FACULTAD DE CIENCIAS

Facultad de Ciencias

Proyecto Fin de Carrera

Aplicación móvil para un Sistema de Gestión Educativa

(Mobile application for an Education Management System)

Para acceder al Título de

INGENIERO EN INFORMÁTICA

Autor: Pablo Martínez Pérez
Director: Carlos Blanco Bueno
Codirector: Celestino Güemes Seoane

Febrero - 2014

Agradecimientos

Este proyecto es la finalización de varios años de trabajo y esfuerzo, que no hubieran sido posible de realizar sin el apoyo de las personas que durante tantos años me han acompañado, motivado y ayudado.

En especial a mi familia y a mis padres, cuyo soporte moral y económico ha sido clave para completar estos estudios. La motivación que desde pequeño me han inculcado para realizar unos estudios universitarios y alcanzar un mejor futuro profesional y personal.

También recordar a todos los compañeros de la facultad y amigos, por tantos momentos compartidos, en especial a Diego y Juan Carlos por la amistad que nos une y los momentos vividos por Europa durante nuestro Erasmus, sobre todo a diego que me tuvo que aguantar un año de compañero.

A todas esas personas tan especiales que conocí en Austria y que significó el gran cambio en mi vida y en mi futuro. En especial a mi novia Saana, por creer en las casualidades, en el futuro y ser mi gran apoyo en estos últimos años de estudios y en especial en mi proceso de emigración.

A todos esos políticos que llenan nuestro camino de piedras haciendo negocio con nuestros servicios públicos y acabando con la igualdad de oportunidades para acceder a la universidad. También precarizando nuestro futuro y obligándonos a emigrar en busca de unas condiciones dignas.

A nuestros profesores, que siguen luchando por darnos la mejor educación a pesar de los obstáculos que les ponen día a día.

A los compañeros de Atos y en especial al grupo de Yedra, por los momentos vividos, la experiencia lograda y todo el apoyo que me han dado durante este proyecto.

A todos y cada uno de ellos, gracias de corazón por ayudarme a llegar hasta aquí.

Resumen

El crecimiento del uso de dispositivos móviles ha modificado sustancialmente el uso de servicios en internet y el modo de acceso a estos, atrayendo a su vez a un gran número de usuarios que nunca antes habían hecho uso de servicios en red a través de dispositivos multimedia u ordenadores. En este marco ha sido necesario para un gran número de empresas y organismos adaptar sus servicios al uso a través de dispositivos móviles, bien sea adaptando sus páginas web al uso a través del navegador de estos dispositivos o creando aplicaciones exclusivas para los distintos sistemas operativos.

Muchos de estos sistemas se presentan demasiado obsoletos para ser sujetos a cambios drásticos en su interfaz o su funcionamiento, es por tanto necesario la creación de nuevos sistemas informáticos que ofrezcan los mismos servicios haciendo uso de tecnologías más modernas y escalables.

Con este análisis se exponen las razones que han llevado a diseñar el sistema que nos ocupa. El objetivo inicial del proyecto es proporcionar una aplicación para dispositivos móviles que permita a los usuarios del sistema de seguimiento educativo de Yedra hacer uso de los servicios presentes en la versión web, dada la mala experiencia de usuario de la versión web actual en dispositivos móviles. Esta aproximación inicial ha llevado a tomar la decisión de diseñar el sistema completo, incluyendo el servidor encargado de manejar la conexión con las aplicaciones y la base de datos, dado que el sistema actual es demasiado obsoleto para comunicarse eficientemente con las aplicaciones móviles.

Este proyecto ha sido realizado en la empresa "Atos Worldgrid" creando este sistema para incluirlo en los servicios que la Consejería de Educación de Cantabria proporciona a los ciudadanos de dicha región. Los objetivos son proporcionar a los ciudadanos cántabros con hijos escolarizados en la región una aplicación móvil capaz de realizar el seguimiento de las notas, faltas de asistencia, información de centro y profesores de sus hijos, y la comunicación interna en el sistema a través de mensajes entre usuarios.

Palabras Clave: Aplicación móvil, servidor, sistema de administración educativa, Yedra, Educación, Java, Spring, MyBatis, Javascript, Titanium Appcelerator

Abstract

The increase use of mobile devices has substantially changed the use of internet services and how the people access to them. This increase has also brought large number of users who have never before made use of internet services through computers or mobile devices. It has been necessary for a large number of companies to adapt their services to be used through mobile devices, either adapting their services to be used through the browser of these devices or creating unique applications for each operating system.

Many of these software systems are too old to be upgraded and add new features or change their frontends; therefore it is necessary to create new software systems using newer technologies to provide the same services for these devices.

The aim of the project is to develop a mobile application that allows users to use the web services of the Yedra education managment system. This aim has led to créate the whole software system, including the server that handle the requests between the mobile applications and the database, because the system that provide now those services cannot comunicate with mobile devices.

This project has been developed by me in "Atos Worldgrid" and it will be integrated in the services of the Government of Cantabria. The aim is to provide a mobile application for the citizens who have children in the education system of Cantabria, to manage the grades, the attendance, the information of the schools and teachers and the internal communication in the system.

Keywords: Mobile Application, server, education management system, Yedra, education, Java, Spring, MyBatis, Javascript, Titanium Appcelerator

Tabla de Contenidos

1 INTRODUCCIÓN 9
1.1 SISTEMAS DE GESTIÓN EDUCATIVA 9
1.2 SISTEMA YEDRA 10
1.3 MOTIVACIÓN 10
1.4 OBJETIVOS DEL PROYECTO 12

2 HERRAMIENTAS Y MÉTODOS 13
2.1 TECNOLOGÍAS Y HERRAMIENTAS 13
2.1.1 Tecnologías de red 13
2.1.1.1 Protocolo HTTP 13
2.1.1.2 Interfaz REST 15
2.1.2 Frameworks para el desarrollo del servidor 17
2.1.2.1 MyBatis 17
2.1.2.2 Spring MVC 18
2.1.2.3 Spring Security 20
2.1.3 Framework para el desarrollo de aplicaciones móviles 20
2.1.3.1 Appcelerator Titanium 21
2.2 METODOLOGÍAS 22

3 CAPTURA Y ANÁLISIS DE REQUISITOS 24
3.1 REQUISITOS FUNCIONALES 24
3.2 REQUISITOS NO FUNCIONALES 25
3.3 DIAGRAMA DE CASOS DE USO 26

4 DISEÑO DEL SISTEMA 29
4.1 DISEÑO ARQUITECTÓNICO 29
4.1.1 Diseño del Servidor 29
4.1.2 Diseño de la aplicación móvil 31
4.2 DISEÑO DETALLADO 32
4.2.1 Diseño del servidor 32
4.2.2 Diseño de la aplicación móvil 45

5 IMPLEMENTACIÓN 47
5.1 IMPLEMENTACIÓN DEL SERVIDOR 47
5.1.1 Implementación de la API REST 48
5.1.2 Implementación de la capa de negocio 51
5.1.3 Implementación de la capa de acceso a datos 54
5.1.4 Implementación del modelo de seguridad 58
5.2 IMPLEMENTACIÓN DE LA APLICACIÓN MÓVIL 60
5.2.1 Implementación de la capa de acceso a datos 60
5.2.2 Implementación de las vistas y controladores 62

6 PRUEBAS 67
6.1 PRUEBAS UNITARIAS 67
6.2 PRUEBAS DE INTEGRACIÓN 67
6.3 PRUEBAS DE SISTEMA 68
6.4 PRUEBAS DE ACEPTACIÓN 68

7 CONCLUSIONES Y TRABAJO FUTURO 69
7.1 CONCLUSIONES 69
7.2 PLANIFICACIÓN Y EVOLUCIÓN DEL SERVIDOR 70

8 REFERENCIAS 71

Índice de figuras

FIGURE 1 - INTERFAZ REST 16
FIGURE 2 - MYBATIS LOGO 17
FIGURE 3 - SPRING LOGO 18
FIGURE 4 - FRONT CONTROLLER PATTERN 19
FIGURE 5 - TITANIUM LOGO 21
FIGURE 6 – ARQUITECTURA TITANIUM 22
FIGURE 7 - METODOLOGÍA SCRUM 23
FIGURE 8 - DIAGRAMA DE CASOS DE USO GENERAL 26
FIGURE 9 - DIAGRAMA DE CASO DE USO MENSAJES 26
FIGURE 10 - ARQUITECTURA DE TRES CAPAS 29
FIGURE 11 - ARQUITECTURA MVC 31
FIGURE 12 - DISEÑO DEL SERVIDOR 32
FIGURE 13 - DIAGRAMA DE COMPONENTES 44
FIGURE 14 - WIREFRAME INFORMACION DE ALUMNO FIGURE 15 - WIREFRAME SELECCION DE ALUMNOS 45
FIGURE 16 - WIREFRAME NOTAS DEL ALUMNO FIGURE 17 - WIREFRAME FALTAS DE ASISTENCIA 46
FIGURE 18 - WIREFRAME MENSAJES FIGURE 19 - WIREFRAME INFORMACION DE USUARIO 46
FIGURE 20 - ESTRUCTURA DE LA IMPLEMENTACIÓN 47
FIGURE 21 - CONTROLADOR DE ALUMNO 48
FIGURE 22 - CONTROLADOR DE NOTAS 49
FIGURE 23 - CONTROLADOR DE FALTA 49
FIGURE 24 - CONTROLADOR DE MENSAJES RECIBIDOS 50
FIGURE 25 - CONTROLADOR DE MENSAJES ENVIADOS 51
FIGURE 26 - SERVICIO DEL RECURSO NOTAS 52
FIGURE 27 - SERVICIO DEL RECURSO PERSONA 52
FIGURE 28 - SERVICIO DEL RECURSO FALTA 53
FIGURE 29 - SERVICIO DEL RECURSO MENSAJES - MÉTODO CREAMENSAJE 54
FIGURE 30 - INTERFAZ ALUMNOMAPPER 54
FIGURE 31 - ARCHIVO XML DE LA CONSULTA DE ALUMNO 55
FIGURE 32 - CONSULTA SQL DE ALUMNO 55
FIGURE 33 - CONSULTA DEL RECURSO NOTAS 56
FIGURE 34 - CONSULTA DEL RECURSO FALTAS 57
FIGURE 35 - CONSULTA USUARIO Y PERFILES FIGURE 36 - CONSULTA UPDATEINTENTOSFALLIDOS 57
FIGURE 37 - CONFIGURACION SPRINGSECURITY 58
FIGURE 38 - RESTTOKEN CON LAS CREDENCIALES DEL USUARIO 58
FIGURE 39 - AUTHENTICATIONPROVIDER, MÉTODO DE AUTENTICACIÓN 59
FIGURE 40 - LLAMADA AL RECURSO USUARIO 61
FIGURE 41 - LLAMADA PARA JUSTIFICAR UNA FALTA 62
FIGURE 42 - PANTALLA DE LOGIN FIGURE 43 - PANTALLA PRINCIPAL 63
FIGURE 44 - PANTALLA DE INFORMACIÓN DE USUARIO FIGURE 45 - PANTALLA DE INFORMACIÓN DE MATRÍCULA 64
FIGURE 46 - PANTALLA DE NOTAS 1ª EVAL FIGURE 47 - PANTALLA DE NOTAS 2ª EVAL 64
FIGURE 48 - PANTALLA DE LISTA DE FALTAS FIGURE 49 - PANTALLA DE VISTA DE CALENDARIO 65
FIGURE 50 - PANTALLA DE JUSTIFICACIÓN FIGURE 51 - PANTALLA DE CREACIÓN DE MENSAJE 66
FIGURE 52 - PANTALLA DE CONFIRMACIÓN DE MENSAJE FIGURE 53 - PANTALLA DE MENSAJES 66

Índice de tablas

TABLE 1 - REQUISITOS FUNCIONALES 25
TABLE 2 - REQUISITOS NO FUNCIONALES 25
TABLE 3 - CASOS DE USO: PLANTILLA DE CONSULTA DE ALUMNOS 27
TABLE 4 - CASOS DE USO: PLANTILLA DE JUSTIFICA FALTAS 28
TABLE 5 - CASOS DE USO: PLANTILLA DE CREAR MENSAJE 28
TABLE 6 - TABLA DE RECURSOS 33

1 Introducción

A lo largo de los siguientes capítulos se realizará una descripción del trabajo realizado durante seis meses en el desarrollo de un servidor y una aplicación móvil para el sistema de educación educativa Yedra, usado por la consejería de educación de Cantabria.

Dicho trabajo se ha realizado en la empresa *Atos Worldgrid* y ha contado con el apoyo del equipo de administración y desarrollo de Yedra, el cual ha proporcionado el conocimiento necesario de la base de datos sobre la que se desarrolla el sistema actual y de las necesidades de los usuarios para el desarrollo de una aplicación móvil que mejore la experiencia de usuario mediante el acceso a través de *Smartphone*.

1.1 Sistemas de Gestión Educativa

Un sistema de gestión educativa es un sistema de software encargado de administrar y gestionar uno o varios centros educativos. Este tipo de sistemas permiten almacenar información de cursos, alumnos y docentes y permiten administrar y realizar el seguimiento de los resultados de los alumnos, las estadísticas del centro y la comunicación interna y externa.

Este tipo de sistemas se ha convertido en una pieza clave en la administración de los centros educativos, pues facilita la gestión de estos organismos a nivel provincial, comunitario o nacional y ayuda a gestionar las estadísticas curriculares del alumnado, la asignación de recursos y alumnos en función de los criterios legales, y a la comunicación con los centros, con los docentes y con las familias.

En nuestro país cada comunidad autónoma posee las competencias para gestionar su sistema educativo, por tanto estos sistemas pertenecen a las consejerías de educación de cada comunidad autónoma.

Éstas poseen varios sistemas de software encargados de realizar la gestión de los centros educativos. Algunas comparten sistemas que se originaron de un primer proyecto de software de gestión educativa pero una vez adquirido, cada comunidad se ha encargado de evolucionarlo y añadir funcionalidades propias. De este modo, pese a compartir diseños parecidos en la base de datos o en las funcionalidades de la web, las comunidades poseen servicios y funcionalidades distintas.

1.2 Sistema Yedra

El sistema de gestión educativa de Cantabria (en adelante Sistema Yedra) se encarga de gestionar la información de todos los centros educativos de Cantabria, desde centros de educación infantil, a colegios, institutos de secundaria y centros de Formación Profesional; con vistas a integrar en un futuro los centros de idiomas y los conservatorios de música de la comunidad.

Es por tanto un sistema que soporta un funcionamiento y un gran numero de esquemas organizativos distintos, para lo cual ofrece un sistema web desde el cual los usuarios, en base a sus permisos, pueden consultar y gestionar información personal o de los centros. También permite acciones más complejas como calculo y asignación de becas, generación de horarios y generación de nominas

El sistema se apoya sobre tres bases de datos distintas (SEGEDU, GESCEN y MODACC), las cuales sirven como soporte de seguimiento educativo, gestión de centros y módulo de acceso respectivamente. SEGEDU y GESCEN comparten información que se sincroniza diariamente para mantener la coherencia, siendo la primera de ellas la utilizada para tareas de consulta de información sobre alumnos, cursos y centros; y la segunda la utilizada por los servicios web dedicados a la gestión por parte del personal administrativo.

Durante el análisis de requisitos se llegó a la conclusión que para cumplir los objetivos del proyecto no sería necesario conectar con la base de datos GESCEN, por tanto las bases de datos implicadas en el proyecto serán SEGEDU, para la consulta y la gestión de datos, y MODACC para el acceso a las credenciales de usuario.

1.3 Motivación

En los últimos años el aumento del uso de *Smartphones* ha cambiado el mercado y el modo de acceso a internet de los usuarios. Esto también ha permitido un aumento de los usuarios con un perfil de conocimientos en internet bajo que posiblemente encontraban en el uso del PC un obstáculo en el aprendizaje y el manejo de la navegación en internet.

Internet ha pasado de usarse como una herramienta de consulta esporádica en nuestro tiempo de ocio a una herramienta que interactúa en todo momento con nuestra vida diaria y nos facilita el aprendizaje, la comunicación y el acceso a servicios en internet desde la palma de nuestra mano.

Este cambio ha afectado de manera importante a las empresas que ofrecen servicios a través de la red, de forma que para mantener la usabilidad y la experiencia de usuario de sus productos han necesitado adaptar sus páginas web para su uso en un *Smartphone*, o bien diseñar aplicaciones específicas en cada plataforma móvil que permitan al usuario acceder de manera cómoda y rápida a sus servicios.

La motivación de este proyecto se ha basado en este cambio del modelo de uso y del perfil del usuario. La plataforma Yedra ha sufrido desde sus inicios una infrautilización por parte de los padres en Cantabria, los principales argumentos han sido una mala experiencia de usuario en la navegación a través de la página web, una mala gestión del servicio por parte de los centros, y una dificultad por parte de los usuarios para recordar sus credenciales de acceso o para sentarse frente a un PC para consultar esta información. Estos dos últimos argumentos se engloban dentro de la problemática que existe entre los múltiples perfiles de usuario del servicio, pues si bien hay padres que trabajan, tienen acceso y se manejan perfectamente en un PC y en internet, existen otros para los que se hacía imposible usar el servicio, bien por falta de PC o conexión a internet o por falta de conocimiento.

En los últimos años con la introducción de los *Smartphones* este último perfil de usuario ha disminuido drásticamente, pues el acceso a internet se ha vuelto casi universal y el nivel de aprendizaje necesario para hacer uso de internet ha disminuido. Con este cambio en el modo de acceso a internet de los usuarios se ha comprobado que la experiencia de usuario al acceder a Yedra se ha visto muy perjudicada, pues las páginas web del sistema no están diseñadas para usarse en pantallas pequeñas y táctiles y no se dispone de una aplicación específica para *Smartphones* para acceder al servicio.

Por tanto la motivación para realizar este proyecto ha sido diseñar y proporcionar el sistema necesario para que los usuarios puedan acceder a las funcionalidades actuales del sistema Yedra a través de sus *Smartphones*. Para ello se deberá crear un sistema software como un servicio *SaaS*, que nos permitirá crear toda la lógica de negocio en el servidor y construir una aplicación móvil de uso sencillo y adaptada a todo tipo de usuarios que represente la información requerida de una forma simple en todo tipo de *Smarthpones*.

