largo del resto de fases.
- **Documentación**: descripción y explicación del desarrollo de cada una de las fases anteriores así como la relacionada con los artefactos generados.

3.6.2. Pre-planificación

La primera reunión entre el alumno y el tutor – Valentín Cardeñoso – fue el día 13 de febrero de 2014. Su finalidad fue introducir al alumno en la asignatura y al proyecto que le competía.

Posteriormente, se necesitaron un par de reuniones para definir el plan de actuación inicial. Sirvieron también para exponer claramente que se esperaba del proyecto y cuál era su alcance. Además se propusieron diversas tecnologías que el alumno debía analizar para así elegir la más adecuada para el desarrollo posterior de la aplicación. Se estimaron por lo menos dos semanas para la realización de dicho análisis y estudio.

Gracias a las reuniones previas, se pudo realizar un análisis de las tareas necesarias para alcanzar a completar los objetivos planteados en este trabajo fin de grado. A continuación se enumeran:

- Aprender y analizar el nuevo paradigma que la web ha conformado. Así como conocer entornos de desarrollo y lenguajes de programación relacionados con la web. Aprender a estructurar una aplicación web y a manejar frameworks de aplicaciones webs.
- Debatir y analizar una metodología a seguir y generar una planificación del proyecto.
- Aprender a gestionar un desarrollo de un sistema informático amplio mediante el uso de sistemas de control de versiones.
- Acometer un análisis de los requisitos propios del sistema, partiendo de los objetivos dispuestos por los interesados en el proyecto.
- Proporcionar una solución a los requisitos elicitados mediante la realización de un diseño de la aplicación.
- Implementar el diseño realizado mediante un prototipo funcional.
- Mantener y gestionar una documentación actualizada del código así como de diversos artefactos originados durante la ejecución del proyecto.
- Depurar los diversos errores y fallos que vayan surgiendo en la fase de implementación para así conseguir un prototipo lo más robusto posible.

Una vez definidas las tecnologías a emplear, se dejó un amplio margen de tiempo para que el alumno las estudiara y se familiarizara con ellas. De esta forma se le permitiría tener una visión global de las capacidades y limitaciones que éstas tuvieran y tenerlo en cuenta a la hora de plantear la solución al problema en cuestión.

Este proceso de aprendizaje inicial demoró más o menos un mes y le sirvió para que tuviera una imagen general de las tecnologías elegidas y cómo debían interaccionar.

Una vez concluido lo anterior, realizó la siguiente estimación general para poder llegar a presentar el trabajo en la primera convocatoria de 2014.

Name	Begin date	End date
Fase de iniciación	13/02/14	03/04/14
Primera reunión	13/02/14	13/02/14
Reuniones iniciales	14/02/14	20/02/14
Análisis de las tecnologías	21/02/14	06/03/14
Experimentación inicial	07/03/14	03/04/14
Fase de análisis	04/04/14	24/04/14
Fase de diseño	25/04/14	15/05/14
Fase de implementación	16/05/14	19/06/14
Fase de seguimiento	04/04/14	26/06/14

Tabla 2: Estimación preinicial

Sin embargo, después de un mes de las primeras entrevistas con el tutor, se evidenció que la planificación anteriormente esbozada era irrealista con la nueva situación que tuvo que hacer frente el alumno cuando terminó el primer cuatrimestre del cuarto curso del grado.

La principal dificultad encontrada fue a la hora de compaginar el resto de asignaturas del segundo cuatrimestre. Especialmente la asignatura de prácticas en empresa – que el alumno comenzó el día 5 de marzo – fue la que más tiempo tuvo que dedicar. Al día juntando desplazamiento y estancia en el puesto de trabajo demoraba alrededor de siete horas, teniendo sólo fines de semana libres. Una vez concluidas a inicios de julio, la empresa le brindó la posibilidad de continuar trabajando durante seis meses más, oferta que el alumno aceptó por la gran oportunidad que suponía para su desarrollo profesional y formativo.

Además de esta situación, tuvo que cursar la asignatura propuesta para el segundo cuatrimestre, que por la forma de como se impartía motivaba y fomentaba con creces a la experimentación y aprendizaje auto-didáctico adicional, lo que provocó que el alumno realizara un gran número de horas extracurriculares.

Todo ello motivó que el tiempo destinado al proyecto durante este tiempo fuese muy reducido e incluso en varios momentos prácticamente nulo, por lo que fue verdaderamente complicado seguir los plazos marcados al inicio.

Sobre julio y ante la inminente fecha de presentación de septiembre, el alumno decidió por voluntad propia dejar el puesto de trabajo en agosto para en lo posible dedicarlo íntegramente al desarrollo y finalización del proyecto. Sin embargo, al final del mes, la no trivialidad del problema a resolver y más aún la dificultad de los frameworks empleados no hicieron más que imposibilitar llegar a la fecha prevista.

3.6.3. Plan de proyecto

Tras las adversidades mencionadas en la sección anterior el alumno retomó el trabajo de fin de grado en el mes de octubre. Se realizó una nueva planificación para ser ejecutada en los siguientes meses, comenzando primero con la especificación del plan de fases.

3.6.3.1. Plan de fases

Siguiendo las especificaciones descritas en UPEDU, se dividió el proyecto en cuatro fases bien diferenciadas. En cada una de ellas se especifica un número finito de iteraciones así como el intervalo de tiempo necesario para realizarlas.

Fase	Número Iteraciones	Fecha de inicio	Fecha de fin	Duración
Inicio	1	06/10/14	15/10/14	8 días
Elaboración	1	16/10/14	20/11/14	26 días
Construcción	4	21/11/14	08/01/15	35 días
Transición	1	09/01/15	12/01/15	2 días

Tabla 3: Plan de fases

Como se puede apreciar, la primera fase es más breve de lo normal. La razón principal radica a que parte ya se realizó anteriormente.

La fecha de entrega final sería la convocatoria extraordinaria de enero, cuyo día límite era el 12.

Los hitos más importantes que marcan el final de cada fase se detalla en la siguiente tabla.

Fases	Hito
Inicio	• Entendimiento pleno de los objetivos del sistema y de sus requisitos y restricciones. • Obtención de todos los requisitos de usuario. • Primer boceto de la arquitectura candidata. • Enumeración de los casos de uso necesarios. • Modelo de dominio.
Elaboración	• Refinamiento del modelo de dominio. • Elaboración de los casos de uso. • Realización de los diagramas de secuencia de análisis. • Refinamiento de los diagramas de diseño. • Elaboración de la arquitectura lógica de todo el sistema.
Construcción	• Implementación de los casos de uso contemplados. • Documentación de la aplicación. • Versión incompleta de la memoria y anexos. • Depuración de la implementación.
Transición	• Prueba final de la implementación. • Memoria y entregables finalizados.

Tabla 4: Hitos requeridos por cada fase

3.6.3.2. Análisis de riesgos

A continuación se detallan las posibles eventualidades que pueden afectar negativamente a la planificación realizada previamente. Sólo se han destacados aquellos más importantes y que posean una frecuencia razonable para ser considerados.

RI – 1	Planificación demasiado optimista
Descripción	Las estimaciones de las tareas a realizar no están ajustadas a la realidad y demoran más tiempo de lo previsto inicialmente.
Efecto	Retraso en las tareas.
Gravedad	Alta
Frecuencia	Moderada
Acción correctora	Definir intervalos temporales más flexibles y amplios con el objetivo de conseguir una holgura adicional.
Plan de contingencia	Invertir más horas/hombre en la realización de la tarea actual y consecutivas hasta volver a cumplir los plazos de la planificación.

Tabla 5: Riesgo – 1

RI – 2	Pérdida de información
Descripción	Pérdida de parte de información relacionada con el proyecto.
Efecto	Retraso en las tareas.
Gravedad	Alta
Frecuencia	Mínima
Acción correctora	Emplear sistemas de gestión de configuraciones y realizar frecuentes commits para mantener la información sincronizada.
Plan de contingencia	Invertir más horas/hombre en la realización de las tareas que se hayan visto afectadas por este suceso.

Tabla 6: Riesgo – 2

RI – 3	Planificación irreal
Descripción	Las estimaciones realizadas de las tareas no son realistas e imposibilitan llegar a la fecha límite de entrega.
Efecto	Retraso de la fecha de entrega a la siguiente convocatoria.
Gravedad	Extraordinariamente alta
Frecuencia	Media
Acción correctora	Seguimiento continuo de la planificación y actuación rápida y contundente ante posibles desviaciones.
Plan de contingencia	Retraso de la fecha de entrega a la siguiente convocatoria.

Tabla 7: Riesgo – 3

RI – 4	Desconocimiento de las tecnologías empleadas
Descripción	Necesidad de invertir tiempo adicional en el aprendizaje de una tecnología concreta que es empleada en el sistema.
Efecto	Retraso en las tareas afectadas por dicho desconocimiento.
Gravedad	Media
Frecuencia	Media
Acción correctora	Realizar una planificación en la que se tenga en cuenta este tipo de situaciones.
Plan de contingencia	Invertir más horas/hombre para la adquisición del conocimiento necesario o en caso excepcional retrasar la tarea.

Tabla 8: Riesgo – 4

RI – 5	Enfermedades, trabajos y eventos imprevistos
Descripción	Eventualidades imprevistas que surgen después de concretar la planificación y que se caracterizan porque son difíciles de predecir.
Efecto	Retraso en las tareas.
Gravedad	Dependiendo de la eventualidad.
Frecuencia	Impredecible
Acción correctora	Definir intervalos temporales más flexibles y amplios con el objetivo de conseguir una holgura adicional para poder mitigar posibles efectos adversos.
Plan de contingencia	Invertir más horas/hombre para subsanar el retraso producido.

Tabla 9: Riesgo – 5

RI – 6	Equipo de desarrollo afectado
Descripción	El recurso hardware empleado para el desarrollo del proyecto sufre una avería que lo inutiliza o es perdido.
Efecto	Retraso en las tareas.
Gravedad	Aumenta con el tiempo.
Frecuencia	Impredecible
Acción correctora	Contar con equipos de repuesto.
Plan de contingencia	Encontrar un nuevo equipo lo antes posible e invertir más horas/hombre ante posibles retrasos producidos.

Tabla 10: Riesgo – 6

RI – 7	Cambios en los requisitos
Descripción	Se produce un cambio sustancial sobre un requisito concreto.
Efecto	Retraso en las tareas.
Gravedad	Aumenta con el tiempo.
Frecuencia	Media
Acción correctora	Contar con una planificación cuyas tareas tengan holgura suficiente como para poder compensar el restraso ocasionado.
Plan de contingencia	Invertir más horas/hombre ante posibles retrasos producidos.

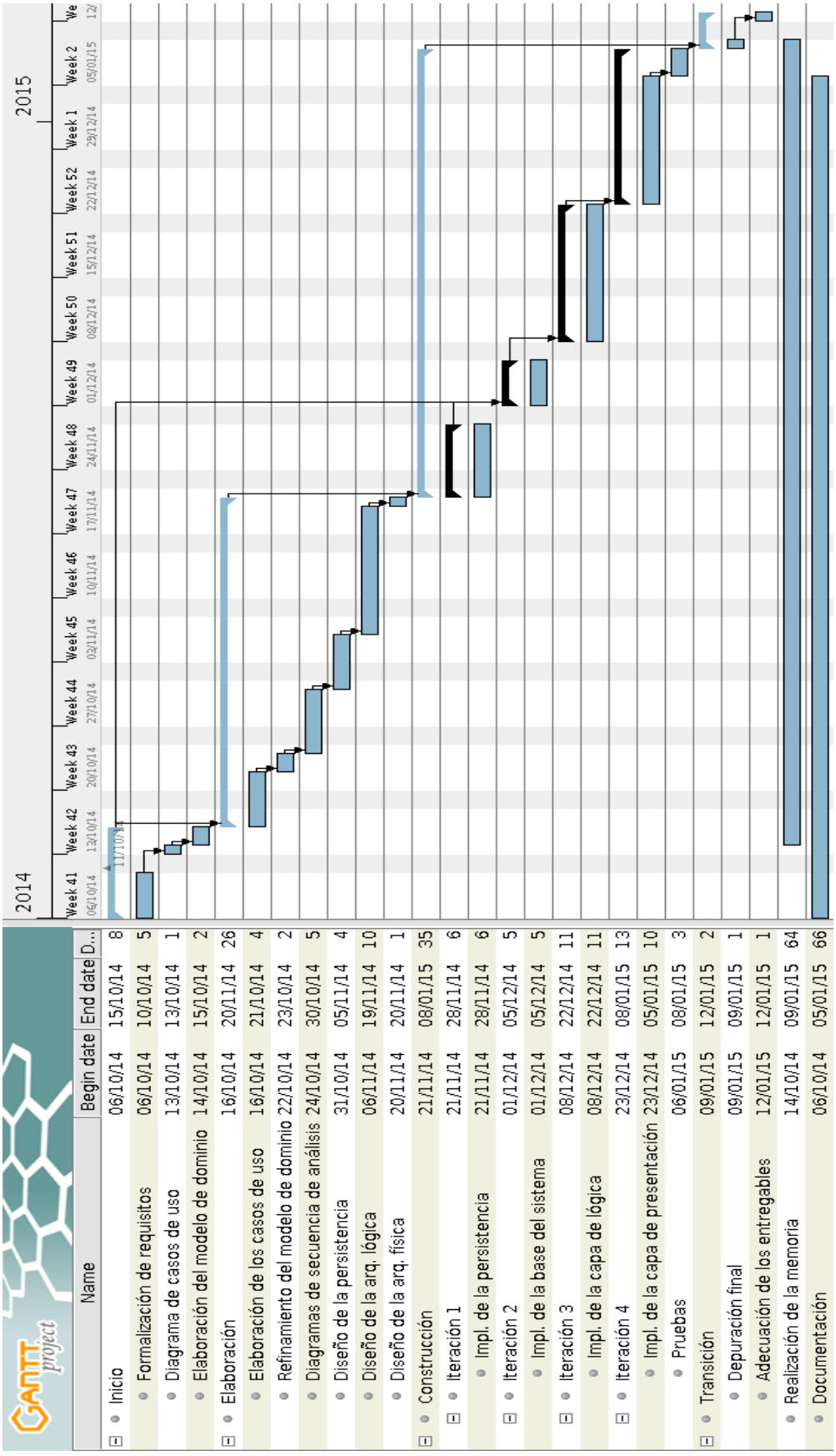
Tabla 11: Riesgo – 7

RI – 8	Interpretación incorrecta de los requisitos
Descripción	Al transformar un requisito de usuario a de sistema se puede pervertir el objetivo real que se demandaba.
Efecto	Retraso en las tareas afectadas por la mala interpretación.
Gravedad	Aumenta con el tiempo.
Frecuencia	Media
Acción correctora	Revisar y explicar todos los requisitos elicitados al cliente.
Plan de contingencia	Invertir más horas/hombre hasta mitigar los retrasos producidos.

Tabla 12: Riesgo – 8

3.6.3.3. Planificación inicial

En esta sección se detallará la planificación realizada a partir de octubre y cuyo objetivo primordial fue poder generar los artefactos necesarios para la convocatoria de enero de 2015.



Name	Begin date	End date	D...
Inicio	06/10/14	15/10/14	8
Formalización de requisitos	06/10/14	10/10/14	5
Diagrama de casos de uso	13/10/14	13/10/14	1
Elaboración del modelo de dominio	14/10/14	15/10/14	2
Elaboración	16/10/14	20/11/14	26
Elaboración de los casos de uso	16/10/14	21/10/14	4
Refinamiento del modelo de dominio	22/10/14	23/10/14	2
Diagramas de secuencia de análisis	24/10/14	30/10/14	5
Diseño de la persistencia	31/10/14	05/11/14	4
Diseño de la arq. lógica	06/11/14	19/11/14	10
Diseño de la arq. física	20/11/14	20/11/14	1
Construcción	21/11/14	08/01/15	35
Iteración 1	21/11/14	28/11/14	6
Impl. de la persistencia	21/11/14	28/11/14	6
Iteración 2	01/12/14	05/12/14	5
Impl. de la base del sistema	01/12/14	05/12/14	5
Iteración 3	08/12/14	22/12/14	11
Impl. de la capa de lógica	08/12/14	22/12/14	11
Iteración 4	23/12/14	08/01/15	13
Impl. de la capa de presentación	23/12/14	05/01/15	10
Pruebas	06/01/15	08/01/15	3
Transición	09/01/15	12/01/15	2
Depuración final	09/01/15	09/01/15	1
Adecuación de los entregables	12/01/15	12/01/15	1
Realización de la memoria	14/10/14	09/01/15	64
Documentación	06/10/14	05/01/15	66

Figura 1: Planificación inicial

3.6.3.4. Desviaciones y dificultades sufridas

En este apartado se comentaran las distintas desviaciones sufridas a lo largo de la ejecución de la planificación y las razones que motivaron la cadena de retrasos que impidieron llegar a la fecha límite.

Inicialmente se decidió emplear Ruby on Rails para hacer tanto la parte del front-end como el back-end. Esta elección fue tomada por el alumno después de ver la ventajas que este framework ofrecía y además por su deseo de aprender un nuevo lenguaje y framework de desarrollo web diferente al que ya conocía (PHP).

Sin embargo, poco a poco se fue evidenciando ciertas limitaciones que eran bastante tediosas de solucionar, como por ejemplo, conseguir una aplicación que siguiera el patrón de una única página o programar ciertos componentes visuales.

La solución fue reformular la forma de abordar el problema a resolver. Para ello se decidió mantener Ruby on Rails, pero cambiando su cometido principal. Dado su gran potencial produciendo servidores de tipo RESTful, se eliminó toda la lógica perteneciente a la capa de presentación y se programó para servir únicamente como un servidor REST mediante una API web service.

Esto llevó a cierta eliminación de código relacionado con la capa de presentación y a la reprogramación de ciertas partes del código – el cual fue generado en el mes de agosto intensivamente. Se comenzó la transformación el 29 de octubre y llevó prácticamente un mes realizar todas las modificaciones pertinentes; lo que fue añadiendo retrasos continuados sobre el resto de tareas que se habían planificado anteriormente.

Además del surgimiento esporádico de errores al ejecutar la aplicación a finales del mes de diciembre obligaron al alumno a posponer la fecha de entrega.

A principios de enero, las etapas de análisis y diseño estaban finalizadas. Quedaba por terminar la implementación del código y la realización de la memoria.

A partir de febrero el tiempo disponible para la realización de estas tareas disminuyó a raíz de la concesión al alumno de una beca remunerada, la cual finalizó en mayo. La carga diaria necesaria era de 5 horas (incluyendo transporte) y motivó a que tuviera que compaginar ambas obligaciones.

A lo largo de esos meses, se dedicaba en lo posible las tardes para continuar con la implementación y la confección de la memoria.

En este periodo la única complicación encontrada fue la complejidad y dificultad de los frameworks empleados, sobretodo aquel relacionado con el cliente (ExtJS). A menudo surgían nuevas situaciones que obligaban al alumno a estudiar una solución que podría demorar un par de días. Estos desfases se resolvían en lo posible invirtiendo más horas/hombre.

Capítulo 4

ANÁLISIS

4.1. Introducción

En el presente capítulo se expondrán todas las cuestiones relativas al análisis realizado durante el desarrollo de la aplicación.

En esta primera sección se comentará brevemente el propósito y el alcance de este capítulo.

A continuación se encuentra el análisis de contexto, cuya finalidad es conocer las características y peculiaridades que deben ser consideradas para un correcto desarrollo.

Posteriormente, se analizan los perfiles y tareas existentes en el sistema a desarrollar. Continuando con la obtención de los requisitos del sistema y la elaboración de los casos de uso.

El capítulo finaliza con el modelado del dominio, en el cual se describen los conceptos esenciales y se explica el proceso llevado para obtenerlos.

4.1.1. Propósito

El de detectar y definir las funciones básicas del sistema, describiendo las principales interacciones de los distintos actores con el sistema, y proponer diversos modelos que servirán de base para la siguiente fase de diseño.

4.1.2. Alcance

Detallar los aspectos esenciales para realizar un correcto proceso de diseño en el siguiente paso del proyecto.

4.2. Análisis de contexto

La aplicación está orientado a un público muy variado tanto por su diversidad cultural como por su nivel en el uso y conocimiento de nuevas tecnologías.

El público que realizará ejercicios para mejorar su pronunciación – o posibles visitantes de la aplicación – será tremendamente heterogéneo ya que proviene de distintas partes del mundo, lo que implica la existencia de una gran variedad de sensibilidades y formas de pensamiento característicos. Sin embargo, el resto de roles se puede considerar que lo realizarán personas nativas al lenguaje de estudio, es decir, personas de nacionalidad española o hispanohablantes. Esto conlleva que las tareas realizadas por estos últimos se puedan diseñar atendiendo a los paradigmas propios de la población hispana.

Recientemente e impulsado por grandes entidades en Internet, se tiende cada vez más a realizar interfaces de usuarios más sencillas y minimalistas. Este factor debe tenerse en cuenta a la hora de diseñar tanto el estilo como el flujo de las interfaces. De esta forma se consigue que gran parte del público interaccione con ellas de una forma más intuitiva y adecuada sin necesidad de explicaciones o ayudas adicionales.

Cabe también señalar, que dentro del público existen distintos grados de dominio de la tecnología actual. Lo que conlleva a definir metáforas lo más identificativas y comunes a todo el espectro y conseguir también un flujo de control lo más natural y lógico posible.

El acceso al contenido de la web ya no sólo se hace mediante un ordenador convencional. Desde hace

casi un lustro, se ha producido una explosión de nuevos dispositivos – *smartphones* y *tablets* – que acceden directamente a la web. Esto motiva a diseñar aplicaciones webs que puedan adaptarse a las nuevas características de estos dispositivos. Ya no solo por el nivel de cómputo que poseen, que actualmente se asemeja al de ordenadores, sino a las dimensiones de la pantalla, que en lo posible se tiene que conseguir una convergencia entre ellos.

4.3. Análisis de perfiles

En esta sección se definirán los distintos perfiles que existirán en el sistema junto con una descripción en la que se detalle sus características propias y su alcance de actuación dentro del sistema. Siendo los posibles roles los siguientes:

- **Visitante**: cualquier persona que acceda a la aplicación web únicamente para obtener información relacionada con esta. Tiene la posibilidad de ver aquella documentación e información que se haya catalogado como de acceso público. No requiere ningún tipo de privilegio o identificación previa.
- **Donante**: aquella persona que cede voluntariamente una o varias locuciones al sistema. Para realizar esta cesión deberá seguir un procedimiento distinto al de realización de ejercicios de pronunciación. Necesita estar identificado previamente ante el sistema y tener el rol especificado en su usuario.
- **Estudiante**: persona interesada en emplear la aplicación para mejorar su pronunciación del español a través de la realización de ejercicios. Puede ser de cualquier nacionalidad y posee al menos un nivel básico de la lengua de estudio. Necesita estar identificado previamente y pertenecer al menos a un grupo.
- **Gestor**: profesional que realiza operaciones de gestión sobre el conjunto de corpus existente en el sistema. Realiza tareas de organización, borrado, clasificación y análisis de corpus dentro de la aplicación web. Requiere estar identificado previamente y tener los privilegios adecuados para realizar las operaciones mencionadas anteriormente.
- **Evaluador**: profesional que evalúa y puntúa las grabaciones existentes en una muestra de corpus. Necesita estar identificado ante el sistema y tener especificado dicho rol.
- **Administrador**: persona encargada de realizar operaciones básicas de mantenimiento y gestión de usuarios y perfiles asociados. Normalmente será un grupo muy reducido de personas, incluso llegando a ser una única persona. Nunca podrá realizar operaciones relacionadas con la gestión de corpus. Debe estar identificado en el sistema previamente.

4.4. Análisis de tareas

Se realizó un pequeño análisis de las posibles tareas existentes en la aplicación a desarrollar.

- Un visitante busca información o documentación pública.
- Un usuario se identifica ante el sistema.
- Un usuario actualiza su información personal.
- Un estudiante accede a un ejercicio.

- Un estudiante ve información de un ejercicio.
- Un estudiante ve el *feedback* de un ejercicio que haya realizado.
- Un administrador realiza operaciones de mantenimiento.
- Un gestor crea un nuevo corpus.
- Un gestor reorganiza un conjunto de corpus.
- Un gestor elimina un corpus o grabación del servidor.
- Un evaluador puntúa y corrige un ejercicio realizado por un estudiante.
- Un estudiante realiza un ejercicio.
- Un donante dona una locución.
- Un gestor envía un nuevo conjunto de corpus al sistema.
-

4.5. Especificación de requisitos

En este apartado se detallarán tanto los requisitos de usuario y requisitos de sistema, siendo éstos la combinación de los funcionales, no funcionales y de información. De esta forma se puede concretar el alcance y el qué debe hacer la aplicación además de imponer ciertas restricciones.

4.5.1. Requisitos de usuario (objetivos)

En este subapartado se recogen aquellos objetivos o requisitos de usuario que fueron enumerados por el usuario. Se caracterizan por su poco grado de detalle y simplicidad léxica.

Identificador	OBJ-01
Descripción	La aplicación tiene que poder grabar desde un navegador web y almacenar la grabación de forma adecuada en el servidor.

Identificador	OBJ-02
Descripción	Se debe permitir la gestión a través de la aplicación de todas las grabaciones almacenadas, pudiendo reorganizarlas, eliminarlas y añadir información adicional.

Identificador	OBJ-03
Descripción	Sólo los usuarios de tipo gestor pueden crear, manipular y gestionar los corpus.

Identificador	OBJ-04
Descripción	Tiene que tener la capacidad de poder enviar un gran número de grabaciones al servidor.

Identificador	OBJ-05
Descripción	Sólo los evaluadores podrán evaluar los ejercicios realizados por los estudiantes.

Identificador	OBJ-06
Descripción	Debe existir información y documentación pública que puede ser accesible por cualquier visitante de la aplicación.

Identificador	OBJ-07
Descripción	Un estudiante registrado puede realizar ejercicios y ver el feedback resultante tras la realización del ejercicio en cualquier momento.

Identificador	OBJ-08
Descripción	Se debe dar la capacidad de que usuarios que no pertenezcan a ningún curso puedan donar grabaciones.

Identificador	OBJ-09
Descripción	Los donantes pueden listar y ver información de todos los corpus que hayan cedido al sistema.

Identificador	OBJ-10
Descripción	La aplicación debe contar con un portal de administración básico para realizar operaciones de mantenimiento y pequeños ajustes de configuración de la aplicación.

Identificador	OBJ-11
Descripción	La aplicación web debe ser accesible desde cualquier navegador moderno y su visualización deber ser idéntica en todos ellos.

Identificador	OBJ-12
Descripción	La programación de la aplicación debe ser puramente orientada a objetos.

Identificador	OBJ-13
Descripción	La aplicación debe ser robusta y recuperarse de todos los posibles errores.

4.5.2. Requisitos funcionales

A continuación se expondrán y describirán los requisitos funcionales que la aplicación debe satisfacer. Estos requisitos determinan qué es lo que tiene que hacer la aplicación realizada.

Identificador	FRQ-01
Referencia a	OBJ-10
Descripción	El sistema deberá permitir la gestión de usuarios por parte de un usuario administrador.

Identificador	FRQ-02
Referencia a	
Descripción	El sistema deberá ser capaz de guardar y restablecer una sesión de usuario.

Identificador	FRQ-03
Referencia a	OBJ-03, OBJ-02
Descripción	El sistema deberá permitir la manipulación de los corpus almacenados a los usuarios de tipo gestor.

Identificador	FRQ-04
Referencia a	
Descripción	El sistema deberá permitir a los usuarios gestores subir corpus empleando la aplicación o mediante un programa externo.

Identificador	FRQ-05
Referencia a	OBJ-05
Descripción	El sistema deberá permitir a los usuarios evaluadores anotar y evaluar los resultados obtenidos tras la realización de un ejercicio por parte de un usuario de tipo estudiante.

Identificador	FRQ-06
Referencia a	OBJ-08
Descripción	El sistema deberá dar la posibilidad de donar corpus a los usuarios donantes sin que éstos estén inscritos a cualquier tipo de grupo o ejercicio.

Identificador	FRQ-07
Referencia a	OBJ-01
Descripción	El front-end deberá tener la capacidad de grabar audio y enviarlo al back-end.

Identificador	FRQ-08
Referencia a	OBJ-07
Descripción	Un usuario de tipo estudiante podrá ver los resultados y anotaciones de sus ejercicios realizados.

Identificador	FRQ-09
Referencia a	OBJ-07
Descripción	El sistema deberá permitir a un usuario de tipo estudiante realizar ejercicios mediante el front-end.

Identificador	FRQ-10
Referencia a	OBJ-03
Descripción	El sistema deberá permitir a un usuario de tipo gestor añadir y eliminar campos de un nodo.

Identificador	FRQ-11
Referencia a	OBJ-05
Descripción	El sistema deberá permitir a un usuario de tipo evaluador añadir y eliminar campos de una evaluación de una muestra.

Identificador	FRQ-12
Referencia a	
Descripción	El sistema deberá permitir a un usuario visualizar la información de su perfil.

4.5.3. Requisitos no funcionales

Los siguientes requisitos aunque no determinan el qué debe hacer la aplicación son también de vital importancia ya que imponen restricciones que deben ser completamente satisfechas.

Identificador	NFR-01
Referencia a	
Descripción	El sistema debe tener implementado un sistema de permisos que definan qué tareas pueden llevar a cabo cada usuario.

Identificador	NFR-02
Referencia a	
Descripción	El back-end debe ser un servidor de tipo REST que ofrezca una API por la cual se pueda acceder a sus servicios.

Identificador	NFR-03
Referencia a	
Descripción	El back-end deberá tener un sistema de acreditaciones de usuarios mediante sesiones y tokens de acceso.

Identificador	NFR-04
Referencia a	
Descripción	El back-end deberá permitir la existencia de más de una sesión de usuario activa.

Identificador	NFR-05
Referencia a	
Descripción	El back-end deberá contar con un sistema de seguridad seguro y fiable.

Identificador	NFR-06
Referencia a	OBJ-12
Descripción	La programación de la aplicación debe ser orientada a objetos y seguir patrones de diseño adecuados al contexto de la aplicación.

Identificador	NFR-07
Referencia a	OBJ-13
Descripción	La aplicación debe ser robusta y recuperarse del 95% de los posibles errores. Salvo aquellos relacionados intrínsecamente al framework empleado para el front-end.

Identificador	NFR-08
Referencia a	
Descripción	El front-end empleará las últimas tecnologías de la web, concretamente HTML5/CSS3 y JavaScript.

Identificador	NFR-09
Referencia a	
Descripción	El back-end deberá aportar una API clara y robusta.

Identificador	NFR-10
Referencia a	OBJ-11
Descripción	El front-end deberá ser utilizado en cualquier dispositivo con soporte HTML5.

Identificador	NFR-11
Referencia a	OBJ-11
Descripción	El front-end debe ser convergente a todos los dispositivos soportados.

Identificador	NFR-12
Referencia a	
Descripción	El sistema deberá ser extensible pudiendo añadir nuevas funcionalidades de forma fácil y sencilla.

Identificador	NFR-13
Referencia a	
Descripción	Todo el desarrollo deberá realizarse con software libre y respetar sus licencias.

4.5.4. Requisitos de información

Seguidamente se muestra los requisitos de información asociados a la aplicación. Además de una descripción de cada uno de ellos se enuncian sus campos asociados.

Identificador	IRQ-01	
Nombre	Role	
Descripción	Define un rol de usuario que determinará que tipo de acciones y tareas se le es permitido hacer.	
Campos	Actions	Conjunto de acciones que determinan su grado de actuación ante el sistema.

Identificador	IRQ-02	
Nombre	User	
Descripción	Entidad que especifica los campos más relevantes de un usuario del sistema.	
Campos	Username	Nombre unívoco de un usuario.
	Contraseña	Mediante la cual se puede acreditar ante el sistema.
	Name	Nombre real del usuario.
	Surname	Apellidos del usuario.
	Email	Dirección de correo electrónico de contacto.
	Status	Determina en que estado se encuentra la cuenta de usuario, es decir, si está pendiente de aprobación, bloqueado...
	Role	Referencia a un rol único que determinará la forma en la que podrá utilizar el sistema.

Identificador	IRQ-03	
Nombre	Session	
Descripción	Dispondrá de la información suficiente para permitir a un usuario a través de ésta identificarse ante el sistema.	
Campos	User	Usuario al que está asociada la sesión.
	Access token	Token unívoco de acceso al sistema.
	Where	Ubicación donde se creó la sesión.
	Who	Información de la aplicación que creó la sesión.

Identificador	IRQ-04	
Nombre	Field Metadata	
Descripción	Información abstracta que determina el tipo y esencia de un Field	
Campos	Key	Nombre que tomará el Field.
	Sort	Especifica el tipo del valor que se almacenará en el Field.
	SourceFn	Concreta la fuente de información por la cual se acotará el número de valores posibles.
	Info	Información adicional.

Identificador	IRQ-05	
Nombre	Field	
Descripción	Entidad que será capaz de almacenar cualquier tipo de valor.	
Campos	Data	Es el campo donde se almacenará el valor.
	ReadOnly	Determina si su edición está bloqueada o no.
	Field Metadata	Referencia a su metadata correspondiente.
	Field	Referencia al campo que sobreescribirá su valor.

Identificador	IRQ-06	
Nombre	Schema	
Descripción	Determina la estructura básica de un nodo y cuya finalidad está muy cercana al de una plantilla.	
Campos	Name	Nombre del esquema.
	Sort	Determina a que tipo de nodo corresponde.
	User	Referencia al usuario que creó el esquema.

Identificador	IRQ-07	
Nombre	Corpus	
Descripción	Conjunto de datos y materiales sobre una determinada materia.	
Campos	Name	Nombre del corpus.
	Schema	Referencia al esquema que replicó.
	User	Referencia al usuario que lo realizó.
	Corpus Sessions	Colección de sesiones de corpus que penden de él.

Identificador	IRQ-08	
Nombre	Corpus Session	
Descripción	Concreta una sesión, es decir, una toma de muestras dentro de una franja de tiempo.	
Campos	Name	Nombre de la sesión.
	Schema	Referencia a su esquema.
	User	Referencia al usuario que realizó la sesión.
	Corpus Samples	Colección de muestras relacionadas con el corpus.
	Corpus Chapters	Colección de sesiones de iteraciones que penden de ella.

Identificador	IRQ-09	
Nombre	Corpus Chapter	
Descripción	Concreta una iteración de capturas.	
Campos	Name	Nombre de la iteración.
	Schema	Referencia a su esquema.
	User	Referencia al usuario que realizó el nodo.
	Corpus Captures	Colección de capturas que penden de ella.

Identificador	IRQ-10	
Nombre	Corpus Capture	
Descripción	Corresponde a una captura de un corpus.	
Campos	Name	Nombre de la sesión.
	Schema	Referencia a su esquema.
	User	Referencia al usuario que realizó el nodo.

4.6. Definición de actores

Un actor es aquella entidad externa del sistema que interactúa de una forma concreta y que realiza un conjunto de tareas y acciones en el sistema. Es por tanto que es muy importante concretarlos si se quiere desarrollar una aplicación que satisfaga todas sus necesidades.

En este caso, los actores corresponden a los perfiles analizados en el apartado 4.3 del presente documento.

4.7. Diagrama de casos de uso

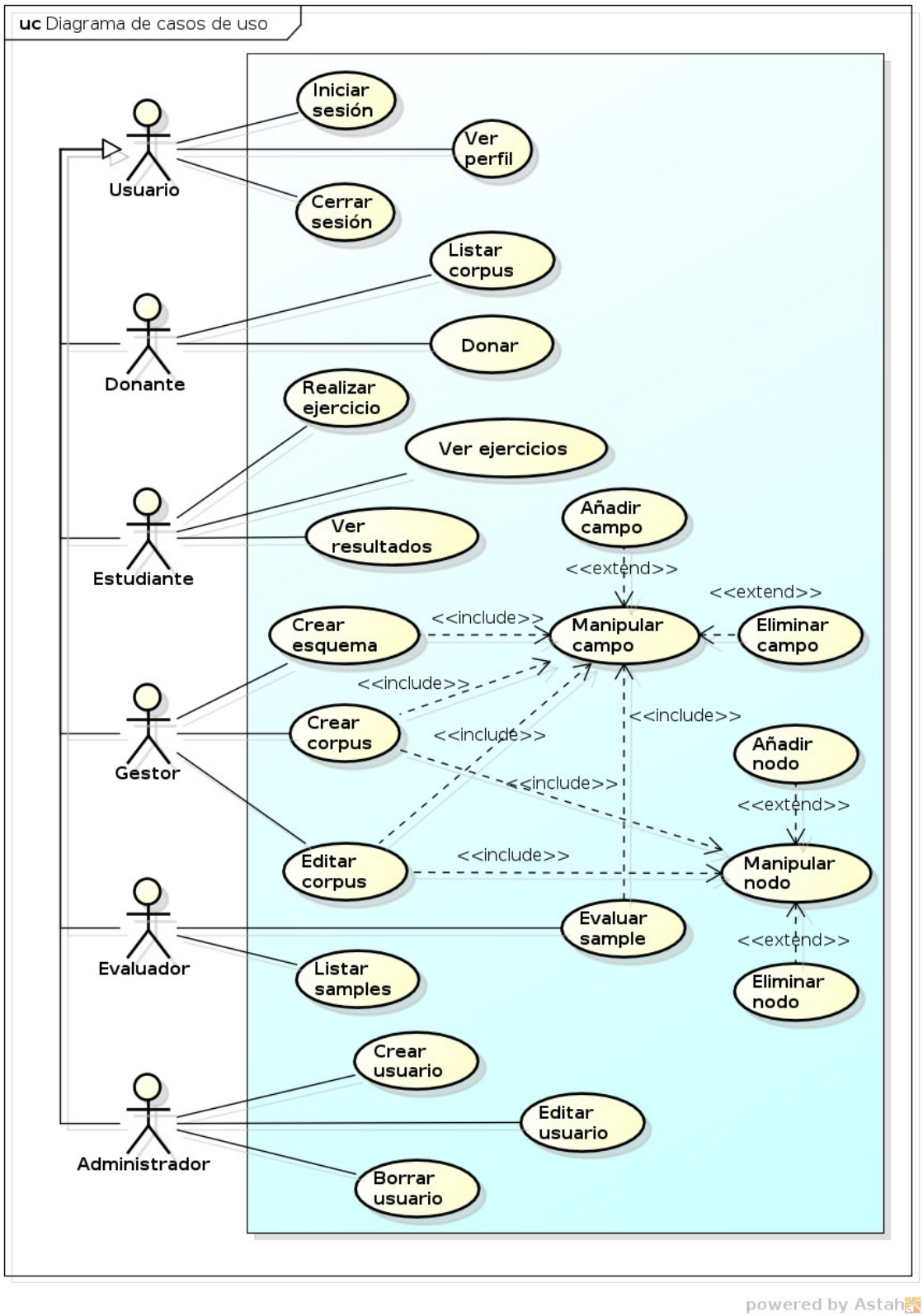


Figura 2: Diagrama de casos de uso

4.8. Especificación de casos de uso

A continuación se describen los diversos casos de uso obtenidos a partir de los requisitos.

UC-01	Iniciar sesión	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario quiera iniciar sesión.	
Cubre	FRQ-02	
Precondición	No exista una previamente una sesión activa.	
Secuencia normal	Paso	Acción
	1	El actor Usuario selecciona la opción de “Iniciar sesión”.
	2	El sistema muestra una ventana emergente para identificarse.
	3	El actor rellena los campos con su usuario y contraseña y pulsa el botón de acceder.
	4	El sistema comprueba las credenciales introducidas por el usuario, crea una sesión y cambia el entorno de la aplicación.
Postcondición	Se crea una sesión válida y se actualiza el entorno de la aplicación.	
Excepciones	Paso	Acción
	4	Si las credenciales son inválidas el sistema lo notifica y, a continuación, este caso de uso continúa en el paso 3.

UC-02	Ver perfil	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario quiera ver su perfil.	
Cubre	FRQ-12	
Precondición	El usuario está identificado.	
Secuencia normal	Paso	Acción
	1	El actor Usuario selecciona la opción de “Ver perfil”.
	2	El sistema recupera los datos del usuario del servidor junto con todas las sesiones activas y expone la información recabada en una nueva ventana.
Postcondición	El sistema muestra al usuario sus datos más actualizados.	

UC-03	Cerrar sesión	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario quiera cerrar sesión.	
Precondición	Exista previamente una sesión activa.	
Secuencia normal	Paso	Acción
	1	El actor Usuario selecciona la opción de “Cerrar sesión”.
	2	El sistema elimina la sesión tanto de la aplicación como del servidor y cambia el entorno de la aplicación.
Postcondición	Se revoca la sesión previa y se actualiza el entorno de la aplicación.	

UC-04	Añadir campo	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario quiera añadir un campo a un nodo o a otro campo.	
Cubre	FRQ-10, FRQ-11	
Secuencia normal	Paso	Acción
	1	El actor Usuario selecciona la opción de “Añadir campo”.
	2	El sistema muestra una ventana con los distintos tipos de campo disponibles.
	3	El actor Usuario selecciona un tipo de campo.
	4	El sistema crea un nuevo campo y lo añade al elemento seleccionado.
Postcondición	Creado un nuevo campo y añadido al elemento seleccionado.	
Excepciones	Paso	Acción
	3	Si el actor Usuario cancela la selección del tipo, el sistema cancela la operación y cierra la ventana de selección. El caso de uso queda sin efecto.

UC-05	Eliminar campo	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario quiera eliminar un campo de un nodo o de otro campo.	
Cubre	FRQ-10, FRQ-11	
Secuencia normal	Paso	Acción
	1	El actor Usuario selecciona la opción de “Eliminar campo” sobre un campo concreto.
	2	El sistema destruye el campo del elemento padre.
Postcondición	El campo seleccionado está eliminado del elemento padre.	

UC-06	Manipular campo	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario quiera añadir o eliminar un campo.	
Dependencias	UC-04, UC-05	
Secuencia normal	Paso	Acción
	1	Si el actor desea eliminar un campo concreto, se realiza el caso de uso *Eliminar campo [UC-05]*.
	2	Si el actor desea añadir un nuevo campo, se realiza el caso de uso *Añadir campo [UC-04]*.

