Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

REALIZACIÓN DE UN SISTEMA WEB COMPLEJO PARA USO DIDÁCTICO

TRABAJO FIN DE MÁSTER

Daniel Benítez Águila

2014

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

REALIZACIÓN DE UN SISTEMA WEB COMPLEJO PARA USO DIDÁCTICO

Autor
Daniel Benítez Águila

Director
Joaquín Salvachúa Rodríguez

Departamento de Ingeniería de Sistemas Telemáticos

2014

Resumen

El desarrollo de aplicaciones web está sufriendo grandes cambios en los últimos años. El crecimiento de servicios en la nube es una de las principales razones de esos cambios así como el incremento del número de tecnologías y soluciones asociadas a ello.

Este Trabajo Fin de Máster se centra en la creación de un sistema web complejo para uso didáctico. Esto es un sistema web que pueda servir de ayuda para aquellos principiantes que deseen iniciarse en el desarrollo web actual.

Se hará una breve introducción, donde se expondrá el contexto bajo el cual se va a desarrollar el trabajo y se definirán qué objetivos quieren alcanzarse con la realización del mismo.

La principal meta de este trabajo es el diseño, desarrollo e implementación del sistema web. Para llevar a cabo dicha tarea, se analizarán las diferentes tecnologías que se estiman más novedosas y significativas en el ámbito del desarrollo de aplicaciones web, ya que se considera de vital importancia tener una panorámica de qué hay a día de hoy para poder elegir aquello que se más se ajuste a nuestras necesidades. Estas tecnologías serán lenguajes de programación, frameworks, motores de plantillas, bases de datos y plataformas de despliegue.

Se explicará en detalle todos los componentes que forman el sistema web, desde el diseño del modelo de datos hasta su despliegue en la nube. Se describirá el funcionamiento del sistema web a través de las diferentes páginas que se muestran en la interfaz de usuario para facilitar el uso, la navegabilidad y la comprensión del sistema.

Por último, se comentarán las conclusiones obtenidas de la creación del sistema web. También se propondrán diferentes líneas de investigación y actuación futuras para la integración de nuevas funcionalidades en el sistema.

Abstract

Web application development is facing big changes within last years. The growth of the services in cloud is the main reason of these changes and the increasing number of technologies and solutions associated with this.

This Master Thesis focuses on the creation of a complex web system for educational use. This is a web system that can be helpful for beginners who wish to start the current web development.

Will be a brief introduction, where will be exposed the context in which it will develop the work and will be defined what objectives are to be achieved with the realization.

The main goal of this work is the design, development and implementation of the web system. To carry out this task, will be analyzed the different technologies that are considered most innovative and significant in the field of web application development, as it is considered vital to have an overview of what's on today to choose what fits best to our needs. These technologies will be programming languages, frameworks, template engines, databases and deployment platforms.

Will be explained in detail all components that form the web system, from the design of the data model to its cloud deployment. Will be described the operation of web system through the different pages that are displayed in the user interface in order to ease use, navigability and understanding of the system.

Finally, will be discussed the conclusions obtained from the creation of the web system. Also, will be proposed different lines of research and future actions for the integration of new functionalities of the system.

Índice general

Índice de figuras

Índice de tablas

Siglas

AJAX	*Asynchronous JavaScript And XML*
API	*Application Programming Interface*
CRM	*Customer Relationship Management*
CSS	*Cascade Style Sheet*
DBMS	*DataBase Management System*
EJS	*Embedded JavaScript*
Haml	*HTML Abstraction Markup Language*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
JSP	*JavaServer Pages*
MVC	Modelo-Vista-Controlador
NoSQL	*Not Only SQL*
npm	*node package manager*
ORM	*Object Relational Mapping*
REST	*REpresentational State Transfer*
SQL	*Structured Query Language*

1 Introducción

Este documento describe el trabajo realizado en el Trabajo Fin de Máster de la titulación Máster Universitario en Ingeniería de Redes y Servicios Telemáticos para el diseño, desarrollo e implementación de un sistema web complejo para uso didáctico.

1.1 Contexto

El desarrollo de aplicaciones web ha sufrido un enorme cambio en los últimos años. Antes, el desarrollador que decidía aventurarse en publicar una aplicación de este tipo se encargaba de realizar todo el proceso al completo: escribir el código, disponer de un equipo donde desplegar la aplicación y el almacenamiento de los datos, obtener una dirección IP estática y dirección web para mostrar al público dicha aplicación, hacer labores de mantenimiento y gestión, etc. Desde hace unos años esta situación ha cambiado, en cierta parte debido a la proliferación de las tecnologías en la cloud o nube, que facilitan la tarea del desarrollo de aplicaciones en la web. Esto se consigue abstrayendo y simplificando al desarrollador de funciones como el mantenimiento y el despliegue, para que éste únicamente se centre en el código. Todo ello además prácticamente a coste cero o con precios bastante asequibles.

Ahora bien, un programador que tenga como propósito desarrollar una aplicación web en la nube desde cero se plantea una serie de preguntas: ¿qué lenguaje de programación debería utilizar? ¿qué framework es el más adecuado para ese lenguaje? ¿el almacenamiento de los datos en bases de datos relacionales o NoSQL? Éstas y otras preguntas, que pueden llegar a abrumar al desarrollador, son fruto de la gran variedad de tecnologías existentes hoy en día.

1.2 Objetivos

El presente trabajo tiene por objetivo principal el diseño, desarrollo e implementación de un sistema web complejo para uso didáctico. Para ello se creará una aplicación web que servirá como modelo base a los estudiantes de grado o cualquier otra persona interesada en el desarrollo web para que después puedan reproducir el sistema de manera completa sin problemas o inventar uno nuevo fundamentándose en este trabajo.

Los objetivos secundarios que se pretenden conseguirse con la realización de este trabajo son los siguientes:

- Utilizar las tecnologías del momento que existen en el ámbito del desarrollo de aplicaciones web.

- Mejorar mi formación así como profundizar en conceptos y competencias de otras asignaturas del Máster relacionadas con el entorno web como "Aplicaciones y Sistemas Colaborativos en la Web 2.0" y "Sistemas de Información y Bases de Datos Web".

1.3 Estructura del documento

A continuación, se hace una breve descripción del contenido de cada uno de los capítulos que componen la memoria.

En el capítulo 1, **Introducción**, se expone el contexto de desarrollo del proyecto, así como los objetivos que se pretenden conseguir.

En el capítulo 2, **Estado del arte**, se presenta una descripción general de las diferentes tecnologías existentes relacionadas con el propósito del proyecto, con el fin de situar al lector en el contexto de trabajo. De manera más concreta, se profundiza en los lenguajes de programación más utilizados para el desarrollo de sistemas web, sus frameworks y motores de plantillas más conocidos, las diferentes opciones para el almacenamiento de los datos (bases de datos) y las plataformas de despliegue en la nube.

En el capítulo 3, **Sistema web completo para uso didáctico**, se aborda de manera detallada el desarrollo del sistema web. En primer lugar se realiza una descripción general del sistema. Seguidamente expone el modelo de datos en que se basa dicho sistema. Después se explica la instalación de todos los componentes necesarios así como la implementación del *back-end*. Tras esto, se centra en los recursos que utilizará el sistema, así como en los modelos y vistas que definen y representan esos recursos. Por último se detalla el despliegue del sistema en la nube.

En el capítulo 4, **Resultados**, se describen los resultados obtenidos como consecuencia de la finalización del proyecto, que explican el funcionamiento del sistema web a través de las diferentes pantallas que se muestran en la interfaz de usuario.

En el capítulo 5, **Conclusiones**, se explica qué conclusiones se sacan del trabajo realizado, así como las posibles líneas futuras de trabajo.

