Universidad Complutense de Madrid

Facultad de Informática

Proyecto de Sistemas Informáticos 2010/2011

Desarrollo de una aplicación Web para la gestión de Entornos Virtuales

Autores:

Jesús Matías Almaraz Hernández

Pablo Campos Cantero

Tamara Castelo Delgado

Profesor director:

Rubén Manuel Santiago Montero

Madrid, 2011

ÍNDICE

1. Cesión de derechos

Autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Madrid, a 1 de Julio de 2011.

Jesús Matías Almaraz Hernández

Pablo Campos Cantero

Tamara Castelo Delgado

2. Agradecimientos

Queremos agradecer a nuestro director de proyecto, Rubén, por su apoyo y ánimo constantes durante todo el curso en la realización de este proyecto.

Y obviamente a nuestros familiares y amigos, que tanto preguntaban una y otra vez de qué iba el proyecto sin entender nada.

En definitiva, a todos aquellos a los que no hemos podido dedicar todo el tiempo que se merecen durante este duro año. Prometemos devolveros todo lo que nos habéis dado.

3. Resumen

El objetivo de este proyecto es proporcionar a un potencial usuario una aplicación para la gestión de entornos virtuales sobre las que se realizarán las prácticas de las asignaturas de una determinada titulación dada. A su vez, distinguimos tres tipos o niveles diferentes de usuarios: alumnos, profesores y administradores. Los cuales tendrán acceso a diferentes funcionalidades y recursos en función de su nivel de autenticación en el sistema.

Esta interfaz de usuario con la que se proveerá a la aplicación es del tipo Web, siendo así accesible e intuitiva de cara a los posibles usuarios, ya que destaca por su claridad y fácil uso de la misma.

Como es lógico, la aplicación dispone de una base de datos en la que se gestionen y manejen todos los datos correspondientes a los diferentes alumnos, profesores y administradores. Así pues, amén de tener que comunicarse la interfaz con la base de datos para la autenticación de usuarios, en la aplicación es posible la realización de consultas y modificaciones en la base de datos a través de la interfaz.

Palabras clave: JSON, Sinatra, Ruby, Sequel, Cloud Computing, Aplicación Web, jQuery, AJAX.

4. Abstract

The goal of this project is to provide to a potential user an application for the management of virtual environments that will be of use in the performance of different activities required in a course of a given degree. Along with it, we distinguish three different types or levels of users: students, teachers and administrators, to which we will give access to different functionalities and resources depending on their level of authentication in the system.

The provided user's interface is web type, being in this way accessible and intuitive to the future users. With it we offer an application that stands out for its clarity and simplicity of use.

As it is expected, the system counts with a data base where all the specifics data of students, teachers and administrators is managed and administered. Therefore, in order to communicate the interface with the data base for the users authentication, the application provides the possibility of performing consults and modifications in the data base through the interface.

Keywords: JSON, Sinatra, Ruby, Sequel, Cloud Computing, Web Application, jQuery, AJAX.

5. Introducción y objetivos

Este proyecto gira en torno a la implementación de una nueva aplicación que gestione las asignaturas de laboratorio de una titulación universitaria. En este momento solo presentamos el entorno Web y su comunicación con la base de datos pero la aplicación completa está pensada para que, mediante Cloud Computing, pudiésemos manejar los ordenadores de, por ejemplo, los laboratorios de la Facultad de Informática de la UCM. Así pues, al identificarse un alumno en el sistema, en su sesión tendría disponibles todas las máquinas virtuales de las asignaturas de laboratorio en las que el alumno está matriculado y que aparecen debidamente registradas en la base de datos de la aplicación. La primera pregunta que nos viene a la cabeza es sencilla. ¿Qué es Cloud Computing exactamente?

5.1 Cloud Computing

Traducido del inglés “computación en nube”. En este tipo de computación todo se ofrece como servicio, de modo que los usuarios puedan acceder a los servicios disponibles en la nube de Internet sin necesidad de entrar en la gestión de los recursos que usan.

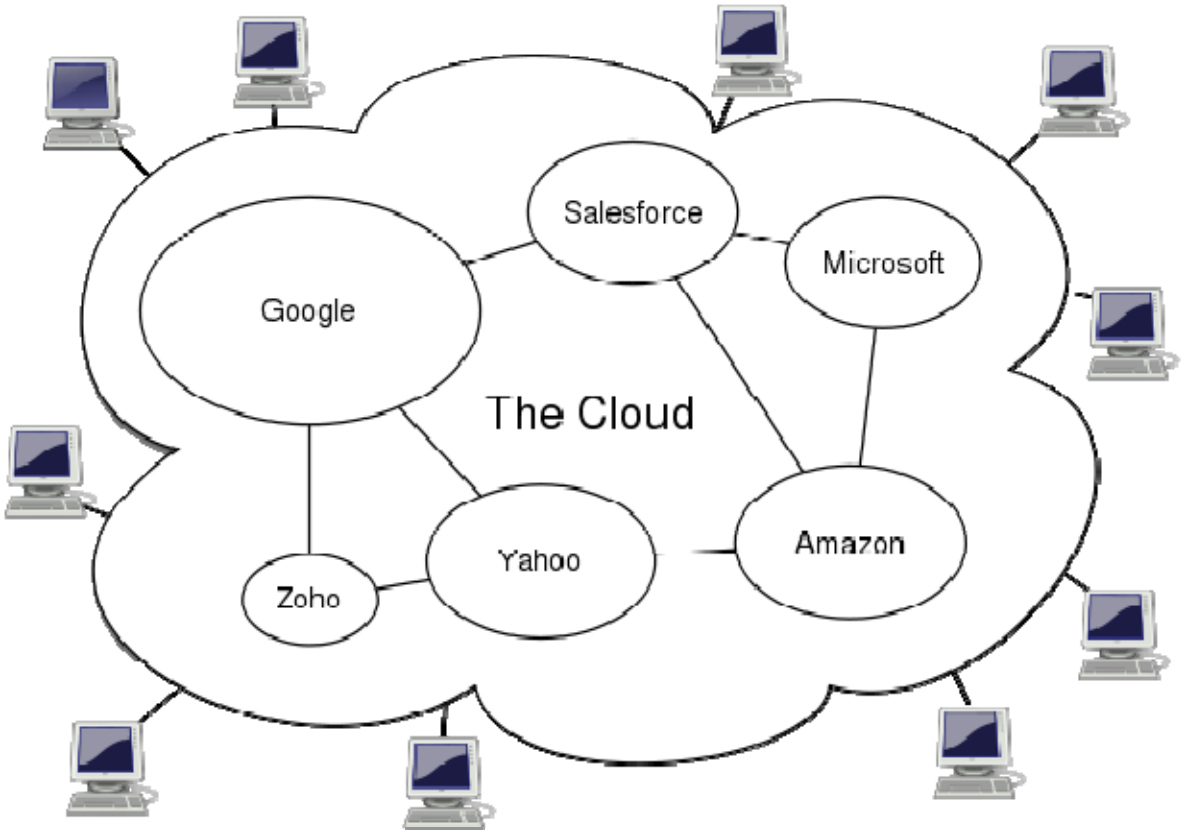
Es un paradigma en el que la información se almacena de manera permanente en servidores en Internet y se envía a cachés temporales de cliente.

Cloud Computing se refiere tanto a las aplicaciones entregadas como a servicios a través de Internet, como al hardware y los sistemas software en centros de datos que proporcionan estos servicios.

Alguno de los aspectos novedosos del Cloud Computing es por ejemplo el generarle al usuario la ilusión de que dispone de infinitos recursos de computación y que estos estarán disponibles según los vaya pidiendo.

Otra característica importante es que los usuarios finales pueden acceder al servicio en cualquier momento y en cualquier lugar, compartir datos y colaborar con más facilidad.

Algunos ejemplos de Cloud Computing son: AmazonWeb Services, Google AppEngine y Microsoft Azure.



Ventajas:

- Integración probada de servicios Red. Por su naturaleza, la tecnología de "Cloud Computing" se puede integrar con mucha mayor facilidad y rapidez con el resto de sus aplicaciones empresariales

- Prestación de servicios a nivel mundial. Las infraestructuras de "Cloud Computing" proporcionan mayor capacidad de adaptación, recuperación de desastres completa y reducción al mínimo de los tiempos de inactividad.

- Una infraestructura 100% de "Cloud Computing" no necesita instalar ningún tipo de hardware. La belleza de la tecnología de "Cloud Computing" es su

simplicidad… y el hecho de que requiera mucha menor inversión para empezar a trabajar.

- Implementación más rápida y con menos riesgos. Podrá empezar a trabajar muy rápidamente gracias a una infraestructura de "Cloud Computing". No tendrá que volver a esperar meses o años e invertir grandes cantidades de dinero antes de que un usuario inicie sesión en su nueva solución. Sus aplicaciones en tecnología de "Cloud Computing" estarán disponibles en cuestión de semanas o meses, incluso con un nivel considerable de personalización o integración.

- Actualizaciones automáticas. Si actualizamos a la última versión de la aplicación, nos veremos obligados a dedicar tiempo y recursos a volver a crear nuestras personalizaciones e integraciones. La tecnología de "Cloud Computing" no le obliga a decidir entre actualizar y conservar su trabajo, porque esas personalizaciones e integraciones se conservan automáticamente durante la actualización.

- Contribuye al uso eficiente de la energía. En este caso, a la energía requerida para el funcionamiento de la infraestructura. En los datacenters tradicionales, los servidores consumen mucha más energía de la requerida realmente. En cambio, en las nubes, la energía consumida es sólo la necesaria, reduciendo notablemente el desperdicio.

Posibles inconvenientes:

- La centralización de las aplicaciones y el almacenamiento de los datos origina una interdependencia de los proveedores de servicios.

- La disponibilidad de las aplicaciones están desatadas a la disponibilidad de acceso a Internet.

- Los datos "sensibles" del negocio no residen en las instalaciones de las empresas por lo que podría generar un contexto de alta vulnerabilidad para la sustracción o robo de información.

- La disponibilidad de servicios altamente especializados podría tardar meses o incluso años para que sean factibles de ser desplegados en la red.

- Seguridad. La información de la empresa debe recorrer diferentes nodos para llegar a su destino, cada uno de ellos (y sus canales) son un foco de inseguridad. Si se utilizan protocolos seguros, HTTPS por ejemplo, la velocidad total disminuye debido a la sobrecarga que requieren estos protocolos.

- Se deja la responsabilidad del almacenamiento de datos y su respectivo control al proveedor de la nube, con los problemas de seguridad que eso pudiera plantear. Por otro lado, se especula que, si el proveedor solamente ofrece determinadas aplicaciones y servicios, estas serán las únicas que se podrán utilizar, poniendo posibles cotas a la libertad del usuario para instalar nuevas aplicaciones.

Tipos de clouds:

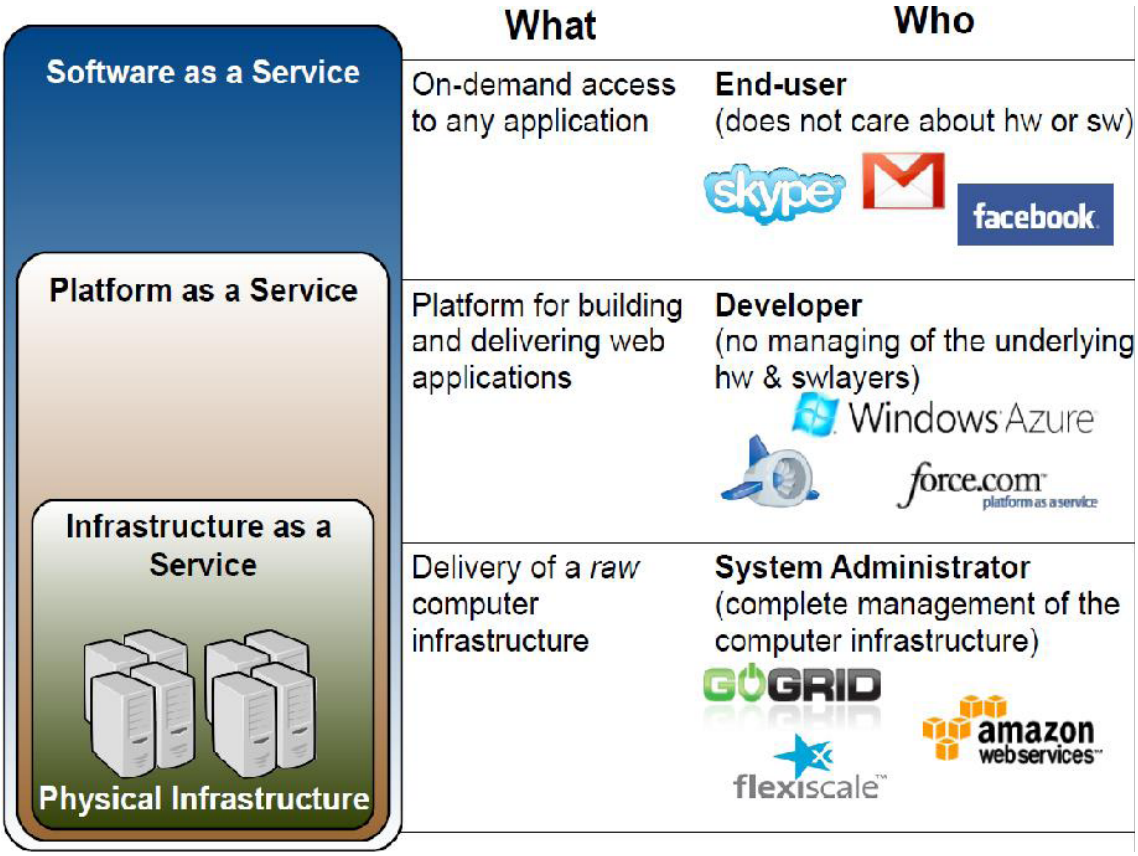
- **Públicos**: En los clouds públicos las infraestructuras del cloud como el servidor y los sistemas de almacenamiento entre otros, son compartidas por diferentes clientes bajo el modelo de pago por uso. Las infraestructuras del cloud son manejadas por terceros. Son infraestructuras virtualizadas, fácilmente accesibles, y gestionadas a través de una interfaz Web que dé autoservicio.

- **Privados:** En los clouds privados las infraestructuras son manejadas por un solo cliente o clientes que es el propietario de las mismas y que decide que usuarios tiene acceso a ellas.

- **Híbridos:** En los clouds híbridos se combinan los dos modelos anteriores. Se posee un cloud privado que se amplía mediante uno público por necesidades de escalabilidad.

Añaden la complejidad de determinar cómo distribuir las aplicaciones a través de estos ambientes diferentes.

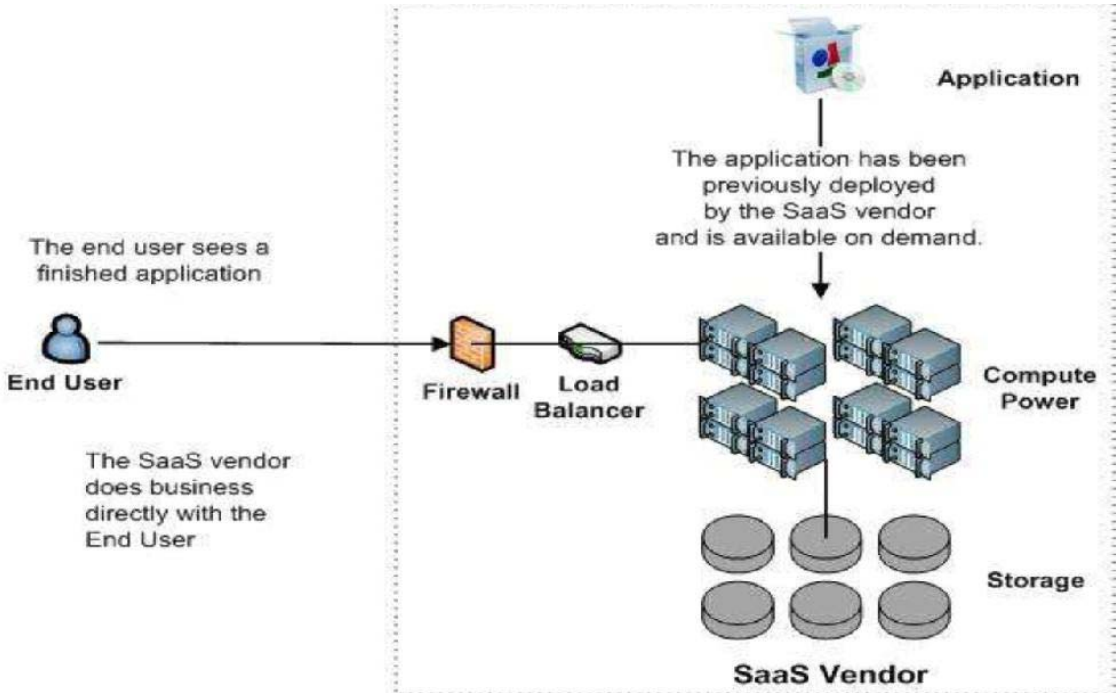
SaaS, IaaS y PaaS: Las tres clases de Cloud Computing



Software as a Service (SaaS): Software como Servicio, comprende la capa más alta. Modelo de distribución de software donde una empresa sirve el mantenimiento, soporte y operación que usará el cliente durante el tiempo que haya contratado el servicio.

Hay una única instancia del software de los servicios que se ejecuta en la infraestructura del proveedor siendo éste el que aloja la información del usuario que se utiliza en las aplicaciones.

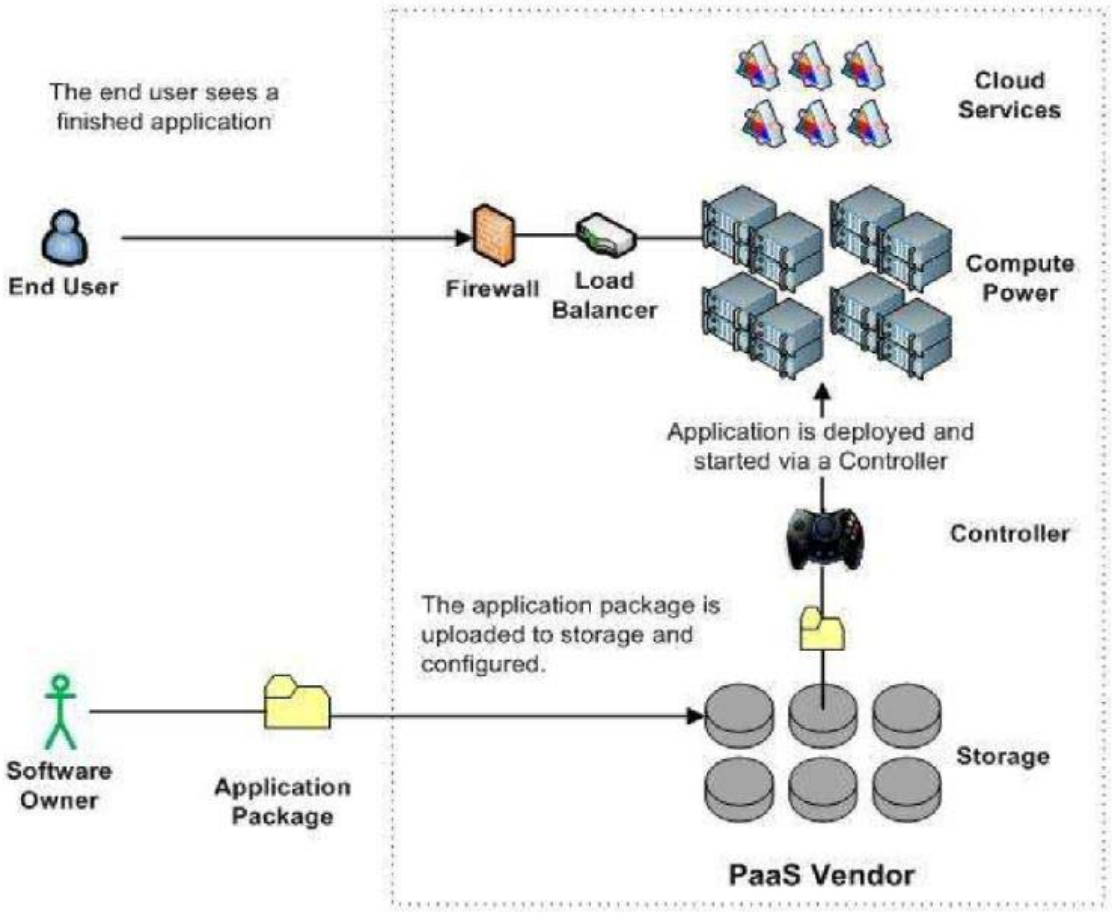
Algunos ejemplos de proveedores de SaaS son Salesforce, Documany, TeamBox, Gupigupi, Kubbos, Basecamp y Gmail, Google Apps



Platform as a Service (PaaS): Plataforma como Servicio, es la capa del medio. Suele identificarse como una evolución de SaaS, es más bien un modelo en el que se ofrece todo lo necesario para soportar el ciclo de vida completo de construcción y puesta en marcha de aplicaciones y servicios Web totalmente disponibles en Internet. Ofrece servicios de desarrollo, testeo, implantación, hosting y mantenimiento de aplicaciones.