1.4 Objetivos del proyecto

El principal objetivo del proyecto es crear un sistema software que pueda ofrecer unos servicios en el móvil parecidos a los ofrecidos actualmente a través de la página web. Para ello será necesario la creación de una aplicación móvil y de un servidor nuevo que dé soporte a la aplicación, pues el sistema actual no permite la integración de un cliente de estas características.

Por tanto los objetivos específicos serán los siguientes:

1. Crear una aplicación móvil que permita a los usuarios gestionar los servicios de Yedra; pudiendo consultar notas, faltas de asistencia o información de sus hijos así como su buzón de correo interno.

2. Crear un servidor que dé servicio a esta aplicación móvil, el cual deberá implementar una interfaz REST y no guardar información de sesión o cookies en el servidor para evitar la sobrecarga del sistema.

3. Crear una arquitectura de servidor escalable que permita la evolución para desarrollar nuevos servicios que soporten otros perfiles de usuario (ej. Profesores).

4. Proteger la comunicación entre la aplicación móvil y el servidor, dada la protección legal que requiere dicha información.

5. Diseñar un sistema de seguridad en el servidor que soporte distintos perfiles de usuario, niveles de seguridad y proteja los servicios web en base a dichos perfiles.

6. Diseñar una interfaz gráfica en la aplicación móvil adaptada a distintos formatos de pantallas, de fácil lectura y uso y que permita recordar las credenciales de usuario sin comprometer la seguridad.

2 Herramientas y métodos

En este apartado se expondrán las diferentes tecnologías, herramientas y métodos utilizados en el desarrollo del proyecto.

En un proyecto de este alcance, donde se requiere diseñar un servidor y una aplicación móvil que permita a los usuarios utilizar una serie de servicios, influyen una serie de tecnologías y protocolos de red que deben ser tomados en cuenta. Del mismo modo disponemos de una infinidad de lenguajes de programación y frameworks de desarrollo para diseñar el sistema, de todos ellos elegiremos los que mejor se adapten a nuestras necesidades y a las de la empresa.

2.1 Tecnologías y herramientas

A continuación se describen las características de los protocolos utilizados y de los frameworks de desarrollo de los distintos componentes del sistema.

2.1.1 Tecnologías de red

En este capítulo se describen el protocolo de comunicación utilizado entre cliente-servidor y la arquitectura utilizada para el diseño de los servicios web en el servidor.

2.1.1.1 Protocolo HTTP

El protocolo HTTP es un protocolo de aplicación para la comunicación entre sistemas distribuidos usado en las transacciones de la World Wide Web (Gourley & Totty, 2002). Este protocolo permite la comunicación entre una gran variedad de clientes y un servidor y soporta diversas configuraciones de red.

La comunicación entre los clientes y el servidor de una aplicación web suele realizarse sobre el protocolo TCP/IP pero cualquier otro protocolo de transporte es soportado por HTTP.

El proceso de comunicación entre el cliente y el servidor consiste en parejas de peticiones y respuestas, el cliente inicia la comunicación con un mensaje de petición (*HTTP request*) al servidor y este le responde con otro mensaje de respuesta (*HTTP response*). El mensaje de petición debe ir dirigido a un determinado recurso del servidor y para acceder a estos recursos se hace uso de la URI (*Uniform Resource Identifier*) para identificarlos y un método de petición.

Los métodos de petición determinan la acción que se aplica al recurso identificado por la URI; de los ocho métodos existentes los más importantes y los utilizados en el proyecto son GET, POST, PUT y DELETE:

- GET: obtiene una representación del recurso identificado por la URI. Este método solo puede ser utilizado solo obtener datos del servidor.

- POST: Crea un nuevo recurso cuya información es enviada al servidor en el cuerpo del mensaje. Este método también puede ser utilizado para actualizar información de un recurso.

- PUT: Actualiza un recurso identificado con la URI con la información enviada en el cuerpo del mensaje, si dicho recurso no existe crea uno nuevo asignándole los datos enviados.

- DELETE: Borra el recurso identificado mediante su URI.

Tras enviar un mensaje de petición con uno de los anteriores métodos el servidor responde con una serie de códigos que indican al cliente como debe interpretar la respuesta del servidor. El protocolo HTTP define una serie de códigos divididos en códigos de información, de éxito, de redirección, de error por parte del cliente o de error por parte del servidor.

Por último la estructura de las peticiones y respuestas se compone del cuerpo del mensaje y de una serie de cabeceras que definen los parámetros de la transacción. Estas cabeceras son parejas clave-valor que definen parámetros como el tipo de contenido aceptado, el tipo de contenido retornado, parámetros de autenticación, parámetros de cache, de fecha de la transacción o de codificación.

2.1.1.2 Interfaz REST

REST (*REpresentational State Transfer*) es una arquitectura de desarrollo web apoyada totalmente sobre el protocolo HTTP (Allamaraju, 2010) que nos permite crear servicios o aplicaciones que pueden ser usadas por cualquier dispositivo que entienda HTTP. Por tanto es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a internet.

Al igual que el protocolo HTTP, la arquitectura se divide entre clientes y servidores donde los clientes mandan peticiones que los servidores procesan y retornan la respuesta apropiada.

Las APIs de una interfaz REST se definen en base a *resources*, que pueden ser cualquier objeto, información o concepto que es direccionado mediante una URI y es accedido por el cliente mediante peticiones.

El uso de una arquitectura REST nos proporciona una gran escalabilidad de nuestra aplicación, un desarrollo de componentes independientes y una gran modularidad que nos permite separar el control de seguridad de la interfaz de los *resources*, reduciendo la latencia y la complejidad del sistema.

Una de las principales características de este protocolo es que no mantiene ningún estado ni guarda información entre las transacciones, lo que puede suponer un problema para las aplicaciones web que necesitan conocer información del cliente que realiza las peticiones, sin embargo el cliente es el responsable de cachear las respuestas del servidor y mantener su propio estado entre transacciones. Por ese motivo el uso de una arquitectura REST en lugar de una SOAP o XML-RPC es la más indicada para el uso de aplicaciones móvil del lado del cliente, pues el número de usuarios potenciales crece y no es necesario mantener una sesión en el servidor con la información de cada uno porque la metodología de uso en un móvil es distinta a la metodología de un cliente web en un ordenador.

Existen recomendaciones de calidad a la hora de diseñar una arquitectura REST en una aplicación web y sobre todo a la hora de diseñar la API:

- Uso correcto de URIs: Las URIs representan al *resource* al que se accede y estos nombres deben ser únicos, no implicar ninguna acción sobre el resource, deben ser independientes del fromato, mantener una jerarquía lógica y evitar los filtrados de información. La estructura de la URL sería la siguiente:

```
http://{dominio}/{ruta del recurso}?{consulta de filtrado}
http://yedra..educantabria.es/alumno?{consulta de filtrado}
```

- Uso correcto de HTTP: Los nombres de resources no deben implicar ninguna acción, para ello disponemos de los métodos de HTTP que nos permiten manipular el recurso como se indicó en el apartado anterior.

- Implementación de Hypermedia: El cliente no tiene que conocer todos los detalles de la API por lo que el servidor debe proporcionar información extra para acceder a los recursos. Por ejemplo después de crear un nuevo recurso mediante una petición POST el servidor debe responder incluyendo la URL para acceder al nuevo recurso en la cabecera *Location*.

En la siguiente figura podemos observar un ejemplo de diseño de una arquitectura REST:

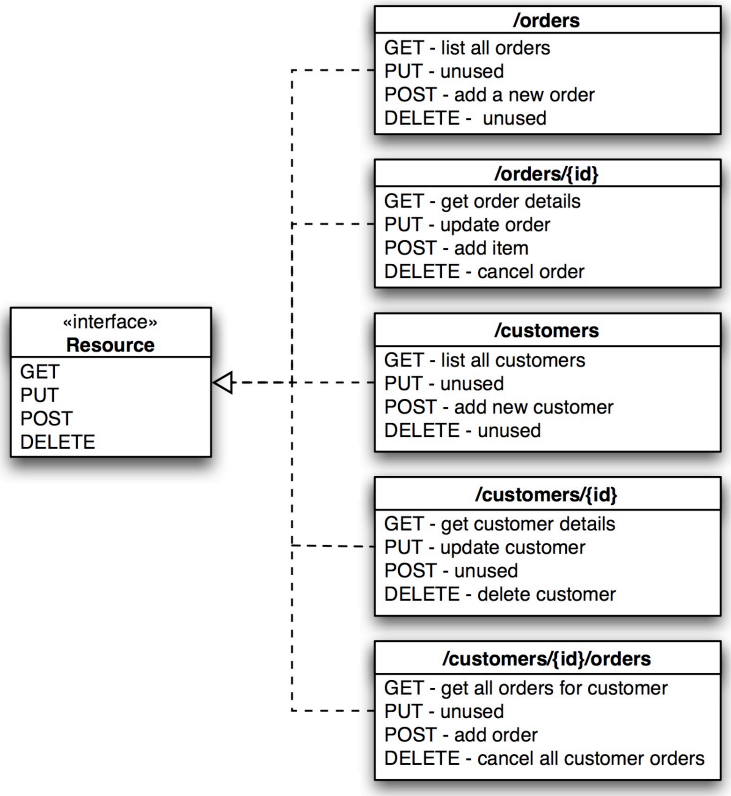

Figure 1 - Interfaz REST

2.1.2 Frameworks para el desarrollo del servidor

A la hora de implementar un servidor web tenemos a nuestra disposición un gran número de herramientas, lenguajes y frameworks que nos proporcionan una gran flexibilidad para adaptar el servidor a nuestras necesidades. En este proyecto ha sido necesario utilizar una serie de frameworks con los que se ha trabajado en otros proyectos de la empresa para facilitar el mantenimiento y evolución del software en las siguientes etapas.

Un servidor web normalmente consta de varias capas de diseño independientes que abstraen la arquitectura y la dividen en varias áreas funcionales, como pueden ser la persistencia de datos, la conexión con la base de datos, la seguridad, la capa de servicios o la capa de presentación. Cada una de estas etapas puede ser diseñada con distintos frameworks que nos proporcionan características y funcionalidades concretas. En este proyecto se ha utilizado el framework MyBatis para implementar la capa de persistencia, que se encarga de conectar el servidor con la base de datos y realizar consultas a esta. Para el manejo de datos, el diseño de servicios y el diseño de la API se ha utilizado el framework Spring MVC y para el manejo de la seguridad Spring Security.

Por último el proyecto se ha construido con Apache Maven y funciona sobre un servidor Apache Tomcat.

2.1.2.1 MyBatis

Figure 2 - MyBatis Logo

MyBatis es un framework de persistencia escrito en Java con soporte para consultas SQL, *stored procedures* o mapeados avanzados (Reddy, 2013). En lugar de utilizar los procedimientos manuales de conexión, manejo de parámetros o obtención de resultados que utiliza JDBC para acceder a una base de datos, MyBatis usa procedimientos simples y anotaciones en XML con los que se pueden configurar mapeados de primitivas, tablas o Java POJOs (*Plain Old Java Objects*) para obtener datos de la base de datos.

Existen otros frameworks, como *Hibernate*, que mapean objetos Java a tablas de la base de datos y permiten realizar operaciones sobre esas tablas tratándolas

como simples objetos. Sin embargo se aconseja usar este tipo de métodos en bases de datos normalizadas, ya que de lo contrario se pueden producir fallos de integración.

En este proyecto nos encontramos con una base de datos que ha sido objeto de numerosos cambios en virtud de añadir nuevas funcionalidades en cada cambio legal o nuevos requerimientos de la Consejería de Educación, por tanto la evolución de la base de datos dentro de unos protocolos de normalización ha sido difícil de gestionar.

Por tanto, a propuesta de la empresa se decidió prescindir de frameworks más maduros como Hibernate y apostar por uno especializado en bases de datos no normalizadas como es MyBatis.

2.1.2.2 Spring MVC

Figure 3 - Spring Logo

Spring MVC es un framework para el desarrollo de aplicaciones web bajo el patrón Modelo-Vista-Controlador en la plataforma Java EE (Johnson, Hoeller, Arendsen, Risberg, & Sampaleanu, 2005). Este framework define una serie de interfaces que definen la comunicación interna de los componentes de los modelos con las vistas y los controladores y que el programador se encarga de implementar según los requerimientos de su aplicación.

Estas interfaces implementan la API de Java Servlet y están diseñadas alrededor de un controlador principal llamado *DispatcherServlet*, este controlador implementa el *Front Controller Design Pattern* y se encarga de recoger las peticiones HTTP del cliente y asignarlas a los controladores mapeados para manejar cada recurso de nuestro servidor. Este controlador principal se define en los ficheros de configuración de Spring y nos permitirá seleccionar una o varias URL que definirán los Servlets que manejaran las peticiones.

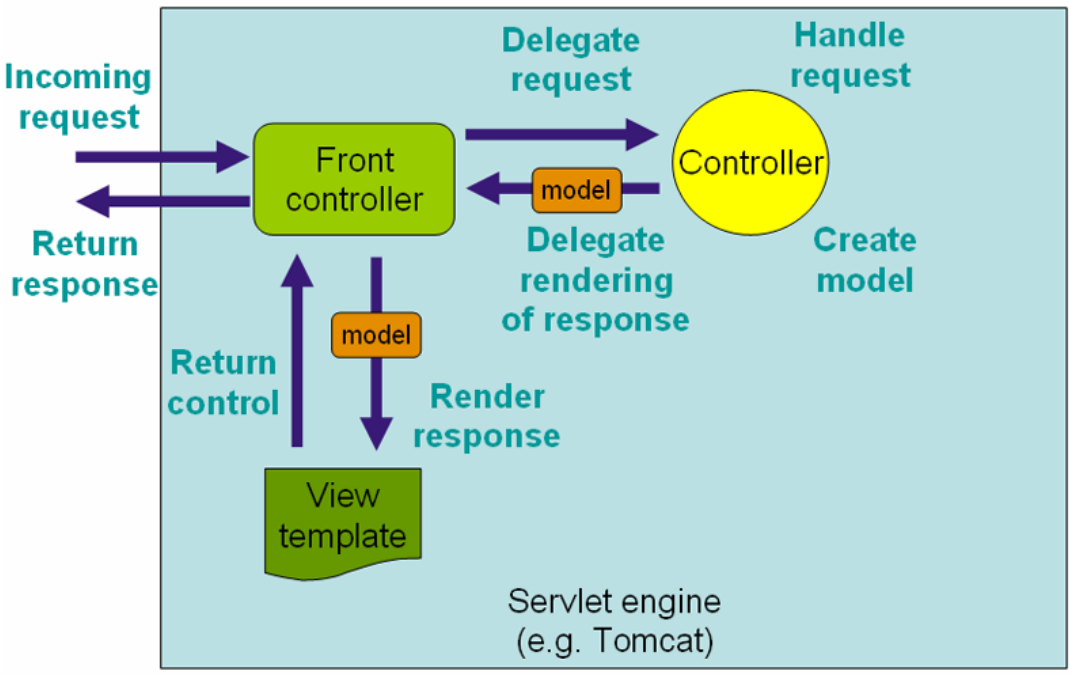

Figure 4 - Front Controller Pattern

Una vez el Dispatcher ha recogido la petición y la envía a los manejadores, estos eligen el controlador que recibirá la petición en base a atributos o condiciones en la URL. Una vez en el controlador, la petición es ejecutada por el método correspondiente y envía la respuesta al cliente.

En el diseño del servidor deberemos crear un controlador por cada recurso que queramos manejar, este diseño se especificará con más detalles en los siguientes apartados.

Cada controlador hará uso de una serie de servicios que implementan la capa de negocio, éstos son los encargados de modificar o obtener la información de los recursos, los cuales se definen en objetos del tipo *Java Beans*, que definen una serie de atributos y *getters & setters* para modificarlos.

Estos objetos sirven para mapear la información de la base de datos, acceder a ella durante la ejecución o modificarla. El framework MyBatis nos permitirá almacenar la información de las consultas a la base de datos en dichos objetos y realizar las operaciones necesarias.

2.1.2.3 Spring Security

Spring Security es un framework integrado en SpringSource que proporciona autenticación, autorización y otra serie de características de seguridad para aplicaciones empresariales.

Las características de Spring Security utilizadas en el proyecto nos permiten verificar las credenciales de usuario, asignarle un perfil de usuario si estas son correctas y permitir o denegar el acceso a diversos recursos de la API dependiendo del perfil de uso y los permisos asignados a dicho usuario.

Para ello Spring Security nos permite configurar diversos métodos de autenticación, usar sesiones de usuario o interceptar las cabeceras de autenticación HTTP si lo usamos en una interfaz REST, crear patrones de autorización a URLs o a métodos de la interfaz. Esto nos proporciona una gran flexibilidad para adaptar el sistema de seguridad a nuestros requisitos.

Este framework nos proporciona implementaciones que podemos integrar fácilmente en nuestra aplicación para obtener la información de usuario de la base de datos y comprobar sus credenciales de manera automática, sin embargo, en este proyecto debemos proporcionar un sistema de seguridad basado en perfiles de usuario que se encuentran en la base de datos y que es el sistema utilizado en los servidores de Yedra, por lo tanto haciendo uso de Spring Security se ha implementado un proveedor de autenticación (Authentication Provider) y un punto de entrada (Authentication Entry Point) para dar el mismo soporte que el servidor actual. Esta implementación se explicará a fondo en posteriores apartados.

2.1.3 Framework para el desarrollo de aplicaciones móviles

El desarrollo de aplicaciones para *Smartphones* ha sufrido un incremento espectacular en los últimos años, el número de usuarios de estos dispositivos y el cambio en los hábitos de uso de internet ha provocado que muchas empresas diseñen aplicaciones para proporcionar a estos usuarios el acceso a los servicios que antes ofrecían exclusivamente a través de páginas web.