UC-07	Añadir nodo	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario de tipo gestor quiera añadir un nuevo nodo a un nodo concreto.	
Cubre	FRQ-03	
Precondición	El usuario está identificado y posee el rol de gestor.	
Secuencia normal	Paso	Acción
	1	El actor Gestor selecciona la opción de “Añadir nodo”.
	2	El sistema muestra una ventana con los distintos esquemas disponibles y solicita el nombre del nuevo nodo.
	3	El actor Gestor selecciona un esquema y introduce el nombre.
	4	El sistema replica el esquema como un nuevo nodo y lo añade al nodo actual.
Postcondición	Creado un nuevo nodo y añadido al nodo seleccionado.	
Excepciones	Paso	Acción
	3	Si el actor Usuario cancela la selección del esquema, el sistema cancela la operación y cierra la ventana de selección. El caso de uso queda sin efecto.

UC-08	Eliminar nodo	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario de tipo gestor quiera un subnodo de un nodo concreto.	
Cubre	FRQ-03	
Precondición	El usuario está identificado y posee el rol de gestor.	
Secuencia normal	Paso	Acción
	1	El actor Gestor selecciona la opción de “Eliminar nodo”.
	2	El sistema destruye el nodo seleccionado.
Postcondición	El nodo queda completamente eliminado del sistema.	

UC-09	Manipular nodo	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario gestor quiera añadir o eliminar un subnodo de un nodo concreto.	
Dependencias	UC-07, UC-08	
Precondición	El usuario está identificado y posee el rol de gestor.	
Secuencia normal	Paso	Acción
	1	Si el actor desea eliminar un nodo concreto, se realiza el caso de uso *Eliminar nodo [UC-08]*.
	2	Si el actor desea añadir un nuevo nodo, se realiza el caso de uso *Añadir nodo [UC-07]*.

UC-10	Crear esquema	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un gestor quiera crear un esquema.	
Precondición	El usuario está identificado y posee el rol de gestor.	
Secuencia normal	Paso	Acción
	1	El actor Gestor selecciona la opción de “Crear esquema”.
	2	El sistema carga una nueva ventana y solicita el nombre y el tipo al que pertenece el esquema.
	3	El actor Gestor rellena los campos relacionados.
	4	El sistema crea el esquema y habilita la manipulación de campos.
	5	Se realiza el caso de uso *Manipular campo [UC-06]*.
	6	Si el actor Gestor sigue manipulando el esquema, el caso de uso continúa en el paso 5.
	7	El actor Gestor selecciona la opción “Enviar”.
	8	El sistema envía el esquema creado y lo hace persistente en el servidor.
Postcondición	Se crea un nuevo esquema en el sistema.	
Excepciones	Paso	Acción
	3	Si el actor Gestor cancela la introducción de los datos, el sistema cancela la operación y le devuelve a la página principal. El caso de uso queda sin efecto.
	4	Si ya existe un esquema con ese mismo nombre, el sistema notifica al usuario del error producido y, a continuación, este caso de uso continúa en el paso 2.
	5	Si el actor Gestor cancela la creación del esquema, el sistema cancela la operación y le devuelve a la página principal. El caso de uso queda sin efecto.

UC-11	Editar corpus	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un gestor quiera editar un corpus.	
Cubre	FRQ-03	
Precondición	El usuario está identificado y posee el rol de gestor.	
Secuencia normal	Paso	Acción
	1	El actor Gestor selecciona la opción de “Editar corpus”.
	2	El sistema carga el corpus.
	3	Se realiza el caso de uso *Manipular nodo [UC-09]*.
	4	Se realiza el caso de uso *Manipular campo [UC-06]*.
	5	Si el actor sigue manipulando los nodos o los campos, el caso de uso continúa en el paso 3.
	8	El actor Gestor selecciona la opción “Enviar”.
	9	El sistema hace persistente las modificaciones realizas por el actor.
Postcondición	Las modificaciones sobre el corpus quedan realizadas.	
Excepciones	Paso	Acción

	8	Si el actor Gestor cancela la edición, el sistema anulas las modificaciones y le devuelve a la página principal. El caso de uso queda sin efecto.

UC-12	Crear corpus	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un gestor quiera crear un nuevo corpus.	
Cubre	FRQ-04	
Precondición	El usuario está identificado y posee el rol de gestor.	
Secuencia normal	Paso	Acción
	1	El actor Gestor selecciona la opción de “Crear corpus”.
	2	El sistema carga una nueva ventana y solicita el nombre del corpus y el esquema que se desea replicar.
	3	El actor Gestor introduce la información pedida.
	4	El sistema replica el esquema y confecciona un nodo corpus y habilita la edición del corpus.
	5	Se realiza el caso de uso *Manipular nodo [UC-09]*.
	6	Se realiza el caso de uso *Manipular campo [UC-06]*.
	7	Si el actor sigue manipulando los nodos o los campos, el caso de uso continúa en el paso 5.
	8	El actor Gestor selecciona la opción “Enviar”.
	9	El sistema envía el corpus creado y lo hace persistente en el servidor.
Postcondición	Se hace persistente el nuevo corpus.	
Excepciones	Paso	Acción
	3	Si el actor Gestor cancela la introducción de los datos, el sistema cancela la operación y le devuelve a la página principal. El caso de uso queda sin efecto.
	4	Si ya existe un esquema con ese mismo nombre, el sistema notifica al usuario del error producido y, a continuación, este caso de uso continúa en el paso 2.
	8	Si el actor Gestor cancela la creación del corpus, el sistema cancela la operación y le devuelve a la página principal. El caso de uso queda sin efecto.

UC-13	Listar corpus	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un donante quiera ver los corpus disponibles.	
Cubre	FRQ-06	
Precondición	El usuario está identificado y posee el rol de donante.	
Secuencia normal	Paso	Acción
	1	El actor Donante selecciona la opción de “Listar corpus”.
	2	El sistema muestra una lista con todas aquellas sesiones de corpus activas para donar locuciones.
Postcondición	El sistema muestra una lista de las sesiones de corpus disponibles para la donación.	

UC-14	Donar	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un donante quiera donar una locución.	
Cubre	FRQ-06, FRQ-07	
Precondición	El usuario está identificado y posee el rol de donante.	
Secuencia normal	Paso	Acción
	1	El actor Donante selecciona la opción de “Donar” sobre una sesión concreta.
	2	El sistema crea una nueva muestra en el sistema con la información de la sesión del corpus y del donante. Muestra el primer nodo de la sesión.
	3	El actor Donante realiza los ejercicios propuestos en el nodo actual y pulsa al botón siguiente.
	4	El sistema envía los datos al servidor.
	5	El sistema informa del fin de la donación y devuelve al actor Donante a la página principal.
Postcondición	El actor ha donado correctamente las diversas locuciones existentes en el corpus.	
Excepciones	Paso	Acción
	5	Si quedan nodos pendientes en la jerarquía del corpus por realizar, el sistema muestra el nodo y, a continuación, este caso de uso continúa en el paso 3.

UC-15	Ver ejercicios	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un estudiante quiera ver la lista de ejercicios disponibles.	
Cubre	FRQ-09	
Precondición	El usuario está identificado y posee el rol de estudiante.	
Secuencia normal	Paso	Acción
	1	El actor Estudiante selecciona la opción de “Ver ejercicios”.
	2	El sistema muestra una lista con todas aquellas sesiones de corpus activas para realizar ejercicios.
Postcondición	El sistema muestra una lista de las sesiones de corpus disponibles para el estudiante actual.	

UC-16	Realizar ejercicio	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un estudiante quiera realizar un ejercicio.	
Cubre	FRQ-07, FRQ-09	
Precondición	El usuario está identificado y posee el rol de estudiante.	
Secuencia normal	Paso	Acción
	1	El actor Estudiante selecciona la opción de “Realizar ejercicio” sobre una sesión concreta.
	2	El sistema crea una nueva muestra en el sistema con la información de la sesión del corpus y del estudiante. Muestra el primer nodo de la sesión.

	3	El actor Estudiante realiza los ejercicios propuestos en el nodo actual y pulsa al botón siguiente.
	4	El sistema envía los datos al servidor.
	5	El sistema informa del fin del ejercicio y devuelve al actor Estudiante a la página principal.
Postcondición	El actor ha realizado el ejercicio correctamente y las diversas locuciones existentes en el corpus se han almacenado.	
Excepciones	Paso	Acción
	5	Si quedan nodos pendientes en la jerarquía del corpus por realizar, el sistema muestra el nodo y, a continuación, este caso de uso continúa en el paso 3.

UC-17	Ver resultados	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un estudiante quiera ver los resultados de un ejercicio realizado.	
Cubre	FRQ-08	
Precondición	El usuario está identificado y posee el rol de estudiante.	
Secuencia normal	Paso	Acción
	1	El actor Estudiante selecciona la opción de “Ver resultados”.
	2	El sistema muestra una lista con todos los ejercicios realizados y permite la selección sólo de aquellos que tengan al menos una evaluación.
	3	El actor Estudiante selecciona una evaluación concreta de un ejercicio resuelto.
	4	El sistema carga la sesión junto con la información almacenada en la evaluación.
Postcondición	El estudiante ve los resultados de una evaluación concreta.	
Excepciones	Paso	Acción
	3	Si el actor Estudiante cancela la selección, el sistema le devuelve a la página principal y, a continuación, este caso de uso queda sin efecto.

UC-18	Listar samples	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un evaluador quiera ver la lista de muestras disponibles.	
Cubre	FRQ-05	
Precondición	El usuario está identificado y posee el rol de evaluador.	
Secuencia normal	Paso	Acción
	1	El actor Evaluador selecciona la opción de “Listar samples”.
	2	El sistema muestra una lista con todas aquellas muestras disponibles para su evaluación.
Postcondición	El sistema muestra una lista con todas aquellas muestras disponibles para su evaluación.	

UC-19	Evaluar sample	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un evaluador quiera evaluar una muestra concreta.	
Cubre	FRQ-05	
Precondición	El usuario está identificado y posee el rol de evaluador.	
Secuencia normal	Paso	Acción
	1	El actor Evaluador selecciona la opción de “Evaluar” sobre una muestra concreta.
	2	El sistema crea una nueva evaluación en el sistema con la información de la muestra del corpus y del evaluador.
	3	El actor Evaluador evalúa los distintos campos pertinentes de la muestra.
	4	El sistema envía los datos al servidor.
	5	El actor Evaluador finaliza la evaluación pulsando la opción de “Publicar”.
	6	El sistema hace persistente la evaluación y la publica.
Postcondición	El actor ha donado correctamente las diversas locuciones existentes en el corpus.	
Excepciones	Paso	Acción
	5	Si el actor Evaluador pulsa la opción “Eliminar”, el sistema elimina la evaluación del sistema y le devuelve a la página principal y, a continuación, este caso de uso queda sin efecto.
	6	Si el actor Evaluador pulsa la opción “Salir”, el sistema mantiene las evaluaciones realizadas y le devuelve a la página principal y, a continuación, este caso de uso queda sin efecto.

UC-20	Crear usuario	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un administrador quiera crear un nuevo usuario.	
Cubre	FRQ-01	
Precondición	El usuario está identificado y posee el rol de administrador.	
Secuencia normal	Paso	Acción
	1	El actor Administrador selecciona la opción de “Crear nuevo usuario”.
	2	El sistema muestra un formulario con todos los campos relacionados con un usuario.
	3	El actor Administrador rellena todos los campos y selecciona la opción “Crear”.
	4	El sistema crea un nuevo usuario con la información proporcionada.
Postcondición	Creado un nuevo usuario con los datos proporcionados.	
Excepciones	Paso	Acción
	3	Si el actor Administrador cancela la operación, el sistema le devuelve a la página principal y, a continuación, este caso de uso queda sin efecto.
	4	Si el nombre de usuario ya existe en el sistema, el sistema le notifica del error producido y, a continuación, este caso de uso continúa en el paso 2.

UC-21	Editar usuario	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un administrador quiera editar un usuario del sistema.	
Cubre	FRQ-01	
Precondición	El usuario está identificado y posee el rol de administrador.	
Secuencia normal	Paso	Acción
	1	El actor Administrador selecciona la opción de “Editar” sobre un usuario concreto.
	2	El sistema muestra un formulario con todos los campos relacionados con el usuario seleccionado.
	3	El actor Administrador realiza las modificaciones pertinentes y selecciona la opción “Guardar”.
	4	El sistema guarda las modificaciones realizadas.
Postcondición	El usuario queda actualizado tras las modificaciones realizadas.	
Excepciones	Paso	Acción
	4	Si el actor Administrador cancela la edición, el sistema le devuelve a la página principal y, a continuación, este caso de uso queda sin efecto.

UC-22	Borrar usuario	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un administrador quiera eliminar un usuario del sistema.	
Cubre	FRQ-01	
Precondición	El usuario está identificado y posee el rol de administrador.	
Secuencia normal	Paso	Acción
	1	El actor Administrador selecciona la opción de “Borrar” sobre un usuario concreto.
	2	El sistema solicita al actor Administración que confirme la operación de borrado.
	3	El actor Administrador confirma la operación.
	4	El sistema elimina el usuario.
Postcondición	El usuario queda eliminado del sistema.	
Excepciones	Paso	Acción
	3	Si el actor Administrador no confirma la operación, el sistema le devuelve a la página principal y, a continuación, este caso de uso queda sin efecto.

4.9. Modelo de dominio

4.9.1. Introducción

En este apartado se detalla el proceso seguido para obtener las entidades más importantes del problema para finalmente conformar el diagrama de clases del modelo de dominio. Gracias a él se pueden representar las relaciones existentes entre las distintas entidades que componen el dominio del problema a resolver.

4.9.2. Planteamiento inicial

No cabe duda que uno de los objetivos más importantes del sistema es la gestión de corpus. Sin embargo, con este sistema se quiere dar un paso más allá que la simple implementación de un comportamiento concreto; en otras palabras, se quiere desarrollar un sistema capaz de abstraer su comportamiento y poder lidiar con los distintos contextos que se le proporcionen.

Un corpus es un conjunto de datos y materiales respecto a una determinada materia. Se podría modelar un corpus atendiendo únicamente al lingüístico en el que nos encontramos actualmente, incluso los datos que éste almacena podrían estar predefinidos y ser inmutables.

Sin embargo, se concluyó que era preferible que el sistema fuese configurable y adaptable para poder amoldarse a cualquier contexto, de esta forma se cubrían varios tipos de gestión de corpus y sobretodo cualquier tipo de información o dato almacenado en ellos.

Se podría decir por tanto, que en ciertas partes del sistema, ya no solo se está modelando el problema, sino se está definiendo una especie de metamodelo que modela metadatos relacionados con el problema, esto también permite que cualquier persona ajena a la implementación pueda modificar el comportamiento sin tener que modificar el código.

La abstracción se realizó concretamente a la hora de modelar un corpus y sobretodo al modelar los campos asociados a este.

Respecto al modelado del corpus – aunque posteriormente se vio que era posible haberlo generalizado aún más y haber llegado a modelar cualquier tipo de corpus – se prefirió por criterios de simplicidad limitarlo al contexto existente. Es decir, un corpus podría haber sido un conjunto indeterminado de nodos que tuvieran la suficiente información como para permitir definir el comportamiento y jerarquía de éste. Sin embargo, en este caso, se limitó voluntariamente la jerarquía, impidiendo así poder modificarla conceptualmente sin tener que modificar el comportamiento implementado. Esta decisión aunque simplifica su gestión, limita la libertad de definición y elaboración de un corpus.

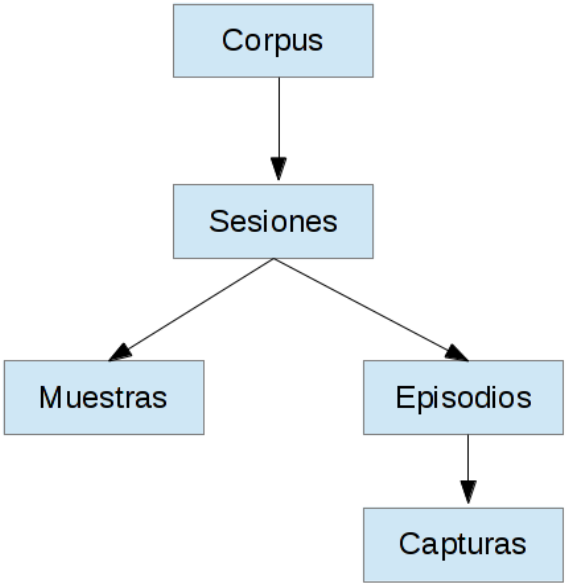


Figura 3: Jerarquía del corpus

La jerarquía elegida fue la siguiente:

- Todo corpus tiene un conjunto indeterminado de sesiones.
- Cada sesión tiene un conjunto indeterminado de muestras relacionadas con los episodios que componen la sesión. Cada sesión por tanto, tiene un número indeterminado de episodios.
- Cada episodio tiene un conjunto indeterminado de capturas.
- Todos los nodos que componen la jerarquía están a su vez compuestos por un número indeterminado de campos de información.

Sin embargo, donde se consiguió una abstracción completa fue a la hora de modelar los campos. Se podría haber definido un campo con un dato, y dependiendo del contexto donde se emplease representaría una u otra información. La solución fue definir un campo como un dato asociado a un conjunto de metadatos, además de tener la posibilidad de estar compuesto por otros campos. Dicha solución permite representar cualquier tipo de dato, ya sea éste una simple cadena, el nombre de un usuario o un árbol.

Todo lo anteriormente expuesto junto con los requisitos iniciales motivó a confeccionar un abanico de entidades necesarias para el correcto modelado del problema a resolver. A continuación se describen:

- **Action**: identificada unívocamente por su nombre, representa la capacidad de un usuario de realizar una tarea o acción determinada en el sistema. Por tanto, el conjunto de tareas posibles del sistema estará limitado y controlado por las instancias de esta clase. De esta forma se consigue un sistema de política de privilegios aumentando así la seguridad y control de la aplicación.
- **Role**: determina el rol o alcance de acción de un usuario. En realidad se comporta como un lote de acciones.
- **User**: entidad que representa a un usuario en el sistema. Contiene toda la información pertinente de una persona física que quiera emplear el sistema.
- **Schema**: representa un esquema de un corpus, sesión, iteración o captura. Actúa como un template de cómo está estructurado la clase que esquematiza.
- **Field**: es un campo que posee un valor y un conjunto de metadatos que especifican su tipo y comportamiento. Es una de las clases más importantes de la aplicación. Con ella se puede construir cualquier tipo de campo, incluso campos compuestos.
- **FieldMetada**: conjunto de valores que definen los metadatos de un *Field*. Confiere el comportamiento al campo e indica cómo se debe tratar.
- **CorpusNode**: clase abstracta para simplificar la compresión del diagrama y permitir la

generalización de los nodos. Tanto el nodo padre – corpus – como sus hijos – sesiones, episodios y capturas – son nodos de un árbol jerarquizado. Sin embargo, las muestras aunque pendan del nodos de las sesiones, no son un nodo real.

Los nodos son agregaciones también de campos (*Fields*) y son los únicos capaces de añadir información a éstos. Pueden almacenar cualquier tipo de campo, incluso pueden tener campos compuestos.

- **Corpus**: es una entidad que representa un corpus. La estructura de un corpus es de tipo árbol, siendo éste el nodo padre y del cual penden el resto de nodos – sesiones, iteraciones y capturas.
- **CorpusSession**: determina un periodo o sesión de toma de muestras. Está compuesto por un conjunto de episodios y el conjunto de muestras correspondientes a la sesión.
- **CorpusChapter**: representa una iteración o episodio, es decir, un conjunto de capturas con un procedimiento concreto y especificado en dicho nodo.
- **CorpusCapture**: es el último nodo (nodo hoja) y contiene el ejercicio que debe desarrollar el usuario, en otras palabras, es el nodo donde se obtiene la locución del estudiante siguiendo las sentencias definidas en la captura – o recursos multimedia.
- **CorpusSample**: corresponde a una muestra determinada de una sesión concreta de un corpus y realizada por un estudiante o un donante.
- **Evaluation**: entidad que representa una evaluación realizada por un usuario evaluador sobre una muestra concreta.

4.9.3. Diagrama de clases

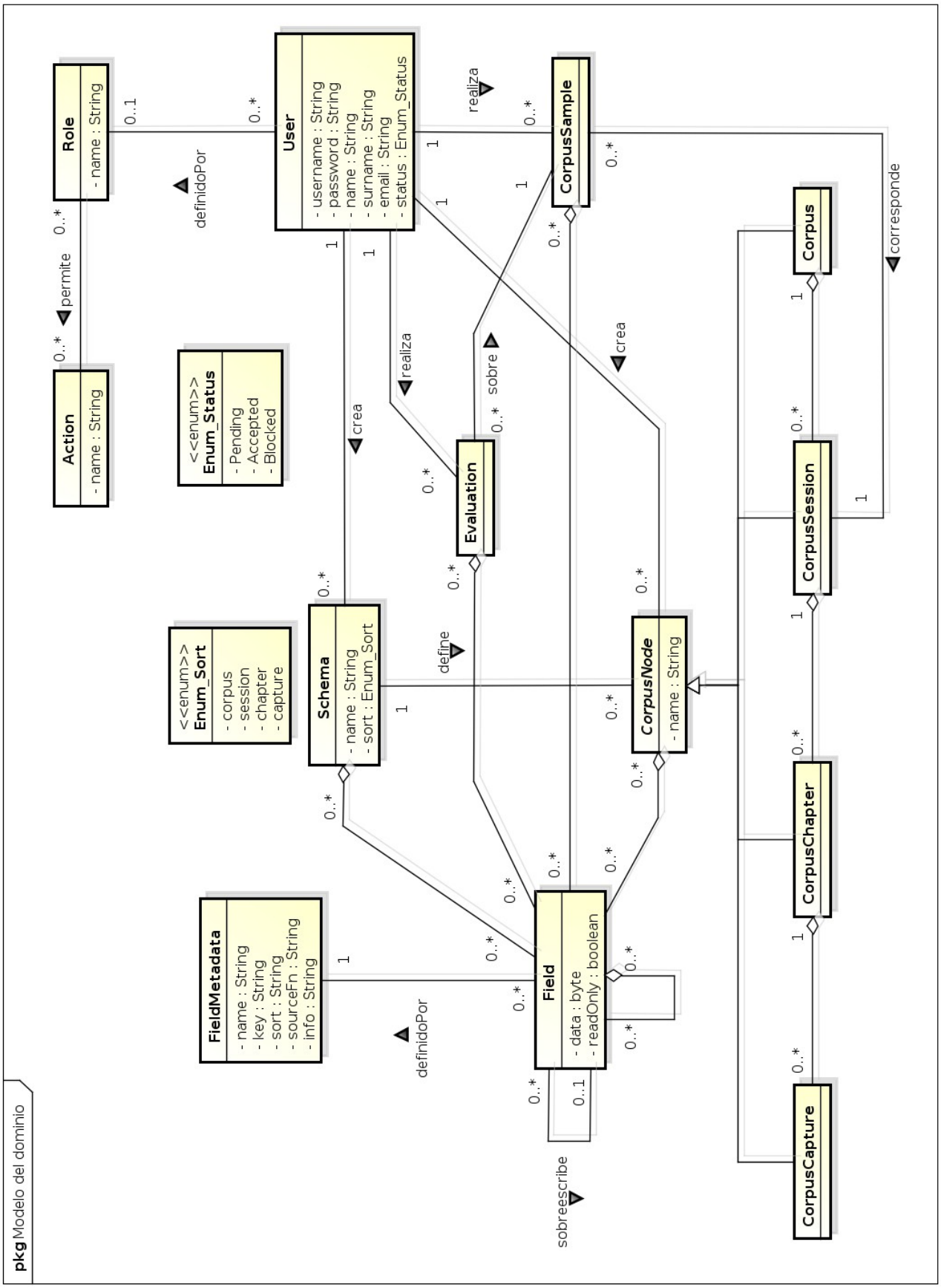


Figura 4: Modelo del dominio

4.10. Diagramas de secuencia de análisis

En este apartado se muestran los distintos diagramas de secuencia relacionados con los casos de uso descritos anteriormente. Se caracterizan por estar muy detallados ya que pertenecen a la fase de análisis.

