eman ta zabal zazu

Escuela Universitaria de Ingeniería Técnica Industrial de Bilbao

Grado en Ingeniería Informatica de Gestión y Sistemas de Información

Trabajo Fin de Grado

2014 / 2015

BABELIUM, MIGRACIÓN A HTML5

MEMORIA DEL PROYECTO

DATOS DE LA ALUMNA O DEL ALUMNO	DATOS DEL DIRECTOR O DE LA DIRECTORA
NOMBRE: JON	NOMBRE: JUAN ANTONIO
APELLIDOS: LACHÉN LEAL	APELLIDOS: PEREIRA VARELA
	DEPARTAMENTO:
FDO.:	FDO.:
FECHA:	FECHA:

Anexo II

Índice general

Índice de figuras

Índice de tablas

DESCRIPCIÓN DEL PROYECTO

Babelium es una aplicación web cuyo objetivo es el aprendizaje de idiomas a través de la red, creando una comunidad en la que los nativos de un idioma ayuden a otros a dominarlo de forma que todos salgan beneficiados.

El modo en el que se realiza esta ayuda es mediante vídeos, los usuarios se graban practicando con ejercicios de los vídeos de esta web, utilizando las herramientas dadas por la misma, y otros usuarios más tarde evaluarán su ejercicio.

Este proyecto surge de la necesidad de adaptar la aplicación web a las nuevas tecnologías que están creando un estándar (HTML5 y JavaScript). Esta aplicación está desarrollada en Flex, un conjunto de tecnologías de Adobe para la creación de una aplicación web enriquecida basada en Flash, las cuales se están quedando ya antiquadas.

El proyecto consiste en la migración de las funcionalidades y apariencia de la web original a HTML5. Para ello, debemos analizar las tecnologías actuales disponibles, seleccionar las herramientas a utilizar, analizar todas las funcionalidades de la aplicación web Babelium basada en Flash, emplear todo lo anterior en buscar la mejor forma de llevar a cabo la migración, implementar la aplicación y documentar todo lo realizado.

DEFINICIONES Y ACRÓNIMOS

Framework: Estructura conceptual y tecnológica que puede servir de base para la organización y desarrollo de software. Puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

RIA: Rich Internet Application, es decir, aplicaciones de Internet enriquecidas. Son aplicaciones web con las características de las aplicaciones de escritorio.

Flex: Es un kit de desarrollo de software para la creación de RIA basado en la plataforma de Adobe Flash.

MVC: Modelo-vista-controlador, es un patrón de arquitectura de software que separa los datos y la lógica de negocio de la interfaz de usuario.

Template: Plantilla que contiene el código HTML5 que estructura una pantalla de la aplicación web.

HTML5: Quinta revisión del lenguaje básico de la World Wide Web utilizado en el diseño de paginas web. Este lenguaje soporta elementos multimedia tales como videos, audio, canvas y gráficos vectoriales.

API: Conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software.

PHP: Acrónimo de Hypertext Preprocessor, lenguaje de código abierto del lado del servidor orientado al desarrollo web.

Plugin: Complemento de una aplicación que aporta una funcionalidad específica nueva.

MV: Máquina virtual, software que simula ser un equipo independiente capaz de ejecutar programas como si fuera un ordenador de verdad.

SSH: Secure Shell, protocolo que sirve para acceder a máquinas remotas mediante una conexión segura establecida con claves RSA.

RSA: Rivest, Shamir y Adleman, sistema criptográfico de claves públicas, basado en el uso de una clave pública que puede ser difundida y una privada que es única y prueba de que el que la usa es el propietario.

INTRODUCCIÓN

1. Antecedentes

Origen del proyecto

La aplicación a desarrollar no es una aplicación nueva, Babelium es una aplicación existente implementada mediante Flash, por lo que este proyecto consiste en la actualización del Babelium existente.

Decidí desarrollar este proyecto tras reunirme con mi tutor, quién me explico en que consistía y que se encontraba disponible. Me llamó la atención el hecho de que la que se encarga de mantener en pie la aplicación es la misma comunidad de usuarios, y me sorprendió ver la variedad de ejercicios que los usuarios suben a la aplicación con el objetivo de ayudar a otros.

Desde un principio quería realizar un proyecto basado en una aplicación web o para móvil debido a que es lo más utilizado en la actualidad. Ver lo interesante que podría resultar esta aplicación y el hecho de que en el proyecto debería utilizar un framework en auge en la actualidad me llevo a aceptar la propuesta.

Situación previa

Se puede acceder a la versión Flash de Babelium desde la dirección http://babeliumproject.com/. Esta aplicación cuenta con diversas funcionalidades para los usuarios, entre las que destacan:

- Gestión de cuentas de usuario.
- Inclusión de nuevos ejercicios al repertorio de la aplicación.
- Configuración de cámara y micrófono.
- Grabación y gestión de respuestas a ejercicios.
- Evaluación de ejercicios ajenos.
- Subtitulado de videos existentes que no están preparados para su uso como ejercicios.

Con todas estas funcionalidades vemos que Babelium es una aplicación muy desarrollada y que funciona gracias a sus usuarios.

Anteriormente al desarrollo de este proyecto, ya existía un prototipo de Babelium en HTML5. Este prototipo surge de un proyecto de fin de carrera cuyo objetivo era el estudio de la viabilidad de actualizar Babelium a HTML5. El prototipo, obviamente, no incluye todas las funcionalidades

mencionadas anteriormente pero si permite el uso de cuentas de usuario y la visualización y realización de ejercicios.

Este proyecto de actualización de Babelium empieza donde terminó el proyecto mencionado, retomando el prototipo y convirtiéndolo en una aplicación web operativa.

2. Viabilidad

En este apartado se analizará la viabilidad del proyecto, basándonos desde dos puntos de vista, el tecnológico y el económico.

Tecnológico

Desde el punto de vista tecnológico, nos interesa saber si disponemos del software necesario y adecuado para realizar la actualización. Es cierto que algunas herramientas pueden implicar gastos, pero en esta sección lo que nos importa es que esas herramientas existan.

En un apartado posterior se dará una explicación mas detallada de cada herramienta mencionada.

Para la implementación de la aplicación web, lo básico es disponer de un entorno de desarrollo que permita el uso de HTML5 y Javascript, así como sus respectivas librerías y frameworks. Esta necesidad se satisface con herramientas como el entorno de desarrollo NetBeans o simples editores de texto especializados para programación como SublimeText3. Ambas aplicaciones son gratuitas.

La siguiente necesidad durante el desarrollo de la aplicación es un entorno de desarrollo web que permita montar aplicaciones web en un servidor propio con el fin de ir probando la aplicación web para visualizar los progresos, realizar pruebas y conectar con bases de datos. Para cubrir esta necesidad disponemos de aplicaciones como WAMP o, de nuevo, NetBeans. Una vez más, ambas aplicaciones gratuitas.

También es necesaria un sistema de gestión de versiones de la aplicación para organizar la aplicación y disponer de versiones anteriores en caso de que surjan problemas. La solución a esta necesidad es GitHub una aplicación web que, además de almacenar los ficheros que conforman la aplicación, se encarga de realizar un seguimiento de versiones de la misma. Además de esto incluye distintos apartados para ayudar con la organización, como foros entre los miembros del equipo de trabajo, sistema de asignación de tareas, etc. Por supuesto, también es una aplicación gratuita.

Para poder poner en funcionamiento la aplicación, es necesario disponer del API de Babelium para su uso. Esto quiere decir que es necesario disponer de un servidor de Babelium online sobre el que poder realizar consultas y del que obtener información de usuarios y ejercicios. Para

poder disponer de un API de Babelium estable sobre el que se van a realizar cambios y ajustes con frecuencia para adaptarlo a la nueva aplicación es necesario recurrir a herramientas que permitan crear máquinas virtuales tales como, VMWare o Digital Ocean.

En el caso de usar VMWare, se dispondría de una máquina virtual gratuita en local, es decir, en el mismo equipo sobre el que se está desarrollando la aplicación.

Con Digital Ocean se alquila un servidor para que se mantenga online las 24 horas del día sin que se ejerza ningún estrés adicional en el equipo sobre el que se desarrolla la aplicación y con control total de dicho servidor.

Podemos ver que tenemos de nuevo opciones gratuitas, pero, a parte de ser mucho mas cómodo alquilar el servidor, Babelium requiere tener instaladas aplicaciones y versiones de aplicaciones muy concretas (ffmpeg, red5, Zend Framework y soporte de tareas automáticas con cron) que si no alquilas, no dispondrías de ellas.

El resto de necesidades, como herramientas para llevar a cabo la documentación, creación de diagramas, etc, son cubiertas fácilmente por aplicaciones gratuitas comúnmente conocidas, tales como LibreOffice, Cacoo o Visual Paradigm.

Viendo que estas necesidades están cubiertas sin ningún tipo de problema, podemos asumir que disponemos de la tecnología necesaria para llevar a cabo el proyecto.

Económico

En este apartado debemos considerar los gastos generados durante la realización del proyecto. Estos gastos tienen distintos orígenes:

- Mano de obra

Para calcular el coste de la mano de obra debemos tener en cuenta el número de horas de trabajo necesarias para llevar a cabo el proyecto. Como se mostrará más adelante en la sección de planificación, la duración del proyecto es de 409 horas, siendo el sueldo de un programador junior como yo aproximadamente 12€/hora:

$$\frac{409\,Horas \times 12\,€}{Hora} = 4.908\,€$$

- Materiales y herramientas

Como hemos visto en el apartado de viabilidad desde el punto de vista tecnológico, se puede realizar el proyecto sin gastar ni un solo euro en herramientas ni software. En mi caso, por comodidad, he utilizado la aplicación Digital Ocean durante los últimos 4 meses del proyecto para la implementación de la aplicación. Si cuando creas la cuenta en la aplicación eres invitado por un usuario, recibes una bonificación de 10€, en mi caso

fui invitado por mi tutor. Alquilar un servidor tiene un coste de 10€ al mes, por consiguiente, con toda la información reunida:

$$(4 \times 10\,€) - 10\,€ = 30\,€$$

- Gastos indirectos

Estos gastos se refieren a los gastos por luz y conexión a Internet, ambos consumidos indirectamente durante la realización del proyecto.

Siendo el coste de la luz utilizada cerca de unos 20€/mes, la conexión a Internet 22€/mes y la duración del proyecto 9 meses:

$$\left(\frac{20\,€}{Mes} + \frac{22\,€}{Mes}\right) * 9\,Meses = 378\,€$$

- Gastos totales

En la siguiente tabla se reflejan todos los datos recabados hasta ahora y cuales son los gastos totales del proyecto:

Mano de obra	4.908 €
Materiales y herramientas	30 €
Gastos indirectos	378 €
Total	5.316 €

Tabla 1: Gastos totales

3. Propósito

Adaptar la aplicación web Babelium a las nuevas tecnologías para permitir que siga evolucionando como herramienta de aprendizaje de idiomas comunitaria y que la gente siga ayudándose mutuamente.

4. Ámbito

La aplicación web seguirá dirigida a gestionar vídeos educativos con el fin de que los usuarios puedan practicar idiomas y grabarse haciendo ejercicios, así como la gestión de los usuarios, sus aportaciones y sus evaluaciones (tanto dadas como recibidas).

5. Razones de la migración

La necesidad de la migración surge del abandono del desarrollo de Flash para dispositivos móviles y la donación de Flex a la comunidad por parte de Adobe. Seguir desarrollando funcionalidades en Flash sería una pérdida de tiempo, el cual se podría emplear en migrar la aplicación a HTML5 usando un framework de JavaScript como Backbone.js o Angular.js, la tecnología que está estandarizada en Internet en la actualidad.

Además, el prototipo de Babelium anterior utilizaba una implementación ad-hoc de todas las funcionalidades JavaScript, lo que hacía que fuera muy difícil de extender por alguien que quisiera añadir funcionalidades.

Con la migración se logran también otras ventajas, como por ejemplo:

- La aplicación web pasa a ser compatible con la gran mayoría de los dispositivos.
- El desarrollo es mas sencillo ya que HTML5 cuenta con una comunidad y documentación inmensa.
- No se ve afectado por las acciones de ninguna empresa privada (Adobe abandonando Flex, por ejemplo).
- Se unifican todas las herramientas necesarias para el desarrollo de una aplicación web como Babelium en HTML5 y no se depende de tecnologías privadas de terceros

---

OBJETIVOS

---

1. Objetivos del proyecto

Este proyecto cubre los siguientes objetivos:

- Analizar las distintas funcionalidades de Babelium Project para su migración a HTML5.
- Elección de las herramientas apropiadas para llevar a cabo la migración.
- Realizar una documentación exhaustiva sobre lo desarrollado durante el proyecto.
- Obtener una aplicación web dinámica de una sola página basada en HTML5 y Javascript con sus funcionalidades operativas.
- Migrar la aplicación a HTML5 utilizando un framework de JavaScript.

2. Objetivos personales

En este proyecto, a parte de los objetivos mencionados anteriormente, yo, como programador, tengo unos objetivos personales que quiero desarrollar a lo largo del proyecto:

- Optimización, si es posible, de las funcionalidades de la aplicación web respecto a la versión en Flex.
- Dominio del desarrollo de aplicaciones web de una sola página.
- Obtener conocimientos sobre los distintos frameworks de Javascript disponibles.
- Ser capaz de llevar a cabo la implementación de un proyecto y documentarlo adecuadamente de forma independiente.
- Lograr que este trabajo sirva de referencia tanto para mí como para otros desarrolladores de aplicaciones similares.

ARQUITECTURA

1. Arquitectura del código

Debido a la elección de Backbone.js como framework de la implementación del proyecto, su arquitectura consiste en un Modelo-vista-controlador, el cual consiste en dividir en tres componentes distintos los datos (Modelo), la lógica de negocio (Controlador) y la interfaz de usuario (Vista).

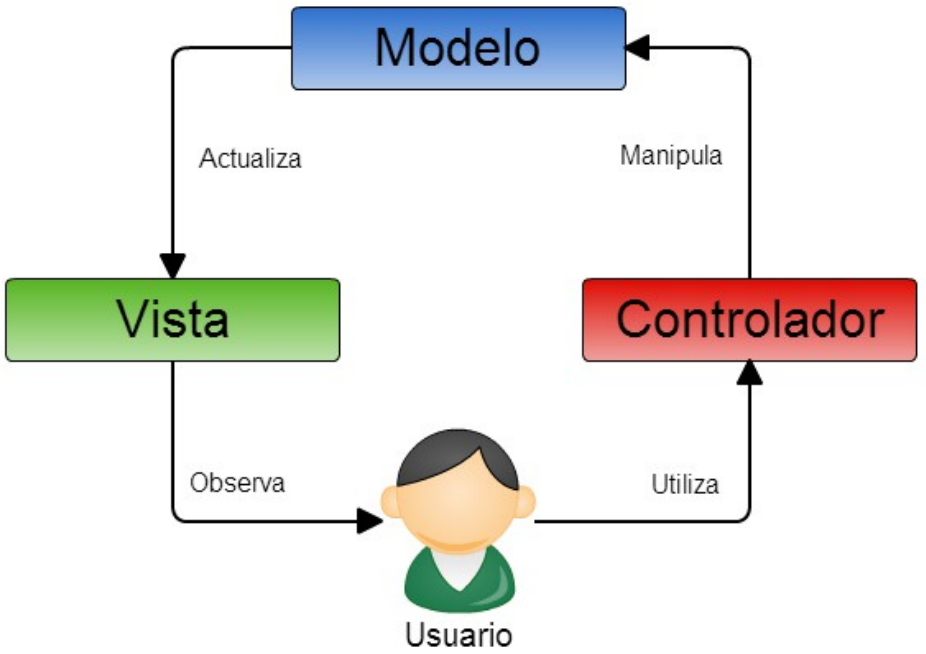

Ilustración 1: Modelo MVC

Su funcionamiento consiste en que el usuario interactúa con la interfaz, el controlador recibe la acción realizada y gestiona el evento que se desencadena con un gestor de eventos. Después, el controlador entra en contacto con el modelo, realiza los cambios necesarios y actualiza la interfaz de usuario, quedando en espera de la próxima interacción.

Además, se busca que la aplicación web sea de una sola página con elementos dinámicos. Esto es, que todas las funcionalidades de la aplicación web estén disponibles en una sola página a través de botones y menús dinámicos en los que se pueda navegar sin necesidad de estar cargando una nueva página por cada sección de la aplicación.

2. Arquitectura de la aplicación web

La arquitectura empleada en este proyecto es la de Cliente-Servidor. Esta arquitectura divide la aplicación en 2 partes:

- El cliente es la parte utilizada por los usuarios que les permite realizar solicitudes al servidor.
- El servidor se encarga de procesar las solicitudes de los clientes y les responde con un resultado.

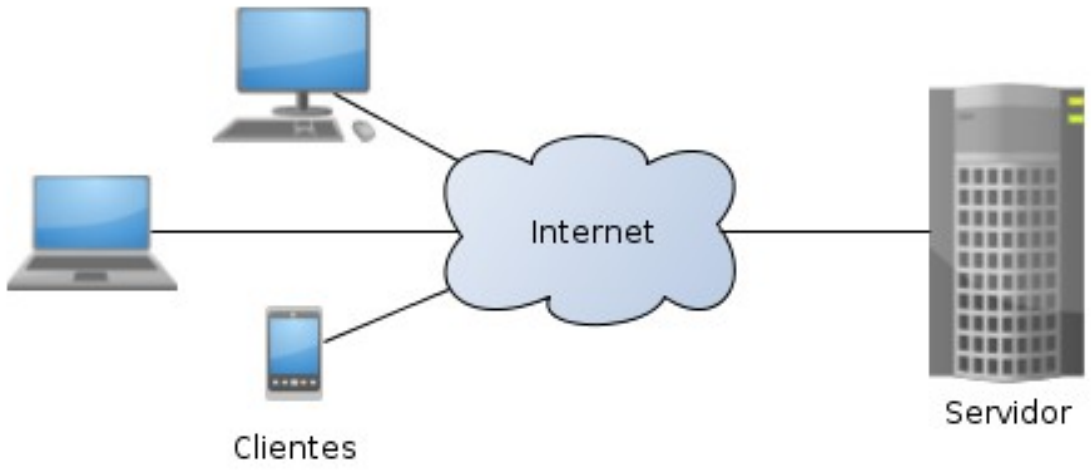

Ilustración 2: Arquitectura cliente-servidor

Esta arquitectura permite que los usuarios hagan uso de la aplicación sin necesidad de instalar ningún software adicional.

Para llevar a cabo la comunicación entre el cliente y servidor, la aplicación utiliza la tecnología PHP, que combinada con un servidor web Apache y una base de datos MySQL, permite al servidor responder a todas las solicitudes de los clientes.

Cliente

La parte de cliente, al estar basada en Backbone.js, adquiere la siguiente estructura:

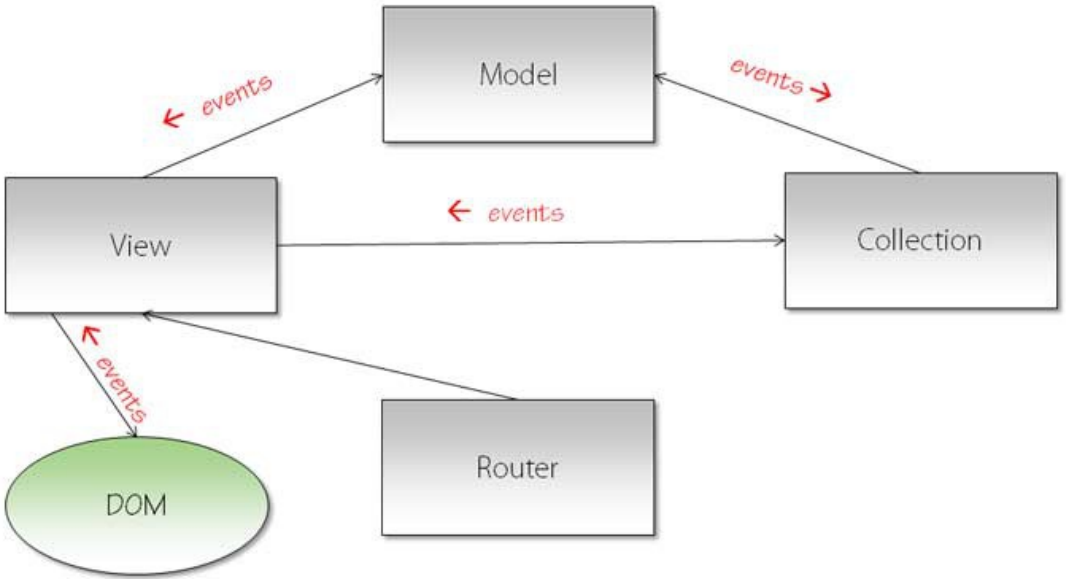

Ilustración 3: Arquitectura Backbone.js

Sigue una estructura MVC en la que la comunicación entre las distintas partes se realiza mediante eventos. Los resultados de esos eventos se muestran mediante las vistas a los usuarios.