2 Estado del arte

2.1 Lenguajes de programación

2.1.1 Java EE

Basada en Java, uno de los lenguajes de programación más populares en la actualidad y el estándar de facto de los desarrolladores durante la última década, la plataforma Java EE (Enterprise Edition) es el "estándar en software de empresa dirigido por la comunidad" [1]. Consiste en una API (*Application Programming Interface*) y un entorno de tiempo de ejecución para el desarrollo de software empresarial (software "a gran escala, multicapa, escalable, confiable y seguro"[2]), que amplían las capacidades de la implementación básica de Java, Java Standard Edition [3]. Esta plataforma utiliza una aproximación multicapa a la hora de desarrollar, utilizando un enfoque dividido en tres capas:

- La capa web: todos los componentes que se encargan de manejar la interacción entre clientes (navegador, aplicación *standalone*…) y la capa de negocio.
- La capa de negocio: encargada de la lógica de negocio, esto es, la funcionalidad de un dominio de negocio concreto.
- La capa de sistemas de información empresariales: consiste en el código encargado del mantenimiento de servidores de bases de datos, sistemas de planificación y otras fuentes de datos.

Entre las APIs concretas que se especifican en Java EE destacan las siguientes:

Tabla 1. Módulos destacados de Java EE

Nombre	Propósito
javax.servlet.*	Manejar peticiones HTTP (*HyperText Transfer Protocol*)
javax.websocket.*	Manejar conexiones WebSocket
javax.faces.*	Diseñar interfaces de usuario
javax.jms.*	Crear, enviar, recibir y leer mensajes de un sistema de mensajería empresarial
javax.batch.api.*	Lanzar aplicaciones en *background* que necesitan ser ejecutadas de manera periódica

2.1.2 Ruby

Ruby es un lenguaje de programación concebido por Yukihiro Matsumoto el 24 de Febrero de 1993, como un lenguaje de scripting orientado a objetos que fuera natural, en contraste con Perl, del que pensaba que era un "lenguaje de juguete" y Python, del que sentía que la orientación a objetos era un "añadido" al lenguaje. Así, basándose en sus favoritos lenguajes: Perl, Smalltalk, Eiffel, Ada y Lisp; Ruby fue creado como un lenguaje que permitiera programación funcional e imperativa. El enfoque principal del lenguaje es tener apariencia natural por encima de la simpleza, intentar conseguir que el usuario nunca se sienta "perdido" con el lenguaje y disfrutara utilizándolo. Entre las características principales de Ruby, podemos recalcar:

- Todo en Ruby es visto como un objeto.
- Todo en Ruby es flexible y redefinible. Por ejemplo, todos los operadores básicos pueden ser redefinidos o llamados mediante otra sintaxis definiendo un nuevo método que los utilice.
- El empleo de bloques, una construcción similar a una función construida mediante clausuras, otorga a Ruby una gran expresividad.
- Empleo de herencia únicamente explícita.
- Manejo de excepciones, con características similares a Java y Python.
- Utilización de un recolector de basura.
- Existencia de API para llamar a Ruby desde C.
- Capacidad multihilo incluso en sistemas que no lo soportan nativamente.
- Carga dinámica de librerías de extensión (si el sistema operativo lo permite).

Además del lenguaje básico, existen distintas implementaciones de Ruby que extienden sus capacidades, como JRuby, que permite a Ruby utilizar la máquina virtual de Java o MacRuby, que permite integrar Ruby con las librerías Cocoa de Apple, para el desarrollo de aplicaciones sobre esta plataforma [4][5].

2.1.3 JavaScript

JavaScript es, junto a HTML5 (*HyperText Markup Language*) y CSS3 (*Cascading Style Sheets*), el estándar de facto en el que está definida la gran mayoría de páginas web presentes en Internet. Es un lenguaje ligero, interpretado, orientado a objetos, dinámico que puede utilizarse de manera imperativa, funcional y orientada a objetos. Fue lanzado por primera vez, con el nombre LiveScript en septiembre de 1995, concebido como un lenguaje ligero que equivaliera a una versión portable del Java de Sun Microsystems. Hoy en día, el nombre JavaScript es una marca registrada de Oracle y el

lenguaje está sufriendo un repunte en popularidad, pasando de ser considerado una herramienta para amateurs a uno de los lenguajes más populares actualmente, con presencia en los clientes, servidores web y un proyecto de estandarización (CommonJS) para definir una librería principal que permita el uso de JavaScript fuera de los navegadores. Entre los usos principales que se le ha dado a lo largo del tiempo, podemos dividirlos en dos grupos, los aplicados a las páginas web y los aplicados fuera del navegador:

1. Dentro del navegador:

    - Carga de nuevo contenido mediante envío de datos al servidor de forma síncrona de manera nativa o asíncrona mediante AJAX (*Asynchronous JavaScript And XML*).

    - Animación de elementos de la página (ampliado mediante la librería jQuery).

    - Validación de datos introducidos por el usuario.

    - Transmisión de información del usuario.

2. Fuera del navegador:

    - Microsoft permite el uso de HTML5 y JavaScript para la construcción de aplicaciones nativas sobre Windows 8.

    - Gran cantidad de pequeñas aplicaciones software, como las extensiones de Google Chrome, los widgets de Apple y los gadgets de Microsoft están implementadas en JavaScript.

    - Ciertas herramientas de la suite creativa de Adobe, como Photoshop o Illustrator permiten la creación de scripts mediante JavaScript.

    - El motor de creación de videojuegos Unity soporta una versión modificada de JavaScript para la creación de scripts.

Entre las características más destacadas del lenguaje están:

- Soporte de programación estructurada basada en C.

- Síncrono y/o asíncrono.

- Tipado dinámico.

- Basado en objetos.

- Basado en prototipos para la implementación de herencia.
- Funciones como métodos y constructores.
- Soporte para expresiones regulares de manera similar a Perl [6][7].

2.1.4 PHP

PHP es un lenguaje de scripting de servidor destinado al desarrollo web, aunque también es utilizado como lenguaje de propósito general. El proyecto que dio lugar a PHP fue desarrollado por Rasmus Lerdorf en 1994, como un conjunto de scripts en Perl que utilizaba para gestionar su página web personal. Llamó a esto PHP/FI: *Personal Home Page/Forms Interpreter*. Tras esto, liberó el PHP Tools versión 1.0 para acelerar la detección de bugs y mejorar el código. Tras ello, la comunidad tomó el mando del desarrollo y en 1997 el ya lenguaje de programación pasó a tener como nombre un acrónimo recursivo PHP: Hypertext Preprocessor. Actualmente, el lenguaje tiene tres usos principales:

- Scripting en el lado de servidor.
- Scripting en línea de comandos.
- Creación de aplicaciones de escritorio, utilizando el paquete PHP-GTK.

PHP es soportado por todos los sistemas operativos mayoritarios y permite escoger el paradigma de programación: programación procedural, orientada a objetos o una mezcla de ambas. Entre las características más destacadas de PHP, podemos enumerar, además de lo ya nombrado:

- Soporte para un amplio rango de bases de datos.
- Soporte para comunicación con otros servicios mediante protocolos como LDAP, IMAP, SNMP, POP3, HTTP, etc.
- Soporte para apertura de sockets de red e interacción mediante cualquier otro protocolo.
- Capacidad de procesamiento de texto, entre los que destaca la compatibilidad con expresiones regulares de Perl y herramientas para parsear documentos XML.
- Soporte (y existencia) de incontables extensiones al lenguaje [8][9].

2.1.5 Python

Python es un lenguaje de alto nivel de propósito general, creado por Guido van Rossum a partir de 1989 como sucesor del lenguaje ABC. La versión actual del lenguaje

es Python 3.0, lanzada el 3 de Diciembre de 2008. Sus características principales son: orientación multiparadigma, soporta orientación a objetos, programación estructurada, programación funcional y orientación a aspectos, tipado dinámico, sintaxis escueta, delimitación de bloques mediante indentación (en lugar de llaves o paréntesis), alta extensibilidad mediante un núcleo básico reducido y una gran cantidad de librerías estándar, recolector de basura automático y resolución de nombres tardía (se ligan los nombres de métodos y variables durante la ejecución del programa). Además de las características relativas estrictamente al lenguaje en sí, su creador y la comunidad anima a los desarrolladores a seguir una serie de convenciones y usos, principalmente:

- El Zen de Python, cuyo comienzo es:
    - *Beautiful is better than ugly*
    - *Explicit is better than implicit*
    - *Simple is better than complex*
    - *Complex is better than complicated*
    - *Readability counts* [10]
- Rechazo de la posibilidad de realizar una acción de varias maneras: una acción debería tener siempre una y solamente una manera obvia de ser realizada.
- Disfrutar del lenguaje: el propio nombre del lenguaje es una referencia a Monty Python, por lo que se anima a incluir referencias a este hecho en la documentación, aparte de, simplemente intentar que el uso del lenguaje sea divertido.