4.10.1. UC – 1: Iniciar sesión

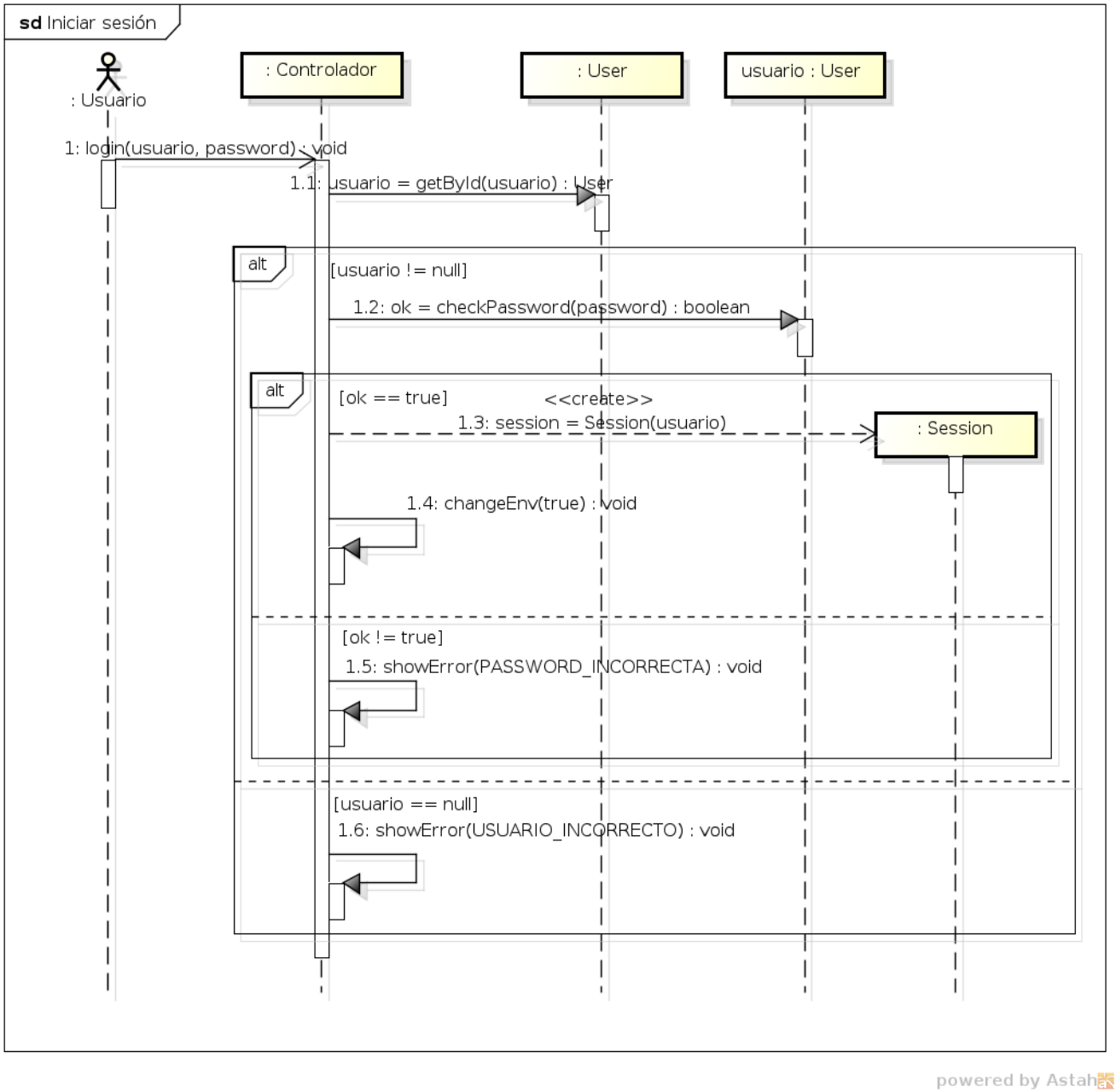


Figura 5: UC – 1, Iniciar sesión

4.10.2. UC – 2: Ver perfil

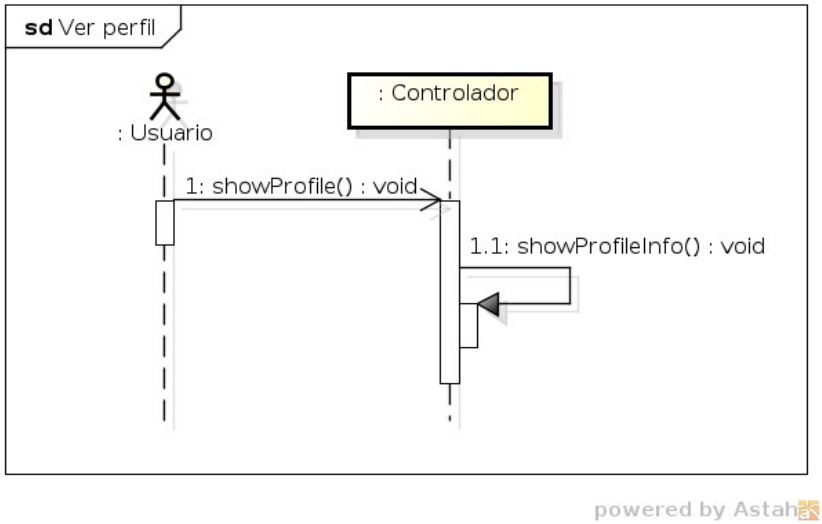


Figura 6: UC – 2, Ver perfil

4.10.3. UC – 3: Cerrar sesión

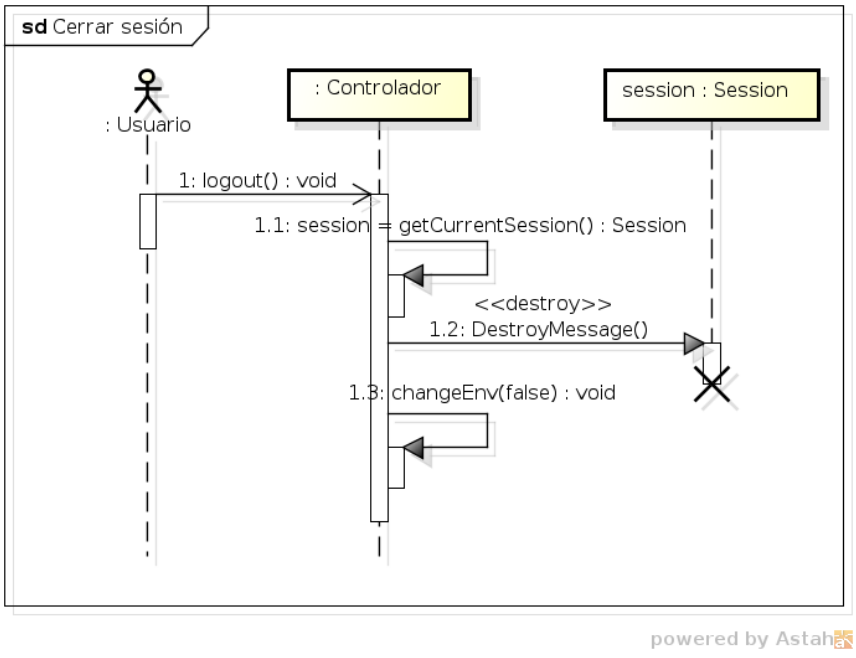


Figura 7: UC – 3, Cerrar sesión

4.10.4. UC – 4: Añadir campo

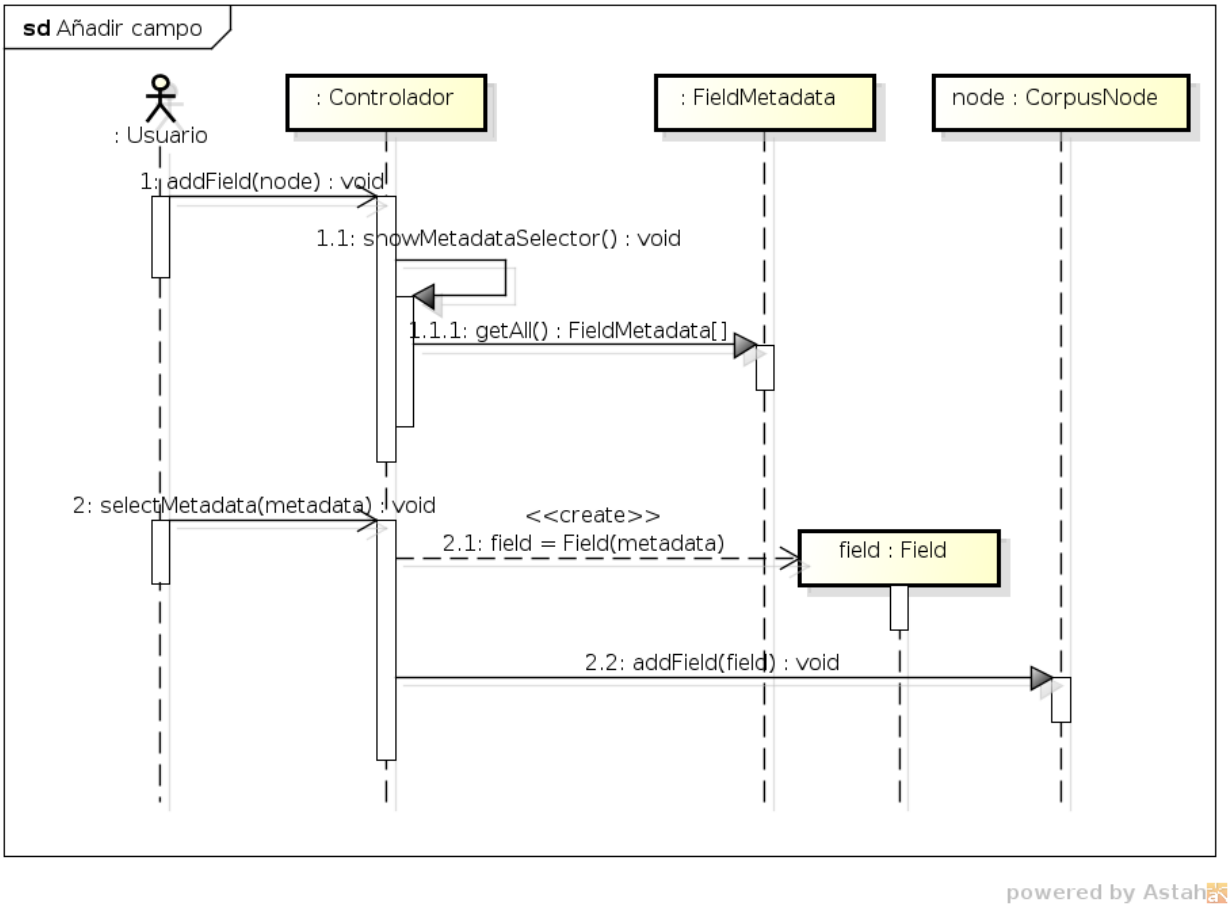


Figura 8: UC – 4, Añadir campo

4.10.5. UC – 5: Eliminar campo

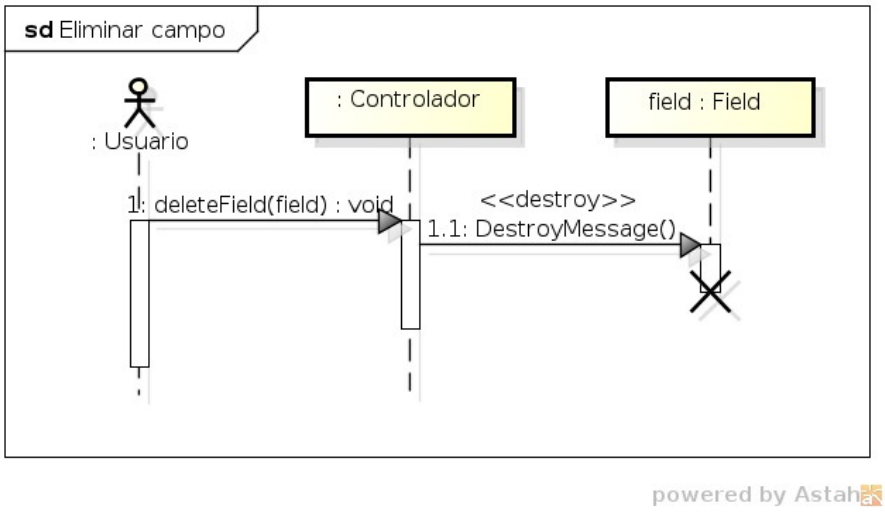


Figura 9: UC – 5, Eliminar campo

4.10.6. UC – 7: Añadir nodo

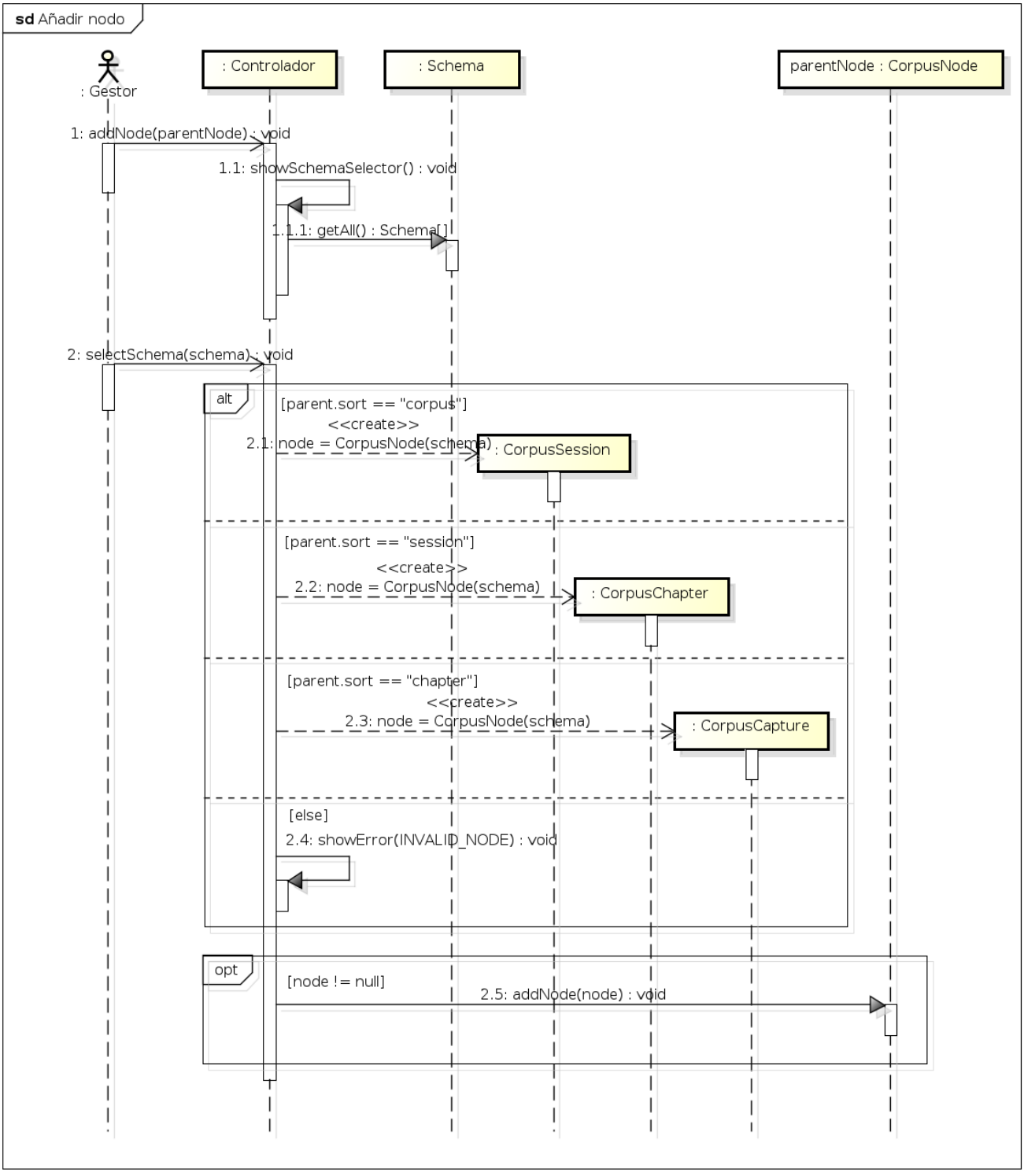


Figura 10: UC – 7, Añadir nodo

4.10.7. UC – 8: Eliminar nodo

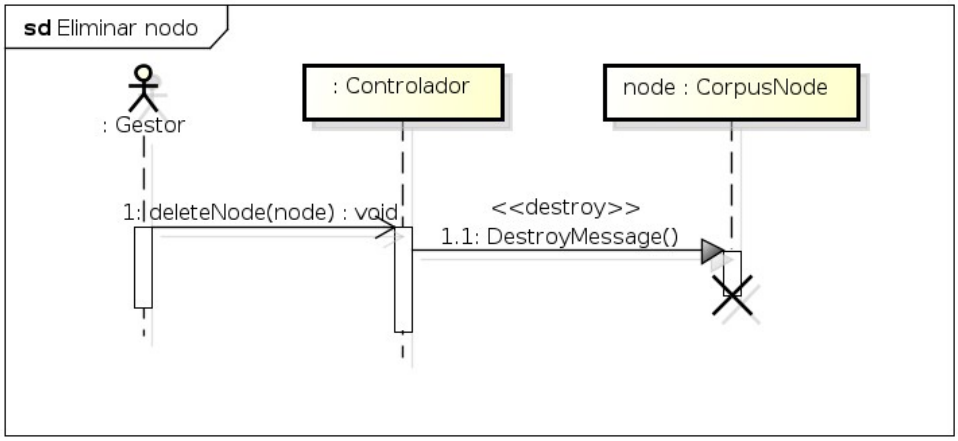


Figura 11: UC – 8, Eliminar nodo

4.10.8. UC – 13: Listar corpus

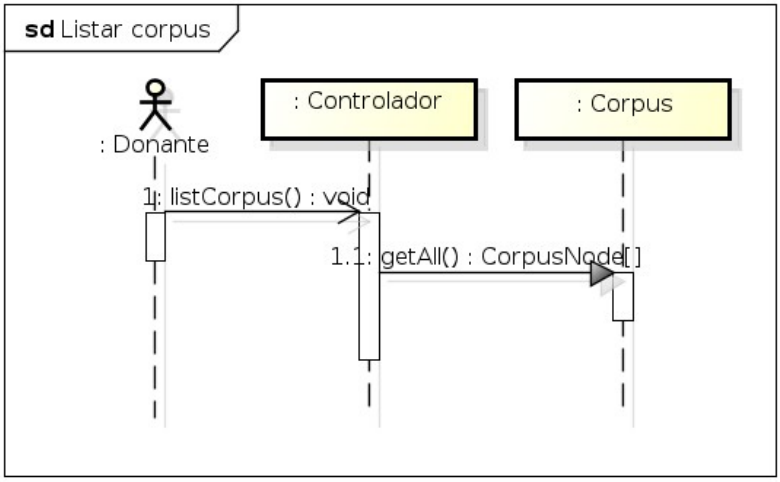


Figura 12: UC – 13, Listar corpus

4.10.9. UC – 10: Crear esquema

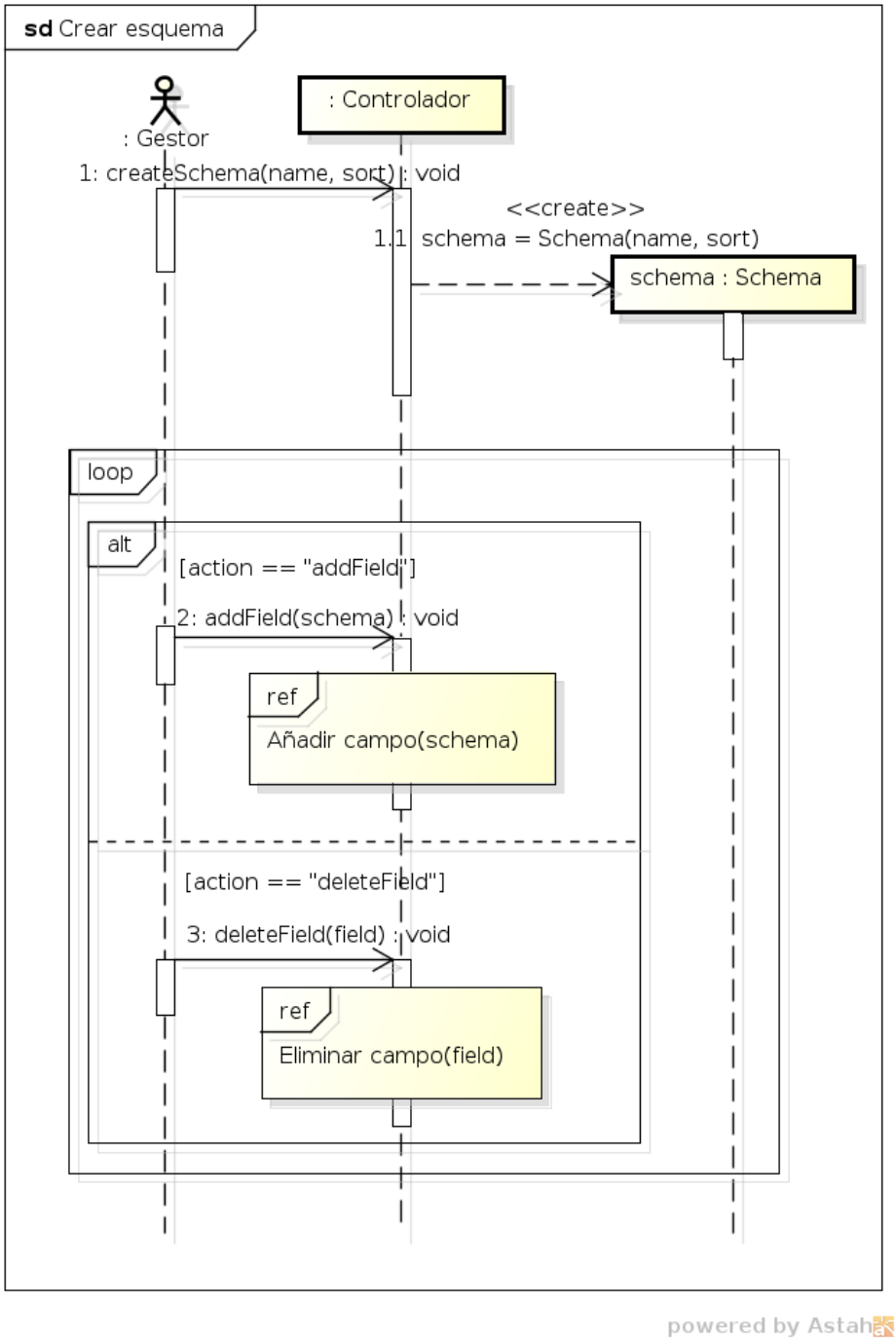


Figura 13: UC – 10, Crear esquema

4.10.10. UC – 11: Editar corpus

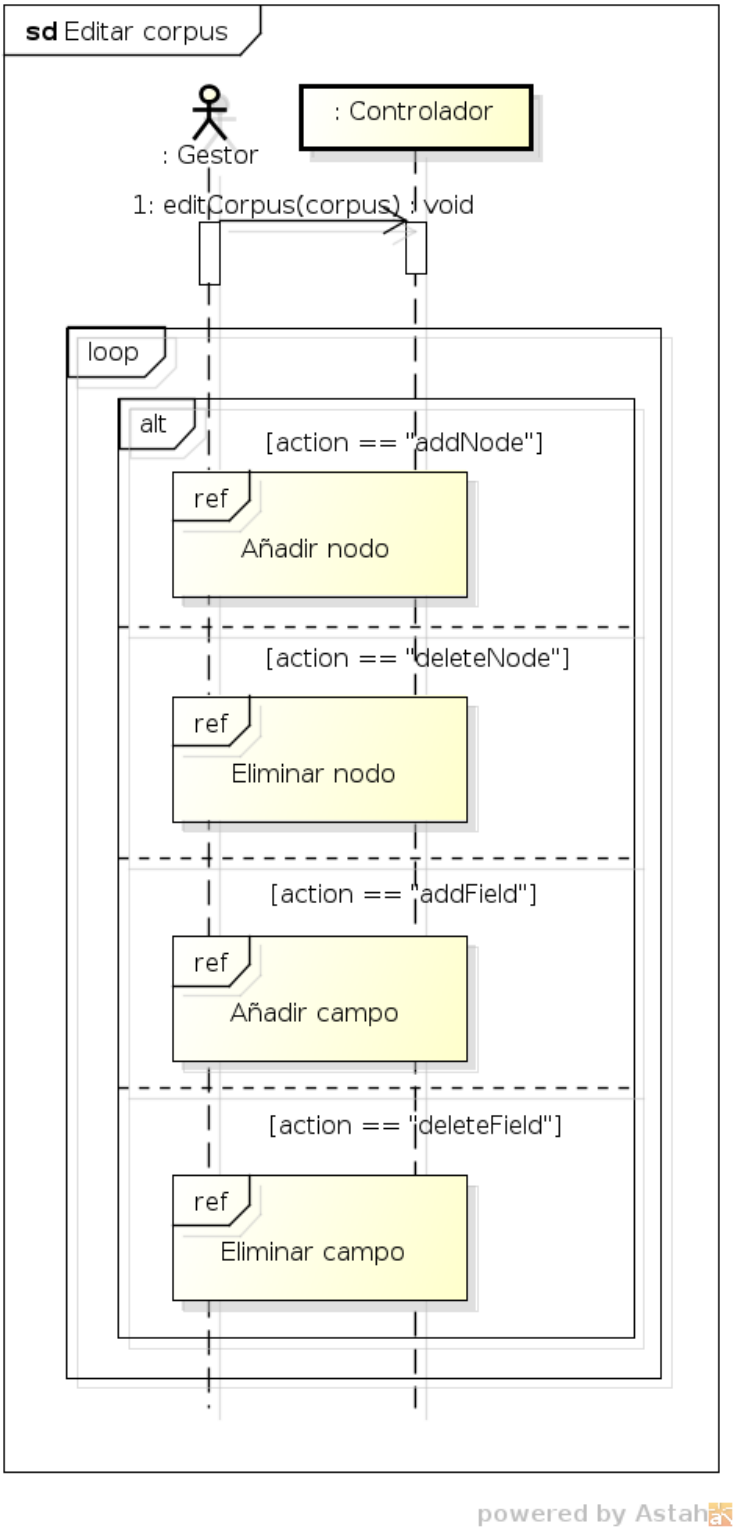


Figura 14: UC – 11, Editar corpus

4.10.11. UC – 12: Crear corpus

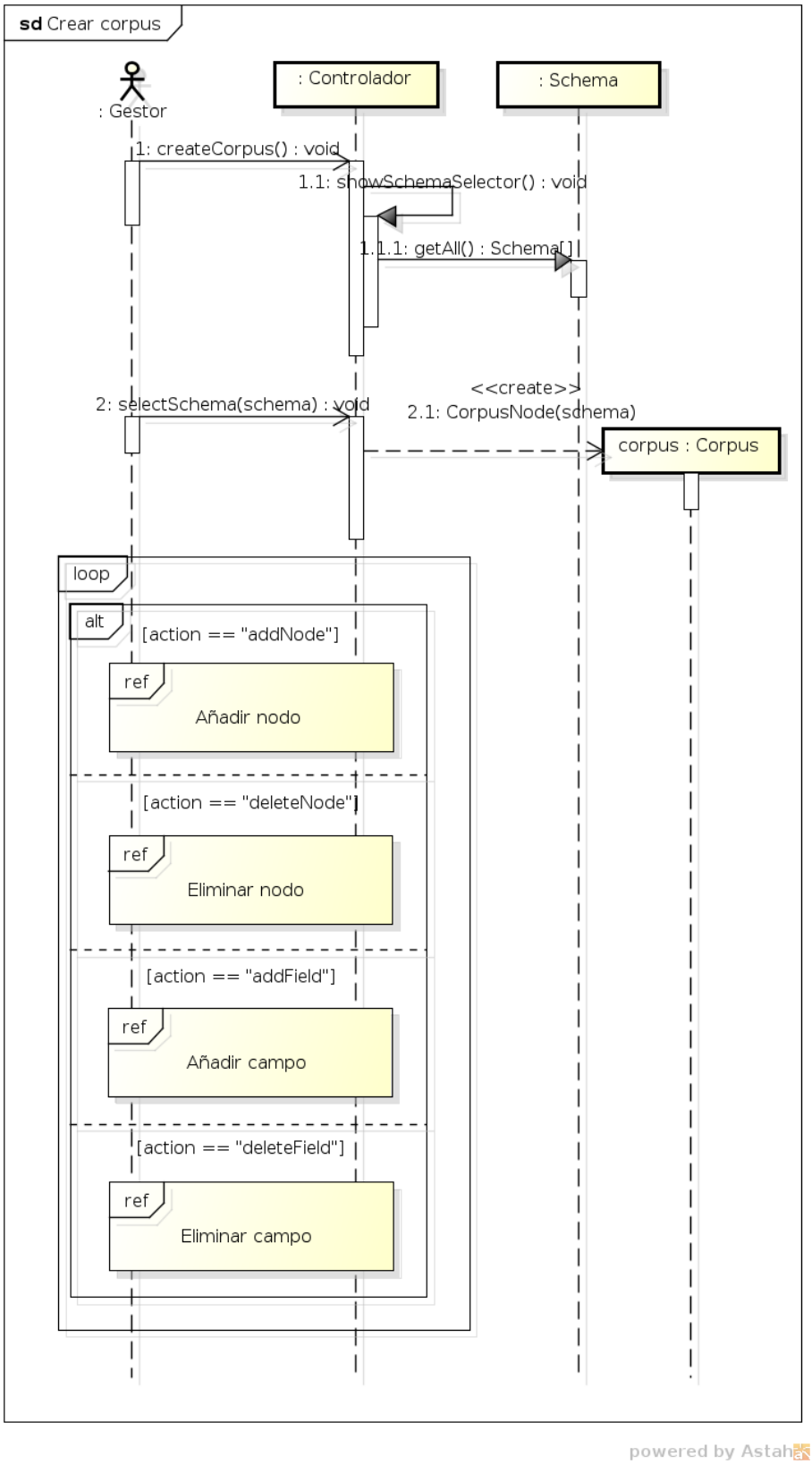


Figura 15: UC – 12, Crear corpus

4.10.12. UC – 14: Donar

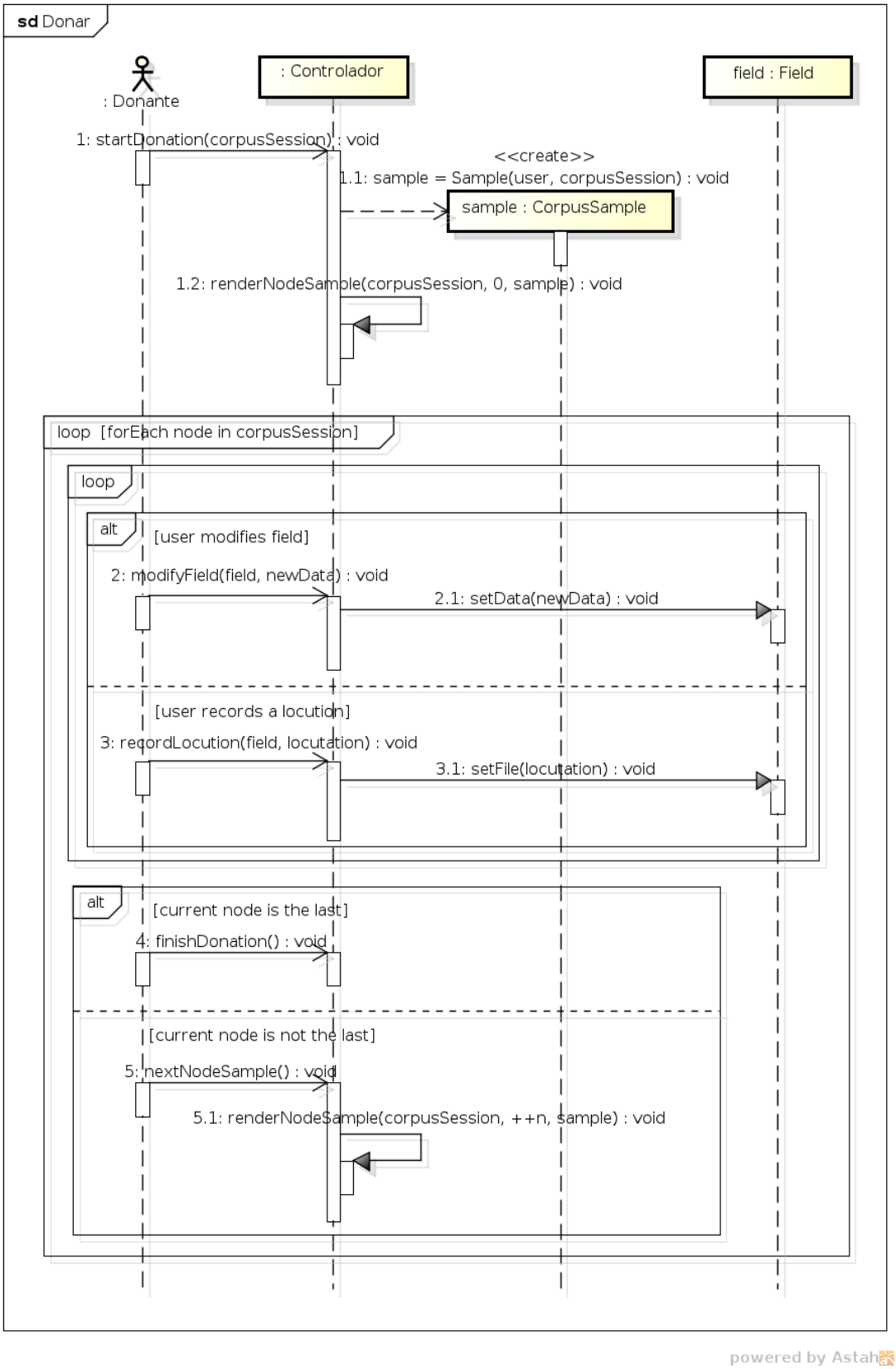


Figura 16: UC – 14, Donar

4.10.13. UC – 15: Ver ejercicios

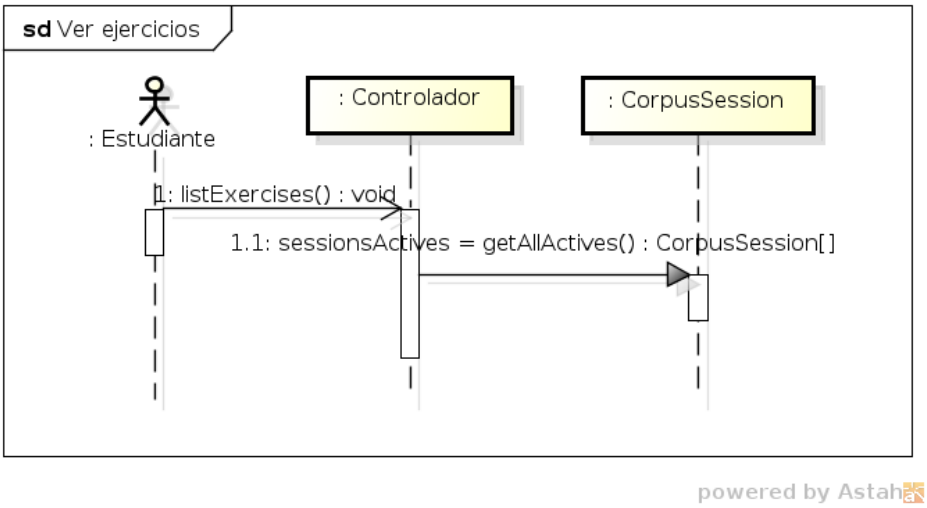


Figura 17: UC – 15, Ver ejercicios

4.10.14. UC – 16: Realizar ejercicio

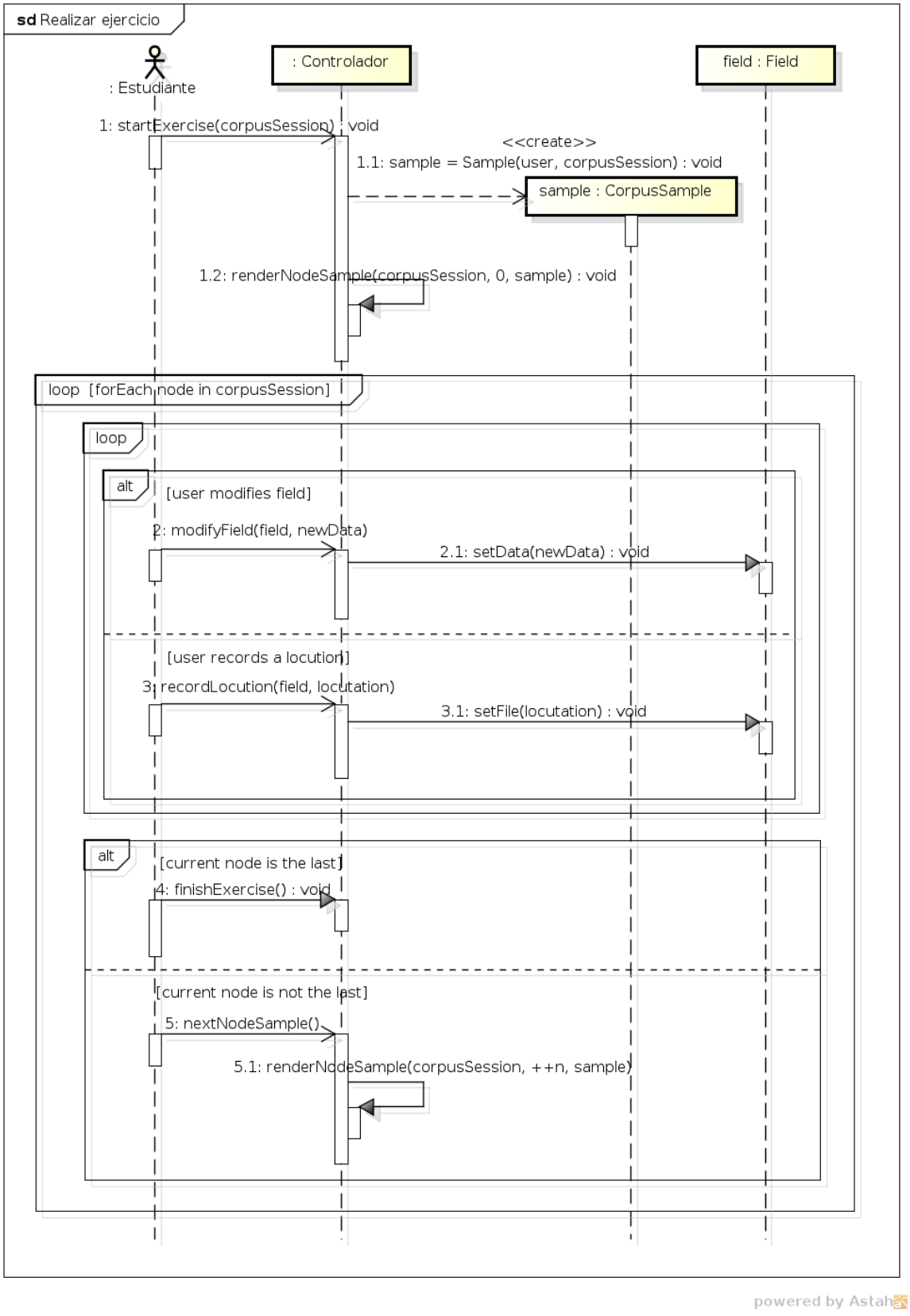


Figura 18: UC – 16, Realizar ejercicio

Capítulo 5

DISEÑO

5.1. Introducción

5.1.1. Propósito

La fase de diseño de software es un proceso que emplea determinados principios y técnicas para detallar y definir un sistema con el suficiente detalle como para permitir su implementación.

Es la continuación del resultado obtenido durante la etapa de análisis, en la cual se planteó qué debería hacer el sistema y con qué recursos. La etapa de diseño por tanto detalla el cómo satisface las anteriores especificaciones.

5.1.2. Alcance

Determinar una arquitectura del sistema que permita continuar con el proceso de desarrollo del software, concretamente la fase de implementación, además de resolver todos y cada uno de los requisitos que están relacionados con el uso de la tecnología propuesta, interfaces de usuario, comunicación entre los componentes...

5.2. Solución propuesta

Ante la naturaleza de los requisitos obtenidos durante la etapa de análisis, la arquitectura que mejor casa es la de cliente-servidor. La principal motivación de este planteamiento es la necesidad de desarrollar una aplicación web que realice operaciones sobre información almacenada en una base de datos centralizada.

Esto conlleva a que se tenga que conformar dos partes de desarrollo paralelas con un objetivo común, cubrir los requisitos del sistema. Por ello, la solución propuesta se describirá desde dos puntos, el del cliente y el del servidor.

5.2.1. Cliente

Por tanto, la primera consideración es pensar en la tecnología relacionada con el cliente, cómo debe ser abordada y en cómo influye en el servidor. Las siguientes características son inherentes a este tipo de tecnología:

- La interfaz gráfica emplea un lenguaje de maquetación denominado HTML que permite definir la estructura de la página web, por él mismo no brinda la posibilidad de programar el comportamiento de la página.
- El protocolo de comunicación entre el cliente y el servidor de la Web es HTTP. A diferencia de otros protocolos, HTTP es por naturaleza sin estado; es decir, las transacciones entre diversos nodos de cómputo se realizan de forma atómica e independientemente de la secuencia anterior.

Además las especificaciones recopiladas en la fase de análisis obliga que el contenido de las páginas generadas sea dinámico, lo que conlleva a pensar en un lenguaje de programación capaz de cubrir esta característica. Las páginas webs admiten diversos lenguajes de scripting, sin embargo, JavaScript se ha perfilado por mérito propio como el lenguaje de programación por defecto de la web, ya que ha excedido con creces las expectativas iniciales que se tenían de él. Esto ha llevado a que surja una nueva manera

de conformar el contenido dinámico de una página, concretamente dos vertientes que en ocasiones pueden conjugarse:

- Páginas web de **contenido dinámico** (Dynamic web page): la maquetación de la página y la manipulación de su información se delega exclusivamente al servidor. El cliente realiza una petición HTTP al servidor y éste, dependiendo del contexto (y sesión), ensambla una determinada página web y la devuelve como recurso de contenido fijo al cliente. En esta vertiente el recurso y servicio están fusionados. En cada momento que se desee cambiar el contenido de la página se debe realizar el proceso descrito anteriormente y volver a pedir un recurso nuevo.
- HTML dinámico (**DHTML**): nueva vertiente cuya explosión ha tenido lugar hace un par de años con la estandarización de HTML5 y por la gran adopción y uso de JavaScript como lenguaje de programación de la Web. El cambio del contenido así como la estructura de la página web está en su mayoría controlado por scripts desarrollados en este lenguaje de scripting. Gracias a él se puede acceder y manipular fácilmente el DOM (Modelo de Objetos del Documento, en español). El grado de modificación es mayor que en la anterior vertiente y una vez cargada la página, solo sería necesario descargar pequeñas partes de información, por ejemplo mediante AJAX, lo que reduce considerablemente el tráfico de datos.

Actualmente DHTML se está imponiendo sobre las páginas web de contenido dinámico. Grandes compañías como Google (con Gmail como bandera), Facebook, Twitter... han desarrollado gran parte de su espacio web empleando DHTML. Los beneficios de emplearlo son numerosos, sin embargo, el contenido dinámico aún así posee ciertas características exclusivas.

Un problema que podría surgir al emplear DHTML es que una parte importante de la lógica de negocio se realiza en la parte del cliente. Esto genera clientes pesados y puede conllevar a problemas de eficiencia debido a que ciertos clientes podrían no tener cómputo suficiente para realizar las operaciones requeridas (por ejemplo *smartphones)*. Además, al contar con tecnologías recientes, y conociendo la problemática existente respecto a los navegadores, es posible que su comportamiento sea distinto e incluso errático. Aunque la mayor desventaja es que complica la depuración del código y la obtención de errores debido a que el servidor es completamente ajeno a cómo se ejecuta el código en la parte del cliente (salvo que se desarrollen mecanismos apropiados).

Cabe también destacar que mediante el uso exclusivo de contenido dinámico, los mecanismos de seguridad son menores que con DHTML. Esto se debe principalmente a que el servidor es la única entidad que tiene acceso al código y la única que lo ejecuta. El cliente solamente renderiza los recursos confeccionados por el servidor. Gracias a ello, desde el servidor se puede implementar medidas de seguridad, como comprobación de origen de datos, uso de nonce para comprobar la integridad y veracidad de los datos, etc...

En cambio, con DHTML, el cliente conoce todo el código ya que es necesario hacerlo público para que pueda ser ejecutado localmente mediante el uso de JavaScript. Esto tiene varias implicaciones respecto a la seguridad. El cliente, puede voluntariamente modificar el script local e incluso comunicarse con el servidor de forma interesada e ilícita. Esto da lugar a diversos riesgos de seguridad como puede ser c*ross-site scripting* y *clickjacking*. Es obligación por tanto del servidor implementar los mecanismos necesarios para mitigar y eliminar tales vulnerabilidades. Nunca se debería confiar en los clientes.

Finalmente se optó por el uso exclusivo de DHTML en la parte del cliente. Más concretamente empleando la técnica de *Single-page application* (SPA). Consiste en que toda la lógica reside en una

única página web y no necesita recargarse. Es una perspectiva más cercana a una aplicación de escritorio ya que la navegación y actualización de la información es fluida al no existir cargas entre páginas. Esta técnica es posible en su mayor parte gracias a HTML5, el cual permite un control más afinado del DOM y las posibilidades que éste ofrece.

La principal motivación de elegir esta opción fue la facilidad de, una vez realizada la aplicación, empaquetarla en contenedores para ser empleada en otros entornos, como por ejemplo:

- Empaquetar la aplicación web en una extensión de un navegador web. De esta forma no sería necesario pedir constantemente los recursos necesarios al servidor – ya que estaría en el cliente – sino únicamente los servicios que éste proporcione a la aplicación web.
- Convertirla en una aplicación Android o iOS mediante el uso de *Apache Cordova*. Se ejecuta el contenido de la aplicación web en el navegador nativo y permite tener acceso al dispositivo mediante una API – que proporciona a través del DOM – y que puede ser empleada mediante JavaScript.
- Empaquetarla en una aplicación web del sistema operativo, como por ejemplo Windows8 y FirefoxOS.

Sin embargo, la confección de una SPA no es una tarea fácil y más si es una aplicación web con cierta complejidad. Es prácticamente imprescindible emplear un framework de JavaScript que proporcione un conjunto de herramientas y APIs para hacer este proceso lo más sencillo posible. Una de las ventajas que poseen además es que abstraen el comportamiento peculiar de cada navegador (surgido por la pasada lucha de navegadores). Los frameworks más importantes para desarrollo de SPA son:

- Meteor.js: su peculiaridad es que no es sólo un framework JavaScript, sino que es un *full-stack*, es decir, proporciona herramientas en todas las partes del desarrollo, ya sea en la parte del cliente, servidor y sistema de datos. Tiene un sistema de comunicación entre el cliente y servidor propio, basado en el esquema publicador-subscriptor y permite una mayor eficiencia en la actualización de los datos consiguiendo incluso hacerlo en tiempo real (muy baja latencia). En la parte del servidor emplea NodeJS, que simplemente es la adaptación del motor de JavaScript V8 del navegador Google Chrome para que sea posible ejecutar JavaScript de forma nativa. Como sistema de base de datos emplea MongoDB, un sistema NoSQL basado en documentos, concretamente en BSON – cuyas siglas son Binary JSON y son representaciones binarias de estructuras JSON, las cuales son más eficientes tanto en almacenamiento como en velocidad. Como se puede apreciar, gracias a Meteor se consigue que exista un único lenguaje de programación común a todas las partes.
- Ember.js: evolución de SproutCore 1.0, es un framework JavaScript de la parte del cliente que emplea el patrón MVC. Aspira a ser un framework que hereda las mejores prácticas de cada entorno y cuyo mantra es “convención sobre configuración” – heredado de su creador que es miembro de jQuery y por supuesto de Ruby On Rails. Permite el bindeado de doble dirección de datos y proporciona un sistema de rutas y de componentes.
- AngularJS: creado y mantenido por Google, se ha perfilado como un referente a la hora de construir aplicaciones de única página. Sigue el patrón MVC al igual que otras alternativas y permite el bindeado bidireccional de datos. La peculiaridad de este framework es que está muy focalizado también en facilitar y fomentar el desarrollo de pruebas del código a la par que se desarrolla la aplicación.

- ExtJS: es un framework para crear aplicaciones web interactivas de una única página empleando para ello técnicas de peticiones de tipo Ajax, manipulación del DOM y uso de DHTML. Además ofrece un gran número de componentes (*widgets*) que poseen su propia lógica y pueden ser ampliamente configurados tanto en lo visual como en su comportamiento ante distintos eventos provocados por el usuario o el sistema.

  Otra ventaja muy importante es su capacidad de convergencia, es decir, su habilidad intrínseca de adaptarse a cualquier tipo de dispositivo, ya sea de escritorio como móvil, o entre distintos navegadores web. Consiguiendo así un código más limpio y ocultando toda la lógica asociada a mejorar el soporte.

  También consigue de una forma muy lograda una programación orientada a objetos, obteniendo así gran parte de las ventajas que este tipo de paradigma proporciona.

  A partir de la versión Ext 2.1 se publica el código mediante dos posibles licencias, GPLv3 y una comercial. Sigue también el patrón MVC aunque a partir de la versión 5.0 también permite el patrón MVVM y el bindeado bidireccional.

Como se puede apreciar, en muchos aspectos, los frameworks anteriormente mencionados son muy parecidos. Lo que motiva a que también se tenga en cuenta la forma de hacer de cada uno de ellos. Por esa razón, el alumno escogió emplear ExtJS ya que durante su instancia en una empresa privada, durante las prácticas en empresa, estuvo desarrollando pequeños componentes empleando dicho framework.

5.2.2. Servidor

Una vez definida las tecnologías y arquitecturas empleadas en el cliente es necesario especificar aquellas relacionadas con el servidor.

Debido al tipo de cliente que se ha optado en el apartado anterior, la lógica de negocio se ha desplazado considerablemente al lado del cliente. Por tanto podemos llegar a decir que ya no es necesario crear un servidor web sino un servicio web que proporcione un conjunto de servicios a través de una API bien conocida. A este tipo de arquitectura se le denomina como “servidor delgado”, en contraposición de la terminología “cliente pesado”.

A la hora de elegir el estilo de la arquitectura software del servicio se eligió REST (en español, transferencia de estado representacional) frente a estilos como CORBA o SOAP.

Los servidores REST proporcionan un conjunto de recursos (definidos como sustantivos) perfectamente identificados empleando para ello una única URL. Estos recursos proporcionan información relacionada con la operación que se haya realizado sobre él. Las peticiones HTTP poseen un tipo de operación (o “verbo”) asociada. Los verbos más comunes son POST, GET, PUT y DELETE y normalmente se suelen equiparar con las operaciones CRUD de las bases de datos (create, retrieve, update y delete respectivamente). Es por esta razón, que se puede acceder directamente a un conjunto de datos realizando una determinada petición HTTP sobre un recurso conocido.

Las ventajas que ofrece este tipo de servidores son las siguientes:

- Rendimiento: se emplea las cabeceras de las peticiones HTTP para especificar la forma de acceso a un recurso. Incluso la respuesta puede venir codificada en JSON, lo que disminuye con creces el payload.

- API más sencillas: la especificación de un conjunto de recursos con sus operaciones (o verbos) permitidas facilitan definir y publicar una API sencilla y accesible desde cualquier tipo de aplicación que emplee HTTP.
- Permite cambiar fácilmente el comportamiento de un verbo de un recurso concreto.
- Al basarse en una comunicación sin estado, está exenta de problemas en el mantenimiento de un canal. Esto es, cortes en la comunicación, latencias elevadas... Todas las operaciones se realizan de forma atómica.
- Permite una gran escalabilidad de los recursos. Ya que todos los recursos poseen una URI concreta, se puede levantar diversas instancias del servicio en distintos servidores y enlazarlos mediante un proxy de balanceo por ejemplo. De esta forma el acceso a los recursos por parte de los clientes es transparente.

Uno de los mejores candidatos para realizar esta tarea es Ruby On Rails (RoR). De forma nativa ejecuta un servidor RESTful al que se le puede definir un conjunto de rutas para especificar la ubicación del recurso y sus métodos asociados.

Además, posee un soberbio mecanismo de consulta a la información almacenada en la base de datos. Emplea el patrón Active Record, es decir, cuando se especifica una entidad de un modelo junto con sus campos y relaciones, automáticamente genera las tablas pertinentes y los métodos de acceso y manipulación de los datos (métodos accesors y mutators).

A su vez, los controladores pueden ser empleados perfectamente como gestores de las operaciones sobre los recursos. Siendo éstos las entidades del modelo.

Por ejemplo, realizando la siguiente petición HTTP:

GET http://host.com/users/1

Se obtendría la información relacionada con el usuario con identificador único 1. Lo positivo de las rutas es que se pueden encadenar y así obtener operaciones más complejas. Por ejemplo:

GET http://host.com/users/1/sessions/

En esta petición se obtendrían todas las sesiones existentes del usuario con identificador único 1.

Como se ve, existe una gran libertad y sencillez de conformar nuevas operaciones.

Sin embargo, no todas las operaciones deben proporcionar la información solicitada por una petición HTTP. Es posible que exista la necesidad de limitar el acceso a determinados tipos de información o de recursos.

Para ello, se puede emplear fácilmente la cabecera *Authentication* de las peticiones HTTP. Aquellas rutas que necesiten una autenticación especial sólo mostrarán la información solicitada si el valor de dicha cabecera es válido.

5.3. Persistencia

En este apartado se detallará toda la información relacionada con el modelado de datos, es decir, como se estructuran y almacenan las entidades del modelo en una base de datos. Primero se comenzará con el diagrama entidad-relación, el cual permite representar visualmente las entidades del dominio así como sus asociaciones y atributos. Posteriormente se expondrá el esquema relacional obtenido tras la resolución de asociaciones complejas del anterior diagrama. Gracias a él se refleja el esquema que tendrá la base de datos.

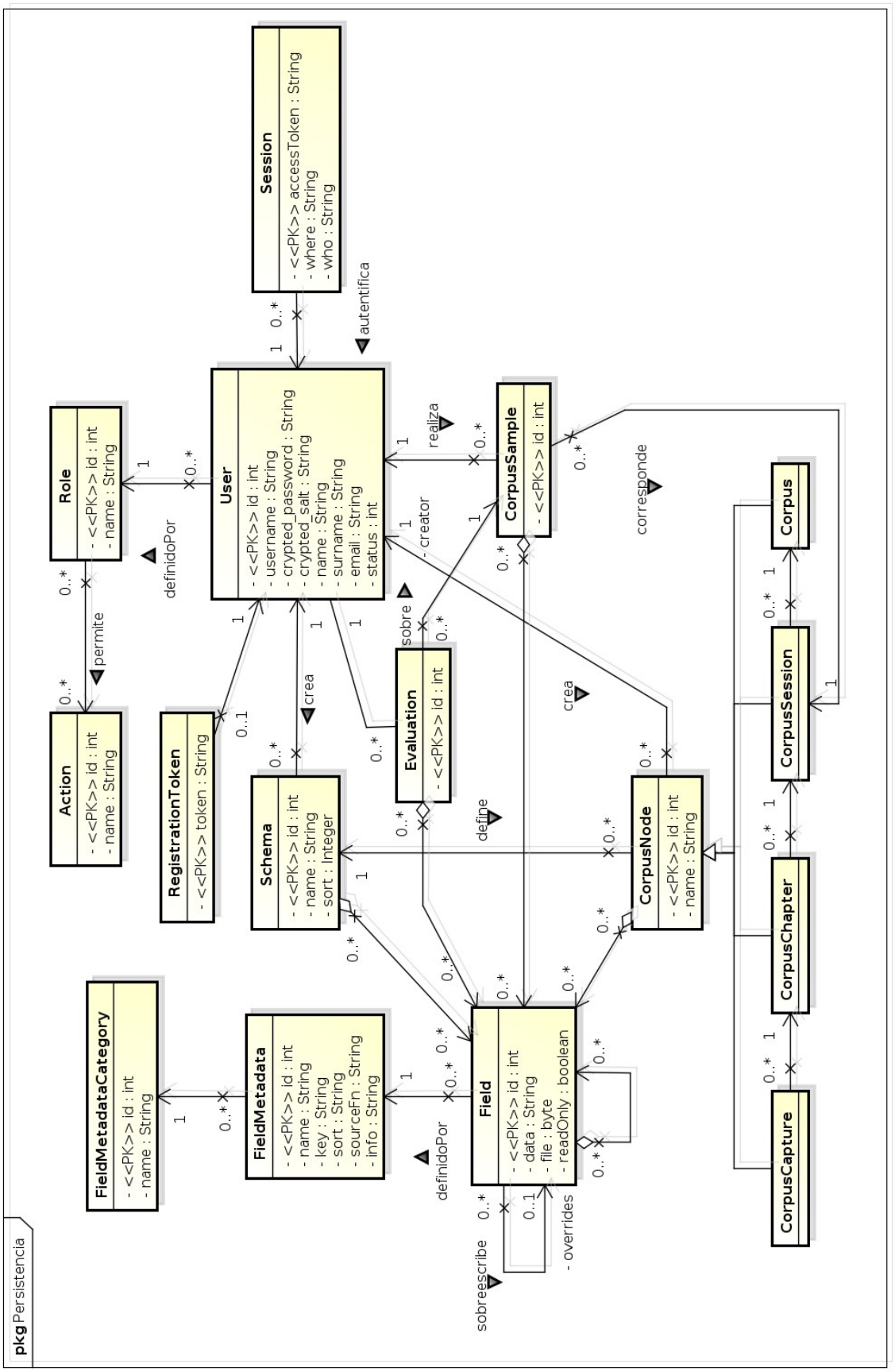


Figura 19: Diagrama Entidad-Relación

Finalmente se resuelven aquellas asociaciones complejas que existen en el anterior diagrama entidad-relación. Ejemplos de asociaciones complejas pueden ser especializaciones y asociaciones muchos a muchos. El esquema relacional resultante es el siguiente:

actions (**id**, name)

roles (**id**, name)

actions_roles (**action_id, role_id**)

users (**id**, username, crypted_password, password_salt, name, surname, email, status, role_id)

sessions (**access_token**, where, who, user_id)

registration_tokens (**token**, user_id)

field_metadata_categories (**id**, name)

field_metadatas (**id**, name, key, sort, sourceFn, info, field_metadata_category_id)

fields (**id**, data, file, read_only, field_metadata_id, overrides_id [field])

fields_fields (**field_id, parent_id** [field], order)

schemas (**id**, name, sort, user_id)

fields_schemas (**field_id, schema_id**, order)

corpuses (**id**, name, schema_id, creator_id [user])

fields_corpuses (**field_id, corpus_id**, order)

corpus_sessions (**id**, name, schema_id, creator_id [user], corpus_id)

fields_corpus_sessions (**field_id, corpus_session_id**, order)

corpus_chapters (**id**, name, schema_id, creator_id [user], corpus_session_id)

fields_corpus_chapters (**field_id, corpus_chapter_id**, order)

corpus_captures (**id**, name, schema_id, creator_id [user], corpus_capture_id)

fields_corpus_captures (**field_id, corpus_capture_id**, order)

corpus_samples (**id**, user_id, corpus_session_id)

fields_corpus_samples (**field_id, corpus_samples_id**, order)

evaluations (**id**, user_id, corpus_sample_id)

fields_evaluations (**field_id, evaluation_id**, order)

5.4. Arquitectura lógica

En este apartado se explicará en detalle las razones por las cuales se ha conformado la arquitectura lógica propuesta. La necesidad de realizar este proceso durante la fase de diseño es porque permite describir adecuadamente los diversos componentes y subsistemas que componen el sistema a diseñar junto con las relaciones existentes entre ellos.