Al desarrollar estas aplicaciones en primer lugar se debe decidir el tipo de dispositivo para el cual se diseñará la aplicación, este puede ser un *Smartphone* o *Tablet* usando un determinado sistema operativo (*Android*, *iOS*, *Windows Phone*, *Blackberry OS*). En este caso se deben diseñar aplicaciones distintas en cada uno de los sistemas operativos, lo cual aumenta el coste, el tiempo y la complejidad del desarrollo del proyecto.

Existen otras herramientas y frameworks, como por ejemplo PhoneGap, que nos permiten desarrollar aplicaciones genéricas que después pueden ser usadas en cualquier dispositivo haciendo uso de su motor de renderizado web, estas aplicaciones se desarrollan usando herramientas de desarrollo web como HTML, CSS Javascript. La ventaja de éstas es que con un solo diseño y desarrollo podemos obtener una aplicación que puede ser utilizada en distintas plataformas, sin embargo nos impide utilizar las APIs y tecnologías propias de cada una de ellas.

Existen otros frameworks, como Appcelerator, en los que el desarrollo de la aplicación se realiza en torno al Titanium SDK, que son un conjunto de librerías propias y de acceso a las diferentes APIs de las plataformas. De esta forma el desarrollador puede utilizar algunas características de cada plataforma desde una capa de abstracción superior, utilizando Javascript, y que serán compiladas a para ser comprendidas por el sistema operativo de cada Smartphone.

2.1.3.1 Appcelerator Titanium

Figure 5 - Titanium Logo

Titanium es un framework de desarrollo multiplataforma que proporciona APIs y librerías para desarrollar aplicaciones escritas en Javascript para la mayoría de sistemas operativos actuales (*Android*, *iOS*, *Windows Phone*, *Blackberry OS*).

El SDK de Titanium (Anderson, 2013) nos permite construir aplicaciones que utilicen funcionalidades nativas de cada una de las plataformas, como pueden ser los elementos gráficos, el sistema de archivos, la red, el sonido o los sensores.

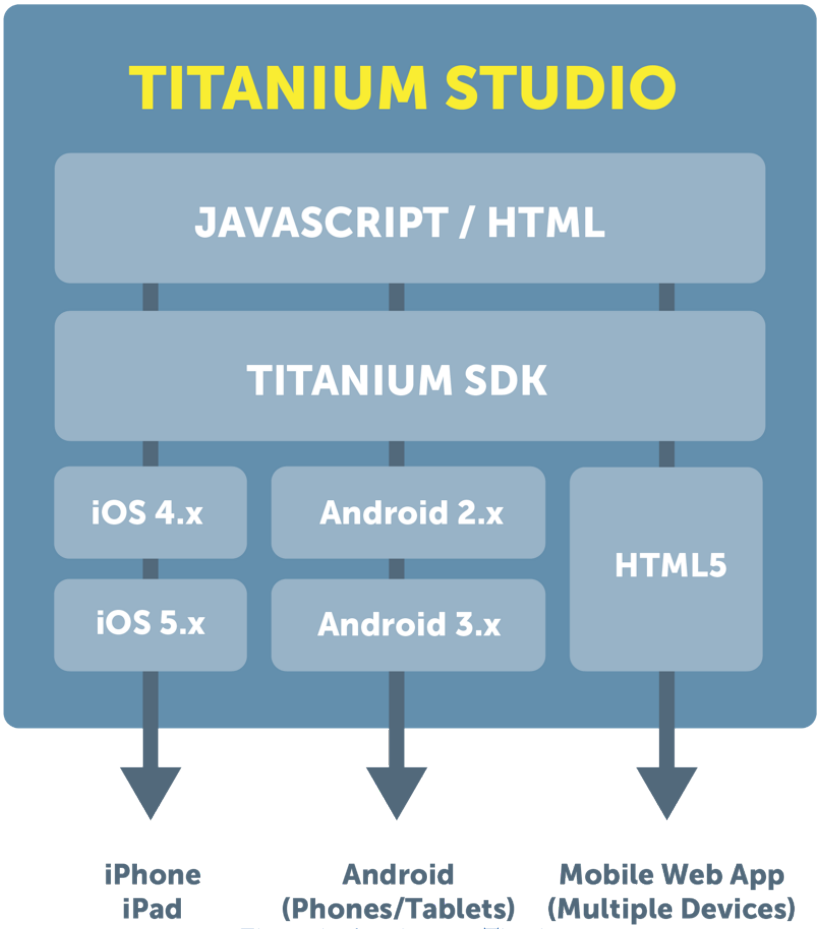

Figure 6 – Arquitectura Titanium

Para trabajar con Titanium disponemos de un IDE (*Titanium Studio*) que incluye todas las herramientas necesarias para desarrollar, testear e instalar la aplicación en diversos dispositivos. Dicho IDE contiene el SDK de Titanium y soporte para varios frameworks de Javascript y Ajax, lo que nos permitirá hacer llamadas a los servicios web que diseñemos a través del protocolo HTTP.

2.2 Metodologías

Una de las primeras decisiones y un factor importante dentro de un proyecto de software es la metodología de desarrollo, la cual definirá cómo se desarrollará el proceso de producción del proyecto. Seguir los preceptos de una metodología permitirá dividir las diferentes etapas del desarrollo y calcular los esfuerzos necesarios para completarlas.

En la realización de este proyecto se ha utilizado una metodología de tipo ágil, la cual sigue un esquema de desarrollo iterativo e incremental. Dentro de este tipo de metodologías se ha escogido la metodología SCRUM por los siguientes motivos:

- Se basa en un desarrollo iterativo, fácilmente adaptable a un proyecto en el que participa un solo desarrollador.
- Permite una gran flexibilidad ante los cambios de requisitos en cualquier parte del proyecto.
- El esfuerzo a realizar es controlable a corto plazo, ya que la carga de cada iteración se define por el esfuerzo de trabajo empleado en la anterior.
- En cada iteración se deben generar trozos ejecutables del proyecto, lo que permite una continua revisión del software creado.

El esquema de una planificación SCRUM (Schwaber & Beedle, 2001) muestra la división del proyecto en "sprints" que tienen una duración determinada y que deben generar una nueva funcionalidad:

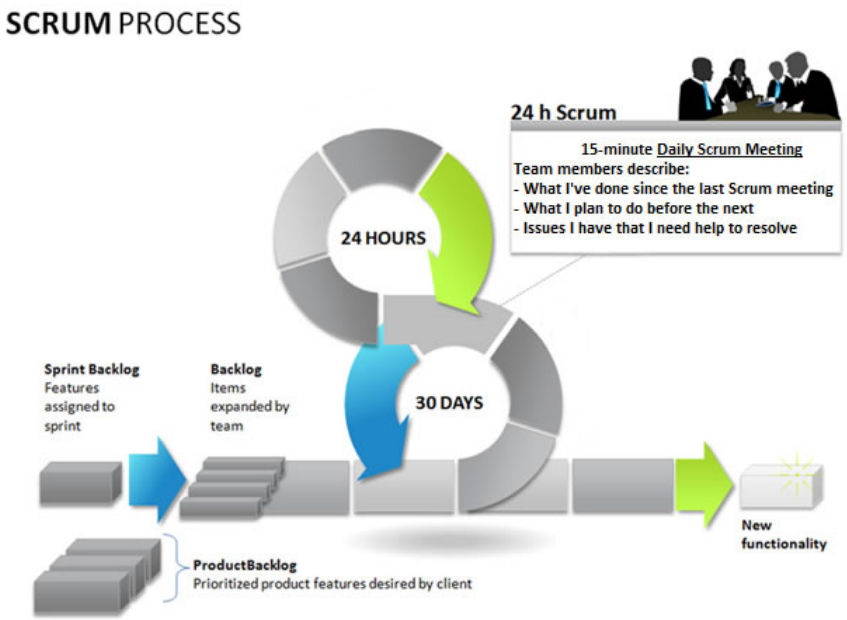

Figure 7 - Metodología Scrum

Cada uno de estos "sprints" se planifican en una reunión previa donde se definen los objetivos que se pretenden alcanzar, a continuación se realiza el sprint, durante el cual se debe realizar también un seguimiento con el fin de controlar si se están alcanzando los objetivos, finalmente se realiza una revisión de los resultados obtenidos y si son satisfactorios se proceden a definir los objetivos del siguiente.

3 Captura y análisis de requisitos

En este capítulo se describen los requisitos que se han tomado en consideración al diseñar la aplicación, desde la idea inicial que se propuso por parte de la empresa con varios agentes y una visión más a largo plazo del proyecto, hasta la decisión final teniendo en cuenta el tiempo disponible y el alcance que debe tener un proyecto de fin de carrera.

3.1 Requisitos funcionales

A continuación se describen los requisitos funcionales que se decidieron en las diversas reuniones con el coordinador del proyecto y el personal de Yedra, estos requisitos están determinados por las funciones actuales que posee la versión web y que son las más utilizadas por los usuarios:

Referencia	Descripción del Requisito
RF01	El usuario podrá obtener la información de los hijos registrados en el sistema
RF02	El usuario podrá obtener su información de usuario
RF03	El usuario podrá obtener la información de las faltas de asistencia de su hijo
RF04	Las información de las faltas de asistencia debe incluir la fecha, la asignatura, el tramo horario, el tipo de falta, si está justificada, y en ese caso el motivo y el comentario de justificación
RF05	El usuario podrá justificar las faltas (**RF04**) que consten como injustificadas, seleccionando el motivo de la falta y pudiendo añadir un comentario.
RF06	El usuario podrá obtener la información del centro de cada uno de sus hijos (**RF01**).
RF07	El usuario podrá obtener la información del tutor de cada uno de sus hijos (**RF01**).
RF08	El usuario podrá obtener las calificaciones del curso de cada hijo (**RF01**).
RF09	El usuario podrá ver los mensajes enviados y recibidos, así como el número de recibidos sin leer. Se le permitirá borrar los mensajes seleccionados.
RF10	El usuario podrá mandar un mensaje al tutor cada hijo (**RF01**).
RF11	El usuario deberá introducir su nombre de usuario y contraseña después de instalar la aplicación, una vez

	logueado satisfactoriamente se mantendrá guardada en el dispositivo evitando tener que loguearse siempre.
RF12	La pantalla principal debe mostrar la lista completa de hijos (**RF01**) y un menú con los accesos a "Información" y "Mensajes".
RF13	La pantalla de faltas (**RF05**) dispondrá de una vista de calendario y una vista detallada ordenada por fecha.
RF14	Cada hijo (RF01) puede estar matriculado más de una vez, en el mismo centro o en centro distintos. Las consultas de cada hijo deben ir identificadas con su id de matrícula.

Table 1 - Requisitos funcionales

3.2 Requisitos no funcionales

Los requisitos no funcionales del sistema son aquellos que describen cualidades o criterios que sirven para juzgar el funcionamiento general del sistema, en lugar de las funcionalidades específicas. Estos requisitos comprenden las características de seguridad, disponibilidad, accesibilidad, escalabilidad, extensibilidad, tolerancia a fallos, usabilidad, compatibilidad, etc.

Referencia	Descripción del requisito	Categoría
RNF01	El servidor debe discriminar a los usuarios logueados por el perfil de usuario en el sistema, permitiendo acceder a los recursos a los usuarios con perfil "Padre/Madre".	Seguridad
RNF02	La aplicación móvil debe ser compatible con los sistemas operativos móviles Android y iOS	Compatibilidad
RNF03	La carga de la información desde el servidor a la aplicación móvil debe ser rápida	Rendimiento
RNF04	El servidor deberá permitir una fácil evolución en el futuro para añadir nuevos recursos y el acceso a ellos a distintos perfiles, extendiendo el requisito (**RFN01**).	Extensibilidad, Seguridad
RNF05	La aplicación debe seguir una diseño modular para facilitar su mantenimiento y extension	Mantenibilidad, Extensibilidad
RFN06	El sistema debe diseñarse teniendo en cuenta las evoluciones planteadas en Yedra en cuanto a la creación de múltiples matriculas por alumno.	Portabilidad, Extensibilidad, Estabilidad

Table 2 - Requisitos no funcionales

3.3 Diagrama de casos de uso

A continuación se representará el diagrama de casos de uso de los requisitos para nuestro sistema, en todos ellos el actor corresponde a un usuario con perfil "Padre/Madre" en el sistema Yedra.

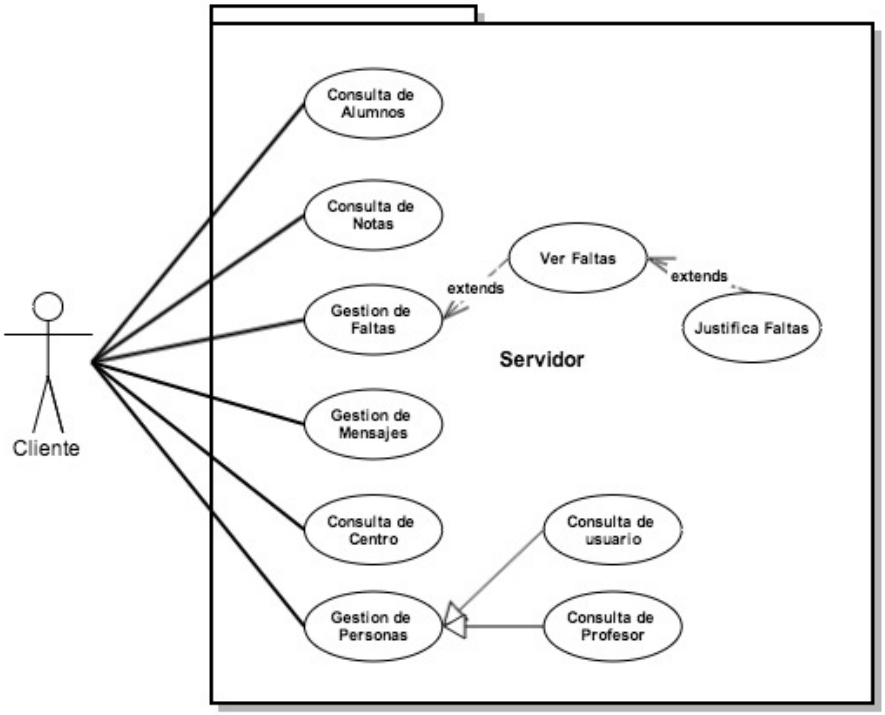

Figure 8 - Diagrama de Casos de Uso general

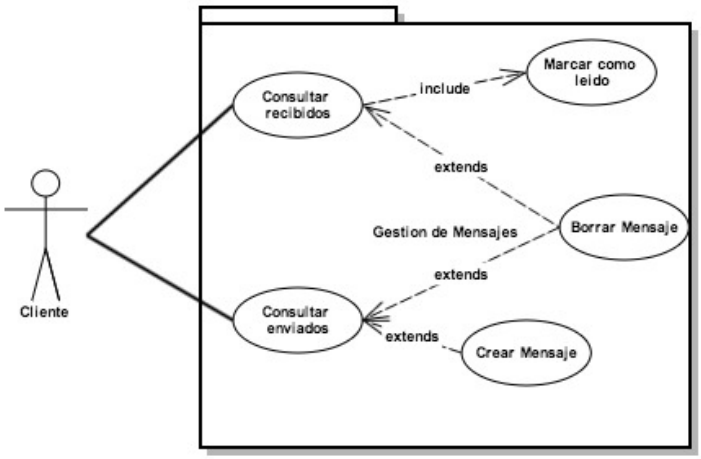

Figure 9 - Diagrama de Caso de uso Mensajes

Estos casos de uso poseen un diseño que sigue los mismos patrones una vez que el usuario realiza a petición de la información en la aplicación. No obstante se describen los casos de uso menos triviales y como se obtiene la información desde que el usuario realiza la petición hasta que es recibida y presentada en la pantalla:

Plantilla	
ID	P01
Nombre	Consulta de Alumnos
Descripción	El usuario abre la aplicación móvil y carga la información de sus hijos
Actores	Usuario con perfil "Padre/Madre"
PRE:	-
Flujo principal	1. El usuario introduce sus credenciales de acceso en la pantalla de login. 2. La aplicación móvil manda una petición para obtenerla información de los hijos del usuario 3. El servidor comprueba que las credenciales son correctas y asigna al usuario los perfiles que posee en la base de datos 4. Si el usuario tiene perfil "Padre/Madre" se devuelve un mensaje a la aplicación móvil con los datos de los hijos con matrículas en el año actual. 5. Se envía a la aplicación la información de usuario, que queda guardada en el dispositivo. 6. La interfaz cambia a la vista de inicio y presenta la lista de los hijos y las matriculas de cada uno de ellos.
Flujo alternativo	Flujo alternativo: El usuario ya ha introducido sus credenciales previamente y han sido validadas: 1. En lugar de la pantalla de login, se pasa directamente a la de presentación de la lista de los hijos.

Table 3 - Casos de uso: Plantilla de Consulta de alumnos

Plantilla	
ID	P02
Nombre	Justifica Faltas
Descripción	El usuario selecciona una de las faltas de asistencia injustificadas y procede a justificarla seleccionando un motivo y añadiendo un comentario
Actores	Usuario con perfil "Padre/Madre"
PRE:	El usuario ha consultado las faltas de una de las matriculas de uno sus hijos.
Flujo principal	1. El usuario selecciona la falta injustificada que quiere consultar en detalle. 2. Se abre una nueva pantalla donde se presenta el detalle de la falta incluyendo fecha, tramo horario, nombre de la asignatura y un selector de motivo de falta. 3. El usuario selecciona el motivo y añade un comentario y pulsa aceptar. 4. La aplicación móvil envía una petición al servidor indicando el id, motivo y comentario de la falta a justificar. 5. El servidor crea una nueva justificación y devuelve una respuesta al servidor.