Otra característica importante es que no hay descarga de software que instalar en los equipos de los desarrolladores.

Ejemplos: Microsoft Azure, Google App Engine



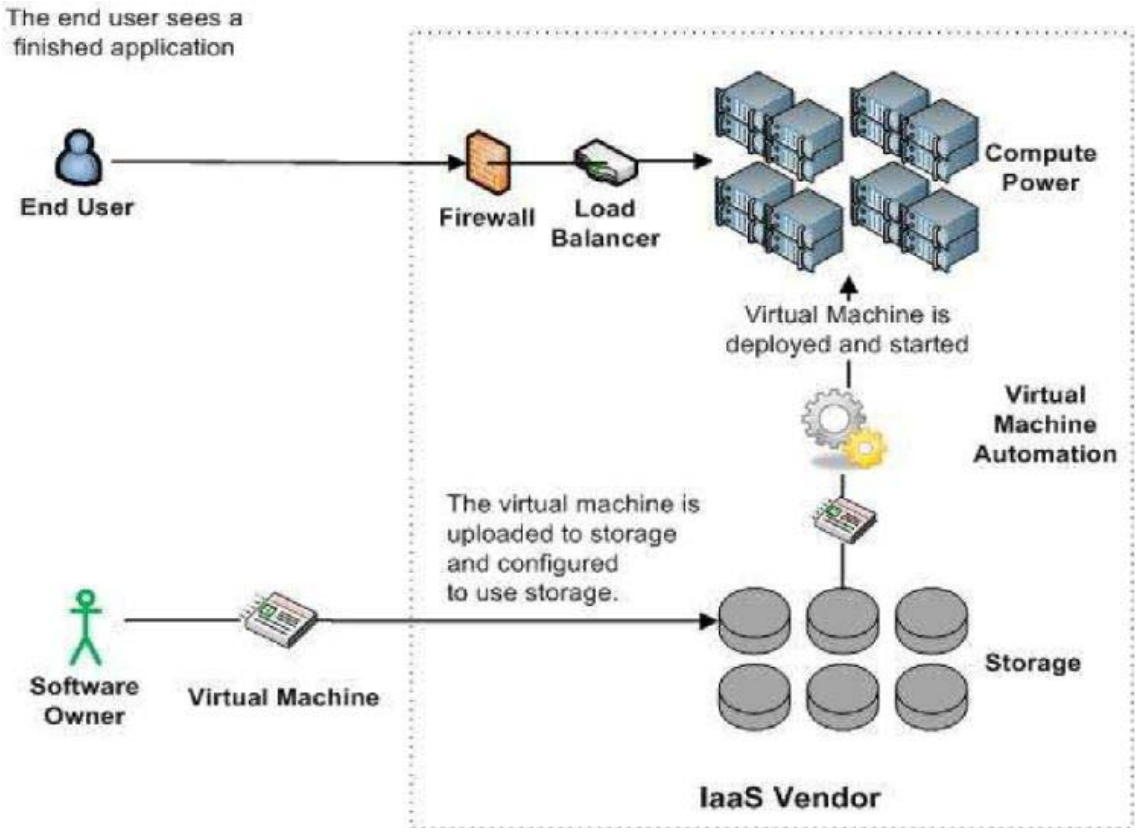
Infrastructure as a Service (IaaS): Infraestructura como Servicio, constituye la capa inferior. Consiste en proporcionar al usuario almacenamiento básico y capacidades de cómputo a través de la red. Para el usuario supone la externalización de esos recursos, sin necesidad de administrarlos ni controlarlos, pagando únicamente por el consumo que se hace de ellos.

En la actualidad hay un número creciente de proveedores que ofrecen soluciones de IaaS con capacidad elástica.

El aprovisionamiento de estos servicios se hace de manera integral a través de la Web.

Ejemplos: El más conocido es Amazon Web Services (con sus servicios EC2 y S3) y GoGrid.

Otro ejemplo más novedoso es Joyent cuyo producto principal es una línea de servidores virtualizados, que proveen una infraestructura en-demanda altamente escalable para manejar sitios Web, incluyendo aplicaciones Web complejas escritas en Ruby, PHP, Python o Java.



5.2 Tecnologías Web

Para la realización de esta aplicación Web se han usado tecnologías vanguardistas como las siguientes que pasamos a enumerar y comentar brevemente:

- **Ruby:** Es un lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad. Orientado totalmente a tratar todo como un objeto.

- **Sequel:** Es el toolkit para las bases de datos de Ruby.

- **JavaScript:** Lenguaje de programación interpretado. Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador Web permitiendo mejoras en la interfaz de usuario y páginas Web dinámicas.

- **CSS:** Es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML).

- **AJAX:** ***A****synchronous* ***J****avaScript* ***A****nd* ***X****ML* (JavaScript asíncrono y XML), es una técnica de desarrollo Web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en background.

- **jQuery:** biblioteca JavaScript. Simplifica la forma de interactuar con documentos HTML y agrega interacción con la técnica AJAX a páginas Web.

- **Sinatra:** Framework para el desarrollo de aplicaciones Web con el lenguaje de programación Ruby.

5.3 Aplicación Web

Se ha hablado anteriormente que este proyecto gira en torno a una aplicación Web, pero… ¿Qué es eso exactamente? Lo explicamos de forma breve y concisa:

Son aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor Web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores Web en la que se confía la ejecución al navegador.

La arquitectura de una aplicación Web, explicada por encima, es la siguiente: normalmente se encuentra estructurada como una aplicación de tres-capas. En su forma más común, el navegador Web ofrece la primera capa y un motor capaz de usar alguna tecnología Web dinámica (ejemplo: PHP o Ruby on Rails) constituye la capa de en medio. Por último, una base de datos constituye la tercera y última capa.
El navegador Web manda peticiones a la capa de en medio que ofrece servicios valiéndose de consultas y actualizaciones a la base de datos y a su vez proporciona una interfaz de usuario.

A partir de los siguientes puntos de esta memoria se desarrollaran mucho más en profundidad todos estos conceptos así como la forma que hemos tenido de integrar los mismos dentro de nuestra aplicación.

6. Base del proyecto

En este apartado comentaremos de forma clara y concisa todas las herramientas y tecnologías que hemos usado para llevar a cabo la aplicación.

6.1 Tecnologías empleadas

6.1.1 Sinatra

Sinatra es un lenguaje de dominio específico (DSL) de Ruby. Un DSL es un lenguaje de programación dedicado a un problema de dominio en particular, o una técnica de representación o resolución de problemas específica. En este caso, Sinatra es una framework para aplicaciones Web, gratuito y de código libre bajo licencia MIT (Massachusetts Institute of Technology), que alcanzó su versión 1.0 el 23 de Marzo de 2010. Surge como alternativa a otros frameworks similares pero mucho mas potentes y destinados a aplicaciones de gran tamaño, como Ruby on Rails, de hecho una de sus grandes características es su sencillez y la simplicidad de uso. Esto permite la elaboración de aplicaciones Web en un corto espacio de tiempo, lo que le hace muy adecuado para dar interfaz Web a servicios ya existentes. Otra de las características de sinatra es permitir en pocas líneas de código la creación de servicios.

Si se desean funciones auxiliares que ayudan a crear formas, conectar a bases de datos, o cualquiera de los miles de otras funciones que ofrece Rails, no se van a encontrar. En cambio, Sinatra es más simple.

Por otro lado, Rails requiere rutas diferencias para definir cómo la aplicación Web va a responder a las solicitudes, que se conecta a los controladores apropiados y modelos. Sinatra funciona de una forma mucho más simple. Cuando se declara un nuevo "get " o "post" de acción, Sinatra se añade automáticamente la ruta, y simplemente comenzará a responder a las peticiones que se le formulen.

6.1.2 JavaScript

JavaScript es un lenguaje de scripting basado en objetos no tipado y liviano, utilizado para acceder a objetos en aplicaciones. Principalmente, se utiliza integrado en un navegador Web permitiendo el desarrollo de interfaces de usuario mejoradas y páginas Web dinámicas.

El lenguaje fue inventado por Brendan Eich en la empresa Netscape Communications, la que desarrolló los primeros navegadores Web comerciales. Apareció por primera vez en el producto de Netscape llamado Netscape Navigator 2.0.

JavaScript ha tenido influencia de múltiples lenguajes y se diseñó con una sintaxis similar al lenguaje de programación Java, aunque es fácil de utilizar para personas que no programan.

Todos los navegadores modernos interpretan el código JavaScript integrado dentro de las páginas Web. Para interactuar con una página Web se provee al lenguaje JavaScript de una implementación del DOM.

Sus características más importantes son:

1.- JavaScript es un lenguaje interpretado, es decir, no requiere compilación. El navegador del usuario se encarga de interpretar las sentencias JavaScript contenidas en una página HTML y ejecutarlas adecuadamente.

2.- JavaScript es un lenguaje orientado a eventos. Cuando un usuario pincha sobre un enlace mueve el puntero sobre una imagen se produce un evento. Mediante JavaScript se pueden desarrollar scripts que ejecuten acciones en respuesta a estos eventos.

3.- JavaScript es un lenguaje orientado a objetos. El modelo de objetos de JavaScript está reducido y simplificado, pero incluye los elementos necesarios para que los scripts puedan acceder a la información de una página y puedan actuar sobre la interfaz del navegador.

6.1.3 **AJAX**

Acrónimo de Asynchronous JavaScript And XML (JavaSript asíncrono y XML) o AJAX, es una técnica de desarrollo Web creada con el fin de crear aplicaciones interactivas. Estas se ejecutan en el navegador de los usuarios y mantienen comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de refrescarla, lo que significa aumentar la interactividad, velocidad y usabilidad en la misma. AJAX es una combinación de cuatro tecnologías ya existentes:

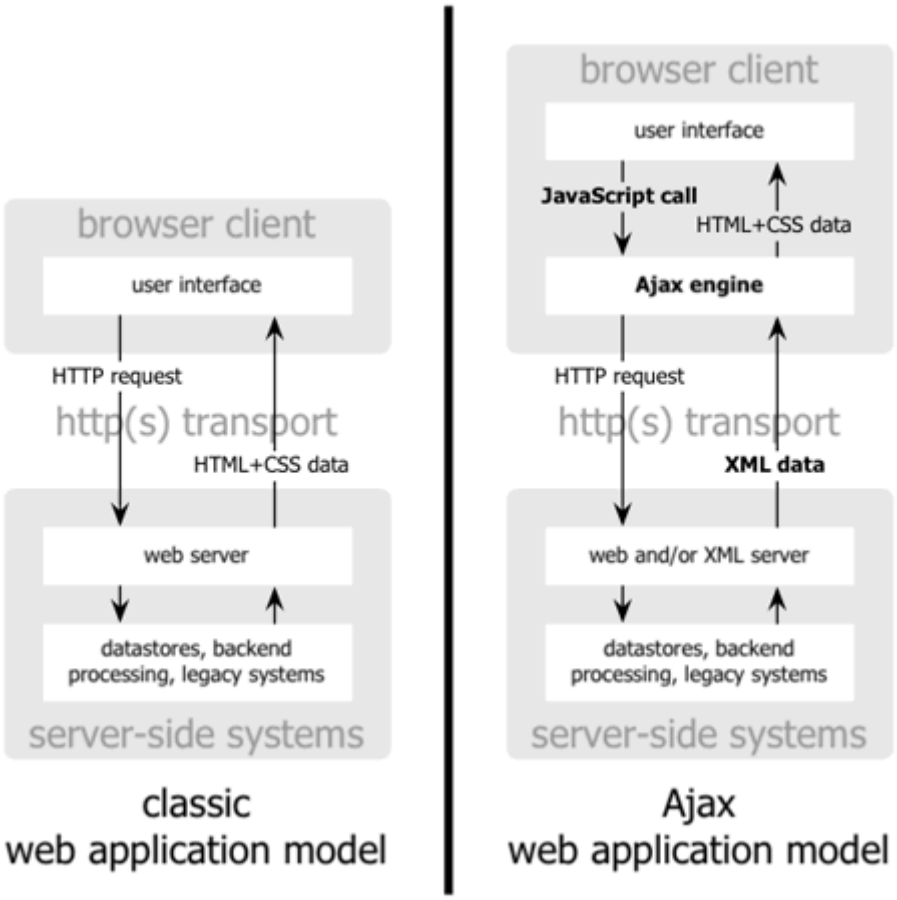
- XHTML (o HTML) y hojas de estilo en cascada (CSS) para el diseño que acompaña a la información.

- Document (DOM) accedido con un lenguaje de scripting por parte del usuario, especialmente implementaciones ECMAScript como JavaScript y JScript, para mostrar e interactuar dinámicamente con la información presentada.

- El objeto XMLHttpRequest para intercambiar datos asincrónicamente con el servidor Web.

- XML es el formato usado comúnmente para la transferencia de vuelta al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSON y hasta EBML.

Cuando se combinan estas tecnologías en el modelo AJAX, las aplicaciones funcionan mucho más rápido, ya que las interfaces de usuario se pueden actualizar por partes sin tener que actualizar toda la página completa. Por ejemplo, al rellenar un formulario de una página Web, con AJAX se puede actualizar la parte en la que se elige el país de residencia sin tener que actualizar todo el formulario o toda la página Web completa.



6.1.4 jQuery

jQuery es una biblioteca o framework gratuito de JavaScript, que permite la realización de programas JavaScript de una forma simple y sencilla, creando páginas Web de las aplicaciones dinámicas complejas. Según su creador John Resig, jQuery es "una librería JavaScript muy rápida y muy ligera que simplifica el desarrollo de la parte de cliente de las aplicaciones Web".

jQuery tiene diversas prestaciones, entre las que destacan: el control de navegador de usuario, que permite despreocuparse de la compatibilidad de los scripts con los distintos navegadores existentes; mayor facilidad en la creación de aplicaciones del lado cliente, es decir, interfaces de usuario, efectos dinámicos o aplicaciones que hacen uso de AJAX.

Todo esto convierte a jQuery en un elemento indispensable para el desarrollo rápido y eficaz de aplicaciones Web, sin perder los detalles visuales ni las necesidades técnicas.

6.1.5 DataTables

DataTables es un plugin para la librería jQuery. Es una herramienta altamente flexible que añade controles de interacción avanzados a cualquier tabla HTML. Entre sus características, se puede encontrar:

- Longitud de paginación variable
- Filtrado en tiempo de ejecución
- Control eficiente de ancho de columnas
- Mostrar información de casi cualquier tipo de fuente DOM, JavaScript array, AJAX y server-side processing (PHP, C#, Perl, Ruby, AIR, Gears etc)
- Completamente internacionalizable.
- Sistema robusto y muy testeado
- Gran variedad de plugins incluidos

6.1.6 Hojas de estilo CSS

CSS es el acrónimo de Cascading Style Sheets, cuyo significado literal es Hojas de Estilo en Cascada. Se utiliza para dar estilo a documentos HTML y XML, separando el contenido de la presentación.

Los estilos definen la forma de mostrar los elementos. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a ella en las que aparezca ese elemento. De esta forma, CSS permite controlar el estilo y formato de múltiples páginas Web al mismo tiempo.

CSS funciona a base de reglas, esto es, declaraciones sobre el estilo de uno o más elementos.

La regla tiene dos partes: un selector y la declaración, estando esta última compuesta por una propiedad y el valor que se le asigne.

El selector funciona como enlace entre el documento y el estilo, especificando los elementos que se van a ver afectados por esa declaración. La declaración es la parte de la regla que establece cuál será el efecto.

6.1.7 Ruby

Lenguaje creado por Yukihiro "Matz" Matsumoto, quien empezó a trabajar en Ruby el 24 de febrero de 1993, y lo presentó al público en el año 1995. En el círculo de amigos de Matsumoto se le puso el nombre de "Ruby" (en español *rubí*) como broma aludiendo al lenguaje de programación "Perl" (*perla*). Diferencias en rendimiento entre la actual implementación de Ruby (1.9.2) y otros lenguajes de programación más arraigados han llevado al desarrollo de varias máquinas virtuales para Ruby. Entre ésas se encuentra JRuby, un intento de llevar Ruby a la plataforma Java.

El creador del lenguaje, Yukihiro "Matz" Matsumoto, ha dicho que Ruby está diseñado para la productividad y la diversión del desarrollador, siguiendo los principios de una buena interfaz de usuario.Sostiene que el diseño de sistemas necesita enfatizar las necesidades humanas más que las de la máquina:

A menudo la gente, especialmente los ingenieros en computación, se centran en las máquinas. Ellos piensan, "Haciendo esto, la máquina funcionará más rápido. Haciendo esto, la máquina funcionará de manera más eficiente. Haciendo esto..." Están centrados en las máquinas, pero en realidad necesitamos centrarnos en las

personas, en cómo hacen programas o cómo manejan las aplicaciones en los ordenadores. Nosotros somos los jefes. Ellos son los esclavos.

Ruby sigue el "principio de la menor sorpresa", lo que significa que el lenguaje debe comportarse de tal manera que minimice la confusión de los usuarios experimentados. Matz ha dicho que su principal objetivo era hacer un lenguaje que le divirtiera él mismo, minimizando el trabajo de programación y la posible confusión. Él ha dicho que no ha aplicado el principio de menor sorpresa al diseño de Ruby. Pero sin embargo la frase se ha asociado al lenguaje de programación Ruby. La frase en sí misma ha sido fuente de controversia, ya que los no iniciados pueden tomarla como que la características de Ruby intentar ser similares a las características de otros lenguajes conocidos. En mayo de 2005 en una discusión en el grupo de noticias comp.lang.ruby, Matz trató de distanciar Ruby de la mencionada filosofía, explicando que cualquier elección de diseño será sorprendente para alguien, y que él usa un estándar personal de evaluación de la sorpresa. Si ese estándar personal se mantiene consistente habrá pocas sorpresas para aquellos familiarizados con el estándar.