Servidor

Para explicar la estructura del servidor, lo mas sencillo será hacerlo mediante el siguiente esquema:

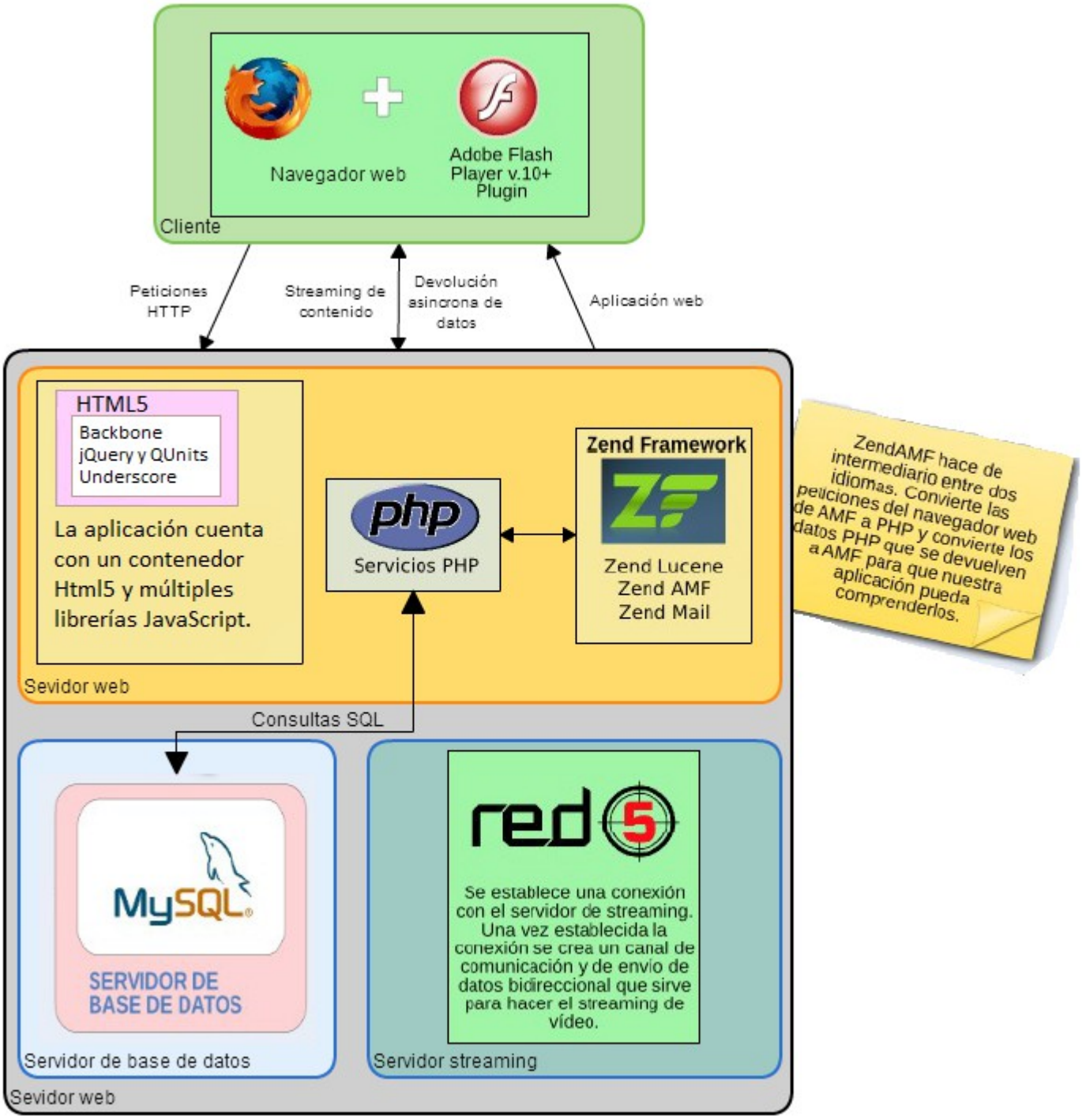

Ilustración 4: Arquitectura del servidor

La plataforma Babelium hace uso de diversas tecnologías en el lado del servidor, como el servidor HTTP Apache, el servidor de *streaming* Red5 o el servidor de bases de datos MySQL.

Por otro lado, gran parte de la parte servidor se centra en el conjunto de servicios de Babelium implementados como scripts PHP. Estos scripts permiten, por una parte, comunicar e interpretar las acciones del cliente

JavaScript con la lógica de negocio del servidor y, por otra parte, ejecutar distintas tareas de manipulación de archivos de audio y video relacionados con la grabación de ejercicios y respuestas de los usuarios.

HERRAMIENTAS

1. Herramientas de aprendizaje

Todo proyecto requiere una fase previa de aprendizaje y formación. Este paso es facilitado gracias a la ayuda de Google que nos permite buscar toda la información que queramos rápidamente pero los que tienen mayor mérito son los que aportan esa información a Internet.

En este apartado me gustaría dejar constancia de aquellas personas/webs que más han aportado a mi aprendizaje sobre lo trabajado durante el desarrollo del proyecto.

1. http://addyosmani.github.io/backbone-fundamentals/

   En esta web podemos acceder al libro "Backbone fundamentals" en el que se explica de forma detallada el framework Backbone. Este libro lo empecé a utilizar como referencia una vez ya había comenzado a utilizar Backbone y ya conocía los fundamentos sobre su uso.

   Gracias a los múltiples ejemplos que contiene, gran parte de las ideas sobre el diseño y estructura de la aplicación provienen de él.

2. https://github.com/babeliumproject/html5-standalone-site

   En este repositorio se encuentra un diseño inicial de la aplicación web Babelium utilizando Html5. Menciono este repositorio porque al principio del proyecto, cuando tenia las manos vacías, me mostraba un ejemplo de lo que debía lograr y el diseño aproximado de la aplicación.

   Disponer de este repositorio me permitió empezar el proyecto con una idea clara de lo que debía lograr y me facilitó la transformación de la aplicación a Html5 para poder centrarme cuanto antes en las funcionalidades con Backbone.

3. https://www.youtube.com/user/DevCodela

   Al principio del proyecto, cuando no conocía Backbone, decidí buscar un canal de Youtube que ejerciera de profesor, para agilizar mi toma de contacto con el framework. Fue una decisión acertada, ya que comprendí los conceptos básicos bastante mas rápido que si los hubiera leído de un manual.

4. http://codebeerstartups.com/2012/12/a-complete-guide-for-learning-backbone-js/

   Esta web contiene tutoriales con muchos ejemplos sobre el uso de Backbone parte por parte. Antes de conocer la existencia del libro mencionado en el punto 1 de este apartado, buscaba ejemplos sobre el uso de los elementos de Backbone aquí pero una vez que las dudas que surgían se volvían mas complejas me dejó de ser de utilidad.

   El dueño del tutorial ha hecho un magnifico trabajo con su guía que permite aprender a utilizar Backbone a un nivel básico de forma rápida y sencilla.

5. Mi tutor Juanan Pereira

   Juanan me ha ayudado a lo largo de toda la duración del proyecto, de principio a fin. No solo ha supervisado mi trabajo, también me ha ayudado en la toma de decisiones desde un punto de vista objetivo, permitiéndome pensar por mi mismo y desarrollar mis propias soluciones.

2. Herramientas de documentación

Es importante realizar una buena elección de herramientas de documentación a la hora de llevar a cabo un proyecto. La claridad del mismo depende ello.

LibreOffice

En mi caso, me he decantado por *LibreOffice* ya que es el software de ofimática con el que más tiempo he tratado, es gratuito y además tiene todas las funciones que necesito para crear la documentación de un proyecto adecuadamente.

Ilustración 5: LibreOffice

Además del editor de texto, hacen falta más herramientas para la documentación. Un ejemplo es el software para realizar diagramas y esquemas, para ello yo he elegido los siguientes:

Cacoo

Para la creación de diagramas y esquemas para uso personal, durante el proyecto he utilizado *Cacoo*, una herramienta de dibujo online útil que ayuda a realizar esquemas y

Ilustración 6: Cacoo

diagramas de todo tipo, pudiendo acceder a él a través de Internet y guardar mis esquemas online.

Tom's planner

Para realizar el esquema inicial de organización del proyecto he utilizado un software online llamado *Tom's planner* que me ha facilitado la tarea de hacer un diagrama de Gantt, ya que con otras herramientas es bastante mas complejo. Con esta herramienta simplemente nos da la plantilla y permite ajustar todos los parámetros del diagrama para dejarlo a gusto del usuario.

Ilustración 7: tom's planner

Visual Paradigm

Para los diagramas relacionados con la implementación del proyecto incluidos en esta documentación he utilizado *Visual Paradigm.* Es una aplicación que he utilizado durante los 2 últimos años de mi grado y estoy familiarizado con ella.

Es simple, práctica e incluye todo lo necesario para realizar diagramas de clases, casos de uso, secuencia, etc.

Ilustración 8: Visual Paradigm

3. Herramientas de implementación

Durante la implementación del proyecto, he hecho uso de múltiples herramientas, de las cuales la mayoría, o las había usado alguna vez suelta o bien no las conocía en absoluto. Este proyecto me ha ayudado a darme cuenta de lo necesarias y frecuentes que son muchas de ellas.

NetBeans

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Además cuenta con una importante cantidad de extensiones que permiten a los usuarios tener acceso a opciones que el software original no permite.

Ilustración 9: NetBeans

La plataforma NetBeans permite que las aplicaciones basadas en ella puedan ser extendidas fácilmente por otros desarrolladores de software.

En principio, tenía previsto usar Eclipse, una herramienta muy similar a NetBeans, ya que es la herramienta que hemos utilizado a lo largo del grado debido a las indicaciones de nuestros distintos profesores, pero me he decantado por NetBeans.

Este cambio de opinión surge a partir de la experiencia adquirida en unas practicas realizadas en una empresa durante los meses en los que he trabajado el proyecto. Descubrí NetBeans y aprendí a usarlo para adaptarme a las necesidades de la empresa y me pareció más cómoda que eclipse y con mas posibilidades.

SublimeText3

Es un editor de texto orientado a programación y es simple y fácil de utilizar. Resalta con distintos colores los conjuntos de caracteres en función del tipo de elemento y de fichero que se este modificando, facilitando así su lectura.

Ilustración 10: SublimeText3

Esta herramienta la he utilizado al final de la implementación, como alternativa a NetBeans. Esto se debe a que, a partir del momento en el que empecé a trabajar con el servidor de Babelium para acceder a información de usuarios y vídeos, surgieron ciertas incompatibilidades entre el servidor que genera NetBeans para alojar mi código y el servidor remoto en el que se encuentra Babelium. Tras intentar encontrar una forma de arreglar el problema decidí seguir adelante usando esta herramienta junto con *WAMP* para poder usar mi aplicación en la web. Resultó ser la alternativa mas cómoda.

WAMP

Es un software que permite crear nuestro propio servidor http Apache, usado principalmente para enviar páginas web estáticas y dinámicas a la red.

Ilustración 11: WAMP

Gracias a esta aplicación ha sido posible hacer las pruebas y simulaciones necesarias de comunicación entre la aplicación web y el servidor de Babelium.

La razón por el que ha sido usada es la previa experiencia con ella en algunas asignaturas. Se monta el servidor en 1 clic y ya puedes empezar a

usarlo, no es necesario buscar otras herramientas cuando wamp no supone ningún problema.

Digital Ocean

Es un proveedor estadounidense de servidores virtuales privados. No solo dispone de servidores en los EE.UU., también dispone de otros centros en lugares como Londres, Singapur, San Francisco o Amsterdam.

Ilustración 12: Digital Ocean

El papel que juega Digital Ocean en el proyecto consiste en mantener en línea el servidor que contiene y ejecuta el API de Babelium. Este servidor es el que recibirá todas las consultas realizadas por los usuarios y el encargado de, tras ejecutar los procesos necesarios, devolver una respuesta.

En principio, lo que se consigue con Digital Ocean se podría crear de otra forma, como por ejemplo, montando una máquina virtual en la que se carga el API de Babelium. De esta manera sería gratuito, tendría acceso a la máquina ya que se estaría ejecutando desde el mismo equipo y estaría a mi entera disposición cuando quisiera.

A pesar de todas estas ventajas, no se ha hecho así por múltiples razones que fueron surgiendo en las reuniones con Juanan:

1. Montar el API de Babelium puede resultar complejo si no se tiene experiencia previa.

2. El tutor necesita tener acceso al servidor para poder asistir con los problemas que puedan surgir y para manipular los ficheros con libertad. Siendo de sobremesa el equipo en el que se ha trabajado, supondría más problemas que ventajas.

3. El disco duro del equipo donde se ha desarrollado el proyecto se estropeó durante la etapa en la que se estaba montando el servidor, al no haber logrado que funcionase, no existían copias de seguridad de la máquina virtual.

Todas estos problemas en conjunto, convirtieron la idea de alquilar un servidor en algo aconsejable.

PuTTY

Para poder conectarse al servidor alquilado en Digital Ocean, es preciso utilizar un protocolo de conexión remota y un método de autentificación. El protocolo que se ha elegido para ello es SSH y como método de autentificación seguro se ha decidido el uso de claves RSA.

Para poder hacer todo esto, se ha utilizado la herramienta PuTTY, que es un cliente de SSH (entre otros) y a su vez, permite la creación de claves RSA para su uso en las conexiones mediante SSH.

Ilustración 13: PuTTY

Gracias a esta herramienta, se dispone de una clave privada y una pública. La clave pública fue compartida con el tutor del proyecto para permitirle conectarse al servidor también y la privada se encuentra en el equipo en el que se está desarrollando el proyecto.

Para poder conectar con el servidor alquilado, primero se debe autorizar el acceso al servidor desde la página web de Digital Ocean. Para ello, se debe compartir con Digital Ocean la clave pública. Una vez hecho esto, ya se puede acceder al servidor con la clave pública y la privada.

MySQL Workbench

Este proyecto dispone de una base de datos en la que se almacena toda la información de los usuarios, los ejercicios, las evaluaciones, etc.

Esta base de datos es MySQL y se encuentra alojada en el servidor alquilado en Digital Ocean, es decir, para usarla es necesario establecer una conexión con el servidor.

A lo largo del desarrollo de proyecto es necesario revisar la base de datos, ya sea para realizar pruebas, saber que datos se almacenan o gestionar los datos guardados. Para ello, se podría utilizar la misma consola de PuTTY y acceder a la base de datos, pero, para facilitar las cosas y dar mayor movilidad, se decidió usar una herramienta visual de gestión de bases de datos. En este caso, esa herramienta es MySQL Workbench.

Ilustración 14: MySQL Workbench

MySQL Workbench es una herramienta visual de diseño de bases de datos, nos permite hacer todo lo que haríamos si solo hubiera una consola pero mediante una interfaz gráfica. Esto facilita el manejo de la base de datos.

Esta herramienta ha sido utilizada principalmente durante las pruebas, para observar el comportamiento de los datos una vez se envían al servidor y borrar elementos innecesarios. Por ejemplo, durante la implementación de la funcionalidad de registro de usuarios de la aplicación, había que eliminar los intentos fallidos de la base de datos manualmente y para poder seguir haciendo pruebas.

Esta herramienta también dispone de un cliente SSH, mediante el cual se puede acceder a la base de datos del servidor directamente usando una clave pública o privada.

Babelium API

Se trata del conjunto de funciones y procedimientos que emplea la parte del servidor de la aplicación a desarrollar en este proyecto. El proyecto depende en gran medida de ella.

Ilustración 15: Babelium

Esta herramienta es la encargada de recibir, interpretar y responder las consultas de los usuarios de Babelium. La API de Babelium se encuentra alojada en el servidor alquilado de Digital Ocean y mediante PuTTY se accede a ella en caso de necesitar modificarlo.

Cuando se estableció la API en el servidor alquilado, ésta no contaba con algunos de los servicios necesarios para que el proyecto funcionase. El tutor del proyecto se ha encargado de proporcionar los servicios que ya existían en la versión de Flash, mientras que yo estaba a cargo de modificarlos para ajustarlos a las necesidades del proyecto.

Los servicios de los que dispone el API de Babelium se describen en el Anexo 1.

Qunit

Toda buena aplicación debe haber pasado por una serie de pruebas y tests que permitan asegurar que funciona correctamente en todas las situaciones. Estas pruebas pueden ahorrar una cantidad de tiempo importante localizando errores y además se pueden reutilizar en el futuro en caso de que se vayan a realizar cambios en la aplicación.

Para realizar pruebas con la aplicación desarrollada en este proyecto se ha decidido utilizar QUnit, que es un potente y sencillo framework JavaScript de pruebas unitarias.

Ilustración 16: QUnit

Para realizar las pruebas, solo se necesita descargar los ficheros js y css del framework, crear un archivo html con una estructura proporcionada por QUnit y obtenemos una página en la que se mostrarán los resultados de las pruebas que se implementen.

Las pruebas realizadas se basan en ejecutar una función de la aplicación y pasar como parametro a QUnit el resultado esperado de la función, si este resultado cumple la aserción realizada en la prueba entonces se muestra la función junto a un cuadro verde en la página creada para QUnit, en caso contrario, se mostrará un cuadro rojo y la explicación del error.

Control del Proyecto

Todo proyecto está dividido en distintas fases: La introducción, el aprendizaje, la puesta en marcha, la planificación y la ejecución de la misma.

Habiendo visto ya la introducción y con el proyecto avanzando, en este apartado vamos a tratar los distintos puntos de la planificación y control del proyecto. Con este apartado se quiere mostrar al lector el esfuerzo requerido para realizar el proyecto y la distribución de las tareas a lo largo del tiempo.

1. Alcance

A continuación se muestra el diagrama EDP (Estructura de Descomposición del Proyecto) en el que se encuentran las tareas a realizar a lo largo del proyecto. Más abajo se concreta en que consiste cada tarea mencionada:

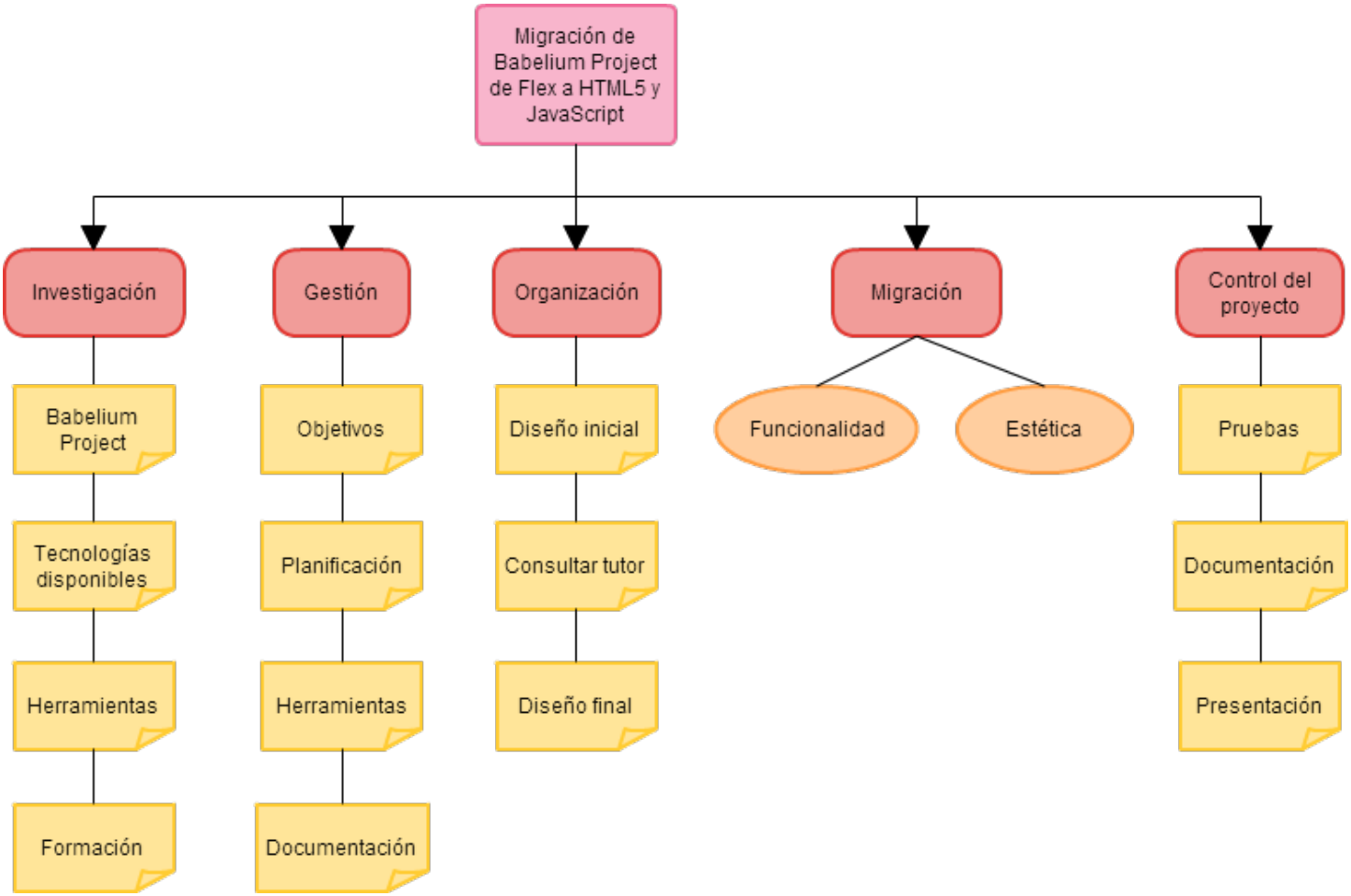

Ilustración 17: Diagrama EDP

Investigación

Durante la investigación la prioridad era recopilar información sobre todo lo que pudiera ser de utilidad durante el desarrollo del proyecto.

Lo primero sobre lo que se ha investigado es Babelium Project. Es una aplicación web existente sobre la que se trabaja, por lo que es totalmente necesario conocer todos los detalles sobre ella.

Conociendo mejor Babelium Project y sus funcionalidades, el nuevo objetivo era investigar las tecnologías disponibles, ya que Juanan, planteó el uso de ciertas tecnologías para el desarrollo del proyecto y se debía considerar si alguna de estas era apropiada o había alguna mejor.

Estas tecnologías son los frameworks MVC de javascript más conocidos a día de hoy, que son las siguientes:

- Ember
- Backbone
- Angular
- Knockout

Tras realizar una investigación inicial sobre cada uno de estos frameworks, se llegó a la conclusión de que Knockout no es una buena opción para este proyecto. Knockout se basa más en enlazar datos con elementos de la aplicación web, cosa que se puede conseguir con jQuery, librería que ya se sabía que se iba a utilizar desde un principio.

De esta forma, la investigación se centro en los 3 frameworks con mas fuerza en la actualidad y sobre los que más debate hay: Angular, Backbone y Ember.

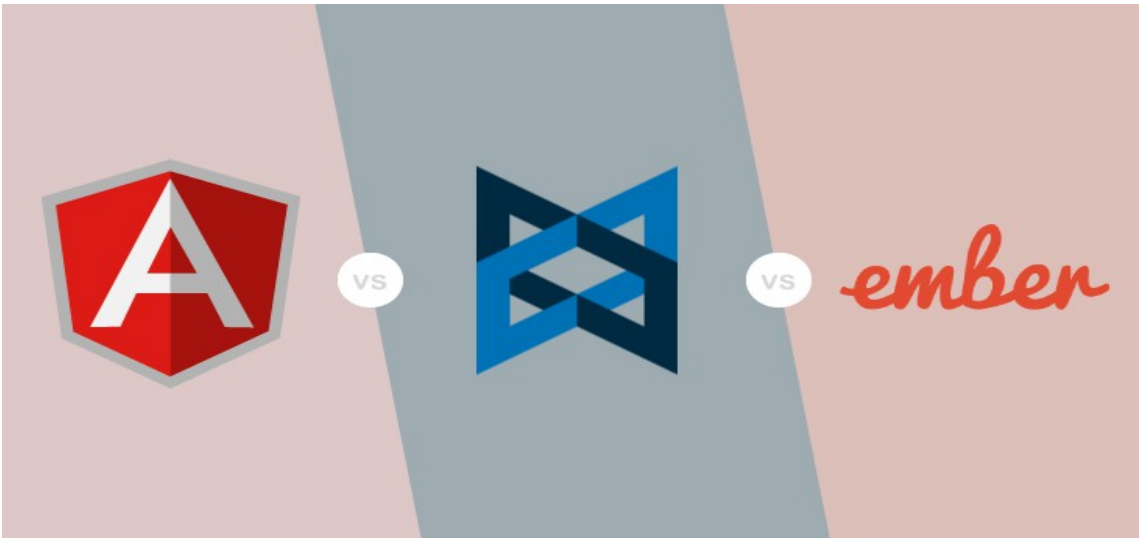

Ilustración 18: Angular vs Backbone vs Ember

Al investigar los 3 frameworks MVC de Javascript pude aprender el funcionamiento básico de cada uno de ellos y que me podría aportar cada uno. He comprobado el gran debate que surge en todos los grupos de trabajo a la hora de seleccionar el framework adecuado para sus proyectos y aprendido los criterios a tener en cuenta.