Table 4 - Casos de uso: Plantilla de Justifica Faltas

Plantilla	
ID	
Nombre	Crear mensaje
Descripción	El usuario manda un mensaje al tutor de uno de los hijos
Actores	Usuario con perfil "Padre/Madre"
PRE:	El usuario ha consultado las faltas o notas de una de las matriculas de uno sus hijos.
Flujo principal	1. El usuario pulsa el botón de enviar mensaje y se muestra la pantalla. 2. Se realiza una petición al servidor para obtener la información del tutor de la matricula asociada al hijo. 3. El usuario escribe el mensaje y pulsa enviar. 4. Se envía al servidor una petición de creación de mensaje con un asunto predefinido, el cuerpo de mensaje y el id del tutor como destinatario. 5. El servidor realiza una consulta a la BD para obtener el id del usuario que envía el mensaje e incluirlo como remitente. 6. El servidor inserta el mensaje en la BD y devuelve la respuesta a la aplicación móvil. 7. La aplicación muestra la confirmación del envío.

Table 5 - Casos de uso: Plantilla de Crear Mensaje

4 Diseño del sistema

Para la descripción del diseño del sistema se ha seguido la certificación del estándar ISO 12207 (Standardization), que define distintos tipos de diseño de software, como son el diseño arquitectónico y el diseño detallado.

4.1 Diseño Arquitectónico

El diseño arquitectónico representa el más alto nivel de abstracción en la estructura de un sistema. El diseño arquitectónico escogido se basa en una arquitectura cliente-servidor en el cual la aplicación móvil realizará las peticiones a un programa que actuará como servidor, ésta es una separación lógica que ejecutará el programa servidor en una máquina centralizada y las distintas aplicaciones móviles en los terminales de los usuarios.

4.1.1 Diseño del Servidor

Internamente el diseño arquitectónico del servidor se basa en una arquitectura de tres capas, donde existe una separación lógica de la capa de acceso a datos, la capa de negocio y la capa de presentación de datos. Este diseño permite realizar cambios internos en cada una de las capas sin que el resto de ellas se vean afectadas.

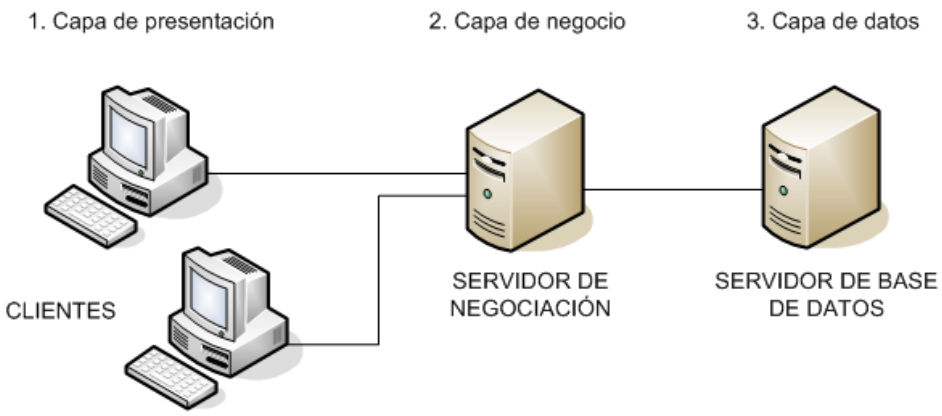

Figure 10 - Arquitectura de tres capas

Capa de presentación

- Se comunica únicamente con la capa de negocio.
- Define una API que recoge las peticiones del cliente y crea los mensajes de respuesta.
- Realiza el filtrado de las peticiones de usuario de errores de formato.
- Representa cada recurso como un objeto independiente que puede ser creado, consultado, modificado o borrado.

Capa de negocio

- Representa la mayor parte del programa, donde se realizan las principales operaciones con los datos.
- Se comunica con la capa de presentación, la cual recibe las solicitudes del cliente, y devuelve los resultados a ésta para enviar la respuesta de la petición.
- Se comunica con la capa de persistencia o datos, para obtener o insertar datos a través del gestor de la base de datos.

Capa de persistencia o de datos

- Representa el manejo y almacenamiento de datos y está compuesta por uno o varios gestores de bases de datos.
- Solo es accedida por la capa de negocio del programa.
- Facilita el acceso a la base de datos por medio de frameworks que manejan la apertura y cierre de conexiones, cachean los datos y convierten los datos en objetos fácilmente manejables.

Haciendo uso de esta arquitectura en el servidor permitimos una fácil evolución del proyecto y nos permite planificar el esfuerzo de desarrollo de cada nuevo recurso que añadiremos a la API. De este modo una vez definido un nuevo recurso al que el usuario tendrá acceso, definiremos en la capa de presentación las operaciones CRUD (Create, Read, Update, Delete) del recurso que estarán disponibles para el usuario y las respuestas del servidor a cada una; a continuación en la capa de negocio crearemos las operaciones de manejo de datos necesarias para proporcionar las respuestas esperadas y en la capa de persistencia crearemos las consultas a la base de datos para obtener o crear los datos requeridos.

4.1.2 Diseño de la aplicación móvil

El diseño arquitectónico de la aplicación móvil se basa en una arquitectura del tipo Modelo-Vista-Controlador (Fling, 2009), que permite la separación de la interfaz gráfica de usuario, los datos y la lógica de negocio y la gestión de eventos y comunicaciones.

Este diseño permite separar las vistas del manejo de los eventos provocados por el usuario y los datos que se representaran en la pantalla.

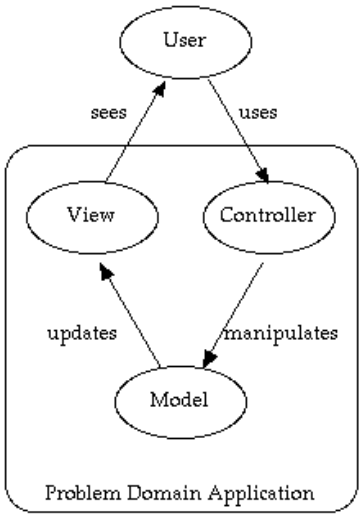

Figure 11 - Arquitectura MVC

Modelo

Representa los objetos y las estructuras de datos que se manejan en la aplicación y se muestran en la interfaz gráfica. El controlador es el encargado de modificar los datos presentes en el modelo.

Controlador

Responde a los eventos generados por el usuario o al aplicación, generalmente modificando las vistas o el modelo. Es el encargado de generar las peticiones de información al modelo o de realizar peticiones de red al servidor.

Vista

Representa los distintos modelos en las pantalla presentando una interfaz gráfica al usuario, los eventos generados en la vista son manejados por el controlador.

4.2 Diseño detallado

Una vez definido el diseño arquitectónico de la aplicación debemos adaptar los requerimientos a la arquitectura dada. Para ello, el desarrollo del sistema comienza con el diseño en detalle del servidor de la aplicación, el cual debe proporcionar la lógica adecuada para soportar las peticiones de datos de la aplicación móvil y los requerimientos de seguridad del sistema.

4.2.1 Diseño del servidor

Como vimos en el apartado anterior, la arquitectura del servidor se divide en tres capas lógicas que separan las distintas funcionalidades del sistema. A continuación se explicará en detalle como los requisitos del proyecto se integran en la arquitectura diseñada y en la infraestructura de las bases de datos de Yedra.

Anteriormente vimos como el sistema Yedra se apoya sobre tres bases de datos diferentes, de las cuales dos de ellas serán las que necesitaremos conectar para proporcionar las credenciales de acceso y los datos al usuario en este proyecto. Esta conexión se realizará mediante la capa de persistencia o capa de datos utilizando el framework MyBatis, que nos permitirá configurar la conexión y definir los objetos que mapearemos de la base de datos.

En el siguiente diagrama vemos en detalle cómo se integran las diferentes capas y módulos del sistema:

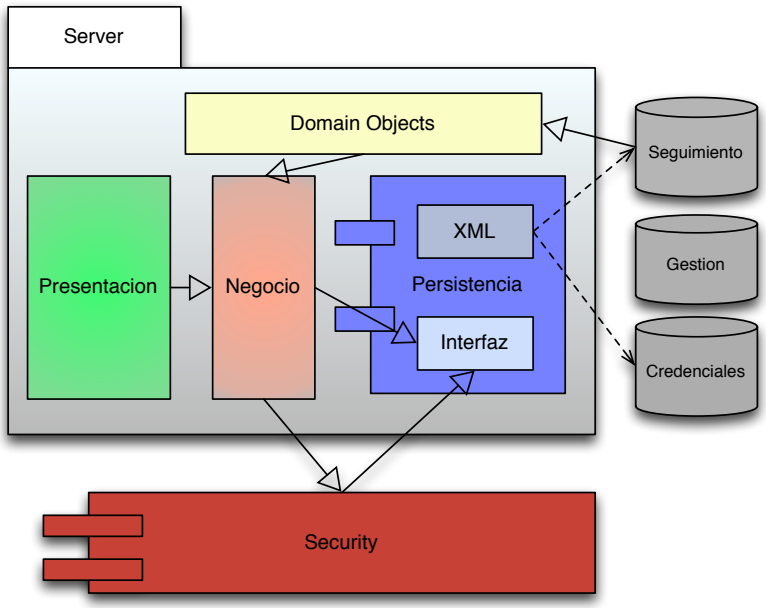

Figure 12 - Diseño del servidor

En base a los requisitos del proyecto se han diseñado una serie de recursos que definen la API de llamadas del servidor, estos recursos engloban las distintas entidades que el usuario podrá consultar o modificar en la base de datos a través del servidor. Cada uno de los recursos, siguiendo las directrices de diseño de una arquitectura REST, serán visible por un patrón de URL definido, sobre el cual se podrán mandar desde la aplicación móvil peticiones para crear, consultar, modificar o borrar dependiendo de los requisitos dados.

Los recursos que componen la API y las operaciones de cada uno son los siguientes:

Recurso	URL	Peticiones	Respuestas
Tutor	/user	GET	HTTP 200
Alumno	/alumno	GET	HTTP 200
Profesor	/matricula/{matricula}/profesor	GET	HTTP 200 HTTP 404
Centro	/matricula/{matricula}/centro	GET	HTTP 200 HTTP 404
Notas	/matricula/{matricula}/notas	GET	HTTP 200 HTTP 404
Faltas	/matricula/{matricula}/faltas	GET POST	HTTP 200 HTTP 201 HTTP 403 HTTP 404
Mensajes	/mensajes/recibidos /mensajes/recibidos/{id} /mensajes/enviados /mensajes/enviados/{id}	GET POST PUT DELETE	HTTP 200 HTTP 201 HTTP 204 HTTP 404

Table 6 - Tabla de recursos

Una vez definido los recursos que integrarán nuestra API, entraremos en detalle a definir los atributos presentes en cada recurso, los parámetros que necesitan y las posibles respuestas que entrega el servidor en cada caso.

Tutor

El recurso Tutor corresponde al usuario de la aplicación. Dentro del sistema Yedra es definido como tutor cualquier usuario con perfil "Padre/Madre" que puede corresponder también al tutor legal de un menor. Cada Tutor es representado por un objeto de tipo *Persona* que es identificado mediante un id llamado *x_usuario*, el cual corresponde al identificador de usuario en la tabla *TLUSUARIOS* de la base de datos *SEGEDU*.

El objeto Persona guardará los resultados de las consultas a la base de datos para obtener los datos de un usuario, que contendrán el nombre, apellidos, dni, dirección, teléfono y correo electrónico.

Para realizar la búsqueda de los datos de usuario se utilizará el nombre de usuario que es enviado con las credenciales en cada petición. Como vemos en la tabla anterior, al realizar una petición **HTTP GET - yedra/user** el servidor responde con **HTTP 200 OK** y un string JSON conteniendo los datos como vemos en el siguiente ejemplo:


```
"persona": {
   "apellido1": "Echevarría",
   "apellido2": "Revuelta",
   "c_numide": "13909939E",
   "c_postal": 39300,
   "calle": "José María Cabañas",
   "email": null,
   "escalera": "1º",
   "letra": null,
   "localidad": "Torres",
   "municipio": "Torrelavega",
   "nombre": "Ana Belén",
   "numCalle": 35,
   "piso": "A",
   "provincia": "Cantabria",
   "telefono": 625586155,
   "x_usuario": 500359033
}
```

Alumno

El recurso Alumno representa uno o varios hijos del usuario. En el sistema solo permitiremos al usuario retornar información sobre los hijos que estén matriculados en algún centro en el curso actual.

El objeto Alumno guardará los datos leídos en la consulta y contendrán el identificador x_alumno, el nombre, apellidos y la imagen del alumno almacenada. También una lista de una o varias matriculas del curso actual que contiene el identificador x_matricula, el nombre del centro, el nombre abreviado del curso, el nombre completo del curso y el grupo del mismo. Existen casos, como alumnos de Formación Profesional, donde éstos pueden matricularse de asignaturas de varias modalidades y por lo tanto poseen más de una matrícula en el curso actual. También se da el caso de la evolución planteada en el sistema, que integrará conservatorios y Escuelas de idiomas y por tanto los alumnos tendrán más de una matrícula. Por este motivo, para los recursos de notas, faltas, profesor y centro se utilizará el identificador de matrícula en lugar del identificador de alumno, pues este puede tener más de una.

Para realizar la búsqueda de la información de los hijos del usuario se utilizará al igual que en el caso anterior el nombre de usuario enviado con las credenciales en cada petición. En la tabla anterior se indica que al enviar una petición **HTTP GET - yedra/alumno** el servidor responde con **HTTP 200 OK** y un string JSON conteniendo los datos como vemos en el siguiente ejemplo de un solo alumno con varias matriculas:


```
"alumno": [
   {
      "apellido1": "Klein",
      "apellido2": "Echevarría",
      "matriculas": [
         {
            "centro": "39012121 - CPR INF-PRI-SEC Centro nº 602503",
            "cursoActual": "4º E.S.O.",
            "grupo": "4ºD ESO",
            "x_matricula": 798907
         },
         {
            "centro": "39009468 - IES Centro nº 602432",
            "cursoActual": "1º de Bachillerato (Ciencias y Tecnología)",
            "grupo": "1C",
            "x_matricula": 904664
         },
         {
            "centro": "39009468 - IES Centro nº 602432",
            "cursoActual": "1º de Bachillerato (Ciencias y Tecnología)",
```

```
                    "grupo": "1C",
                    "x_matricula": 1026313
                }
            ],
            "nombre": "Marta",
            "x_alumno": 465668
        }
    ]
```


Profesor

El recurso Profesor obtiene los datos del profesor-tutor asignado a una matriculación. Cada hijo o usuario matriculado poseerá en el sistema un tutor para cada una de sus matrículas.

Este recurso es representado y guardado en un objeto Persona, pues los atributos coinciden con los del usuario que obtiene sus datos personales, sin embargo en este caso los datos personales del tutor se limitarán al identificador x_usuario, nombre, apellidos, email y teléfono. Estos datos permitirán al usuario enviar un mensaje directo al tutor de uno de sus hijos incluyendo su identificador como destinatario, al igual que comunicarse a través del email de trabajo o de su teléfono en caso de alguna urgencia.

Para realizar la búsqueda de la información del profesor se utilizará el identificador de matrícula y, al igual que en casos anteriores, el nombre de usuario enviado con las credenciales de cada petición. Este último es importante para restringir en la base de datos que quien realiza la petición tiene relación con el identificador de matrícula y por tanto es padre o tutor del alumno. Comprobando esta restricción en la consulta de la base de datos aumentará el rendimiento y disminuirá el tráfico de red y peticiones a la base de datos.

En la tabla anterior se indica que al enviar una petición **HTTP GET - yedra/matricula/{matricula}/profesor** el servidor lee el identificador {matricula} incluido en la URL para realizar la búsqueda y responde con **HTTP 404 NOT FOUND** si no encuentra dicho profesor, o en caso contrario **HTTP 200 OK** y un string JSON conteniendo la información:

```
"persona": {
   "apellido1": "Martín",
   "apellido2": "Lopez",
   "email": "maria.martin@educantabria.com",
   "nombre": "María",
   "telefono": 625345678,
   "x_usuario": 512398033
}
```


Centro

El recurso Centro obtiene la información del centro de estudios del hijo en base al identificador de matrícula pasado como parámetro. Dicha información es almacenada en el objeto Centro que contiene como atributos el identificador de centro en la base de datos, el nombre, la dirección, el correo y el teléfono de la secretaría de centro.

Para realizar la búsqueda se sigue el patrón explicado en el recurso anterior, utilizando el identificador de matrícula y el nombre de usuario como medida de seguridad. En la tabla se indica que al enviar una petición **HTTP GET - yedra/matricula/{matricula}/centro** el servidor lee el identificador {matricula} incluido en la URL para realizar la búsqueda y responde con **HTTP 404 NOT FOUND** si no encuentra el centro, o en caso contrario **HTTP 200 OK** y un string JSON conteniendo la información:


```
"centro": {
   "codigo": 39009468,
   "correo": "secretaria@iesmarquesdesantillana.com",
   "direccion": "AV.DE ESPAÑA,2 - Torrelavega",
   "nombre": "Centro nº 602432",
   "telefono": 942881600
}
```


Notas

El recurso Notas obtiene las calificaciones del hijo en el curso actual, en concreto las notas de la matricula asociada. Dichas calificaciones corresponden a las evaluaciones oficiales de cada curso, ya que los centros no tienen obligación de registrar los exámenes realizados durante el curso. El resultado de la consulta es guardado en una lista de objetos NotaAsignatura, que contienen atributos del nombre de la asignatura, la modalidad a la que pertenece, la convocatoria o evaluación a la que corresponde, la calificación,

el orden de convocatoria, el identificador de la nota en base de datos y un id de convocatoria final o extraordinaria.