Ruby es orientado a objetos: todos los tipos de datos son un objeto, incluidas las clases y tipos que otros lenguajes definen como primitivas, (como enteros, booleanos, y "nil"). Toda función es un método. Las variables siempre son referencias a objetos, no los objetos mismos. Ruby soporta herencia con enlace dinámico, mixins y métodos Singleton (pertenecientes y definidos por un sola instancia más que definidos por la clase). A pesar de que Ruby no soporta herencia múltiple, la clases pueden importar módulos como mixins. La sintaxis procedural está soportada, pero todos los métodos definidos fuera del ámbito de un objeto son realmente métodos de la clase Object. Como esta clase es padre de todas las demás, los cambios son visibles para todas las clases y objetos.

Ruby ha sido descrito como un lenguaje de programación multiparadigma: permite programación procedural (definiendo funciones y variables fuera de las clases haciéndolas parte del objeto raíz Object), con orientación a objetos, (todo es un

objeto) o funcionalmente (tiene funciones anónimas, clausuras o closures, y continuations; todas las sentencias tiene valores, y las funciones devuelven la última evaluación). Soporta introspección, reflexión y metaprogramación, además de soporte para hilos de ejecución gestionados por el intérprete. Ruby tiene tipado dinámico, y soporta polimorfismo de tipos (permite tratar a subclases utilizando la interfaz de la clase padre). Ruby no requiere de polimorfismo de funciones al no ser fuertemente tipado (los parámetros pasados a un método pueden ser de distinta clase en cada llamada a dicho método).

6.1.8 **Sequel**

Sequel nos sirve para manejar las bases de datos desarrolladas en Ruby. Y ofrece seguridad para subprocesos, la agrupación de conexiones DSL y un conciso para la construcción de consultas SQL y esquemas de tabla. Por otra parte incluye una capa ORM para la asignación de registros a objetos Ruby y manejo de los registros asociados.

También soporta características avanzadas de base de datos, tales como declaraciones preparadas, variables ligadas, procedimientos almacenados, puntos de retorno, aislamiento de la transacción, configuraciones maestro-esclavo…

Finalmente, indicar que Sequel cuenta actualmente con adaptadores para ADO, Amalgalite, DataObjects, DB2, DBI, Firebird, Informix, JDBC, MySQL, Mysql2, ODBC, OpenBase, Oracle, PostgreSQL, SQLite3, y Swift.

6.1.9 XML

Siglas en inglés de *e**X**tensible **M**arkup **L**anguage* ('lenguaje de marcas extensible'), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

XML proviene de un lenguaje inventado por IBM en los años setenta, llamado GML (*Generalized Markup Language*), que surgió por la necesidad que tenía la empresa de almacenar grandes cantidades de información. Este lenguaje gustó a la ISO, por lo que en 1986 trabajaron para normalizarlo, creando SGML (*Standard Generalized Markup Language*. A partir de él se han creado otros sistemas para almacenar información.

El XML se caracteriza por su extensibilidad, sencillez, por su alta compatibilidad entre diferentes aplicaciones y flexibilidad para estructurar documentos.

Un documento XML está formado por el prólogo y por el cuerpo del documento así como texto de etiquetas que contiene una gran variedad de efectos positivos o negativos en la referencia opcional a la que se refiere el documento, hay que tener mucho cuidado de esa parte de la gramática léxica para que se componga de manera uniforme.

A continuación pasamos a enumerar y comentar breve, pero más detalladamente, las partes de un documento XML.

Partes de un documento XML:

- **Prólogo**

Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas.

- **Cuerpo**

A diferencia del prólogo, el cuerpo no es opcional en un documento XML, el cuerpo debe contener un y solo un elemento raíz, característica indispensable también para que el documento esté bien formado.

- **Elementos**

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

- **Atributos**

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Deben ir entre comillas.

- **Entidades predefinidas**

Entidades para representar caracteres especiales para que, de esta forma, no sean interpretados como marcado en el procesador XML.

6.2 Aplicación Web

6.2.1 ¿Qué es una aplicación Web?

Las aplicaciones Web son aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor Web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores Web en la que se confía la ejecución al navegador.

Las aplicaciones Web son populares debido a lo práctico del navegador Web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones Web sin distribuir e instalar software a miles de usuarios potenciales. Existen aplicaciones como los webmails, wikis, weblogs, tiendas en línea y la propia Wikipedia que son ejemplos bien conocidos de aplicaciones Web.

Es importante mencionar que una página Web puede contener elementos que permiten una comunicación activa entre el usuario y la información. Esto permite que el usuario acceda a los datos de modo interactivo, gracias a que la página responderá a cada una de sus acciones, como por ejemplo rellenar y enviar formularios, participar en juegos diversos y acceder a gestores de base de datos de todo tipo.

A diferencia de los primeros tiempos de la computación cliente servidor, donde cada aplicación tenía que ser instalado independientemente en cada ordenador personal y los programas clientes servían como interfaz de usuario, las aplicaciones Web generan dinámicamente una serie de páginas en un formato estándar, como HTML o XHTML, soportados por los navegadores Web comunes. Se utilizan lenguajes interpretados en el lado del cliente, directamente o a través de plugins tales como JavaScript, Java, Flash, etc., para añadir elementos dinámicos a la interfaz de usuario. Generalmente cada página Web en particular se envía al cliente como un documento estático, pero la secuencia de páginas ofrece al usuario una experiencia interactiva. Durante la sesión, el navegador Web interpreta y muestra en pantalla las páginas, actuando como cliente para cualquier aplicación Web.

Las interfaces Web tienen ciertas limitaciones en las funcionalidades que se ofrecen al usuario. Hay funcionalidades comunes en las aplicaciones de escritorio como dibujar en la pantalla o arrastrar-y-soltar que no están soportadas por las tecnologías Web estándar. Los desarrolladores Web generalmente utilizan lenguajes interpretados (scripts) en el lado del cliente para añadir más funcionalidades, especialmente para ofrecer una experiencia interactiva que no requiera recargar la página cada vez (lo que suele resultar molesto a los usuarios). Recientemente se han

desarrollado tecnologías para coordinar estos lenguajes con las tecnologías en el lado del servidor. Como ejemplo, el mencionado anteriormente AJAX es una técnica de desarrollo Web que usa una combinación de varias tecnologías.

Ventajas

- Ahorra tiempo: Se pueden realizar tareas sencillas sin necesidad de descargar ni instalar ningún programa.
- No hay problemas de compatibilidad: No hace falta crear diferentes clientes el función de cada sistema operativo. Basta tener un navegador actualizado para poder utilizarlas.
- No ocupan espacio en nuestro disco duro.
- Actualizaciones inmediatas: Como el software lo gestiona el propio desarrollador, cuando nos conectamos estamos usando siempre la última versión que haya lanzado.
- Consumo de recursos bajo: Dado que toda (o gran parte) de la aplicación no se encuentra en nuestro ordenador, muchas de las tareas que realiza el software no consumen recursos nuestros porque se realizan desde otro ordenador.
- Multiplataforma: Se pueden usar desde cualquier sistema operativo porque sólo es necesario tener un navegador.
- Portables: Es independiente del ordenador donde se utilice (un PC de sobremesa, un portátil...) porque se accede a través de una página Web (sólo es necesario disponer de acceso a Internet). La reciente tendencia al acceso a las aplicaciones Web a través de teléfonos móviles requiere sin embargo un diseño específico de los ficheros CSS para no dificultar el acceso de estos usuarios.
- La disponibilidad suele ser alta porque el servicio se ofrece desde múltiples localizaciones para asegurar la continuidad del mismo.

- Los virus no dañan los datos porque éstos están guardados en el servidor de la aplicación.

- Colaboración: Gracias a que el acceso al servicio se realiza desde una única ubicación es sencillo el acceso y compartición de datos por parte de varios usuarios. Tiene mucho sentido, por ejemplo, en aplicaciones online de calendarios u oficina.

Inconvenientes

- Habitualmente ofrecen menos funcionalidades que las aplicaciones de escritorio. Se debe a que las funcionalidades que se pueden realizar desde un navegador son más limitadas que las que se pueden realizar desde el sistema operativo. Pero cada vez los navegadores están más preparados para mejorar en este aspecto.

- Al igual que lo que mencionábamos previamente cuando estábamos tratando las pegas del Cloud Computing, aquí ocurre algo análogo. La disponibilidad depende de un tercero, el proveedor de la conexión a Internet o el que provee el enlace entre el servidor de la aplicación y el cliente. Así que la disponibilidad del servicio está supeditada al proveedor.

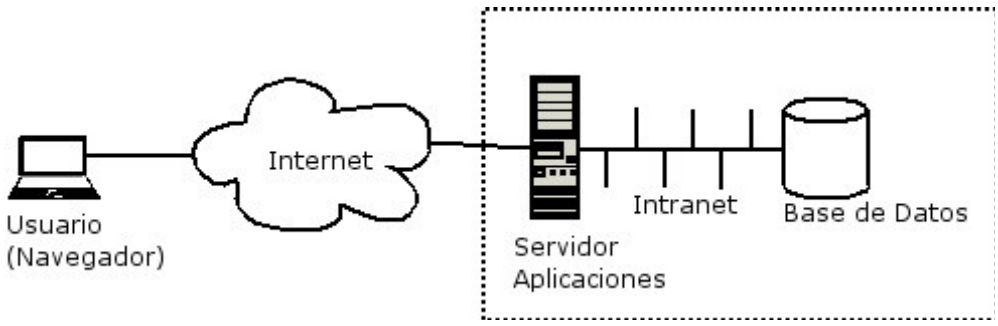
Los lenguajes de programación empleados para el desarrollo de apliaciones Web son diversos, pero destacan, entre otros, los siguientes: PHP, JavaScript, Perl, Ruby, Phyton, XML o ASP.NET. Aunque este último no es un lenguaje de programación como tal, sino una arquitectura de desarrollo Web en la que se pueden usar por debajo diferentes lenguajes.

6.2.2 Arquitectura de una aplicación Web:

Hace algún tiempo, los sitios Web tradicionales que se limitaban a mostrar información se han convertido en aplicaciones capaces de una interacción más o

menos sofisticada con el usuario. Inevitablemente, esto ha provocado un aumento progresivo de la complejidad de estos sistemas y, por ende, la necesidad de buscar opciones de diseño nuevas que permitan dar con la arquitectura óptima que facilite la construcción de los mismos.

El usuario interacciona con las aplicaciones Web a través del navegador. Como consecuencia de la actividad del usuario, se envían peticiones al servidor, donde se aloja la aplicación y que normalmente hace uso de una base de datos que almacena toda la información relacionada con la misma. El servidor procesa la petición y devuelve la respuesta al navegador que la presenta al usuario. Por tanto, el sistema se distribuye en tres componentes: el navegador, que presenta la interfaz al usuario; la aplicación, que se encarga de realizar las operaciones necesarias según las acciones llevadas a cabo por éste y la base de datos, donde la información relacionada con la aplicación se hace persistente. Esta distribución se conoce como el modelo o arquitectura de tres capas.



En la mayoría de los casos, el navegador suele ser un mero presentador de información (modelo de cliente delgado), y no lleva a cabo ningún procesamiento relacionado con la lógica de negocio. No obstante, con la utilización de applets, código JavaScript y DHTML la mayoría de los sistemas se sitúan en un punto intermedio entre un modelo de cliente delgado y un modelo de cliente grueso (donde el cliente realiza el procesamiento de la información y el servidor sólo es responsable de la administración de datos). No obstante el procesamiento realizado

en el cliente suele estar relacionado con aspectos de la interfaz (como ocultar o mostrar secciones de la página en función de determinados eventos) y nunca con la lógica de negocio.

En todos los sistemas de este tipo y ortogonalmente a cada una de las capas de despliegue comentadas, podemos dividir la aplicación en tres áreas o niveles:

1. **Nivel de presentación**: es el encargado de generar la interfaz de usuario en función de las acciones llevadas a cabo por el mismo.

2. **Nivel de negocio**: contiene toda la lógica que modela los procesos de negocio y es donde se realiza todo el procesamiento necesario para atender a las peticiones del usuario.

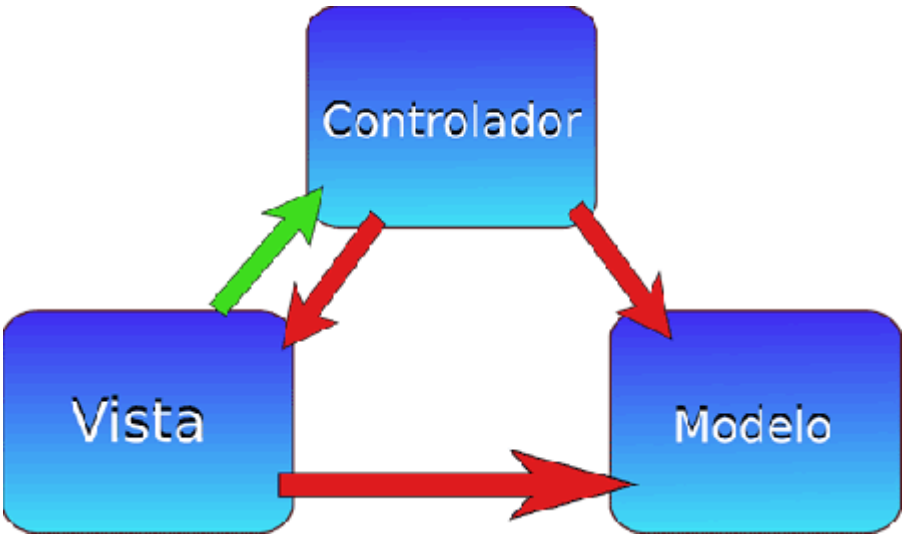
3. **Nivel de administración de datos**: encargado de hacer persistente toda la información, suministra y almacena información para el nivel de negocio.

Los dos primeros y una parte del tercero (el código encargado de las actualizaciones y consultas), suelen estar en el servidor mientras que la parte restante del tercer nivel se sitúa en la base de datos (notar que, debido al uso de procedimientos almacenados en la base de datos, una parte del segundo nivel también puede encontrarse en la misma). Teniendo en cuenta estas características en la arquitectura de los sistemas Web, a continuación veremos algunos patrones de diseño de aplicación básica que pueden facilitar un diseño apropiado para este tipo de sistemas.

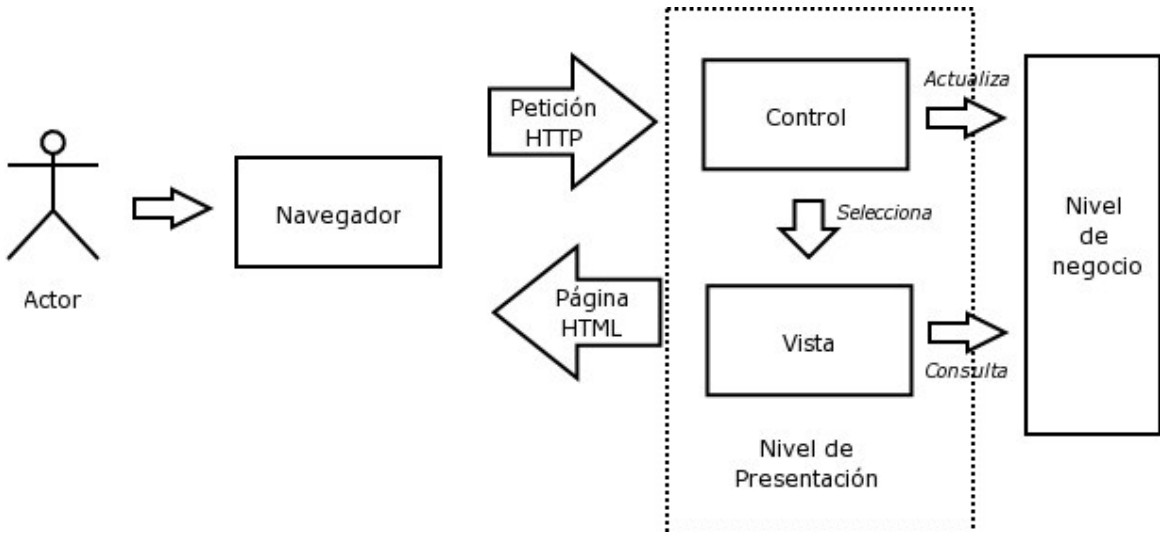
Uno de los patrones que ha demostrado ser fundamental a la hora de diseñar aplicaciones Web es el patrón **Modelo-Vista-Controlador (MVC)**. Este patrón propone la separación en distintos componentes de la interfaz de usuario (vistas), el modelo de negocio y la lógica de control. Una vista es una "fotografía" del modelo (o una parte del mismo) en un determinado momento. Un control recibe un evento disparado por el usuario a través de la interfaz, accede al modelo de manera adecuada a la acción realizada, y presenta en una nueva vista el resultado de dicha

acción. Por su parte, el modelo consiste en el conjunto de objetos que modelan los procesos de negocio que se realizan a través del sistema.

El modelo no tiene que acceder ni a la vista ni al controlador. La vista tiene que poder acceder al modelo (obviamente para representarlo) y al controlador (para enviar las entradas que hace el usuario). Y el controlador ha de acceder al modelo (para conocer los datos y eventualmente pedir el cambio de estos) y a la vista para indicar los cambios en los datos.



En una aplicación Web, las vistas serían las páginas HTML que el usuario visualiza en el navegador. A través de estas páginas el usuario interactúa con la aplicación, enviando eventos al servidor a través de peticiones HTTP. En el servidor se encuentra el código de control para estos eventos, que en función del evento concreto actúa sobre el modelo convenientemente. Los resultados de la acción se devuelven al usuario en forma de página HTML mediante la respuesta HTTP.



La clave está en la separación entre vista y modelo. El modelo suele ser más estable a lo largo del tiempo y menos sujeto a variaciones mientras que las vistas puede cambiar con frecuencia, ya sea por cambio del medio de presentación (por ejemplo HTML a WAP o a PDF) o por necesidades de usabilidad de la interfaz o simple renovación de la estética de la aplicación. Con esta clara separación las vistas pueden cambiar sin afectar al modelo y viceversa. Los controladores son los encargados de hacer de puente entre ambos, determinando el flujo de salida de la aplicación (qué se ve en cada momento).

Si tomamos como referencia la plataforma J2EE, las vistas podrían ser JSPs (o plantillas Velocity o documentos XML tratados con XSLT, ...) los controladores serían servlets y el modelo podría implementarse utilizando EJBs u objetos Java normales en combinación con frameworks de persistencia como Hibernate o JDO.

A la hora de utilizar el MVC en aplicaciones Web es conveniente utilizar un único servlet como controlador para toda la aplicación. Este control gestiona todas las peticiones, incluyendo invocaciones a servicios de seguridad, gestión de

excepciones, selección de la siguiente vista, etc. Esto se conoce como el patrón Front Controller (controlador frontal o fachada). El poder centralizar en un solo punto servicios como la gestión de conexiones a base de datos, comprobaciones de seguridad o gestión de errores favorecen que la aplicación sea mucho más robusta y aísla de todos estos aspectos al resto de componentes.