Dado que el sistema está dividido en dos partes bien diferenciadas – cliente y servidor – es necesario especificar la arquitectura lógica de cada una de ellas por separado.

5.4.1. Cliente

En el desarrollo que nos acontece, el cliente es el que mayor lógica de negocio posee. Además de ser el encargado de gestionar las vistas también debe lidiar con la forma de confeccionar ciertos objetos, como pueden ser los corpus.

Analizando el tipo de problema que se tenía que abordar y el tipo de tecnología que se iba a emplear, el alumno consideró que la mejor forma de dar una solución adecuada sería realizar un desarrollo basado en componentes.

El principal objetivo que busca este tipo de desarrollos es la reducción o gestión de la complejidad del sistema mediante una división de éste en partes más pequeñas (“divide y vencerás”). Estas partes – denominadas componentes – poseen una lógica propia que relaciona varias entidades del modelo y adquieren unas responsabilidades frente al exterior.

Otro objetivo que se busca es promover la reusabilidad del código producido, es decir, reducir la necesidad de generar código nuevo o duplicado. Gracias a ello, no es necesario testear las partes del código que se hubieran generado sin haber seguido este tipo de desarrollo, sino que un testeo exhaustivo del componente garantiza que su comportamiento va a ser igual en todas las partes del código donde se soliciten sus servicios.

Un componente debe tener un contrato definido, en otras palabras, un acuerdo frente a otras entidades de que su comportamiento – el cual está encapsulado en él mismo y no permite su introspección – será tal y como se define en éste. A una determinada llamada de una operación con un tipo concreto de parámetros siempre debe responder de la misma forma (excepto que dependa de una secuencia). Este contrato se puede especificar fácilmente mediante definiciones de interfaces, que cada componente internamente debe realizar, ofreciéndolas al exterior para que puedan ser utilizadas.

Otra obligación de los componentes es que deben ser independientes del resto del código. Sin embargo, pueden emplear otras interfaces de otros componentes. Este hecho aumenta considerablemente su portabilidad a otras aplicaciones.

En última instancia, el desarrollo basado en componentes es una evolución *natural* de la orientación basada en objetos, ya que, esta última encapsula el comportamiento y estado de una entidad, mientras que un componente encapsula un conjunto de comportamientos y estados de varias entidades fuertemente relacionadas. Es decir, un componente encapsula lógica.

Permite a su vez a ofrecerlos a otros proyectos o desarrollos, lo que conlleva a una reducción en los costes de mantenimiento. Además, ofreciendo el componente a otros clientes, aumenta su robustez mediante la recopilación de errores producidos o comportamiento erráticos surgidos en otros escenarios

no contemplados en la fase de pruebas.

Después del análisis realizado y tras varias iteraciones en la fase del diseño, se obtuvieron los siguientes paquetes (divisiones del diagrama de clases de diseño y que tienen grandes posibilidades de promocionarse como componentes independientes):

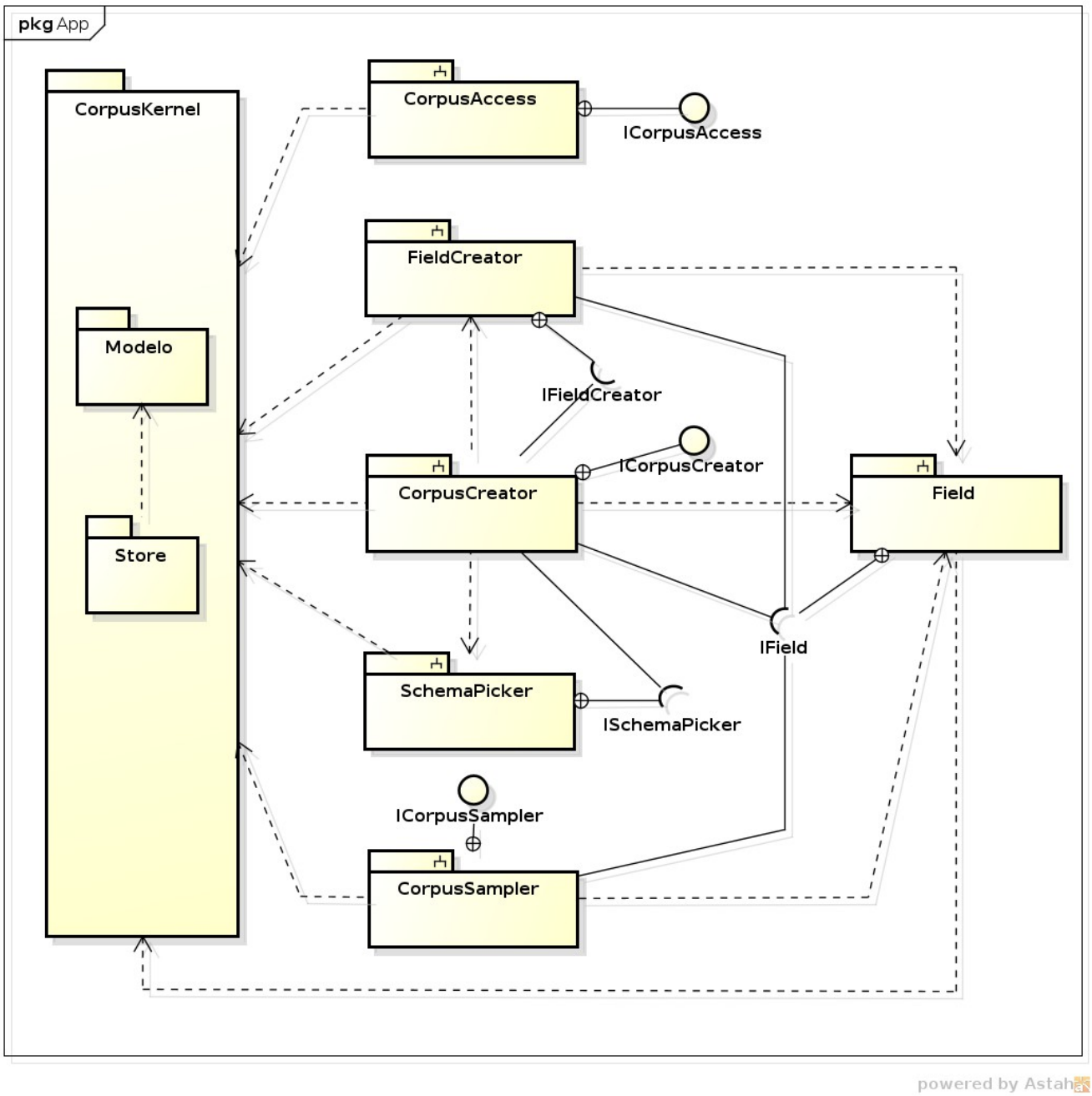


Figura 20: Arquitectura lógica del cliente

Como se puede observar, existen 7 distintos paquetes cuyas funcionalidades y responsabilidades están bien definidas.

El paquete *CorpusKernel*, es un paquete que actúa como una especie de repositorio. En él están contenidos los modelos del sistema y los almacenes de sus instancias. Las entidades del modelo están localizadas en el paquete *modelo* y permiten su reutilización en otras partes del código así como en los componentes. Además, existen almacenes que permiten la gestión de las entidades instanciadas. Tanto los modelos y los almacenes tienen estipulados los mecanismos para acceder al servidor y manejar su persistencia en éste. Esto permite que si alguna entidad o recurso del servidor es modificado, tan solo sea necesario modificar las entidades necesarias y no el resto del código dependiente.

El resto de paquetes se pueden convertir perfectamente en componentes independientes, los cuales ofrecen y demandan diversas interfaces.

El componente *CorpusAccess* gestiona toda la lógica asociada con la identificación y autenticación del usuario ante el servidor. Mediante su API se puede crear una aplicación que actúa en consecuencia con el estado de la sesión del usuario. Además proporciona una interfaz gráfica que permite al usuario introducir sus credenciales para ser autentificado ante el sistema.

El componente *FieldCreator* proporciona una interfaz gráfica que permite al usuario elegir un campo concreto de la gama de tipos de campos disponibles.

El componente *SchemaPicker*, al igual que FieldCreator, permite al usuario escoger un tipo de esquema mediante una interfaz gráfica.

El componente *CorpusCreator* gestiona todo lo relacionado con el proceso de creación y especificación de un corpus. Permite crear la jerarquía de los nodos y añadir los diversos campos que estos necesiten. También permite guardar el corpus de forma local codificado en JSON o enviarlo al servidor para que pueda ser realizado.

El componente *CorpusSampler* permite al usuario realizar los corpus previamente creados en el servidor. El usuario al principio debe seleccionar una sesión concreta y el componente automáticamente creará en el servidor una muestra. Posteriormente, el componente ira renderizando los diversos nodos existentes en el corpus a medida que el usuario los realice.

El componente *Field* posee toda la lógica relacionada con un campo. Su interfaz es fundamental para el resto de componentes y código. Encapsula todo el comportamiento que se podría esperar de un *input* avanzado, como por ejemplo, permitir previsualizar el contenido de los ficheros seleccionados o descargados. Además, la interfaz gráfica y el comportamiento viene determinada por el tipo de campo que se haya proporcionado.

Además del concepto de componente, es interesante mencionar que es para el alumno el término “módulo”.

Un módulo en la práctica es igual que un componente, pero conceptualmente es algo más que eso. Un componente es un fragmento de código que permite realizar tareas sencillas, como por ejemplo, crear un campo, seleccionar un elemento de una lista... Sin embargo, un módulo sería un componente que permite realizar una tarea compleja, como puede ser crear un corpus. Los módulos poseen la suficiente autonomía para que que ellos mismos confeccionen una aplicación propia.

De los anteriores componentes, tres de ellos son considerados además como módulos: CorpusAccess, CorpusCreator y CorpusSampler.

A continuación se desglosarán los diversos paquetes de la aplicación web.

5.4.1.1. CorpusKernel

Como se comentó, posee toda la parte esencial del sistema: el núcleo. En él están definidos los distintos modelos del dominio y los almacenes que gestionan sus instancias en la parte del cliente.

El mapa conceptual del modelo es prácticamente el mismo, salvando que ahora están definidas las operaciones necesarias en la parte del cliente así como diversas modificaciones para adaptarse adecuadamente a la tecnología elegida para la aplicación web.

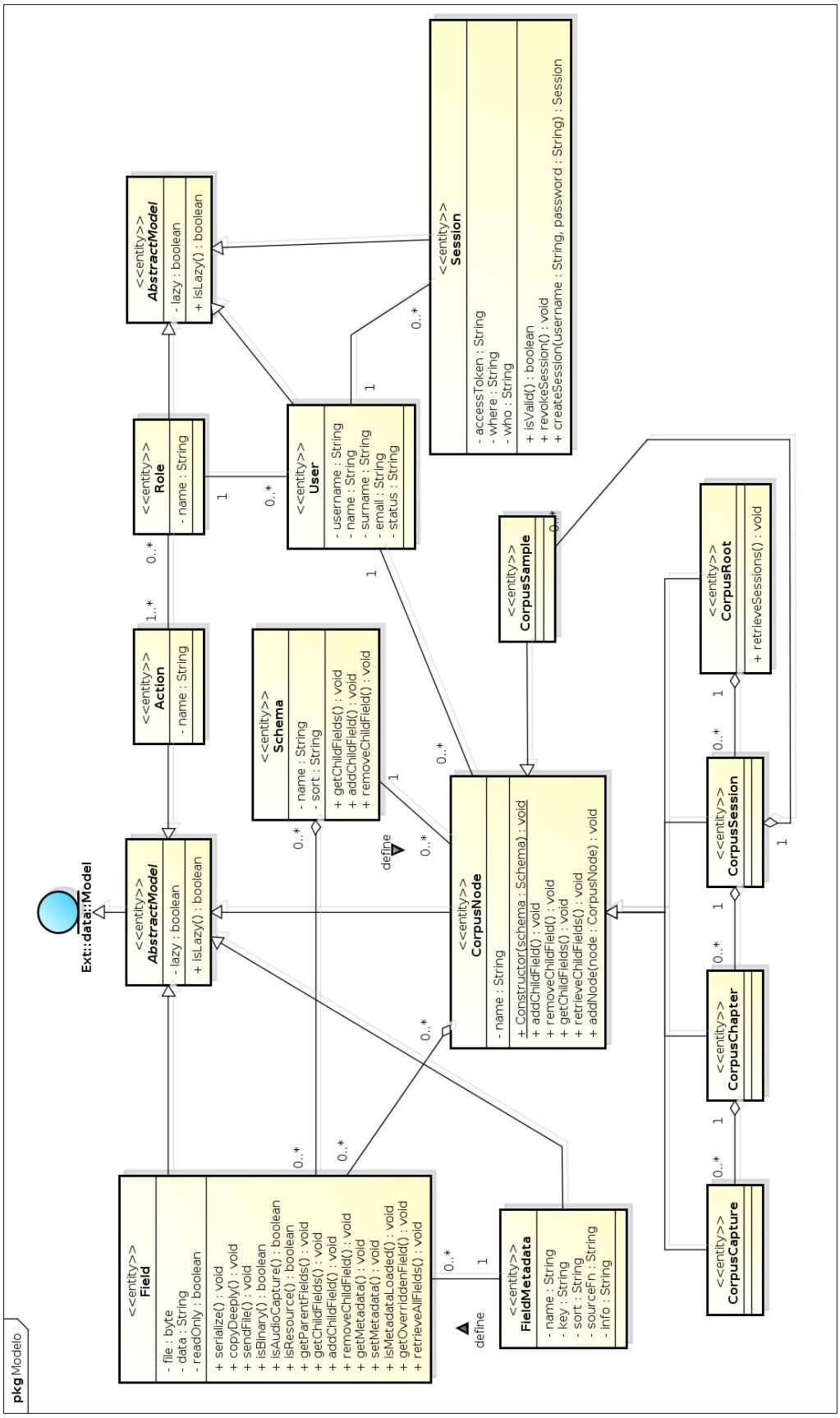


Figura 21: Modelo del componente CorpusKernel

Como se puede observar, todos los modelos son especializaciones de *AbstractModel*. Gracias a ello, se consigue que todas las entidades posean las mismas características que se esperan de un Active Record, es decir, todas las entidades tienen la lógica y operaciones suficientes para tratar su propia persistencia en el sistema; por ello no es necesario recurrir a otros patrones como puede ser DAO, brokers o gestores de persistencia. A su vez, se especializan de *Ext::data::Model* que es la clase del framework para la especificación de entidades del modelo.

En el paquete de almacenes sólo están aquellos almacenes necesarios para el correcto funcionamiento de la aplicación. No todas las entidades requieren almacenes propios ya que normalmente serán entidades que estén muy relacionadas con otras o bien existan un número muy reducido por lo que su gestión sea prácticamente trivial.

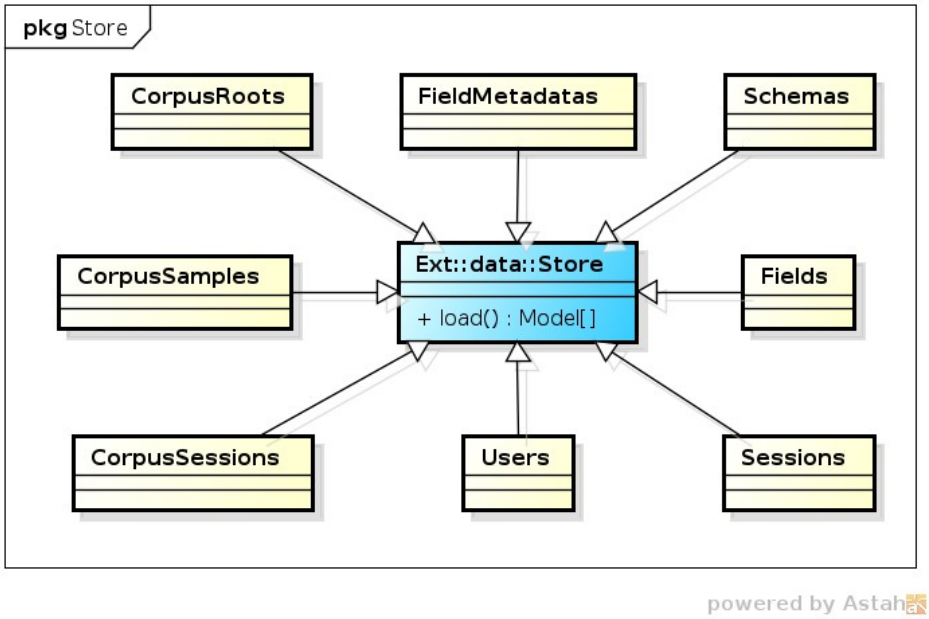


Figura 22: Almacenes del componente CorpusKernel

Al igual que sucedía con las entidades del componente, todos los almacenes son especializaciones de *Ext::data::Store*, que es una clase del framework que posee toda la lógica de un almacén de entidades.

Finalmente, se muestra la figura que relaciona ambas capas y permite visualizar el contenido global del componente.

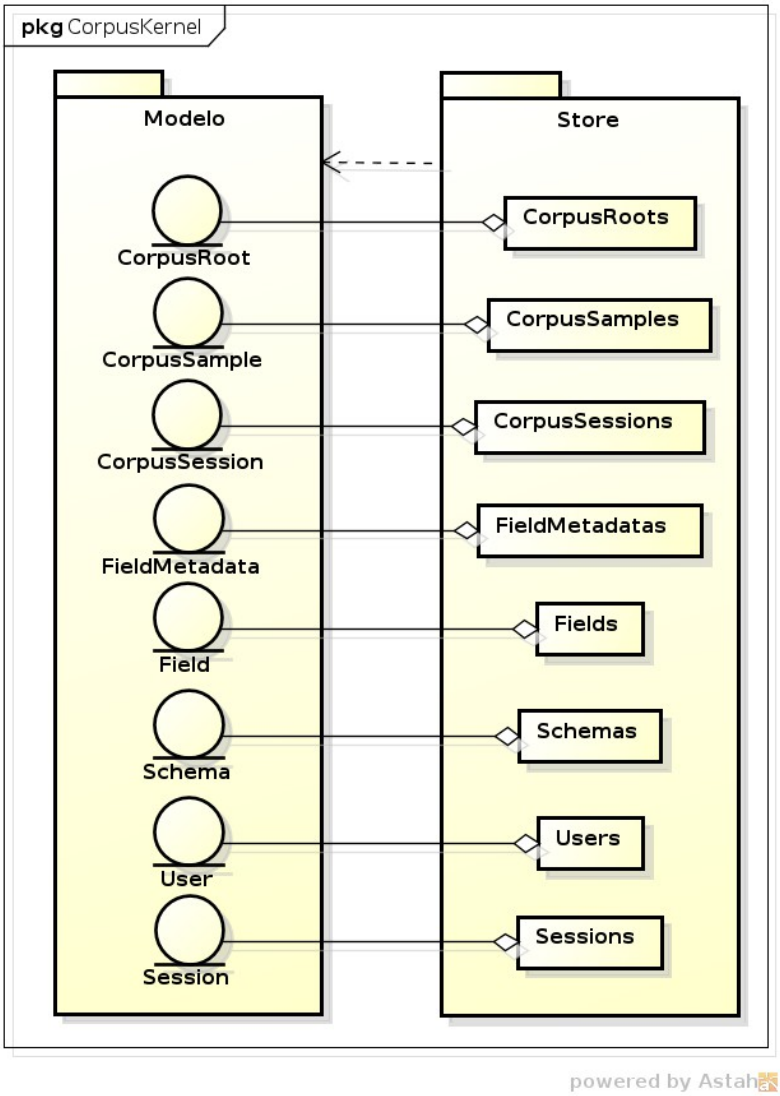


Figura 23: Componente CorpusKernel

5.4.1.2. CorpusAccess

Es el módulo encargado de la gestión y control de la sesión del usuario frente al servidor. Proporciona una interfaz gráfica para la acreditación de usuario.

Está compuesto por dos capas, modelo y presentación. Comenzaremos por esta última.

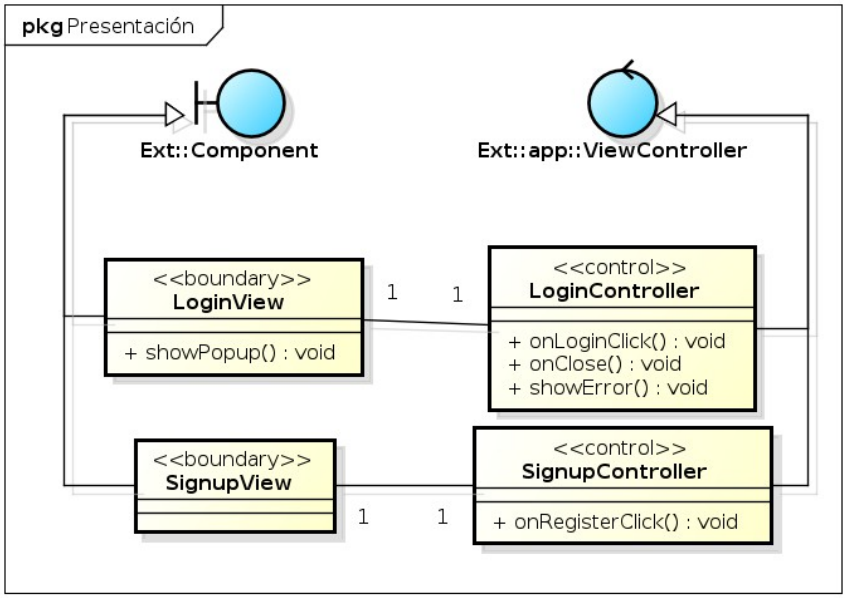


Figura 24: Presentación del componente CorpusAccess

No existe la necesidad de más vistas ya que el cometido esencial de este módulo es la gestión de la sesión del usuario y realizar solicitudes de nuevos registros en el sistema.

La clase *Ext::Component* representa cualquier elemento visual del framework ExtJS, por esa razón, todas las vistas deben especializarse de ella. Por otra parte, sus controladores tienen especificado los manejadores de las distintas señales posibles de la vista (por ejemplo, cuando el usuario pulsa el botón de loguear o bien si cancela la identificación).

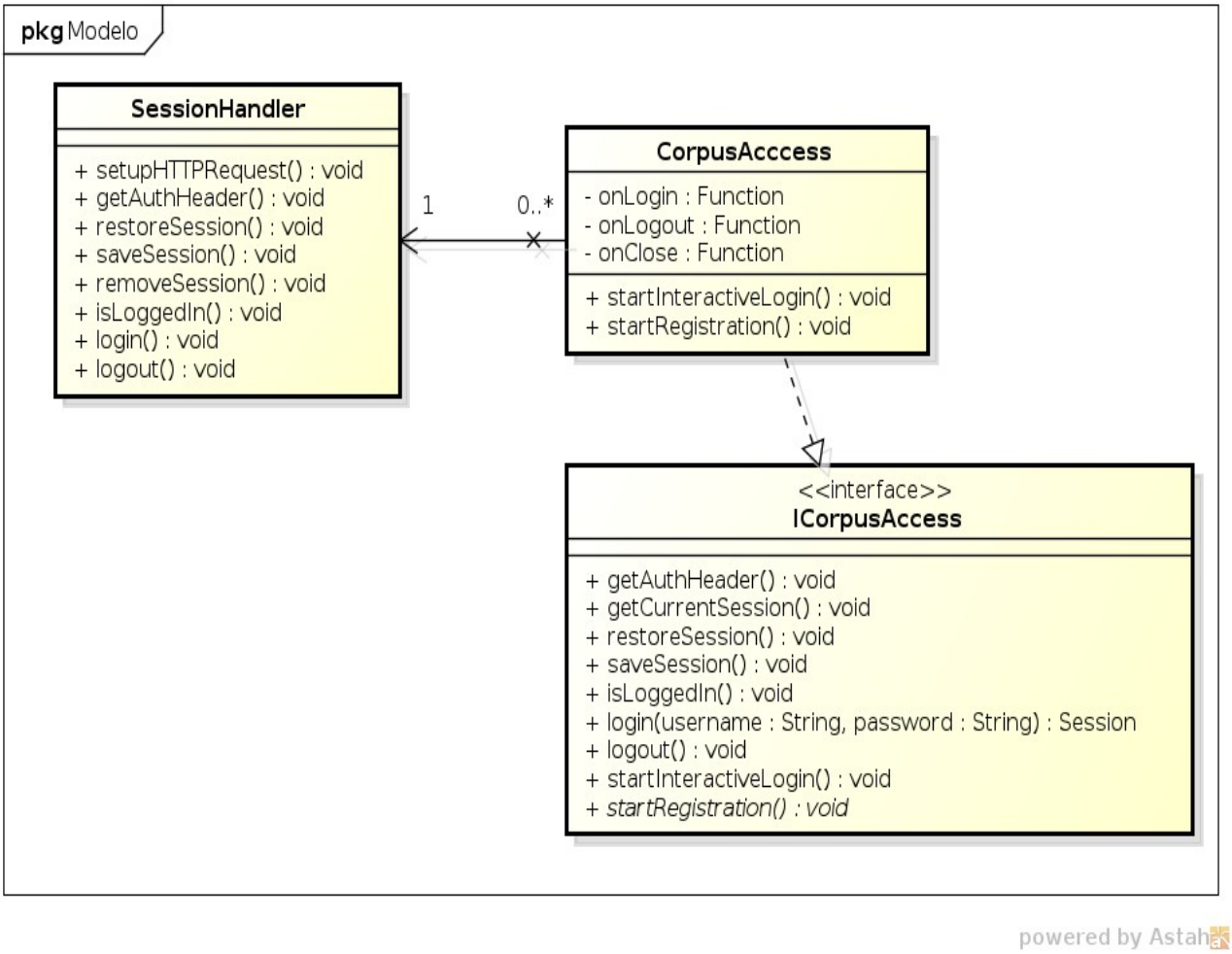


Figura 25: Modelo dle componente CorpusAccess

El modelo en cambio posee toda la lógica de comunicación con el servidor y las negociaciones necesarias para iniciar una nueva sesión en ambos lados del sistema.

Tiene especifica la interfaz que ofrece como componente al exterior (ICorpusAccess) y posee también la clase que la realiza. Sin embargo, se pensó que todo el núcleo de autenticación estaría mejor concentrarlo en una clase por separado (SessionHandler).

Finalmente el esquema del módulo sería el siguiente:

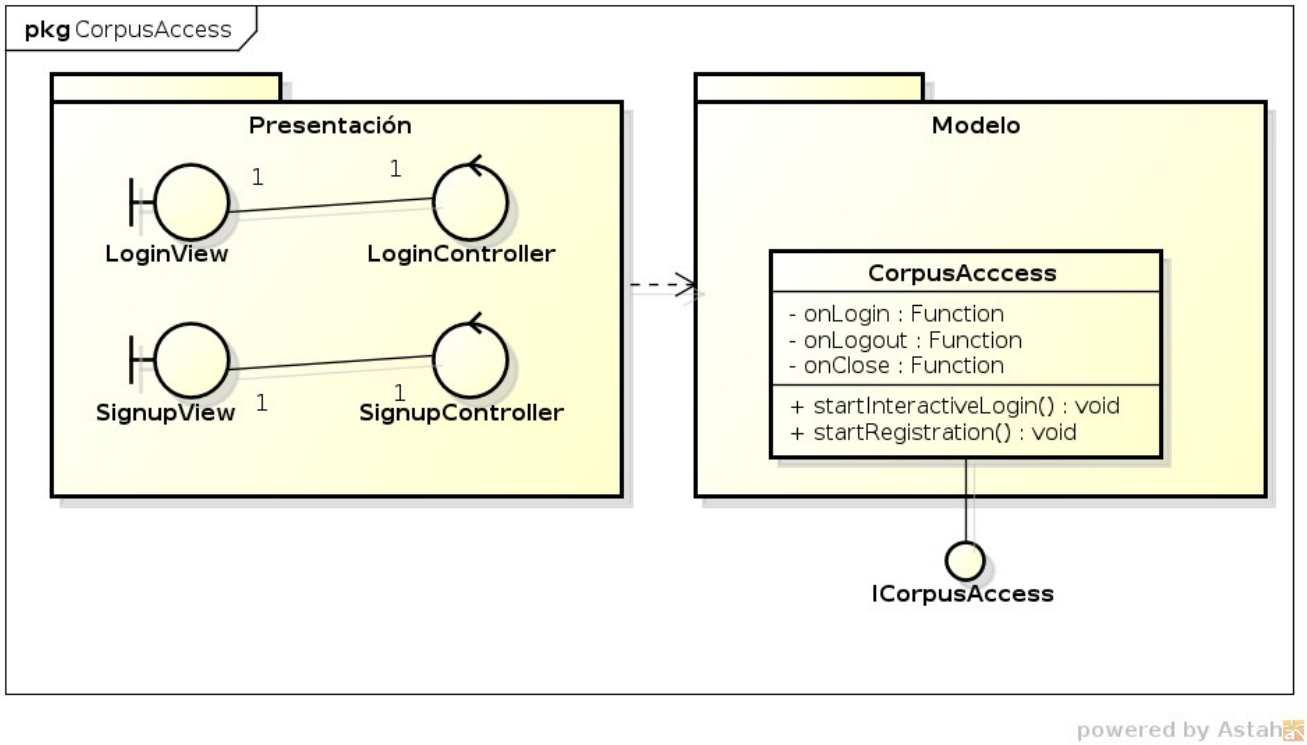


Figura 26: Componente CorpusAccess

Dado que JavaScript de forma nativa permite la programación basada en eventos, da la posibilidad que la comunicación entre el modelo y el controlador se realice utilizando este paradigma. Por esa razón, no existe una dependencia real entre ambas capas aunque en la figura anterior se haya reflejado. El hecho de que una entidad dispare un evento y otra escuche no debería considerase como que una clase mande un mensaje (realice una llamada a un método) a otra, incluso de considerarse, sería en la dirección contraria, ya que el modelo escucha el evento del controlador.

5.4.1.3. CorpusCreator

Gracias a este módulo, un usuario puede generar y editar un corpus mediante una interfaz gráfica de apoyo.

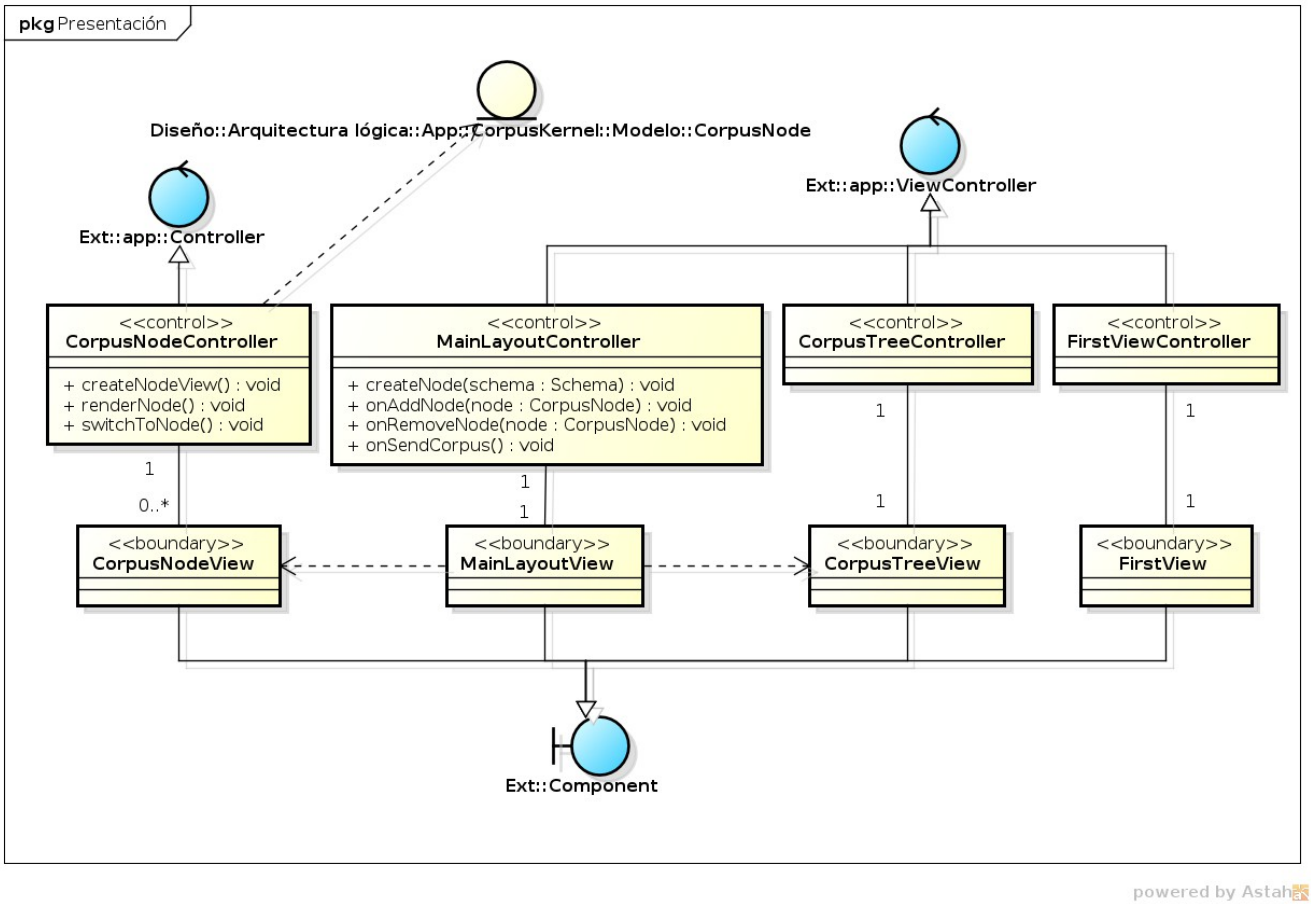


Figura 27: Presentación del componente CorpusCreator

La vista principal está especifica en la clase *MainLayoutView* y a su vez está compuesta por dos subvistas, *CorpusTreeView y CorpusNodeView*. La primera consigue mostrar la jerarquía del corpus mediante un menú visual en forma de árbol, mientras que la segunda – que ocupa la mayor parte de la vista principal – está destinada a mostrar el contenido, es decir, el conjunto de campos, del nodo seleccionado. Existe una vista inicial, *FirstView*, que permite seleccionar el tipo de modo, creación de un nuevo corpus o edición de uno existente, el cual puede ser cargado mediante un fichero JSON local o remoto.

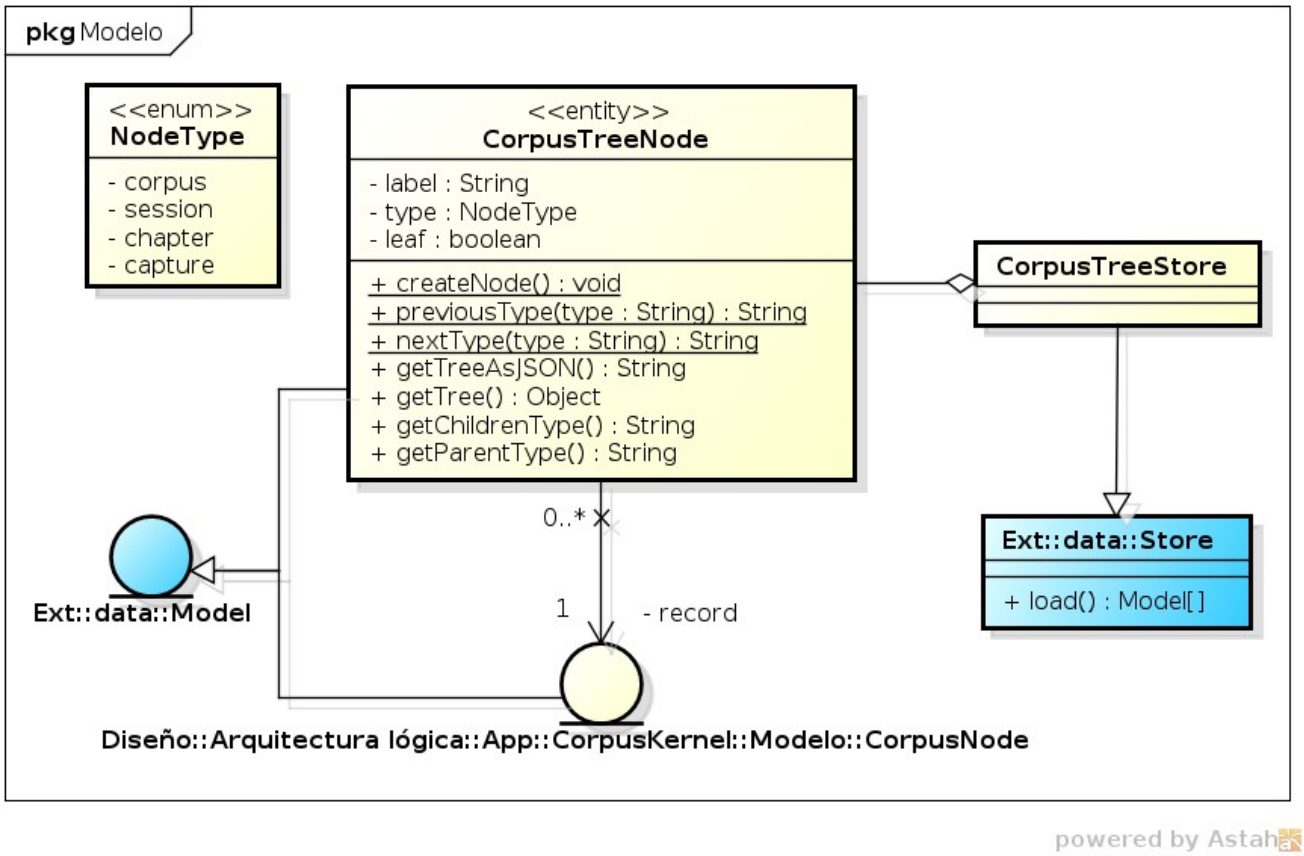


Figura 28: Modelo del componente CorpusCreator

La clase *CorpusTreeNode* representa una entidad *CorpusNode* empaquetada en un elemento capaz de añadirla en una vista en forma de árbol. El almacén que guarda la jerarquía es *CorpusTreeStore*.

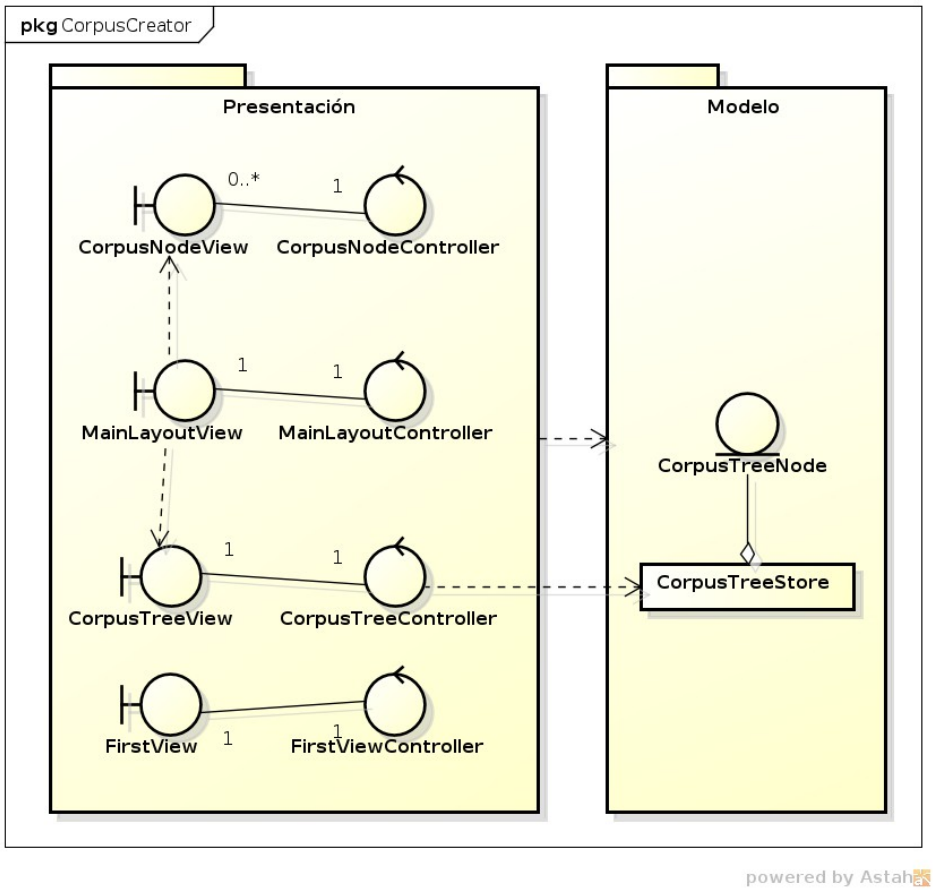


Figura 29: CorpusCreator

La figura anterior muestra la arquitectura del componente y las relaciones existentes entre las diversas capas.

5.4.1.4. Field

El componente Field es una pieza muy importante en el sistema. Es capaz de conformar campos avanzados con un determinado comportamiento dependiendo de su naturaleza.

Emplea activamente la entidad del modelo *Field* y confecciona un determinado elemento interactivo acorde con su información contenida (estado). Gracias a ello, el usuario siempre modifica las entidades *Field* utilizando el mismo componente, lo que permite a su vez unificar y generalizar su lógica asociada.