Para realizar la búsqueda se sigue el patrón explicado en el recurso anterior, utilizando el identificador de matrícula y el nombre de usuario como medida de seguridad. En la tabla se indica que al enviar una petición **HTTP GET - yedra/matricula/{matricula}/notas** el servidor lee el identificador {matricula} incluido en la URL para realizar la búsqueda y responde con **HTTP 404 NOT FOUND** si no encuentra notas, o en caso contrario **HTTP 200 OK** y un string JSON conteniendo la información. En este ejemplo de dos asignaturas observamos que solo se ha calificado la primera evaluación:


```
"nota": [
    {
      "califica": 583,
      "convFinal": 0,
      "convocatoria": "1ª Evaluación",
      "materia": "Biología y geología",
      "nota": "8",
      "orden": 1,
      "tipo": "Modalidad"
    },
    {
      "califica": 0,
      "convFinal": 0,
      "convocatoria": "2ª Evaluación",
      "materia": "Biología y geología",
      "nota": null,
      "orden": 2,
      "tipo": "Modalidad"
    },
    {
      "califica": 0,
      "convFinal": 1628,
      "convocatoria": "Final Ordinaria",
      "materia": "Biología y geología",
      "nota": null,
      "orden": 3,
      "tipo": "Modalidad"
    },
    {
      "califica": 0,
      "convFinal": 1629,
      "convocatoria": "Final Extraordinaria",
      "materia": " Biología y geología ",
      "nota": null,
      "orden": 4,
      "tipo": "Modalidad"
```

```
        },
        {
            "califica": 587,
            "convFinal": 0,
            "convocatoria": "1ª Evaluación",
            "materia": "Ciencias para el mundo contemporáneo",
            "nota": "6",
            "orden": 1,
            "tipo": "Comunes"
        },
        {
            "califica": 0,
            "convFinal": 0,
            "convocatoria": "2ª Evaluación",
            "materia": "Ciencias para el mundo contemporáneo",
            "nota": null,
            "orden": 2,
            "tipo": "Comunes"
        },
        {
            "califica": 0,
            "convFinal": 1628,
            "convocatoria": "Final Ordinaria",
            "materia": "Ciencias para el mundo contemporáneo",
            "nota": null,
            "orden": 3,
            "tipo": "Comunes"
        },
        {
            "califica": 0,
            "convFinal": 1629,
            "convocatoria": "Final Extraordinaria",
            "materia": "Ciencias para el mundo contemporáneo",
            "nota": null,
            "orden": 4,
            "tipo": "Comunes"
        }
    ]
```


Faltas

El recurso Faltas obtiene las faltas de asistencia asociadas a una matriculación y permite al usuario crear una justificación. Las faltas están clasificadas en tres tipos, faltas injustificadas, faltas justificadas y retrasos. Estos últimos no son justificables en el sistema. Las faltas son representadas por un objeto Falta que contiene de atributos el identificador de la falta, el identificador de la justificación, el id de usuario que creó la justificación, la materia a la que pertenece, la fecha, el tramo horario, el tipo de falta, un

booleano si ha faltado el día completo, el motivo de justificación y la notificación correspondiente.

Para realizar la búsqueda se sigue el patrón explicado en el recurso anterior, utilizando el identificador de matrícula y el nombre de usuario como medida de seguridad. En la tabla anterior se indican las dos operaciones disponibles sobre el recurso, obtener todas las faltas y crear una nueva justificación.

Para obtener todas las faltas de asistencia asociadas a las matriculas se debe realizar una petición **HTTP GET - yedra/matricula/{matricula}/faltas** indicando el id de matrícula en la URL, de esta forma si encuentra alguna falta recibiremos una respuesta y un JSON con la lista de faltas como vemos en el siguiente ejemplo:


```
"falta": [
    {
        "diaCompleto": false,
        "fecha": "08/11/2012",
        "id": 47826205,
        "materia": "Matemáticas I",
        "motivo": 0,
        "notificacion": "Ha estado con gripe",
        "observacion": null,
        "tipo": "J",
        "tramo": "13:35 - 14:25",
        "usuarioCreacion": 0,
        "x_justifica": 0
    },
    {
        "diaCompleto": false,
        "fecha": "08/11/2012",
        "id": 47826007,
        "materia": "Ciencias para el mundo contemporáneo",
        "motivo": 0,
        "notificacion": null,
        "observacion": null,
        "tipo": "I",
        "tramo": "12:40 - 13:30",
        "usuarioCreacion": 0,
        "x_justifica": 0
    }
]
```


Para justificar una de las faltas se debe realizar una petición **HTTP POST - yedra/matricula/{matricula}/faltas** siguiendo las mismas condiciones. En el cuerpo de la petición se debe incluir un JSON con el identificador de la falta, el motivo de justificación y el contenido del campo observación con las

explicaciones del usuario. El servidor puede respondernos con **HTTP 404 NOT FOUND** si no encuentra la falta o el tipo no es "injustificada", **HTTP 403 FORBIDDEN** si la falta está asociada a otra matricula o **HTTP 201 CREATED** en caso de éxito. En el siguiente ejemplo se muestra un ejemplo de justificación enviada con la petición POST:

```
{
    "id":47826205,
    "motivo":2180,
    "observacion":"Ha estado con gripe"
}
```

Mensajes

El recurso mensaje permite la comunicación con los usuarios del sistema mediante un buzón de mensajes. Mediante el envío de las credenciales de usuario, éste puede obtener su lista de mensajes enviados y recibidos, obtener mensajes de forma individual a través del identificador del mensaje, marcar mensajes como leídos y borrar mensajes. Los mensajes son identificados por un objeto Mensaje que contiene de atributos el identificador del mensaje, del remitente, del destinatario o lista de destinatarios, el nombre del remitente y destinatario, la fecha de envío, el asunto, el contenido y un campo indicando si ha sido leído.

Para realizar la búsqueda se utilizará el nombre de usuario enviado con las credenciales, al igual que para añadir el identificador del remitente al crear un nuevo mensaje. De esta forma aseguramos que un usuario autentificado envía mensajes solo a su nombre y solo necesite enviar el identificador del remitente, el asunto y el cuerpo del mensaje.
Para obtener todos los mensajes enviados o recibidos se debe realizar una petición **HTTP GET - yedra/mensajes/enviados** o **HTTP GET - yedra/mensajes/recibidos** y recibiremos un JSON con la lista de mensajes. También podemos obtener un solo mensaje añadiendo un identificador a la URL. La lista retornada por el servidor sigue el siguiente patrón:

```
"mensaje": [
        {
            "asunto": "Examen de matematicas",
            "fechaEnvio": "08/11/2012",
            "leido": "S",
            "listaDestinatarios": [],
            "mensaje": "Prueba, cuerpo del mensaje",
            "nombreDestinatario": "David Muñoz",
            "nombreRemitente": "Rocio Gomez"
            "x_desmenusu": null,
            "x_mensaje": 886689,
            "x_usuario_destinatario": 50087886,
            "x_usuario_remitente": 500885335,
        },
        {
            "asunto": "Examen de matematicas",
            "fechaEnvio": "08/11/2012",
            "leido": "S",
            "listaDestinatarios": [],
            "mensaje": "Prueba, cuerpo del mensaje",
            "nombreDestinatario": "David Muñoz",
            "nombreRemitente": "Rocio Gomez"
            "x_desmenusu": null,
            "x_mensaje": 886689,
            "x_usuario_destinatario": 50087886,
            "x_usuario_remitente": 500885335,
        }
]
```


Para marcar un mensaje recibido como leído se debe realizar una petición **HTTP PUT - yedra/mensajes/recibidos/{id}** especificando el identificador del mensaje. Del mismo modo para borrar alguno de los mensajes se debe realizar una petición **HTTP DELETE - yedra/mensajes/enviados/{id}** o bien **HTTP DELETE - yedra/mensajes/recibidos/{id}** donde la respuesta del servidor en estos casos será **HTTP 204 NO CONTENT** en caso de éxito o **HTTP 404 NOT FOUND** si no se ha encontrado el mensaje indicado.

Para crear un mensaje nuevo se debe realizar una petición **HTTP CREATE - yedra/mensajes/enviados** e incluir en el cuerpo de la petición un JSON conteniendo el identificador del destinatario, el asunto y el cuerpo del mensaje siguiendo la siguiente estructura:

```
{
   "x_usuario_destinatario": 21477,
   "asunto": "PruebaCliente",
   "mensaje":" Este es un mensaje de prueba"
}
```

En caso de éxito el servidor retornará una respuesta **HTTP 201 CREATED** y el identificador del nuevo mensaje creado en la cabecera *Location.*

Seguridad

La seguridad en el sistema deberá interceptar las peticiones HTTP enviadas al servidor y decodificar y leer las credenciales (usuario y contraseña) incluidas en la cabecera *Authoritation*. Una vez obtenido el nombre de usuario el sistema deberá encargarse de buscar en la base de datos su contraseña, su perfil de acceso y el estado de su cuenta, dado que el usuario puede estar bloqueado o la cuenta o la contraseña pueden haber expirado. A continuación, si la cuenta está operativa y la contraseña es correcta le permite el acceso y se realiza la petición indicada.

El perfil de acceso del usuario debe permitir discriminar al tipo de usuario para acceder a los distintos recursos del sistema. En este caso solo un usuario con perfil "Padre/Madre" puede acceder a los recursos diseñados pero el diseño permite añadir permisos a nuevos perfiles o crear nuevos recursos para otros perfiles específicos.

A continuación se muestra el diagrama de clases del sistema, integrando cada recurso en la arquitectura del servidor:

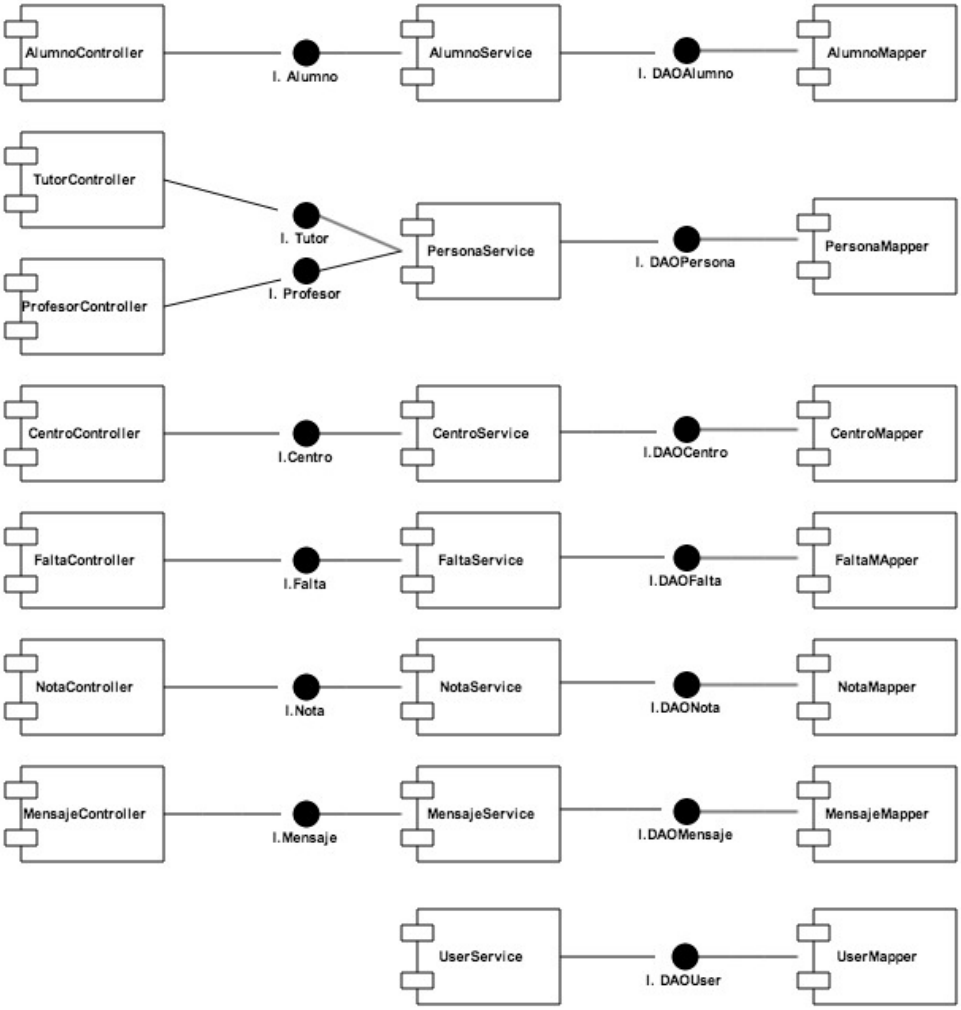

Figure 13 - Diagrama de componentes

4.2.2 Diseño de la aplicación móvil

Como vimos en el apartado del diseño de la arquitectura (ver índice 4.1.2), la aplicación móvil sigue un patrón Modelo-Vista-Controlador que separa la interfaz gráfica de la petición de datos y su almacenamiento. El diseño de la aplicación móvil parte del diseño de la interfaz gráfica y el flujo de controles, que permitirá obtener la información del servidor mediante llamadas y manejar las distintas vistas de la interfaz.

Para ello se han diseñado una serie de *Wireframes* que proporcionan una guía visual inicial de la estructura que diseñaremos en cada pantalla. Estos *Wireframes* definen el flujo de navegación y la estructura de datos y controles, donde se han definido pantallas para seleccionar alumnos, visualizar su información, sus notas y sus faltas; visualizar información del usuario, mensajes y envío mensajes.

A continuación vemos los *wireframes* de las pantallas más representativas:

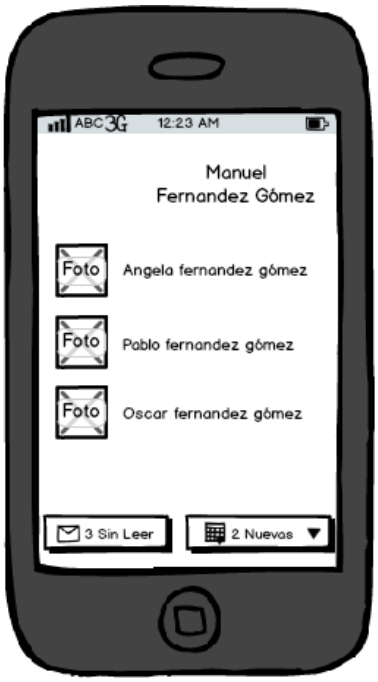

Figure 14 - Wireframe Informacion de alumno

Figure 15 - Wireframe Seleccion de Alumnos

La pantalla principal mostrará la lista de los hijos del usuario, pudiendo acceder a la información personal de cada hijo y mostrando accesos directos a sus notas y faltas.

Figure 16 - Wireframe Notas del alumno

Figure 17 - Wireframe Faltas de asistencia

La pantalla de notas mostrara la lista de notas ordenada por evaluaciones. En la pantalla de faltas de asistencia mostraremos las faltas ordenadas por fechas pudiendo verlas en detalle y justificar aquellas que sea posible.

La bandeja de mensajes mostrará diferentes pantallas para mensajes enviados y recibidos, indicando los que se encuentran sin leer. Por ultimo en otra pantalla mostraremos la información de usuario almacenada en el sistema.

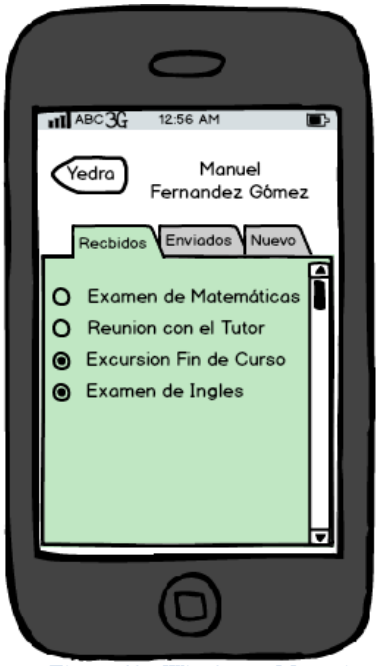

Figure 18 - Wireframe Mensajes

Figure 19 - Wireframe Informacion de usuario

5 Implementación

Una vez diseñado en detalle todo el sistema iniciaremos la implementación del mismo con las tecnologías elegidas. Dicha implementación se divide en la implementación del servidor y, una vez realizada y probada, se completa con la implementación de la aplicación móvil.

5.1 Implementación del servidor

La implementación del servidor como se indicó anteriormente (ver índice 2.1.2), se ha llevado a cabo haciendo uso de los frameworks Spring y MyBatis. Dicho sistema se ha implementado siguiendo el diseño de los recursos de la API expuestos en el apartado anterior, por el cual se implementan las funcionalidades de cada recurso en cada una de las capas y se realizan las pruebas unitarias antes de continuar con el siguiente, pudiendo dividir y estimar la implementación de forma precisa en cortos periodos de trabajo.

Dado que la estructura de los recursos es la misma en cada capa, la implementación se enfocará en describir como se diseña cada capa del sistema para los recursos dados, poniendo como ejemplo uno o varios de ellos.

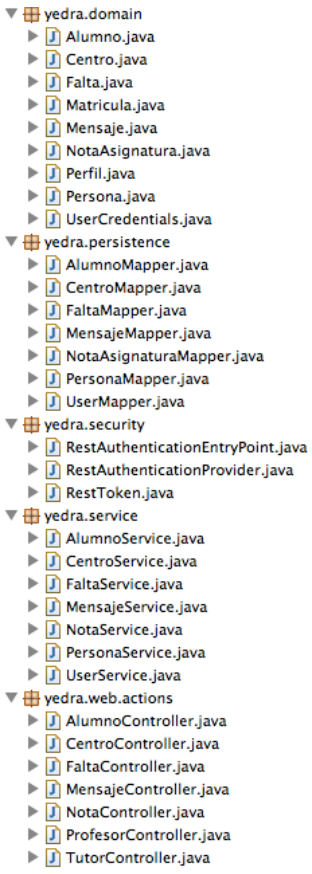

Figure 20 - Estructura de la implementación

5.1.1 Implementación de la API REST

La implementación de la capa de presentación del servidor, o API de la aplicación en este caso se lleva a cabo por medio de las clases *@Controller*. Estas definen las URL que acceden a cada recurso y los métodos que interceptan cada tipo de llamada HTTP. Dichos métodos se encargarán de leer los parámetros de la URL o del cuerpo de la petición, llamar a los métodos correspondientes en la capa de servicio y proporcionar la respuesta adecuada, manejando también los posibles errores (Mak, Long, & Rubio, 2010).