La siguiente tabla contiene las características que se han tenido en cuenta a la hora de tomar la decisión:

	Antiguedad	Comunidad	Velocidad	Peso	Aprendizaje	Proyectos
Ember	4 años	Pequeña	Alta	Medio	Complejo	Medianos y grandes
Angular	3 años	Amplia	Muy alta	Alto	Muy complejo	Medianos y grandes
Backbone	5 años	Amplia	Alta	Muy bajo	Sencillo	Pequeños y medianos

Tabla 2: Comparación frameworks

Con esta información resumida se puede comprender fácilmente la razón por la que me decanté por Backbone. La aplicación web a desarrollar no es lo suficientemente grande y compleja como para que merezca la pena utilizar Angular o Ember, los considero los 2 mejores frameworks y con mas posibilidades de cara al futuro, pero Ember es demasiado nuevo y no tiene una comunidad lo suficientemente amplia y Angular es un framework muy

pesado y complejo. Backbone es más sencillo, está bastante completo, me aporta una mayor transparencia en el código y dispone de una comunidad lo suficientemente grande como para poder aprender rápidamente. Tiene todo lo necesario para cumplir los objetivos del proyecto adecuadamente.

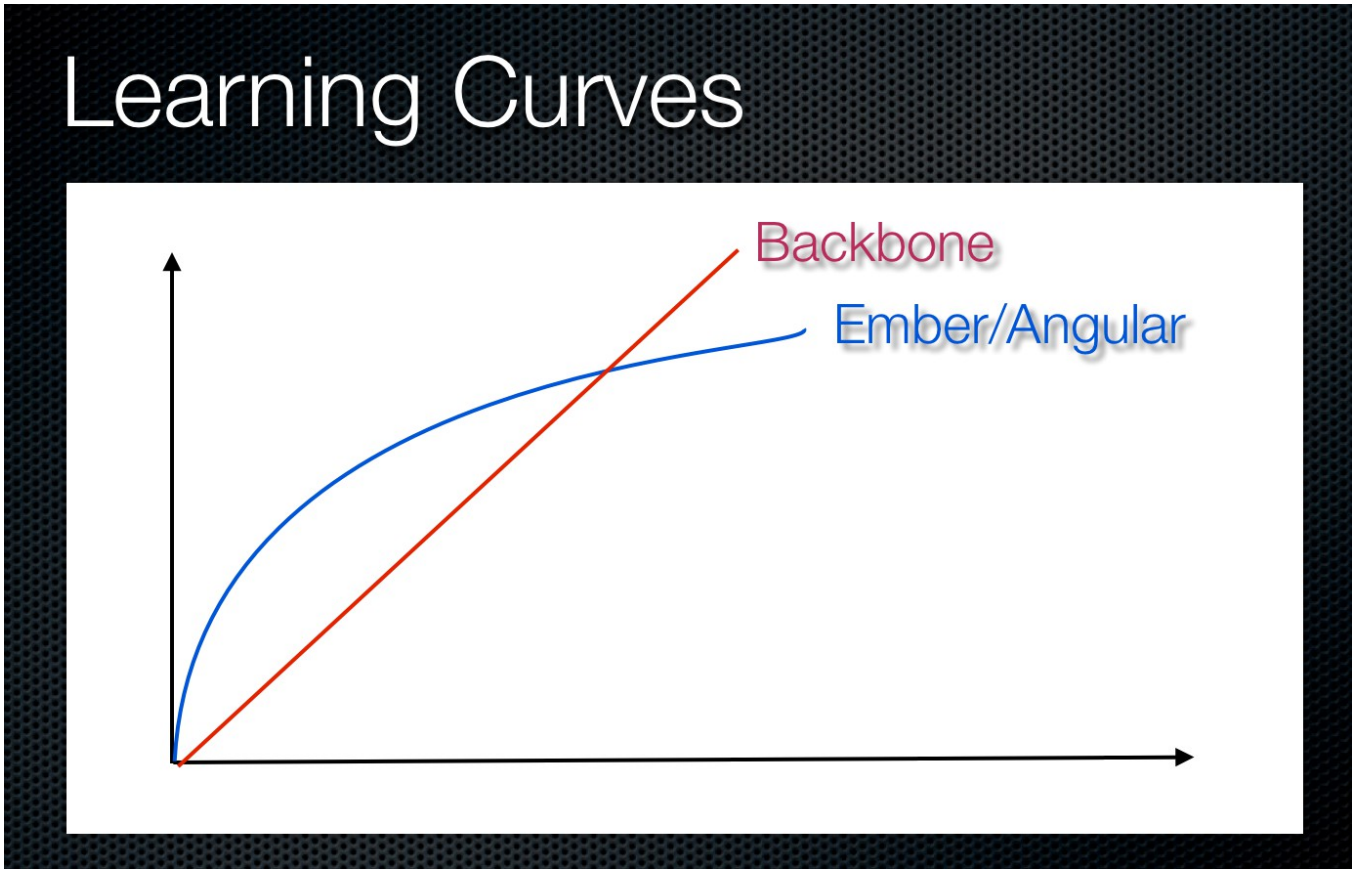

Ilustración 19: Curvas de aprendizaje Backbone vs Ember/Angular

Aquí se muestra un gráfico sencillo para representar las curvas de aprendizaje de los 3 frameworks, se ve que Backbone es el mas fácil de aprender.

Una vez investigadas las tecnologías disponibles, la elección de las herramientas a utilizar se convirtió en el siguiente objetivo, aunque, con toda la investigación realizada y la experiencia adquirida a lo largo de los años estudiando en la UPV/EHU, me decanté enseguida por las expuestas ya en el apartado de herramientas de este documento.

Por último, se ha dedicado una gran parte del tiempo empleado en esta etapa a la formación sobre las herramientas y tecnologías que se van a utilizar, como se van a utilizar y para que.

Gestión

En esta etapa me he dedicado a realizar la documentación inicial del proyecto, estableciendo las bases, los distintos apartados a tratar y decidiendo que voy a incluir en cada apartado.

Este apartado no tiene ninguna parte importante porque el apartado en sí es uno de los mas importantes en todos sus puntos.

Primero, he establecido los objetivos del proyecto junto con mi tutor para saber que puntos voy a tener que cubrir. Es necesario marcar los objetivos porque, sin ellos, la toma de decisiones carece de sentido. La lista de los distintos objetivos del proyecto se encuentra en el punto de la documentación de objetivos del proyecto.

A partir de estos objetivos, es necesario fijar una planificación adecuada para llevarlos acabo de forma organizada. Para ver la planificación de este proyecto en detalle ir al apartado de planificación dentro del control del proyecto.

Durante la gestión me dedico a asegurarme de que dispongo de las herramientas necesarias para que el proyecto cumpla los objetivos fijados y a reflejar la información necesaria sobre las mismas en este documento.

La documentación del proyecto está presente en todo momento, tomando nota de todos los acontecimientos, progresos e ideas que surjan. Todas las herramientas usadas y todo lo relacionado con la realización del proyecto debe quedar reflejado en este documento.

Organización

Durante esta etapa me he encargado de dar forma al proyecto. Obtengo una versión preliminar del resultado y voy rellenando huecos teniendo en todo momento al tutor informado sobre como quiero realizar los cambios.

Una vez obtenido el diseño final, es el momento de empezar con la migración a HTML5.

Migración

En esta fase, entro ya con todo preparado, simplemente me he dedicado a llevar a cabo la migración, realizando todos los cambios y anotaciones pensadas durante la planificación y el diseño final. Al terminar de migrar cada funcionalidad se mantiene informado a Juanan y se consulta con él para ver si está satisfecho con el resultado del trabajo.

Control del proyecto

Una vez terminadas todas las demás fases, he realizado todas las pruebas necesarias para comprobar que la aplicación web funciona correctamente y que se han cumplido todos los objetivos establecidos al comienzo del proyecto.

Como ultimó paso de todos queda terminar la documentación del proyecto con los resultados obtenidos y preparar su presentación.

2. Metodología

Para este proyecto se ha utilizado una metodología en la que el trabajo se encuentra dividido en paquetes y se presenta en prototipos. Cada paquete contiene una carga de trabajo calculada en horas y los paquetes no tienen que ser necesariamente iguales.

Primero, tras estudiar a fondo la antigua aplicación web Babelium, se crea una lista que incluya todos los pasos a realizar para terminar el proyecto. En esta lista deben anotarse primero las tareas en bloques grandes, para luego dividirlos en bloques de tareas mas pequeños, así sucesivamente hasta obtener tareas que no puedan ser divididas.

Cuando se tiene la lista de las tareas indivisibles, se agrupan en paquetes de trabajo. En este caso, se intentó obtener paquetes que tuvieran tareas o bien similares o bien relacionadas, para no dejar alguna funcionalidad sin completar al terminar un paquete.

Con la división en paquetes de trabajo terminada y esquematizada para su fácil visualización, cada semana se selecciona un paquete cuya carga de trabajo equivale alrededor de 15 horas. Por lo que la intención inicial siempre ha sido terminar un paquete por semana o cada 2 semanas en función de las circunstancias.

Al final de cada semana, independientemente de que el paquete seleccionado esté terminado, hay que reunirse con el tutor para poder hacer el seguimiento del trabajo. Los progresos se muestran mediante prototipos, que consisten en la ultima versión de la aplicación web con las funcionalidades del paquete elegido incorporadas.

En caso de que, ya sea por falta de tiempo o por cualquier otro motivo, un paquete, o parte de él, no se completa en la semana designada, éste pasará a formar parte de la tarea de la semana siguiente con la mayor prioridad. Es decir, si una tarea queda incompleta deberá ser completada antes de empezar con el próximo paquete durante la semana siguiente.

A continuación, se incluye un diagrama para facilitar la comprensión de esta metodología.

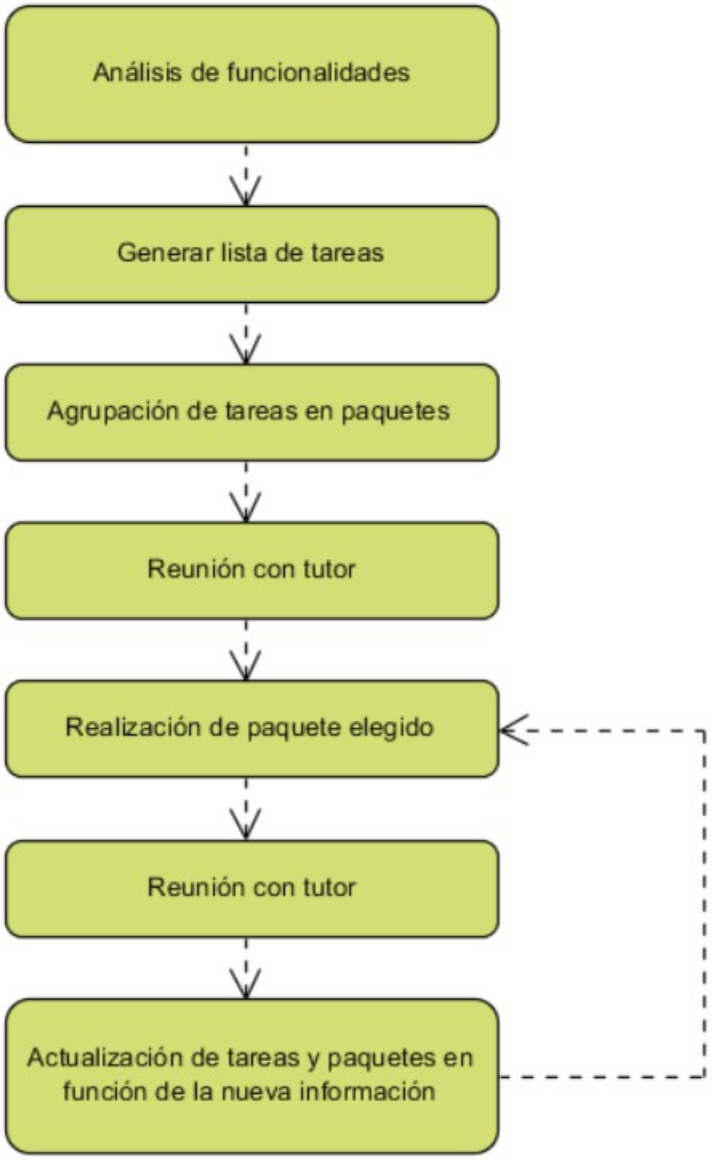

Ilustración 20: Diagrama metodología de trabajo

Es muy simple y habrá tantas iteraciones como paquetes salgan, sin olvidar que puede ser que se formen paquetes nuevos en cada entrega de un prototipo.

Las ventajas que tiene este sistema es la gran capacidad de respuesta ante imprevistos, la flexibilidad a la hora de realizar las tareas semanales y que se pueden observar los resultados a través de los prototipos generados con cada iteración.

La desventaja principal de este método es que es difícil de documentar. Si surgen imprevistos las decisiones se toman al momento y se documenta más tarde cuando el asunto ya está mas frío.

3. Planificación

Debido a la metodología escogida en el punto anterior, establecer una planificación se hace algo más difícil de lo normal, ya que la velocidad de trabajo puede variar cada semana en función de los problemas que puedan aparecer y las tareas elegidas.

He creado un diagrama de Gantt para tratar de describir la duración de cada sección establecida en el diagrama EDP de la descripción del alcance del proyecto.

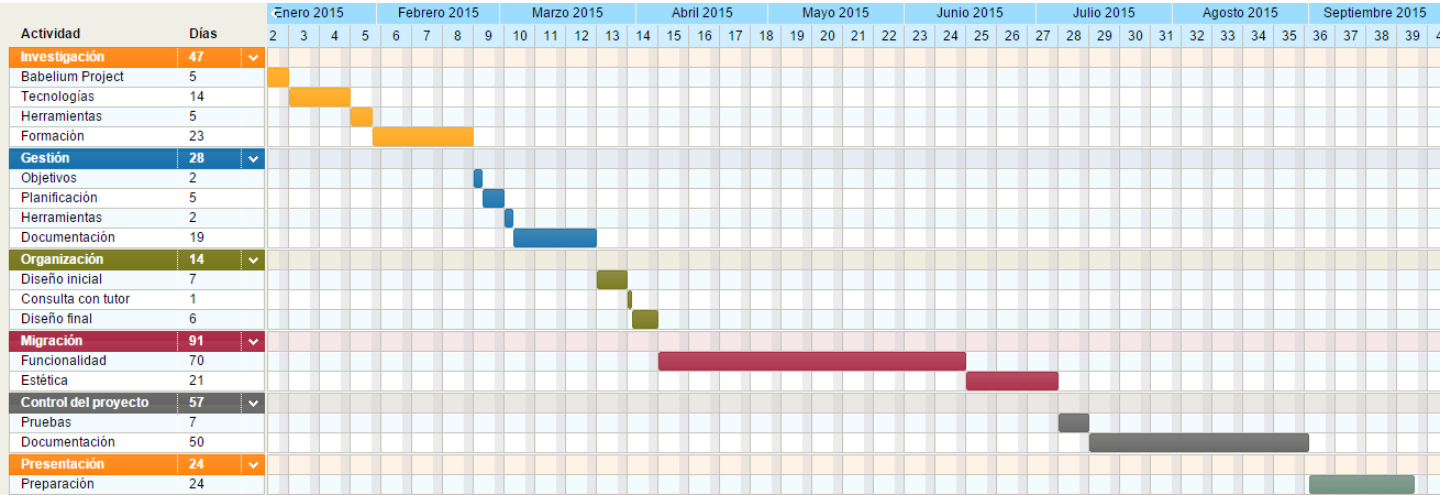

Ilustración 21: Diagrama de Gantt inicial

Este diagrama contiene la primera estimación que he realizado de la duración de las distintas partes del proyecto. Forma parte de la planificación inicial previa al comienzo de la implementación.

Debido al desconocimiento de algunas de las tecnologías a utilizar y la variedad de funcionalidades de la aplicación web a desarrollar, las duraciones de las secciones de investigación e implementación son mayores y abarcan gran parte de la duración del proyecto.

Además, teniendo en cuenta que algunas secciones del proyecto podrían tener una duración mayor que la esperada, mi tendencia siempre ha sido prolongar las duraciones un poco mas allá de mi estimación.

Según esta estimación de tiempo podemos calcular la cantidad de horas de trabajo que supone realizar el proyecto. Para ello debemos tener en cuenta lo siguiente:

Desde enero hasta agosto he estado de practicas en una empresa trabajando de lunes a viernes, por lo que por cada día de entre semana estimo una media de 1 hora de trabajo, influenciado también por días en los que, por cualquier razón, no haya podido trabajar en el proyecto.

Los fines de semana se estima una media de trabajo de 3 horas por día.

Esto supone que, siendo la planificación del proyecto de una duración de 261 días de los cuales 187 pertenecen al grupo de entre semana y 74 al grupo de fin de semana, tenemos:

$$(187\times1)+(74\times3)=409\,Horas$$

A continuación, se muestra el diagrama de Gantt que representa los tiempos reales que ha tomado realizar cada parte del proyecto para poder contemplar la diferencia de lo estimado con lo real.

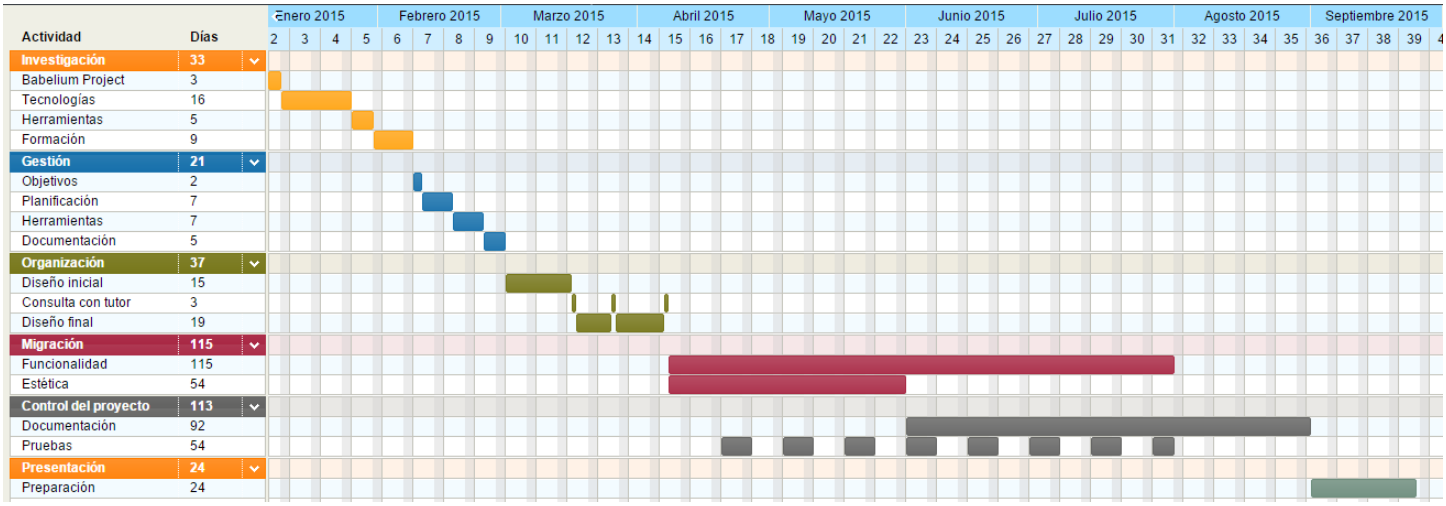

Ilustración 22: Diagrama de Gantt real

Este es el diagrama realizado durante la etapa final del proyecto, una vez ya familiarizado con el proyecto y sus tareas. Representa las etapas del proyecto tal y como han sido.

El número de horas de trabajo es prácticamente el mismo, ya que las fechas de comienzo y fin son las mismas en ambos casos y el ritmo de trabajo a sido el estimado. A pesar de que durante la migración se realizan múltiples tareas simultáneamente, esto no afecta a la cantidad de horas diarias ya que la duración de estas tareas es ampliada en varios días respecto a la planificación inicial.

No es ninguna sorpresa que la etapa de migración sea la mas abultada. Esto se debe a que contiene todo el trabajo de implementación de la aplicación.

Como se puede comprobar, la implementación está dividida en dos grandes partes, estética y funcionalidad. Ambas partes se desarrollan al mismo tiempo al principio, pero la estética llega un punto en el que se deja

preparada y no se vuelve a cambiar, pasando así a centrarse completamente en las funcionalidades restantes.

Durante la implementación, a parte de realizar una rutina de pruebas periódicas para comprobar que todo va como debe, también se desarrolla la documentación ya que los avances deben quedar reflejados en ella.

Podemos observar una clara diferencia en todos los sentidos entre los dos diagramas, lo cual demuestra que no fui capaz de comprender la dificultad de cada paso al principio, pero si identifiqué correctamente las tareas que me llevarían mas tiempo y me darían mas problemas.

Las 2 principales diferencias entre la planificación inicial y la planificación real son:

- Durante la planificación inicial se consideró que las etapas de preparación e investigación abarcarían una mayor cantidad de tiempo al comienzo del proyecto, pero no fue así. Gracias a la experiencia con las herramientas utilizadas y la sencillez de Backbone se empezó a implementar enseguida.

- En la planificación inicial no se consideró la necesidad de combinar algunas tareas durante el desarrollo del proyecto, como por ejemplo, la realización de pruebas y documentación mientras aún se está implementando. Había una visión de las tareas lineal.

4. Control

Teniendo en cuenta los problemas que puede suponer la metodología elegida, mi tutor me sugirió que, como ya he explicado en el apartado de metodología, dividiera todas las tareas que tengo que realizar en tareas más pequeñas y crease paquetes de trabajo para realizar en un tiempo estimado por mí.