A continuación se muestra la arquitectura lógica del componente en cuestión:

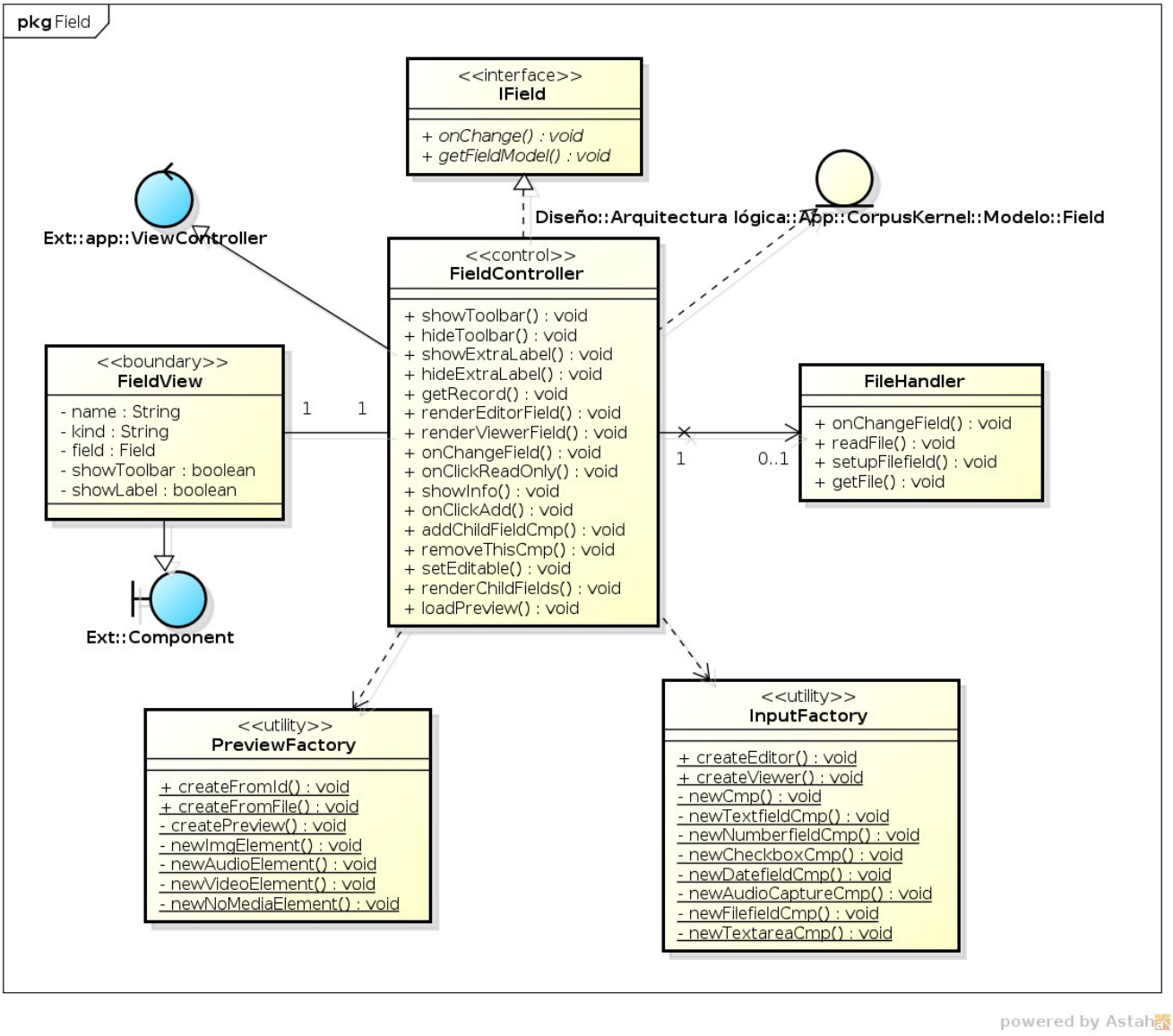


Figura 30: Field

Sólo existe una vista, la cual está lo suficientemente parametrizada como para ser ampliamente modificable y poder adaptarla a los diversos tipos de campo. La vista está compuesta por una etiqueta descriptiva, una barra de edición y un *input*. Éste último es creado de forma transparente por el controlador de la vista utilizando la clase factoría *InputFactory*.

Un campo viene definido por su tipo, que puede ser “string”, “date”, “text”, “number”.....; y por su contexto: editor (editor) o visor (viewer).

El tipo determina la clase del input que se genera y añade a la vista, mientras que su contexto determina su forma de uso. Es decir, los campos cuyo contexto sea editor, permitirán al usuario realizar operaciones de gestión (añadir y eliminar subcampos, eliminar el campo actual...) del elemento, sin embargo, cuando es visor sólo permite la visualización de su contenido si es sólo lectura, y en caso contrario, permitiría cambiar sólo el valor asociado a éste.

La interfaz que proporciona el componente es muy básica, ya que es preferible sólo permitir aquel acceso puramente necesario para su correcta interacción.

- El método onChange, permite capturar el evento que sucede cuando el valor del campo es cambiado por el usuario.
- Por otra parte, el método getFieldModel, permite recuperar el modelo asociado al componente y poder hacer operaciones sobre éste. Esto permite por ejemplo tener la capacidad de enviar la información contenida al servidor.

El componente también permite gestionar ficheros multimedia como contenido del campo. Tiene además la capacidad de generar una *preview* con el contenido del fichero multimedia especificado. Gracias a ello el usuario puede ver el contenido en el propio navegador sin necesidad de recurrir a otras herramientas. Debido a que los corpus que se manipulan en este sistema son lingüísticos, los visores son sólo capaces de manejar audios, vídeos e imágenes. Para realizar dicho cometido, se emplea la factoría *PreviewFactory*.

5.4.2. Servidor

Debido a que gran parte de la lógica del sistema está localizada en el cliente, la arquitectura del servidor es bastante sencilla.

Cuenta con dos únicas capas, la de despliegue y la del modelo. La primera contiene todas las clases necesarias para la gestión de los recursos de servidor REST. Poseen lógica suficiente para manejar los objetos del modelo, he ahí la dependencia existente entre ambas capas.

La capa del modelo posee las entidades esenciales del modelo de dominio más alguna clase extra necesaria para la correcta gestión del sistema.

El acceso a la persistencia se realiza a través de los Active Record de los modelos, por esa razón, no existe capa de persistencia como tal.

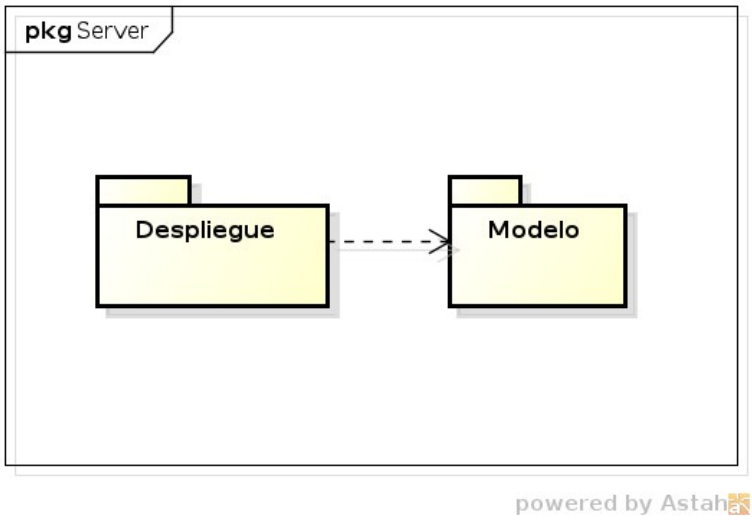


Figura 31: Arquitectura lógica del servidor

Si nos centramos más en cada paquete podremos hacer una descripción de cada clase que lo compone.

5.4.2.1. Capa de despliegue

El cometido de la capa de despliegue es proporcionar las clases necesarias para permitir el acceso a los distintos recursos que componen el servidor REST. Las rutas definidas en el servidor enlazan la tupla recurso-verbo con una operación determinada de una clase.

Por ejemplo, si un cliente solicita el recurso User junto con el verbo get; al no proporcionar ningún parámetro, la clase Users (todos los controladores están en plural) accederá al método index (que proporciona todos los usuarios) debido a que existe una ruta que relaciona ambas.

Por convención, los siguientes métodos realizan las siguientes operaciones:

- **index**: método asociado cuando se pide el recurso sin especificar su identificador único y se emplea el método GET en la petición HTTP (GET /resource). Devuelve todas las instancias de dicho recurso.
- **create**: ejecutado cuando se solicita el recurso mediante el verbo POST y especificando, normalmente, la información necesaria para crear un nuevo recurso (POST /resource). Su comportamiento es prácticamente idéntico a la operación CRUD *create*.
- **show**: llamado cuando la petición HTTP solicita el recurso mediante el verbo GET y pasando un identificador único (GET /resource/234). Devuelve la instancia del recurso solicitado al igual que se podría esperar de la operación CRUD *retrieve*.
- **update**: método asociado a la petición HTTP realizada mediante el verbo PUT (o PATCH). Al igual que con el método *create*, es necesario proporcionar en el cuerpo de la petición las modificaciones solicitadas sobre un recurso en concreto (PUT /resource). Se asemeja a la operación CRUD *update*.
- **destroy**: llamado cuando la petición HTTP, especificando un identificador único de recurso, emplea el verbo *delete* (DELETE /resource/234). Elimina la instancia del recurso del sistema, tal y como se espera de la operación CRUD *delete*.

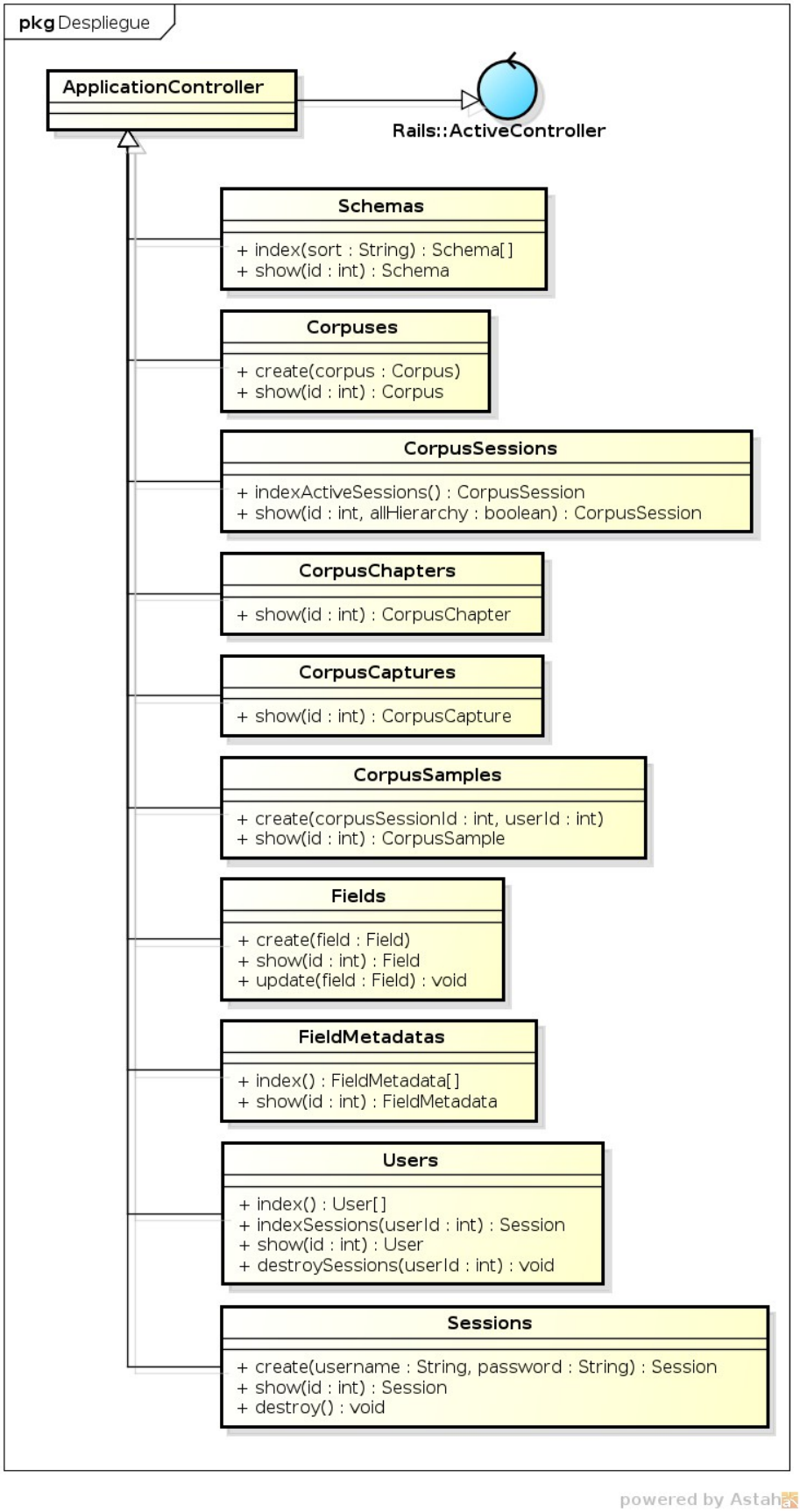


Figura 32: Capa de despliegue del servidor

Todos los controladores en Ruby on Rails heredan del controlador Application (a su vez de ActiveController), esto se debe a que emplea el patrón de diseño “Controlador de fachada” (Front Controller). Básicamente consiste en que todas las peticiones se realizan realmente sobre un único punto de entrada al servidor web; de esta forma se pueden especificar fácilmente operaciones iniciales y comunes a todos los controladores. Una vez concluido su lógica asociada, actúa como un despachador (dispatcher) de la petición y la dirige al controlador asociado al recurso solicitado por esta. Este comportamiento es muy interesante para por ejemplo la gestión de la autenticación del usuario frente al sistema y el acceso a cierta información.

A continuación, en vez de describir cada operación de los controladores de la capa de despliegue, se realizará una pequeña tabla desde el punto de vista de los servicios/recursos que proporciona. Es decir, se definirá la API REST pública del servidor.

Verbo HTTP	Path	Controlador # Acción	Parámetros	Descripción

GET	/schemas/[:sort]	Schemas#index	• sort: especifica el tipo de esquema. Posibles valores:
			  • corpuses
			  • sessions
			  • chapters
			  • captures
				Devuelve todos los esquemas del tipo solicitado. Si no se proporciona el tipo, se devuelve todos los esquemas existentes.

GET	/schemas/:id	Schemas#show	• id: identificador único del recurso.	Devuelve la instancia del recurso solicitado.
POST	/corpuses	Corpuses#create	• corpus: es obligatorio proporcionar un árbol cuya raíz sea un nodo corpus. Debe estar codificado en JSON.	Crea un nuevo corpus completo, es decir, crea todos los nodos descendientes así como todos los campos existentes.
GET	/corpuses/:id	Corpuses#show	• id: identificador único del recurso.	Devuelve la instancia del recurso solicitado.
GET	/corpus_sessions/active	CorpusSessions#indexActiveSessions	-	Devuelve todas las sesiones activas de todos los corpus existentes en el sistema.
GET	/corpus_sessions/:id?[:allHierarchy]	CorpusSessions#show	• id: identificador único del recurso. • allHierarchy: su valor es booleano y es opcional.	Devuelve el recurso solicitado. Si el parámetro *allHierarchy* es verdadero, devuelve la sesión junto con sus nodos hijos recursivamente.
GET	/corpus_chapters/:id	CorpusChapters#show	• id: identificador único del recurso.	Devuelve la instancia del recurso solicitado.
GET	/corpus_captures/:id	CorpusCaptures#show	• id: identificador único del recurso.	Devuelve la instancia del recurso solicitado.

Verbo HTTP	Path	Controlador # Acción	Parámetros	Descripción
POST	/corpus_samples	CorpusSamples #create	• corpusSessionId: identificador de la sesión a la que pertenecerá la muestra. • userId: identificador del usuario que va a realizar la muestra.	Crea una nueva muestra duplicando todos aquellos campos editables de los nodos descendientes de la sesión.
GET	/ corpus_samples/: id	CorpusSamples #show	• id: identificador único del recurso.	Devuelve la instancia del recurso solicitado.
POST	/fields	Fields#create	• field: toda los campos pertinentes para crear un Field. En formato JSON.	Crea un nuevo Field con los valores suministrados en el cuerpo de la petición HTTP.
GET	/fields/:id	Fields#show	• id: identificador único del recurso.	Devuelve la instancia del recurso solicitado.
PUT	/fields/:id	Fields#update	• field: toda los campos pertinentes para actualizar un Field. En formato JSON.	Actualiza el Field con los datos suministrados. Sirve también para un ficheros asociado al Field.
GET	/field_metadatas	FieldMetadatas# index	-	Devuelve todas las instancias del recurso.
GET	/ field_metadatas /: id	FieldMetadatas# show	• id: identificador único del recurso.	Devuelve la instancia del recurso solicitado.
GET	/users	Users#index	-	Devuelve todas las instancias del recurso.
GET	/ users/:id/sessions	Users#indexSes sions	• id: identificador único del usuario.	Devuelve todas las sesiones del usuario solicitado.
GET	/users/:id	Users#show	• id: identificador único del recurso.	Devuelve la instancia del recurso solicitado.
DELETE	/ users/:id/sessions	Users#destroyS esisons	• id: identificador único del usuario.	Elimina todas las sesiones del usuario solicitado.
POST	/sessions	Sessions#create	• username: nombre del usuario. • password: contraseña del usuario.	Si la identificación es correcta, el sistema crea una nueva sesión y devuelve la autenticación al cliente.
GET	/sessions/:id	Sessions#show	• id: identificador único del recurso.	Devuelve la instancia del recurso solicitado.
DELETE	/sessions/:id	Sessions#destro y	• id: identificador único del recurso.	Elimina la instancia del recurso solicitado.

Tabla 13: API REST del servidor

5.4.2.2. Capa de modelo

A continuación se van a comentar algunos puntos interesantes de esta capa de la parte del servidor.

El mapa conceptual sigue siendo muy parecido respecto al suministrado en la fase de análisis. En esta versión se han añadido las operaciones necesarias para la correcta ejecución de los casos de uso descritos anteriormente.

Se han obviado todos aquellos métodos, sobretodo los relacionados con la gestión de la base de datos, que son proporcionados de forma nativa por la plataforma seleccionada para el servidor (Ruby on Rails).

Se han añadido tres nuevas clases que no existían anteriormente:

- *RegistrationToken*: su finalidad sirve únicamente para verificar el registro de un nuevo usuario mediante la comprobación y validación del email suministrado.
- *Session*: gracias a esta clase se puede hacer un sistema de identificación mediante uso de sesiones temporales. Aumenta considerablemente la seguridad debido a que el cliente no tiene que almacenar ningún dato relacionado con la contraseña original.
- *FieldFactory*: clase factoría que construye objetos de tipo Field. Dado que la construcción de los campos es en cierta forma compleja, la existencia de esta factoría estandariza la forma en la que se confeccionan los campos en el modelo. Todos los campos pueden tener subcampos asociados que a su vez pueden tener otros. Esto motiva la necesidad de realizar una construcción recursiva, además dependiendo de la naturaleza del campo es necesario crearlo de una forma u otra. Por ejemplo, en el caso de los campos que almacenan un fichero, es necesario añadir como subcampos su nombre asociado y otro que especifique el mime de éste. Sin estos campos añadidos sería imposible recuperar un fichero almacenado y servirlo a los clientes.

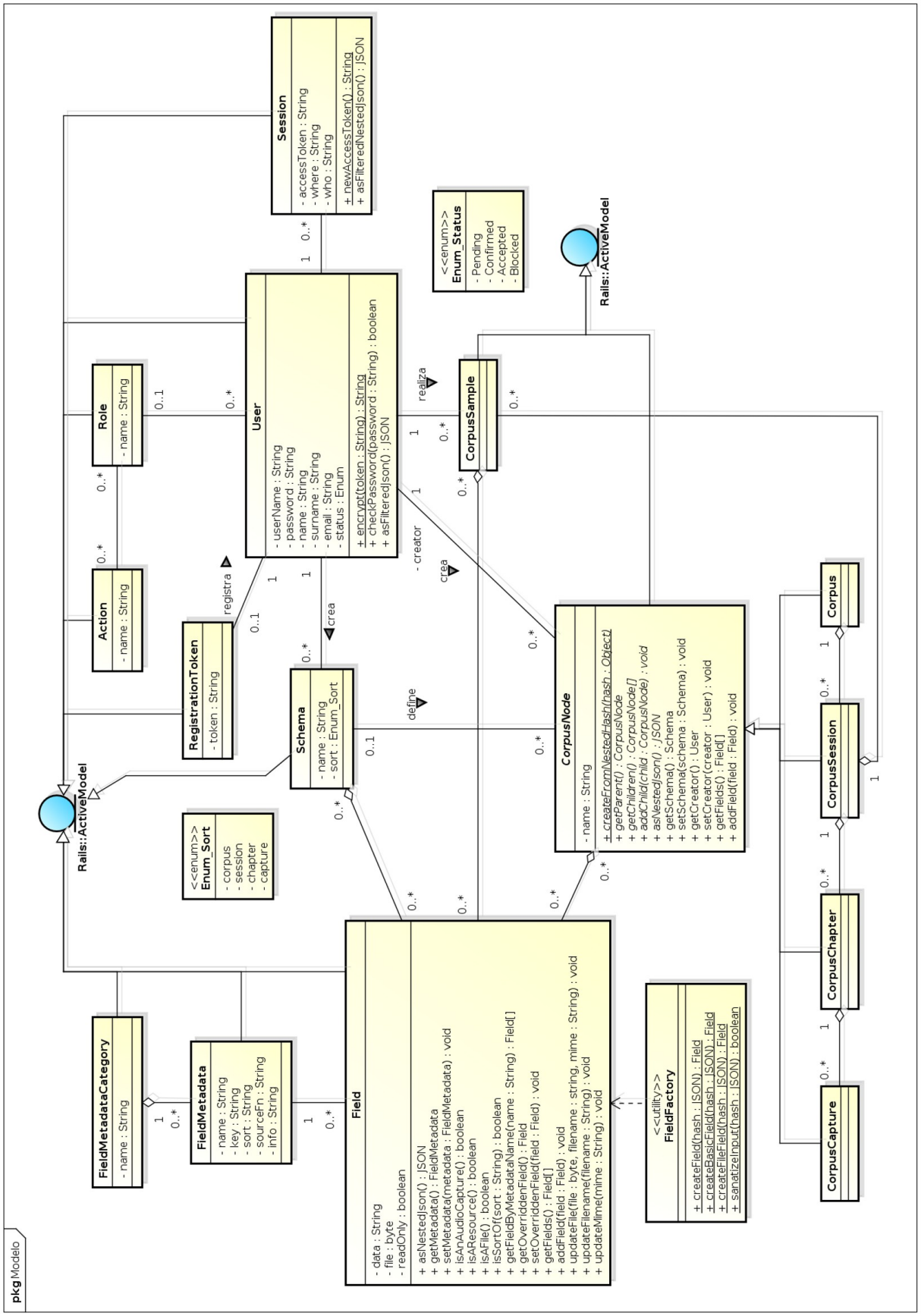
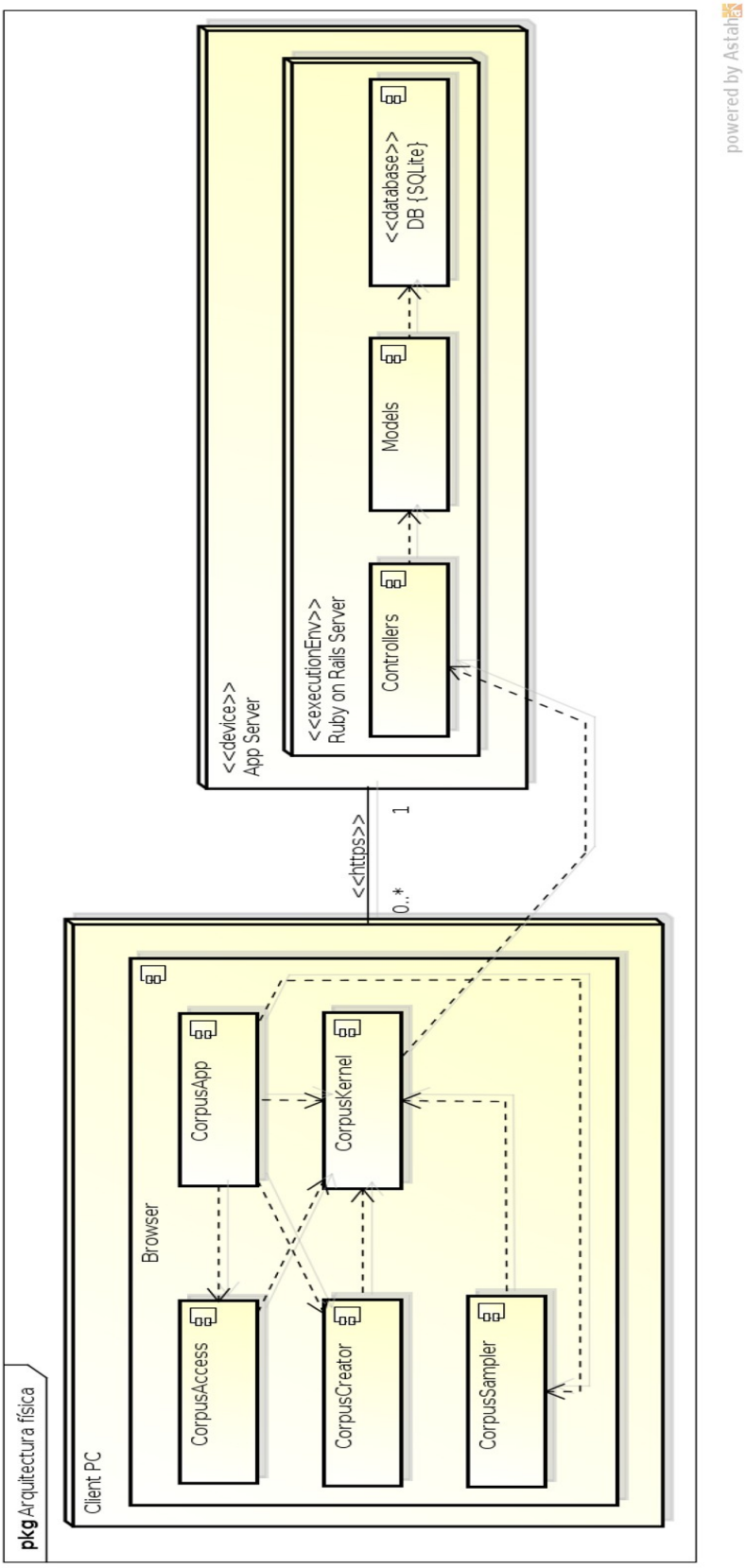


Figura 33: Modelo del servidor

5.5. Arquitectura física



5.6. Diagramas de secuencia

A raíz de que este tipo de diagramas están extremadamente condicionados al paradigma de programación orientado a objetos, resulta complicado representar en ellos interacciones distintas a llamadas de mensajes entre objetos sin caer en ambigüedades o nomenclaturas no estandarizadas. Por ejemplo, JavaScript es también funcional y permite pasar funciones como parámetros de otras funciones, dando lugar a las denominadas *callbacks*. En este caso no existe una forma clara y comúnmente aceptada para representar dichas relaciones. También, en este lenguaje, es posible realizar una programación orientada a eventos, por lo que complica aún su representación en este tipo de diagramas.

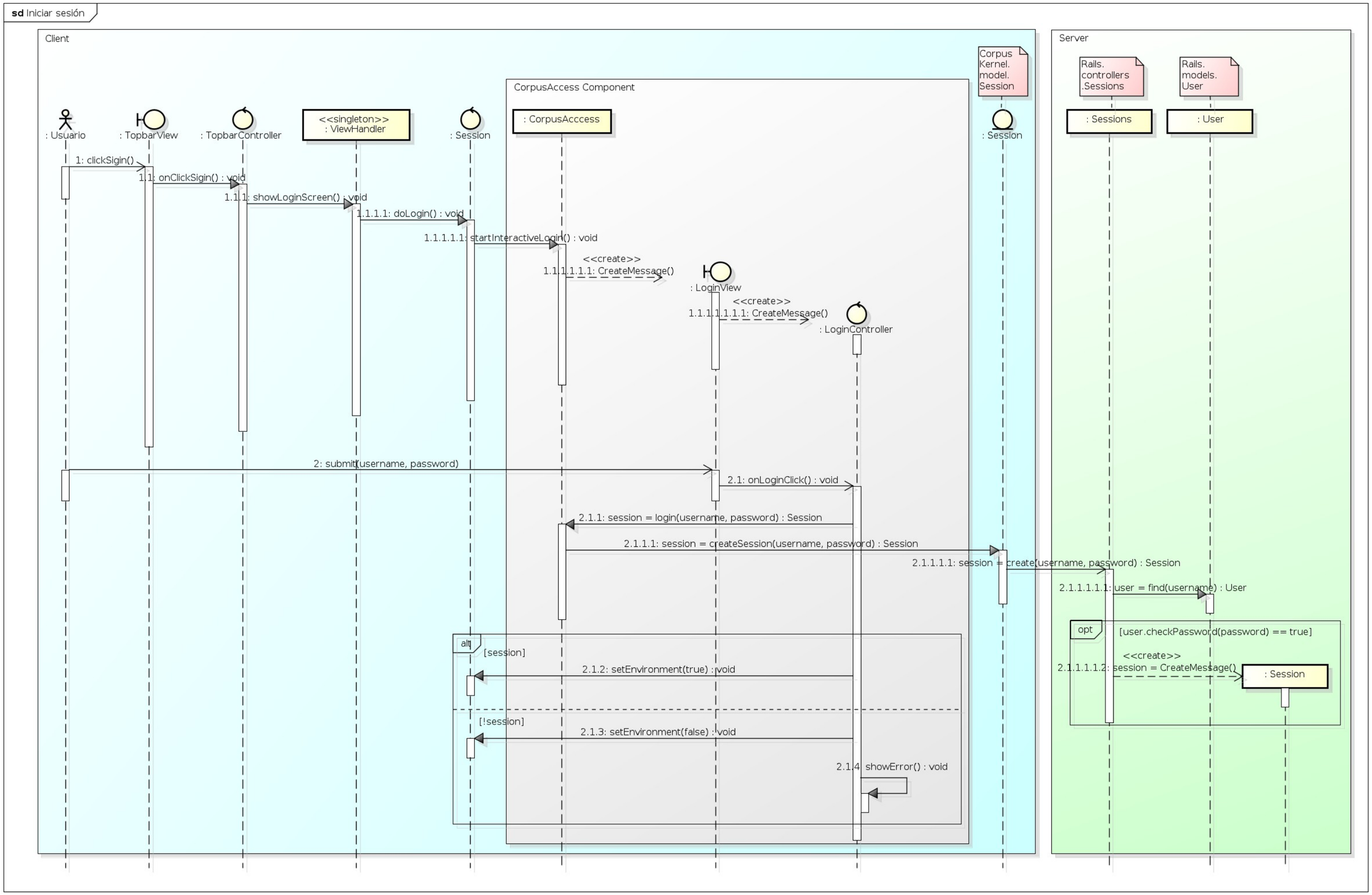
Además de que su tamaño resulta – en muchas ocasiones – que sea imposible insertarlos en un tamaño de hoja estándar (A4). Se añadirán en versión original en el dispositivo de almacenamiento que se va a proveer con este documento.

Debido a la complejidad de representación de los diagramas de secuencia de diseño de este proyecto, únicamente se van a exponer dos. Concretamente el relacionado con el caso de uso Iniciar Sesión y Crear corpus.

El primero servirá para ayudar a comprender las relaciones entre los distintos elementos del sistema así como la interacción entre el cliente y el servidor mediante un caso de uso más o menos sencillo.

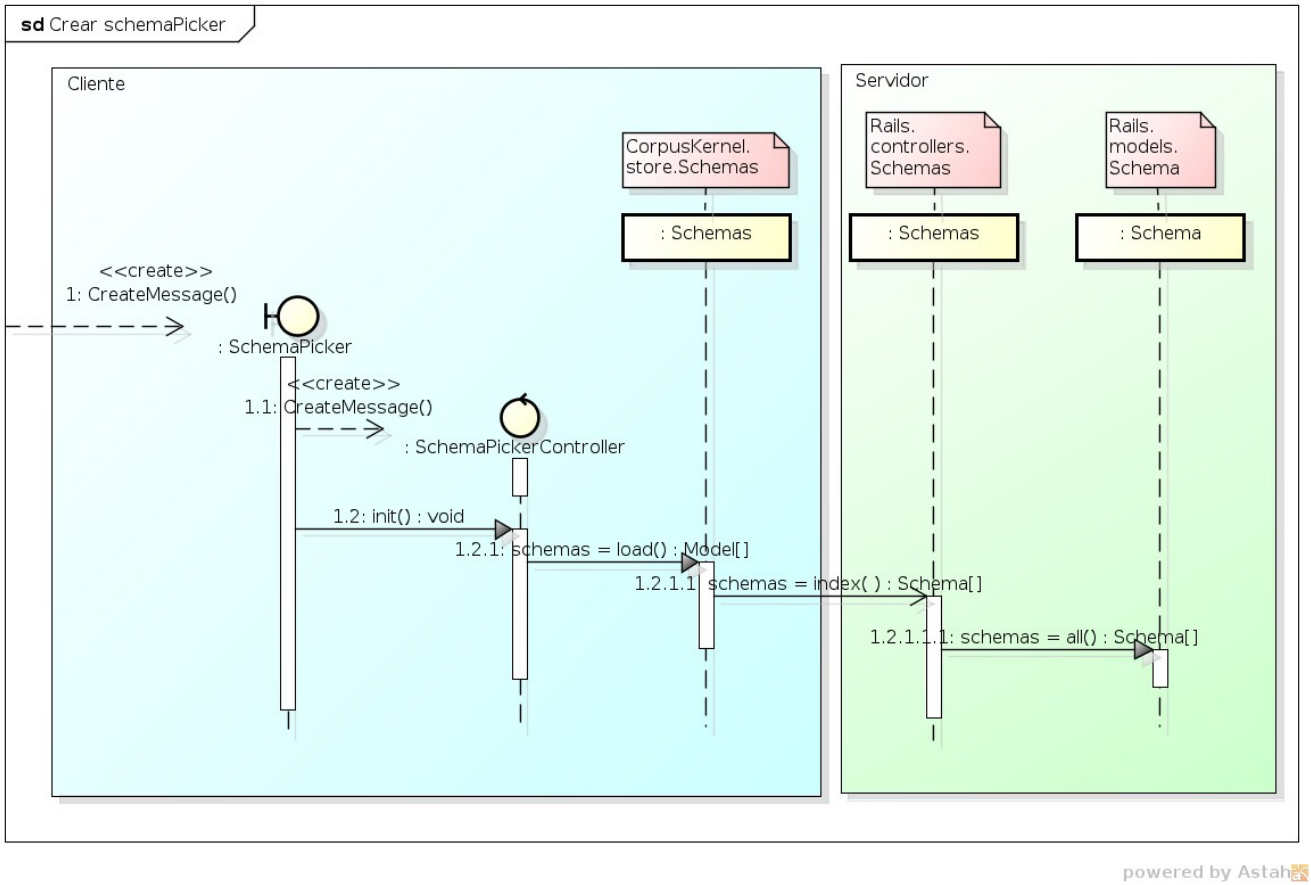
El diagrama del caso de uso *Crear Corpus* muestra mayor detalle que el previo, sin embargo, debido a sus dimensiones, ciertas partes se han obviado.

5.6.1. Caso de uso: Iniciar sesión



5.6.2. Caso de uso: Crear corpus

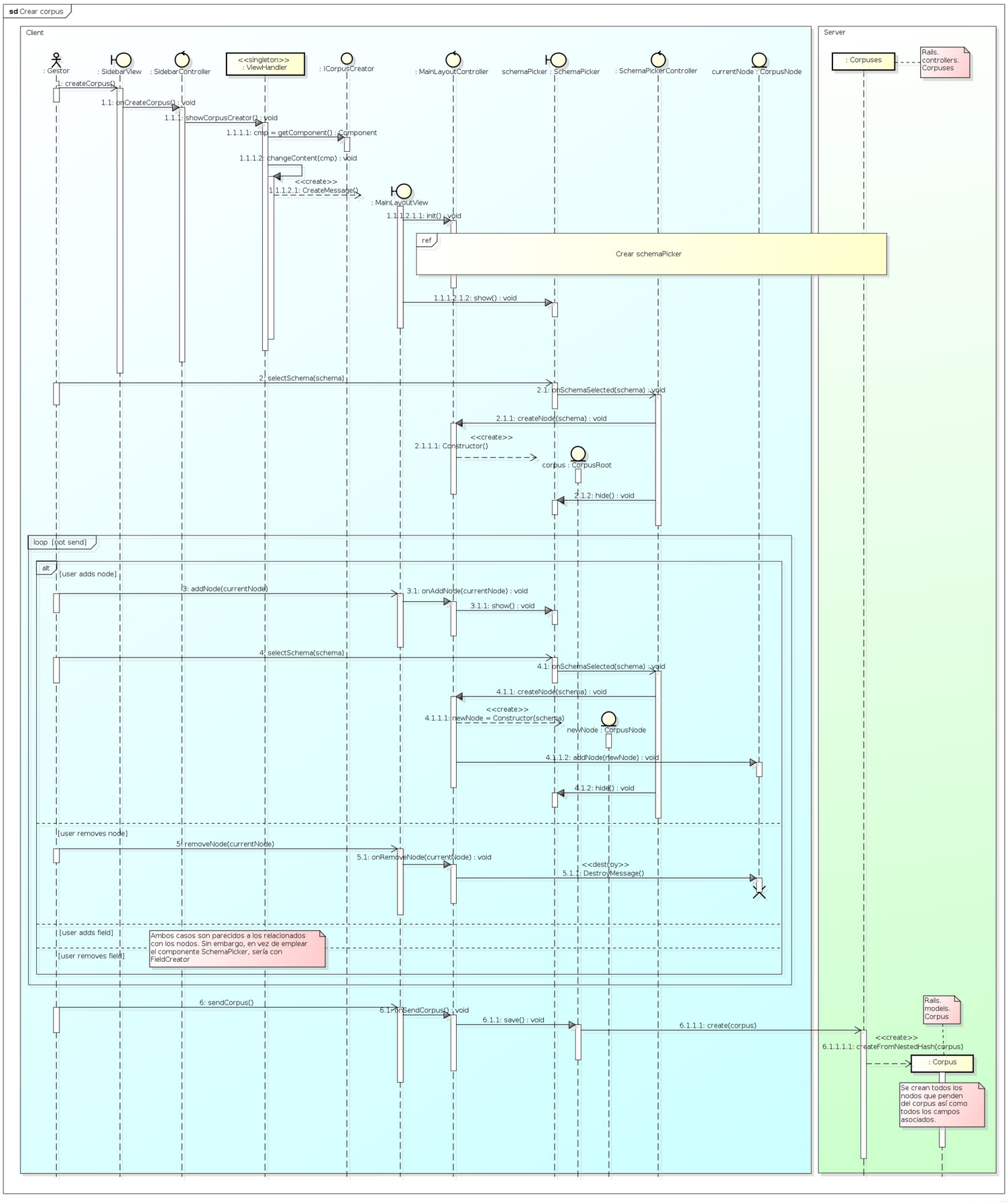
Es necesario mostrar el diagrama de secuencia cuando se crea un nuevo componente *SchemaPicker* con el fin de seleccionar un esquema concreto del sistema. Servirá para poder reducir el tamaño necesario del diagrama relacionado con la creación del corpus.



Como se puede observar, primero se crea el componente, que en este caso es un elemento visual. A raíz de ello, se crea y se asocia la vista al controlador para finalmente iniciarlo.

El proceso es el siguiente: una vez iniciado el controlador, emplea el almacén de esquemas proporcionado por el paquete de CorpusKernel. Al solicitar todos los esquemas existentes del sistema, el almacén internamente se comunica con el servidor y proporciona al componente SchemaPicker todos los esquemas con el objetivo que los pueda mostrar gráficamente al usuario y pueda seleccionar uno en concreto.

En este caso, el esquema seleccionado será el empleado para replicarlo en el nodo seleccionado del corpus.



Capítulo 6

IMPLEMENTACIÓN

6.1. Introducción

En este capítulo se detallan las decisiones tomadas durante la fase de implementación así como las distintas adversidades que surgieron durante dicho proceso y cómo se resolvieron.

También se explicaran diversas funcionalidades que se añadieron adicionalmente al sistema final.

6.2. Dificultades encontradas

6.2.1. Complejidad al desarrollar código mediante ExtJS

Una de las ventajas que ofrece el lenguaje de programación JavaScript es que posee un gran potencial sintáctico. Las posibilidades son muy elevadas, incluso puede simular el comportamiento de otros paradigmas no nativos, como por ejemplo programación orientada a objetos mediante clases. Se debe gracias a su naturaleza de scripting y a tratarse de una programación orientada a objetos mediante uso de prototipos. Además, las funciones son de primera clase, por lo que se permite pasarlas como parámetros a otras funciones (por ejemplo, callbacks). También permite crear funciones anidadas y usar clausuras. Todo ello conlleva a la existencia de numerosos ámbitos (scopes) no triviales a lo largo del sistema.

Al ser un lenguaje tan flexible y por la dificultad de reproducir los distintos ámbitos por un IDE, provoca por ejemplo que sistemas de auto-completado y asistencia típicos de estas herramientas sean prácticamente inútiles dependiendo del tipo de código que se esté generando.

La dificultad encontrada por el alumno fue al emplear el framework de ExtJS y su forma de uso, concretamente en la forma en la que se definen las clases.

ExtJS proporciona un método – Ext.define() – que permite definir fácilmente una clase e internamente simula el comportamiento que se espera de ella (soporte a herencia incluida) mediante la manipulación de diversos prototipos.

Sin embargo, la sintaxis para definir las clases imposibilita que un IDE sea capaz de comprender que se está definiendo exactamente. El siguiente ejemplo define una clase Persona básica:

```
Ext.define(“Persona”, {
    nombre: null,
    diHola: function() {
        alert(“Hola, soy ” + this.nombre);
    }
});
```

El framework internamente confecciona un constructor (Persona) que instancia un objeto con las propiedades definidas mediante otro objeto de configuración ({…}).