Hoy en día Python es utilizado por grandes empresas como Google, Yahoo u organizaciones como la NASA y el CERN. Mediante el uso de librerías como Matplotlib, NumPy y SciPy, Python está ganando muchos adeptos en el ámbito de la computación científica, incluido el análisis estadístico y matricial, que es dominio tradicional de Matlab y R. Además, es un componente estándar de muchas distribuciones Linux, muchas de las cuales usan incluso instaladores basados en él, es el lenguaje principal del microordenador Raspberry Pi y LibreOffice ha portado todo su código de Java a Python.

Entre todos los usos posibles del lenguaje, Python permite:

- Desarrollo Web.
- Acceso a bases de datos.
- Creación de interfaces de usuario de escritorio.

- Cálculo científico y numérico.
- Desarrollo de software y de videojuegos.

Además de todo ello, Python es desarrollado bajo licencia de código abierto [11][12].

2.1.6 Comparativa entre lenguajes

A la hora de comparar los lenguajes de programación expuestos en las secciones anteriores podemos hallar dos tipos de criterios, en función de su naturaleza: por un lado, los criterios cuantitativos, esto es, los basados en datos empíricos, como cantidad de desarrolladores dedicados a cada uno, el número de aplicaciones creadas con cada una de ellos, etc. Por otro lado, tenemos los criterios cualitativos, como son su ámbito de aplicación o, ciertas características como el soporte a un tipo de programación concreta. Ahora bien, un enfoque de tipo: "¿Qué lenguaje es mejor?" no es acertado. La razón es clara, por una causa principal: una pregunta así no es fácilmente resoluble sin un mínimo contexto, es decir, para qué se va a usar el lenguaje y quién lo va a usar.

Criterios cuantitativos

Por considerarlos de menor importancia y únicamente como indicadores de la situación general en que se hallan los lenguajes en cuanto a su uso, utilizaremos distintos rankings de popularidad de lenguajes (ver Tabla 2) para comprobar su adopción y utilización actual (actualizados al primer cuatrimestre de 2014 o, en el caso de RedMonk, a Marzo de 2014):

Tabla 2. Rankings de popularidad de lenguajes (número menos es mejor)

Lenguaje	Índice TIOBE [13]	Language Popularity Index [14]	Popularity of Programming Language [15]	RedMonk Programming Language Rankings [16]
Java	2	2	1	2
Ruby	14	10	10	7
JavaScript	9	12	7	1
PHP	6	6	2	3
Python	8	7	3	5

A la vista de los resultados y sin tener más información acerca de cómo son calculados los puestos, Java es claramente el lenguaje más popular. Ahora bien, es posible obtener más información de estos resultados si estudiamos cómo se calculan estos indicadores:

- El índice TIOBE se crea en base al número de resultados obtenidos en una búsqueda web realizada con diversas frases que contengan el nombre del lenguaje de programación en ellas.

- El Language Popularity Index se basa en un cálculo similar al índice TIOBE, con la salvedad de que desgrana los resultados en tuplas {buscador, lenguaje de programación} y permite descargar la herramienta de código abierto utilizada para verificar los resultados obtenidos.

- El PYPL se basa en el número de búsquedas realizadas relacionadas con los lenguajes (en contraste a los anteriores, que se basaban en el número de resultados).

- El índice RedMonk se basa en la correlación entre la aparición de programas nuevos basados en los lenguajes estudiados (en GitHub) y las discusiones acerca de éstos generadas (en Stack Overflow).

Este conocimiento nos permite extraer cierta información adicional aparte de los números en sí. Principalmente, el índice RedMonk nos permite saber sobre qué lenguajes se está trabajando realmente, ya que el TIOBE y el LPI incluyen toda la documentación existente y búsquedas académicas (este último rasgo compartido con el PYPL). En contraste, el índice RedMonk toma dos variables que, especialmente el número de programas en GitHub, indican fielmente lo que se está trabajando actualmente en esos lenguajes.

Con este único conocimiento, en caso de tener que decidir un lenguaje en el que programar nuestro servidor web, existen dos opciones claras:

- Java: gran documentación disponible, comunidad y búsquedas muy activas y muchos programas nuevos escritos en él.

- En caso de tener como criterio básico el usar el lenguaje "de moda", la balanza se inclinaría hacia JavaScript, como podemos apreciar en el índice RedMonk, a pesar de disponer de mucha menos documentación y búsquedas que Java, se está trabajando más en él actualmente que en ningún otro lenguaje. Este contraste es comprensible si se analizan ambos lenguajes: Java es un lenguaje de propósito general, utilizado constantemente para todo. JavaScript hasta muy recientemente era un lenguaje de segunda, relegado a la parte de cliente en programación web y nada más. Con el desarrollo de Node.js (lo que permite programar el cliente y el servidor de un servicio web en un único lenguaje), librerías como D3, AngularJS, Bootstrap y Express (que han aumentado sus posibilidades increíblemente) y el impulso que están dándole empresas como

Microsoft o Mozilla, el uso de JavaScript ha aumentado exponencialmente en los últimos años.

2.2 Frameworks

Utilizando la descripción de [17], podemos entender un framework como "una estructura de soporte definida sobre la que otras aplicaciones software pueden ser organizadas y desarrolladas." Esta definición puede incluir programas soporte, librerías y scripts de código y cualquier tipo de servicio que permita unir los diferentes componentes de una aplicación. En [artículo de antes] también se realiza la distinción entre componentes "congelados" y componentes "calientes". Los primeros son aquellos que componen la arquitectura básica del sistema software y los segundos son aquellas partes del sistema que están diseñadas específicamente para sistemas software individuales, mediante la adaptación de componentes genéricos. Centrándonos en frameworks de aplicaciones web, Shan y Hua los definen como "plataformas modulares, reutilizables y esqueléticas semi-completas que pueden ser especializadas para producir aplicaciones web personalizadas(...)." Generalmente, estas plataformas implementan el paradigma Modelo-Vista-Controlador (MVC) e integran servicios como búsquedas y control de versiones. Entre las razones ofrecidas para el uso de frameworks, destacan el hecho de que se basan en arquitecturas abiertas estandarizadas y la inclusión en éstos de los servicios básicos de cualquier aplicación web, como una capa de persistencia para el acceso a una base de datos, gestión de usuarios, gestión de grupos y autorizaciones de acceso.

2.2.1 Frameworks de Java EE

Dijimos previamente que en [17] daban dos indicaciones generales a la hora de elegir un framework de aplicaciones Java. La primera era aplicable a todos los lenguajes de programación, pero la segunda se centra en Java y, por tanto, la detallamos aquí. Esta recomendación es: partir desde Spring como plataforma por defecto y añadirle un controlador de front end apropiado. Si el artículo fuera más actual tomaríamos esta orientación más seriamente, ahora bien, dado que el artículo tiene 8 años, incluiremos Spring como uno de los frameworks descritos pero hablaremos también de un servidor de aplicaciones y otros dos frameworks: GlassFish, Apache Tapestry y Spark.

GlassFish

A pesar de no ser un framework sino un contenedor de aplicaciones, debido a su importancia consideramos reseñable hablar de esta tecnología, puesto que se trata de la implementación de referencia de Java EE. Por tanto, tiene implementaciones para todas las tecnologías definidas en esta especificación. Fue concebido por Sun Microsystems y ahora es promovido por la Oracle Corporation. Se trata de un servidor de aplicaciones

de código abierto construido sobre un kernel basado en OSGi que se ejecuta directamente sobre Apache Felix. Entre el resto de tecnologías que utiliza podemos remarcar un derivado de Apache Tomcat como contenedor servlet. Y de las funciones que provee a la aplicación destacan las siguientes:

- Un núcleo ligero y extensible basado en estándares de la OSGi Alliance.
- Un contenedor web.
- Una consola de administración para la configuración y gestión.
- Una herramienta de actualización para componentes software.
- Soporte para clustering y balanceo de carga.