Así pues, realicé un listado de las distintas tareas a realizar durante la implementación de la aplicación web y las dividí en grupos pequeños estimando una duración de una semana por cada paquete:

Paquetes de trabajo

PAQUETE 1	PAQUETE 2	PAQUETE 3
Documentar puntos previos a la implementación	Implementar login/registro/logoff	Implementar página de ayuda y tutoriales
Practica con Backbone	Implementar pagina de datos del usuario	Implementar los tabs de la aplicación
Realizar el calendario de tareas	Mirar que se puede aprovechar del prototipo (Mintzabel)	Documentar lo realizado
	Documentar lo realizado	

PAQUETE 4	PAQUETE 5	PAQUETE 6
Implementar filtro de vídeos	Opciones relacionadas con hacer un ejercicio	Opciones relacionadas con valorar un ejercicio
Implementar subida de vídeos	Documentar lo realizado	Documentar lo realizado
Implementar las opciones de idioma		
Documentar lo realizado		

PAQUETE 7	PAQUETE 8	PAQUETE 9
Realizar pruebas	Opciones relacionadas con subtitular un ejercicio	Implementar sistema de créditos
Corrección de fallos	Documentar lo realizado	Implementar idiomas
Terminar documentación	Implementar acceso al blog	Documentar lo realizado

Ilustración 23: Distribución de tareas inicial

Supuse que cada paquete del 1 al 6 incluido, tendría una duración de una semana y se realizarían en el mismo orden de los paquetes. Después estaban el paquete número 7 para el cual no fijé ninguna duración concreta ya que depende completamente de si funciona todo como debe o no, además era el paquete que mas carga de trabajo tenía respecto a la documentación. Por último, estaban los paquetes 8 y 9 que tenían contenidos para implementar en la aplicación web que se pueden considerar complementarios y que no tienen prioridad respecto al resto del trabajo. Por ello, los dejé en las últimas posiciones, se llevarían a cabo una vez terminados todos los paquetes anteriores.

Esta era una estimación inicial que podía estar sujeta a cambio a lo largo del proyecto. Era muy probable que mi estimación inicial fuera errónea ya que aún no estaba familiarizado con las tareas ni con el framework Backbone, pero era necesario realizar la estimación para cumplir con la metodología.

Tras un par de semanas trabajando con Backbone y comprendiéndolo mejor me di cuenta de lo grandes que eran las tareas asociadas a los paquetes, todavía se podían dividir en tareas aun mas pequeñas, pero anteriormente yo no era consciente de ello.

Con los nuevos conocimientos adquiridos me dispuse a realizar un segundo listado, mucho mas apropiado:

PAQUETE 1
Documentar puntos previos a la implementación
Practica con Backbone
Realizar listado de paquetes de trabajo

PAQUETE 2
Implantación de la estructura del prototipo Babelium
Creación de los modelos Backbone necesarios
Creación de las vistas Backbone necesarias
Aplicar templates del prototipo a las vistas de Backbone

PAQUETE 3
Investigación sobre el uso de listados en Backbone
Creación de una vista capaz de contener listas de vídeos
Creación de la vista para cada vídeo
Utilizar información de vídeos de Babelium para simular modelos

PAQUETE 4
Investigación sobre routers Backbone
Creación de routers para las vistas del paquete 3
Creación de routers para las distintas secciones de la aplicación
Limpieza de ficheros obsoletos pertenecientes al prototipo

PAQUETE 5
Revisión de ficheros CSS del prototipo para eliminar contenido innecesario
Uso de librerías jquery y jquery UI para preparar ventanas para el login y register
Simulación un login básico
Simulación un register básico
Ajuste del modelo usuario para simular los datos de Babelium

PAQUETE 6
Generar vista de la información del usuario
Generar vistas de las secciones About y Config
Creación de un sistema de búsqueda de vídeos según los tags
Investigación sobre paginación en Backbone

PAQUETE 7
Creación de la paginación para la vista del listado de vídeos
Aplicación de la paginación a la busqueda de videos
Revisión de ficheros CSS y limpieza
Importación de librerías online en vez de local

PAQUETE 8
Creación del formulario de upload de videos
Creación de la vista de ejercicios
Investigación sobre PHP y AJAX
Investigación sobre Api de Babelium

PAQUETE 9
Investigación sobre el uso de Digital Ocean
Creación de mi propio servidor Babelium.
Comprensión de los servicios implementados en el servidor Babelium

PAQUETE 10
Reemplazo de simulaciones por llamadas mediante AJAX
Creación de ficheros PHP para usar servicios de Babelium

PAQUETE 11
Importación de ficheros del reproductor de Babelium
Creación de la vista del listado de vídeos a evaluar con paginación
Creación de la vista de evaluación y formulario
Conseguir hacer funcionar el reproductor para practicar y evaluar ejercicios

PAQUETE 12
Uso del reproductor de Babelium para grabar
Uso de templates en ficheros externos

Ilustración 24: Distribución de tareas final

Esta lista ha ido evolucionando a la par que el proyecto, incluye todas las tareas a realizar.

En esta lista podemos ver que las distintas tareas son más específicas que las de la lista anterior, por lo que no puedo comparar la primera lista de tareas con la segunda, son totalmente distintas. Una persona con conocimientos de Backbone sería capaz de entender con facilidad todas y cada una de las tareas.

He ordenado los paquetes en el orden en el que los he escogido durante la planificación.

En principio cada paquete tiene una duración de una o dos semanas y todos han sido contrastados con mi tutor en nuestros controles semanales antes de llevarlos a la practica.

A continuación me dispongo a describir con más detalle el contenido de cada paquete, qué se tiene en cuenta en cada uno de ellos y la razón de las tareas:

- ***Paquete 1***

Antes de hacer nada, consideré necesario establecer las bases de la documentación a seguir durante el proyecto (puntos a comentar, planificación, metodología...).

Otra necesidad era empezar a utilizar el framework Backbone en mi proyecto. Hasta entonces solo había hecho uso de Backbone en ejemplos sencillos, por lo que esto representaba un desafío para el cuál debía prepararme practicando con ejemplos mas complejos. La mayoría de estos ejemplos los he obtenido a partir de los libros y webs de Backbone incluidos en la bibliografía.

Por último, debía realizar la división de las tareas en tareas menores e indivisibles que resultó en la lista de paquetes expuesta en este apartado.

- ***Paquete 2***

En este paquete di paso a la construcción de la estructura de ficheros característica de un proyecto que use Backbone. Esta estructura consiste en dividir los ficheros en subcarpetas dependiendo del papel que cumplan en la aplicación. Genero una carpeta que contenga todos los elementos de Backbone con 3 subcarpetas: vistas, modelos y router. Esta división hace permite distinguir fácilmente cada parte de la arquitectura modelo-vista-controlador.

Además de esto, generé los modelos y vistas de la aplicación que resultaban mas obvios para dejarlos ya hechos.

Lo más complejo de este paquete fue aplicar los templates del prototipo de Babelium a mi proyecto, ya que Backbone los trata de una forma distinta con otra notación.

- ***Paquete 3***

Este paquete surge de una reunión con mi tutor. Acordamos mutuamente que, lo primero que debía lograr con Backbone en mi aplicación era mostrar la lista de ejercicios disponibles, mostrando toda su información. Aún no disponíamos del servidor de Babelium para acceder la información de los ejercicios así que usé un fichero con la información de unos cuantos ejercicios para simular la base de datos.

- ***Paquete 4***

Una vez construida la lista de ejercicios, era necesario poder interaccionar con la aplicación web para acceder a las vistas de los ejercicios. Para ello, necesitaba investigar a cerca de los routers de Backbone.

Los routers son los que gestionan los eventos de la aplicación en función de la URL en la que se encuentre el usuario, por lo tanto, son los que nos permitirán desplazarnos por la aplicación accediendo a las distintas funcionalidades y vistas.

Una vez finalizado este paquete decidí hacer una limpieza, mi repositorio contenía demasiados ficheros obsoletos que utilizaba el prototipo inicial de la aplicación y que ya no hacían falta. Este proceso es lento y costoso pero necesario, además me permite ser capaz de identificar todos los ficheros de mi proyecto y sus contenidos mas fácilmente.

- ***Paquete 5***

Como continuación de la limpieza realizada en el paquete 4, me dispuse a revisar los ficheros CSS de la aplicación web. Este paso es necesario debido a los cambios realizados en la aplicación respecto al prototipo. Algunos elementos debían ser actualizados y otros directamente eliminados.

Después de terminar con la limpieza, decidí que era necesario implementar lo relacionado con la gestión de usuarios cuanto antes para empezar a aplicar sesiones en la aplicación.

Para llevar a cabo el registro de usuario y gestionar la sesiones debía incluir algún cuadro de dialogo para que el usuario introduzca sus datos. La herramienta perfecta para esto es jQuery UI, teniendo en cuenta que las librerías de jQuery se encuentran presentes en el proyecto.

Por último, debía probar las nuevas funcionalidades mediante simulaciones sin conectar con la base de datos de Babelium, una vez más, haciendo uso de modelos creados específicamente para este fin.

- ***Paquete 6***

Para terminar con lo relacionado con los usuarios, este paquete cuenta con la tarea de crear la vista de la página en la que se muestran los datos del usuario actual. Entre estos datos se muestran los idiomas del usuario, tanto los conocidos como los que se quieren aprender, con la respectiva bandera del país de origen de cada idioma y el dominio del usuario sobre el mismo.

A parte de esto, para ir quitando cosas del camino, decidí también preparar las vistas y routers de las secciones de *"About"* y *"Config"*, que son las secciones más ligeras de la aplicación web.

Como estas ultimas tareas son relativamente sencillas, hacía falta incluir en este paquete una tarea de peso, así pues, tras discutirlo con mi tutor, me dispuse a crear un sistema de filtrado que permitiera buscar vídeos mediante sus tags identificadores.

Hasta ahora, la lista de ejercicios que mostraba en la sección de *"Practice"* se encontraba totalmente descontrolada. Es decir, en la misma página había mas de 100 ejercicios. Para arreglar este problema me pareció oportuno investigar cualquier tipo de herramienta que Backbone me pudiera proporcionar para crear una paginación de los ejercicios de forma sencilla.

- ***Paquete 7***

A partir de los investigado en el paquete anterior, empecé aplicando un sistema de paginación a la sección de *"Practice"*, permitiéndome así disponer de 10 ejercicios por página, y una enumeración de paginas por las que se puede ir navegando.

La paginación no solo se aplica a la lista de ejercicios, sino que también es necesario aplicar esta paginación a la lista de ejercicios que se genera al buscar vídeos por sus tags.

Una vez más, consideré necesaria una limpieza de los ficheros CSS, ya que la mayoría de las vistas realizadas en los últimos paquetes han sido creadas y diseñadas por mi. Es necesario incluir todos los nuevos elementos para que la aplicación tenga el aspecto deseado.

Tras la última reunión con mi tutor, me hizo ver que es conveniente importar las librerías desde la nube, para así disminuir el peso de la aplicación. Siguiendo sus consejos, me dispuse a eliminar las librerías

de mi repositorio y a realizar las importaciones necesarias en el fichero html.

- ***Paquete 8***

A estas alturas del proyecto, ya me resultaba imposible continuar sin crear el servidor Babelium para empezar ha realizar pruebas reales y no simulaciones. En caso de que haya problemas, cuanto antes se descubran, mejor.

Hablando con mi tutor respecto a esto último, estábamos de acuerdo. Por eso, en este paquete decidí terminar con una parte más de la aplicación mientras investigaba las herramientas ajax y php.

Estas últimas tareas consisten en la creación del formulario de subida de ejercicios por parte de los usuarios, así como la creación de la vista de ejercicios. Ahora que por fin iba a disponer del servidor, era necesario contar con la vista de los ejercicios para poder realizar todas las pruebas necesarias.

- ***Paquete 9***

Con la ayuda de mi tutor, debía crear un servidor Babelium para mi uso privado durante la implementación del proyecto. Además, una vez montado todo, es necesario un pequeño periodo de adaptación y estudio de la nueva herramienta.

No tenía experiencia previa con la herramienta *Digital Ocean* pero gracias a mi tutor, en seguida me adapté a la novedad y pude estudiar los distintos servicios que me aporta el servidor de Babelium.

- ***Paquete 10***

Este paquete contiene una carga de trabajo particularmente pesada. Esto se debe a la necesidad de volver a revisar todo lo hecho hasta el momento en busca de los puntos en los que se dependa del servidor para alguna funcionalidad, ya sea acceder con un usuario, registrarse, ver un ejercicio, etc.

Este paquete lo seleccioné teniendo en mente la alta probabilidad de que su duración fuera mayor que el máximo que fije en principio (2 semanas por paquete como máximo), pero dividir esta tarea carece de sentido, ya que todos los cambios que hay que realizar consisten en utilizar Ajax y PHP de la misma manera en sustitución de mis simulaciones.

- ***Paquete 11***

Una vez terminado el cambio de simulación a pruebas reales, el siguiente paso a tomar consistía en empezar a utilizar el reproductor de vídeos de Babelium. Ahora que ya recibo la información correcta y real de los ejercicios desde el servidor de pruebas, es necesario tratar la información de los ejercicios para poder utilizarla en esta aplicación web.

Una vez preparada esta última tarea, genero las vistas tanto de la lista de ejercicios a evaluar como de cada uno de los ejercicios para poder evaluarlos. Debido al gran parecido de las secciones *"Practice" y "Evaluate"*, esta tarea no resulta excesivamente complicada.

- ***Paquete 12***

En este último paquete, rematé la aplicación web permitiendo que el reproductor de Babelium permita grabar la realización de un ejercicio y subirlo al servidor de pruebas.

Por último, por recomendación de mi tutor, decidí separar las vistas de la aplicación de sus respectivos templates. De este modo, los templates son más simples de leer y modificar.

5. Riesgos y factibilidad

Todos los proyectos tienen una serie de riesgos durante su elaboración que pueden suponer un cambio importante, un replanteamiento, la pausa o incluso el fin del proyecto. En este proyecto existen una serie de riesgos que voy a enumerar y analizar, estos riesgos están divididos en dos categorías, la primera son los riesgos comunes, que pueden surgir en cualquier proyecto, y en segundo lugar los riesgos específicos de este proyecto.

Las tablas de los riesgos tendrán colores identificadores, que permitirán fácilmente localizar los riesgos de mayor impacto sobre el proyecto. De menor gravedad a mayor: Verde → Amarillo → Naranja → Rojo

- **Riesgos comunes**

- Incapacitación del encargado del proyecto.

Descripción	Por cualquier razón el encargado de realizar el proyecto sufre alguna lesión que no le permite continuar con el proyecto.
Prevención	Ninguna salvo tener cuidado.
Plan de contingencia	Dejar el proyecto en pausa o cerrarlo en función de la gravedad de la lesión.
Probabilidad	Muy improbable.
Impacto	Muy crítico.

Tabla 3: Riesgo incapacitación

- Inexperiencia con las tecnologías a utilizar.

Descripción	El desarrollador del proyecto no dispone de los conocimientos necesarios para llevar a cabo el proyecto.
Prevención	Mantenerse al día sobre las tecnologías a utilizar.
Plan de contingencia	Pausar el proyecto y dedicar no mas de una semana o dos al estudio de dichas tecnologías de forma que se pueda continuar con él.
Probabilidad	Bastante probable
Impacto	Moderado.

Tabla 4: Riesgo inexperiencia

- Mal funcionamiento del hardware.

Descripción	El equipo utilizado para llevar a cabo el proyecto o alguno de sus componentes vitales falla produciendo perdida de datos o ralentización en el trabajo.
Prevención	Tratar el hardware adecuadamente y cuidarlo todo lo posible, además, hacer copias de seguridad y mantener el repositorio actualizado siempre.
Plan de contingencia	Utilizar los datos almacenados en el repositorio en un equipo en condiciones que permitan trabajar.
Probabilidad	Improbable.
Impacto	Crítico.

Tabla 5: Riesgo hardware defectuoso

- Cambio de prioridades o requisitos.

Descripción	El cliente del proyecto introduce algún cambio en el proyecto de forma imprevista que hace cambiar las prioridades o requisitos a la hora de realizarlo.
Prevención	Mantenerse en contacto con el cliente cada poco tiempo para asegurarse de que no se produzcan cambios bruscos y repentinos.
Plan de contingencia	Reunirse con el cliente lo antes posible y replantearse la planificación del proyecto para ajustarse a los nuevos requisitos y prioridades.
Probabilidad	Muy improbable.
Impacto	Crítico.

Tabla 6: Riesgo cambio de prioridades o requisitos

- Perdida total o parcial del código implementado.

Descripción	Por cualquier razón, se pierde el código implementado durante la ejecución del proyecto o la documentación del mismo.
Prevención	Realizar copias de seguridad, no tener todo el contenido del proyecto en un solo sitio y utilizar herramientas de almacenamiento en la nube.
Plan de contingencia	Obtener la copia de seguridad o la versión que almacenamos en la nube y restablecer el proyecto con perdidas mínimas o nulas.
Probabilidad	Improbable.
Impacto	Crítico.

Tabla 7: Riesgo perdida de datos

- La estimación de tiempo de las tareas es errónea.

Descripción	Como en todos los proyectos, es difícil estimar las horas que harán falta para realizar cada tarea, esto puede llevar a que no se llegue a tiempo a alguno de los puntos de control del progreso del proyecto establecidos durante la planificación.
Prevención	Durante la planificación, al estimar la duración de cada tarea, dejar algo más de tiempo por si acaso. Además, cuanto mas se dividan las tareas durante la fase de planificación, mas fácil sera estimar los tiempos de duración de cada una de ellas.
Plan de contingencia	Dedicarle un tiempo a replantear la duración de las siguientes tareas ya que la actual a alterado la duración de las siguientes.
Probabilidad	Bastante probable.
Impacto	Bajo.

Tabla 8: Riesgos planificación errónea

- **Riesgos específicos**

- Framework insuficiente

Descripción	El framework elegido para realizar el proyecto no cumple alguna de las necesidades del mismo.
Prevención	Analizar los frameworks existentes y elegir el más adecuado para el proyecto.
Plan de contingencia	Analizar la situación y decidir si se busca una forma alternativa para cumplir con esas necesidades o se utiliza otro framework o se prescinde de dicha necesidad para no entorpecer el progreso del proyecto.
Probabilidad	Muy improbable.
Impacto	Muy crítico.

Tabla 9: Riesgo framework insuficiente

- Problemas con la migración de la funcionalidad de la aplicación web.

Descripción	Alguna funcionalidad de la aplicación web en Flex no se puede realizar con HTML5.
Prevención	Mantenerse informado sobre las capacidades de HTML5.
Plan de contingencia	Buscar una forma alternativa de llevar a cabo la funcionalidad ya que HTML5 dispone de muchas formas de hacer las cosas, solo hay que estar informado.
Probabilidad	Muy improbable.
Impacto	Moderado.

Tabla 10: Riesgo problemas con migración

6. Resumen: herramientas utilizadas en el control del proyecto

Todo proyecto necesita que su evolución pase por un control continuo. Este control sirve principalmente para comprobar el progreso del proyecto y para asegurarse de que no se pierden de vista los objetivos.

En mi caso, me ha bastado con una sola herramienta, GitHub. Esta herramienta es un repositorio online que permite almacenar en la nube tanto los ficheros de la última versión como los de todas las versiones de la aplicación que hayan sido registradas. En caso de que se necesite algún fichero de alguna versión anterior a la actual, se puede retroceder de versión para acceder a él.

No he necesitado mas herramientas debido a que GitHub, a parte de almacenar los ficheros, incluye distintas funcionalidades de gran utilidad en el desarrollo de un proyecto:

- Generar calendarios de tareas pendientes.
- Gestionar tareas, asignarles tags, miembros del grupo a cargo de ellas, un chat, etc.
- Crear milestones, que son fechas limites para las tareas.
- Incluye una wiki en la que se puede ir tomando nota del proyecto para generar la documentación.

Mi experiencia personal con este repositorio es totalmente positiva, he tenido alguna experiencia previa con él pero nunca con un proyecto de esta complejidad.

He hecho uso de algunas de las funcionalidades mencionadas pero no todas, por ejemplo, a mediados del proyecto mi tutor me dio a conocer la posibilidad de generar listados de tareas para organizarme el trabajo, funcionalidad que yo hasta entonces desconocía. Esta funcionalidad me resulto muy interesante y empecé a usarla junto a los milestones para marcarme el ritmo.

Tanto mis progresos en el proyecto como las tareas pendientes podían ser consultadas por mi tutor en mi repositorio en todo momento, lo cual nos ha aportado grandes facilidades a la hora de mostrar mi trabajo y los prototipos semanales.

La wiki no la utilice porque me resultaba más cómodo utilizar un editor de texto normal y corriente, además, así no dependo de tener acceso a Internet para poder progresar.

7. Especificaciones de Babelium Project

Babelium Project cuenta con una serie de funcionalidades que deben ser reflejadas en la nueva aplicación web con HTML5. La forma mas sencilla y rápida de ver dichas funcionalidades es meterse en la aplicación web Babelium y observar lo que ofrece, y en caso de duda consultar el código de la aplicación.

Durante mi investigación sobre Babelium he anotado las siguientes funcionalidades:

- Sistema para registrarse en la web.
- Sistema de conexión y desconexión de sesiones de usuario.
- Acceso a enlaces externos (blog, por ejemplo).
- Enlaces dentro de la misma aplicación web (tutoriales, distintas pestañas, página de datos de usuario,etc).
- Sistema de subida de vídeos a la aplicación.
- Buscador de vídeos.
- Mostrar ejercicios disponibles.
- Mostrar evaluaciones disponibles
- Configuración tanto del micrófono como de la cámara web.
- Cambiar el idioma de la aplicación:
  - Alemán.
  - Español.
  - Francés.
  - Inglés.
  - Euskera.
- Sistema de créditos:
  - Para poder practicar necesitas créditos.
  - Subtitulando vídeos para que sirvan de ejercicios para otros usuarios produce créditos.
- Todas las posibles opciones dentro de un vídeo en función de lo que se esté haciendo:
  - Haciendo un ejercicio:

        - Elegir rol.
        - Empezar a grabar.
        - Elegir grabar voz o voz y vídeo a la vez.
        - Votar vídeo.
        - Denunciar vídeo.
        - Ver más información.
    - Valorando un ejercicio:
        - Votar distintos valores.
        - Incluir comentario de texto o vídeo.
        - Enviar evaluación.
        - Resetear evaluación.
    - Subtitulando un ejercicio:
        - Generar o borrar subtítulos.
        - Asignar duración y rol de cada subtítulo.
        - Publicar el vídeo.