Si aplicamos esto junto con el patrón Command podemos seguir lo que se conoce como estrategia Comando y Controlador. En este caso el componente Control (entendido dentro del marco MVC y no como el servlet controlador de este caso concreto) está formado por el servlet que recibe las peticiones y por un conjunto de clases implementadas a través del patrón Command en las que delega las tareas a llevar a cabo según la acción invocada. Estos comandos seleccionan la siguiente vista en función de los resultados de su procesamiento y el servlet controlador sirve esta vista al cliente.

La utilización de esta estrategia tiene varias ventajas: permite cambiar fácilmente el procesamiento para una acción determinada sustituyendo el comando que la implementa por otro, permite reutilizar comandos y favorece el encadenamiento de dos o más comandos para la implementación de tareas complejas.

Actualmente existen varios frameworks de desarrollo de aplicaciones Web basados en el patrón MVC como son Struts, Spring, WebWork o Maverick entre otros. El de mayor difusión en la actualidad es Struts. Este framework (al igual que muchos otros) utiliza la estrategia que hemos visto.

Si recordamos los tres niveles en los que dividimos una aplicación Web (presentación, negocio, administración de datos), lo que hemos dicho hasta ahora afecta fundamentalmente al nivel de presentación y, en algunos casos, al nivel de negocio. Respecto a este último no hay ninguna “receta” para asegurar un buen diseño. La aplicación de patrones de diseño conocidos y el respeto a los principios de encapsulación de información y distribución de responsabilidad (que cada objeto haga solo aquello que le es propio) es la mejor manera de conseguir un diseño apropiado.

Un principio que suele resultar de bastante utilidad es agrupar en servicios las operaciones de negocio relacionadas.

Por último hablaremos brevemente del nivel de administración de datos. Este último nivel es proporcionado por el framework de persistencia utilizado junto con la base de datos propiamente dicha. Actualmente existen diversos frameworks de persistencia de datos (Entity beans, Hibernate, JDO, OJB, etc.) que realizan automáticamente el mapeo entre objetos de la aplicación y tablas en la base de datos relacional o incluso podríamos optar por utilizar JDBC.

Cada uno tiene unas características y funcionalidad concretas que obligarán a adaptar el diseño de manera apropiada. Solo apuntar que normalmente, en una aplicación Web una petición HTTP equivale a una transacción. Es decir, si mientras se sirve la petición ocurre un error todo lo hecho a raíz de esa petición debe deshacerse. Por tanto, en función del modelo de persistencia, habrá que actuar de manera que los cambios a la base de datos no se hagan definitivos hasta que la petición se haya completado. La mejor forma de gestionar esto es en el servlet controlador, pues al ser el que finaliza la petición enviando la respuesta al cliente, es el mejor lugar para estar seguros de que todo ha ido bien y hacer definitivos los cambios en la base de datos.

7. Arquitectura y diseño del sistema

En este apartado pasaremos a comentar la estructura de nuestro proyecto, cómo se ha diseñado y de qué manera hemos aplicado las diferentes tecnologías comentadas en el punto anterior.

7.1 Funcionamiento general de la aplicación

Nuestra aplicación consta principalmente de una base de datos, donde se almacena toda la información referente a la aplicación y cuatro interfaces o páginas Web.

En primer lugar, al arrancar la aplicación, nos encontramos con una de las cuatro interfaces que la componen, la de inicio de sesión “Campus-Cloud Login”.

Para poder iniciar una sesión, debemos estar registrados en la base de datos que posee nuestra aplicación. Después en la parte superior derecha de la página, tendremos que poner nuestro nombre de usuario en la casilla “Username” y luego escribir la contraseña en la casilla correspondiente “Password”.

La contraseña o clave es una forma de autenticación que utiliza información secreta para controlar el acceso hacia algún recurso, en este caso hacia la página que se nos redirige, que dependerá del rol que tengamos, ya seamos profesores, estudiantes o administradores. La contraseña se nos pedirá siempre al inicio de sesión, negándonos el acceso siempre y cuando no la conozcamos o hayamos cometido algún error al introducirla.

Otra forma de no tener que escribir siempre los datos de inicio de sesión es, la de seleccionar la opción, “Remember me”, la cual permite mediante el uso de *cookies*, que la próxima vez que ingresemos en nuestra aplicación se accederá directamente a la página de inicio sin necesidad de ingresar el usuario y la contraseña.

Una vez creada la sesión, se le asigna al usuario un nivel de autorización (estudiante, profesor o administrador) del que depende que se nos redirija a una página u otra. Éstas son:

La página del profesorado “Campus-Cloud Teacher”, donde las opciones que se ofrecen a los profesores son las de list, create y delete, con las que podrán gestionar los laboratorios qué más les interesen para sus estudiantes. Podrán consultar la lista de laboratorios almacenados en la base de datos o aquellos nuevos que se vayan creando así como borrarlos.

La página del estudiante “Campus-Cloud Student”, permite a los estudiantes consultar todos los laboratorios almacenados en la base de datos para crear así una conexión con la máquina virtual y poder trabajar con los elegidos.

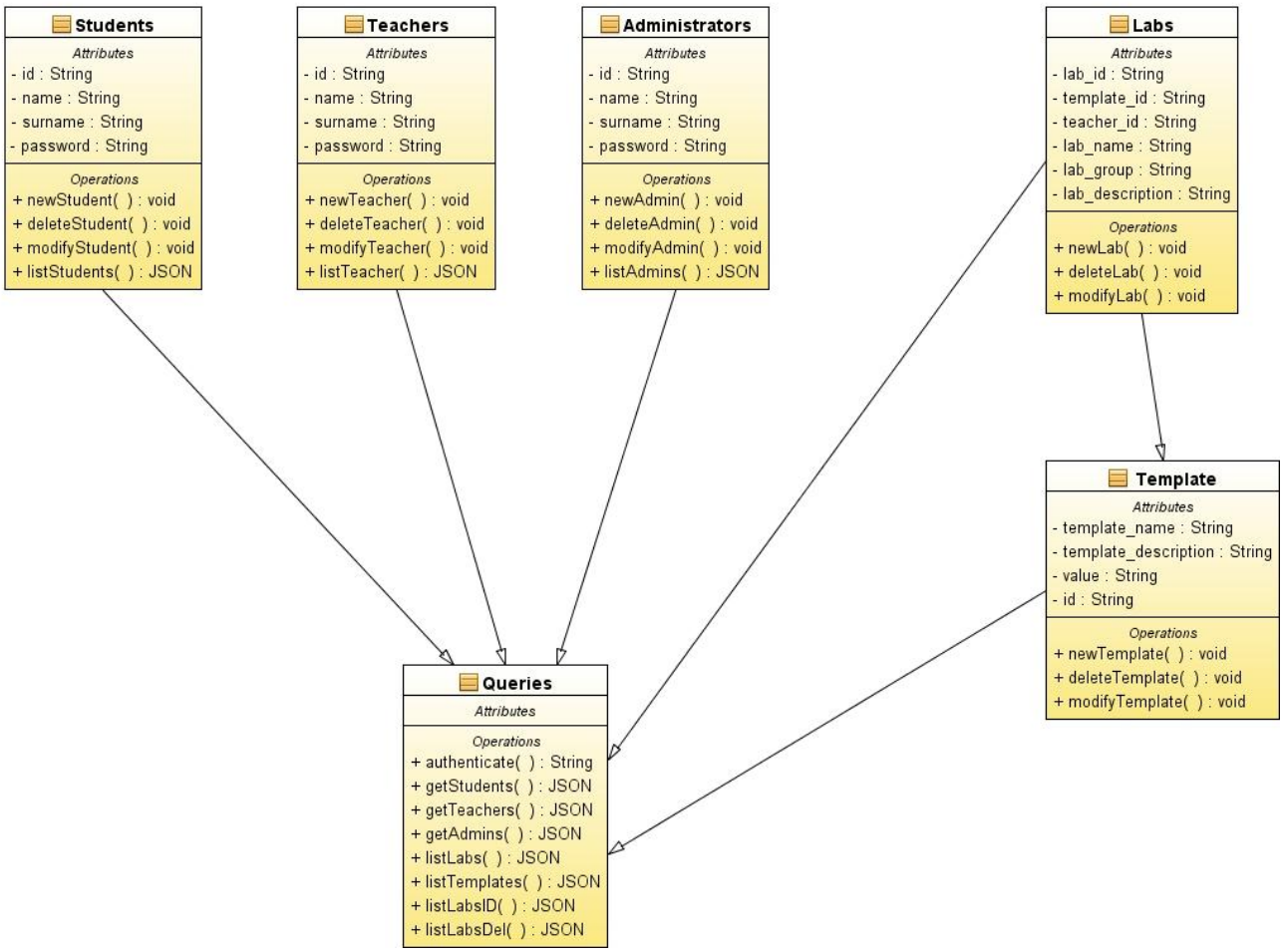
Y por último la página de la administración “Campus-Cloud Administrator”, donde el administrador tendrá todas las opciones correspondientes para manejar y controlar el funcionamiento de la aplicación. Opciones de list, create y delete referentes a cada uno de los bloques de estudiantes, laboratorios, plantillas y profesores con las que podrán listar, añadir y borrar estudiantes de la base de datos, listar, añadir y borrar profesores, al igual que listar, añadir y borrar plantillas y laboratorios.

7.2 Base de datos

Nuestra aplicación esta sustentada por una base de datos desarrollada en lenguaje Ruby con SQLite. La misma sirve para controlar y gestionar los diferentes usuarios del sistema (alumnos, profesores y administradores), así como los laboratorios.

Dicho archivo se llama dataBase.rb

A continuación mostramos un diagrama de clases de nuestra base de datos:



Pasamos a comentarlo brevemente:

Tenemos tres clases con los tres usuarios de nuestro sistema que tienen los mismos atributos (id, name, surname y password) y también las mismas funciones (new, delete, modify y list). También tenemos otra clase Labs que esta ligada a Template y a Teacher ya que tanto el identificador del Template como del Teacher son foreign key en Labs. A su vez, todas las clases están relacionadas con Queries, que son las diferentes consultas que tenemos que realizar para un correcto funcionamiento de nuestra aplicación.

Las funcionalidades de cada diferente tipo de usuario pasaremos a comentarlas más adelante cuando estudiemos los casos de uso del sistema.

En este punto cabe indicar que todos los cambios que se realicen dentro de la aplicación Web son estables dentro de la base de datos, ya que se guardan en un archivo llamado pruebaBD.

Destacar que la primera vez que la aplicación arrancó, lo hace con un administrador por defecto, ya que son los administrador los que pueden dar de alta profesores, alumnos…

Y también, es obligado mencionar que en nuestra base de datos usamos Sequel.

Sequel es un plugin de Ruby que actúa como software intermedio (middleware). Y nos ofrece la posibilidad de portar nuestra base de datos de SQLite a otro sistema gestor como podría ser MySQL o PostgreSQL, ya que Sequel nos abstrae de la base de datos final poniendo sus métodos a nuestra disposición.

Esto hace que sea bastante funcional y pueda ser ejecutada en gran cantidad de equipos independientemente del sistema gestor de bases de datos que use cada equipo en cuestión, ya que Sequel lo que hace es traducir el código de la base de datos desarrollada en código específico para cada sistema gestor de bases de datos.

Hablar de un sistema altamente portable como el que nos ofrece Sequel es una gran ventaja de nuestra aplicación.

7.3 Arquitectura de la aplicación

Nuestro sistema usa el patrón Modelo-Vista-Controlador, el cual se mencionó en el apartado 6.2.2 del presente documento.

7.3.1 Modelo

El modelo es aquello que soporta los datos (o capa de negocio). La representación específica de la información con la cual el sistema opera.

En nuestro caso el modelo está formado por la base de datos escrita en lenguaje Ruby, explicada en el punto anterior de esta memoria.
La cual se corresponde con el archivo dateBase.rb.

7.3.2 Vista

La vista presenta el modelo en un formato adecuado para interactuar.
Dentro de nuestra aplicación la vista está formada por los archivos CSS, los JavaScript y los HTML. A saber:

7.3.2.1 CSS

Archivos Index.css y Login.css

Son las hojas de estilo de las pantallas de index y login respectivamente, esto es, los elementos propios de la aplicación. Además de estas, hay otras hojas de estilo por defecto para los elementos de jQuery y DataTable.

7.3.2.2 JavaScript

Archivos indexAdmin.js, indexTeacher.js, indexStudent.js y login.js

Son archivos de funciones JavaScript de las que se hace uso. Son homólogas a las de las hojas de estilo, salvo que hemos diferenciado en función del tipo de usuario.
En este caso, también hay archivos JavaScript por defecto que marcan el comportamiento de los elementos de jQuery y DataTable.

7.3.2.3 HTML

Archivos indexAdmin.html, indexTeacher.html, indexStudent.html y login.html
Son archivos HTML que utiliza nuestra aplicación y que se "visten" con las hojas de estilo y las funciones JavaScript comentadas antes.

De nuevo, son análogos a los mencionados previamente, diferenciamos en virtud del tipo de usuario que usa la aplicación.

7.3.3 Controlador

El controlador responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista.

En nuestro caso corresponde al archivo campus-cloud-server.rb

Este es el archivo principal, donde comienza la ejecución de la aplicación. Levanta el servidor de Sinatra y captura los GET y POST que llegan del cliente redirigiendo esas acciones a campus-cloud-server.rb. Configura el puerto y el host al que se conectará el cliente, pudiendo estar esta configuración implícita en el archivo o indicándolo como opciones del comando al ser ejecutado.

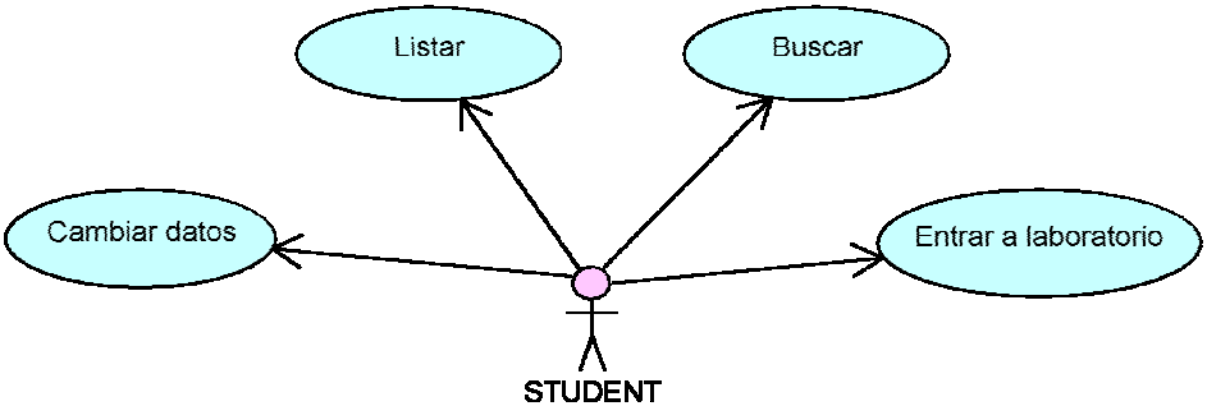
7.4 Casos de uso del sistema

Los diferentes casos de uso de nuestra aplicación los mostraremos y comentaremos con la ayuda de unos diagramas de casos de uso en función del usuario que se encuentre en el sistema y con las pertinentes capturas de la aplicación.

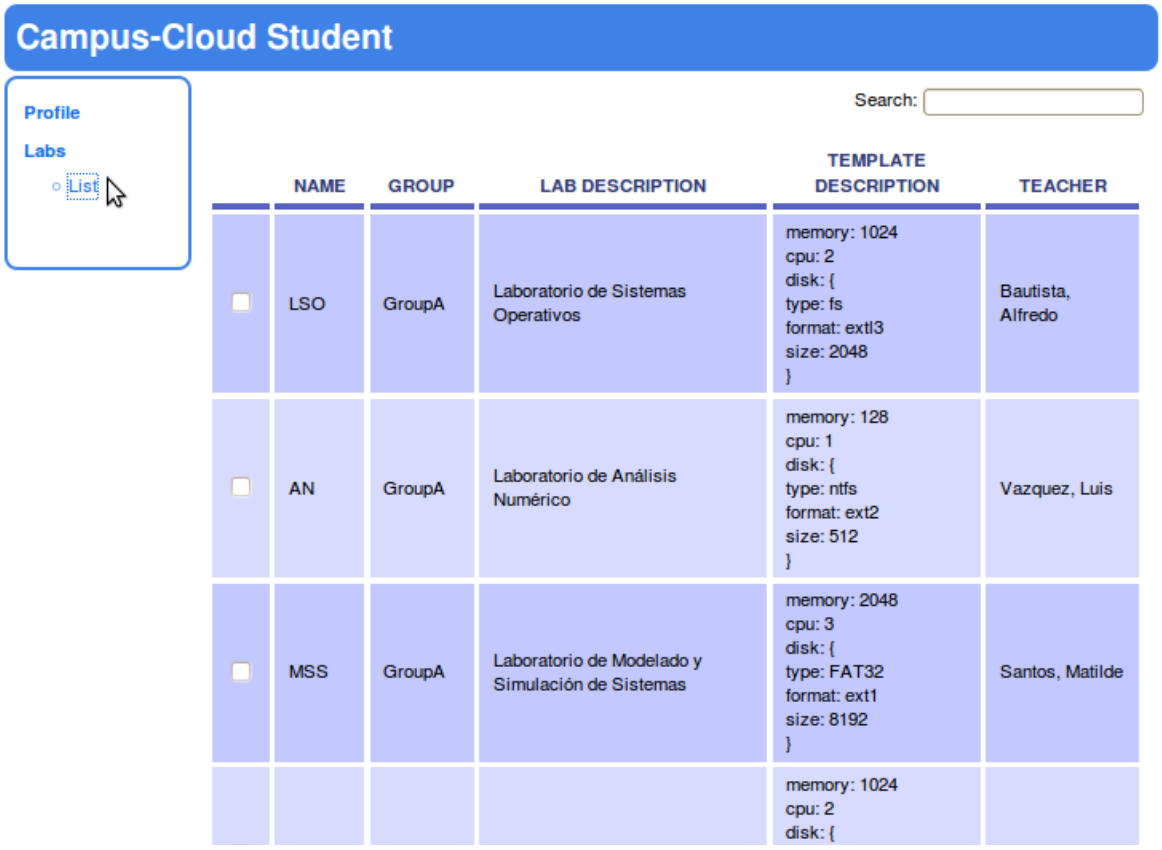
Un diagrama de casos de uso es una especie de diagrama de comportamiento. Define una notación gráfica para representar casos de uso llamada modelo de casos de uso. Esta notación gráfica define la naturaleza de un caso de uso en concreto.

Mencionar que, todas las acciones que se van a explicar deben estar en ejecución para que se puedan llevar a cabo.