Spring

Spring, liberado por primera vez en 2004, es un framework multicapa de aplicaciones de Java EE, que incluye: un contenedor ligero para la configuración y cableado automáticos de objetos de aplicación mediante inyección de dependencias, una capa de abstracción para la gestión de transacciones, una capa de abstracción JDBC (Java DataBase Connectivity), funcionalidad de programación orientada a aspectos e integración con mapeadores objeto-relacional, configuración mediante JMX, una mejora de la API de envío de mensajes JMS, soporte para tests unitarios, su propio framework MVC y soporte para la construcción rápida de aplicaciones mediante la solución "convención sobre configuración", entre otros módulos. Sus componentes básicos pueden ser utilizados para cualquier aplicación Java pero existen módulos de expansión que permiten construir aplicaciones web sobre Java EE utilizándolo. La versión más actual es la 4.0.3, liberada el 27 de Marzo de 2014.

Apache Tapestry

Tapestry es un framework de aplicaciones de Java creado como un proyecto de la Fundación Apache, similar a JavaServer Faces. Sus características principales son:

- Adhesión a cuatro principios:
    - Estructura estática y comportamiento dinámico.
    - API adaptativa.
    - Diferenciación entre API interna y pública.
    - Asegurar retrocompatibilidad (desde la versión 5, en 2008).

- Prioridad a la simplicidad, sencillez de uso y productividad del programador.
- Convención sobre configuración, esto es, elimina prácticamente toda la configuración en XML.
- Aproximación modular al desarrollo web, mediante el uso de asociaciones entre cada componente de la interfaz de usuario y su clase Java correspondiente.

Spark

Siguiendo el ejemplo de Sinatra, un nuevo tipo de frameworks ha ganado popularidad en los últimos años: frameworks ligeros, sencillos, orientados al despliegue de aplicaciones lo más rápidamente posible. El primer framework de este tipo que vamos a ver es Spark. Este framework no necesita Java EE, únicamente Java y provee una API limitada pero muy focalizada basada en tres pilares:

- Objetos Request y Response: permiten controlar las cabeceras HTTP y sus funcionalidades. Equivalente a la API de Servlets.
- Routes: los puntos de unión entre la aplicación y el protocolo HTTP. Permite establecer funciones de callback a realizar para cada método HTTP específico.
- Filters: pequeños métodos que pueden ser insertados antes o después de la ejecución de una petición y permiten ser activados únicamente en ciertas rutas.

Toda la configuración de la aplicación se realiza mediante código Java, no existen configuraciones XML o anotaciones

2.2.2 Frameworks de Ruby

Dentro de Ruby vamos a presentar los dos frameworks más populares y, además, contrapuestos entre ellos: uno es un "monstruo" monolítico que cubre todos los aspectos necesarios y el otro es un framework ligero destinado a pequeños productos.

Ruby on Rails

Creado por David Heinemeier Hansson en 2004, Ruby on Rails es un framework open source full stack: permite crear aplicaciones, que éstas tomen datos del servidor, interaccionar con una base de datos y mostrar plantillas. Además, promueve el uso de patrones de diseño conocidos, como *Convention Over configuration, Don't Repeat Yourself.*

Toda la organización se realiza mediante la convención Modelo-Vista-Controlador, donde:

- Un modelo es una tabla en la base de datos y un fichero Ruby.
- Un controlador es un componente de Rails que responde a peticiones externas del servidor web a la aplicación determinando qué archivo de vista debe mostrar.
- Una vista es un archivo erb, que se compila en un HTML en tiempo de ejecución.

Además, Rails incluye servicios para facilitar la tarea de desarrollo como scaffolding ("andamiaje"), que permite construir una web básica automáticamente, o un servidor web escrito en Ruby.

Sinatra

Sinatra es un framework totalmente antagónico a Ruby On Rails. No sigue el paradigma Modelo-Vista-Controlador, depende de la interfaz del servidor Rack y está pensado para desplegar aplicaciones web en Ruby con el mínimo esfuerzo.

2.2.3 Frameworks de JavaScript

La situación respecto a JavaScript en el servidor es "curiosa", estamos hablando de un lenguaje pensado originalmente como lenguaje ligero de scripting para la parte de cliente web que se ha convertido en el "lenguaje para todo" en los últimos años y, gracias a Node.js, una de las tecnologías de servidor más utilizadas. Es pertinente, por tanto, dejar claro que, actualmente, Node.js es el estándar de servidor en JavaScript. Es por esto que primero lo describiremos en sí y después detallaremos algún framework sobre él.

Node.js

Node.js inició una revolución en el mundo del desarrollo web: provocó que JavaScript pasará de estar confinado al lado de cliente, a permitir desplegar un servidor web completo basado en este lenguaje. En concreto, Node.js es una "plataforma construida sobre el motor de JavaScript de Chrome para construir fácilmente aplicaciones rápidas y escalables. Node.js utiliza un modelo E/S no bloqueante basado en eventos que lo hace ligero y eficiente, perfecto para aplicaciones de tiempo real y uso intensivo de datos que se ejecutan en sistemas distribuidos" [18]. Esto es, node.js es una librería de JavaScript para el desarrollo de servidores web. Sobre ésta, se han desarrollado numerosos frameworks web. En este trabajo, veremos dos ejemplos estándar de distintos paradigmas: el framework ligero y sencillo similar a Sinatra y el framework full stack, completo y lleno de características.

Express

Express es un framework accesible y ligero que permite construir aplicaciones de cualquier tamaño. Entre sus características se incluyen la creación de rutas (concepto explicado en Spark), parseo del cuerpo de las peticiones, parseo de peticiones URL, manejo de cookies, autenticación básica HTTP y renderizado de plantillas. Además, soporta diversos lenguajes de simplificación como Sass, Less (CSS) o Jade (HTML).

Meteor

Meteor es un framework full stack que utiliza Node.js como back-end pero, en la práctica, trabaja como una plataforma independiente. Una aplicación Meteor es una mezcla de JavaScript corriendo en el cliente, JavaScript en el servidor dentro de un contenedor Node.js y todo el HTML, CSS y resto de recursos que lo soportan. Está basado en siete principios:

- No enviar HTML a través de la red, únicamente datos y permitir al cliente decidir cómo renderizarlos.
- Escribir las partes de cliente y servidor en un único lenguaje: JavaScript.
- Utilizar la misma API transparente para acceder a la base de datos desde el cliente y el servidor.
- Compensación de latencia: usar precargado y simulación de modelos para hacer que la latencia de conexión a la base de datos sea nula.
- Reactividad full-stack: todas las capas, desde la base de datos a la plantilla, deben tener una interfaz dirigida por eventos disponible.
- Adoptar el ecosistema: Meteor es código abierto y anima a adoptar, no reemplazar, las herramientas ya disponibles.
- Simplicidad: la mejor manera de hacer que algo parezca simple es haciéndolo simple. En Meteor esto se consigue mediante una API limpia.

Meteor está formado por dos partes:

- Una librería de paquetes y módulos necesarios en la aplicación, como webapp y templating, que permiten manejar conexiones HTTP entrantes y crear plantillas que se actualizan ante cualquier cambio de los datos.
- Una herramienta de línea de comandos llamada meteor, que permite empaquetar la aplicación, realizando todas las tareas necesarias, como

compilar lenguajes como CofeeScript, minimizar CSS, descargar paquetes necesarios, etc. [19]

2.2.4 Frameworks de PHP

Symfony

El primer framework que vamos a estudiar de PHP, Symfony, es definido como "una librería PHP que realiza dos tareas diferentes:

- Proporciona una selección de componentes Symfony2 y algunas librerías de terceros.
- Define una configuración adecuada e incluye una capa que integra todas las diferentes partes (componentes, librerías, etc.)

El objetivo de la plataforma es integrar muchas herramientas independientes con el fin de proporcionar una experiencia coherente al desarrollador [20]. Estos componentes que nombra son un conjunto de librerías, de las cuales, las más importantes son:

- HttpFoundation: clases para manejar sesiones y cargar archivos.
- Routing: sistema de enrutado para asignar una URI específica a cierta información acerca de cómo se debe manejar una petición.
- Templating: librería para utilizar plantillas

Laravel

Construida utilizando los componentes de Symfony, Laravel es la plataforma PHP más popular en el 2013 y está basada en las siguientes claves de diseño:

- Enrutado RESTful.
- Utiliza Eloquent ORM (*Object Relational Mapping*) para imponer restricciones sobre relaciones entre objetos de la base de datos.
- Permite utilizar las plantillas nativas de PHP o el motor ligero Blade.
- Apropiado tanto para la creación de pequeñas aplicaciones o de sistemas de producción empresariales.
- Permite la ejecución de tests unitarios de manera sencilla.