Algunas de las funcionalidades no serán incluidas en este proyecto, bien porque la carga de trabajo general del proyecto es muy elevada o bien por petición de mi tutor:

- La aplicación está unicamente disponible en ingles.
- Los créditos se tendrán en cuenta en los datos del usuario pero no se implementa el sistema de créditos
- El subtitulado de vídeos queda descartado de la aplicación debido a la ineficiencia que supone para la misma, es mejor que los usuarios subtitulen los vídeos por su cuenta y los suban ya subtitulados.

Framework Backbone

Ilustración 25: Backbone

Backbone es un framework desarrollado por Jeremy Ashkenas cuyo principal objetivo es el diseño de aplicaciones web de una sola página basadas en el paradigma de diseño Modelo-Vista-Controlador.

Como ya he explicado anteriormente, he elegido este framework por su sencillez y transparencia. Backbone se puede aprender fácilmente debido a su gran parecido al uso de javascript normal, esto a su vez

permite dotar de una gran transparencia a la aplicación en comparación con otros frameworks.

Los frameworks como Ember o Angular, al incorporar tantas facilidades, acortamientos en el código y funcionalidades de gestión de la aplicación, puede suponer un problema en cuanto a transparencia.

Gracias a la transparencia de Backbone he sido capaz de definir la aplicación y sus funcionalidades de principio a fin sin ningún proceso que desconozca. Esto supone una ventaja en cuanto a facilidad a la hora de manipular el código en caso de necesitar realizar una modificación, pero una desventaja directa de esto es la cantidad de código.

Imagen comparación de código entre ember backbone y angular.

Backbone se basa en dividir el código de la aplicación en 3 partes, al ser un framework MVC, la división es lógica, pero hay un elemento más, las colecciones. A continuación incluyo una explicación de mi entendimiento de los 4 elementos clave del framework Backbone, en la que resumo todo lo necesario para comprender el framework a nivel básico por lo menos:

1. Modelos

Los modelos contienen la información de los objetos utilizados en la aplicación, en el caso de Babelium, se consideran modelos cada usuario, cada ejercicio y cada respuesta a un ejercicio.

El modelo es el encargado de guardar los datos como pueden ser los idiomas que domina el usuario, la duración de un ejercicio, el ID del usuario que ha realizado un ejercicio, etc.

2. Vistas

Son las encargadas de mostrar la información de la aplicación por pantalla a los usuarios. Esta información es una combinación de los distintos templates que conforman la apariencia de la aplicación y los modelos asociados a las vistas.

Las vistas pueden tener un template asociado, que es un trozo de código en HTML5 que configura la apariencia de la vista.

Además de asociarles un template, también se les puede asociar un modelo o una colección de modelos. Si se asocia un modelo o una colección a una vista, su información se encontrará disponible para su uso en el template de la vista en todo momento. Esto resulta útil, por ejemplo, en la pagina que muestra los datos de un usuario, si a la vista que la gestiona le asocio el modelo del usuario actual, podré acceder a la información del usuario fácilmente para poder incluirla en el template de la vista.

Una de las funciones principales de las vistas es el control de eventos, se pueden definir unos eventos a los que asociar una acción.

Por ejemplo, en una vista genero 2 botones y en el código de la vista incluyo un evento de clic sobre el botón numero dos al que asocio una acción que crea un dialogo de texto saludando. Si hago clic en el botón uno, no pasará nada, si hago clic en el botón número dos, aparecerá el dialogo.

Se pueden llegar a crear eventos y acciones de mayor complejidad que esta pero durante el proyecto no ha surgido la necesidad de utilizar eventos complejos.

3. ***Colecciones***

Las colecciones son listas de modelos. Estos modelos pueden ser volcados sobre la colección, bien manualmente generando cada modelo a añadir, o bien mediante un fichero php, el cual Backbone se encarga de utilizar para obtener los modelos a añadir.

Para poner un ejemplo de mi caso, dispongo de la colección VideoList, esta colección, en su código, tiene una variable que apunta a un fichero php. En el momento en el que genero una instancia de esa colección, el fichero php apuntado se ejecuta y realiza una consulta en la base de datos de Babelium para obtener la lista de ejercicios disponibles. La respuesta de este fichero php se almacena en la colección en forma de modelos, obteniendo así la información directamente de la base de datos, sin necesidad de transformarla.

Para ayudar a comprender mejor el uso combinado de vistas y colecciones voy a describir un ejemplo sobre el uso de una colección asociada a una vista.

En el caso de esta aplicación, esta situación surgió por la necesidad de mostrar dentro de una vista múltiples vistas, me refiero a la lista de ejercicios disponibles. En este caso hace falta crear múltiples vistas dentro la vista de la sección *"Practice"*. Para conseguir esto, asocio la colección de ejercicios a la vista general, y dentro de ella, iterando la colección, genero una vista nueva por cada modelo extraído de la colección.

4. ***Routers***

Aquí llegamos al elemento más interesante, los routers. Me parecen los más interesantes porque desde aquí es desde donde se controla todo el funcionamiento de la aplicación prácticamente. Un router es la C (controlador) del MVC.

Los routers suelen gestionarse todos desde un solo fichero, para mantener el proyecto limpio. En el caso de esta aplicación ha sido así.

Su función en la aplicación consiste en dirigir la aplicación, es decir, en función de la URL en la que el usuario esté navegando, en pantalla se mostrará una vista u otra. Además, también son los encargados de hacer que el navegador anote en el historial las distintas URLs de la aplicación en las que se ha navegado, especialmente útil para habilitar el uso del botón de "Retroceder" en la aplicación web.

Los routers se dividen en 4 partes fundamentales:

- Definición de rutas y función de cada una.

```
// Defino las rutas y las funciones de cada ruta
routes:
        {
            "Home": "goHome",
            "Practice/page_:p": "goPractice",
            "Practice/exercise/:name": "goPrExercise",
            "Evaluate/page_:p": "goEvaluate",
            "Evaluate/response/:name" : "goEvExercise",
            "Subtitle": "goSubtitle",
            "Config": "goConfig",
            "About": "goAbout",
            "User_info": "goUserInfo",
            "Help/page_:p": "goHelp",
            "Upload": "goUpload",
            "Search/:terms/page_:p": "goSearch"
        },
```

Ilustración 26: Direcciones del router Backbone

Podemos ver que primero definimos cada ruta, a veces haciendo uso de variables para representar el número de una página por ejemplo, y después se indica que acción se va a realizar.

Si en la URL aparece http://.../Practice/page_5, se ejecutará la función goPractice y se le pasa como parámetro la variable p, que este caso es 5.

- Implementación de las distintas funciones.

En esta parte simplemente se implementan las funciones declaradas en la parte anterior.

- Generación de la instancia del router.

```
// Instancio el router que he configurado
var myTodoRouter = new TodoRouter();
```

Ilustración 27: Instancia de router

Con esto, se crea la instancia de router que lo pone en funcionamiento, sin este paso, el router nunca haría nada, ya que no existiría.

- Permitir al historial tomar nota de los routers visitados.

Esto es opcional, ya que puede que a veces no interese que quede todo registrado.

```
// Permito que el historial tome nota de los routers visitados
Backbone.history.start();
```

Ilustración 28: Inicia el historial del router

IMPLEMENTACIÓN

Una vez hemos comprendido como funciona Backbone, en esta sección se va a explicar como se ha desarrollado la aplicación y algunos de los múltiples problemas que han surgido en el camino.

1. Toma de contacto

Habiendo investigado ya tanto las herramientas a utilizar y teniendo los conocimientos suficientes de Backbone como para empezar a implementar la aplicación, lo primero fue analizar el prototipo de Babelium en HTML5 existente (Mintzabel) para ver qué se podía aprovechar y qué no.

El prototipo no dispone de demasiada funcionalidad y además utiliza librerías anticuadas e incluso elementos relacionados con Flex. Ésto llevó a la conclusión de que lo único que iba a resultar útil era el diseño de la aplicación, es decir, el fichero html y los múltiples ficheros CSS3.

Tras eliminar las librerías obsoletas, limpiar código inútil y mostrar al tutor el resultado del filtrado del prototipo, se comenzó a desarrollar la aplicación.

Al principio, para ir cogiendo soltura con Backbone, se decidió hacer una simulación simple del login. El objetivo era ser capaz de manipular la interfaz de usuario para mostrar sus datos desde un fichero JSON externo. Para ello se genera un modelo de usuario y se emplean las vistas para que se muestre la información. Además, también se reciclaron los templates del prototipo para su uso en las vistas Backbone.

Tras esta pequeña práctica, Juanan sugirió que el siguiente paso fuese ser capaz de mostrar una lista de videos con su información en el apartado de práctica de la aplicación. Las URL de los videos y sus datos se encontraban en un fichero JSON externo.

2. Listado de vídeos

En este paso debía cargar la lista de vídeos del JSON en una colección Backbone y mediante vistas mostrarlos como yo quisiera.

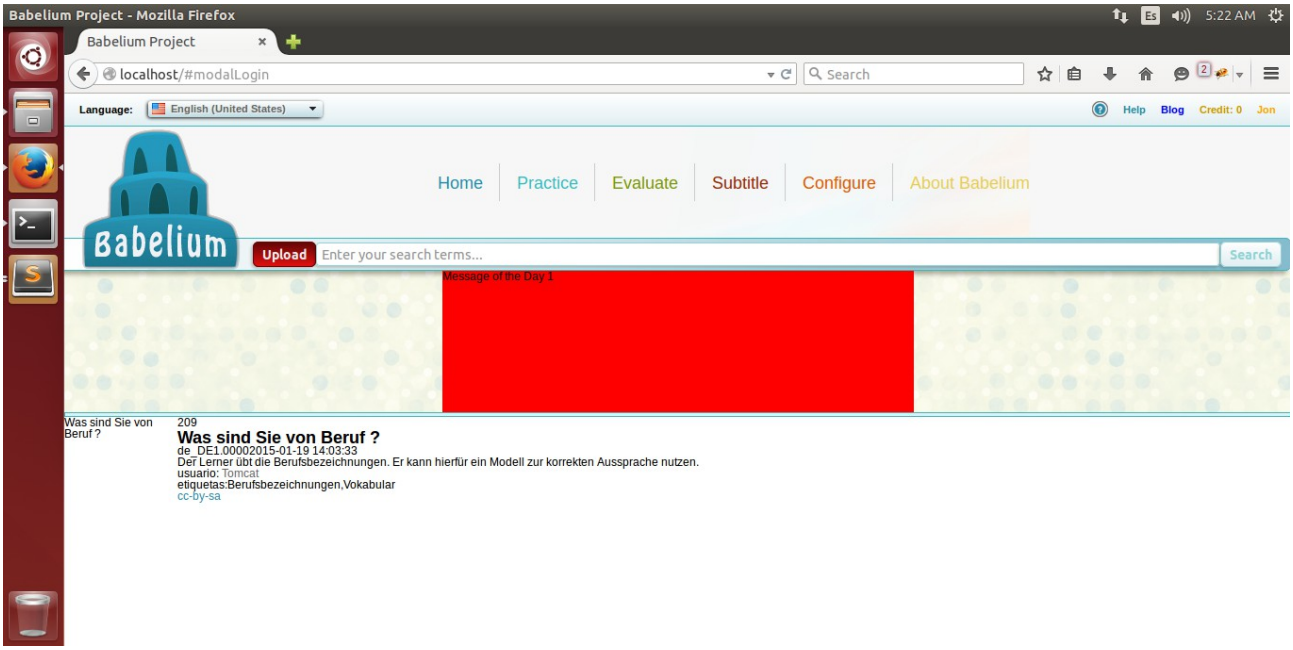

Ilustración 29: Vista básica de listado

Esta imagen muestra el resultado al terminar con este paso, aun no mostraba la lista de videos pero lo único que faltaba era recorrer la colección en la que guardaba todos los videos y generar una vista por cada uno. De esta forma se mostraban todos los videos en una columna.

Ser capaz de realizar este paso me permitió comprender mejor las colecciones y los templates.

Una vez el tutor dio su visto bueno al resultado, se decidió empezar a utilizar routers. Teniendo operativos un login, la visión de datos del usuario y la carga de la lista de videos era necesario poder acceder a cada funcionalidad en un apartado distinto.

3. Routers

Gracias al libro *"Developing Backbone.js Applications"* fue fácil crear los distintos routers para establecer la estructura de la funcionalidad de la aplicación. Ahora cada funcionalidad tenía su sitio.

Dada la sencillez de este paso, se dedicó cierto tiempo a la limpieza de algunos ficheros provenientes del prototipo, principalmente CSS3. Aún había cosas obsoletas.

El siguiente paso decidido en la reunión con el tutor fue dejar presentable la lista de videos, separarla en 2 columnas y establecer un sistema de paginación.

A pesar de que se decidió ésto, se dejó de lado por una semana porque aún se estaba limpiando los CSS3 y no se quería dejar el proceso a medias. De esta forma se formó un nuevo paquete de tareas para realizar en acompañamiento de la limpieza.

4. Limpieza y jQuery

Ante la necesidad de incluir jQuery en la aplicación, se tomó la decisión de crear un paquete de trabajo que consistiera en aplicar jQuery al proyecto y añadir cuadros de dialogo a la funcionalidad de login mediante jQuery UI. Aprovechando esto, también se prepara lo necesario para simular el registro de un usuario en la aplicación.

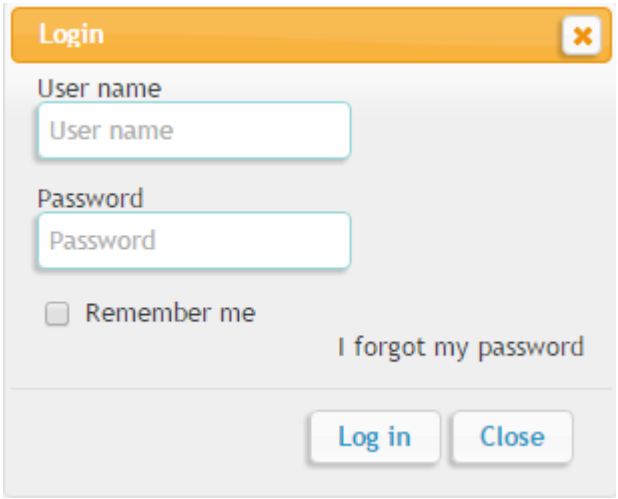

Ilustración 31: Diálogo login

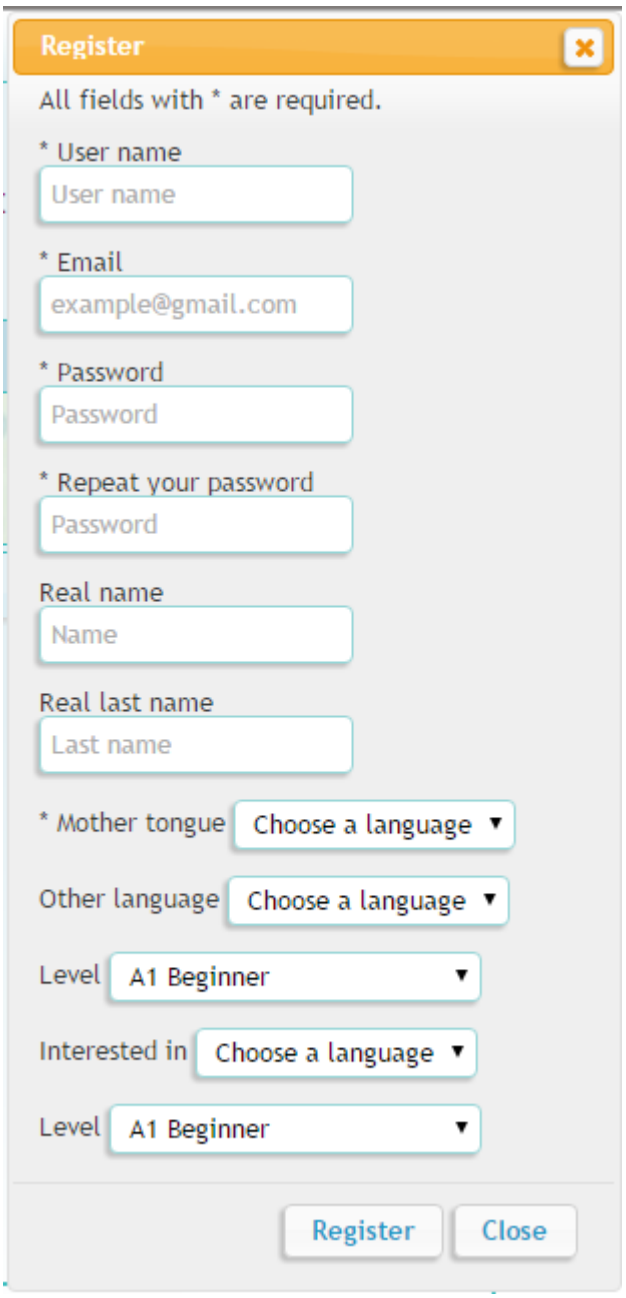

Ilustración 30: Diálogo register

Los resultados de este paso fueron muy positivos para la aplicación, ya que la dotaron de un mejor aspecto y ya no era necesario realizar el login desde la consola del navegador.

Con esta parte hecha, se procedió a retomar el paso acordado previamente, la paginación de videos.

5. Paginación, filtrado y vistas

Tras cierta investigación, se encuentra un plugin de Backbone para paginación, Backbone Paginator.

Prefería utilizar un plugin de Backbone antes que crear yo mismo el sistema de paginación para evitar conflictos e incompatibilidades en el futuro, pero surgieron algunos problemas problemas:

- Habría que tener otro objeto distinto más en el proyecto, PaginableCollection.
- Era necesario deshacer lo hecho en el paquete de trabajo en el que se creó la lista de videos y se mostraba en pantalla y volver a plantear desde 0.
- A pesar de que se intentó utilizar, no llegó a funcionar en ningún momento.
- La cantidad de tiempo que llevaría volver a hacer todo para un novato en Backbone sería mayor que si se creara un sistema de paginación propio.

Por todas estas razones, se decidió seguir avanzando y crear un sistema de paginación propio. Si se hubiera considerado la paginación antes de establecer el listado de videos hubiera merecido la pena, pero teniéndolo hecho ya, es retroceder.

Antes de hacerlo se consideró necesario esperar a la reunión con el tutor para plantear el cambio. Para no ir a la reunión con las manos vacías, se tomaron algunas tareas de la lista de tareas para dejarlas terminadas:

- Vista de la pestaña About

Una vista simple cuyo template solo contiene texto y una referencia a un email.

- Vista de la pestaña Configuration

Esta vista contiene 2 pestañas creadas mediante jQuery UI en las que se accede a las opciones de configuración de micrófono y cámara. En cada pestaña se incluye un objeto flash que es el encargado de la configuración. Actualmente estos objetos no hacen nada porque la

aplicación no usa flash pero qué se va hacer con ellos no se encuentra dentro del alcance de este proyecto.

- Vista de la información del usuarios

Antes se podía ver por pantalla la información puesta sin ningún formato u organización, ahora se deja presentable.

- Sistema de filtrado de vídeos según tags

Se incluye una barra de búsqueda para la pestaña practice con la que se puede localizar vídeos escribiendo alguno de sus tags.

Para lograr esto, en el mismo router, antes de cargar las vistas de la página de vídeos, se hace un filtrado con el que se separan los vídeos que coincidan con el tag y solo se muestran esos.

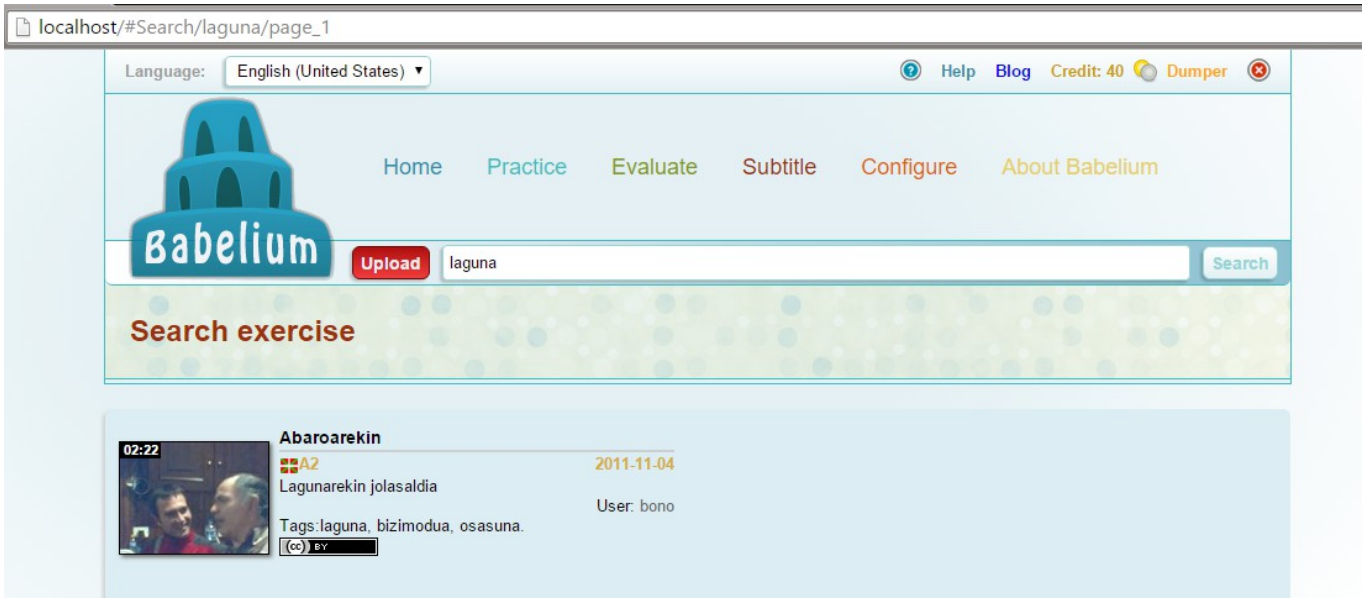

Ilustración 32: Búsqueda por tags

En esta imagen se observa el resultado de la búsqueda de laguna, en este caso solo hay un vídeo que coincida. Como se puede comprobar, la palabra laguna aparece reflejada en la URL de la aplicación.

A parte de eso, también se puede comprobar la nueva apariencia de los vídeos en la lista. Este cambio se debe al reajuste de los nombres de los elementos CSS3, la limpieza había provocado un problema con algunos elementos. Tras recurrir a versiones anteriores como referencia mediante la herramienta GitHub se solucionó el problema.