Sin embargo, un IDE únicamente ve como se está llamando a un método de aridad dos cuyo primer argumento es un string y el segundo un objeto. Nunca podrá discernir que se esta definiendo una entidad, sino que ve una simple llamada a un método en concreto.

Este efecto hizo que el alumno tuviera que desarrollar toda la parte del cliente sin ningún tipo de asistencia por parte del IDE, como por ejemplo auto-completado, visualización de las propiedades de un determinado objeto... Lo que provocó en ocasiones a duplicación de funcionalidades (el alumno olvidaba la existencia de algunos métodos ya implementados), lentitud en la codificación – al tener que memorizar y recordar el comportamiento de cada clase y método – y finalmente al surgimiento de diversos errores por referencias vacías, objetos con comportamientos distintos a los esperados y valores incorrectos. Este último efecto motivó a que parte de la codificación se realizara no solo en el IDE sino en el debugeador que proporciona los navegadores más comunes.

Afortunadamente en la especificación que sigue JavaScript – ECMAScript – en su versión más reciente, la sexta, concreta una semántica para la definición de clases mediante la palabra reservada “class”. También permite reflejar fácilmente la herencia mediante la palabra reservada “extends”.

6.2.2. Controladores de vista de la parte del cliente

Normalmente en el patrón Modelo-Vista-Controlador se suele asociar un controlador único por vista. Esto permite que exista una relación cerrada entre ambas entidades y permita una mejor facilidad a la hora de codificar la lógica asociada a éstas.

El proceso de implementación de la parte del cliente comenzó con la versión 4.0 del framework ExtJS. Aunque seguía el patrón MVC, la peculiaridad de los controladores de vista es que eran únicos en toda la aplicación, es decir, una sola instancia del controlador gestionaba un tipo indefinido de vistas.

En ocasiones este comportamiento era bastante adecuado sobretodo en la gestión de los recursos, un solo manejador para todas las vistas asociadas. Pero por otra parte, existían continuos conflictos que debían tenerse en cuenta para la correcta ejecución, concretamente en la resolución y manejo de las señales disparadas por las vistas.

Por ejemplo, imaginemos una ventana sencilla que pide las credenciales del usuario. Existe un botón que el usuario debe pulsar para iniciar el proceso de identificación al que identificaremos como “loginButton”.

En el controlador único podremos definir un observador que estará a la escucha de los evento onClick de ese botón. Cuando existe una única instancia de la vista y el usuario pulsa el botón, el controlador atrapa el evento y ejecuta el método asociado.

El problema radica cuando existe más de una instancia de la vista de login. Teniendo dos vistas (o varias) un usuario podría rellenar los campos acreditativos de una única ventana y pulsar su correspondiente botón. ¿Qué pasaría entonces? El controlador al igual que antes se percataría del evento disparado por la vista, sin embargo, ejecutaría el método asociado a dicho evento para todas las instancias de la vista. En la primera vista, con los campos rellenados, su comportamiento sería el esperado; sin embargo, en el contexto de las demás vistas darían fallos ya que los campos no están debidamente cumplimentados. Este fenómeno dificulta con creces la gestión del comportamiento de las vistas ya que para un único evento de una vista concreta se disparan n manejadores con contextos distintos y no solicitados.

¿Por qué ocurre esto? Dentro del DOM, existe una característica que permite realizar consultas para obtener un tipo concreto de elementos. Estas consultas se denominan “selectores” y pueden ser empleadas tanto para definir reglas CSS para un determinado tipo de elementos como para obtener las referencias a los nodos mediante JavaScript. Normalmente los selectores devuelven un vector con diversos elementos.

En el caso anterior, el selector podría ser perfectamente “input.loginButton”, lo que significa que se quiere buscar todos los elementos de tipo *input* que tengan asociada la clase *loginButton*. Cuando existe una única instancia de la vista, sólo se devuelve un vector de longitud uno; sin embargo, cuando se han creado distintas instancias de la misma vista, el vector que se devuelve es igual al número de éstas.

El controlador por tanto, internamente, asocia los métodos de los elementos empleando este tipo de selectores, lo que motivaba que cuando una determinada vista disparaba el evento, el controlador a su vez disparaba el resto.

La primera forma de abordar este comportamiento no deseado fue mediante comprobaciones del estado de cada uno de los elementos. Si no satisfacían el estado esperado, simplemente no se ejecutaba ninguna operación sobre ellos. Esto dio lugar a una complejidad en los métodos asociados a la gestión de señales del controlador. Pronto se vio que la solución no era para nada aceptable.

El alumno empezó a considerar la idea de confeccionar un nuevo tipo controlador, reutilizando en lo posible el comportamiento del controlador nativo para intentar mitigar posibles estados inestables que podrían surgir, los cuales podrían afectar negativamente el comportamiento del framework.

La idea fue la siguiente, seguir empleando los selectores para recuperar los elementos hijos de la vista pero especificando un único contexto asociado a una vista concreta. Una vista por tanto, cuando se conformaba, debía crear una instancia del controlador y asociarse. Una vez realizada la asociación, el controlador sólo escuchaba eventos definidos por los selectores de los elementos hijos o por la propia vista.

Para conseguir este comportamiento, el alumno tuvo que estudiar y analizar el código interno del framework a la hora de gestionar los eventos y controladores. Esto hizo que la clase creada tuviera el comportamiento deseado por el alumno sin llegar a producir efectos laterales en el framework.

La versión 5.0 de ExtJS además diversas características añadieron la clase “ViewController”, la cual era un controlador asociado únicamente a una instancia de un tipo de vista. El nuevo controlador realizaba el comportamiento que había conseguido definir el alumno además de integrarlo nativamente en el framework y proporcionar un conjunto de interesantes funcionalidades. Todo ello motivó a actualizar de versión del framework empleado para la creación del cliente.

6.2.3. Cambio de versión en el cliente

Debido al gran número de mejoras que existían entre la nueva versión (5.0) respecto con la que se empezó a desarrollar (4.0), el alumno consideró que era más que adecuado actualizar el framework y así beneficiarse de las diversas ventajas que ofrecían.

Aunque la actualización no fue del todo sencilla. Por ejemplo, se tuvo que portar un gran número de controladores que el usuario había creado al nuevo tipo de controlador que se ofrecía en esta nueva versión. Además amoldarse a las buenas prácticas que habían especificado para emplearlos correctamente.

También surgieron fallos y errores en diversos puntos del código ya que o bien el comportamiento había cambiado sutilmente o la forma con la que se hacía ya no estaba admitida.

6.2.4. Definición del modelo

Tanto en el cliente como en el servidor existieron grandes dificultades para entender cómo se debía especificar correctamente las entidades y relaciones del modelo.

En el caso del servidor, que se empleaba los *active records* del framework Ruby on Rails, fue un proceso arduo y complicado que necesitó recurrir a diversas documentaciones y artículos relacionados con la gestión de datos, además de invertir un considerable tiempo de experimentación para llegar a interiorizar los conceptos.

Uno de los problemas que encontró el alumno a la hora de afrontar el aprendizaje de Ruby on Rails es su máxima de "convención sobre configuración". El lenguaje de programación Ruby permite la metaprogramación – código que genera código – y posibilita la capacidad de crear de forma transparente y automática nueva lógica al sistema. Todos estos procesos son totalmente ajenos al desarrollador, hasta el punto que ciertas operaciones simples que realiza provocan grandes efectos en el sistema. La sensación que daba al alumno es que ciertas cosas salían de la nada. Esto es lo que provoca la convención sobre configuración, existe una forma universalmente aceptada de operar y con ello se consigue simplificar la codificación; pero a su vez, dificulta el entendimiento a nuevos desarrolladores que sean ajenos a estas convenciones que se han ido perfilando a lo largo del desarrollo de Ruby on Rails.

A diferencia de la parte del servidor, que existía una amplia documentación, el framework ExtJS era más laxo a la hora de explicar con detalle la gestión de los datos. Aunque en general se explicaba correctamente su gestión, diversos aspectos concretos (como definición correcta de relaciones muchos a muchos) apenas tenían documentación relacionada. Esto llevó al alumno a experimentar con el código fuente del framework y hacer un estudio de la parte que confeccionaba ExtJS para este cometido en el DOM.

6.2.5. Gestión de la asincronía

JavaScript es un lenguaje de scripting que únicamente se ejecuta en un único hilo y es por naturaleza síncrono. Existen prácticas, por ejemplo los Workers, que permiten ejecutar distinto código en varios hilos. Al ser un lenguaje orientado a la web, permite realizar peticiones HTTP, las cuales, añaden asincronía al sistema y hace más complicado su codificación.

Sin embargo, JavaScript permite pasar como parámetros de funciones otras funciones que pueden actuar como callbacks. Esto permite que secciones asíncronas puedan ser manejadas apropiadamente con este tipo de funciones. Aunque un uso inadecuado de callbacks pueden conllevar a lo que se denomina *callback hell*. Básicamente es una secuencia de callbacks anidadas que complican la legibilidad y entendimiento del código.

Dada la naturaleza de la aplicación web que se quería desarrollar, siguiendo el patrón single-page, es inevitable que desde el código JavaScript tenga que realizar un gran número de llamadas asíncronas para obtener ciertos recursos y datos. A su vez, el framework proporciona mecanismos para satisfacer las dependencias del código, pero aún así existe una cierta complejidad en su gestión. La mejor forma de afrontar este tipo de fenómenos es considerar conceptualmente que JavaScript es puramente asíncrono. Lo que motivó al alumno a desarrollar diversas técnicas y mecanismos para reducir esta complejidad.

Uno de los mecanismos más utilizados fue emplear callbacks, en lo posible de un único nivel de profundidad. Sin embargo, existían ciertas tareas que se desarrollaban en paralelo y que debían ser adecuadamente orquestadas. Para resolver esta situación el alumno desarrolló unas entidades que

simulaban el comportamiento de un semáforo; gracias a ello se podía especificar una callback que sólo se ejecutaba si n veces se desbloqueaba el cerrojo definido.

En algunas partes del código incluso el alumno experimentó con las nuevas características de ECMAScript 6, en concreto con las promesas (promises).

6.2.6. Reducción de consumo de datos y descarga perezosa

Uno de los fuertes de ExtJS es la gestión de los datos descargados del servidor relacionados con objetos del modelo. El framework permite definir modelos que traducirán automáticamente la respuesta recibida del servidor en entidades válidas del modelo. Además proporciona almacenes que permiten su guardado y gestión.

Cuando se crea una instancia de un modelo – denominada registro (record) – está se marca de forma automática como fantasma, es decir, no existe dicho registro en el servidor y en el caso de enviarla se hará mediante la operación de creación definida.

Por otra parte, el framework puede descargar distintos registros del servidor, lo que conlleva a que existan representaciones locales de los objetos del dominio del servidor en cada cliente. Debe existir por tanto una política adecuada para que diversas modificaciones realizadas sobre ellos sean finalmente consistentes en el sistema.

A su vez, los registros pueden tener otros registros asociados si así está definido en las relaciones del modelo. El servidor puede tener dos tipos de comportamiento cuando se pide un registro concreto: proporcionar todos los registros asociados con toda su información o únicamente devolver el identificador único de cada registro asociado.

Con la primera vertiente, se aumenta considerablemente el tráfico de datos – ya que se duplicaría en muchas ocasiones información – pero se consigue que todos registros dependientes estén disponibles tras la petición.

La segunda vertiente consigue reducir el tráfico de datos ya que sólo proporciona la información completa del registro solicitado y únicamente los identificadores únicos del los registros asociados. Si algún registro asociado ya estaba previamente descargado en el cliente, el framework proporciona toda la información que dispone de él. Gracias a ello se reutiliza eficientemente los diversos registros existentes y que están almacenados en el sistema.

Por ejemplo, un usuario posee un rol. Si al principio se han pedido todos los roles del servidor, cuando posteriormente el cliente pida el usuario y éste indique que su rol tiene un identificador determinado, el framework puede automática relacionar ambos registros y permitir su navegabilidad. Sin embargo, si no se han pedido los roles y se solicita el usuario éste tendrá como rol un simple identificador. Si posteriormente se quiere acceder al rol, se tendría que realizar una petición HTTP, lo que implica que sea una llamada asíncrona y tenga que ser correctamente tratada.

La carga perezosa por tanto conlleva muchas ventajas pero también obliga a ser cautelosos a la hora de acceder a la información y añade complejidad en el sistema.

Un efecto negativo del framework del cliente (ExtJS), es que considera un registro como no fantasma (representación real de una instancia concreta de una entidad del modelo del servidor) incluso aunque

éste solamente haya proporcionado su identificador. Cuando se quiera disponer de toda la información el cliente cree que posee toda – por tanto no realiza la petición – y devuelve el resto de campos que compone el modelo como null.

La solución que dio el alumno fue que desde el servidor siempre y cuando ensamble una asociación a un registro concreto de forma perezosa (sólo con su identificador único) se añada un nuevo atributo “lazy” definido como true.

En la parte del cliente se modificó la clase abstracta de la que se especializan los registros y se modificó su comportamiento, siempre y cuando esté definido el atributo lazy como true, es necesario forzar una actualización del registro en el cliente.

6.3. Funcionalidad adicional añadida al sistema

Durante la fase de implementación se fue observando un par de funcionalidades que su adición al sistema supondrían un beneficio positivo.

6.3.1. Soporte de internacionalización en el cliente

El alumno entendió que aunque en los requisitos iniciales no se había contemplado la posibilidad de crear una aplicación web que pudiera ser traducida fácilmente a otros idiomas, la naturaleza del problema motivaba a que así fuese. Esto se debe principalmente a que una parte de los usuarios del sistema tendrá diversas lenguas nativas distintas al español, y al brindar esta posibilidad, permitirá a los alumnos emplear el sistema en su idioma nativo.

No se percató hasta la fase de implementación que el framework con el que se estaba desarrollando la aplicación web no contaba con un sistema i18n (internacionalización) para permitir la traducción y adecuación de su contenido a distintos idiomas.

La forma que tiene el framework para admitir sencillas traducciones es especificar los mensajes como atributos de la vista y generar tantas clases – con dichos atributos traducidos – como idiomas se quiera contemplar. Sería mediante una sobre-escritura de clases lo que permitiría permutar entre los diversos idiomas. Esto implica que por cada vista se tengan que hacer n clases adicionales para cada idioma, lo que conlleva a que sea complicado gestionar las distintas versiones en una aplicación de cierto tamaño. También imposibilita la existencia de un recurso centralizado que posea los mensajes y que ayudaría con creces en las tareas de traducción.

Sin embargo esta aproximación es superficial y no puede considerarse como una solución real al problema en cuestión. Por esa razón el alumno ideó un sistema propio de i18n para el framework de ExtJS.

Los objetivos que se buscaban además con este nuevo sistema eran los siguientes:

- Todos los mensajes de la vista propuestos a traducción deberían estar en un único fichero. Este formato ayudaría a emplearlo en otras aplicaciones relacionadas con la traducción de ficheros.
- El enlazado (bindeado) de los mensajes con la vista, dependiendo del idioma seleccionado, debe ser lo más sencillo para el desarrollador.
- El conjunto de mensajes de una clase deben estar completamente jerarquizados en el fichero atendiendo a una ruta que relaciona unívocamente a la clase en cuestión.

Se empleó el patrón MVVM para conseguir un enlazado automático y transparente entre los distintos elementos y su mensaje asociado. Esto fue posible gracias a la nueva versión ExtJS 5.0 que añade este nuevo patrón de forma nativa.

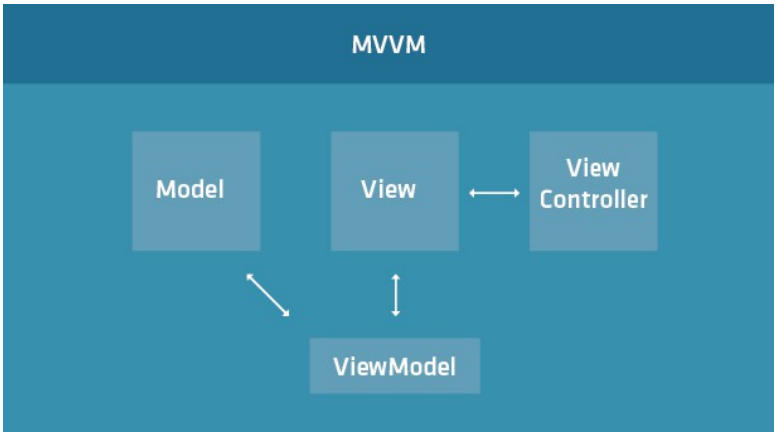


Figura 38: Patrón MVVM de ExtJS

Este patrón añade una nueva entidad respecto al patrón MVC, el ViewModel. Este elemento relaciona el modelo con la vista mediante un enlazamiento de los datos, en otras palabras, en vez de definir los contenidos imperativamente (mediante setters del controlador, es decir, el cómo) se realiza de forma declarativa (qué debe haber). De esta forma se puede especificar en cada vista que en cierto elemento se espera un tipo de mensaje y ya será el ViewModel el encargado – dependiendo del idioma seleccionado – de resolverlo con el mensaje correspondiente.

Todo el mecanismo se implementó en una única clase denominada *I18n* y fue añadida al paquete *CorpusKernel*. Cada clase que contiene los mensajes traducidos de un componente se denominan recursos locales y deben ser nombrados con el identificador del lenguaje.

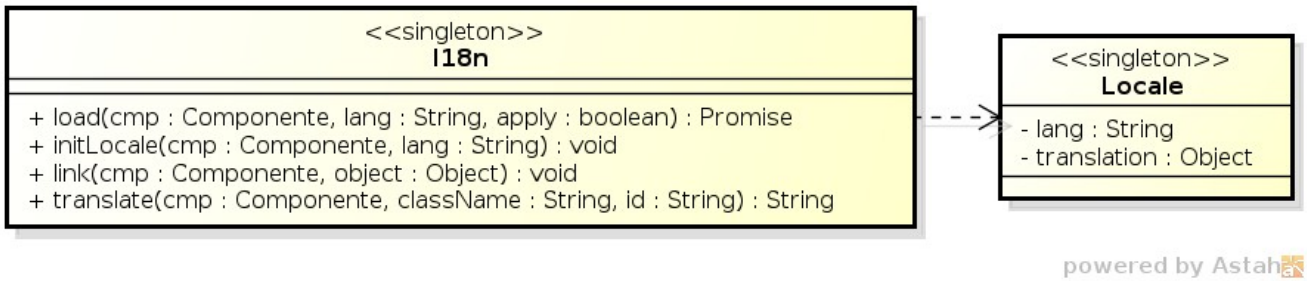


Figura 39: Clases relacionadas con el sistema i18n

La clase Locale es la encargada de almacenar la traducción de los mensajes. Todo conjunto de mensajes relacionados de una vista tienen que estar unívocamente identificados empleando el namespace de ésta.

El siguiente ejemplo muestra el recurso locale en inglés del componente Field.

```
Ext.define("CorpusKernel.component.field.locale.en", {

	singleton: true,

	lang: "en",

	translation: {
		component: {
			field: {
				Field: {
					toggleText   : "Allow override",
					addText      : "Add a new subfield",
					deleteText   : "Delete this field",
					infoText     : "Information about this field",
					infoTitleText: "Information"
				}
			}
		}
	}

});
```

Como se puede ver se ha especificado un conjunto de mensajes con la siguiente ruta “component.field.Field”. Dicha ruta especifica el namespace de la vista a la que le corresponde dichos mensajes. Para crear un locale para el idioma japonés, tan solo sería necesario replicar el locale, renombrarlo como jp.js y traducir los strings que contiene. El atributo lang es opcional ya que el tipo de idioma se puede inferir del nombre del fichero.

La clase i18n por otra parte posee los métodos necesarios para realizar la carga y bindeado de los datos de los recursos locales.

El método *load*, recupera el recurso locale correspondiente al lenguaje seleccionado (parámetro *lang*) del servidor. El parámetro *apply* permite decidir si se quiere aplicar el nuevo recurso locale al ser descargado, por defecto es true.

Como se puede observar, el método devuelve una promesa, es decir, un objeto de JavaScript especializado en la gestión de tareas asíncronas. Gracias a ella se simplifica el código relacionado con los estado internos de la tarea en cuestión, o bien tiene éxito o fallo, pero no ambos.

El método *link*, sirve para enlazar una vista determinada al recurso locale existente en el componente. Este método se debe llamar siempre cuando se inicializa el ViewModel de una vista que posea traducciones ya que de otra forma no existiría la relación entre ambos. Internamente bindea el método translate a la instancia de la vista y crea un alias *t*.

Finalmente, el método *translate*, permite obtener un mensaje concreto (identificado por el parámetro *id*) de una clase determinada (identificada por el parámetro *className*) del recurso locale del componente proporcionado como primer parámetro.

Volviendo al ejemplo anterior de Field, una vez definido su locale, tan solo es necesario especificar el ViewModel que enlaza los datos de la vista con el modelo.

```
viewModel: {
        formulas: {
                toggleText: function() {
                        return this.getView().t("toggleText");
                },
                addText: function() {
                        return this.getView().t("addText");
                },
                deleteText: function() {
                        return this.getView().t("deleteText");
                },
                infoText: function() {
                        return this.getView().t("infoText");
                }
        }
}
```

Como la vista ya está vinculada tan solo es necesario llamar al método *translate* o como en este caso a su alias *t*. Cuando se inicie una nueva instancia de la vista, automáticamente pedirá los mensajes al recurso locale y los mostrará en pantalla.

6.3.2. Soporte para componentes en el cliente

Uno de los requisitos que tenía que contar el framework del cliente era la posibilidad de desarrollar componentes independientes para así poder reflejar las especificaciones definidas en la fase de diseño.

ExtJS emplea el término componente para referirse a elementos visuales que poseen una funcionalidad concreta. Por ejemplo, un textfield, además de ser un elemento input del DOM, posee una lógica que le permite interaccionar con otros componentes así como proporcionarle funcionalidades adicionales y avanzadas. También se denominan widgets.

Por esa razón, el alumno empezó a diferenciar ambos sentidos de la palabra componente. Por una parte estaría los *xcomponentes* que hacen referencia a los componentes visuales del framework, y *wcomponentes* a los que se entiende en el desarrollo basado en componentes.

Al principio se comenzó a definir los distintos componentes del sistema de forma independiente, es decir, cada uno de ellos tenían su propia lógica que conseguía independizarlos y encapsularlos del resto del código, sin embargo, esto llevó a un aumento considerable en líneas de código así como a una discrepancia entre ellos.

La solución pasaba por la unificación del término en una clase que aglutinara todas las características que se esperaba de un componente. Gracias a ello sólo existía un código relacionado con la conformación de estas entidades y así se conseguía además reducir la complejidad y prueba del sistema.

Lo que se consiguió fue una abstracción plena del código relacionado con el componente frente al resto, gracias a la eliminación de cualquier dependencia externa y a la publicación de una API con la que el resto de componentes y la propia aplicación debe interaccionar.

El principal objetivo era conseguir un comportamiento muy parecido al de *Ext.app.Application* pero manteniendo la estabilidad y la consistencia de la aplicación. Sin embargo, tal y como está diseñado ExtJS sólo puede existir una única instancia cuya clase herede de *Ext.app.Application*. Ya que por ejemplo si existiera más de una [instancia] la gestión global de rutas (identificadores de fragmento de la URI) sería llamada por cada una de éstas lo que provocaría hilos de ejecución erróneos o ciclos infinitos.

También realiza operaciones indeseadas como la gestión de los profiles (adecuaciones del sistema dependiendo del dispositivo).

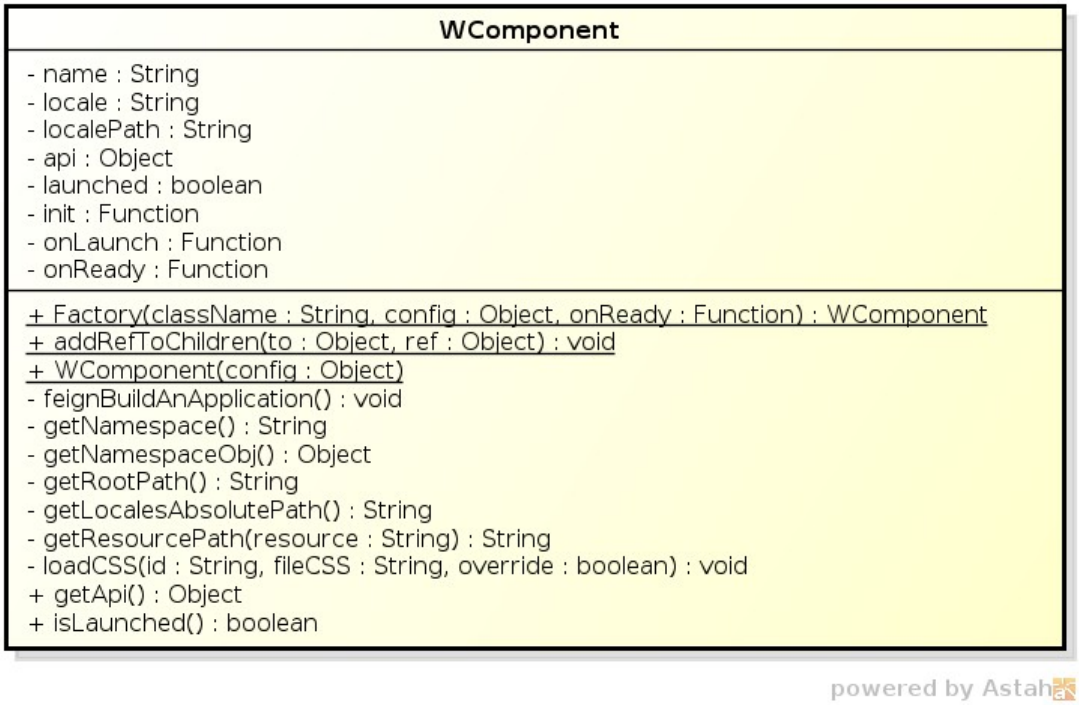
Ext.app.Application es una especialización de *Ext.app.Controller* y éste a su vez de *Ext.app.BaseController*. Es por ésta razón por la que WComponent heredó de *Controller* ya que permitía emplear su lógica sin necesidad de modificar la clase *Application*.

El constructor de BaseController permite emplear la clase Observable y así ser capaz de gestionar y manipular los eventos de la aplicación. Esta funcionalidad permite por tanto escuchar eventos propios del componente creado y dispararlos.

Por otra parte, el constructor de *Controller* inicializa sus getter y llama al constructor de su clase padre, *BaseController*. Lo más interesante del prototipo *Controller* es que es capaz de gestionar automáticamente las dependencias definidas en las configuraciones *models*, *views*, *stores* y *controllers*.

Todo ello simplificó el código necesario respecto a si se hubiera realizado sin ninguna herencia. Sin embargo, la construcción de los *Ext.app.Controller* realmente la realiza la clase que hereda de *Ext.app.Application*. Es por tanto, que el método constructor de WComponent tiene que realizar un comportamiento parecido al que realiza *Ext.app.Application* en este aspecto para impedir que surjan errores o avisos por parte de ExtJS.

Además, se añadió soporte nativo al sistema i18n creado por el alumno – el cual fue explicado en la anterior sección.



El atributo *name* sirve para especificar un identificador común para el componente en el sistema.

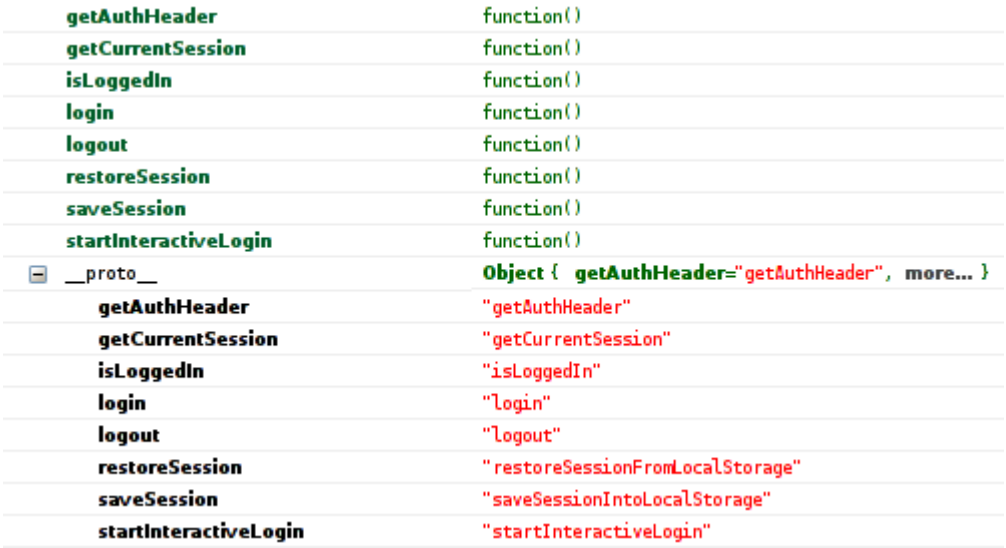
El atributo *api* es un array asociativo con las referencias a los distintos métodos que se quiere publicar dentro de la API del componente. También acepta especificar rutas como valores del array, lo que permite no solo publicar métodos pertenecientes al propio componente, sino incluso métodos de objetos definidos como propiedades de la clase.

Imaginemos un objeto A que tiene un método foo, y un componente C que tiene un atributo myObj referenciado a A. Podremos publicar el método foo del objeto A como API del componente si especificamos la siguiente ruta: “myObj.foo”.

Un ejemplo de una especificación de una API empleando rutas:

```
api: {
        getAuthHeader        : "getAuthHeader",
        getCurrentSession    : "getCurrentSession",
        restoreSession       : "restoreSessionFromLocalStorage",
        saveSession          : "saveSessionIntoLocalStorage",
        isLoggedIn           : "isLoggedIn",
        login                : "login",
        logout               : "logout",
        startInteractiveLogin: "startInteractiveLogin"
},
```

Cuando se crea una instancia del wcomponente (theObject) en cuestión, automática resuelve las rutas definidas en la propiedad api. Si llamamos al método *getApi* de *theObject* después de la instanciación obtenemos el siguiente objeto:



Como se ve, es un objeto cuyas rutas ya han sido totalmente resueltas ya que son referencias a funciones concretas del componente. Además permite, a través de su prototipo, conocer el objeto original que se definió en la clase.

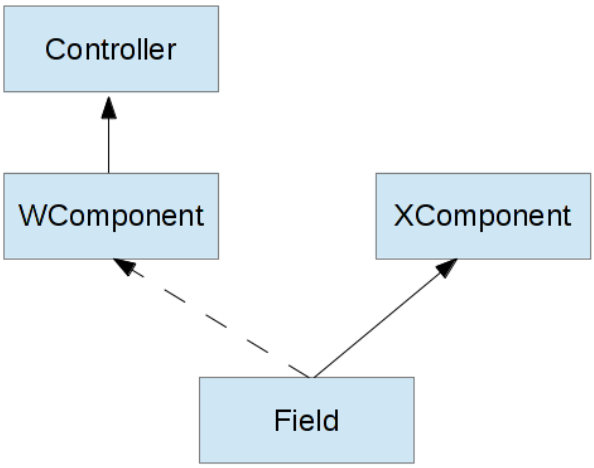
Para llamar a un método de la API tan solo sería necesario llamarla de la siguiente forma:

theObject.getApi().isLoggedIn();

El atributo *locale* permite especificar el locale que se asociará por defecto cuando se crea una nueva instancia. Por ejemplo, si su valor inicial es “es”, buscará el recurso locale correspondiente al idioma español y lo enlazará al objeto instanciado.

Si posteriormente se llama al método *setLocale(“en”)*, automáticamente se cargará el recurso locale correspondiente al idioma inglés.

El método *Factory* sirve para resolver la pseudo-herencia múltiple de los componentes. Cualquier clase puede heredar de otra clase, sin embargo, si quiere constituirla como wcomponente es necesario que también herede la funcionalidad de éste. La herencia de la funcionalidad del wcomponente se realiza mendiante mixin, lo que únicamente copia las propiedades definidas en éste al objeto original. De esta forma se resuelve cualquier tipo de ambigüedad que surge cuando se realizan herencias múltiples.



En este caso, la clase Field es una especialización de XComponent (componente visual de ExtJS) pero además añade la funcionalidad contenida en WComponent. Por tanto, Field se convierte en un wcomponente que hereda de xcomponente.

6.4. Software empleado

A continuación se enumeran las distintas aplicaciones y programas que se emplearon para realizar el sistema y que son necesarios para su correcto funcionamiento.

6.4.1. Cliente

- FontAwesome: iconos codificados como fuente. Se empleó la versión 4.3.0.

  https://github.com/FortAwesome/Font-Awesome

- ExtJS: como framework del cliente. Se comenzó con la versión 4.2.1.883 y se mantuvo durante diez meses. Posteriormente se actualizó a la versión 5.0.0.970 y tras las diversas actualizaciones pertinentes se alcanzó la versión definitiva 5.1.1.451.

  http://www.sencha.com/legal/gpl/

- AudioRecorder: permite realizar grabaciones de audio mediante HTML5 y visualizar las ondas. Desarrollado por Matt Diamond bajo la licencia MIT.

  https://github.com/mattdiamond/Recorderjs

- Gema – JSDuck: generador de documentación para el framework ExtJS. Versión: 5.3.4. https://github.com/senchalabs/jsduck

6.4.2. Servidor

- Rbenv: gestor de entornos para Ruby. Versión 0.4.0-146-g7ad01b2 https://github.com/sstephenson/rbenv
- Ruby: lenguaje primordial de Ruby on Rails. Se empleó la versión 2.1.3.
- Gema – Rails: framework para hacer servidores webs. Se utilizó la versión 4.1.6.
- Gema – Sqlite3: sistema de gestión de bases de datos. Versión: 1.3.9.

Capítulo 7

DEPURACIÓN Y PRUEBAS

7.1. Introducción

Las pruebas son una parte importante para proporcionar información objetiva sobre la calidad de un artefacto software desarrollado. Su objetivo principal es encontrar posibles errores y defectos surgidos durante la fase de implementación. Para ello se realizan un conjunto de posibles escenarios, o batería de pruebas, que definen las distintas pruebas que el software desarrollado debe pasar correctamente para poder así concluir que es válido.

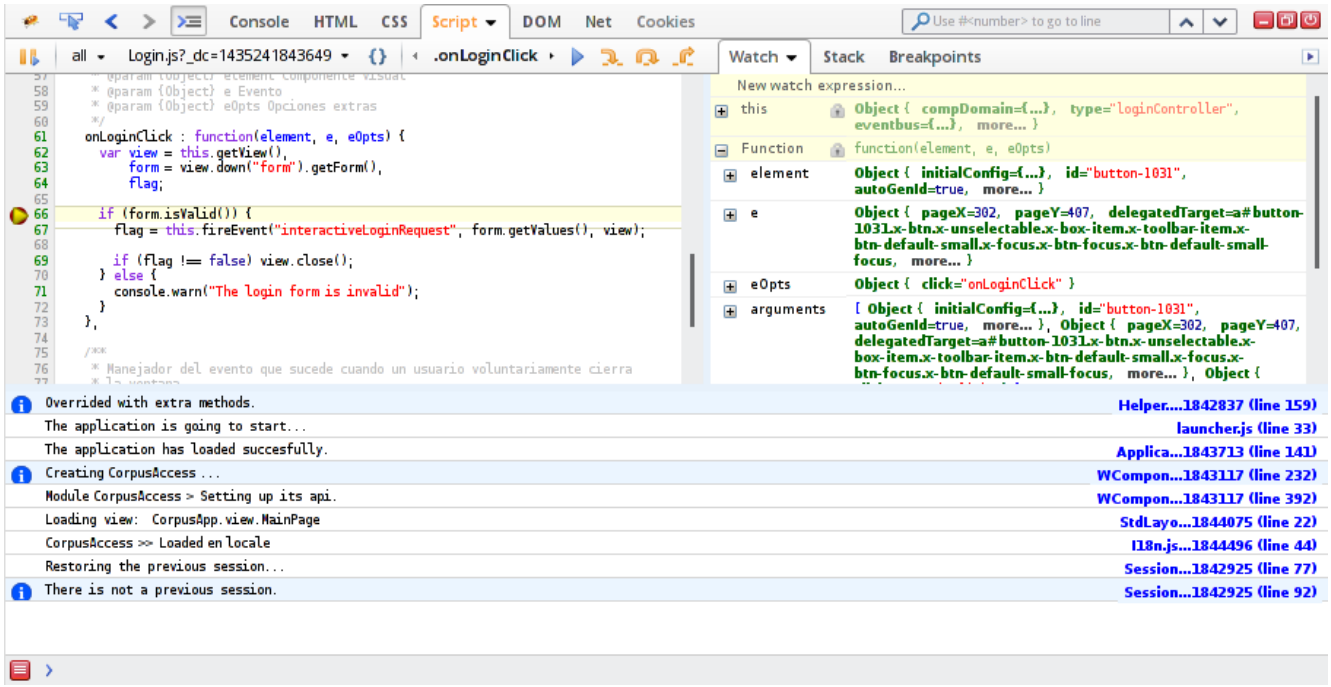
Sin embargo, en este desarrollo no se realizó ningún test unitario o de integración. Es decir, no se realizó una fase centrada exclusivamente en este apartado. Aunque si que se emplearon diversas técnicas y herramientas durante todo el proceso de implementación relacionadas con este tema así como el desarrollo y realización de baterías de pruebas sencillas.

Dado que todo el código está desarrollado por una única persona, durante el proceso de codificación también se vio obligado a realizar distintas pruebas “ad-hoc” para comprobar su validez y robustez. Dichas pruebas fueron de carácter temporal y nunca se documentaron.

7.2. Debugeador

Dada la naturaleza del framework ExtJS empleado en el cliente así como el desarrollado de aplicaciones web, el desarrollador tuvo que usar activamente un debugeador de JavaScript.

El debugeador elegido fue Firebug – el cual es una extensión del navegador web Firefox – que posee un conjunto de características que le hacen ser el mejor dentro del ámbito opensource.



Como se puede observar en la figura anterior, en la línea 66 se ha añadido un punto de parada. Estos puntos sirven para detener voluntariamente la ejecución del código en una determinada línea (incluso permiten que esté condicionada a una guarda). Cuando la ejecución está detenida por un *breakpoint*, permite analizar el valor de todas las variables que componen el ámbito de la función (área de la

derecha). Incluso en la parte inferior se proporciona una consola para poder inyectar código JavaScript en ese mismo ámbito, lo que permite realizar operaciones de comprobación más avanzadas e incluso programar directamente y conocer su resultado inmediato.

Finalmente en la pestaña “Stack”, se puede ver todo el hilo de ejecución que se ha realizado hasta el momento, es decir, la traza de todas las llamadas de las funciones padres. Lo más interesante además de conocer la traza real del sistema, es que se puede cambiar el ámbito actual a otros ámbitos anteriores proporcionando así una gran herramienta para conocer las causas de un comportamiento errático.

La extensión también permite registrar todas las peticiones que realiza la página durante su carga. Esta parte es esencial para este tipo de aplicaciones de single-page. Para conseguir un mejor rendimiento, es preferible ir cargando aquellos recursos necesarios durante la marcha. Por esa razón, conocer como se van pidiendo es esencial para comprobar que el código se está ejecutando correctamente. En la siguiente figura se muestra el contenido que se envía al servidor cuando se emplea su API para acreditarse ante el sistema.