CodeIgniter

El último framework que veremos sobre PHP es el más ligero de los tres, siguiendo con la convención de añadir un framework rápido para cada lenguaje. No necesita

prácticamente configuración y tiende a la simpleza y flexibilidad de código frente a librerías monolíticas.

2.2.5 Frameworks de Python

El uso de Python tiene ciertas ventajas, que se pueden resumir en:

- El propio lenguaje: Python tiene fama de ser más rápido, menos costoso y más divertido que otros lenguajes.
- La diversidad: existe una gran cantidad de frameworks disponibles.
- Desarrollo rápido: esta diversidad se amplía cada día debido al desarrollo de nuevos frameworks y a la corrección de bugs y adición de nuevas características a los ya existentes.
- Python es código abierto, un punto positivo siempre.

Django

Este framework fue creado especialmente para un portal de noticias, desarrollado con la rapidez de desarrollo en mente. El framework más "pesado" de los analizados. Posee una gran comunidad muy activa. Su principal punto fuerte es la cantidad de herramientas que lleva incluidas por defecto, entre las que destaca una interfaz de administración. Además de ello, tiene su propio lenguaje de plantillas y posee una gran documentación.

Web2py

Otro framework full stack para Python. Las principales razones para su uso son:

- Retrocompatibilidad garantizada hasta la versión 1.0.
- Sin necesidad de instalación ni configuración: el entorno de desarrollo está basado en web y se accede a él a través del navegador.
- Incluye un servidor web, una base de datos relacional, un entorno de desarrollo integrado, una capa de abstracción para tratar con SQL en tiempo real, múltiples sistemas de autenticación, un sistema de logging de errores, múltiples sistemas de cacheo para estabilidad y jQuery para AJAX y efectos.
- Previene gran cantidad de fallos de seguridad, como Cross Site Scription o Code Injection.
- Compatible con gran cantidad de bases de datos, como SQLite, PostgreSQL, MySQL, MSSQL, FireBird, Oracle, etc.

- Gran cantidad de documentación.

Bottle

Por último, hablaremos de Bottle, un framework que sigue la tendencia que hemos comentado previamente, un micro framework ligero, rápido y que sigue la filosofía *What You See Is What You Get* (WYSIWYG). Bottle se distribuye como un único fichero y no tiene m□a dependencias que la librería estándar de Python. Dispone de las siguientes funciones:

- Enrutado: permite mapear funciones a URLs específicas.
- Plantillas: motor propio y soporte para otros, como Mako, Jinja2 o Cheetah.
- Utilidades: acceso a datos de formularios, subida de ficheros, cookies, cabeceras y otros metadatos relaciones con HTTP.
- Servidor HTTP de desarrollo interno y soporte para otros, como fapws3, bjoern o Google App Engine.

2.3 Motores de plantillas

Una vez decidido el lenguaje y framework sobre el que desarrollar el sistema web, el siguiente paso a decidir es, siguiendo el patrón MVC (Modelo-Vista-Controlador), el motor de plantillas que utilizaremos. Un motor de plantillas se define como una herramienta que permite procesar y generar código HTML dinámico a partir de variables o expresiones (condicionales, bucles, etc.). La utilidad de este tipo de herramientas reside en que separan la lógica de la presentación, lo que permite modificar cada pieza de manera independiente. En general, un motor de plantillas permite controlar la presentación sin tener que modificar a mano el código HTML, además de poder replicar el mismo diseño en todo el servicio de manera sencilla.

Análogo al patrón que hemos seguido en la sección anterior, esta sección está dividida en función del lenguaje de programación que se quiera utilizar, para cada uno de ellos detallaremos los ejemplos más significativos de los motores de plantillas disponibles. Como nota aclaratoria, es importante resaltar que no siempre es necesario utilizar el mismo lenguaje de programación en el framework y en el motor de vistas. En general, nos enfrentaremos a dos casos posibles, en función de las características del framework:

- El framework incluye su propio motor de plantillas: en ese caso, si deseamos utilizarlo, por razones obvias sí usaremos el mismo lenguaje que el del framework.

- El framework no incluye su propio motor de plantillas o no deseamos utilizarlo: es posible utilizar motores independientes si el framework permite integrar una llamada a éstos para generar el HTML dinámico.

2.3.1 Motores de Java

JSP

JSP (*JavaServer Pages*) [21] es una tecnología incluida desde la plataforma Java EE 5 que permite desarrollar contenido web dinámico de manera rápida y sencilla. Es la evolución de los Servlet, que genera todo el contenido HTML desde código escrito en Java. Estas páginas JSP mezclan el código HTML con etiquetas particulares que incluyen sintaxis Java. Otra característica destacable es que puede evitarse la repetición de código a través de la directiva include.

StringTemplate

StringTemplate [22] es un motor de plantillas para Java (también C# y Python) de código abierto. Sus plantillas se diferencian del resto de plantillas habituales, donde habitualmente entre corchetes angulares u otros delimitadores aparece la dupla <atributo-expresión>. StringTemplate trata todo como texto y después renderiza llamando únicamente a un método: ST.render().

2.3.2 Motores de Ruby

Haml

Haml (*HTML Abstraction Markup Language*) [23] es un motor de plantillas que permite expresar la estructura de un documento HTML de manera no repetitiva y sencilla. Se basa en el principio "*markup should be beautiful*", proporcionando una sintaxis elegante, fácil de entender haciendo uso de la indentación, eliminando así la dificultad de la apertura y cierre de las etiquetas HTML.

El uso de Haml es bastante simple, las etiquetas HTML se sustituyen por el símbolo de porcentaje seguido del nombre de la etiqueta (%body, %div). Tras esto, se pone el símbolo de igualdad para indicar a Haml que evalúe el código Ruby que aparecerá a la derecha de tal símbolo. Los atributos de las etiquetas HTML se escribirán de la siguiente manera: {:nombreAtributo => "valorAtributo"}. Si la etiqueta posee más de un atributo, estos irán separador por comas. Otro aspecto a destacar de la facilidad de uso de este motor de plantillas es que posibilita escribir los atributos class e id de las etiquetas con una notación similar a CSS.

Slim

Slim [24] es un lenguaje de plantillas que tiene por objetivo reducir la sintaxis de HTML. Inicialmente fue concebido para ver cuánto se podía eliminar de una plantilla

HTML estándar. Su sintaxis es simple, elegante y soportada por la mayoría de los frameworks de Ruby. Soporta un modo 'sin lógica' similar a Mustache a través de un plugin. Slim utiliza Temple para realizar el parseado y la compilación y tamibén Tilt, una interfaz genérica de múltiples motores de plantillas.

Este motor de plantillas que ofrece un alto rendimiento en lo que al renderizado se refiere, superando a otros motores como ERB y Haml [25].

Liquid

Liquid [27] es un lenguaje de marcado que se caracteriza por generar plantillas seguras que no puedan afectar al servidor donde van a ser renderizadas. Tiene sus orígenes en el sistema de comercio electrónico Shopify que permite crear tiendas online de manera sencilla. Tras ser lanzada en 2006 en producción, su uso se ha extendido a todo tipo de aplicaciones web. Liquid, al igual que el Slim, es 'sin estado', lo que permite separar la compilación del renderizado.

2.3.3 Motores de JavaScript

Mustache

Mustache [27] está considerado como la base de los motores de plantillas de JavaScript y, además, está disponible para la mayoría de otros lenguajes de programación mayoritarios. Su funcionamiento se define como "sin lógica" debido a que únicamente realiza cambios a etiquetas, no contiene bucles ni cláusulas condicionales. Una plantilla Mustache únicamente referencia métodos de la vista. Otros motores de JavaScript, como Handlebars, se basan en él.

EJS

EJS (*Embedded JavaScript*) es un motor de plantillas que permite trabajar con código JavaScript incrustado. Este motor es adecuado para generar plantillas en las que no hay código potencialmente peligroso incrustado. EJS permite ejecutar y/o parsear JavaScript tanto en el lado servidor como en el cliente. Además, posibilita el uso de *partials* para incluir y renderizar sub plantillas evitando la repetición de código.