7.4.1 Diagrama de casos de uso del estudiante



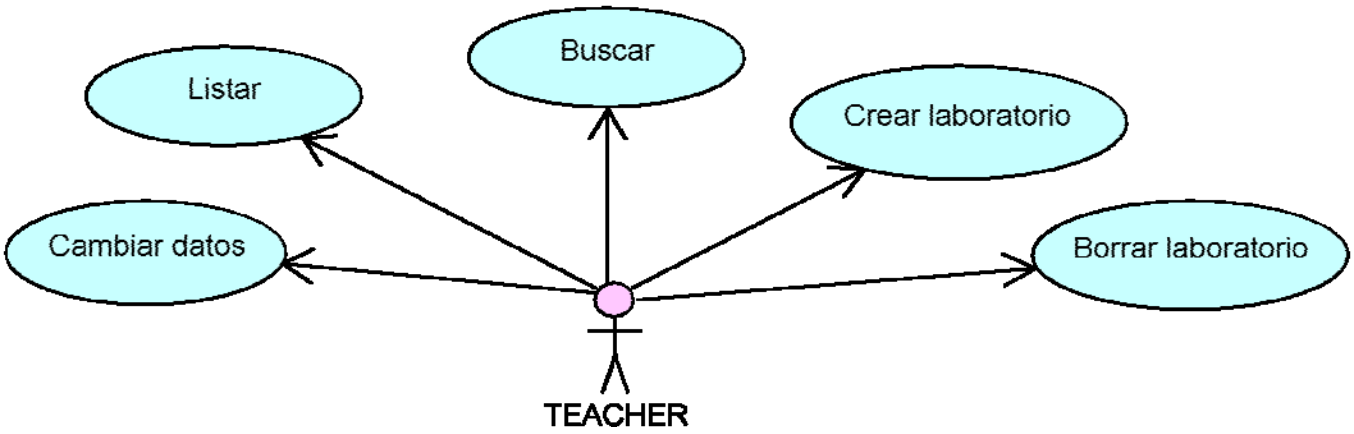
Listar: "List" permite a todos los estudiantes consultar o listar todos los laboratorios almacenados en la base de datos o los laboratorios creados por ellos si así lo hiciesen. Por ello, una vez que seleccionamos la opción de "List", se hará una petición a nuestro servidor, para obtener así los datos referentes a los laboratorios, obtenidos en formato JSON que ayuda de otra función (mostrarJSON()) transformaremos dichos datos en una tabla cuyos campos serán el nombre del laboratorio, grupo, descripción del laboratorio, descripción de la plantilla y nombre y apellidos del profesor. En el caso de que no haya ningún laboratorio almacenado en la base de datos, la consulta que hacemos a la misma, nos devolverá un mensaje del tipo "The are no data to show" avisándonos de ello.



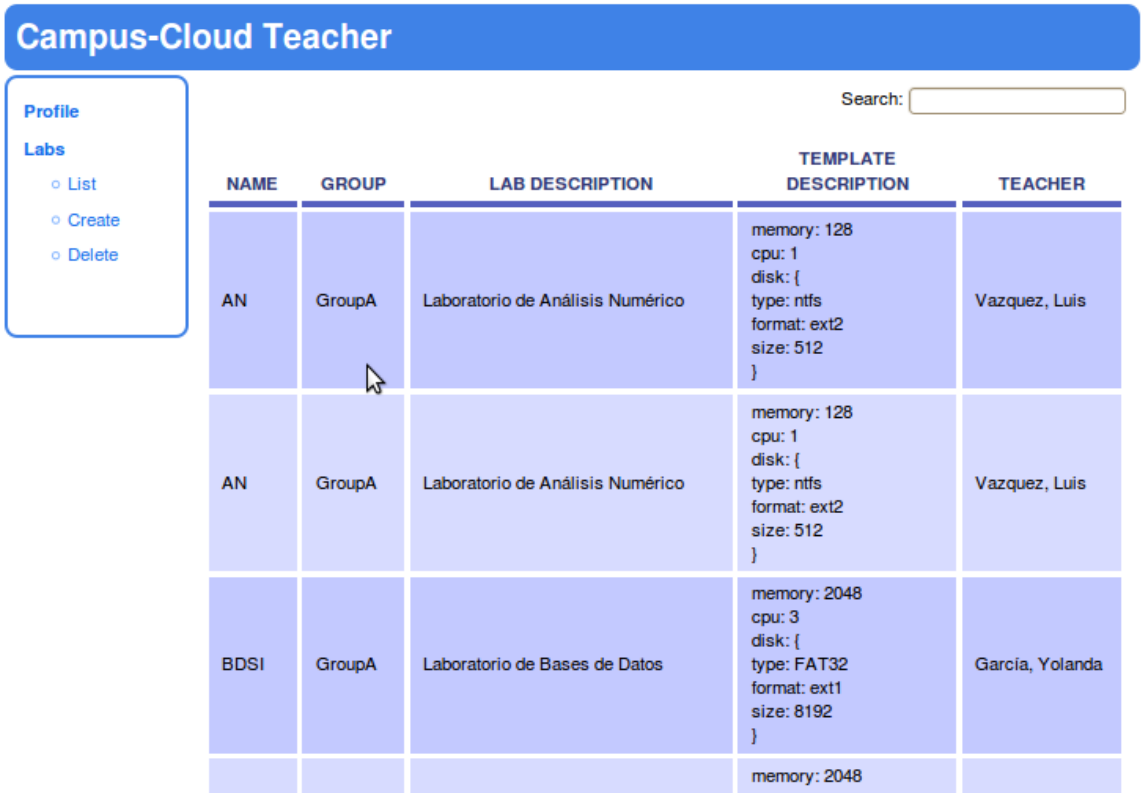
Buscar: "Search" es un cuadro de texto que nos permite hacer búsquedas de forma rápida y sencilla, simplemente con rellenar el cuadro que nos aparece en la parte superior derecha de la parte central de la página, con cualquiera de los datos referentes a las columnas de la tabla que tengamos en ese momento.

Entrar a laboratorio: Una máquina virtual es un software que emula a una computadora y puede ejecutar programas como si fuese una computadora real. Dicho esto, podemos decir, que la función del botón "Enter Lab" es la de crear una máquina virtual con las características descritas anteriormente, para que cada estudiante pueda pasar a utilizarla como si fuera real y la suya propia.

7.4.2 Diagrama de casos de uso del profesor



Listar: "List" permite a todos los profesores consultar o listar todos los laboratorios almacenados en la base de datos o los laboratorios creados por ellos si así lo hiciesen. Por ello, una vez que seleccionamos la opción de "List", se hará una petición a nuestro servidor, para obtener así los datos referentes a los laboratorios, obtenidos en formato JSON que ayuda de otra función (mostrarJSON()) transformaremos dichos datos en una tabla cuyos campos serán el nombre del laboratorio, grupo, descripción del laboratorio, descripción de la plantilla y nombre y apellidos del profesor. En el caso de que no haya ningún laboratorio almacenado en la base de datos, la consulta que hacemos a la misma, nos devolverá un mensaje del tipo "The are no data to show" avisándonos de ello.

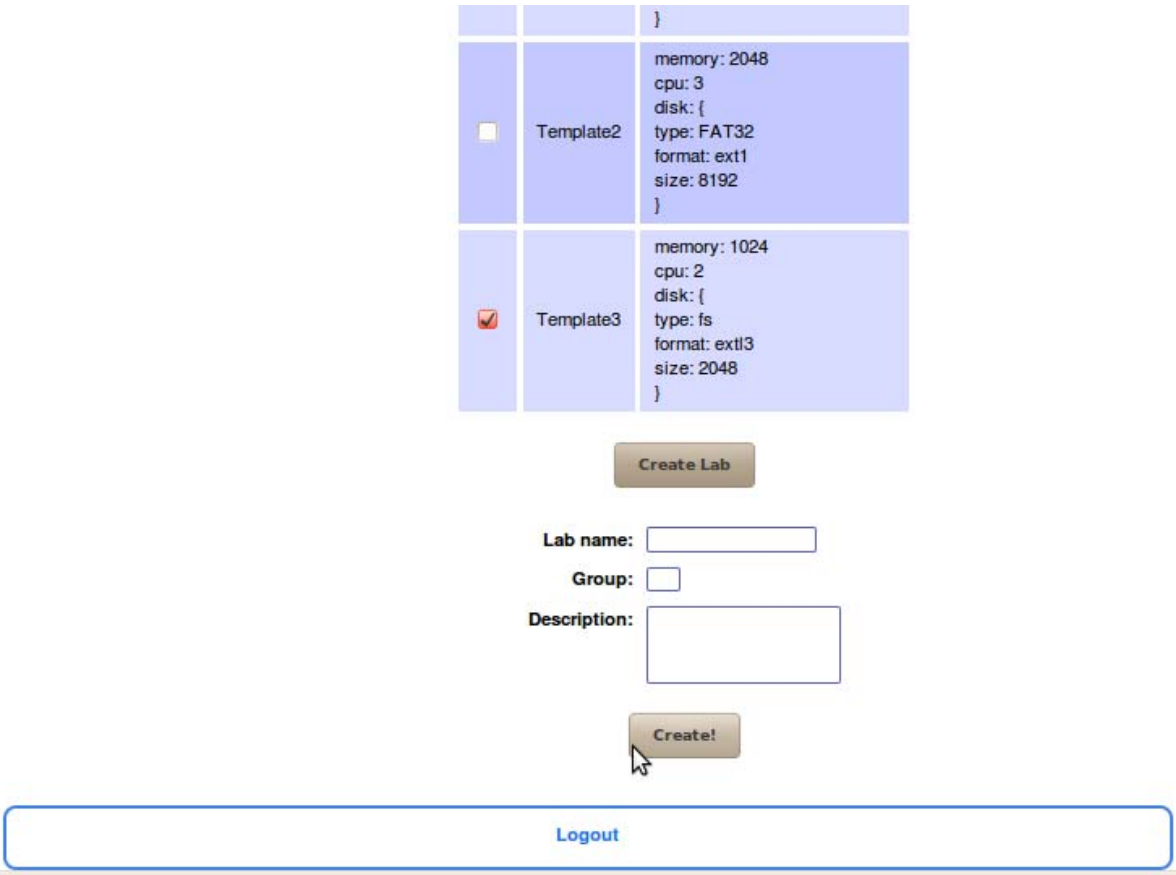


Buscar: "Search" es un cuadro de texto que nos permite hacer búsquedas de forma rápida y sencilla, simplemente con rellenar el cuadro que nos aparece en la parte superior derecha de la parte central de la página, con cualquiera de los datos referentes a las columnas de la tabla que tengamos en ese momento.

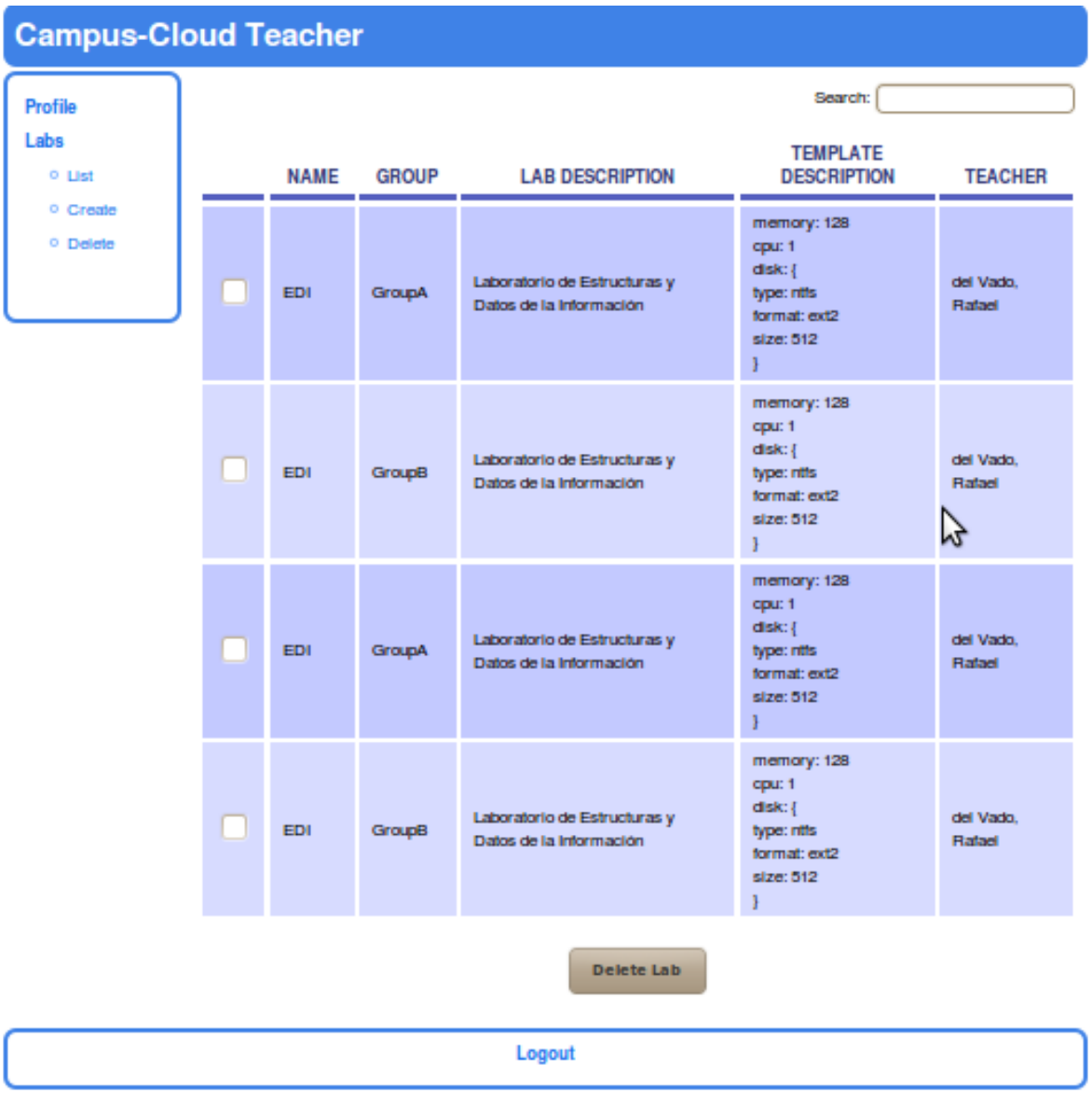
Crear Laboratorio: "Create" permite crear un nuevo laboratorio, a partir de las plantillas almacenadas en la base de datos. Para ello primero debemos seleccionar una de las plantillas que nos aparecen (siempre y cuando haya alguna almacenada en la base de datos), ya que si pulsamos el botón "Create Lab" directamente sin seleccionar ninguna de ellas, nos aparecerá un mensaje de error del tipo "Please, check one template", avisándonos que debemos seleccionar una de ellas antes de

activar el botón. Una vez elegida la plantilla y activado el botón de crear laboratorio, procedemos a la creación.

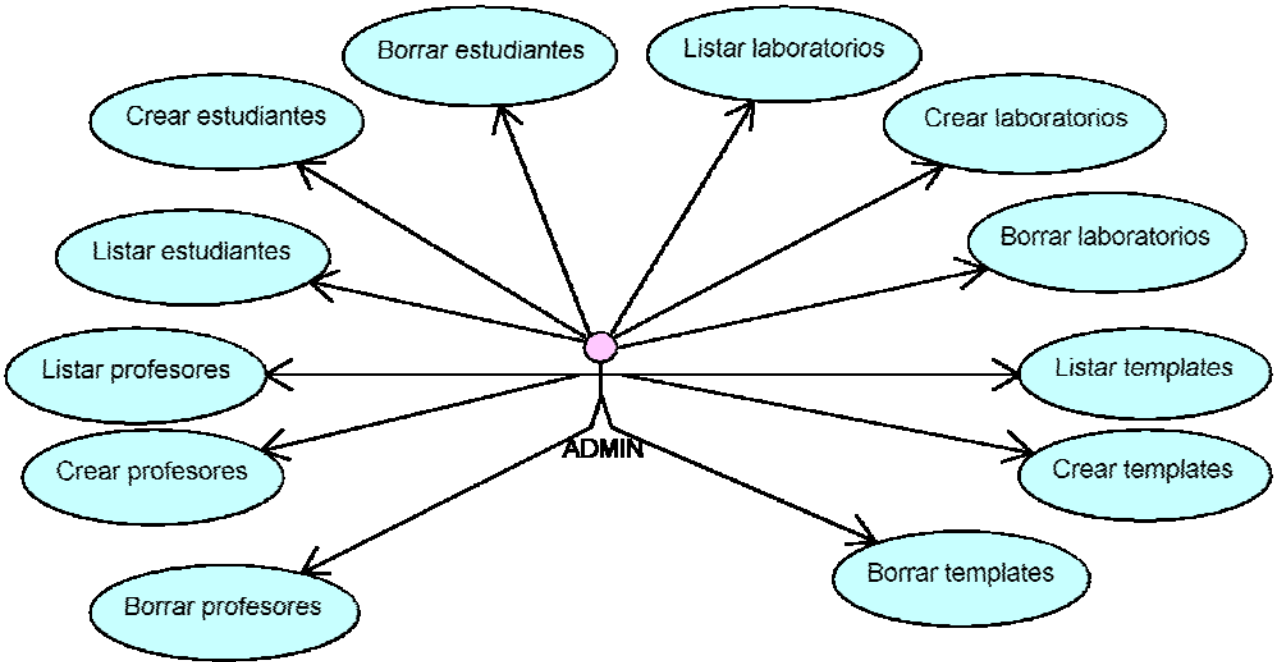
Para crear un nuevo laboratorio, ahora tendremos que rellenar los siguientes cuadros de texto que nos aparecen: nombre del laboratorio, que como mínimo debe estar compuesto por dos letras, sino se mostrará un mensaje de error del tipo “Lab name length must be 2 letters at least", el grupo, que debe estar compuesto por una letra, sino se mostrará un mensaje de error del tipo “Lab group must be 1 letter” y la descripción compuesta también por letras y que debemos de rellenar porque de lo contrario, se nos avisará de ello a través de un mensaje del tipo “Lab description is empty. Please, enter a description for this lab”. Se nos informará también, a través de un mensaje de tipo “The lab has been created successfully”, la correcta creación de un laboratorio.



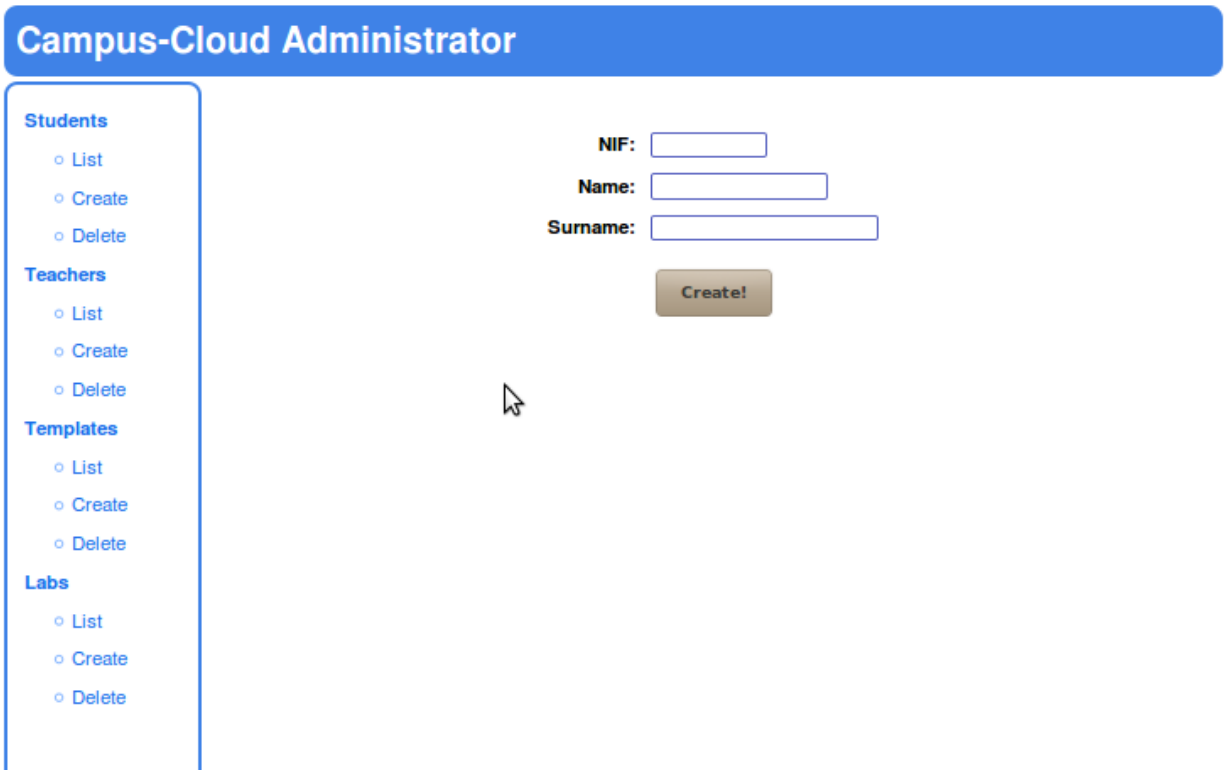
Borrar Laboratorio: "Delete", gracias a esta opción, podemos borrar los laboratorios que hayan sido creados anteriormente o que estuvieran almacenados en la base de datos. Para ello debemos seleccionar con el checkbox la fila correspondiente a borrar, de la tabla de laboratorios y con solo pulsar el botón delete éste se borrará. En el caso de no seleccionar ninguno de ellos se mostrará un mensaje informativo del tipo "Please, check one lab for delete" avisándonos del correcto procedimiento que conlleva eliminar un laboratorio.