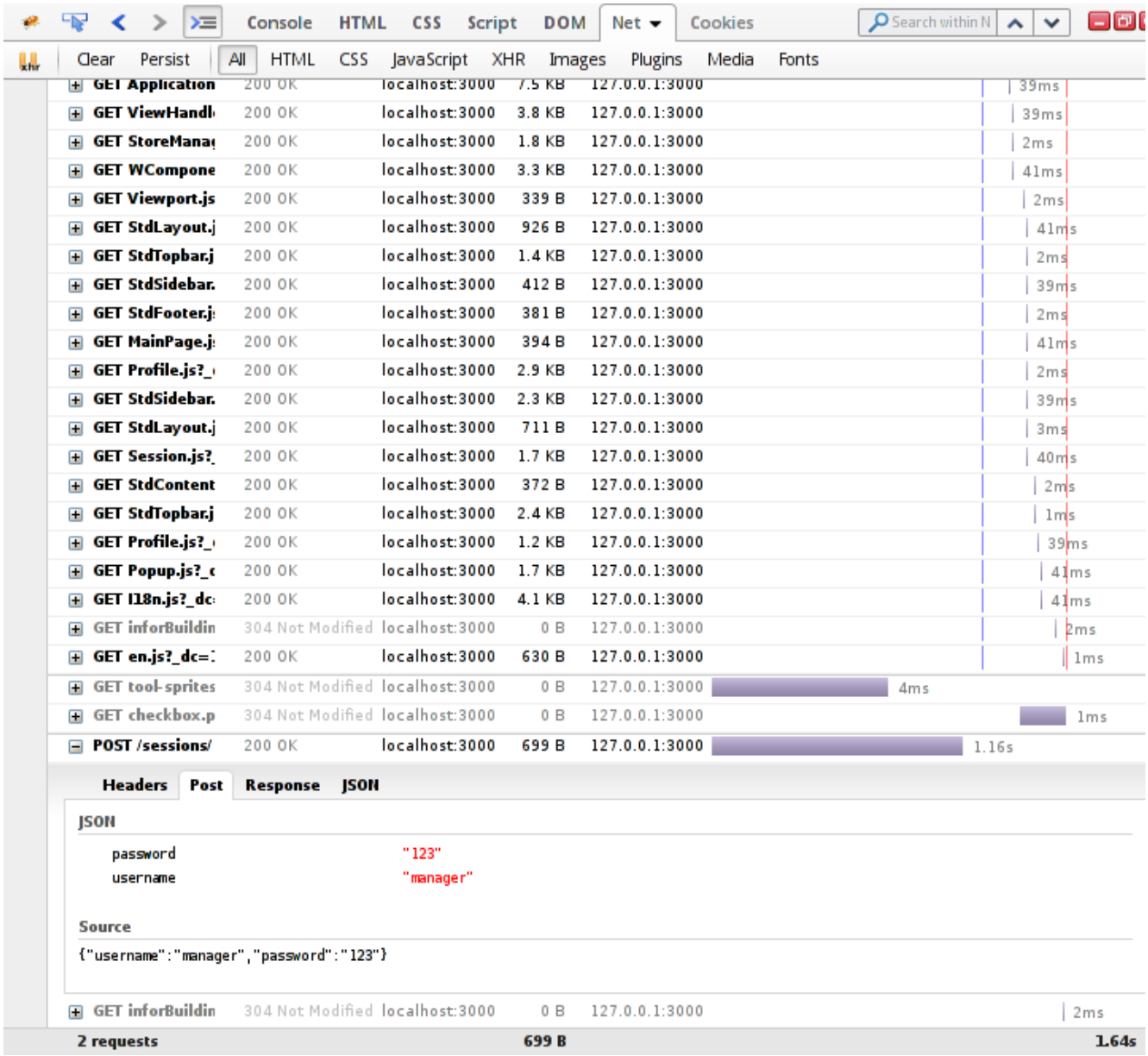
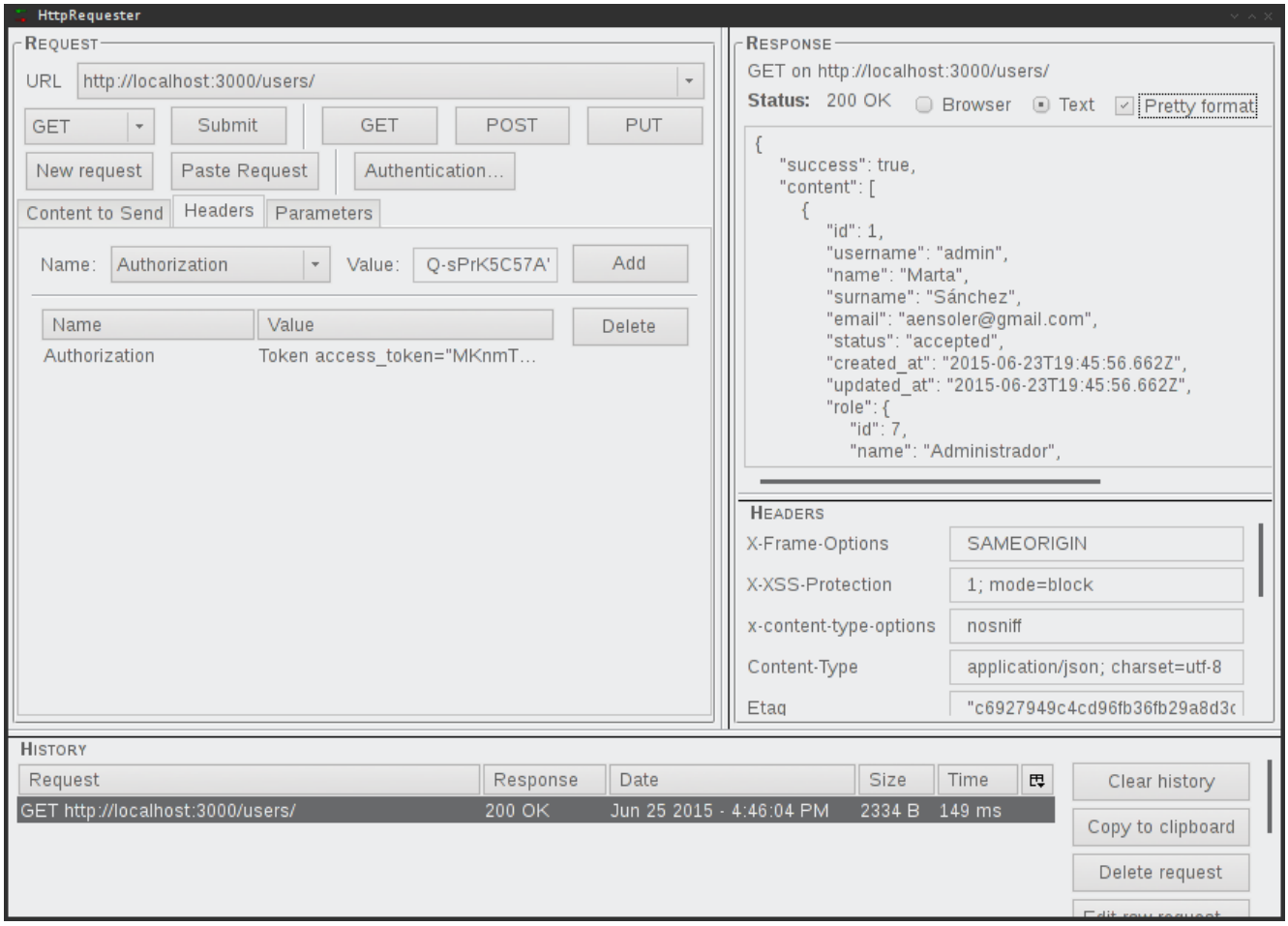


Figura 44: Post de una petición empleando Firebug

Como se puede ver, existe un gran número de peticiones de tipo GET. Esto es porque cada clase se especifica en un fichero JavaScript distinto y debe ser debidamente descargado por el cliente. Poder conocer el orden y los tiempos de descarga en este caso es crucial porque en caso de error supondría que una clase por ejemplo no estuviera disponible en el sistema.

Para realizar las primeras pruebas sobre la API Rest del servidor se empleó la extensión de Firefox "HttpRequester". Mediante una sencilla interfaz permite configurar distintas peticiones HTTP. Esta herramienta fue de vital importancia cuando se estaba constituyendo la API ya que proporcionaba toda información necesaria para comprobar si se estaba comportando correctamente o no.



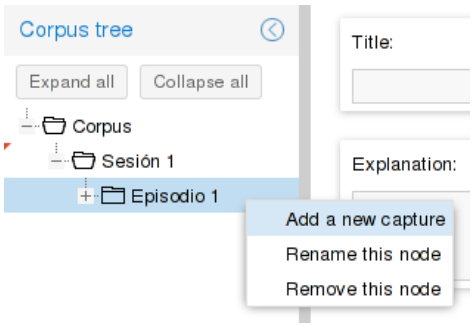
Sin embargo, cuando el cliente fue creciendo, se prescindió de HttpRequester ya que las comprobaciones necesarias o bien se realizan mediante la aplicación web o bien se analizaban empleando Firebug.

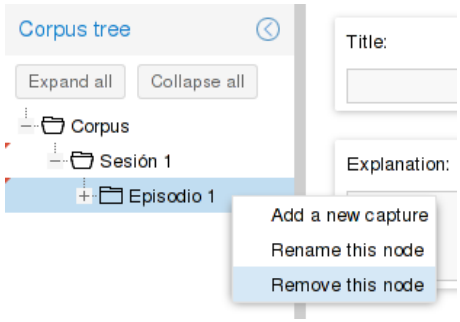
7.3. Batería de pruebas

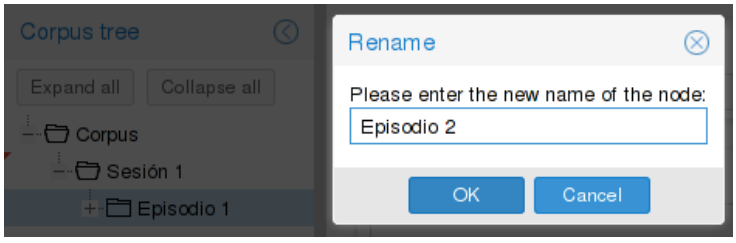
A continuación se exponen los distintos casos de pruebas definidos junto con su resultado obtenido al ejecutarlas sobre el sistema.

CP-01	
Descripción	Se crea un nuevo nodo del corpus mediante la interfaz y se selecciona un esquema concreto.
Entrada	Nuevo nodo y esquema seleccionado.
Resultado esperado	Se duplican todos aquellos campos definidos en el esquema y se añaden al nodo en cuestión. Además, todos ellos tendrán especificado que sobreescriben a los nodos del esquema.
Resultado obtenido	Correcto.

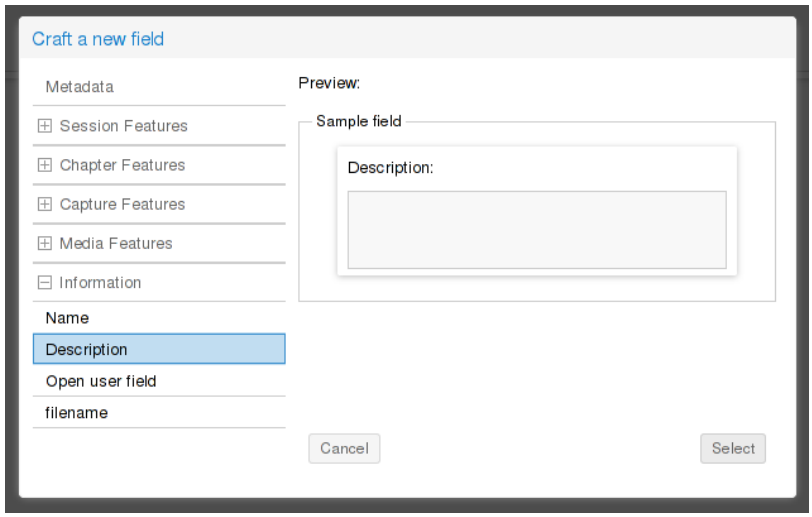
CP-02

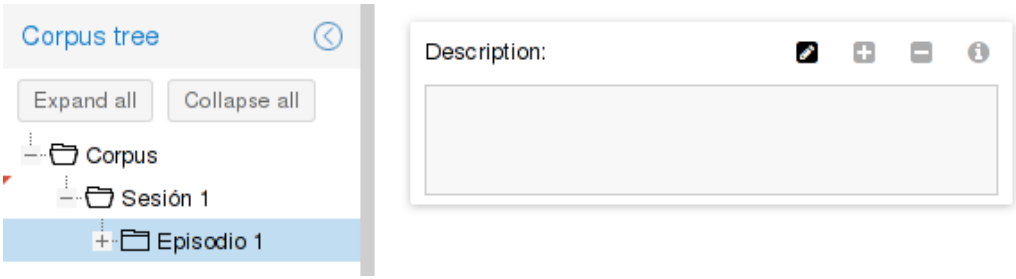
Descripción	Se crea un nuevo nodo del corpus mediante la interfaz.
Entrada	
Resultado esperado	Se añade visualmente al árbol jerárquico de la interfaz y se resuelven correctamente las relaciones entre nodos padres e hijos.
Resultado obtenido	Correcto.

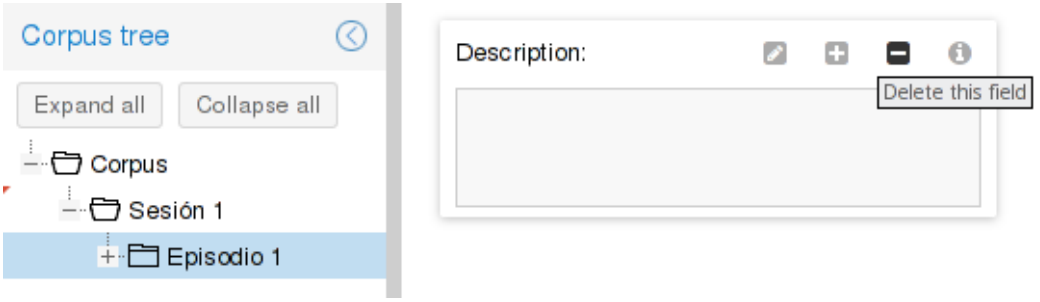
CP-03	
Descripción	Se elimina un nodo del corpus mediante la interfaz.
Entrada	
Resultado esperado	Se elimina el nodo actual así como todos los registros asociados a dicho nodo. Tanto el registro que lo define como toda la rama que pende de él.
Resultado obtenido	Correcto.

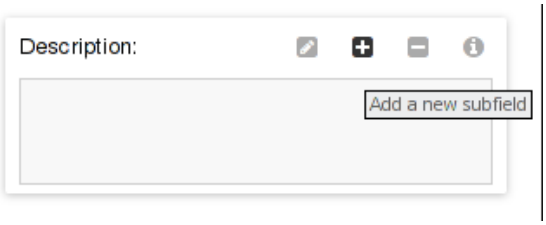
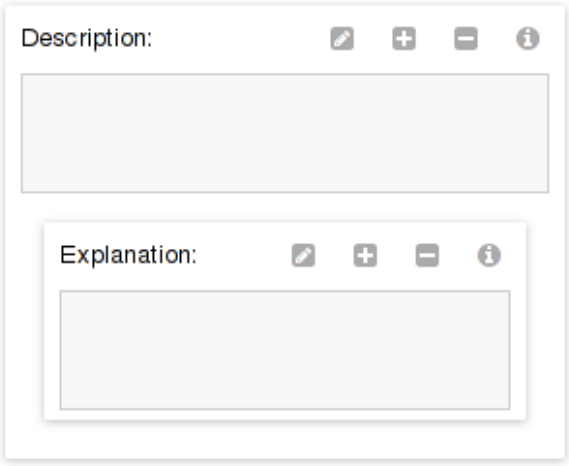
CP-04	
Descripción	Se modifica el nombre de un nodo del corpus mediante la interfaz.
Entrada	
Resultado esperado	Se modifica el campo relacionado con el nombre del registro que representa el nodo del corpus seleccionado.
Resultado obtenido	Correcto.

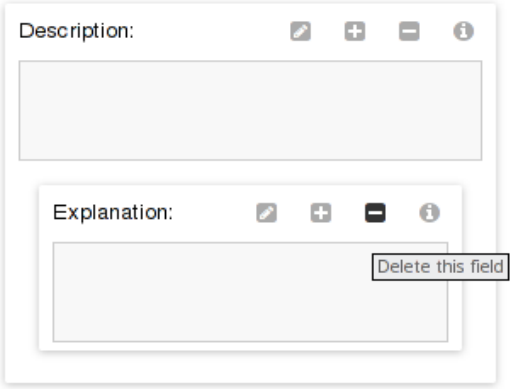
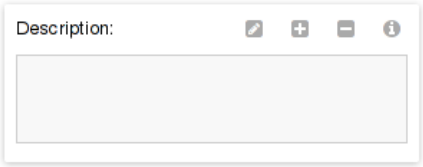
CP-04

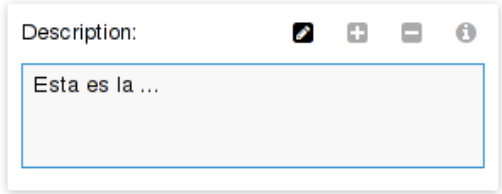
Descripción	Se crea un nuevo campo seleccionado el tipo de campo.
Entrada	
Resultado esperado	Se crea un nuevo campo visual siguiendo el tipo de campo seleccionado.
Resultado obtenido	Correcto.

CP-05	
Descripción	Se añade un nuevo campo a un nodo del corpus concreto.
Entrada	Componente visual de un campo.
Resultado esperado	Se añade el componente visual al nodo seleccionado y se resuelven las asociaciones existentes en sus modelos. 
Resultado obtenido	Correcto.

CP-05	
Descripción	Se elimina un campo concreto de un nodo del corpus.
Entrada	
Resultado esperado	Se elimina el registro que representa el campo seleccionado así como todos los registros de sus campos hijos. Se eliminan todos los componentes visuales que se vean afectados por esta eliminación de registros.
Resultado obtenido	Correcto.

CP-06	
Descripción	Se añade un nuevo campo a un campo concreto.
Entrada	
Resultado esperado	Se añade el componente visual al nodo seleccionado y se resuelven las asociaciones existentes en sus modelos. 
Resultado obtenido	Correcto.

CP-07	
Descripción	Se elimina un campo concreto de un campo.
Entrada	
Resultado esperado	Se elimina el registro que representa el campo seleccionado así como todos los registros de sus campos hijos. Se eliminan todos los componentes visuales que se vean afectados por esta eliminación de registros. 
Resultado obtenido	Correcto.

CP-08	
Descripción	Se establece el campo como de solo lectura.
Entrada	
Resultado esperado	Se mantiene el valor previo y se inhabilita la posibilidad de modificación.
Resultado obtenido	Correcto.

CP-09	
Descripción	Se envía un corpus nuevo al servidor.
Entrada	Jerarquía del corpus en formato JSON.
Resultado esperado	El servidor recibe la jerarquía y crea los nodos y campos pertinentes. Si existen campos con contenido multimedia, devuelve los nuevos identificadores asociados a dichos campos.
Resultado obtenido	Correcto.

CP-10	
Descripción	Se envía un campo con contenido multimedia al servidor.
Entrada	Identificador del campo multimedia y el blob junto con su nombre y mime.
Resultado esperado	El servidor recibe el blob y lo almacena en el campo especificado y crea dos subcampos asociados con el nombre dado al blob y su mime.
Resultado obtenido	Correcto.

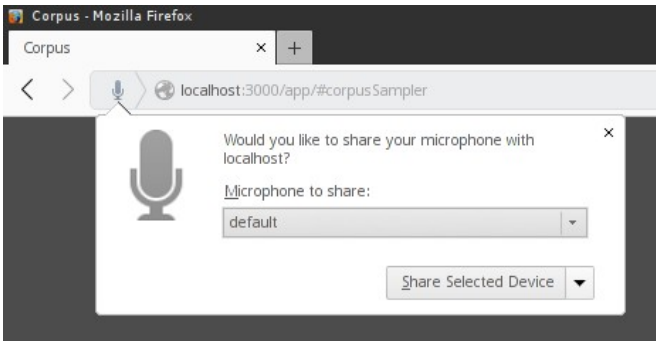
CP-11	
Descripción	Se sincroniza las modificaciones realizadas sobre un corpus.
Entrada	Corpus editado junto con todas las operaciones de modificación realizadas sobre sus elementos.
Resultado esperado	El cliente transforma cada una de las modificaciones realizadas sobre cualquier elemento del corpus en operaciones válidas. Una vez generadas la operaciones pertinentes, se envía en orden al servidor.
Resultado obtenido	Correcto.

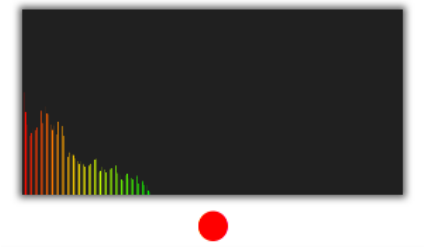
CP-12	
Descripción	Se lista las sesiones activas.
Entrada	-
Resultado esperado	El servidor busca todas las sesiones cuyos campos asociados al intervalo de tiempo de muestreo este el día actual.
Resultado obtenido	Correcto.

CP-13	
Descripción	Se inicia una nueva muestra.
Entrada	El cliente indica al servidor que se va a realizar una muestra de una sesión concreta.
Resultado esperado	El servidor crea una nueva muestra y duplica todos los campos editables que posea la jerarquía de la sesión seleccionada y se añaden a la muestra. Todos los nuevos campos tienen especificado que sobre-escriben a los campos originales. El servidor devuelve la muestra con todos los campos asociados al cliente.
Resultado obtenido	Correcto.

CP-14	
Descripción	Se renderiza un nodo del ejercicio.
Entrada	Registro del nodo del corpus y campos asociados a la muestra.
Resultado esperado	El cliente renderiza los campos asociados al nodo del corpus pero sobre-escribiendo aquellos contenidos por la muestra.
Resultado obtenido	Correcto.

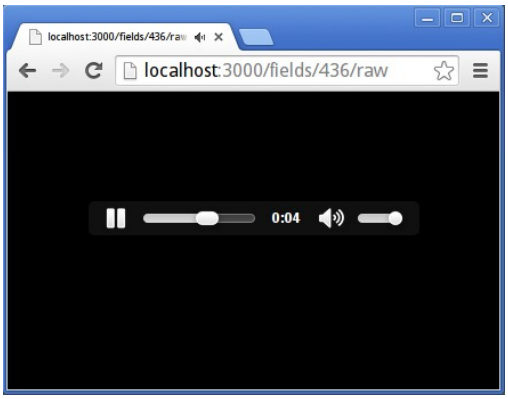
CP-15	
Descripción	Se finaliza un nodo del ejercicio.
Entrada	Jerarquía del corpus en formato JSON.
Resultado esperado	El cliente envía los datos de los campos editables de dicho nodo, elimina la vista generada y renderiza el siguiente nodo del ejercicio.
Resultado obtenido	Correcto.

CP-16	
Descripción	Inicio de la captura.
Entrada	Nodo de tipo captura que posea un campo de captura de audio.
Resultado esperado	El cliente pide al navegador que solicite al usuario que active la captura de audio. 
Resultado obtenido	Correcto.

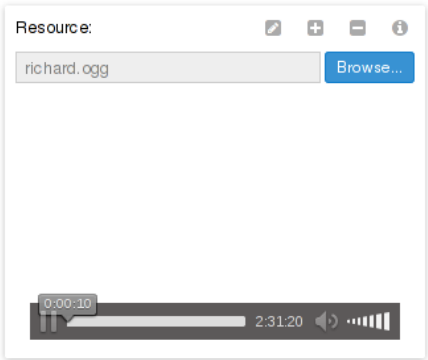
CP-17	
Descripción	Precaptura del audio.
Entrada	Flujo de audio.
Resultado esperado	El cliente renderiza un analizador de frecuencia para mostrar al usuario que se está capturando el audio. 
Resultado obtenido	Correcto.

CP-18	
Descripción	Captura del audio.
Entrada	Flujo de audio.
Resultado esperado	El cliente comienza a capturar el flujo del audio y lo detiene cuando se finaliza la captura. Con el flujo capturado se conforma un blob de tipo WAV y se representa visualmente su contenido. 
Resultado obtenido	Correcto.

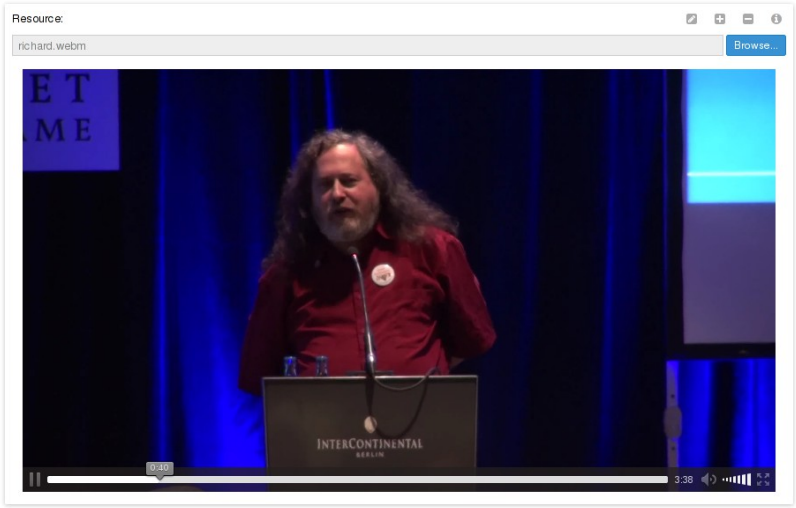
CP-19	
Descripción	Se envía la captura al servidor.
Entrada	Identificador del campo de tipo captura y su blob que contiene el flujo de audio capturado.
Resultado esperado	El servidor recibe la captura y lo almacena en el campo correspondiente. Crea dos subcampos para indicar el nombre de la captura y su mime asociado (WAV).
Resultado obtenido	Correcto.

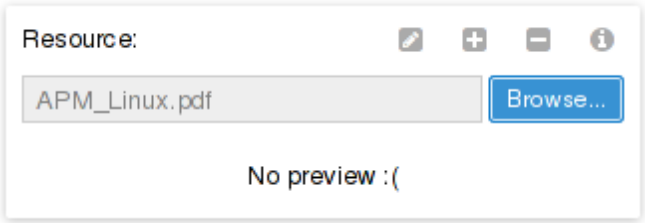
CP-20	
Descripción	Se solicita una captura.
Entrada	Identificador de la captura.
Resultado esperado	El servidor proporciona el blob asociado a la captura especificada. 
Resultado obtenido	Correcto.

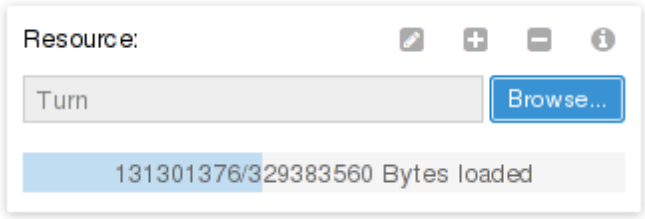
CP-21	
Descripción	Se solicita una imagen como valor de un campo multimedia.
Entrada	Ubicación de la imagen.
Resultado esperado	El cliente lee el fichero y renderiza una visualización de su contenido. 
Resultado obtenido	Correcto.

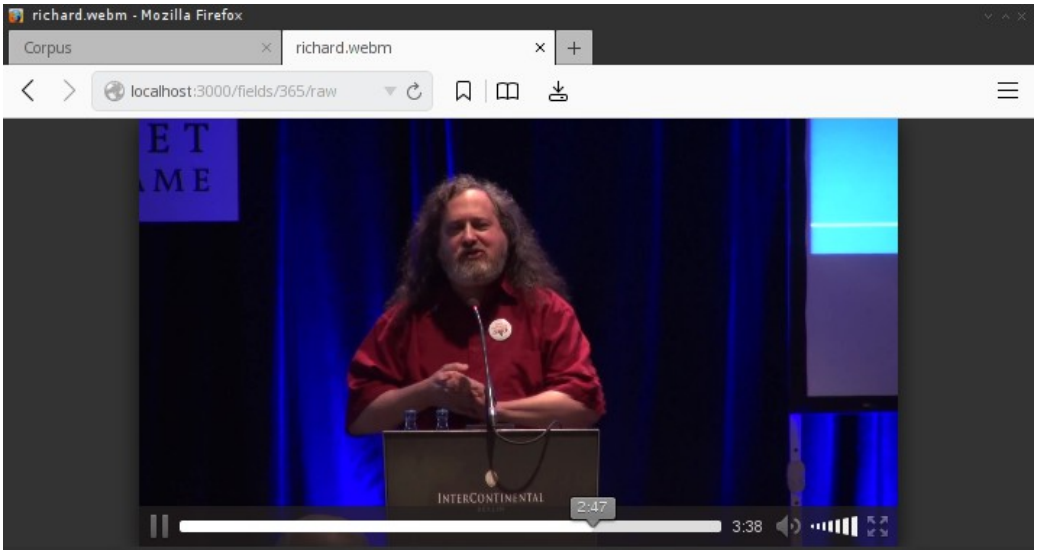
CP-22	
Descripción	Se selecciona un audio como valor de un campo multimedia.
Entrada	Ubicación del audio.
Resultado esperado	El cliente lee el fichero y renderiza una visualización de su contenido. 
Resultado obtenido	Correcto.

CP-23	
Descripción	Se selecciona un vídeo como valor de un campo multimedia.
Entrada	Ubicación del vídeo.
Resultado esperado	El cliente lee el fichero y renderiza una visualización de su contenido.

	
Resultado obtenido	Correcto.

CP-24	
Descripción	Se selecciona un tipo de fichero distinto a una imagen, vídeo o audio como valor de un campo multimedia.
Entrada	Ubicación del fichero multimedia.
Resultado esperado	El cliente lee el fichero y muestra un aviso indicando que no es posible visualizar su contenido.
	
Resultado obtenido	Correcto.

CP-25	
Descripción	Se carga un fichero multimedia.
Entrada	Ubicación del fichero multimedia.
Resultado esperado	El cliente muestra una barra de carga para indicar la usuario el proceso de lectura antes de renderizar su contenido.
	
Resultado obtenido	Correcto.

CP-26	
Descripción	Se solicita un recurso multimedia.
Entrada	Identificador del campo multimedia.
Resultado esperado	El servidor proporciona el blob asociado al campo especificado. 
Resultado obtenido	Correcto.

Capítulo 8

CONCLUSIONES

En el presente capítulo se expondrán los distintos logros y evidencias que se han conseguido y plasmado durante todo el intervalo que ha conllevado la ejecución del trabajo de fin de grado.

Una vez concluido el desarrollo del proyecto, el resultado final ha sido:

- La especificación e implementación general de un entorno de desarrollo relacionado con la gestión de corpus.
- La creación de una aplicación web que permite visualmente realizar diversas operaciones ofrecidas por dicho entorno. Concretamente la gestión y manipulación de los corpus almacenados en el sistema, la realización de ejercicios por parte de estudiantes y la adquisición de muestras.

Se ha comprobado empíricamente que la solución que se dio para abordar el problema de gestión y manipulación de los contenidos de un corpus ha sido muy positiva y en cierto grado elegante. Elegante en referencia a que aunque conceptualmente pueda resultar sencilla, el potencial que posee es muy elevado en comparación con otras especificaciones más complejas. Y positiva respecto a que ha sido suficiente como para concluir con los requisitos descritos.

Se debe principalmente al metamodelo definido para modelar su estructura y jerarquía así como a los datos que almacena. Aunque en el proyecto actual se limitó a una estructura previamente definida (Raíz → Sesiónes → Episodios → Capturas), se ha comprobado con el paso del tiempo que era perfectamente posible haber eliminado esta limitación artificial, lo cual hubiera brindado total libertad a las aplicaciones a definir y emplear una jerarquía más conveniente a su forma de proceder.

Gracias a la forma en la que se define y especifica la información contenida en cada nodo de un corpus, es posible almacenar cualquier tipo de dato. El abanico de tipos posibles conceptualmente es ilimitado ya que permite desde el guardado de simples cadenas de texto hasta ficheros y estructuras más complejas como pueden ser árboles y listas.

A la vez, el tipo de servidor que se usó finalmente ha proporcionado un número de ventajas respecto a la primera aproximación que se hizo. Al focalizar el servidor únicamente al suministro de servicios mediante una API Rest se ha conseguido desacoplar perfectamente su funcionalidad del resto de aplicaciones. Además, el empleo del protocolo HTTP para la solicitud de servicios y recursos facilita que cualquier tipo de aplicación que se desarrolle pueda usar fácilmente la API que proporciona, ya que dicho protocolo es independiente de la arquitectura del sistema nativo como del lenguaje de programación usado en su codificación. Lo que conlleva a la afirmación de que el resultado final que se ha obtenido con la implementación del servidor, ha sido la creación de un entorno de desarrollo que puede ser explotado por aplicaciones externas y de terceros.

Respecto a la aplicación web desarrollada, también se puede constatar que su diseño ha sido bastante adecuado para el tipo de problema que se quería abordar. La modularización de sus funcionalidades en componentes ha permitido realizar un desarrollo iterativo por el cual se añadían nuevas funcionalidades mediante la implementación de nuevos componentes y posterior inserción en la aplicación. La especificación de un módulo núcleo – que implementa los modelos así como los protocolos de comunicación y acceso a los servicios del servidor REST – permite que otros componentes generados puedan emplearlos como clases base del entorno. La gran ventaja es que los componentes que emplean dicho módulo pueden reutilizar estas clases que se proporcionan sin tener que conocer su funcionamiento interno, ni las configuraciones necesarias para emplear los servicios del servidor web. Además, al existir un módulo base, cualquier modificación realizada en el servidor se puede adaptar fácilmente en las clases afectadas sin llegar a repercutir en el funcionamiento del resto de componentes.

Las funcionalidades implementadas finalmente en la aplicación web han sido la plena gestión de los nodos de un corpus así como los campos que los componen. También permite la toma de muestras de los corpus previamente definidos en el sistema y podría ser considerado como una primera versión de la realización de ejercicios por parte de los estudiantes.

Una vez expuestas las conclusiones relacionadas con el sistema generado se continuará con aquellas vinculadas al proyecto en general.

La realización del proyecto ha supuesto al estudiante un cúmulo de nuevos conocimientos y mejora de capacidades previas. Aunque esta asignatura es en cierta parte distinta al resto de asignaturas del ciclo formativo, permite aglutinar diversos aspectos que fueron desarrollados y explicados anteriormente de forma independiente. Ello hace que se consiga una visión general y real de que es el desarrollo de un proyecto y cuales son sus límites y forma de proceder.

El alumno ha afianzado numerosos conceptos relacionados con la ingeniería software y el correcto desarrollo de software. Por otra parte, gracias al desarrollo del sistema, el alumno ha adquirido una formación adicional respecto a las tecnologías empleadas:

- Ha adquirido un nivel aceptable del lenguaje de programación Ruby.
- Se ha familiarizado con el framework Ruby on Rails, sobretodo en lo relacionado con la gestión de modelos, persistencia y construcción de servidores REST.
- Ha adquirido un gran dominio, tanto conceptualmente como técnico, en el lenguaje de scripting JavaScript.
- Conoce con gran detalle el funcionamiento del framework empleado en el cliente – ExtJS – ya que en numerosas ocasiones tuvo que leer y estudiar exhaustivamente su código fuente para poder conocer su comportamiento real, las limitaciones que poseía y posibles formas para manipularlo sutilmente.

El alumno puede asegurar rotundamente que los conocimientos obtenidos durante la realización del trabajo han supuesto y supondrán una ventaja muy significativa para su formación académica y profesional. El tiempo que ha dispuesto en lectura de documentación y artículos sobre la web y las incipientes tecnologías, le han labrado una concepción de cómo está evolucionando el desarrollo software en estas áreas.

Por otra parte, a raíz de la naturaleza de la asignatura y por el tiempo invertido, ha permitido al alumno conocer sus fortalezas y vicios debido a que fueron aflorando y evidenciándose a lo largo del transcurso del proyecto.

Una mejora que debe hacerse es acerca de la gestión y planificación del proyecto. En diversas ocasiones ha sido evidente que su forma de proceder no ha sido la mejor indicada, lo que llevó diversas complicaciones y demoras. En ocasiones priorizó inadecuadamente las tareas a realizar y los objetivos a cumplir. Sin embargo, ser consciente de estos problemas ayudó en cierta medida a focalizarse más en estos aspectos e intentar mitigarlos lo máximo posible. Lo más importante – desde la vista del alumno – es que ha comprendido la gran importancia que tiene la planificación y seguimiento de un proyecto, y no solo conceptualmente, ya que experimentó de primera mano las posibles consecuencia negativas por no seguirlo.

Otro aspecto a mejorar está relacionado con la documentación. El alumno focalizó parte del tiempo en la adquisición de conocimientos prácticos/teóricos así como en la experimentación de las tecnologías de la web. Ello motivó que la documentación diaria generada a lo largo del proyecto fuese escasa. En la recta final del proyecto el alumno se percató de la gran importancia que es mantener una generación de documentación constante en el tiempo y generosa en su contenido.

Capítulo 9

TRABAJO FUTURO

En el presente capítulo se analizará y replanteará las líneas de trabajo futuro del sistema Corpus. La principal motivación de este apartado es por tanto definir una líneas para un desarrollo posterior y evidenciar las funcionalidades no implementadas en el transcurso de la ejecución del proyecto.

- Permitir ordenar los distintos campos dentro de un nodo cuando se está en modo edición. Actualmente el componente CorpusCreator no permite la especificación de la posición de los campos. Aunque en el modelo está representado – la tabla asociativa del nodo y los campos posee un atributo “order” – no está siendo empleado por el cliente. El principal inconveniente es la relativa complejidad que posee el framework ExtJS para definir correctamente las asociaciones entre las distintas entidades del modelo.

  Esfuerzo aproximado: 2 semanas en portar todas las entidades afectadas.

- Implementar la funcionalidad de que un evaluador pueda corregir y puntuar una muestra concreta realizada por un estudiante. Hace falta crear la entidad “Evaluation” en el modelo y codificar toda la lógica asociada a la gestión visual de los nuevos campos. La idea es que una evaluación vaya añadiendo tantos campos como correcciones sobre una muestra haya realizado. Cada uno de ellos referenciaría al campo de la muestra como “override”. Mediante esta forma se puede posteriormente vincular y relacionar las distintas evaluaciones a sus correspondientes campos de la muestra.

  Esfuerzo aproximado: 6 semanas.

- Implementar la creación y gestión de esquemas en el cliente. Realizadas ambas como un componente independiente. Su codificación sería muy parecida a la realizada en el componente CorpusCreator.

  Esfuerzo aproximado: 4 semanas.

- Portar la aplicación web a distintas plataformas. Empaquetarla como extensión de un navegador web, como aplicación web del sistema operativo (FirefoxOS, Windows8) o como aplicación para Android y iOS (Cordova).

  Esfuerzo aproximado: 2 semanas por cada portabilidad.

- Mejorar la eficiencia del código mediante operaciones de concatenación de ficheros y *minificación* del código existente. Closure Compiler, JSMin y YUI Compressor son buenos candidatos. Sin embargo es recomendable emplear GruntJS para automatizar este tipo de operaciones.

  Esfuerzo aproximado: 3 semanas.

- Emplear un framework para la generación de pruebas unitarias para toda la aplicación web. Esto permitiría perfilar un código más estable, seguro y robusto. Las mejores opciones son Mocha, Qunit, Jasmine y Karma.

  Esfuerzo aproximado: 6 semanas.

- Emplear la sentencia “use strict” en todos los ficheros JavaScript de la aplicación web. Gracias a ella se eliminan posibles errores ocultos mediante excepciones, mejora la optimización al prohibir ciertas expresiones y finalmente encauza la sintaxis del código a las futuras especificaciones de ECMAScript.

  Esfuerzo aproximado: 5 semanas.

- Mejorar el componente relacionado con la realización de ejercicios por parte de los estudiantes. Permitiendo configurar el comportamiento de cada captura según sus campos de configuración. Por ejemplo, permitir configurar si el recurso multimedia debe reproducirse automáticamente una vez cargado, si sólo puede ser reproducido una, dos o infinitas veces...

  Esfuerzo aproximado: 2 meses.

Capítulo 10

BIBLIOGRAFÍA

Bibliografía de Ingeniería Software

[Sommerville] Ian Sommerville, Maria Isabel Alfonso, Antonio Botia. "Ingeniería del Software". Pearson, 2005 (7ª edición) ISBN: 978-8478290741 (Última consulta, 11/09/14).

[Larman] Larman Craig. "UML y Patrones. Introducción al Análisis y Diseño Orientado a Objetos y al Proceso Unificado". Prentice Hall, 2004 (2ª edición) ISBN: 978-8420534381 (Última consulta, 02/05/14).

[Arlow] Jim Arlow, Ila Neustadt. "UML 2". Anaya Multimedia, 2006. ISBN: 978-8441520332 (Última consulta, 21/12/14).

[Rumbaugh] Grady Booch, James Rumbaugh, Ivar Jacobson. "El Lenguaje Unificado de Modelado. Manual de Referencia". Pearson Addison Wesley, 2010 (2ª edición) ISBN: 978-8478290871 (Última consulta, 02/04/15).

Bibliografía relacionada con la parte del cliente

[Ninja] John Resig, Bear Bibeault. "Secrets of the JavaScript Ninja". Manning, 2013. ISBN: 978-1933988696 (Última consulta, 26/08/14).

[MDN] Mozilla Developer Network. https://developer.mozilla.org. (Última consulta, 14/06/15).

[Flanagan] David Flanagan. "JavaScript: The Definitive Guide". O'Reilly, 2011. ISBN: 978-0596805524 (Última consulta, 02/10/14).

[Doc-ExtJS] Ext JS Guides. Sencha, 2015. (Última consulta, 20/06/15).

[Blog-ExtJS] Ext JS Blog. Sencha, 2015. (Última consulta, 12/03/15).

Bibliografía relacionada con la parte del servidor

[API-RoR] Documentation of Ruby of Rails 4.2.2. http://api.rubyonrails.org. RoR, 2015.

[Guides-RoR] Ruby on Rails Guides (v4.2.2). http://guides.rubyonrails.org. RoR, 2015.

Bibliografía complementaria

[StackOverflow] https://stackoverflow.com (Última consulta, 20/06/15).

[Wikipedia] https://en.wikipedia.org (Última consulta, 29/06/15).

Capítulo 11

ANEXOS

11.1. Manual de instalación

11.1.1. Introducción

En el presente anexo se explicarán los pasos necesarios para poder instalar el sistema Corpus en cualquier sistema GNU/Linux.

También se detallarán los servicios y subprogramas que se tuvieron que desarrollar para simplificar la gestión de las versiones y dependencias que posee el sistema.

11.1.2. Explicación detallada

Debido a los numerosos elementos que componen el framework de Ruby on Rails – además de las respectivas versiones del lenguaje padre – existe una alta probabilidad de que surjan numerosos errores de dependencias a la hora de desplegar un nuevo servidor web.

Diversas distribuciones GNU/Linux – concretamente las freeze como Debian, Ubuntu... – poseen congeladas las versiones de los paquetes de su repositorio. Esto hace que por una parte aumente considerablemente su estabilidad, pero también provoca que exista un gran retraso entre las últimas versiones y las publicadas en el repositorio.

Por ejemplo, en Debian la versión disponible de Ruby (sin acudir a SID) es la 1.9.1 y sin embargo, la versión más reciente es la 2.2.2. Provoca además que ciertas versiones de ruby no sean capaces de ejecutar correctamente versiones de determinados componentes, por ejemplo rails. Y si además juntamos que normalmente cada servidor web de Ruby on Rails posee unas especificaciones distintas respecto a componentes y versiones requeridas, nos encontraremos fácilmente ante una situación muy complicada de manejar.

Lo anterior hace que sea muy adecuado emplear gestores de versiones de entorno de desarrollo para Ruby. Permite independizar los contextos de cada entorno de desarrollo (cada servidor web) y tener un mejor control sobre las versiones de las dependencias requeridas.

Existen dos grandes gestores, *RVM* y *rbenv*. Aunque al principio el alumno empleó RVM, después de analizar mejor las diferencias y funcionalidades de cada uno, optó finalmente rbenv por todas las ventajas que ofrecía. https://github.com/sstephenson/rbenv/wiki/Why-rbenv%3F

Rbenv permite instalar (mediante un plugin) cualquier versión de Ruby independientemente de la distribución. Además permite sobreescribir el entorno global de Ruby o hacerlo sobre un entorno en concreto. Gracias a ello, cualquier servidor web puede ser desplegado sin tener en consideración la paquetería por defecto de la distribución donde se aloje.