Jade

Jade es un motor ligeramente distinto a los anteriores, diseñado para trabajar en el lado de servidor corriendo bajo Node.js. Este motor es distinto a los dos vistos anteriormente a nivel de sintaxis, puesto que hace un uso intensivo de la indentación. Además, también permite el uso de condicionales, bucles e inserción de código JavaScript. A nivel general, es un lenguaje de programación construido sobre HTML que permite crear páginas web de manera más rápida y sencilla, evitando el uso

extensivo de etiquetas, unido a la capacidad para el uso de variables y scripts JavaScript.

2.3.4 Motores de PHP

Smarty

Sin duda el motor de plantillas más conocido es Smarty. Dispone de un gran comunidad de usuarios y ofrece actualizaciones de forma frecuente (entre 4 y 5 cada año). El código de Smarty es bastante simple y fácil de comprender, además de ofrecer opciones de evaluación (para lenguaje PHP). El rendimiento de las aplicaciones que utilizan este motor de plantillas es bastante satisfactorio [28]. Smarty separa el código HTML del código PHP, de forma que en la parte de HTML utiliza variables y tags propias del motor [29]. De esta forma quedan independizadas la parte de *frontend* y *backend*, lo cual es muy deseable de cara al mantenimiento de las aplicaciones y a la resolución de problemas.

Twig

Este motor es bastante reciente, aunque esto no es un inconveniente para que sea uno de los mejor valorados por los usuarios. Es un producto opensource bajo licencia BSD License, y el código, de los desarrolladores de Symfony, proviene de plantillas de Jinja y Django.

En comparación con otros que disponen de más usuarios y soporte (como Smarty) no envidia nada en cuestión de rendimiento, acercándose bastante a los más rápidos y superando a otros [29][30][31].

2.3.5 Motores de Python

Django

Django, el framework más importante dentro de Python, dispone de su propio motor de plantillas. Está basado en texto, permite definir variables y una lógica básica de control, mediante cláusulas condicionales y bucles y es extensible mediante etiquetas personalizadas a través de código en Python.

Jinja2

Jinja es un motor inspirado en el de Django, pero con capacidades extendidas respecto a éste. Permite la ejecución de código Python arbitrario en un entorno controlado mediante sandboxing, un sistema de debug propio y optimizaciones en su compilador a Python para aumentar su rendimiento.

Mako

La última librería de plantillas de Python que vamos a ver está basada tanto en Djago, como en Jinja 2, como en otros motores de Python. Su API es simple y únicamente involucra el uso de una clase, en tiempo de ejecución se acerca a la velocidad de Jinja2, contiene estructuras de control de flujo basadas en código Python real, permite el filtrado de URL y HTML y soporta desde Python 2.4 hasta Python 3.

2.4 Bases de datos

La necesidad de almacenar información es cada vez más importante, y hoy en día se puede decir que los sistemas de bases de datos son el punto de mayor criticidad en todos lo ámbitos. Son estos sistemas los que almacenan y gestionan todos los datos, de carácter público y personal, bajo las condiciones de replicación, seguridad y privacidad que sean necesarios según la naturaleza de los mismos. A continuación se presentan dos formas de trabajo de estas bases de datos: las basadas por completo en SQL (*Structured Query Language*) y las que no: NoSQL (*Not Only SQL*).

2.4.1 Bases de datos SQL

Se conoce como SQL al lenguaje declarativo que se utiliza para consultar y/o modificar la información almacenada en las bases de datos relacionales. Una base de datos relacional es la que utiliza el modelo relacional para definir las relaciones, condiciones y restricciones de las tablas donde que almacenan la información.

Las bases de datos SQL son bases de datos relacionales que almacenan información de naturaleza heterogénea y que guarda relaciones que pueden ser representadas formalmente según el modelo relacional. Este modelo pretende representar los objetos de la vida real mediante tablas, relaciones y restricciones entre éstas. Las bases de datos relacionales han evolucionado mucho desde sus comienzos, de forma que hoy en día se puede hablar de las bases de datos Objeto-Relacional, las cuales no solo almacenan datos simples, sino que pueden almacenar objetos, documentos estructurados y realizar consultas sobre ellos (como XML's).

Oracle

La solución comercial de Oracle es probablemente la más conocida. Las bases de datos que implementa siguen el modelo de datos relacional, es accesible desde diferentes lenguajes de programación y es soportada por casi todos los sistemas operativos servidores (todas estas características se detallan más adelante en comparación con las del resto de soluciones de base de datos). En cuanto a la arquitectura [32] de las bases de datos Oracle cabe destacar en primer lugar la modularidad o arquitectura en parrilla, que permite utilizar de forma independiente

los distintos módulos según las necesidades que se tengan, de forma que el añadir o prescindir de alguno de éstos no interfiere en la configuración actual. De esta forma, también los costes se pueden ajustar más a las necesidades.

De cara a cómo se explota esta tecnología encontramos una arquitectura cliente-servidor, de forma que los clientes envían las peticiones al servidor y éste interpreta las solicitudes SQL y/o PL/SQL para devolver los resultados. Todas las operaciones de modificación de los datos son tratadas como una transacción, forma que hasta que no se ejecute un *commit* (guardado), no se da por finalizada (es recomendable por si hay que rehacer los cambios, rollbaks). Por supuesto la concurrencia de peticiones es soportada. A esta forma de trabajo también hay que añadir la arquitectura multicapa en tres capas, la cual implementa de forma independiente la autenticación de los clientes por un lado, la conexión a la base de datos por otro y el procesamiento de las peticiones por otro.

Sobre las estructuras que componen estas bases de datos, se tienen las llamadas estructuras físicas y las lógicas. Las primeras son distintos ficheros con una finalidad diferente, y los más destacables son:

- DataFiles: ficheros con los datos de la base de datos
- Control Files: ficheros con parámetros propios de la base de datos
- Redo Logs: guardan todos los cambios que se realizan en la base de datos, útiles para rollbacks y consistencia de datos
- Archive Files: archivado de logs
- Logs de alertas y fallos
- Backup files: ficheros para los backups

En cuanto a la estructura lógica de las bases de datos Oracle, los distintos componentes permiten tener un buen control sobre el espacio en disco utilizado. Estos componentes son:

- Tablespaces: Unidades lógicas de almacenamiento en que se divide la base de datos. Pueden estar en modo online (accesibles) o en modo offline (no accesibles)
- DataBlocks: Cada base de datos es almacenada en bloques, que se corresponden con un conjunto de bytes del espacio físico donde se aloja. Es un parámetro configurable.
- Extends: Conjuntos de DataBlocks

- Segments: Conjutos de Extends. Existen diferentes tipos: Segmentos de datos, de índices, temporales y de *rollback*.

Por otro lado tenemos las distintas estructuras de datos que manejan estas bases de datos. Las principales y básicas con las tablas (como el resto de bases de datos relacionales), no obstante también existen otras como las Vistas (tablas construidas en base a otras tablas a través de consultas), los Clusters (conjuntos de tablas que comparten datos (filas) y son accedidas de forma reiterada), los sinónimos (alias para acceder a tablas, funciones, procedimientos...que deben estar definidos en el diccionario de datos).

Por último mencionar el diccionario de datos que cada base de datos dispone. Se trata de un conjunto de tablas y vistas que contienen información sobre la base de datos, su arquitectura física y lógica, sinónimos, reglas de integridad (*constraints*) de las tablas, etc.

Microsoft SQL

La solución de Microsoft sobre bases de datos es comercial como la de Oracle, y es también relacional. Su arquitectura, por tanto, se basa en tablas donde se almacenan los datos, las cuales guardan relaciones entre ellas. Existen componentes similares a la de Oracle, como son las vistas o los ficheros de logs para la consistencia de datos, aunque éstos son tratados de forma diferente, ya que en Oracle son tres ficheros de tamaño fijo que se van rotando y sobrescribiendo si es necesario, en MSSQL es un único fichero que crece de forma indefinida a menos que se ejecuten sentencias de truncado de este fichero y se comience a sobrescribir). Es una solución en continua evolución que cada vez ofrece mejores opciones.