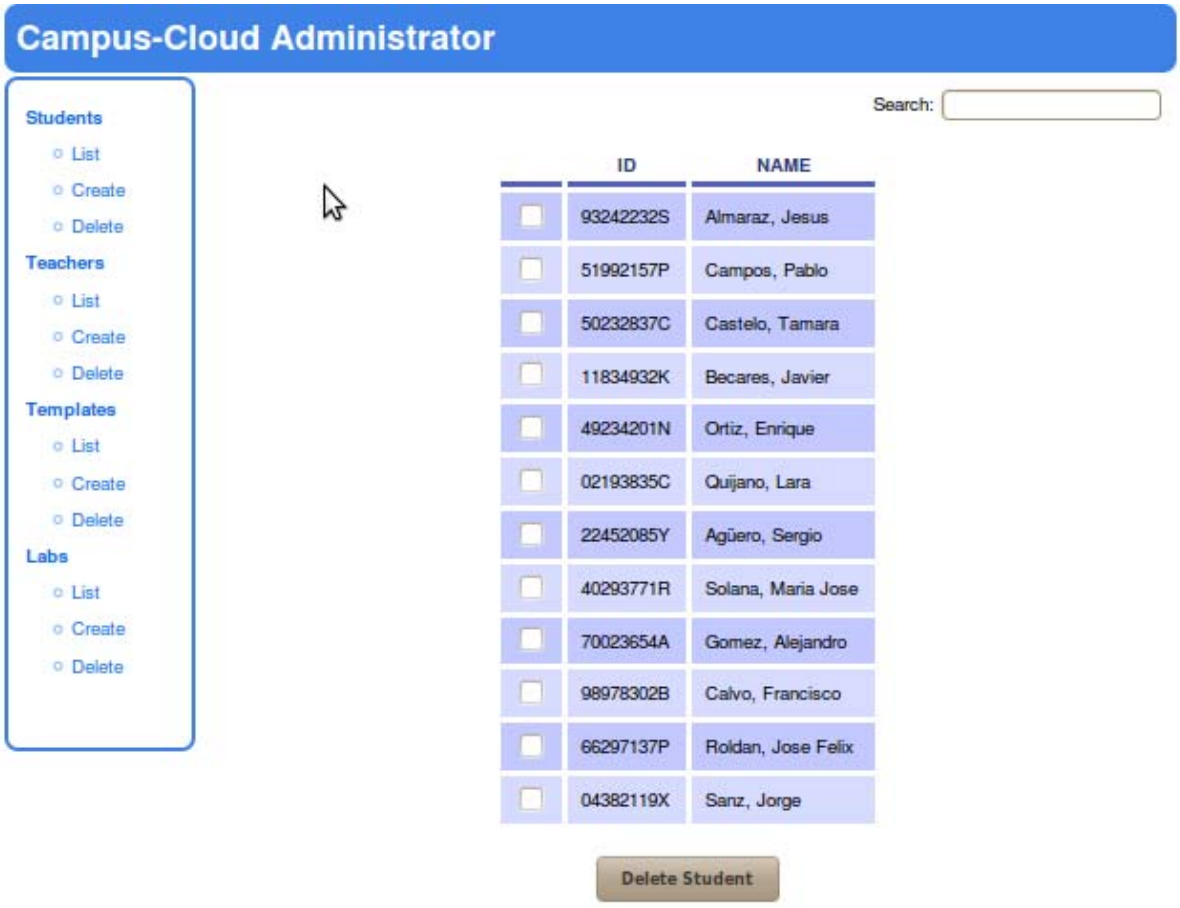
7.4.3 Diagramas de casos de uso del administrador



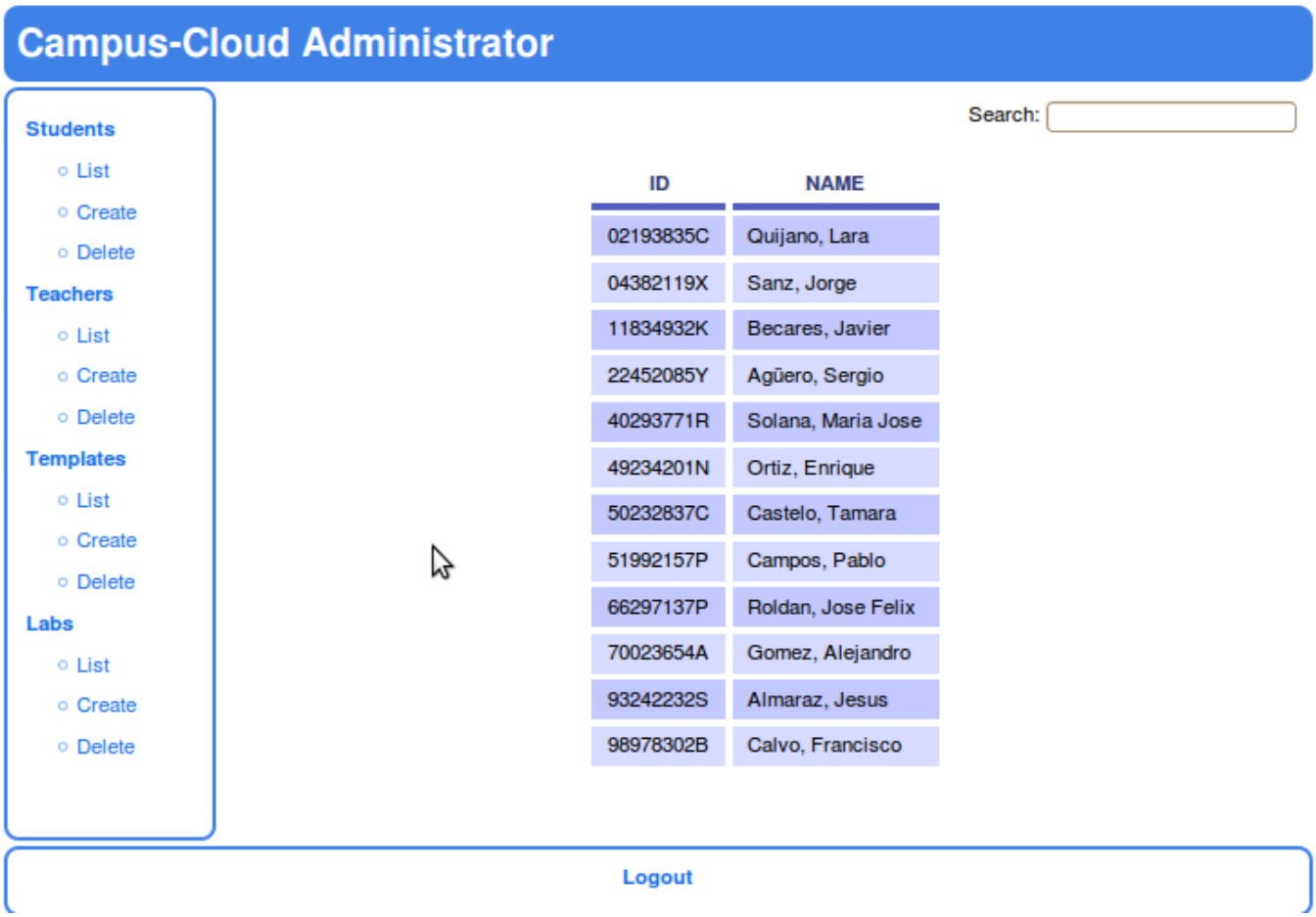
Crear estudiante: "Create" permite crear o insertar un nuevo estudiante en la base de datos. Para ello debemos rellenar los cuadros de texto que nos aparecen en la página. El NIF lo rellenaremos con el dni del estudiante que estará formado por ocho números y una letra, regla a seguir, porque de lo contrario el algoritmo que comprueba la validez del DNI nos devolverá un error del tipo "Invalid NIF". Por último rellenar los cuadros de name y surname con el nombre y apellidos del estudiante. Éstos últimos no podrán dejarse vacíos, pues en ese caso se mostraría un mensaje de error del tipo "Name or Surname is empty".



Borrar estudiante:"Delete" permite borrar estudiantes de la base de datos, simplemente con seleccionar el checkbox de la fila correspondiente y activando el botón de "Delete Student".

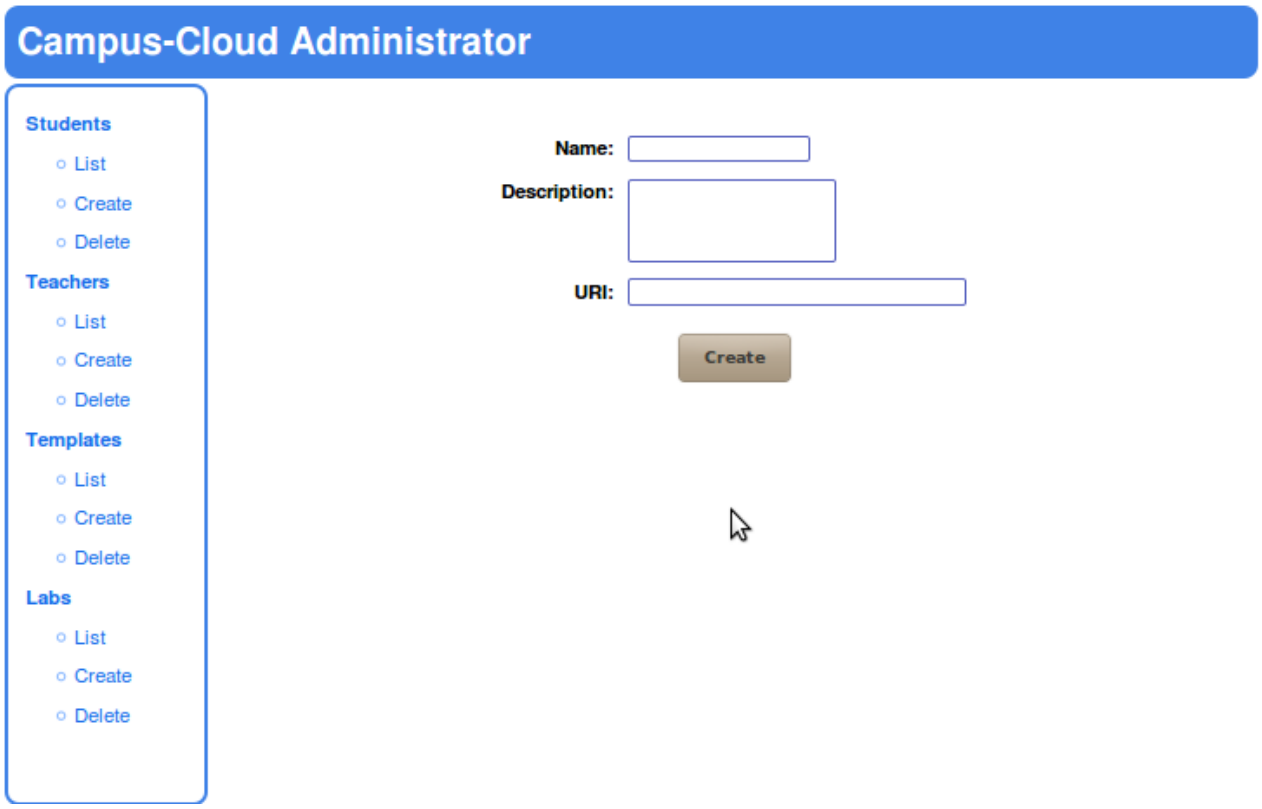


Listar estudiante: "List" lista todos los estudiantes de la base de datos. Se mostrarán en formato tabla con los campos id y name.



Crear plantilla: "Create" permite crear o insertar una nueva plantilla en la base de datos. Para su correcta inserción, debemos rellenar los cuadros de texto que nos aparecen en pantalla. El nombre de la plantilla y la descripción que no se podrán dejar vacíos pues nos aparecería un mensaje del tipo "Name is empty!. Please introduce a template name" o "Description is empty!. Please introduce a template

description" y el campo URI que se corresponderá con una cadena corta de caracteres que identifica inequívocamente un recurso.



Crear laboratorio: "Create" permite crear o insertar un nuevo laboratorio en la base de datos. Para ello debemos seleccionar una plantilla a seguir, pues si no lo hacemos nos aparecerá un mensaje de error del tipo "Please, check one template" y rellenar los cuadros de texto que aparecen en la página. Los cuadros de texto a rellenar son el NIF del profesor, que deberá contener ocho números y una letra ya que sino el algoritmo que comprueba la validez del dni nos mostrará un mensaje de error del tipo "An error ocurred when the server tried to create the Lab.Review Teacher NIF". El nombre del laboratorio es otro campo a rellenar, compuesto al menos por dos letras, porque de lo contrario aparecerá un mensaje de error del tipo "Lab name length must be 2 letters at least" además del grupo, que deberá estar compuesto únicamente por una letra, sino se mostrará un mensaje de error del tipo

"Lab group must be 1 letter". El último campo a rellenar será la descripción del laboratorio, que no se podrá dejar vacío ya que nos aparecería un mensaje de error del tipo "Lab description is empty. Please, enter a description for this lab".

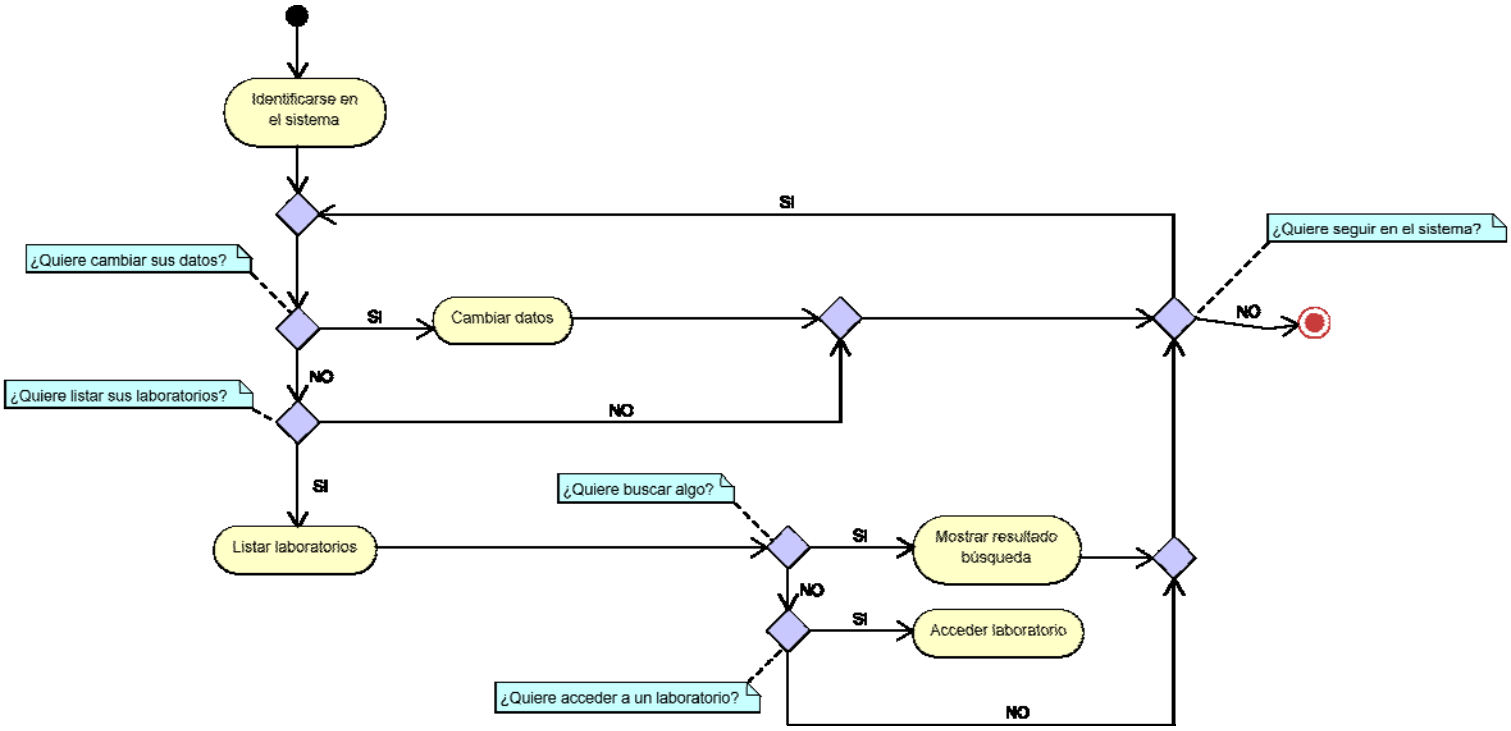
Como se ve en el diagrama, existen más casos de uso, pero no los enumeramos por sencillez de este documento y fundamentalmente porque son análogos a los ya expuestos.