El controlador del recurso que devuelve la lista de alumnos del usuario es el más básico, como vemos a continuación la URL es definida por el *@RequestMapping* y el método *getAlumno* manejará las peticiones HTTP GET a ese recurso, obteniendo la lista de alumnos de la capa de servicio y creando una respuesta con las cabeceras de contenido, la información en formato JSON y la respuesta HTTP 200 OK.

```
@Controller
@RequestMapping(value="/alumno")
public class AlumnoController {

    @Autowired
    private AlumnoService alumnoService;

    @RequestMapping(method=RequestMethod.GET)
    public @ResponseBody ResponseEntity<String> getAlumno() {

        List<Alumno> alumnos = alumnoService.getAlumnos();
        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-Type", "application/json; charset=utf-8");
        return new ResponseEntity<String>(Alumno.toJsonArray(alumnos),headers,HttpStatus.OK);
    }

}
```

Figure 21 - Controlador de Alumno

Los controladores de los recursos Nota, Falta, Centro y Profesor son implementados siguiendo la misma estructura, sin embargo necesitan el identificador de la matrícula del alumno asociada a dichos recursos. Para ello definimos la URL con el identificador de matrícula y el recurso "notas" en el *@RequestMapping* del método, que manejará las peticiones GET y recibe como parámetro el id de matrícula. Este método obtiene la lista de notas de la capa de servicio y devuelve una respuesta creando las cabeceras de contenido, la lista de notas en formato JSON y la respuesta de éxito.

```java
@Controller
public class NotaController {

    @Autowired
    private NotaService notaService;

     * Devuelve las notas del alumno recibiendo su matricula en la peticion
    @RequestMapping(value = "/matricula/{matricula}/notas", method = RequestMethod.GET)
    public @ResponseBody ResponseEntity<String> getNotas(@PathVariable("matricula") int matricula) {

        List<NotaAsignatura> notas = notaService.getNotas(matricula);

        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-Type", "application/json; charset=utf-8");
        if(notas != null) {
            return new ResponseEntity<String>(NotaAsignatura.toJsonArray(notas),headers, HttpStatus.OK);
        }
        return new ResponseEntity<String>(headers, HttpStatus.NOT_FOUND);
    }
}
```

Figure 22 - Controlador de Notas

El controlador de Falta implementa otro método para justificar faltas de asistencia, este intercepta las peticiones con el mismo patrón de URL pero sólo las que posean un método HTTP POST. Este método requiere de un objeto de tipo falta en el cuerpo de mensaje conteniendo el id de la falta a justificar, el motivo y el comentario de justificación. A continuación se comprueba que la falta existe y está asociada a la matricula indicada, retornando un error en caso contrario. Por último le envía el objeto a la capa de servicio y retorna la respuesta HTTP CREATED.

```java
@RequestMapping(value = "/matricula/{matricula}/faltas", method = RequestMethod.POST, consumes = {"application/json"})
public @ResponseBody ResponseEntity<String> justificaFalta(@PathVariable("matricula") int matricula, @RequestBody String json)

    // Transforma el json recibido en un objeto de Falta
    Falta falta = Falta.fromJsonToFalta(json);

    HttpHeaders headers = new HttpHeaders();
    headers.add("Content-Type", "application/json; charset=utf-8");

    // Comprueba que la falta recibida existe en DB y esta asociada a esa matricula
    int idFalta = falta.getId();
    int idMatriculaDeFalta = 0;
    idMatriculaDeFalta = faltaService.matriculaDeFalta(idFalta);

    // Si la falta a justificar no se encuentra en DB retornará HTTP 404
    if (idMatriculaDeFalta == 0) {
        return new ResponseEntity<String>(headers, HttpStatus.NOT_FOUND);
    }

    // Si esta asociada a otra matricula, retornará HTTP 403
    if (idMatriculaDeFalta != matricula) {
        return new ResponseEntity<String>(headers, HttpStatus.FORBIDDEN);
    }

    // Crea una nueva entrada en DB justificando la falta solicitada
    int rows = faltaService.justificaFalta(falta);

    // Si hay exito retorna 201, si la falta ya estuviera justificada retorna HTTP 404
    if (rows > 1) {
        return new ResponseEntity<String>(headers, HttpStatus.CREATED);
    }
    return new ResponseEntity<String>(headers, HttpStatus.NOT_FOUND);
}
```

Figure 23 - Controlador de Falta

El controlador de mensajes implementa dicho recurso interceptando las peticiones a la URL "mensajes", con métodos interceptando mediante el *@RequestMapping* los "enviados" y "recibidos". La estructura para obtener los mensajes enviados o recibidos por el usuario es igual que en los anteriores casos, sin embargo este controlador implementa métodos para borrar mensajes interceptando las peticiones HTTP DELETE, un método para marcar mensajes recibidos como leídos mediante HTTP PUT, y otro para crear mensajes mediante HTTP POST.

```
 * Retorna los mensajes recibidos al usuario identificado.
@RequestMapping(value="/recibidos", method=RequestMethod.GET)
public @ResponseBody ResponseEntity<String> getMensajesRecibidos() {
    List<Mensaje> mensajes = mensajeService.getMensajesRecibidos();

    HttpHeaders headers = new HttpHeaders();
    headers.add("Content-Type", "application/json; charset=utf-8");
    if (mensajes != null) {
        return new ResponseEntity<String>(Mensaje.toJsonArray(mensajes),
                headers, HttpStatus.OK);
    }
    return new ResponseEntity<String>(headers, HttpStatus.NOT_FOUND);
}

 * Cambia el mensaje recibido a "leido".
@RequestMapping(value="/recibidos/{id}", method=RequestMethod.PUT)
public @ResponseBody ResponseEntity<String> updateMensajesLeidos(@PathVariable("id") int mensaje) {

    int rows = 0;
    rows = mensajeService.updateMensajeLeidos(mensaje);

    HttpHeaders headers = new HttpHeaders();
    headers.add("Content-Type", "application/json; charset=utf-8");

    if (rows > 0) {
        return new ResponseEntity<String>(headers, HttpStatus.NO_CONTENT);
    }
    return new ResponseEntity<String>(headers, HttpStatus.NOT_FOUND);

}
```

Figure 24 - Controlador de mensajes recibidos

En la imagen anterior observamos el método que devuelve la lista de mensajes recibidos al interceptar el método GET y el método que marca un mensaje como leído. Este intercepta el método PUT y tras leer el id del mensaje recibido en la URL pasa este identificador a la capa de servicio y devuelve la respuesta HTTP 204 NO CONTENT.

```
 * Borra el mensaje enviado por el usuario, identificado mediante la id del mensaje
@RequestMapping(value="/enviados/{id}", method=RequestMethod.DELETE)
public @ResponseBody ResponseEntity<String> deleteMensajeEnviado(@PathVariable("id") int mensaje) {

    int rows = 0;
    rows = mensajeService.deleteMensajeEnviado(mensaje);

    HttpHeaders headers = new HttpHeaders();
    headers.add("Content-Type", "application/json; charset=utf-8");

    if (rows > 0) {
        return new ResponseEntity<String>(headers, HttpStatus.NO_CONTENT);
    }
    return new ResponseEntity<String>(headers, HttpStatus.NOT_FOUND);
}

@RequestMapping(value="/enviados", method=RequestMethod.POST, consumes = {"application/json"})
public @ResponseBody ResponseEntity<String> crearMensaje(@RequestBody String json) {

    // Transforma el json en un objeto de la clase mensaje
    Mensaje mensaje = Mensaje.fromJsonToMensaje(json);

    int id = 0;
    id = mensajeService.createMensaje(mensaje);

    // Retorna el id del mensaje creado en la cabecera LOCATION, el codigo de buenas practicas nos recomeinta
    // retornar el path absoluto del recurso, en este caso retornamos el relativo (id) para poder comprobarlo
    // mediante la llamada a enviados(id)
    HttpHeaders headers = new HttpHeaders();
    headers.add("Content-Type", "application/json; charset=utf-8");
    headers.add("Location", ""+id);

    if (id != 0) {
        return new ResponseEntity<String>(headers, HttpStatus.CREATED);
    }
    return new ResponseEntity<String>(headers, HttpStatus.NOT_FOUND);

}
```

Figure 25 - Controlador de mensajes enviados

En esta imagen vemos el método encargado de eliminar un mensaje enviado, cuya estructura es la misma que en los métodos de los mensajes recibidos o en el que marca un mensaje como leído. El método que se encarga de la creación de mensaje intercepta las peticiones POST y recibe como parámetro un objeto "mensaje" en formato JSON incluido en el cuerpo de la petición. Una vez construido el objeto es enviado a la capa de servicio y responde con HTTP 201 CREATED y el identificador del nuevo mensaje creado en la cabecera *Location*.

5.1.2 Implementación de la capa de negocio

La implementación de la capa de negocio o servicio se lleva a cabo por medio de las clases etiquetadas como *@Service*, que sirven de enlace entre la API y la capa de persistencia. En esta capa se encarga de manejar los parámetros de seguridad y llamar a las clases de la capa de datos.

Al igual que en la API, la capa de negocio se divide en clases que corresponden a cada recurso, añadiendo además otra clase encargada de manejar las credenciales de usuario y los perfiles de acceso. Estas clases siguen una misma estructura, obteniendo el identificador del usuario que realiza la consulta del contexto de la

petición mediante la clase *SecurityContextHolder*. En este punto se ha comprobado que las credenciales de usuario son correctas y este identificador es pasado como parámetro a la capa de datos y utilizado para obtener los recursos asociados al usuario o para comprobar que los identificadores de matrícula recibidos pertenecen a uno de los hijos del usuario.

```
@Service
public class AlumnoService {

    @Autowired
    private AlumnoMapper alumnoMapper;

    @Transactional
    public List<Alumno> getAlumnos() {
        // Obtiene el usuario logueado que realiza la peticion
        UserCredentials user = (UserCredentials) SecurityContextHolder
                .getContext().getAuthentication().getPrincipal();

        String idUsuario = user.getOid();
        return alumnoMapper.getAlumnosByUsuario(idUsuario);
    }

}
```

Figure 26 - Servicio del recurso Notas

En la imagen anterior observamos la implementación del servicio del recurso "Alumno", donde se obtiene el usuario "logueado" y se envía este identificador a la capa de datos.

```
@Service
public class PersonaService {

    @Autowired
    private PersonaMapper personaMapper;

    @Transactional
    public Persona getTutor() {
        // Obtiene el usuario logueado que realiza la peticion
        UserCredentials user = (UserCredentials) SecurityContextHolder
                .getContext().getAuthentication().getPrincipal();
        String idUsuario = user.getOid();
        return personaMapper.getTutor(idUsuario);
    }

    @Transactional
    public Persona getProfesorByMatricula(int matricula) {
        // Obtiene el usuario logueado que realiza la peticion
        UserCredentials user = (UserCredentials) SecurityContextHolder
                .getContext().getAuthentication().getPrincipal();
        String idUsuario = user.getOid();
        return personaMapper.getProfesor(idUsuario, matricula);
    }
}
```

Figure 27 - Servicio del recurso Persona

En la imagen anterior observamos la implementación del servicio que maneja información del objeto "Persona", esto es, tanto profesores como usuarios. En ambos se utiliza también el identificador de usuario, bien sea para obtener su información o para comprobar que la matricula recibida corresponde a un hijo del usuario.

```
@Transactional
public int matriculaDeFalta(int idFalta) {
    return faltaMapper.getMatriculaDeFalta(idFalta);
}

@Transactional
public int justificaFalta(Falta falta) {
    String idUsuario = getIdUsuario();

    // Obtiene los datos del usuario de DB y lo incluye como creador de la justificacion
    Persona usuarioCreacion = personaMapper.getTutor(idUsuario);
    int x_usuario = usuarioCreacion.getX_usuario();
    falta.setUsuarioCreacion(x_usuario);

    // inserta la justificacion en DB
    int rows =  faltaMapper.justificaFalta(falta);
    rows += faltaMapper.updateFaltaJustificada(falta.getId());
    return rows;
}

@Transactional
private String getIdUsuario() {
    // Obtiene el usuario logueado que realiza la peticion
    UserCredentials user = (UserCredentials) SecurityContextHolder
            .getContext().getAuthentication().getPrincipal();
    return user.getOid();
}
```

Figure 28 - Servicio del recurso Falta

En este caso se muestra la implementación de algunos métodos del servicio de Falta, que proporciona un método para comprobar que cuando se recibe una justificación para una falta, se pueda determinar que ésta pertenece a la matricula indicada y no se puedan producir justificaciones en faltas ajenas. El método que da servicio a la justificación, se encarga de incluir en la justificación el usuario que la ha creado usando las credenciales validadas, así como cambiar el estado de la falta a "justificada".

```java
@Transactional
public int createMensaje(Mensaje mensaje) {
    String idUsuario = getIdUsuario();

    // Obtiene los datos del remitente de DB y lo incluye como remitente
    Persona remitente = personaMapper.getTutor(idUsuario);
    int x_usuario_remitente = remitente.getX_usuario();
    mensaje.setX_usuario_remitente(x_usuario_remitente);

    // inserta el mensaje en DB
    mensajeMapper.insertMensaje(mensaje);
    mensajeMapper.insertDestinatarioMensaje(mensaje);

    // retorna la id del mensaje creado
    return mensaje.getX_mensaje();
}
```

Figure 29 - Servicio del recurso Mensajes - Método creaMensaje

Esta misma técnica es usada en el servicio del recurso Mensaje, al crear un nuevo mensaje se obtiene de la capa de datos un identificador interno de la persona que envía la petición y es incluido como remitente en el objeto enviado a la capa de datos. Posteriormente se hacen dos llamadas para insertar un nuevo mensaje y para asociarlo con el destinatario, finalmente se devuelve el identificador del mensaje creado a la capa de presentación.

5.1.3 Implementación de la capa de acceso a datos

La capa de acceso a datos, o capa de persistencia, es la encargada de manejar la conexión con la base de datos, definir las consultas y mapear los resultados en objetos POJO (*Plain Old Java Object*) que son enviados a la capa de servicio. Esta capa está implementada con el framework MyBatis que, como se explicó anteriormente (ver índice 2.1.2.1) se compone de interfaces Java que son llamadas por la capa de servicio y archivos XML que definen mapeados de resultados a objetos y consultas SQL.

Las interfaces definen el nombre de la consulta, los parámetros y el objeto retornado, como es el caso de la intefaz *AlumnoMapper*:

```java
public interface AlumnoMapper {

    List<Alumno> getAlumnosByUsuario(@Param("idUsuario") String idUsuario);

}
```

Figure 30 - Interfaz AlumnoMapper

Los archivos XML poseen un link a su interfaz y definen las consultas y como mapear los resultados al objeto:

```xml
<mapper namespace="yedra.persistence.AlumnoMapper">

    <cache/>

    <resultMap  id="AlumnoMap" type="Alumno">
        <id     property="x_alumno" column="X_ALUMNO"/>
        <result property="nombre" column="nombre"/>
        <result property="apellido1" column="apellido1"/>
        <result property="apellido2" column="apellido2"/>
        <collection property="matriculas" ofType="Matricula">
            <id property="x_matricula" column="matricula"/>
            <result property="centro" column="centro"/>
            <result property="cursoActual" column="cursoActual"/>
            <result property="cursoAbreviado" column="cursoAbreviado"/>
            <result property="grupo" column="grupo"/>
        </collection>
    </resultMap>
```

Figure 31 - Archivo XML de la consulta de Alumno

```xml
    <select id="getAlumnosByUsuario" resultMap="AlumnoMap">
        SELECT
            alu.T_NOMBRE        as nombre,
            alu.X_ALUMNO,
            mat.x_matricula     as matricula,
            alu.T_APELLIDO1     as apellido1,
            alu.T_APELLIDO2     as apellido2,
            tlf_seldatcen('DEN', mat.X_CENTRO) as centro,
            ofe.D_OFERTAMATRIG  as cursoActual,
            ofe.T_ABREVIATURA   as cursoAbreviado,
            uni.T_NOMBRE        as grupo
        FROM
            TLALUMNOS alu,
            TLFAMILIASALU falu,
            TLTUTORES tut,
            TLEMPLEADOS emp,
            TLUSUARIOS usr,
            TLMATALU mat,
            TLCURSOACA curs,
            TLOFEMATRGEN ofe,
            TLUNIDADESCEN uni

        WHERE
            mat.X_ALUMNO = alu.X_ALUMNO
        AND mat.C_ANNO = curs.C_ANNO
        AND mat.X_OFERTAMATRIG = ofe.X_OFERTAMATRIG
        AND NVL(mat.C_RESULTADO,99) > 1
        AND curs.L_ACTUAL = 'S'
        AND uni.X_UNIDAD = mat.X_UNIDAD
        AND falu.X_FAMILIA = alu.X_FAMILIA
        AND (falu.X_TUTOR1 = tut.X_TUTOR
            OR falu.X_TUTOR2 = tut.X_TUTOR )
        AND tut.C_NUMIDE = emp.C_NUMIDE
        AND emp.X_EMPLEADO = usr.X_EMPLEADO
        AND usr.USUARIO = #{idUsuario}

    </select>
```

Figure 32 - Consulta SQL de Alumno

Mediante el *ResultMap* podemos definir como guardar los resultados de la consulta en el objeto Alumno, en este caso si posee más de una matrícula se creará una colección objetos Matrícula. Mediante las etiquetas XML indicamos que se trata de una consulta *Select* cuyo nombre corresponde con el método *getAlumnosByUsuario* definido en la interfaz, todas las operaciones SQL definidas en estos archivos siguen la misma estructura del ejemplo.

```
<mapper namespace="yedra.persistence.NotaAsignaturaMapper">

    <cache/>

    <select id="getNotas" resultType="NotaAsignatura">
        SELECT decode(mog.X_OFERTAMATRIG, mua.X_OFERTAMATRIG, tlf_seldatmatomg('DES', mma.x_materiaomg),
        tlf_seldatmatomg('DES', mma.x_materiaomg)||' ('||(SELECT T_ABREVIATURA FROM tlofematrgen where x_ofertamatrig = mog.X_OFERTAMATRIG)||')') materia,
        gdm.D_GRUPOMAT tipo,
        CCE.S_CONVOCATORIA convocatoria,
        CCE.N_ORDEN orden,
        TO_CHAR((SELECT NVL((SELECT D_CALIFICA FROM TLCALIFICACIONES WHERE X_CALIFICA = NEV.X_CALIFICA), NEV.N_NOTA)
           FROM TLNOTEVA NEV
         WHERE NEV.X_MATMATRICULA = MMA.X_MATMATRICULA
         AND NEV.X_CONVCENTROOMC = CCO.X_CONVCENTROOMC)) nota,
        (SELECT x_califica
         FROM TLNOTEVA NEV2
         WHERE NEV2.X_MATMATRICULA = MMA.X_MATMATRICULA
         AND NEV2.X_CONVCENTROOMC = CCO.X_CONVCENTROOMC) califica,
         CCO.X_CONVFINAL as convFinal
        FROM TLMATALU MUA,
           TLMATMATRIALU MMA,
           TLCONVCENTROS CCE,
           TLCONVCENOMC CCO,
           TLMATOFEMATRG MOG,
           TLGRUPOMAT GDM,
           TLCONDMATERIAS CON
        WHERE TLF_USUARIO_PADRE_MATRICULA (#{idUsuario}, #{matricula}) = 'S'
        AND MUA.X_MATRICULA = #{matricula}
        AND MUA.X_MATRICULA = MMA.X_MATRICULA
        AND CCE.X_CONVCENTRO = CCO.X_CONVCENTRO
        AND CCO.X_OFERTAMATRIC = MUA.X_OFERTAMATRIC
        AND CCE.X_CENTRO = MUA.X_CENTRO
        AND CCE.C_ANNO = MUA.C_ANNO
        AND MMA.X_MATERIAOMG = MOG.X_MATERIAOMG
        AND GDM.X_GRUPOMAT = MOG.X_GRUPOMAT
        AND GDM.X_GRUPOMAT = CON.X_GRUPOMAT
        AND MOG.X_OFERTAMATRIG = CON.X_OFERTAMATRIG
        ORDER BY materia, orden ASC

    </select>
</mapper>
```

Figure 33 - Consulta del recurso Notas

En la imagen anterior se muestra la consulta que obtiene la lista de notas de una matrícula definida en el archivo XML de MyBatis. En la capa de servicio obteníamos el identificador del usuario "logueado" que era pasado en todas las llamadas a la capa de datos, bien para obtener la información del usuario o bien, como vemos en este caso, para comprobar mediante la función *TLF_USUARIO_PADRE_MATRICULA* que dicho usuario es padre del poseedor de esa matricula.