Por defecto, las consultas de modificación de datos se tratan como transacciones diferentes, de forma que la vuelta atrás en caso de fallo del sistema es menos fiable que en el caso de Oracle, aunque existen métodos para controlarlo (uso de sentencias como GO, o BEGIN TRANSACTION y COMMIT).

Los entornos donde es recomendable el uso de SQL Server son casi los mismos que para DB2 y Oracle: entornos, normalmente empresariales, donde existe un número alto de transacciones a tramitar, un alto grado de concurrencia de aplicaciones, donde la seguridad y consistencia de los datos es crítico y donde se necesite un soporte técnico experto con niveles de servicios (SLA's) que se deben cumplir.

DB2

Esta es la solución que IBM [33],[34] ofrece desde principios de los 80 y que ha evolucionado mucho, en tanto que en sus principios se implementaba en sistemas

propietarios y hoy en día existen versiones disponibles para distintas plataformas (Unix, Linux, Windows).

Los conceptos básicos en cuanto a las bases de datos son los mismos que los anteriores, aunque existen diferencias de implementación (una instancia puede contener más de una base de datos, en Oracle sólo una, por ejemplo).

En cuanto a los entornos donde se utiliza esta solución, destacan aquellos que utilizan SAP, puesto que ambas compañías (SAP e IBM) trabajan muy de cerca y hacen que sus productos sean lo más integrable posible.

MySQL

Solución de base de datos relacional con versión gratuita y de pago [35], que actualmente es propiedad de la compañía de Oracle (anteriormente lo fue de Sun Microsystems).

Dispone de las características propias de las bases de datos relacionales y utiliza el modelo de transacciones ACID comentado en apartados anteriores. Lo que podemos destacar de MySQL son los dos motores de base de datos que ofrece, ambos potentes pero con finalidades diferentes [36].

El primero se llama MyISAM y se trata de un motor que no soporta relaciones de clave foránea (*foreign keys*) y se perfila como un buen motor para aplicaciones cuyo uso sea mayoritariamente de consultas y no de modificaciones de los datos. En segundo lugar está el motor de base de datos llamado InnoDB el cual sí que soporta todas las relaciones entre tablas y ofrece muy buenos resultados con aplicaciones que realizan muchas modificaciones de datos y que requieren una gran concurrencia.

SQLite

SQLite es un sistema de base de datos relacional open source, y está orientado para el tratamiento de información a pequeña o media escala. Entre las ventajas e inconvenientes de esta solución, además de las propias de los sistemas open source, se encuentran la portabilidad (se trata de un fichero donde reside toda la base de datos), la sencillez de uso, transacciones ACID, rapidez de acceso a datos y consistencia de datos incluso ante fallos del sistema [37]. En cuanto a los inconvenientes, encontramos los problemas de seguridad que un fichero que se puede leer con un editor de textos corriente (gedit, notepad…) ofrece. Si no se modifica el código fuente del DBMS (*DataBase Management System*), no existen formas a nivel de DBMS de implementar control de acceso, autenticación, cifrado ni *backup*/restauración de la base de datos. No obstante, en [38] se describen los pasos para implementar todas esta opciones a nivel de DBMS.

En cuanto a los entornos donde es recomendable el uso de SQLite frente a otros, destacan aquellos donde se puede utilizar las bases de datos SQLite (ficheros) como AFF (*Application File Format*), de esta manera no es necesario parsear ficheros del formato que sean, sino que con consultas SQL, sencillas y complejas, se pueden explotar la información contenida en los ficheros de base de datos. Lo único necesario son los motores SQLite, que están disponibles para casi todos los lenguajes de programación (C\C++, Java, JavaScript, Python, Ruby, Visual…). En entornos donde haya un alto nivel de concurrencia de peticiones, alta distribución de las bases de datos...esta solución no es recomendable.

Existen una serie funcionalidades propias de SQL que SQLite no soporta, como son [37]:

Tabla 3. Funcionalidades soportadas por SQLite

Name	Description
RIGHT and FULL OUTER JOIN	LEFT OUTER JOIN is implemented, but not RIGHT OUTER JOIN or FULL OUTER JOIN.
Complete ALTER TABLE support	Only the RENAME TABLE and ADD COLUMN variants of the ALTER TABLE command are supported. Other kinds of ALTER TABLE operations such as DROP COLUMN, ALTER COLUMN, ADD CONSTRAINT, and so forth are omitted.
Complete trigger support	FOR EACH ROW triggers are supported but not FOR EACH STATEMENT triggers.
Writing to VIEWs	VIEWs in SQLite are read-only. You may not execute a DELETE, INSERT, or UPDATE statement on a view. But you can create a trigger that fires on an attempt to DELETE, INSERT, or UPDATE a view and do what you need in the body of the trigger.
GRANT and REVOKE	Since SQLite reads and writes an ordinary disk file, the only access permissions that can be applied are the normal file access permissions of the underlying operating system. The GRANT and REVOKE commands commonly found on client/server RDBMSes are not implemented because they would be meaningless for an embedded database engine.

2.4.2 Bases de datos NoSQL

Las bases de datos tradicionales (relacionales) hoy en día tienen se enfrentan a problemas para los que inicialmente no fueron pensadas, como son la gestión de numerosas solicitudes concurrentes, escalabilidad y alta disponibilidad, reducción de costes operacionales y de gestión, almacenamiento y gestión de grandes cantidades de información (big-data)… Y estos problemas a los que intentan dar solución descubren las debilidades de las bases de datos relacionales, ya que en el intento de solventarlos se penaliza en la velocidad de lectura y escritura, no soportan las grandes cantidades de datos como la que usan los motores de búsqueda u otros grandes sistemas (redes sociales por ejemplo) y presentan dificultades en términos de crecimiento y/o escalabilidad.

En consecuencia a todo lo anterior, surge un nuevo concepto de base de datos, el denominado NoSQL (comúnmente traducido a Not Only SQL, para enfatizar las ventajas que ofrecen frente a SQL). Estas bases de datos plantean soluciones a los problemas enumerados y a un coste mucho menor que las soluciones SQL comerciales (Oracle, MSSQL, DB2...). A continuación se presenta la categorización de soluciones NoSQL atendiendo a su modelo de datos, así como ejemplos de implementación de las mismas [39][40]:

- **Clave-valor (key-value):** Es un modelo muy simple, cada valor se corresponde con una clave, de forma que las consultas son muy rápidas (más que en las relacionales) y soporta un gran número de consultas concurrentes así como el almacenamiento masivo de datos. Ejemplos: Redis, Riak.

- **Orientado a columna (Column-oriented):** Utiliza tablas como modelo de datos, pero no soporta asociación entre ellas. Los datos se almacenan por separado en columnas, y cada columna es índice de la base de datos. El hecho de acceder sólo a las columnas involucradas en las consultas reduce las peticiones de entrada-salida del sistema. Además, en cada consulta cada columna se traduce en un proceso diferente (alto nivel de concurrencia). Ejemplos: Cassandra, Amazon SimpleDB.

- **Orientado a documento (Documento-oriented):** Similar al modelo de clave-valor, pero en este caso los valores son almacenados como documentos estructurados (JSON o XML). Ejemplos: MongoDB, CouchDB.

- **Orientado a grafos (Graph database):** Utiliza grafos como estructuras de datos representados con nodos, propiedades y relaciones entre éstos. Cada elemento tiene un puntero a los elementos adyacentes, por lo que no es necesaria la implementación y el mantenimiento de índices. Ejemplos: Neo4j, Titan.

Estos son los principales modelos de datos implementados en bases de datos NoSQL, aunque existen más categorías (multimodelo, orientada a objetos, xml, grid&cloud y otros). Una lista completa de implementaciones de bases de datos NoSQL agrupada por los diferentes modelos de datos se puede consultar en la web oficial de NoSQL [41].

2.4.3 Comparativas entre bases de datos

A continuación se muestran algunas tablas comparativas que nos ayudarán a entender las recomendaciones de uso de tecnologías en los distintos escenarios propuestos.