En la reunión con el tutor se muestran los nuevos resultados y se trata el asunto de la paginación. Juanan aceptó la solución propuesta y se fijo que lo siguiente era hacer funcionar la paginación. Además Juanan recomendó que las importaciones de las librerías fueran online y no desde un fichero local, ya que así se reduce el peso de la

6. Paginación

El resultado de la paginación fue bastante satisfactorio. Consta de 7 enlaces al pie de la lista de videos, 5 de estos enlaces son números de página (2 anteriores, la actual y 2 posteriores) y a cada lado hay una flecha que permite saltar de 5 en 5 en cada dirección.

Ilustración 33: Paginación

Se puede comprobar como la paginación esta presente en la aplicación y en la URL, ademas de la anotación del número de videos existentes. Se muestran 10 vídeos por página.

Esta funcionalidad también hace falta para la funcionalidad de búsqueda de vídeos, por lo que se utiliza también ahí y se comprueba que funciona sin necesidad de cambiar nada.

Para ayudar a explicar como se ha logrado ésto, se muestra el siguiente esquema:

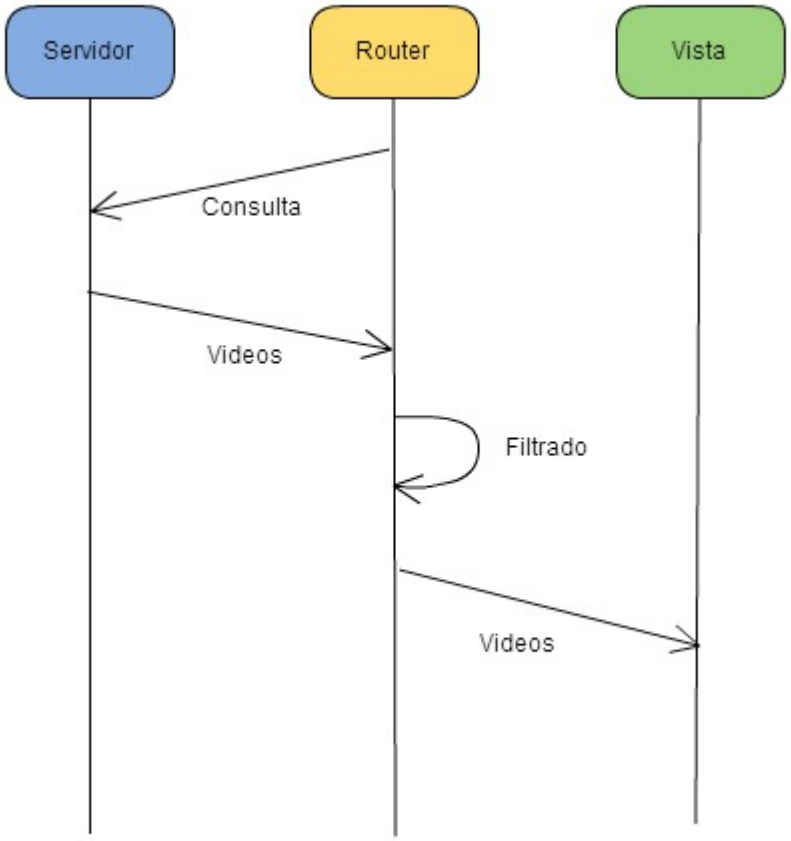

Ilustración 34: Esquema de paginación

1. Primero accedemos a la sección de practice, se cambia la URL y tenemos, en este caso, Localhost/#Practice/page_1.

2. El router de Backbone solicita la lista de ejercicios y guarda el resultado en una colección de videos.

3. El router recorre la colección en busca de los videos que se encuentren entre las posiciones $(página \times 10) - 10$ y $página \times 10$. Siendo la página por defecto la número 1, sería desde el vídeo 0 al 9 incluidos.

4. El router genera las vistas de los vídeos y en la vista principal de la sección practice añade enlaces para navegar por las vecinas.

Como curiosidad sobre este apartado cabe mencionar la cantidad de lineas de código empleadas para poner en funcionamiento la paginación. Para ello, se muestran las siguientes imagenes:

Ilustración 35: Código paginación nueva

Ilustración 36: Código paginación vieja

A la derecha se muestra la miniatura del código inicial empleado para la paginación. A la izquierda se muestra una segunda versión que se creó más adelante. Ambos códigos hacen lo mismo pero el de la izquierda es mucho mas eficiente, con menos iteraciones y más corto.

Este tipo de cosas hacen reflexionar y demuestran que siempre hay espacio para mejoras.

En la reunión con el tutor, tras comprobar que la paginación implementada funciona correctamente y es sencilla, se fijan los nuevos objetivos. La conversación giró en torno a la creciente necesidad de montar el servidor con la API de Babelium para poder sustituir los ficheros JSON por llamadas al servidor mediante PHP.

7. Investigación PHP y AJAX

Debido a la poca experiencia con PHP y AJAX, fue necesario dedicar un tiempo a la formación. Es mejor documentarse y hacerlo bien que hacerlo medio bien y luego tener que corregir los errores.

Se acordó con el tutor que se volvería a hablar del tema una vez se haya investigado lo suficiente.

La investigación se centró en el uso de ficheros PHP a través de AJAX y lo básico de PHP para ser capaz de implementar funciones relativamente sencillas.

A parte de ésto, también se dedicó algo de tiempo a otras cosas:

- Montar un servidor con el API de Babelium

Para que el proyecto pudiera seguir adelante, era necesario empezar a utilizar el API de Babelium y dejar de depender de simulaciones para probar las funcionalidades.

El primer paso, fue generar una máquina virtual e instalar el software necesario para que el API funcionase:

- Apache Web server 2.0+
- MySQL 5.2+
- PHP 5.2+
- ant
- file
- Sox 14.3.2+
- Adobe Flash Player 11.1+
- Zend Framework 1.12
- Adobe Flex SDK 4.6+
- ffmpeg 0.8+
- Red5 1.0+

La guía de instalación se encuentra en el repositorio de Babelium Flex standalone site[1].

Se acordó con el tutor que después de instalar el software listado se esperaría a la siguiente reunión para proceder a montar el servidor juntos.

Unos días antes de la reunión, uno de los riesgos de la lista de riesgos tratada previamente se volvió realidad, el disco duro del equipo en el que se estaba desarrollando el proyecto dejó de funcionar. Esto supuso un gran contratiempo, ya que no disponía de una copia de seguridad de la máquina virtual.

Afortunadamente, el repositorio en el que se encuentra alojado el proyecto tenía la última versión guardada así que las perdidas, en cuanto al código y documentación de la aplicación, fueron nulas.

El disco duro estropeado fue reemplazado 3 días después, pero, para no perder el tiempo, se utilizó un equipo secundario.

- Crear la vista de la sección upload

Esta sección contiene el formulario necesario para subir vídeos a la aplicación web.

En principio se iba a diseñar con la posibilidad de subir un archivo de vídeo desde el equipo del usuario o hacer una grabación usando el reproductor de Babelium para subirlo directamente, pero tras discutirlo con el tutor se llegó a la conclusión de que no puede depender tanto de los elementos Flash de la aplicación vieja porque

[1] https://github.com/babeliumproject/flex-standalone-site

más adelante darán problemas. Por consiguiente, solo se pueden subir vídeos mediante archivos desde el equipo.

- Crear la vista que de un ejercicio

Es la vista que se muestra cuando un usuario selecciona un ejercicio de la lista de videos de practice.

Esta vista utiliza el reproductor de Babelium, lo cual supone problemas. Dado que el reproductor de Babelium es proporcionado por Juanan, las instrucciones para su uso también, pero en caso de fallos, corregirlos puede ser prácticamente imposible sin la ayuda del tutor. Tras aplicar el template adecuado con algunos ajustes a esta vista se obtuvo lo siguiente:

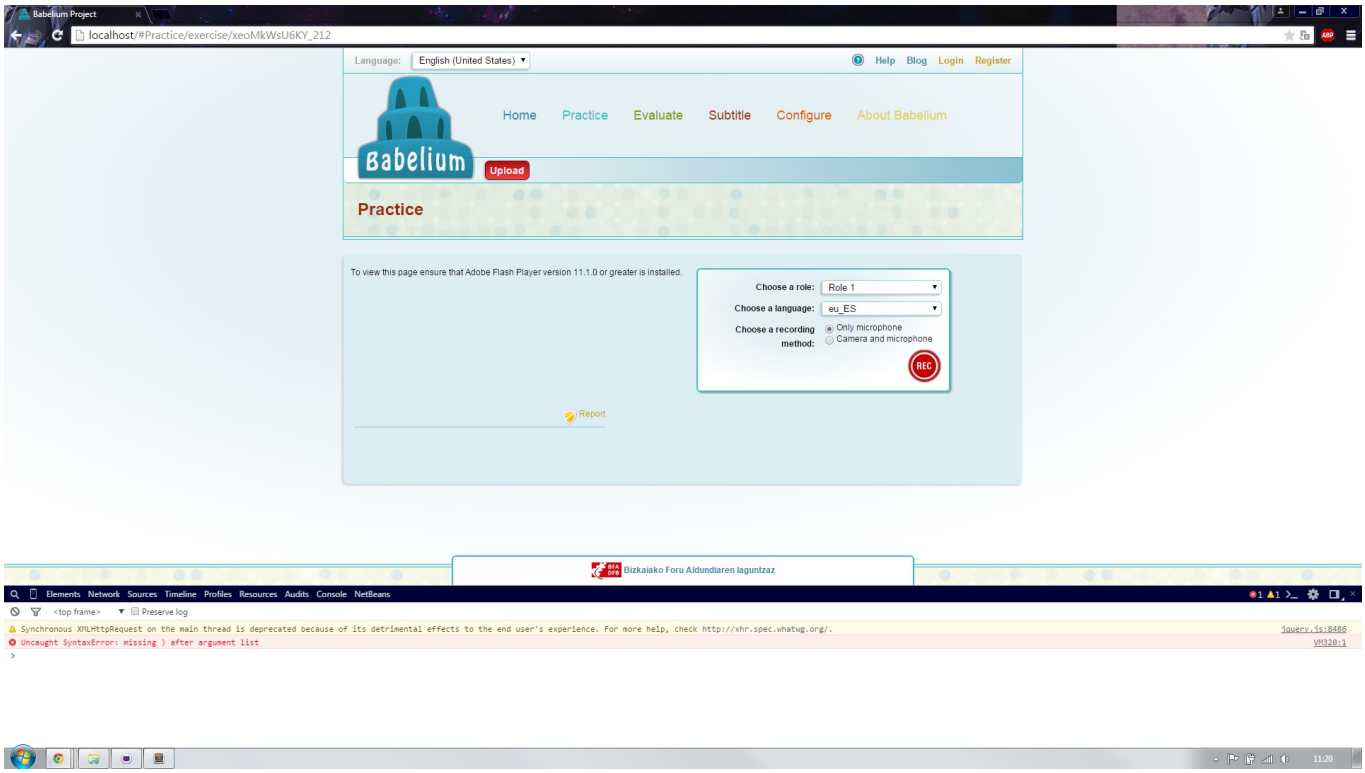

Ilustración 37: Error con Babelium player

El resultado parece correcto, pero el reproductor no se encuentra en ninguna parte. En su lugar aparece una linea de texto que dice:

"To view this page ensure that Adobe Flash Player version 11.1.0 or greater is installed."

Esto desencadeno en horas de búsquedas, pruebas en distintos navegadores, múltiples reinstalaciones de Flash en el equipo, etc.

Tras consultarlo por email con Juanan, se llegó a la conclusión de que la instancia del reproductor de Babelium no se estaba generando adecuadamente.

Para arreglarlo, hubo que entrar en moodle, donde existe una versión del plugin de Babelium en funcionamiento. Investigando el

código fuente y comparando con el utilizado en el proyecto, se descubre el error. El código que se estaba utilizando anteriormente para instanciar el reproductor se encontraba a medias, es decir, solo contenía la parte en la que se quejaba de la falta de Flash.

```
<div id="flashContent">
    <p>To view this page ensure that Adobe Flash Player version 11.1.0 or greater is installed. </p>
    <script type="text/javascript">
        var pageHost = ((document.location.protocol == "https:") ? "https://" : "http://");
        document.write("<a href=\'http://www.adobe.com/go/getflashplayer\'><img src=\'" + pageHost +
            "www.adobe.com/images/shared/download_buttons/get_flash_player.gif\' alt=\'Get Adobe Flash player\' /></a>" );
    </script>
</div>
```

Ilustración 38: Código incompleto Babelium player

Se ve que en este código no se crea nada, tan solo enlaza a la página web de adobe para descargar Flash en el equipo.

En la siguiente imagen podemos ver el objeto babeliumPlayer utilizado correctamente:

```
<object type='application/x-shockwave-flash' id='babeliumPlayer' name='babeliumPlayer'
align='middle' data='http://babeliumproject.com/babeliumPlayer.swf' width='640' height='380'
style='height: 332px; width: 500px;'>
    <param name='quality' value='high'>
    <param name='bgcolor' value='#000000'>
    <param name='allowscriptaccess' value='always'>
    <param name='allowfullscreen' value='true'>
    <param name='wmode' value='window'>
    <param name='flashvars' value='locale=es&amp;forcertmpt=1&amp;jsCallbackObj='>
</object>
```

Ilustración 39: Instancia de objeto Babelium player

Este contratiempo absorbió un par de días de trabajo y, como suele ser a menudo, es a causa de una tontería. En este caso la tontería fue el desconocimiento del uso de objetos mediante HTML5, si se hubiera comprendido correctamente la naturaleza del problema no se habría tardado tanto.

8. API Babelium

En la reunión con el tutor, tras la perdida del disco duro del equipo de trabajo, Juanan sugirió el uso de Digital Ocean para alquilar un servidor en el que montar el API de Babelium.

De esta forma se consigue el control total sobre el servidor y además permite al tutor asistir directamente con cualquier problema que pudiera surgir.

Tras darse de alta en el servicio de Digital Ocean y configurar el servidor con la ayuda de Juanan, se pasó a disponer de un servidor funcionando con el API de Babelium para poder seguir avanzando con el proyecto.

El objetivo de este paso sería aprender a utilizar el servidor mediante las herramientas PuTTY y MySQL Workbench, así como entender los distintos servicios que incluye el API de Babelium.

Una vez se está preparado para seguir, se decide empezar a transformar la aplicación para que deje de usar simulaciones y pase a utilizar llamadas al servidor con datos reales.

9. Comunicación asíncrona con el API de Babelium

Para poder conectar con el servidor desde la aplicación es necesario utilizar llamadas AJAX a ficheros PHP, en las que se pasa la información que el servidor necesita para devolver la respuesta deseada por el usuario. A continuación se muestra un ejemplo con la funcionalidad de login:

```
var context = $(this);
$.ajax({
    url: '/php/login.php',
    type: 'POST',
    dataType: "json",
    data: {
        user: user,
        pass: pass
    }
```

Ilustración 40: Ajax inicio

Podemos ver como se hace una referencia a la localización del archivo PHP a ejecutar en el parametro url, el tipo de consulta que se va a hacer en type, el tipo de datos que se van a enviar en dataType y en data se incluye la información que el servidor necesita para responder, en este caso el usuario y la contraseña del usuario.

```
}).done(function(data) {
    if(data.response !== "wrong_password" && data.response !== "wrong_user" && data.response !== "inactive_user")
    {
        alert("Login successful");
        context.dialog("close");
        var currentUser = new User();
        currentUser.set(data.response);
        new UserNavOn({model: currentUser});
    }
    else
    {
        if(data.response === "wrong_password")
        {
            alert("Password is not correct");
            $('#password').val("");
        }
        else
        {
            if(data.response === "wrong_user")
            {
                alert("User is not correct");
                $('#user').val("");
                $('#password').val("");
            }
            else
            {
                alert("User is not active");
                $('#password').val("");
            }
        }
    }
```

Ilustración 41: Ajax done

Dentro de done() se refleja que es lo que pasa si todo sale bien y el servidor envía una respuesta. En este caso, se comprueba que el servidor no haya contestado que o el usuario o la contraseña son incorrectos o que el usuario no ha activado su cuenta y dependiendo del caso que sea alertara al usuario de una forma u otra.

```
}).fail(function(xhr, status, error) {
    alert("Error: Login failed");
});
```

Ilustración 42: Ajax fail

Por último, dentro de fail() se incluye que debe pasar si se produce un error. En este caso simplemente se avisa al usuario de que se ha producido un error y ya está.

Esta es la estructura normal de una llamada AJAX, pero para cargar colecciones de Backbone no hace falta hacer esto, siempre y cuando al crear una colección se haga incluyendo un apartado url igual que el de AJAX:

```
var VideoList = Backbone.Collection.extend({
    model: Video,
    url: 'php/videoList.php',

    initialize: function()
    {

    }
});
```

Ilustración 43: Colección Backbone usando php

Para que una instancia de la colección VideoList se cargue usando el fichero videoList.php debe ejecutarse la siguiente linea:

```
videos.fetch().done(function ()
{
```

Ilustración 44: fetch de colección

Entre las llaves {} de esa función se incluye lo que se quiere hacer una vez cargada la colección, es el equivalente de done() en AJAX.

En cuanto a los ficheros PHP, para que las llamadas AJAX tengan efecto, los ficheros PHP deben separar los datos enviados desde la aplicación y prepararlos para enviárselos al servidor. Para contactar con el servidor, todos los ficheros PHP de este proyecto utilizan el fichero babelium_gateway, es el encargado de conectar con el servidor alquilado en Digital Ocean.

A continuación se mostrará a modo de ejemplo el fichero PHP que contiene la llamada al servicio processLogin del servidor:

```php
<?php

require_once("babelium_gateway.php");
$CFG = new Config();
$g = new babelium_gateway();

$user = $_POST['user'];
$pass = $_POST['pass'];

$params = array();

$params['username'] = $user;
$params['password'] = sha1($pass);
$login = $g->serviceCall('http','processLogin', $params);
```

Ilustración 45: Fichero php para procesar login

Podemos ver como se extraen los datos enviados por AJAX y se almacenan en el array $params. Como hay una contraseña se encripta antes de enviarla por razones de seguridad y, mediante la función serviceCall de babelium_gateaway.php se envía la petición al servidor junto con los datos necesarios.

Una vez terminado todo este largo proceso de conversión a PHP y AJAX, se decidió implementar la sección de evaluate de la aplicación. Como las funcionalidades son muy similares a las de la sección de practice no surge ningún problema durante la implementación.

10. Practice y evaluate

Tras conseguir hacer funcionar las listas, paginación y carga de videos en la sección de evaluate, se incluyó en el paquete de trabajo poder ver vídeos para corregir y verlos antes de grabarse practicando también.

Para corregir vídeos, existe un formulario basado en estrellas para evaluar el vídeo y además cuenta con un cuadro de texto para dar un mensaje al usuario que grabó el ejercicio.

Para realizar los ejercicios se cuenta con una interfaz basada en botones que permiten empezar, acabar, pausar, reiniciar, cancelar o subir la grabación. Todas estas son funcionalidades del reproductor de Babelium.

El único problema que surgió en este apartado fue que el reproductor decía que no recibía los datos del vídeo a reproducir. Se corrigió cambiando el orden de los parámetros que se pasan a la función que carga un video en el reproductor de Babelium, siguiendo el siguiente orden:

```
init("{{ nombre del servidor }}",
"{{ valor de idioma }}",
"{{ constante que no se debe cambiar }}",
'{{ JSON con los datos del ejercicio }}',
'{{ JSON con los subtítulos del ejercicio }}',
```

```
'{{ JSON con los datos de la respuesta }}',
'{{ JSON con los subtítulos de la respuesta}}');
```

En caso de que se vaya a ver una respuesta se rellenan los 2 últimos y se dejan los 2 anteriores vacíos. Si, por el contrario, queremos ver un ejercicio antes de practicarlo, la forma de que se ejecute es pasando los datos en los campos 4 y 5, dejando vacíos los 2 últimos.

En la reunión siguiente, se tomó la decisión de terminar con la funcionalidad de la aplicación permitiendo grabar ejercicios a los usuarios. Además, por sugerencia de Juanan, se decide que los templates que utilizan las vistas de Backbone sean utilizados desde un fichero html externo a la vista, con el objetivo de facilitar su modificación en el futuro. Si el template es muy grande y esta incluido en la vista, puede ser difícil modificarlo ya que cada línea debe estar metida entre comillas.

Para poder utilizar los templates desde ficheros externos simplemente se debe ejecutar las siguientes lineas al formar la vista. En este caso, es el template de la barra que se muestra si el usuario no está conectado.

```
var ctx = this;
$.get("themes/babelium/templates/userNavOff.html",function(data){
    template = _.template(data,{});
    ctx.$el.html(template);
},'html');
```

Ilustración 46: Referencia a template externo

En cuanto a la grabación de ejercicios, como tan solo consiste en la utilización de la funcionalidad del reproductor de Babelium, no ha supuesto prácticamente ningún problema. El único problema digno de mención que surgió fue que el tamaño que se había puesto al cuadro asignado al reproductor era tan grande que las marcas que aparecen con los tiempos y los subtítulos durante la grabación de un ejercicio no se mostraban, haciendo imposible grabar el ejercicio.

Diagrama de clases

A continuación, se incluye el diagrama de clases que contiene todos los modelos, vistas y colecciones empleados junto con el router.