7.5 Diagramas de actividades

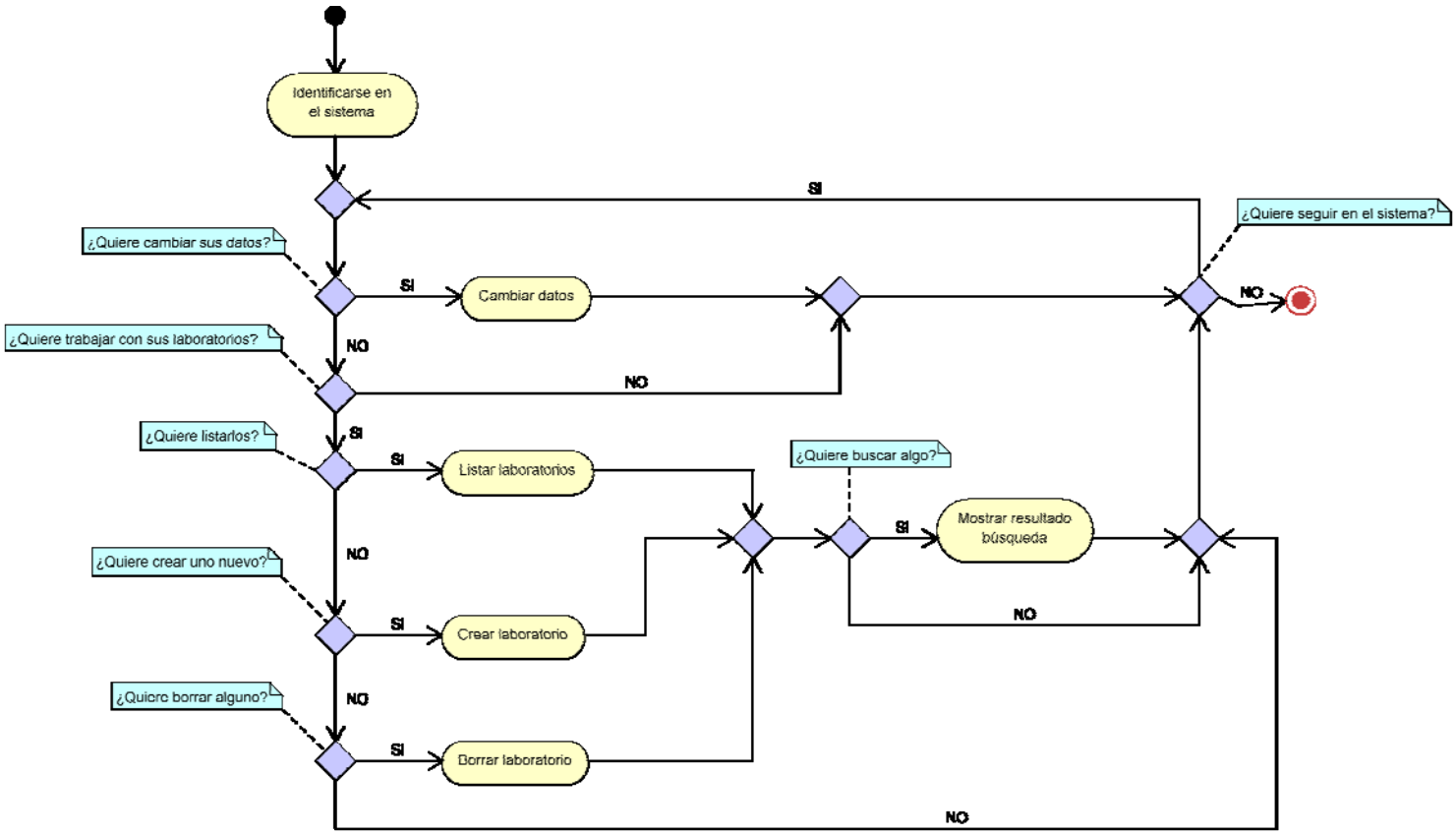
Los siguientes diagramas de actividades permiten observar, a grandes rasgos, la forma que tendrán nuestros tres diferentes tipos de usuarios de interactuar con el sistema, y cómo este reacciona internamente a esas acciones en los principales casos de uso.

Al igual que ocurría en los casos de uso, las actividades aquí mostradas deben de estar en ejecución para que se puedan llevar a cabo.

7.5.1 Diagramas de actividades del estudiante



7.5.2 Diagramas de actividades del profesor

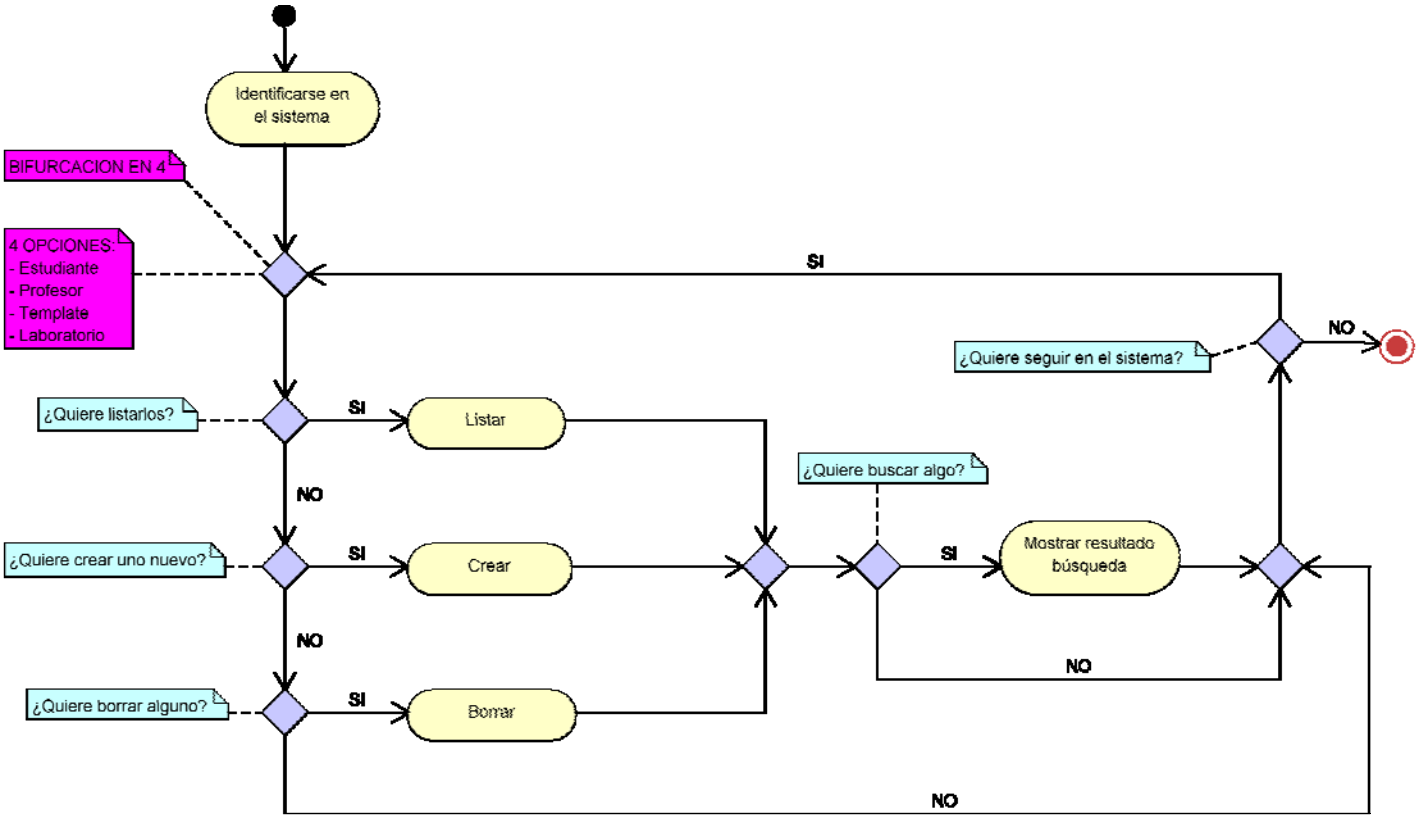


7.5.3 Diagramas de actividades del administrador

En este caso, debido a que el administrador puede trabajar con 4 elementos diferentes, que son: estudiantes, profesores, templates y laboratorios, simplemente hemos diseñado un único diagrama, ya que con los cuatro elementos puede realizar las mismas acciones (listar, crear, borrar y buscar). De esta forma, el administrador, al identificarse y entrar en el sistema, deberá elegir con cual de las cuatro posibilidades diferentes de trabajo va a operar.

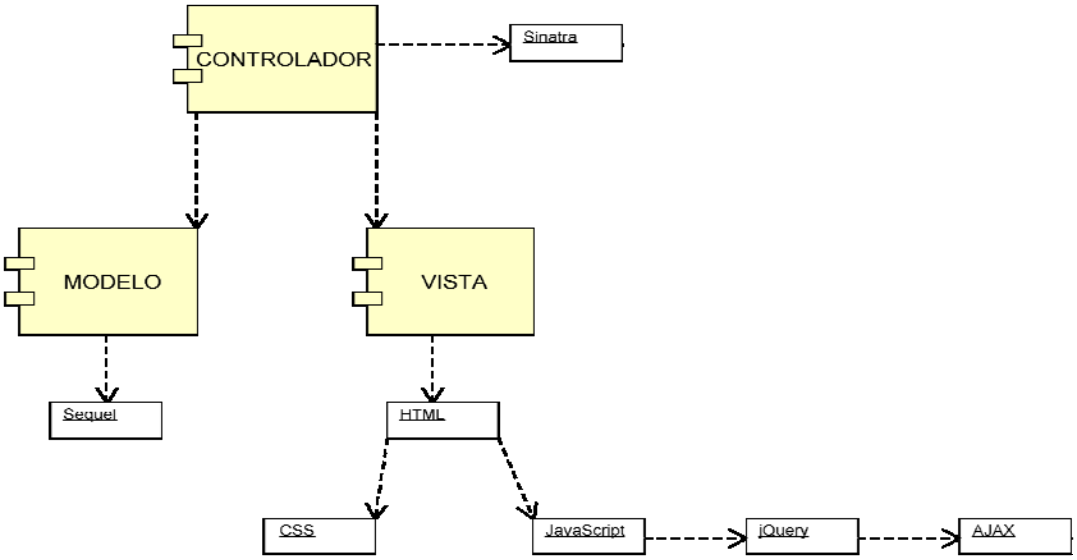
Pero el diagrama de actividades será igual independientemente del elemento con el que se va a trabajar.

La bifurcación en cuatro se muestra en el diagrama con el comentario de color morado.



7.6 Diagrama de componentes:

La siguiente imagen muestra las diferentes tecnologías empleadas, y el papel que juega cada una dentro de la aplicación.



Como se puede ver, el elemento Controlador, hace uso de Sinatra. El Modelo utiliza Sequel para la base de datos de la aplicación mientras que Vista crea HTML de las páginas, los cuales se ayudan de CSS y JavaScript a fin de resultar mas usables y atractivas de cara al usuario.

7.7 Interfaz gráfica

La interfaz gráfica de nuestra aplicación Web ha sido diseñada lo más simple y sencillamente posible de cara a posibles usuarios no demasiado familiarizados con este tipo de interfaces, pero todo ellos sin olvidar el aspecto estético de la misma.

7.7.1 Diseño

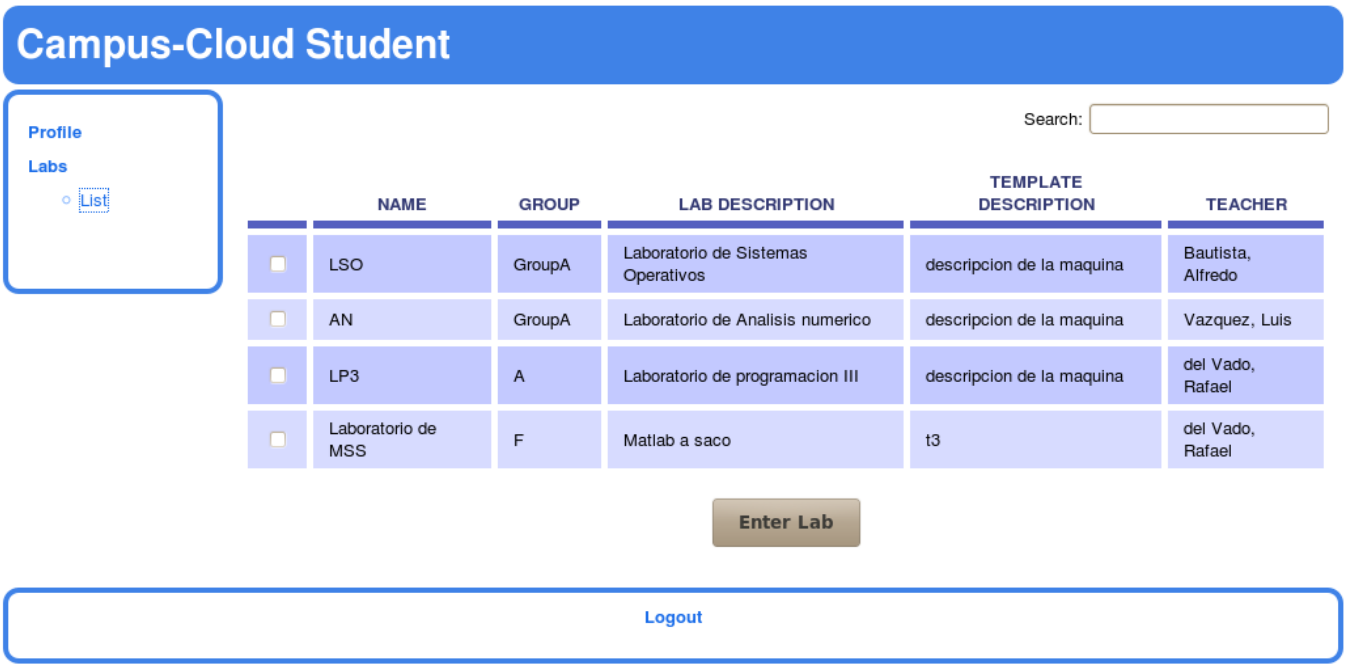
Veamos brevemente el diseño de nuestra aplicación.

7.7.1.1 Diseño Login

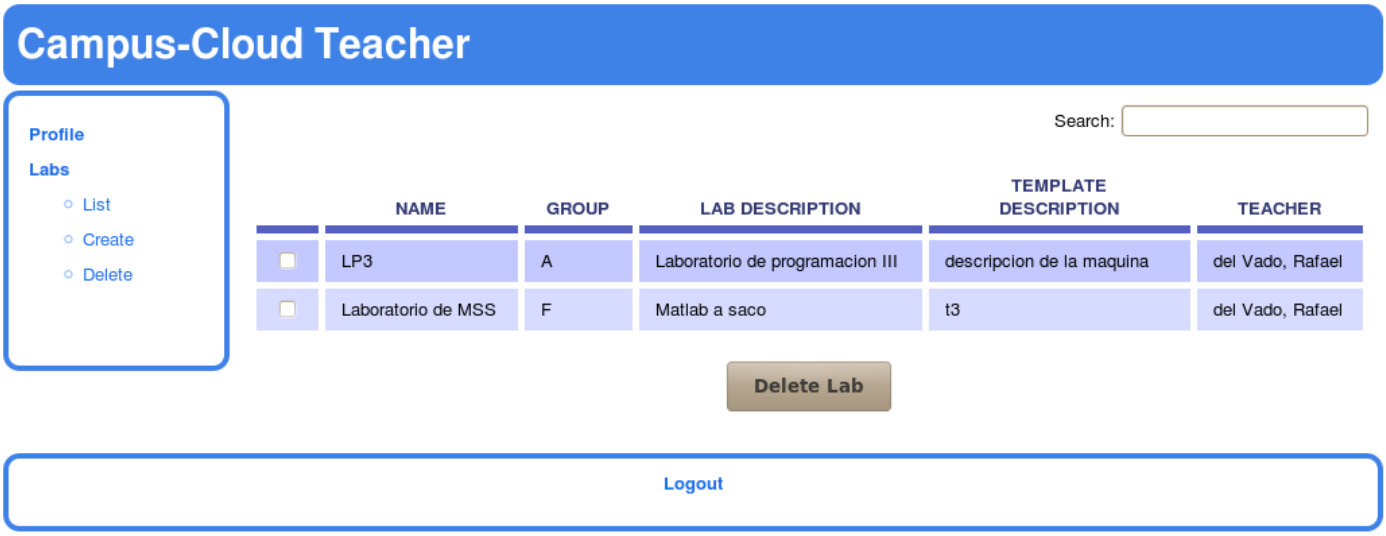
Simplemente volver a recordar que disponemos de un checkbox que el usuario puede marcar si desea que la aplicación le recuerde.

7.7.1.2 Diseño Index

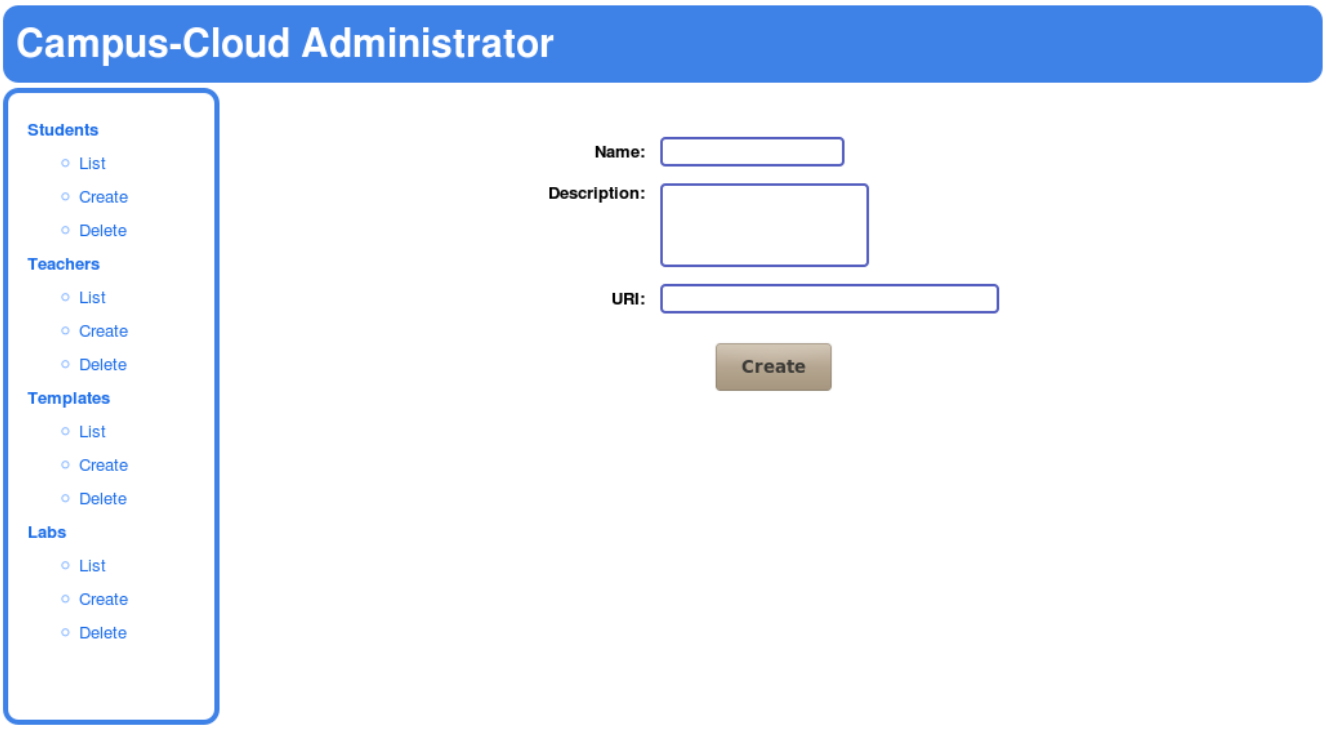
- **Student**



- **Teacher**



- **Administrator**



Podemos observar que el diseño es prácticamente igual para todos los usuarios de la aplicación, diferenciándose únicamente en las funciones que puede ejecutar cada uno de ellos una vez dentro del sistema.
A su vez, se puede comprobar que las páginas constan de cuatro partes bastante diferenciadas, a saber:

- Cabecera o header:

  En la cual recordamos el nombre de la aplicación, "Campus Cloud" e identificamos qué tipo de usuario está dentro del sistema.

- Barra lateral o sidebar:

  En ella mostramos las diferentes opciones de trabajo de las que dispone cada tipo de usuario de la aplicación.