Cuando *rbenv* instala una nueva versión de Ruby la realiza compilando su código fuente. Lo cual permite que sea instalada en una gran cantidad de sistemas, sin embargo, el proceso en ocasiones es demasiado largo, sobretodo cuando la máquina anfitriona tiene poco poder de cómputo (por ejemplo una *raspberry*).

La solución que dio el alumno fue crear un conjunto de scripts para minimizar la complejidad en la

instalación de rbenv y versiones de ruby necesarias. Además compiló dos versiones distintas de Ruby para tres distribuciones distintas con el objetivo de reducir el tiempo necesario para instalar el sistema.

Se creó un nuevo repositorio en GitHub (https://github.com/aensoler/ruby-installer) con los scripts que facilitan la instalación así como las distintas versiones de ruby precompiladas. A continuación se enumeran los distintos scripts existentes:

- ruby-v2.1.3-scratch: instala la versión 2.1.3 mediante su compilación nativa.
- ruby-v2.1.3-debian-7.8-x86_64: instala la versión 2.1.3 compilada para Debian 7.8 x86_64.
- ruby-v2.1.3-ubuntu-14.04-x86_64: instala la versión 2.1.3 compilada para Ubuntu 14.04 x86_64.
- ruby-v2.1.3-raspbian-7-armv6l: instala la versión 2.1.3 compilada para la Raspberry (raspbian).
- ruby-v2.2.2-scratch: instala la versión 2.2.2 mediante su compilación nativa.
- ruby-v2.2.2-debian-7.8-x86_64: instala la versión 2.2.2 compilada para Debian 7.8 x86_64.
- ruby-v2.2.2-ubuntu-14.04-x86_64: instala la versión 2.2.2 compilada para Ubuntu 14.04 x86_64.
- ruby-v2.2.2-raspbian-7-armv6l: instala la versión 2.2.2 compilada para la Raspberry (raspbian).

Para instalar la versión deseada tan solo habría que ejecutar el siguiente comando en una terminal:

curl -sSL https://github.com/aensoler/ruby-installer/raw/master/scripts/ruby-v2.1.3-debian-7.8-x86_64 | bash

Una vez instalada correctamente la versión de Ruby, ya solo sería necesario descargarse la aplicación Corpus.

Gracias a que el sistema se desarrolló empleando el gestor de versiones Git, permite que su instalación sea muy sencilla. Incluso el alumno desarrolló otro script para simplificar aún más dicho proceso.

El script está alojado en los siguientes servidores:

- http://alumnos.inf.uva.es/davsole/corpus-installer
- http://virtual.lab.inf.uva.es:20092/

Para ejecutarlos tan sólo sería necesario ejecutar la siguiente sentencia en una terminal:

curl -sSL http://alumnos.inf.uva.es/davsole/corpus-installer | bash

o

wget -q0- http://virtual.lab.inf.uva.es:20092/ | bash

Este script automáticamente crea una enlace simbólico (app) del contenido de la aplicación Corpus en la carpeta *public* del servidor Corpus. Lo cual permite que desde el propio servidor se tenga acceso a los recursos de la aplicación mediante la ruta *app*.

Sin embargo, los repositorios del código son privados, por lo que excepto que se tenga credenciales es imposible instalar el sistema.

El código se puede encontrar en el directorio “Código” del CD-ROM proporcionado. Además, se ha desarrollado un script bash en el directorio “Instalador” para realizar el proceso de instalación con el código local.

11.1.3. Instalación rápida

A continuación se enumera los diversos pasos que se debe seguir para la instalación y ejecución satisfactoria del sistema Corpus.

1. Instalar la versión 2.1.3 de Ruby en el sistema

   curl -sSL https://github.com/aensoler/ruby-installer/raw/master/scripts/ruby-v2.1.3-scratch | *bash*

   Existen versiones ya compiladas, ver sección 11.1.2.

2. Instalar el sistema Corpus

   En el CD-ROM se puede localizar un script en el directorio “Instalador”. Únicamente hay que ejecutarlo en una terminal.

   ./Instalador/corpus-installer

   Por defecto se instala en la carpeta personal. Se puede cambiar pasando el directorio como primer argumento.

3. Arrancar el servidor Corpus

   cd corpus/corpus-back-end/ && rails server

   Por defecto el servidor arranca en localhost y en el puerto 3000.

4. El acceso a la aplicación Corpus se haría mediante la siguiente URI

   http:// HOST:3000 /app/ *por ejemplo:* http://localhost:3000/app/

   Es importante que esté la última barra diagonal.

11.1.4. Actualización

Si los recursos del sistema de Corpus se obtuvieron mediante el servidor Git, el proceso se reduce a los siguientes comandos:

Para actualizar la aplicación Corpus: *cd corpus-font-end && git pull*

Para actualizar el servidor Corpus: *cd corpus-back-end && git pull*

11.2. Manual de usuario

Este anexo está orientado al usuario final y explica la forma de uso del sistema.

11.2.1. Solicitud de registro

Un usuario no registrado puede solicitar una petición de registro en el sistema. Para ello deberá proporcionar cierta información y esperar la aprobación de un administrador del sistema.

Para ello, el usuario no registrado debe seleccionar la opción ***Sign up*** de la barra superior de la página principal. Aparecerá una ventana emergente solicitando el ingreso de diversos campos, como por ejemplo: nombre, apellidos... **Es importante que se tenga acceso al email proporcionado, ya que será necesario para el resto del proceso.**

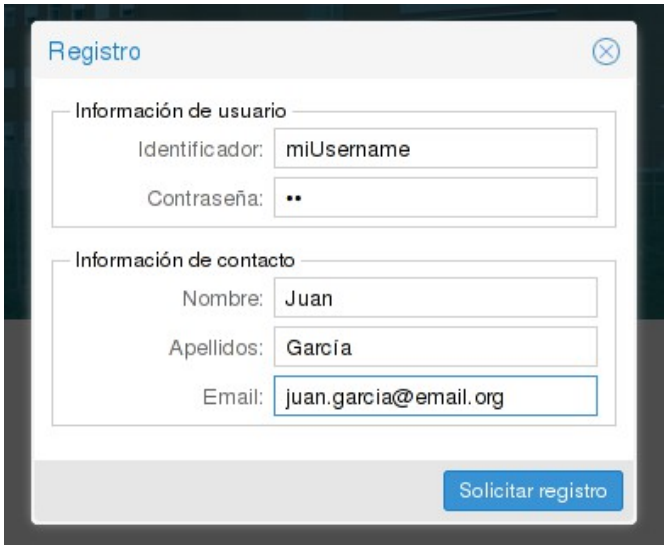


Figura 46: Ventana emergente de registro

Hasta que no se rellenen adecuadamente los campos no podrá enviar la solicitud. La solicitud se habrá enviado si se muestra el siguiente mensaje:

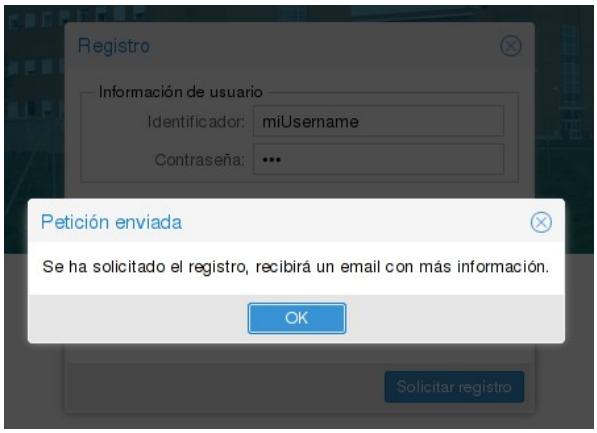
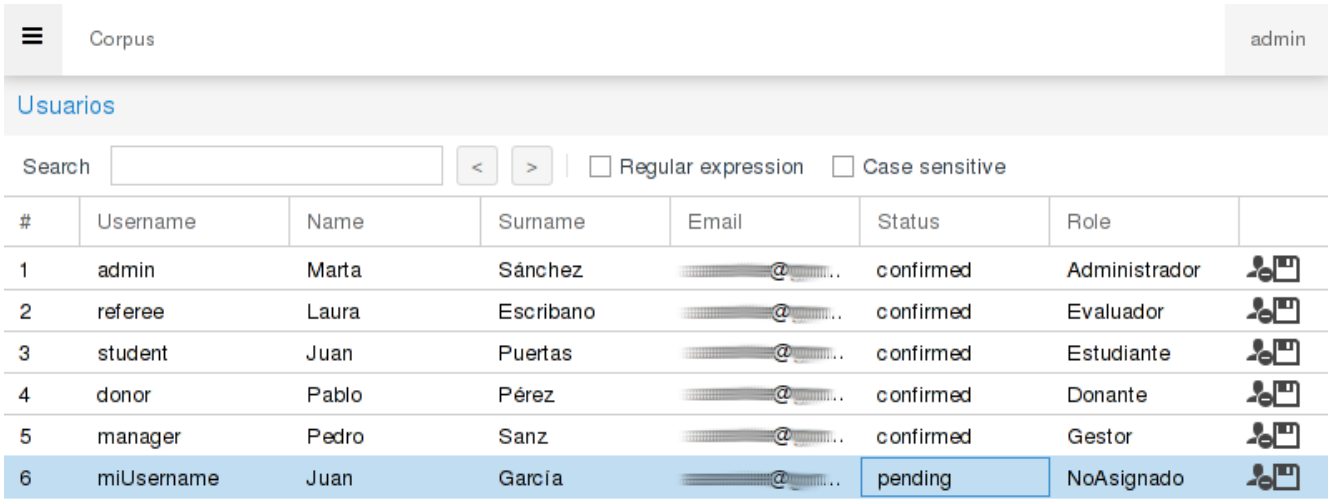


Figura 47: Solicitud enviada

En este punto, el usuario debe esperar hasta que un administrador acepte su petición. Por esa razón, a partir de ahora seremos el rol de administrador.

Siendo administrador y estando autentificado ante el sistema, seleccionaremos la opción *Administrar usuario* del menú lateral de la página; una vez seleccionada aparecerá la siguiente ventana:



#	Username	Name	Surname	Email	Status	Role	
1	admin	Marta	Sánchez	[redacted]	confirmed	Administrador	
2	referee	Laura	Escribano	[redacted]	confirmed	Evaluador	
3	student	Juan	Puertas	[redacted]	confirmed	Estudiante	
4	donor	Pablo	Pérez	[redacted]	confirmed	Donante	
5	manager	Pedro	Sanz	[redacted]	confirmed	Gestor	
6	miUsername	Juan	García	[redacted]	pending	NoAsignado	

Figura 48: Ventana de administración de usuarios

Como se puede observar, la última fila hace referencia a la solicitud que hemos realizado anteriormente. La columna **status** es la que define el estado de registro de un usuario. Sus estados son los siguientes:

- pending: cuenta que aún debe ser aceptada por un administrador.
- accepted: la cuenta ha sido aceptada por un administrador y se ha enviado un email para que el usuario lo valide.
- confirmed: la cuenta ha sido aceptada por un administrador y está validada mediante email.
- blocked: cuenta bloqueada y que no puede realizar ningún tipo de operación sobre el sistema.

En este caso, seguiremos el procedimiento común, especificaremos el estado como aceptado para así enviar un email de validación. También definiremos un rol al usuario, concretamente el de estudiante. Una vez que lo valide podrá en ese mismo momento realizar las acciones definidas en su rol. Para persistir los cambios, es necesario pulsar el botón *save* que se encuentra a la derecha de la fila y que es un icono con forma de disquet.

Ahora retomamos el rol del usuario que realizó la solicitud de registro. Una vez que se ha aceptado la cuenta, se ha enviado un correo automático para poder validar la dirección de correo electrónico que proporcionamos en el formulario.

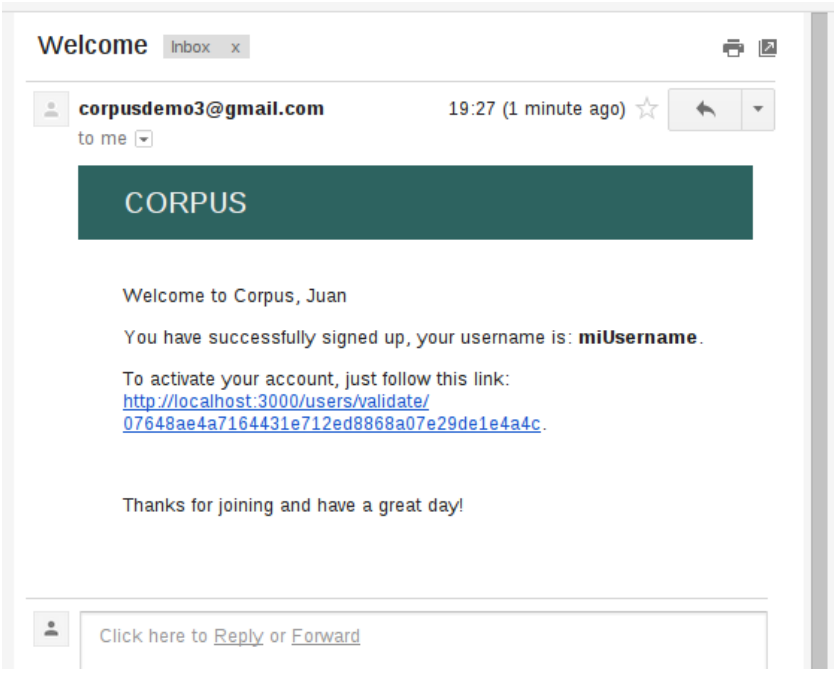


Figura 49: Email de validación de cuenta

Pulsando el enlace proporcionado en el correo validaremos la cuenta y podremos comenzar a realizar las acciones propias de nuestro rol asignado. Para identificarnos tan solo tendremos que pulsar la opción ***Sign in*** de la barra superior de la página principal. Nos aparecerá la siguiente ventana y tendremos que introducir nuestro usuario y contraseña. También se da la posibilidad de almacenar la sesión en el navegador y poder ser reutilizada en otras ocasiones.

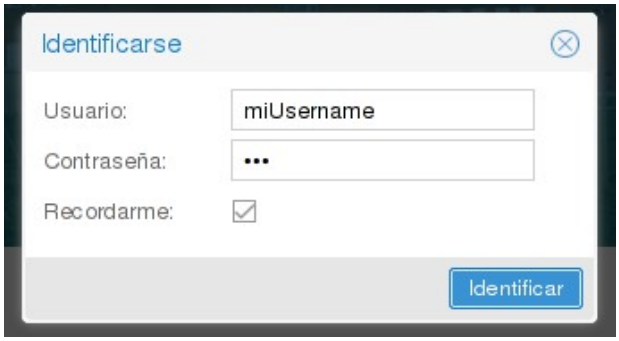


Figura 50: Inicio de sesión

11.2.2. Realizar un ejercicio

Esta acción solamente la pueden realizar los usuarios autentificados ante el sistema y que posean el rol de estudiante.

Primero hay que seleccionar el ejercicio deseado, para ello, hay que pulsar la opción ***Realizar un ejercicio*** del menú lateral de la página. Nos aparecerá un lista con las distintos ejercicios que podremos realizar. En cada elemento aparecerá una descripción del tipo de ejercicio a realizar.

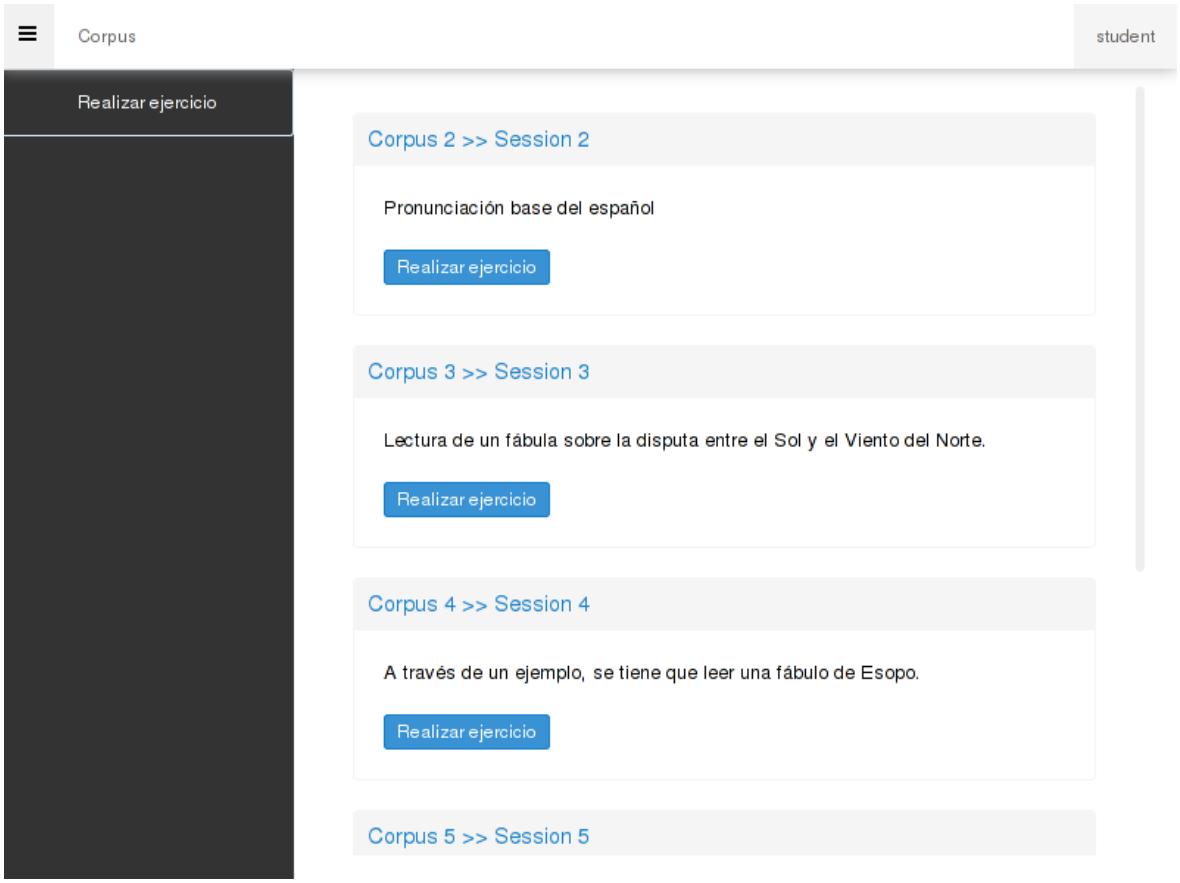


Figura 51: Lista de ejercicios

Para realizar un ejercicio en concreto tan solo tendremos que pulsar el botón “Realizar ejercicio” del ejercicio deseado. Una vez lo hayamos pulsado, se entrará en un nuevo contexto. Las barras superior y lateral desaparecerán y si se realiza una operación no contemplada (por ejemplo, cerrar la ventana) aparecerá un mensaje informativo. En este ejemplo, realizaremos el ejercicio perteneciente al corpus 4.

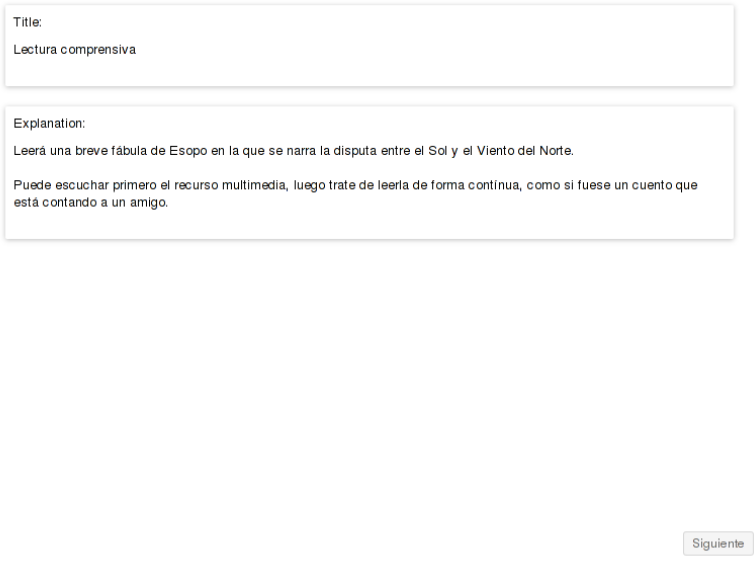


Figura 52: Primera vista del ejercicio seleccionado

Normalmente la primera vista del ejercicio es introductoria, es decir, se explican las distintas tareas que el alumno tendrá que realizar en él. Se podrá avanzar pulsando el botón inferior **Siguiente**. Es posible que en alguna vista existan campos que el estudiante tendrá que rellenar.

Sin embargo, la vista más importante es la relacionada con las capturas de audio. En ocasiones también contiene recursos multimedia que pueden ser visionado por el estudiante. Una vez se visualice un nodo que posea un grabador de audio, aparecerá un mensaje emergente del navegador web preguntando al usuario si autoriza a la aplicación a emplear el micrófono. Mientras que no se acepte, no se podrá realizar ningún tipo de operación.

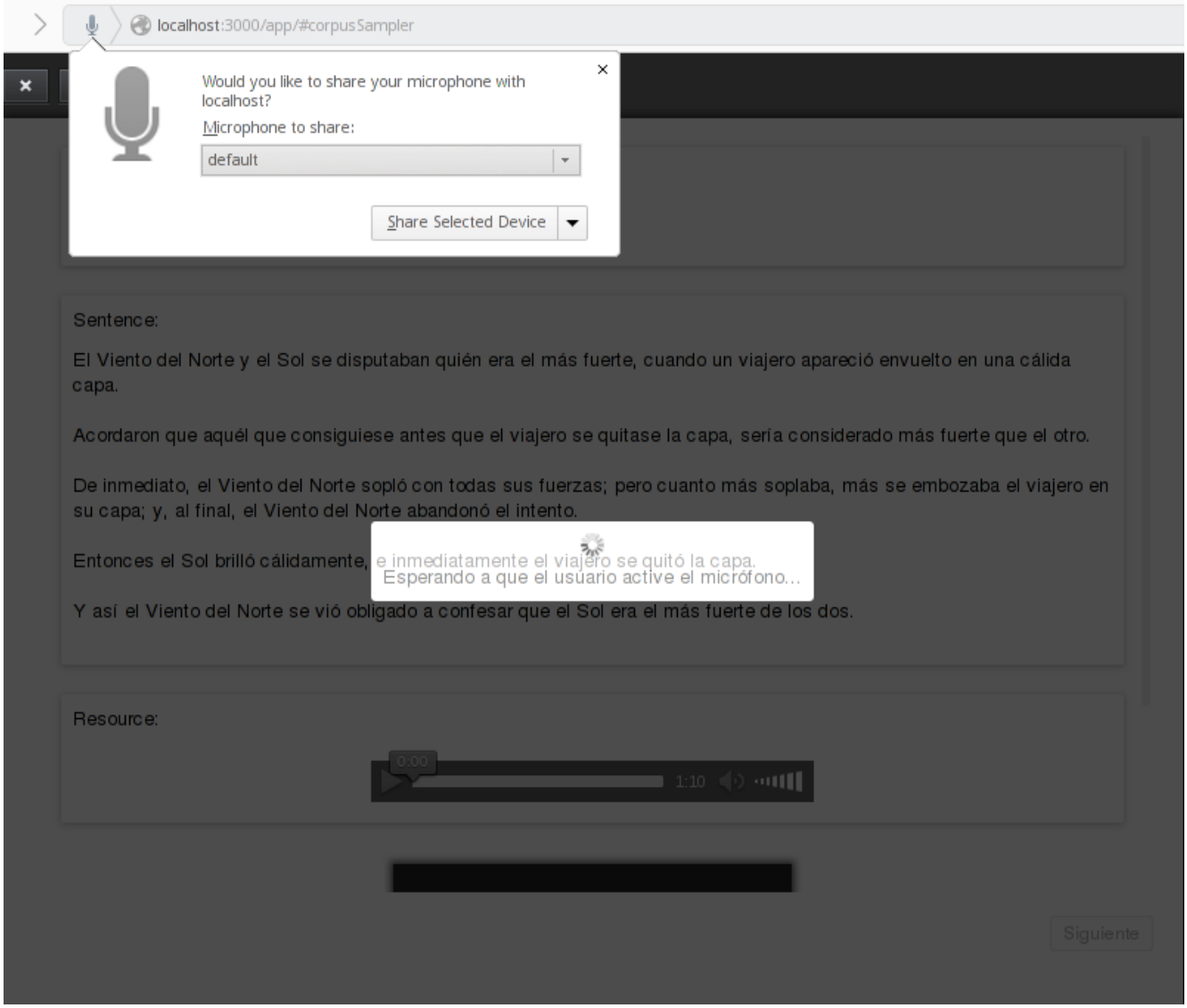


Figura 53: Solicitud de uso del micrófono por parte de la aplicación

Una vez aceptado, se ocultará el dialogo emergente y se permitirá al usuario continuar con el ejercicio. En este caso, existe un recurso multimedia que puede ser empleado libremente por el usuario. Estos recursos pueden ser: audio, vídeo o imagen. También se puede apreciar que el botón *Siguiente* está deshabilitado, la razón es por la existencia del grabador de audio. Hasta que no se realice la captura no se podrá avanzar. Para ello, tan solo hace falta pulsar al r*ec* (botón circular rojo).

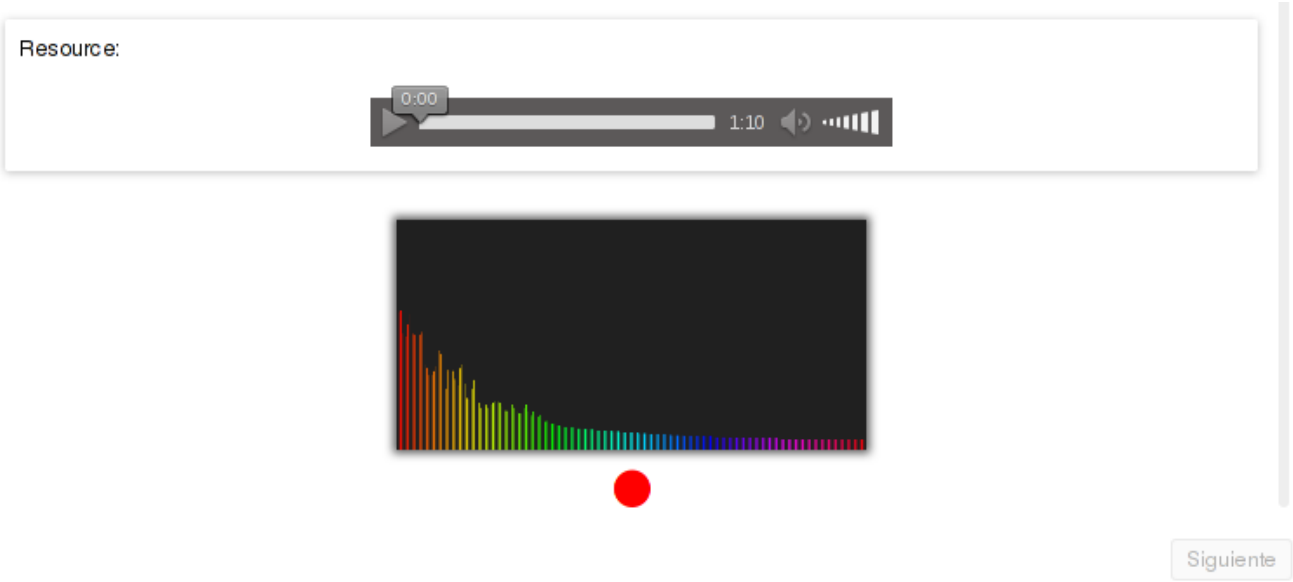


Figura 54: Ejercicio con un campo multimedia y un grabador.

Gracias a la interfaz del grabador de audio, se puede comprobar visualmente si el micrófono funciona correctamente ya que aparecen las frecuencias del audio capturado. Una vez concluida la captura el estudiante tiene que pulsar el botón *stop* (botón cuadrado gris). En ese mismo instante la interfaz del grabador cambiará y mostrará la onda capturada. En este momento se ha habilitado el botón siguiente y el alumno podrá continuar con el ejercicio.

La longitud y forma del ejercicio depende exclusivamente de como se haya estructurado. Cuando se llegue al final, aparecerá un mensaje emergente avisando del fin del ejercicio y devolverá al estudiante al listado de ejercicios disponibles.

11.2.3. Crear un corpus

Esta acción solamente la pueden realizar usuarios autentificados ante el sistema y que posean el rol de gestor.

Para crear un nuevo corpus se debe seleccionar al inicio la opción **Crear corpus** del menú lateral de la página. Una vez pulsada, se cargará una nueva vista en forma de cuadrícula además de una ventana emergente solicitando el esquema que se replicará en el nodo raíz del corpus.

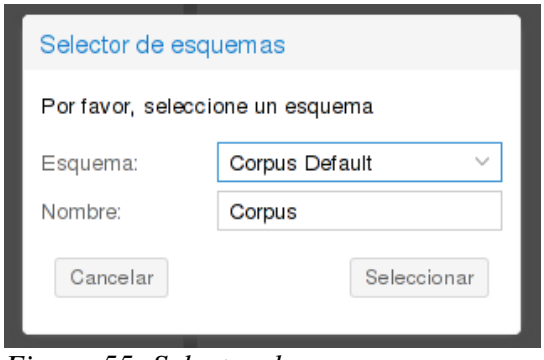


Figura 55: Selector de esquema

Una vez seleccionado el esquema, se creará un nuevo nodo raíz con una réplica de los campos definidos en el esquema. La vista principal tiene la siguiente estructura:

- Parte superior: contiene dos botones distintos: el botón *Exportar* sirve para codificar toda la estructura de un corpus y guardarlo localmente en un fichero JSON; mientras que el de *Enviar* envía todo el corpus al servidor.
- Parte central: destinada a mostrar los campos de cada nodo seleccionado además de permitir su manipulación.
- Parte lateral: se representa la estructura del corpus en forma de árbol. Gracias a ella se pueden añadir, eliminar y mover nodos de la jerarquía. Al seleccionar un nodo concreto se muestran sus campos en la parte central.

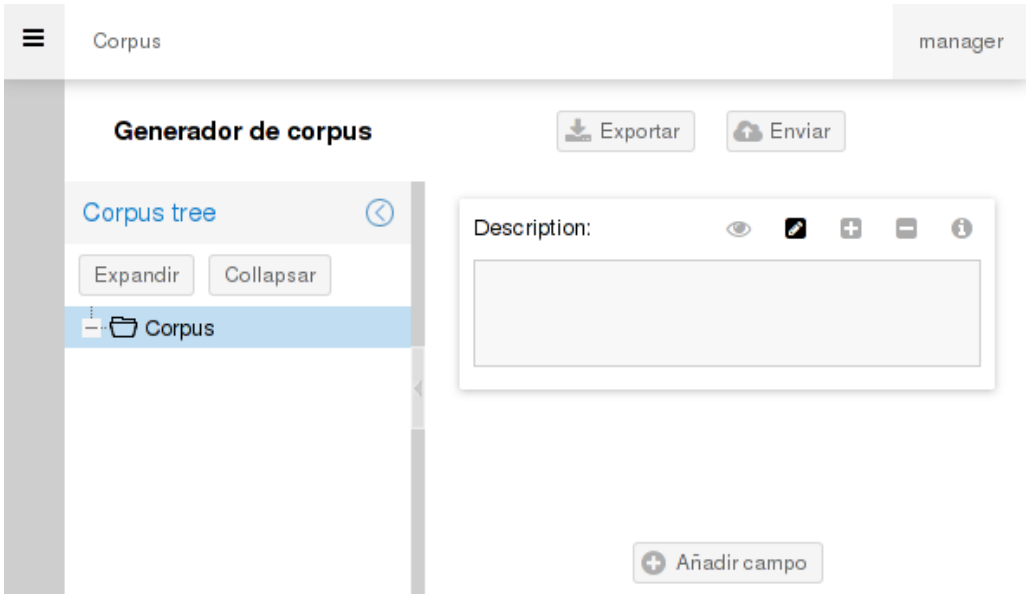


Figura 56: Proceso de creación de un corpus

Para añadir un nuevo nodo solamente se debe pulsar el botón derecho del ratón sobre el nodo padre y seleccionar la opción *Añadir nuevo nodo*.

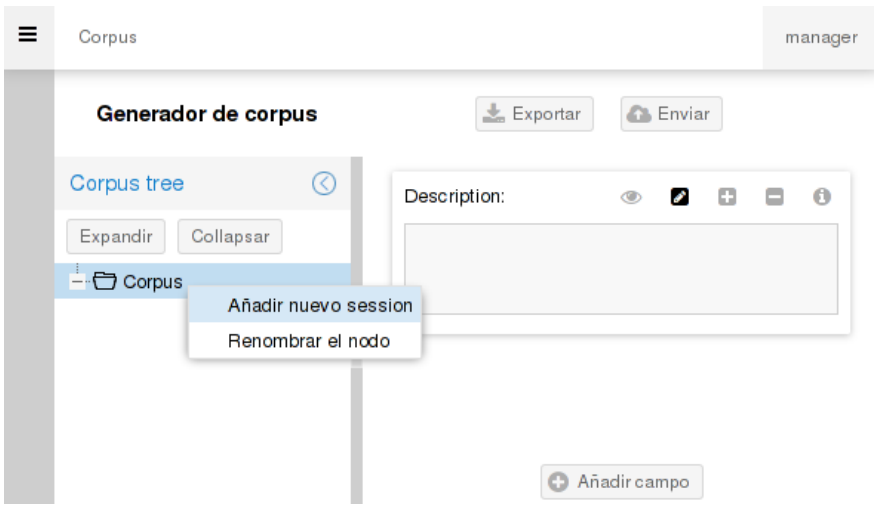


Figura 57: Añadir un nuevo nodo

Al igual que con el nodo raíz, aparecerá nuevamente una ventana emergente para seleccionar el esquema que será empleado como plantilla del nuevo nodo.

Para eliminar un nodo solo hay que pulsar el botón derecho sobre él y seleccionar la opción *Eliminar nodo*. El mismo procedimiento si se quiere renombrar.

Para añadir un nuevo campo al nodo seleccionado hay que pulsar al botón inferior *Añadir campo*. Aparecerá una ventana emergente para seleccionar el tipo concreto de campo que se quiere añadir. Cuando se haya elegido un tipo, únicamente hay que aceptar la ventana emergente y se añadirá automáticamente al final de la colección de campos existentes.

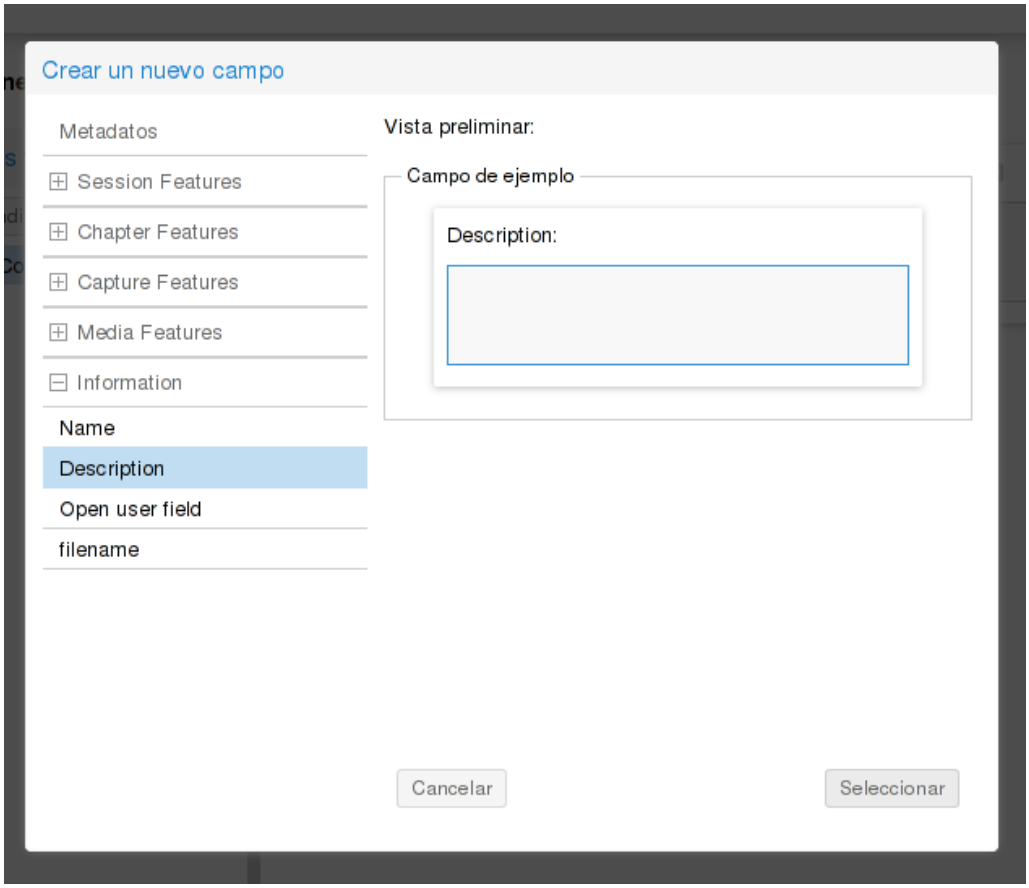


Figura 58: Añadir un nuevo campo al nodo

Los campos poseen opciones especiales que a continuación se van a explicar. Todas estas opciones se pueden realizar mediante la barra superior que poseen.

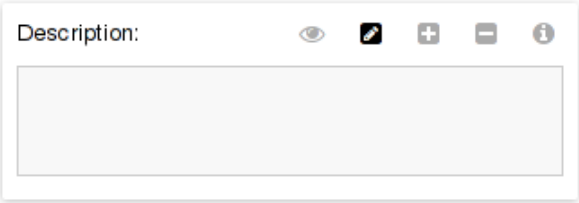


Figura 59: Campo

El icono con forma de ojo permite especificar si el campo se debe mostrar al usuario final, por ejemplo durante la realización de un ejercicio. Los campos de carácter privado e internos deberían estar ocultos para que no puedan ser visualizados por el usuario.

El icono con forma de lápiz permite indicar si se puede modificar su valor o no, es decir, si es editable. Si es un campo de solo lectura (icono en gris) el usuario final solamente podrá ver el valor pero nunca lo podrá editar.

El icono con forma del operador más permite añadir nuevos subcampos al campo actual. El procedimiento es idéntico al realizado previamente cuando se añadió un nuevo campo al nodo seleccionado.

El icono con forma del operador menos permite eliminar el campo seleccionado del elemento padre, ya sea este un nodo o un campo.

Finalmente, el icono con forma de letra i permite mostrar un mensaje emergente con la información relacionada con el campo.

Como se comentó anteriormente, una vez concluida la creación del corpus, puede ser guardado localmente mediante el botón *Exportar* o enviarlo al servidor mediante el botón *Enviar*.

Hay un par requisitos que deben cumplir todos los corpus para ser empleados correctamente cuando se quiera realizar un ejercicio.

- El nodo sesión del corpus tiene que ser hijo del nodo raíz.
- Tiene que tener a su vez un campo que especifique el inicio de la muestra y otro que indique el fin. En caso de no tenerlo, nunca se considerará activa y no podrá ser realizada por un estudiante.

11.2.4. Editar un corpus

Esta acción solamente la pueden realizar usuarios autentificados ante el sistema y que posean el rol de gestor.

Para editar un corpus hay que seleccionar la opción ***Editar corpus*** del menú lateral de la página principal. Una vez pulsado aparecerá una ventana con tres opciones distintas.

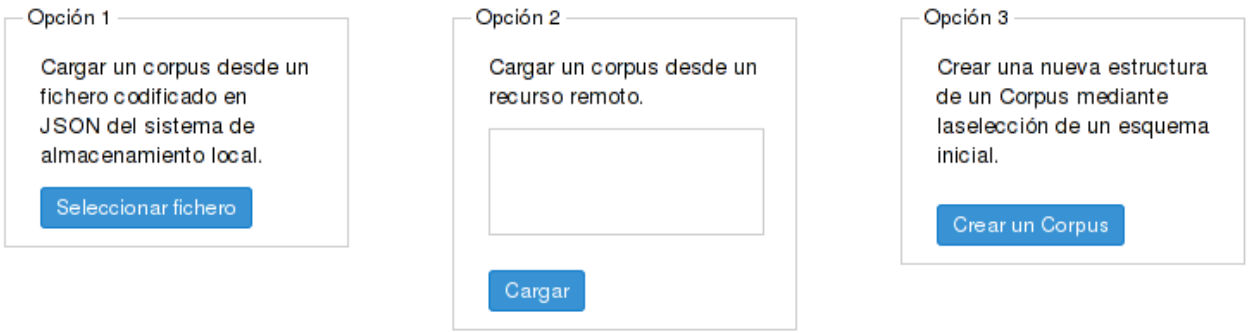


Figura 60: Opciones del editor de corpus

La anterior figura explica cada una de las opciones posibles. Independientemente cual de las tres se haya seleccionado, el comportamiento posterior es muy parecido al que se ha visto en la creación de corpus, concretamente en lo relacionado con la manipulación de nodos y campos.

La única diferencia es que si el corpus a editar existe previamente en el servidor, el proceso de envío ya no manda todo la estructura sino que realiza una sincronización de las distintas modificaciones que se vayan realizando.

11.3. Contenido del CD-ROM

Junto con el presente documento, se adjunta un CD-ROM como soporte común de almacenamiento del proyecto. Dicho soporte posee los siguientes artefactos:

- Fichero único de la versión en formato PDF/A de la memoria del trabajo fin de grado (el presente documento) con nombre *memoria.pdf*.
- Directorio “Código” con todo el código fuente del sistema de Corpus. Posee dos subdirectorios que sirven para dividir el código fuente relacionado con la aplicación y el servidor. Contiene además la documentación propia de cada sistema.
- Directorio “Instalador” que contiene el instalador del sistema Corpus.
- Directorio “Documentación adicional” con documentación relevante en formato electrónico.
    - Fichero Astah con los diagramas empleados en esta memoria. También estarán exportados como imágenes en formato PNG.
    - Recursos multimedia empleados en esta memoria en tamaño y formato original.