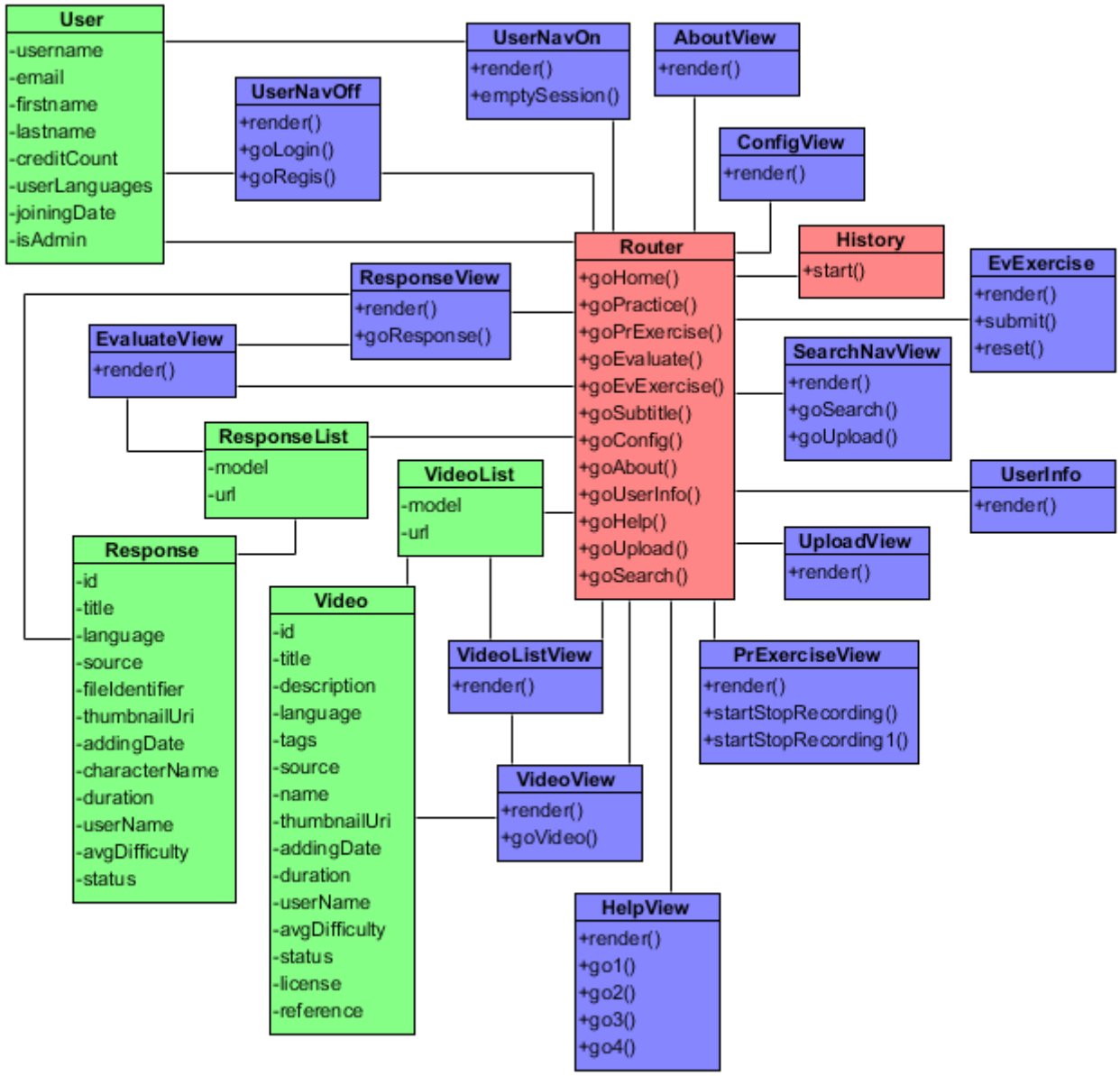

Ilustración 47: Diagrama de clases

El diagrama esta dividido en colores para facilitar la identificación de cada parte del patrón MVC. Los modelos y las colecciones se representan mediante el color verde, las vistas mediante el color azul y con el color rojo el Router y la clase History.

Se indica mediante un guión (-) los atributos de los elementos y mediante una cruz (+) las funciones.

Todos los elementos Backbone (salvo el Router y History) disponen de una función llamada initialize, se ejecuta por defecto al crear una

instancia de dicho elemento. Como la tienen todos, se ha decidido no incluirlo en cada cuadro.

Como era de esperar en una aplicación desarrollada según el patrón MVC, el elemento que cumple la función de controlador, en este caso la clase Router, se encuentra en el centro de la aplicación en todo momento.

DIAGRAMA DE CASOS DE USO

Una vez vistos todos los elementos comprendidos en la aplicación, lo siguiente será observar que posibilidades tiene un usuario de Babelium. Para ello, se dispone de el siguiente diagrama de casos de uso:

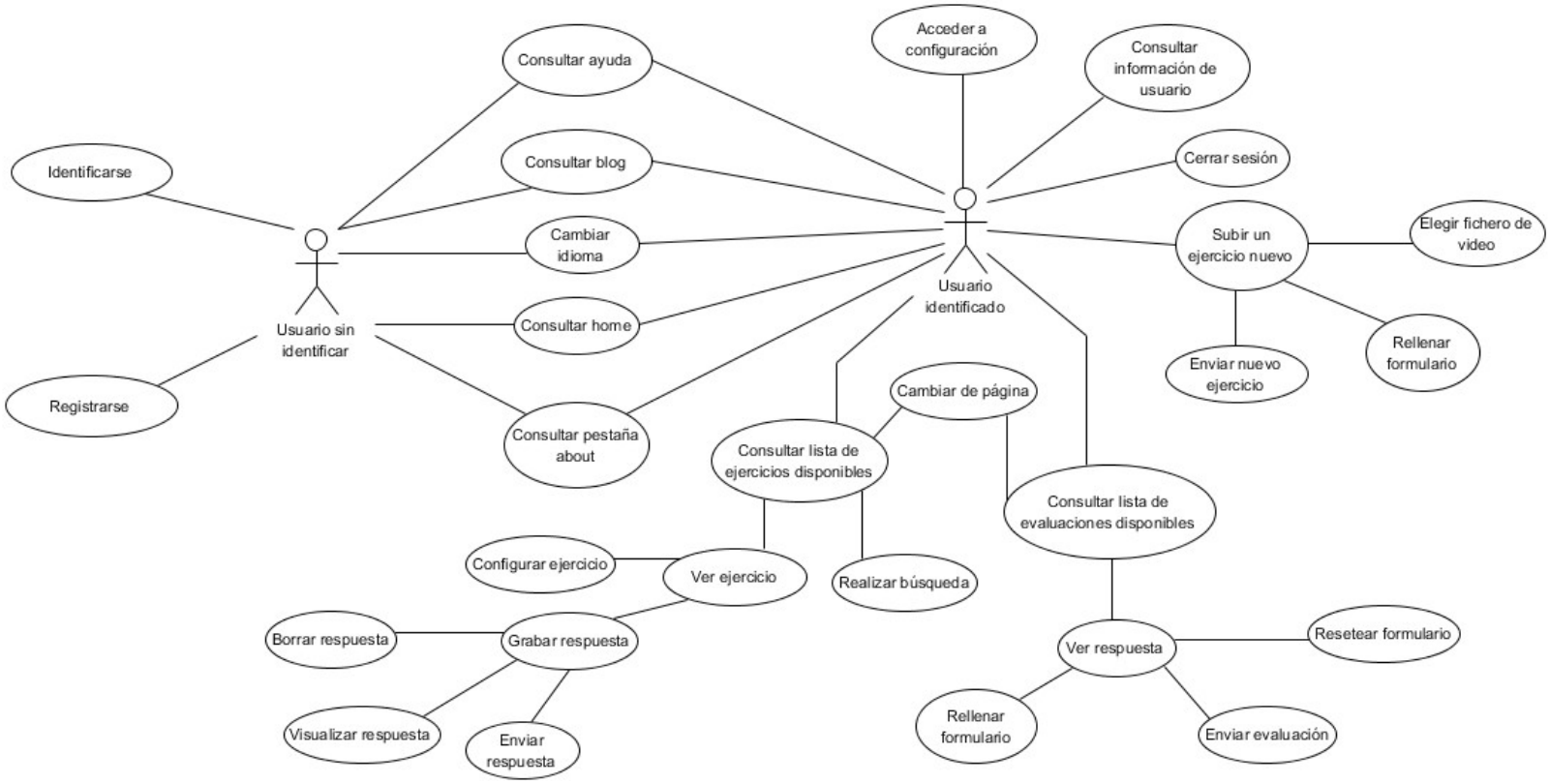

Ilustración 48: Diagrama de casos de uso

El diagrama muestra todas los posibles cursos de acción que un usuario puede tomar, tanto si esta identificado en la aplicación como si no.

Los usuarios sin identificar se ven muy limitados ya que no pueden hacer prácticamente nada.

Los usuarios identificados pueden acceder a todo el contenido de la aplicación salvo a la identificación y el registro, si quiere disponer de estas funcionalidades de nuevo deberá cerrar sesión primero.

DIAGRAMAS DE SECUENCIA

Habiendo visto tanto los elementos que participan en la aplicación y las funcionalidades disponibles para los usuarios, se va a describir en detalle el funcionamiento de algunas de estas funcionalidades. En concreto, van a ser las funcionalidades de identificación y enviar una evaluación.

El sistema de colores utilizado es el mismo que en el diagrama de clases y además, los ficheros php se representan con el amarillo, el API de Babelium con el color rosa, y tanto la interfaz de usuario como el fichero conector entre el API y la aplicación, babelium_gateway, se representan con el azul.

1. Login

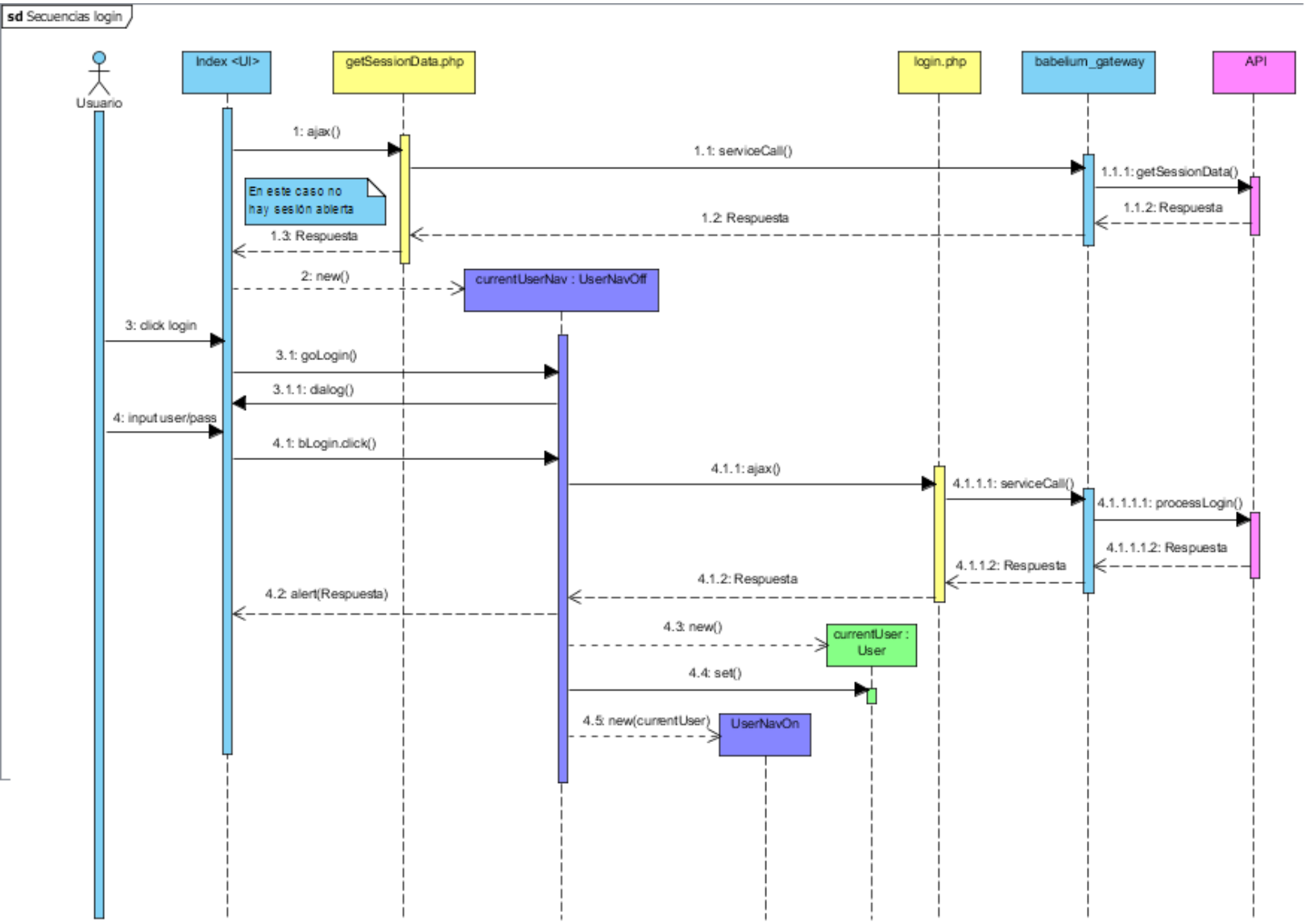

Ilustración 49: Diagrama de secuencia login

Para este diagrama, se ha supuesto la situación en la que el usuario sin identificar acaba de conectarse a la aplicación web. Como acaba de acceder, lo primero que la aplicación hace es comprobar si hay una sesión de usuario abierta en el equipo.

- Del paso 1 al 1.3 se describe el proceso seguido para comprobar si existe una sesión abierta.

- En el paso 2, sabiendo que no hay sesión abierta y el usuario no está identificado, se genera la vista del navegador de usuario que muestra los enlaces a las funcionalidades de login y register.

- Desde el paso 3 al 4.1 se produce la interacción del usuario con la interfaz. El usuario introduce los datos de su cuenta en el cuadro de diálogo que se genera en index con la intervención de currentUserNav y los envía.

- Desde el paso 4.1.1 al 4.2 se gestionan los datos del usuario y se comprueba si son correctos.

- En el paso 4.3 y 4.4 se genera un modelo de usuario en el que se guardan todos los datos del usuario

- Finalmente, en el paso 4.5, se reemplaza la vista del navegador de usuario sin identificar por la de usuario identificado.

2. Cargar ejercicios a evaluar

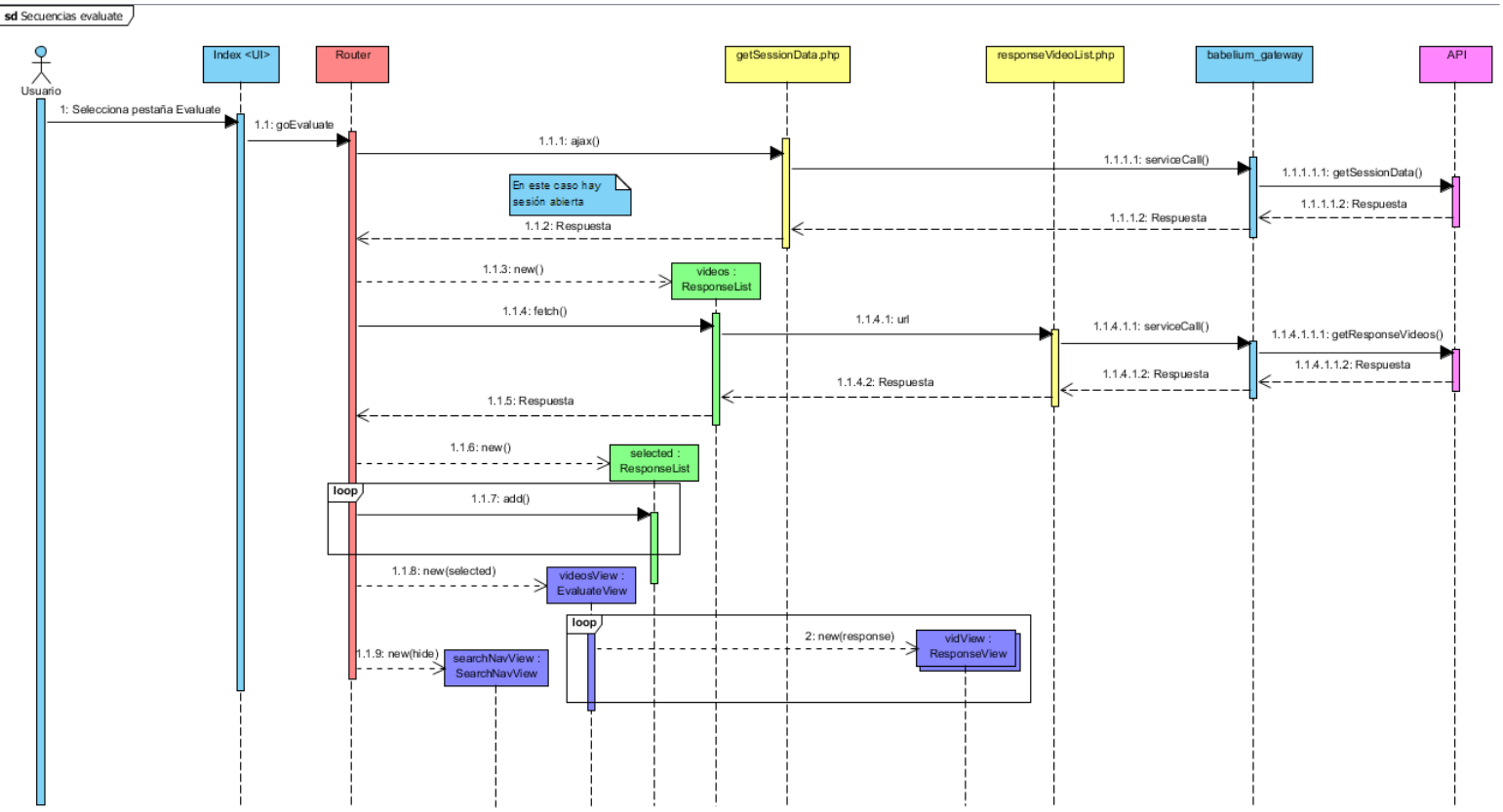

Ilustración 50: Diagrama de secuencia cargar lista de evaluaciones disponibles

La situación de este diagrama es la continuación del anterior, es decir, el usuario se encuentra identificado y desea ver la lista de videos a evaluar.

- Primero se da la orden para acceder a la pestaña de evaluación en los pasos 1 y 1.1.
- Desde el paso 1.1.1 al 1.1.2 se comprueba si existe una sesión abierta, ya que es necesario para acceder a esta funcionalidad.
- En el paso 1.1.3 se genera una colección de ejercicios a evaluar, la cual se llena con todos los ejercicios pendientes de evaluar almacenados en el servidor mediante la función fetch(). La colección almacena en la propiedad url la localización del fichero PHP que debe ejecutar para obtener la información del servidor. Este proceso abarca desde el paso 1.1.4 hasta el 1.1.5.
- En el paso 1.1.6 se genera una segunda colección que servirá para almacenar los ejercicios a mostrar en la página actual.

- Justo después, en el 1.1.7, mediante una iteración implementada en el router, se escogen los ejercicios que tocan en la página actual y se añaden uno a uno a la colección.

- Con la selección hecha, en los pasos 1.1.8 y 2 se genera una vista que contiene las vistas de cada uno de los ejercicios seleccionados.

- Por último, en el paso 1.1.9, se genera la vista de la barra de búsquedas. Esta barra contiene un campo de texto que permite realizar búsquedas en la lista de vídeos para practicar según los tags de los ejercicios. Como ahora estamos en la sección de evaluación no es necesaria, por lo que se oculta.

3. Realizar una evaluación

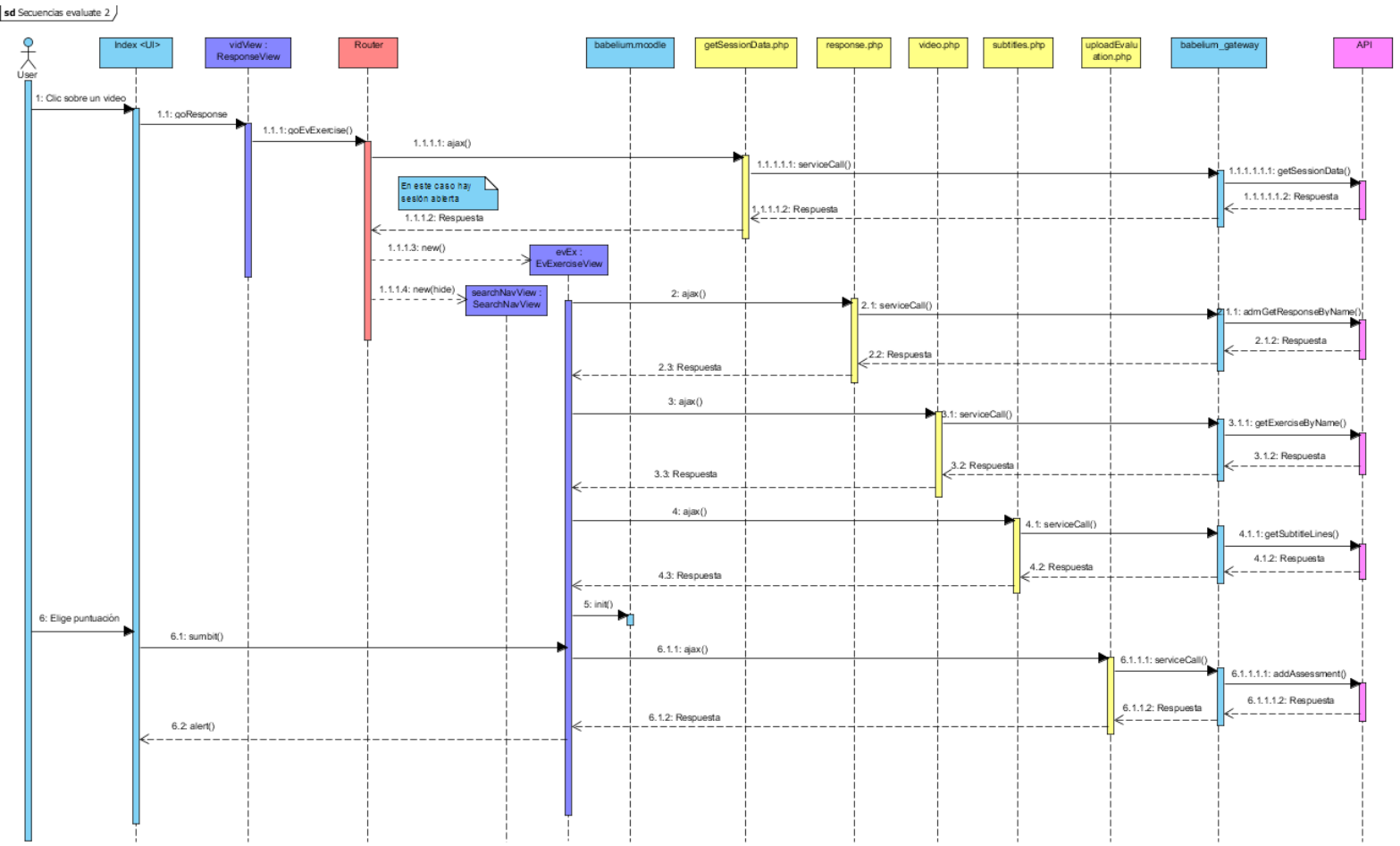

Ilustración 51: Diagrama de secuencia evaluación de un ejercicio

La situación consiste en la continuación del último diagrama, el usuario tiene en pantalla la lista de ejercicios a evaluar disponibles.

- Desde el 1 hasta el 1.1.1 se ve la cadena de llamadas para obtener la vista del ejercicio seleccionado por el usuario.