En el caso de los recursos Mensajes y Faltas es necesario hacer "inserts" y "updates" en la base de datos por lo que hay que manejar la generación de las claves primarias (*PrimaryKey*) y los argumentos necesarios para realizar la inserción. En la siguiente imagen vemos como se ha implementado este caso para el recurso Faltas:

```
<insert id="justificaFalta" parameterType="Falta">
   <selectKey keyProperty="x_justifica" resultType="int" order="BEFORE">
        SELECT TLS_JFAXJUSFALASIALU.nextval as x_justifica FROM DUAL
    </selectKey>
    INSERT INTO TLJUSFALASIALU SELECT
        #{x_justifica},
        #{id},
        #{motivo},
        #{observacion},
        #{usuarioCreacion},
        SYSDATE,
        NULL,
        NULL
    FROM
        tlfalasialu
    WHERE
        X_FALASIALU = #{id}
    AND C_TIPFAL = 'I'
</insert>

<update id="updateFaltaJustificada">
    UPDATE TLFALASIALU
    SET C_TIPFAL = 'J'
    WHERE
        X_FALASIALU = #{id}
</update>
```

Figure 34 - Consulta del recurso Faltas

Por último en la capa de datos también se encuentra las consultas usadas por el modelo de seguridad, donde mediante el uso de un Database Link obtendremos de la base de datos "Modulo de Acceso" (MODACC) el nombre, la contraseña y el estado del usuario en cada petición; así como los perfiles de acceso almacenados en la base de datos SEGEDU. También se ha implementado una consulta para incrementar el número de intentos fallidos del usuario:

```
<select id="getUser" resultMap="UserMap">
    SELECT
        usu.T_LOGIN as username,
        usu.T_CLAVE as password,
        usu.L_ACTIVO as activo,
        usu.L_BLOQUEADO as bloqueado,
        TO_CHAR(usu.oid) as oid
     FROM
        usuarios@modacc usu
    WHERE   usu.t_login = #{user}
</select>

<select id="getPerfil" resultMap="RolesMap">
    SELECT
        per.X_PERFIL as id,
        per.D_PERFIL as perfil
    FROM
        tlperfilusu pusu,
        tlperfiles  per,
        tlusuarios  usus,
        usuarios@modacc usu
    WHERE   usu.t_login= #{user}
    AND     pusu.x_perfil=PER.X_PERFIL
    AND     TO_CHAR(usu.oid)=USUS.USUARIO
    AND     USUS.X_USUARIO=PUSU.X_USUARIO
</select>
```

Figure 35 - Consulta Usuario y Perfiles

```
<update id="updateIntentosFallidos">
    UPDATE usuarios@modacc usu
    SET usu.N_INTENTOS_FALLIDOS_ACCESO = (
        SELECT
            usu.N_INTENTOS_FALLIDOS_ACCESO + 1
        FROM
            usuarios@modacc usu
        WHERE
            usu.t_login = #{user}
    ),
     usu.l_bloqueado = (
        DECODE (
            (
                SELECT
                    usu.N_INTENTOS_FALLIDOS_ACCESO + 1
                FROM
                    usuarios@modacc usu
                WHERE
                    usu.t_login = #{user}
            ),
            3,
            'S',
            'N'
        )
    )
    WHERE
        usu.t_login = #{user}
</update>
```

Figure 36 - Consulta updateIntentosFallidos

5.1.4 Implementación del modelo de seguridad

Como se definió en el apartado de tecnologías, el modelo de seguridad se ha implementado usando SpringSecurity, que nos proporciona restricciones de acceso a recursos, URLs y métodos a través de la asignación de perfiles de usuario. En este caso los perfiles son definidos en base a los almacenados en la base de datos, dando acceso a los recursos solo a los usuarios con perfil "Padre/Madre". Dicho acceso es definido en los ficheros de configuración mediante in patrón que intercepta URLs de la siguiente forma:

```
<security:http pattern="/yedra/**" create-session="stateless" entry-point-ref="authenticationEntryPoint">
        <security:intercept-url pattern='/**' access='ROLE_PADRE' requires-channel='http' />
        <security:http-basic />
</security:http>

<security:authentication-manager>
    <security:authentication-provider ref="restAuthenticationProvider" />
</security:authentication-manager>
```

Figure 37 - Configuracion SpringSecurity

Este framework permite automatizar la búsqueda de los datos de usuario y perfiles implementando el servicio *UserDetailsService*, pero requiere que las tablas de usuario y perfiles sigan los requerimientos del framework. En este caso es imposible implementar el sistema de seguridad de esta forma por la complejidad del esquema de bases de datos de este sistema, por tanto ha sido necesario implementar por completo el sistema de seguridad, diseñando una clase a medida que proporciona acceso al objeto Token con las credenciales recibidas e implementando un *AuthenticationProvider* a nuestra medida para realizar la comprobación de contraseña y asignar perfiles.

```
public class RestToken extends UsernamePasswordAuthenticationToken {

    private static final long serialVersionUID = 1L;

    public RestToken(UserCredentials user, String credentials) {
        super(user, credentials);
    }


    public RestToken(UserCredentials user, String credentials, Collection<? extends GrantedAuthority> authorities) {
        super(user, credentials, authorities);
    }

    @Override
    public UserCredentials getPrincipal() {
        return (UserCredentials) super.getPrincipal();
    }

    @Override
    public String getCredentials() {
        return (String) super.getCredentials();
    }
```

Figure 38 - RestToken con las credenciales del usuario

El token de autenticación ha sido re-diseñado, en lugar de guardar el nombre de usuario, contraseña y perfiles; se ha sustituido el nombre por el objeto completo obtenido de la base de datos, de esta forma obtendremos en la capa de servicio el objeto de usuario con las claves necesarias para las comprobaciones de seguridad en las consultas. Esto nos evita realizar una doble consulta a la base de

datos MODACC en cada petición del usuario aprovechando la realizada para obtener las credenciales, ahorrando tráfico en la red del sistema.

La clase *RestAuthenticationProvider* implementa un *AuthenticationProvider* encargado de comprobar las credenciales obtenidas en la cabecera de cada aplicación, asignar perfiles y proporcionar el acceso a los recursos. En el siguiente ejemplo se muestra la implementación del método encargado de realizarlo:

```
@Override
public Authentication authenticate(Authentication auth) throws AuthenticationException {
    // Obtiene el nombre de usuario y contraseña de los headers de la peticion HTML
    String username = (String) auth.getPrincipal();
    String password = (String) auth.getCredentials();

    // Obtiene el usuario de la BD con el username pedido por el cliente
    UserCredentials user = userService.loadUserByUsername(username);

    if (user == null) {
        throw new BadCredentialsException("Username: "+auth.getPrincipal()+" not found");
    }
    // si el usuario o la cuenta no esta habilitado lanza excepcion
    if(!user.isEnabled()) {
        throw new BadCredentialsException("Username: "+auth.getPrincipal()+" not enable");
    }
    // si el usuario o la cuenta esta bloqueado lanza excepcion
    if(!user.isAccountNonLocked()) {
        throw new BadCredentialsException("Username: "+auth.getPrincipal()+" blocked");
    }
    // Autentica la contraseña enviada por el usuario contra la base de datos
    if (user.getPassword().equals(password)) {

        // Asigna Rol de usuario a los padres
        List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
        List<Perfil> perfiles = user.getPerfiles();

        for (Perfil perfil : perfiles) {
            if (perfil.getPerfil().equals(PADRE)) {
                authorities.add(new SimpleGrantedAuthority(ROLE_PADRE));
            }
            if (perfil.getPerfil().equals(PROFESOR)) {
                authorities.add(new SimpleGrantedAuthority(ROLE_PROF));
            }
        }
        user.setAuthorities(authorities);

        RestToken token = new RestToken(user, user.getPassword(),user.getAuthorities());
        SecurityContextHolder.getContext().setAuthentication((Authentication) token);
        return token;
    } else {

        // La contraseña es incorrecta, aumenta el numero de intentos fallidos y lanza la excepcion en el servidor
        userService.updateIntentosFallidos(username);
        throw new BadCredentialsException("Username: "+user.getUsername()+" not matching with his password");
    }
```

Figure 39 - AuthenticationProvider, método de autenticación

Este método recibe como parámetro un intento de autenticación, compuesto por nombre y contraseña, a continuación realiza una llamada a la capa de servicio para obtener el objeto asociado al nombre de usuario recibido y se realizan comprobaciones del estado de la cuenta. A continuación se comprueban las contraseñas, ambas codificadas en MD5 y si es correcta se le asignan los roles de acceso al usuario, se crea un token con los datos y se inserta en la clase *SecurityContextHolder*, pudiendo obtenerlo después en la capa de servicio.

5.2 Implementación de la aplicación móvil

La implementación de la aplicación móvil se ha realizado con el framework Titanium Appcelerator, que como se explicó en la parte de herramientas (ver índice 2.1.3.1) permite desarrollar aplicaciones escritas en Javascript que son compiladas a código nativo, tanto iOS como Android.

Para la implementación de la aplicación móvil se han dividido los modelos de datos de las llamadas al servidor y la interfaz gráfica. Los modelos implementados siguen la misma estructura de los recursos diseñados en el servidor, la capa de acceso a datos contiene las llamadas al servidor y el almacenamiento de las respuestas en los modelos y las vistas contienen la implementación de los *wireframes* diseñados y el manejo de eventos.

5.2.1 Implementación de la capa de acceso a datos

La capa de datos se compone de un conjunto de clases con funciones que realizan llamadas AJAX a la API del servidor. Cada recurso está representado en una clase la cual contiene cada posible llamada a cada recurso en un método independiente.

La primera llamada que realiza la aplicación corresponde al recurso que devuelve la información de usuario, esta llamada es realizada en la pantalla de login y una vez recibida la respuesta del servidor, guarda las credenciales de usuario en la aplicación, ya que han sido validadas y deben ser utilizadas en cada llamada posterior; llama a la petición de información de alumnos y a la vista de inicio que muestra a los alumnos:

```
var url = "http://localhost:8080/YedraWebService_v1.1.1/yedra/user";
var x_usuario,c_numide,nombre,apellido1,apellido2,telefono,email,calle,numCalle,escalera,piso,letra,c_postal,localidad,municipio

var xhr = Ti.Network.createHTTPClient({
    onload : function() {
        // Una vez validadas, guarda las credenciales en las propiedades de la app
        Ti.App.Properties.setString("user", userCredentials.user);
        Ti.App.Properties.setString("pwd", userCredentials.password);

        // Parsea la persona recibida en las estructuras del programa
        json = JSON.parse(this.responseText).persona;
        x_usuario = json.x_usuario;
        c_numide = json.c_numide;
        nombre = json.nombre;
        apellido1 = json.apellido1;
        apellido2 = json.apellido2;
        telefono = json.telefono;
        email = json.email;
        calle = json.calle;
        numCalle = json.numCalle;
        escalera = json.escalera;
        piso = json.piso;
        letra = json.letra;
        c_postal = json.c_postal;
        localidad = json.localidad;
        municipio = json.municipio;
        provincia = json.provincia;

        usuario = new Persona(x_usuario,c_numide,nombre,apellido1,apellido2,telefono,email,calle,numCalle,escalera,piso,letra,c_

        Ti.API.info("Usuario recibido, lanza vista de alumnos");
        getAlumnos();
        InitView();
    },
    onerror : function(e) {
        Ti.API.info("STATUS: " + this.statusText);
        if (this.status === 401) {
            alert("Contraseña incorrecta");
        }
    },
    timeout : 5000
});

xhr.open('GET', url);

authstr = 'Basic ' + Ti.Utils.base64encode(userCredentials.user + ':' + userCredentials.password);
xhr.setRequestHeader('Authorization', authstr);

xhr.send();
```

Figure 40 - Llamada al recurso usuario

Como vemos en esta porción de código el framework nos proporciona soporte para hacer llamadas asíncronas al servidor, para ello debemos precisar la URL del recurso y el método de la llamada, y añadir las credenciales de usuario codificadas usando Basic Auth a la cabecera *Authorization*. Si el servidor devuelve un error comprobamos si las credenciales han sido incorrectas y en caso de éxito guardamos la información de usuario y lanzamos la siguiente vista.

En el caso de las funciones que realizan peticiones PUT o DELETE la estructura es la misma, cambiando solo el método y añadiendo la URL del recurso correspondiente, no siendo necesario guardar ninguna respuesta del servidor. En los recursos de Faltas y Mensajes se utilizan llamadas POST incluyendo un objeto de cada tipo en el cuerpo de la petición para justificar una falta o crear un mensaje nuevo, en este caso se recibe el objeto como parámetro de la función, se convierte a formato JSON y se añaden las cabeceras indicando el tipo de contenido:

```
// Realiza la peticion al servidor para justificar una falta asociada a una matricula
// Parametro: id de la matricula, objeto Falta
//
// POST - /yedra/matricula/{matricula}/faltas
//
function justificaFalta(matricula, falta) {
    url = faltasURL + matricula + "/faltas";
    var xhr = Ti.Network.createHTTPClient({

        onload : function() {
            Ti.API.info(this.statusText);
            Ti.API.info(this.responseText);
        },
        onerror : function(e) {
            Ti.API.error(e.error);
        },
        timeout : 5000
    });

    // Monta las cabezeras de autenticacion de HTTP y lanza la peticion
    xhr.open('POST', url);
    authstr = 'Basic ' + Ti.Utils.base64encode(userCredentials.user + ':' + userCredentials.password);
    xhr.setRequestHeader('Authorization', authstr);
    xhr.setRequestHeader('Content-Type','application/json; charset=utf-8');

    json = JSON.stringify(falta);
    Ti.API.info(json);
    xhr.send(json);
}
```

Figure 41 - Llamada para justificar una falta

5.2.2 Implementación de las vistas y controladores

La implementación de las vistas de la aplicación se han basado en los *Wireframes* diseñados con la distribución de datos, controles y flujo de las vistas. El departamento de diseño de la empresa se ha encargado de diseñar los iconos, colores y apariencia final de acorde al diseño actual de Yedra.

Para comenzar el usuario introduce sus credenciales en la pantalla de login, que una vez validadas en el servidor quedarán almacenadas en el dispositivo bajo un cifrado MD5 y permitirán acceder a la pantalla principal en sucesivas ocasiones sin necesidad de realizar este paso. A partir de esta pantalla de realizan las llamadas para obtener los datos de usuario y de alumnos registrados y se lanza la pantalla principal:

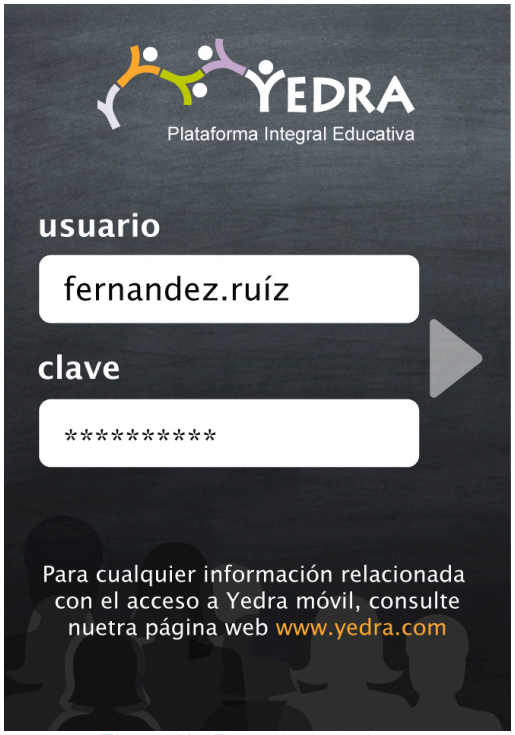

Figure 42 - Pantalla de login

Figure 43 - Pantalla principal

En la pantalla principal se muestra el nombre de usuario, que nos llevará a la pantalla de información de usuario al pulsar sobre él, una lista con los alumnos registrados en el curso actual y las matriculas de cada uno, y una barra inferior con un icono que nos lleva a los mensajes y a la información de usuario. Cada matricula muestra un icono que nos lleva a la pantalla de información del alumno, centro, clase y tutor; otro que nos lleva a la pantalla de notas asociadas a esa matricula y el ultimo que nos lleva a las faltas de asistencia mostrando el número de faltas sin justificar.