- Principal o main:

  Es el área de trabajo en sí, donde se pueden realizar las actividades de las que disponga cada usuario. Por así decirlo, es donde se produce la interacción aplicación-usuario.

- Pie de página o footer:

  En el cual simplemente disponemos de un botón de "Logout" para abandonar el sistema.

7.8 Sinatra

Todo el desarrollo de la aplicación y su posterior ejecución se realiza en este framework. Aprovechando su funcionalidad estrechamente relacionada con REST, la aplicación esta completamente enmarcada en Sinatra.
Un proyecto Sinatra ha de seguir una serie de convenciones.

Estructura básica del proyecto:

Al ser una aplicación modular, esto es, gestionar todo a través de un solo archivo ejecutable, lanzamos Sinatra a través de un archivo llamado config.ru, que simplemente lo que hace es levantar el servidor Sinatra y llamar a nuestra aplicación, contenida en el fichero principal del servidor, campus-cloud-server.rb, que constituye el servidor Sinatra de la aplicación.
Y que contiene:

- Las sentencias de establecimiento del puerto y host desde los que es accesible el servidor Sinatra.

- Contiene los servicios Web del servidor entre los que destaca:

    - get '/' do: Es la función que se ejecuta al acceder a la aplicación con la URL inicial y la que carga la plantilla que es el HTML de la aplicación.

    - Carpeta templates**:** Esta carpeta constituye la ubicación obligatoria del html que publica el servidor.

    - Carpeta public: Esta carpeta contiene a su vez las siguientes carpetas:

        - **CSS**: Contiene las hojas de estilo CSS.

        - **JavaScript:** Contiene los ficheros JavaScript con las funciones jQuery, JavaScript y AJAX.

    - Resto de archivos necesarios: El resto de archivos necesarios para implementar la aplicación se han encapsulado según su funcionalidad en la implementación del patrón modelo-vista-controlador de la aplicación.

7.9 jQuery

jQuery ofrece un amplio abanico de funcionalidades, desde componentes visuales para mostrar información de una manera más intuitiva y atractiva para el usuario, hasta funciones dirigidas a los desarrolladores para facilitar el uso a llamadas que de otra manera serían más complejas.
La funcionalidad AJAX se hizo en un principio usando directamente las sentencias que nos ofrece por si mismas, pero después se optó por usar a tal efecto el módulo de AJAX que jQuery tiene incorporado, que hace muy fácil su utilización.

Se invoca del siguiente modo:

```
$(document).ready(function(){
$.ajax({ });
```

});

La sintaxis de jQuery es muy sencilla. Los comandos se reconocen por comenzar con el símbolo $, de modo que una sentencia se escribiría de la siguiente forma:

$(elemento).evento(función);

Así, un ejemplo de sentencia que recorra una estructura "data" objeto por objeto sería de la siguiente forma:

$.each(data, function(key, val)

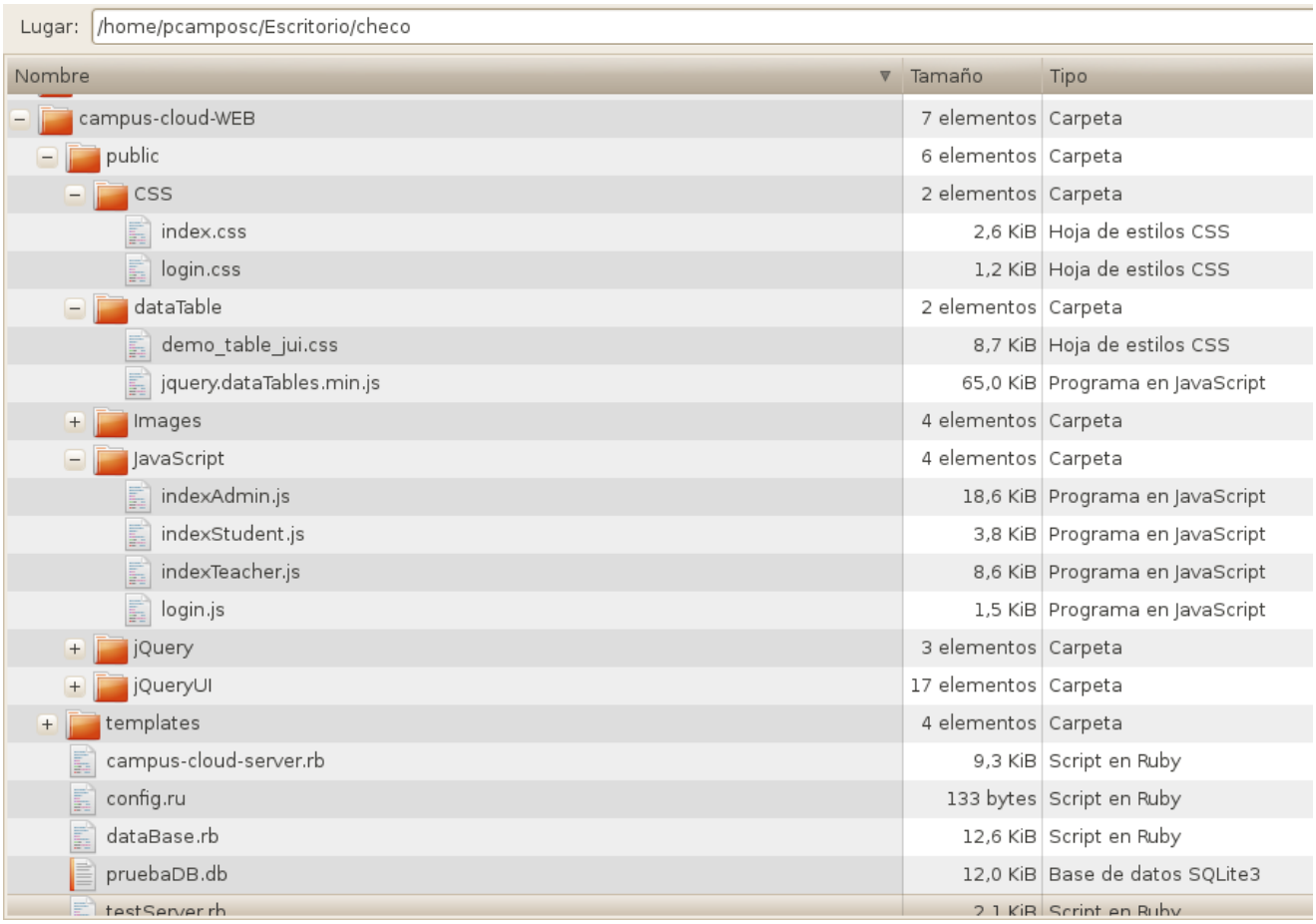
Otro ejemplo:
Añadimos (concatenamos) el elemento "aux" a nuestro contenedor central "container" de la aplicación Web.

$('#container').append(aux);

Además, jQuery dispone de un framework de temas, el llamado Theme Roller, que permite descargar uno de los temas por defecto y la posibilidad de crear un tema propio para CSS.

7.10 Jerarquía de directorios

La jerarquía de directorios de nuestra aplicación Web es la siguiente:



8. Conclusiones y trabajo futuro

8.1 Conclusiones

La asignatura de Sistemas Informáticos está orientada a crear un proyecto de una envergadura mayor a lo que se está acostumbrado en el resto de las asignaturas de Ingeniería en Informática, poniendo en práctica todos los conocimientos y destrezas adquiridos a lo largo de las mismas.

La principal dificultad de este proyecto en particular, radica en el aprendizaje de un lenguaje de programación nuevo para el equipo de desarrollo, como era Ruby, así como la programación Web, un aspecto que no se encuentra entre las asignaturas obligatorias y troncales de estos estudios y que muchos alumnos desconocen al finalizarlos. Al igual ocurre con el mantenimiento de la base de datos con Sequel.

También destacar que el uso de repositorios de control de versiones nos permitieron a cada componente del grupo trabajar con una misma copia del código de forma paralela, sin necesidad de tener que encontrarnos todos en un mismo lugar.

Por tanto, la parte más laboriosa del proyecto ha sido, probablemente, la documentación y aprendizaje de estas tecnologías, que ocuparon la mayor parte del tiempo. De hecho, la primera fase del proyecto se redujo a la búsqueda de las herramientas oportunas y su correcta utilización, para ello se realizaron pequeños tutoriales y notas explicativas, con el fin de facilitar el acceso a estas tecnologías por el resto de miembros. Esta fase que fue realizada durante los dos o tres primeros meses creó las bases instructivas para el desarrollo del proyecto.

8.2. Metodología de trabajo

Se pueden diferenciar tres grandes iteraciones del proyecto:

1. **Primera iteración**: Aprendizaje.

i. Aprendizaje y documentación de las nuevas tecnologías: Sequel, JavaScript, jQuery, AJAX, CSS, lenguaje Ruby…Se utiliza Sinatra como servidor de nuestra aplicación por su sencillez.

ii. Realización de diversos tutoriales que incorporaron todas las tecnologías implicadas en el proyecto.

2. **Segunda iteración:** Inicio del desarrollo de la aplicación.

   i. Desarrollo de la base de datos que sustenta nuestra aplicación, y sus correspondientes bancos de pruebas.

   ii. Creación del esqueleto inicial de la aplicación Web y diseño de la interfaz de usuario, desarrollo del HTML, estilos CSS…

3. **Tercera iteración:** Desarrollo de la aplicación.

   i. Incorporación de una página destinada a la autenticación de usuarios previa al uso de la aplicación. También se añade la gestión y manejo de las correspondientes cookies.

   ii. Comunicación de la interfaz con la base de datos.

   iii. Desarrollo de mejoras en la interfaz.

8.3 Trabajo futuro

En este apartado mostramos unas (las que consideramos más importantes) posibles líneas de trabajo futuro para los potenciales interesados, o incluso nosotros mismos, en continuar desarrollando esta aplicación.

8.3.1 Desarrollo de los entornos virtuales

Como se ha mencionado en la introducción, la principal mejora que se podría añadir a la aplicación es el desarrollo de los entornos virtuales mediante el uso del Cloud Computing, y así ahorrarse la instalación de tanto software en los ordenadores de los laboratorios a cambio de obtener máquinas virtuales del software que se necesitara en función de las asignaturas en las que cada alumno está matriculado o que imparte cada profesor.

Se trata de una mejora tan importante como costosa en tiempo, con lo cual proponemos esta mejora para posibles futuros alumnos de la asignatura de Sistemas Informáticos que deseen continuar con este proyecto.

8.3.2 Control de errores

Una de las mejoras que proponemos para nuestra aplicación es ampliar el control de los posibles errores, que no se ha tratado todo lo que debiera por falta de tiempo. La aplicación controla los principales errores que pudieran surgir, pero evidentemente todo es mejorable.

8.3.3 Interfaz más interactiva

Actualmente la aplicación es sencilla y funcional, pero quizás no demasiado interactiva ni atractiva de cara a un potencial usuario. Existen una serie de

posibilidades, que de implementarse, otorgaría un aire más moderno y más intuitivo a la interfaz, tal y como estamos acostumbrados cuando navegamos por Internet.
Si bien es cierto que una aplicación de este tipo no requiere en primera estancia un diseño más profesional, hay que añadir que hoy en día existen profesionales del diseño Web que podrían hacer un entorno aún más atractivo y práctico para el usuario.

8.3.4 Diseñar el apartado “profile” de la interfaz

La opción “profile” que aparece tanto en la interfaz del estudiante como en la del profesor no está diseñada, se disponen de los métodos necesarios para hacerla funcionar, como pudiera ser listar el nombre, apellidos… de un usuario dado y que por ejemplo, cambie su password de acceso al sistema. Simplemente habría que añadir esa funcionalidad a la interfaz.

9. Glosario

- **JSON:** Acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos.
- **Cloud Computing:** Es un paradigma que permite ofrecer servicios de computación a través de Internet. De modo que los usuarios puedan acceder a los servicios disponibles en la nube.
- **Máquina virtual:** Es un software que emula a una computadora y puede ejecutar programas como si fuese una real.
- **Cloud/Nube:** En el ámbito del Cloud Computing se refiere a Internet y a todos los recursos que se pueden almacenar en él.
- **Google Apps:** Es un servicio ofrecido por Google para uso de sus diversos productos. En él se encuentran diversas aplicaciones para Internet, con un funcionamiento similar a los tradicionales programas para escritorio. Algunos ejemplos serían Gmail, Google Agenda, Talk, Docs y Sites.
- **GoGrid:** Una empresa privada que ofrece un servicio de infraestructura de nube, alojando máquinas virtuales de Linux y Windows gestionadas por un panel de control multi-servidor.
- **Ruby:** Un lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad. Su creador, Yukihiro _matz _ Matsumoto, mezcló partes de sus lenguajes favoritos (Perl, Smalltalk, …) para formar un nuevo lenguaje que incorporara tanto la programación funcional como la programación imperativa.
- **Sinatra:** Sinatra es un DSL escrito en Ruby para el desarrollo de aplicaciones Web con un mínimo esfuerzo.
- **DSL:** En el desarrollo de software a domain-specific language (DSL) es un lenguaje de programación o especificación de lenguaje dedicada al dominio de

problema en particular. Una técnica de representación del problema en particular y/o una técnica de solución particular. El concepto no es nuevo, lenguajes de programación de propósito general y todo tipo de modelado y especificación de lenguajes han existido siempre, pero el término ha pasado a ser de los más populares debido a la subida de los modelos de dominio específico.

- **Autenticación:** En la seguridad de ordenador, la autenticación es el proceso de intento de verificar la identidad digital del remitente de una comunicación como una petición para conectarse. El remitente siendo autenticado puede ser una persona que usa un ordenador, un ordenador por sí mismo o un programa del ordenador. En un Web de confianza, "autenticación" es un modo de asegurar que los usuarios son quién ellos dicen que ellos son - que el usuario que intenta realizar funciones en un sistema es de hecho el usuario que tiene la autorización para hacer así.

- **Cookie:** Es un fragmento de información que se almacena en el disco duro del visitante de una página Web a través de su modo a petición del servidor de la página. Esta información puede ser luego recuperada por el servidor en posteriores visitas. En ocasiones también se le llama "huella".

- **Base de Datos:** es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.. En la actualidad, la mayoría de las bases de datos están en formato digital (electrónico), que ofrece un amplio rango de soluciones al problema de almacenar datos.

- **Aplicación Web:** aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor Web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores Web en la que se confía la ejecución al navegador.

- **Framework:** Es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Suele incluir

soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

- **Repositorio:** Lugar en el que se almacenan los datos actualizados e históricos, a menudo en un servidor. A veces se le denomina depósito o depot. Puede ser un sistema de archivos en un disco duro, un banco de datos, etc.

- **Toolkit:** Literalmente kit de herramientas. Más formalmente: equipo de instrumentos, grupo de programas y rutinas que se utilizan como base para la programación de un nuevo sistema

- **W3C:** El World Wide Web Consortium (W3C) es una comunidad internacional que desarrolla estándares que aseguran el crecimiento de la Web a largo plazo.

- **Ruby on Rails:** Es un framework de aplicaciones Web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración

- **Modelo-Vista-Controlador (MVC):** El paradigma modelo vista controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

- **Capa ORM:** Un ORM o (Object Relation Mapper) es una técnica de programación que nos permite convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, es decir, las tablas de nuestra base de datos pasan a ser clases y los registros objetos que podemos manejar con facilidad.

- **Theme Roller:** ThemeRoller es una herramienta que encontramos en jQuery UI (interfaz de usuario) y que permite ajustar y definir colores, tipografías, etc. de los componentes para interfaz o widgets que ofrece esta biblioteca. Es un forma muy rápida de implementar una interfaz usando todas las posibilidades de jQuery y un recurso muy valioso si es que uno está dando los primeros pasos con esta biblioteca

10. Bibliografía

- Ruby

http://www.ruby-lang.org/es/documentation/

Programming Ruby - The Pragmatic Programmers Guide. Second Edition

Authors: Dave Thomas with Chad Fowler and Andy Hunt

Ed: The Pragmatic Bookshelf

- Sequel

http://sequel.rubyforge.org/documentation.html

- JSON

http://www.json.org/

- Sinatra

http://rubydoc.info/gems/sinatra/1.2.6/file/README.es.rdoc

- HTML, AJAX, JavaScript, CSS, XHTML

http://www.librosweb.es/

http://www.w3schools.com/default.asp

- jQuery

http://docs.jquery.com/Main_Page

Learning jQuery 1.3

Better Interaction Design and Web Development with Simple JavaScript Techniques

Authors: Jonathan Chaffer and Karl Swedberg

Ed: Packt Publishing

- jQueryUI

http://jqueryui.com/demos/

- DataTable

http://www.datatables.net/

ANEXO. Instalación y funcionamiento de la aplicación

El entorno en el que se ha desarrollado el proyecto y para el cual explicamos su instalación es Linux. Mas en concreto hemos usado la distribución más comercial como es Ubuntu.

Esta versión de Linux tiene algunos gestores de paquetes que nos harán más cómoda la instalación de algunos de nuestros componentes. Este gestor tiene dos opciones para instalar software. Una de ellas es mediante el terminal por línea de comandos y la segunda es una interfaz grafica a la que se accede mediante el menú aplicaciones > Centro de Software de Ubuntu.

Por tanto, para este entorno, necesitaremos las siguientes herramientas:

- Ruby:

Mediante el terminal por línea de comandos:

$ sudo apt-get install ruby irb rdoc

Mediante la versión gráfica, buscamos Ruby en el software disponible y elegimos el que se llama Intérprete del lenguaje de scripts orientado a objetos Ruby 1.8. Seleccionamos la opción de instalar.

- SQLite:

Mediante el terminal por línea de comandos:

$ sudo apt-get install sqlite

Mediante la versión gráfica, buscamos Ruby en el software disponible y elegimos el que se llama Interfaz SQLite3 para Ruby 1.8. Seleccionamos la opción de instalar.

- Rubygems:

Para hacerlo de forma manual necesitamos descargar el código fuente de la página Web oficial.

http://rubygems.org/pages/download

Descomprimimos el archivo descargado y, en el terminal, nos colocamos en el directorio donde lo hemos descomprimido.
Finalmente lo instalamos mediante la orden

$ ruby setup.rb

Mediante la versión gráfica, buscamos Ruby en el software disponible y elegimos el que se llama infraestructura de gestión de paquetes para aplicaciones y bibliotecas Ruby. Seleccionamos la opción de instalar.

Una vez instalado Rubygems pasamos a instalar las librerías Ruby que usara nuestro proyecto mediante este gestor de paquetes mencionado. Este gestor solo admite línea de comandos.

- Rack:

$ sudo gem install rack

- Sinatra:

$ sudo gem install sinatra

- Sequel:

$ sudo gem install sequel

- JSON:

Instalaremos la variante de JSON implementado en Ruby puramente. Existen otras variantes implementadas en C, pero debido que la base de nuestro proyecto está en lenguaje Ruby, usaremos la variante puramente escrita en Ruby.

$ sudo gem install json_pure

Una vez instaladas las herramientas, necesitaremos arrancar el servidor de la aplicación. Para ello necesitamos cargarlo mediante el servidor Sinatra.
Para ello, nos colocamos sobre el directorio de la aplicación en el terminal y ejecutamos la siguiente orden:

$ ruby campus-cloud-server.rb

Esto cargará nuestro servidor, el cual por defecto cargará el servidor Sinatra mediante su archivo de configuración. El puerto en el que Sinatra carga nuestra aplicación es por defecto el 4567.

Finalmente podemos acceder a la aplicación mediante la dirección:

http://localhost:4567/