- Entre los puntos 1.1.1.1 hasta el 1.1.1.2, una vez más, se comprueba la sesión del usuario. Puede parecer redundante pero es necesario, todos los usuarios deben estar identificados para acceder al contenido de la aplicación.

- Desde el 1.1.1.3 hasta el punto 5 se crea la vista de la evaluación del ejercicio seleccionado por el usuario. Para generar esta vista, es necesario obtener los datos necesarios del vídeo para poder utilizar el reproductor de Babelium. Estos datos son los datos del ejercicio en sí, los datos de la grabación del usuario que lo contestó y los subtítulos del vídeo. Una vez obtenidos todos estos datos, se ejecuta init() que se encarga de lanzar el reproductor de Babelium usando toda esa información.

- A partir del punto 6, se guardan los datos del formulario de evaluación y se envían al servidor, el cual devuelve una alerta con el resultado de la operación.

PRUEBAS UNITARIAS

Las pruebas unitarias consisten en comprobar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos que componen la aplicación funcionan correctamente por separado.

Para esta labor se ha utilizado QUnit, como ya se ha explicado previamente en la sección de herramientas.

En este caso, se han realizado pruebas sobre los modelos User y Video de la aplicación. Lo que se ha probado es si, al crear un modelo de User y uno de Video, los atributos se inicializan correctamente.

1. User

En esta prueba se genera una instancia de User y un array de idiomas para asignárselos al User creado.

```
var languages = [{'language':'es_ES','level':'7','positivesToNextLevel':'15','purpose':'evaluate'},
                 {'language':'en_GB','level':'4','positivesToNextLevel':'15','purpose':'evaluate'},
                 {'language':'','level':'0','positivesToNextLevel':'','purpose':''}];

var user = new User({username:'test',
                     email:'example@example.com',
                     firstname:'Te',
                     lastname:'St',
                     creditCount:'0',
                     userLanguages:languages,
                     joiningDate:'1-1-1',
                     isAdmin:'No'});
```

Ilustración 52: Instancia de User y array de idiomas

La tercera lengua se rellena con atributos vacíos para probar la reacción de la aplicación.

Todos los atributos de User deben permanecer igual que en la ilustración 52 salvo los atributos *language* y *level* de todas las lenguas del array.

Al crear un User, los valores de los atributos mencionados son transformados para cumplir con las necesidades de la aplicación.

El valor del atributo *language* se reemplaza por una referencia a la bandera representativa del lenguaje y el valor de *level* es reemplazado por su equivalente en la escala de niveles de idiomas (por ejemplo, en_GB de nivel 4 pasa a ser flag_united_kingdom de nivel B2).

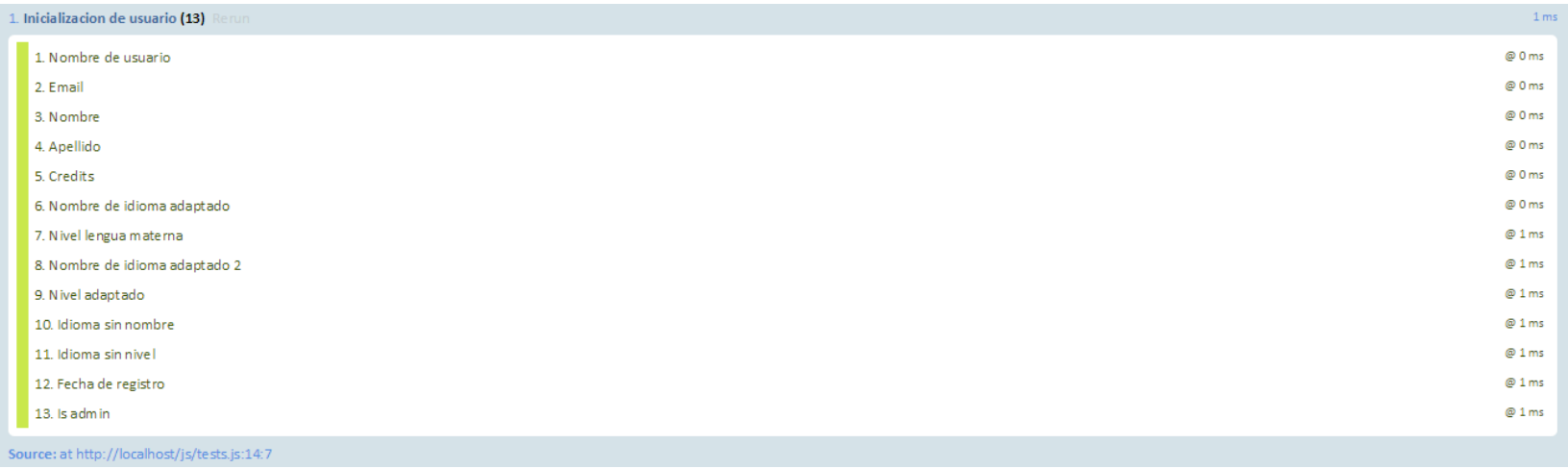

Ilustración 53: Resultados de pruebas sobre User

Podemos ver que los resultados son todos positivos, incluso las pruebas con el idioma sin valores. Esto se debe a que los idiomas sin valores son tratados adecuadamente asignándoles valores que permiten a la aplicación identificar el problema.

2. Video

En esta prueba se generan dos instancias de Video, una correctamente definida y otra definida con valores inapropiados.

```
var video = new Video({id:'1234',
                       title:'Prueba',
                       description:'Pruebapruebaprueba',
                       language:'es_ES',
                       tags:'tag',
                       source:'Prueba.test',
                       name:'Test',
                       thumbnailUri:'Prueba/test',
                       addingDate:'1-1-1',
                       duration:'300',
                       userName:'Tester',
                       avgDifficulty:'3',
                       status:'Avaliable',
                       license:'asdf',
                       reference:'ref'});

var video2 = new Video({language:'',
                        duration:'-300',
                        avgDifficulty:0});
```

Ilustración 54: Instancias de Video

Al generar una instancia de Video, todos los atributos se mantienen igual salvo los atributos *language*, *duration* y *avgDifficulty*. Estos cambios se producen, una vez más, para cumplir con las necesidades de la aplicación.

Los atributos *language* y *avgDifficulty* sufren el mismo cambio que en la prueba anterior (avgDifficulty es el equivalente de level en el modelo Video).

El atributo *duration* es recibido siempre en segundos, pero cuando se quiere mostrar la duración de un vídeo interesa mostrar el tiempo en formato "minutos":"segundos". Para ello, su valor se altera para seguir ese formato. En el caso de video devolverá 05:00 y en el caso de video2 devolverá 00:00 para indicar que algo no va bien.

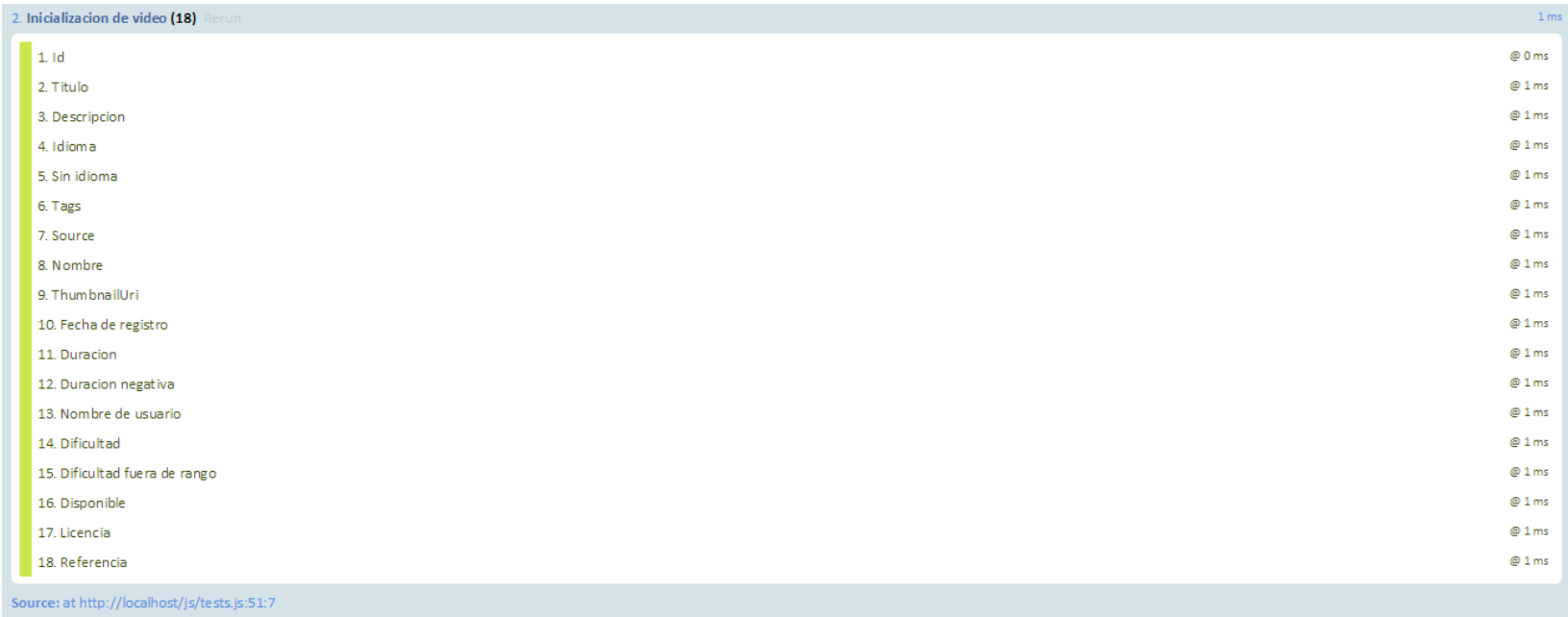

Ilustración 55: Resultados de pruebas sobre Video

Una vez más, todos los resultados son correctos, los modelos funcionan perfectamente.

Conclusiones

1. Resultado del proyecto

Gracias a este proyecto se han adquirido múltiples aptitudes y experiencias:

- Experiencia con proyectos de un tamaño mayor al habitual durante el grado.
- Experiencia documentando aplicaciones.
- Experiencia investigando tecnologías desconocidas.
- Amplios conocimientos sobre Backbone.js.
- Experiencia con herramientas de gestión de proyectos como GitHub.
- Asentamiento de los conocimientos básicos sobre HTML5, jQuery y JavaScript adquiridos durante el grado.
- Conocimientos básicos sobre Angular.js y Ember.js adquiridos durante la investigación.

Personalmente, estoy satisfecho con el resultado del proyecto y lo aprendido durante el camino. Además, se han cumplido todos los objetivos del proyecto y todos los objetivos personales también.

2. Futuro

De cara al futuro, a parte de terminar de atar todos los cabos sueltos que pueda haber, habría sido interesante incluir la funcionalidad de subtitulado de vídeos, pero habiendo sido descartada del proyecto por el tutor no hay mucho que hacer al respecto.

También sería interesante renovar el aspecto general de la aplicación. Tiene un aspecto poco moderno y nada dinámico para la época en la que nos encontramos.

Para terminar, considero que el cambio más necesario e importante para Babelium ahora mismo es dejar de utilizar un reproductor hecho con Flash y empezar a utilizar un reproductor de HTML5.

3. Que cambiaría

Habiendo llegado a este punto del proyecto, una pregunta frecuente puede ser: Si volvieras atrás en el tiempo al momento en el que empezaste el proyecto ¿Qué cambiarías de lo que has hecho?

Estoy bastante contento con el resultado de mi trabajo, así que, en principio, no cambiaría nada. Lo único de lo que me arrepiento es de las pésimas planificaciones iniciales realizadas pero es normal con herramientas y un framework desconocidos.

Una cosa que si que se me ha ocurrido en algún momento es que me gustaría implementar la aplicación usando Angular en vez de Backbone.

Desde la fase del proyecto en la que se investigaron los distintos frameworks, mi interés por Angular era parecido al interés sentido por Backbone, los factores decisivos fueron el tamaño de las librerías y la mayor dificultad de aprendizaje con Angular, pero, si pudiera volver a hacerlo con tiempo y sin presiones, habría sido interesante y habría sido útil para dominar este framework tan bien como domino

Bibliografía

- Addy Osmani. *Developing Backbone.js Applications.* O'Reilly Media(2013)

Referencias Web

- Frameworks: Información sobre los distintos frameworks para poder elegir cual usar en el proyecto.
  - http://www.lostiemposcambian.com/blog/javascript/backbone-vs-angular-vs-ember/
  - http://speckyboy.com/2014/10/14/backbone-vs-knockout/
  - https://www.youtube.com/watch?v=bODntw206ZQ
  - http://blog.fusioncharts.com/2014/08/angularjs-vs-backbone-js-vs-ember-js%E2%80%95choosing-a-javascript-framework-part-2/

- Backbone.js: Información, tutoriales y ejemplos sobre Backbone.js para la implementación de la aplicación.
  - https://www.youtube.com/user/DevCodela
  - http://codebeerstartups.com/2012/12/a-complete-guide-for-learning-backbone-js/
  - http://addyosmani.github.io/backbone-fundamentals/

- Metodologías de trabajo: Información sobre los distintos tipos de metodologías de gestión del trabajo para decidir el que se va a usar.
  - https://www.softeng.es/es-es/empresa/metodologias-de-trabajo.html
  - http://danielgrifol.es/metodologias-agiles-de-desarrollo-de-software/

- Análisis de riesgos: Ejemplos de posibles riesgos durante un proyecto como referencia.
  - https://iaap.wordpress.com/2007/09/26/21-riesgos-tipicos-en-proyectos/

- Gestión de proyectos: Información sobre como gestionar un proyecto y consejos sobre la documentación.
  - http://www.eoi.es/wiki/index.php/Gesti%C3%B3n_de_proyectos

- Digital Ocean: Tutoriales sobre como establecer conexión con el servidor alquilado.
  - https://www.digitalocean.com/community/tutorials/how-to-use-ssh-keys-with-putty-on-digitalocean-droplets-windows-users
  - http://www.softzone.es/manuales-software-2/uso-basico-de-putty-ssh/

- QUnits: Tutoriales sobre el uso de la herramienta JavaScript de pruebas.
  - http://www.sitepoint.com/getting-started-qunit/
  - http://code.tutsplus.com/tutorials/how-to-test-your-javascript-code-with-qunit--net-9077

ANEXO 1

Babelium API

This section describes the PHP API for accessing Babelium services from remote sites.

Classes and interfaces

Auth

Allows the user to authenticate on the Babelium system

Logs the user out and clears the session data.

```
doLogout() : boolean
```

Generates a random communication token that will be used in the following API calls.

```
getCommunicationToken(String $secretKey) : boolean | string
```

Checks the provided authentication data and logs the user in the system if everything is ok

```
processLogin(\stdClass $user) : mixed
```

Sends again the account activation email if the provided user is valid and not active.

```
resendActivationEmail(\stdClass $user) : string
```

Evaluation

This class stores all the methods that have something to do with the evaluation of the exercises and their responses

Adds new assessment data to the provided response

```
addAssessment(\stdClass $evalData)
```

Adds new assessment data to the provided response (plus video-comment data)

```
addVideoAssessment(\stdClass $evalData)
```

Retrieves the details of the assessment(s) of a particular response

```
detailsOfAssessedResponse(int $responseId) : array
```

Retrieves the assessment data of a particular response to build a chart

```
getEvaluationChartData(int $responseId) : array
```

Retrieves the responses which the current user evaluated to another user

```
getResponsesAssessedByCurrentUser() : array
```

Retrieves those responses of the current user that have been evaluated by another user

```
getResponsesAssessedToCurrentUser() : array
```

Retrieves all the responses that haven't been evaluated (and can be evaluated) by the current user

```
getResponsesWaitingAssessment() : array
```

Exercise

Class to perform exercise related operations

Adds a new score to the provided exercise id

```
addExerciseScore(\stdClass $score) : \stdClass
```

Reports about an exercise/video that breaks the rules of the site

```
addInappropriateExerciseReport(\stdClass $report) : int
```

Saves information about a exercise that's being just uploaded and marks it to be reencoded to meet Babelium's video specification via a cron task.

```
addUnprocessedExercise(\stdClass $exercise)
```

Saves information about a exercise that's being published using the webcam.

```
addWebcamExercise(\stdClass $exercise)
```

Filters an exercise list using a language setting of the currently logged-in user

```
filterByLanguage(array $searchList, String $languagePurpose) : array
```

Returns the bayesian average score of an exercise id (the arithmetic average score is not accurate information in statistical terms so a weighted value is used instead)

```
getExerciseAvgBayesianScore(int $exerciseId) : \stdClass
```

Gets information about an exercise using its id

```
getExerciseById($id) : \stdClass
```

Gets information about an exercise using its name (or filehash)

```
getExerciseByName($name) : \stdClass
```

Returns the descriptors of the provided exercise (if any)

```
getExerciseDescriptors(int $exerciseId) : mixed
```

Gets the available subtitle languages for the provided exercise id

```
getExerciseLocales(int $exerciseId) : array
```

Gets a list of all the available exercises sorted by date

```
getExercises() : mixed
```

Gets a list of all the exercises that are available and whose subtitling has not been marked as complete.

```
getExercisesUnfinishedSubtitling() : mixed
```

Gets a list of all the exercises that are available and ready to be practiced (it has subtitles and those subtitles are marked as complete).

```
getRecordableExercises() : mixed
```

Add the descriptors this exercise helps to achieve

```
insertDescriptors(array $descriptorIds, int $exerciseId)
```

Inserts a list of tags in the database.

```
insertTags(array $tags, $exerciseId)
```

Parses a list of language common framework descriptors using their id

```
parseDescriptors(array $descriptors) : array
```

Parses the tags that were sent with the upload form

```
parseExerciseTags(String $tags) : array
```

Check if the user has already rated this exercise today

```
userRatedExercise(\stdClass $score) : \stdClass
```

Check if the user has already reported about this exercise

```
userReportedExercise(\stdClass $report) : \stdClass
```

Register

This class performs signup and user activation operations

Activates the user profile so that the user is able to use the system

```
activate(\stdClass $user)
```

Sign-up a new user in the system

```
register(\stdClass $user)
```

Response

This class performs exercise response related operations

Makes a response public which means it can be assessed by other users with enough knowledge of the target language

```
makePublic(\stdClass $data)
```

Saves a response attempt of the user so that other users can assess his/her work

```
saveResponse(\stdClass $data) : int
```

Subtitle

This class performs subtitle related operations

Returns all the subtitles available to the provided exercise

```
getExerciseSubtitles(int $exerciseId) : mixed
```

Returns an array of subtitle lines for the given exercise.

```
getSubtitleLines(\stdClass $subtitle) : mixed
```

Gets the lines of the provided subtitle identificator

```
getSubtitleLinesUsingId(int $subtitleId) : mixed
```

Wrapper function for adding a new subtitle and subtitle lines to the database.

```
saveSubtitles(\stdClass $subtitleData) : mixed
```

User

This class performs user related operations

Substitute the old password of the user with the new one (both passwords are provided as their SHA1 hashes).

```
changePass($oldpass, $newpass)
```

Delete the videos of the current user given as an array of string video-identifiers.

```
deleteSelectedVideos($selectedVideos)
```

Select the top ten ranked user names and credits, according to their credit count balance.

```
GetTopTenCredited()
```

A keepAlive function called automatically by the client application in order to keep the user session open.

```
KeepAlive()
```

Modifies the languages' knowledge level of current user. First, the method deletes all of the information related to the languages mastered by the users. Then, using the array of UserLanguageVO objects given as argument, it sets all the languages and mastering levels provided.

```
modifyUserLanguages($languages)
```

Update the real name, surname and email of the current user (data is provided as a UserVO object).

```
modifyUserPersonalData($personalData)
```

Update the user provided video exercise properties (title, description, tags, license, reference and language) with the data given as argument (ExerciseVO object).

```
modifyVideoData($videoData)
```

Reset the password of the username or email provided (the user will receive a confirmation mail).

```
restorePass($username)
```

Get all the videos uploaded or recorded by current user.

```
RetrieveUserVideos()
```

VideoProcessor

Helper class to perform media transcoding tasks.

Fixes the Flash Player 11.2.x bug that makes audio-only FLV files non-playable by adding a 8x8px black image for the video stream.

```
addDummyVideo(String $dummyImagePath, String $inputPath, String $outputPath) : String
```

Extract an audio subsample from the input file provided as argument (starting from the $startTime second and up to the $endTime second, applying an optional volume boost between 0 and 2600) and save it as the output file name given as second parameter.

```
audioSubsample($inputFilePath, $outputFilePath, $startTime, $endTime, $volume)
```

Checks if the provided file pertains to a certain mime type category

```
checkMimeType(String $filePath, int $type) : boolean
```

Concatenates the wav files of the provided path that begin with the provided prefix and puts the concatenated wav file in the provided path.

```
concatAudio(String $inputPath, String $filePrefix, String $outputPath)
```

Extract the audio channel (or channels, depending on the number of audio channels given by the $audioChannels parameter) from a video file provided as first argument. The result will be saved in $outputFilePath an encoded with the audio sample rate provided as last argument.

```
demuxEncodeAudio($inputFilePath, $outputFilePath, $audioChannels, $audioSamplerate)
```

Merges two input videos in one output video file with provided dimensions ($widthx$height), muxing the provided input audio file.

```
mergeVideo(String $inputVideoPath1, String $inputVideoPath2, String $outputVideoPath,
String $inputAudioPath, int $width, int $height)
```

Add the audio channel provided in the $inputAudioPath parameter to the $inputVideoPath video file and save the result in $outputVideoPath. The audio channel will be encoded in mp3 format, with 2 channels and a 44100 sample rate.

```
muxEncodeAudio($inputVideoPath, $outputVideoPath, $inputAudioPath)
```

Determines if the given parameter is a media container and if so retrieves information about it's different streams and duration.

```
RetrieveMediaInfo(string $filePath)
```

Returns a provided character long random alphanumeric string

```
str_makerand(int $length, boolean $useupper, boolean $usenumbers)
```

Takes various images from the provided video and creates a folder to store them.

```
 $snapshotHeight)takeFolderedRandomSnapshots(string $filePath, $thumbPath, $posterPath, $thumbnailWidth,
$thumbnailHeight, $snapshotCount)
```

Takes a thumbnail image of the provided video and leaves it at the defined destination.

```
takeRandomSnapshot(string $filePath, string $outputImagePath, $snapshotWidth,
```

Transcodes the provided video file into an FLV container video with stereo MP3 audio.

```
transcodeToFlv(string $inputFilepath, string $outputFilepath, int $preset)
```