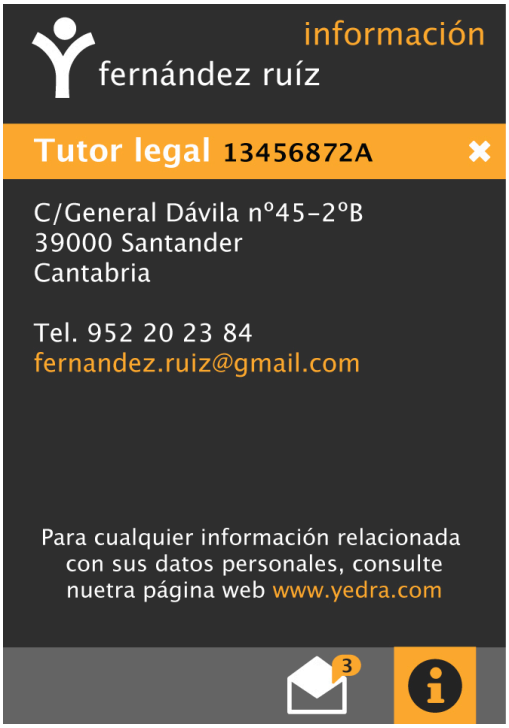

Figure 44 - Pantalla de información de usuario

Figure 45 - Pantalla de información de matrícula

Figure 46 - Pantalla de notas 1ª eval

Figure 47 - Pantalla de notas 2ª eval

La pantalla con las notas asociadas a una matrícula muestra las calificaciones de cada evaluación registrada, mostrando un código de color para cada rango de notas. La navegación se lleva a cabo desplazando el dedo hacia los lados para cambiar de evaluación.

Figure 48 - Pantalla de lista de faltas

Figure 49 - Pantalla de vista de calendario

La pantalla de faltas obtiene la lista de todas las faltas asociadas a la matrícula y las ordena por fecha y agrupa por mes indicando el número de injustificadas. En esta vista (Figura 46) se ordenan por tramo horario y día las faltas que están sin justificar y pulsando el botón derecho de cada fila nos lleva a una pantalla para observar los detalles de la falta y crear una justificación. El icono del calendario nos lleva a la vista por meses (Figura 47) donde podemos ver todos los tipos de faltas distribuidas en cada mes. Si el día aparece en rojo hay alguna falta en algún tramo horario sin justificar, si está en verde hay una o más faltas ya justificadas y si está en gris hay uno a más retrasos en ese día.

En ambas pantallas tenemos un icono que nos lleva a crear un nuevo mensaje para el tutor en caso de que tengamos alguna duda.

Figure 50 - Pantalla de justificación

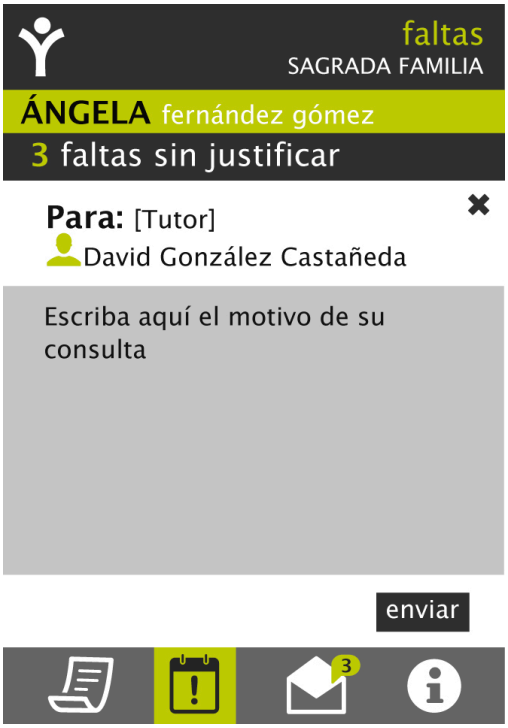

Figure 51 - Pantalla de creación de mensaje

La pantalla de justificación muestra los detalles de la falta y nos ofrece un selector del motivo de falta y un área para escribir un mensaje de justificación, que es enviado al servidor al pulsar el botón "justificar". También posee un icono de mensaje que nos lleva a la pantalla siguiente (Figura 49) donde se realiza una petición con los datos e identificador del tutor y nos permite enviarle un mensaje directo.

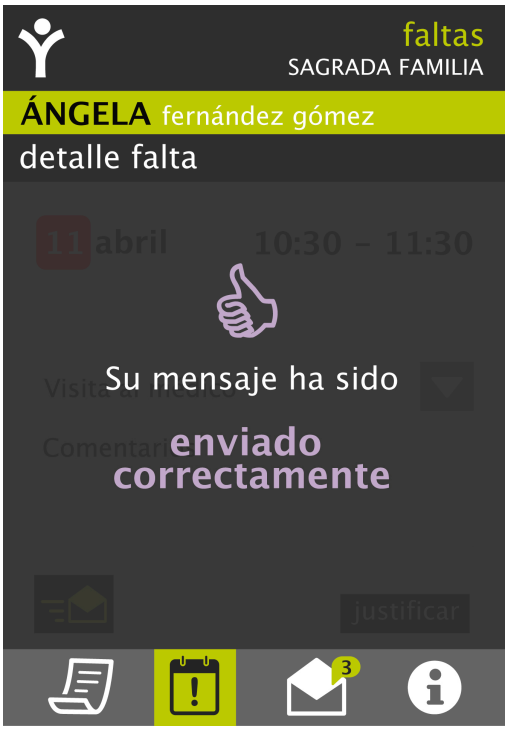

Figure 52 - Pantalla de confirmación de mensaje

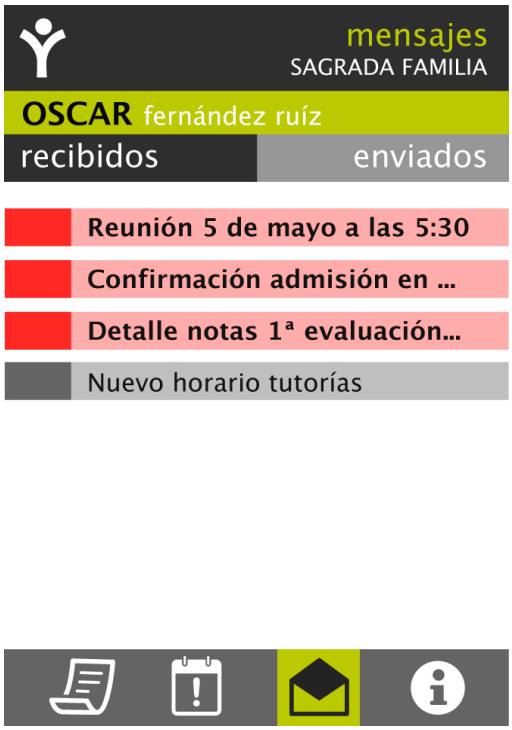

Figure 53 - Pantalla de mensajes

6 Pruebas

Durante el desarrollo del proyecto se han llevado a cabo una serie de pruebas para comprobar el correcto funcionamiento del sistema y la seguridad e integridad del mismo.

Mediante la metodología Scrum se diseñaron los distintos test que se aplicaron a cada recurso en la fase de implementación, permitiendo producir resultados satisfactorios y garantizar el funcionamiento de los recursos que hemos ido creando, de la misma forma evitamos repetir los mismos errores en los nuevos recursos que vamos creando.

Las pruebas se han dividido en cuatro tipos:

- Pruebas unitarias
- Pruebas de integración
- Pruebas de sistema
- Pruebas de aceptación

6.1 Pruebas unitarias

Las pruebas unitarias se han basado en la búsqueda de errores en cada recurso del sistema. Para ello, una vez definidos los recursos y haciendo uso de la metodología elegida, se ha comenzado diseñando las consultas SQL y comprobando que la respuesta obtenida es la correcta, a continuación se ha implementado cada uno, comprobando que las operaciones de la API aceptan las peticiones diseñadas y envían las respuestas correctas.

En cada una de estas pruebas se ha asegurado el correcto funcionamiento del sistema de seguridad, mediante el envío de peticiones con unas credenciales de usuario ya conocidas y alterando en algunos casos información en la base de datos para producir posibles casos excepcionales, alumnos con dobles matriculas o asignación de distintos perfiles de acceso.

En la aplicación móvil las pruebas unitarias se realizaron comprobando el manejo de cada recurso individual, la presentación de los datos y el correcto flujo de actividad entre las pantallas.

6.2 Pruebas de integración

Las pruebas de integración consistieron en comprobar el funcionamiento de todos los módulos integrados en el sistema. Para ello en cada iteración de nuestra metodología ha sido básico probar la integración del módulo de seguridad con

cada uno de los recursos creados, permitiendo su manejo solo a los usuarios registrados con determinado perfil. En el servidor también ha sido una pieza clave la integración del recurso de "Alumnos" y la obtención de sus matrículas con el manejo de otros recursos que requieren de estos identificadores para su correcto funcionamiento.

En la aplicación móvil las pruebas de integración se han llevado a cabo de manera incremental comprobando el correcto funcionamiento del flujo entre pantallas mediante eventos, la petición de información y el lanzamiento de las pantallas de presentación de la información una vez obtenidas las respuestas.

6.3 Pruebas de sistema

Las pruebas de sistema verifican el correcto funcionamiento en todo su conjunto, para ello y al igual que en todo el proceso, las pruebas se han llevado a cabo con la base de datos de desarrollo del sistema Yedra, que contiene la misma estructura que la base de datos del sistema pero con datos barajados por cuestiones de privacidad.

Las pruebas de integración se realizaron una vez el servidor estaba completamente operativo, ejecutándose en el equipo de forma local y utilizando un emulador para ejecutar la aplicación móvil.

Dado que no se ha podido instalar el servidor en las máquinas de la consejería de educación ni sobre la base de datos de producción, estas pruebas no se han podido realizar sobre el equipo en condiciones de carga normales junto el resto de los componentes del sistema Yedra.

6.4 Pruebas de Aceptación

Las pruebas de aceptación validan el funcionamiento del sistema según los requerimientos del proyecto y permiten que el cliente determine su aceptación, teniendo en cuenta el rendimiento y la funcionalidad.

Estas pruebas fueron llevadas a cabo en cada sprint del proceso, validando con la empresa los datos necesarios que debería visualizar el usuario, diseñando y validando la interfaz de la aplicación móvil y aportando mejoras en las pantallas, la navegación y los aspectos de seguridad tales como guardar credenciales de acceso en la aplicación.

7 Conclusiones y trabajo futuro

Una vez finalizado el proyecto podemos sacar las conclusiones finales obtenidas de la realización y ayudar con la planificación de futuras mejoras y ampliaciones en el sistema.

7.1 Conclusiones

El desarrollo de este proyecto tenía como objetivo crear una nueva herramienta capaz de proporcionar una buena experiencia de usuario en el servicio Yedra a través del Smartphone. Para ello se han realizado todas las fases del ciclo de vida software, como son la captura y análisis de requisitos, diseño, implementación y pruebas, cumpliendo satisfactoriamente con los objetivos marcados al inicio del proyecto.

Las metas alcanzadas en el proyecto han sido:

- La creación de una aplicación móvil que permita a los usuarios hacer uso de las mismas funciones presentes en el actual sistema de seguimiento educativo de la Consejería de Educación de Cantabria. Esto se ha conseguido con la creación de una aplicación multi-plataforma con un diseño simple y una interfaz fácil de usar por todo tipo de usuarios.

- El diseño y creación de un nuevo servidor que de soporte a la aplicación móvil y que no comprometa el rendimiento de los equipos de la consejería de educación ni de las aplicaciones allí instaladas. Para ello se diseñó un servidor con un diseño REST que no guarda información de sesión de los usuarios y con unos recursos bien definidos que evitan una excesiva complejidad en las operaciones internas.

- La creación de una arquitectura en el servidor, una API y un sistema de seguridad que permita una evolución del servidor añadiendo nuevos recursos y nuevos perfiles de acceso haciendo. Para ello se implementó un sistema de seguridad basado en perfiles presentes en la base de datos que son asignados a cada usuario en cada petición al servidor, así como restricciones de acceso a los recursos por medio de las URL.

Con esta aplicación, los ciudadanos de Cantabria con hijos escolarizados podrán realizar el seguimiento escolar de todos ellos, consultando información del centro al que acuden, su tutor, sus notas, sus faltas de asistencia, así como justificar dichas faltas de una manera sencilla y rápida, consultar sus mensajes y comunicarse con el tutor de sus hijos.

Esta mejora de los servicios pretende incrementar el uso de la plataforma Yedra entre los usuarios de Cantabria, ya que el actual sistema no permitía un uso agradable a través de tablets o smartphones y la mayoría de padres no hacían uso de la herramienta para aspectos tan importantes como pedir citas directamente al tutor, consultar notas y faltas y justificar estas últimas en el sistema.

En el apartado personal cabe resaltar la satisfacción por tomar parte en un proyecto que afectará a un gran número de personas en nuestra comunidad y sentar las bases de un sistema que en años venideros será ampliado y mejorado para añadir las funcionalidades actuales, de las que hacen uso profesores y personal técnico, a las plataformas móviles del sistema; dado el creciente uso de estas y la mejora que supondría por ejemplo, en el caso de los profesores, poder crear cada día las nuevas faltas de asistencia al inicio de la clase mediante su móvil o registrar las notas de exámenes de una manera rápida y sencilla sin depender de un ordenador. Al comenzar el proyecto mis conocimientos en el desarrollo de este tipo de sistemas era mínimo, pero gracias a los conocimientos adquiridos en la carrera pude asimilar de una manera rápida toda la información relacionada con estas herramientas y tecnologías, hasta el punto de ser capaz de diseñar e implementar un sistema con aplicación comercial y enfrentarme sin ayuda a los continuos problemas y errores que el desarrollo de un sistema de este tipo genera. Este proyecto me ha abierto las puertas al mundo laboral y ha despertado mi interés por el desarrollo de este tipo de sistemas, motivándome para aprender nuevas tecnologías, lenguajes y frameworks de desarrollo; considerando personalmente que la curiosidad y el interés continuo por ampliar conocimientos es una de las semillas fundamentales que la universidad debe implantar en todo ingeniero.

7.2 Planificación y evolución del servidor

Uno de los objetivos del proyecto ha sido la creación de una plataforma capaz de proporcionar las características actuales del sistema yedra a las plataformas móviles. Dado que el sistema actual no soporta una evolución hacia estas plataformas, era básico diseñar un nuevo servidor que nos solo soporte los requerimientos básicos del proyecto si no que sea capaz de integrar nuevos recursos, perfiles de acceso y funcionalidades en un futuro.

Uno de los futuros escenarios planteados durante la fase de captura de requisitos fue crear una aplicación exclusiva para profesores, con características tales como creación de faltas, notas, consulta de horarios y consulta de alumnos. Por ello sería necesario que el servidor sea capaz de dar soporte en un futuro a otros perfiles de acceso, así como la fácil creación de nuevos recursos y la protección de acceso a ellos.

Por tanto, dadas estas premisas, el sistema de seguridad se ha diseñado para que lea los perfiles de usuario de la base de datos del módulo de acceso y le asigne unos roles predefinidos en el controlador de acceso. La configuración actual de este controlador limita el acceso a los recursos creados para aquellos definidos como "Padre/Madre" en la base de datos, pero actualmente el sistema reconoce también a los profesores y les asigna su rol. Una vez definidos recursos de uso exclusivo para profesores, en el archivo de configuración solo sería necesario indicar las URL o los patrones de esas URL (por ejemplo "*yedra/profesores/*")* que queramos proteger del uso de otros usuarios y hacerlas exclusivas para estos perfiles.

La creación de nuevos recursos, como vimos durante la fase de diseño e implementación, sigue un patrón fácilmente escalable y junto con la documentación aportada será relativamente fácil para futuros desarrolladores crear los recursos diseñados a través de las distintas capas de la aplicación, y asignarles un acceso exclusivo a los perfiles deseados.

Como ejemplo, la creación de un recurso que permita consultar su horario a los profesores implicaría la creación de la consulta deseada en la base de datos en la capa de persistencia, el objeto de dominio para guardar los datos, la clase correspondiente en la capa de servicio junto a la extracción de su id de usuario para la consulta, y el controlador junto con el patrón de URL para acceder al recurso (por ejemplo "*yedra/profesores/horario*") indicando en la configuración de seguridad el patrón definido anteriormente.

8 Referencias

Allamaraju, S. (2010). *RESTful Web Services Cookbook* (1st Edition ed.). (M. E. Terseler, Ed.) O'Reilly Media Inc.

Anderson, J. (2013). *Appcelerator Titanium: Up and Running* (1st Edition ed.). O'Reilly Media, Inc.

Fling, B. (2009). *Mobile Design and Development* (1st Edition ed.). (S. Weiss, Ed.) O'Reilly Media.

Gourley, D., & Totty, B. (2002). *HTTP: The definitive Guide* (1st Edition ed.). (L. Mui, Ed.) O'Reilly Media, Inc.

Johnson, R., Hoeller, J., Arendsen, A., Risberg, T., & Sampaleanu, C. (2005). *Professional Java Development with the Spring Framework*. Wiley Publishing, Inc.

Mak, G., Long, J., & Rubio, D. (2010). *Spring Recipes: A Problem-Solution Approach* (2nd Edition ed.). (I. Springer Science+Business Media, Ed.) Paul Manning.

Schwaber, K., & Mike, B. (2001). *Agile Software Development with Scrum* (1st Edition ed.). Prentice Hall.

Standardization, I. -I. (n.d.). Retrieved 25 de 01 de 2014 from Systems and Software Engineering - Software life cycle processes: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43447

Reddy, K. S. (2013). *Java Persistence with MyBatis 3*. Packt Publishing.