Tabla 4. Características de los modelos de datos NoSQL [40]

	Performance	Scalability	Flexibility	Complexity	Functionality
Key-Value	high	high	high	none	variable(none)
Column	high	high	moderate	low	minimal
Document	high	variable(high)	high	high	variable(low)
Graph DB	variable	variable	high	moderate	graph theory
Relational DB	variable	variable	low	moderate	relational algebra

Así mismo, existen empresas que publican, gratuitamente o bajo suscripción, datos comparativos de las distintas bases de datos, de forma que se puede ver la tendencia de uso de estas tecnologías, o cuál es la tecnología líder en el mercado actual…todo esto dependiendo de los factores que tienen en cuenta para realizar estas clasificaciones.

DB-Engines

Este sitio web [42] ofrece comparativa entre casi todas las bases de datos, SQL y NoSQL existentes. Además, ofrece un ranking basado en el número de veces que aparecen en las consultas de los motores de búsquedas (Google y Bing), frecuencia de uso en búsquedas (Google Trends), número de ofertas de trabajo donde se menciona su uso (Indeed, Simply Hired), número de discusiones en foros reconocidos (StackOverFlow y DBA Stack Engine), número de perfiles profesionales donde se mencionan estas tecnologías (Linkedin), número de menciones en redes sociales (Tweets en Twitter). Toda esta información puede consultarse en el sitio web de DB-Engines.

Con estas medidas calculan una media de popularidad de cada sistema, de forma que no indica qué número de sistemas instalados, pero la popularidad puede dar una idea de la tendencia de uso de cada base de datos. La siguiente figura muestra el ranking actual:

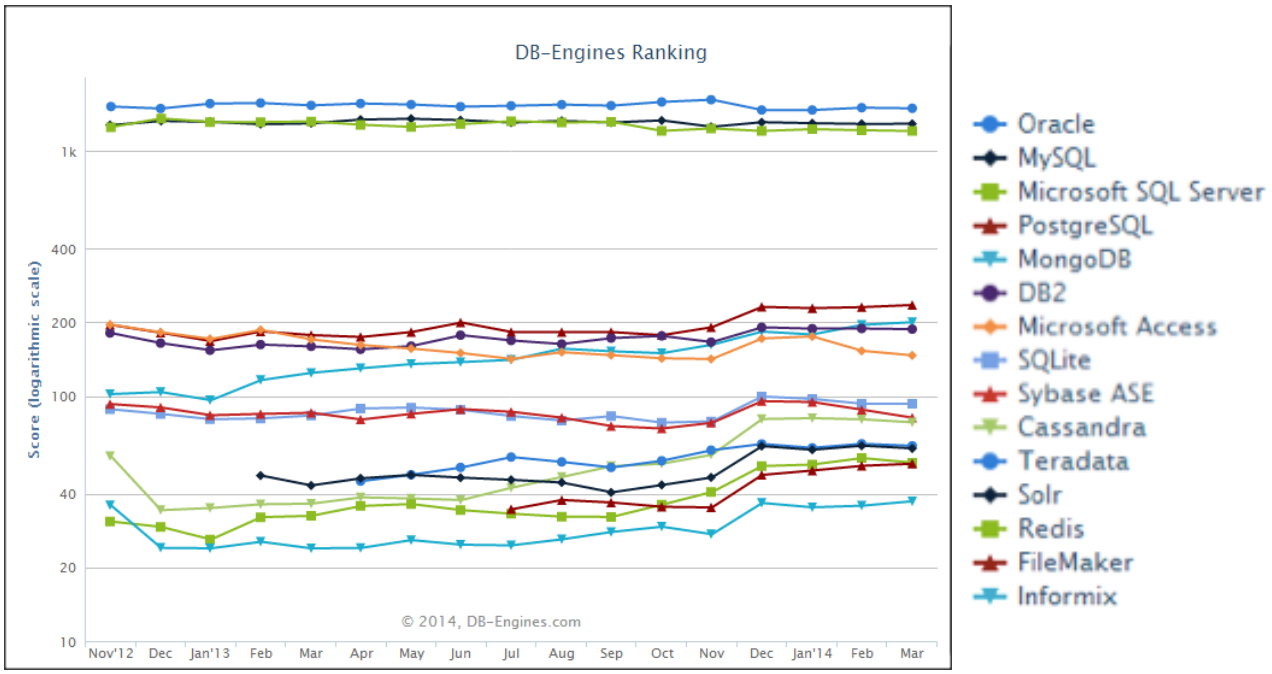


Figura 1. Rankings de bases de datos

Gartner

Reconocida empresa que ofrece comparativas de los líderes de tecnologías en diferentes sectores. En cada caso, tiene en cuenta la situación actual de la tecnología de forma que puntúa cada año las mejoras que deben implementar para poder subir en su ranking (publican siempre todas las características y su relevancia tenidas en cuenta para la evaluación). Aunque es evidente que las compañías más innovadoras de cada tecnología marcan en cierta manera los criterios que se siguen.

Estas comparativas son la referencia de muchas grandes y medianas empresas a la hora de valorar qué tecnología utilizar, o de qué tecnologías valorar para la implantación de una solución final. En cuanto a bases de datos se refiere, el cuadro comparativo de Gartner de 2013 (ver Figura 2) presenta a Oracle como líder del mercado, que es seguido por Microsoft, IBM y SAP.

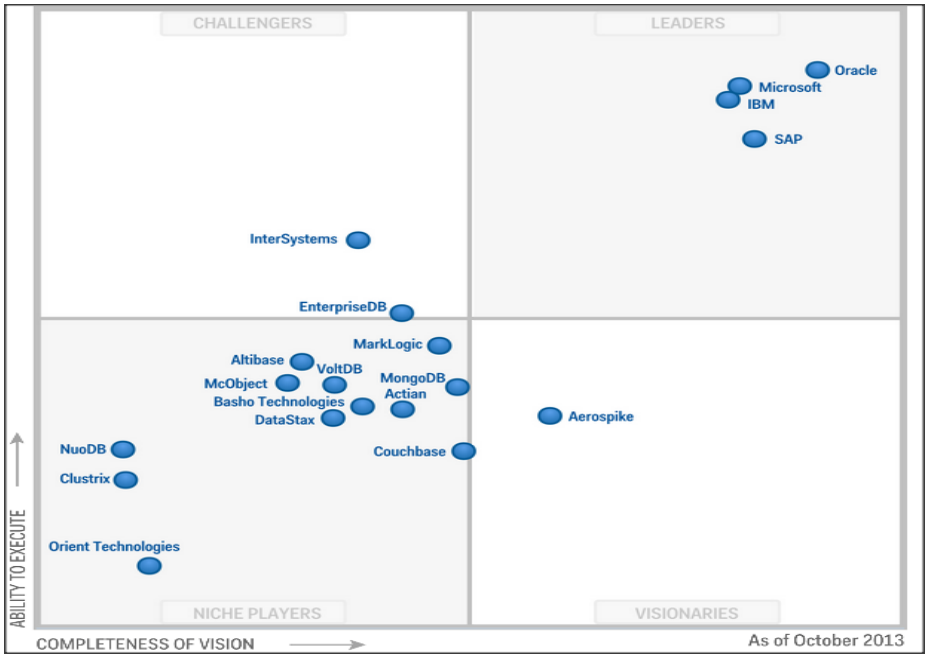


Figura 2. Cuadro comparativo Gartner 2013

2.5 Plataformas de despliegue

Aquí va el estado del arte con el trabajo de la asignatura AGST.

La evolución que ha sufrido en los últimos años las tecnologías de cloud computing ha facilitado el despliegue de aplicaciones en la nube debido a las ya conocidas ventajas que aportan estas tecnologías: reducción de costes, movilidad, flexibilidad, etc.

Existen tres modelos de servicio o capas lógicas de cloud computing [43],[44] dependiendo de la funcionalidad que se necesite (software, plataforma o infraestructura). En cada capa lógica, el usuario y el proveedor del servicio tienen diferentes responsabilidades (ver Figura 3). Estas capas son las siguientes:

- Software as a Service (SaaS): capa superior que permite el acceso a aplicaciones web como correo electrónico, redes sociales o editores de documentos a través de un navegador. Esta capa es la más conocida y utilizada por los usuarios finales de las aplicaciones. Ejemplos: Google Docs, Office 365, Gmail, Dropbox.

- Platform as a Service (PaaS): capa intermedia que proporciona recursos como servidores de aplicaciones, herramientas de desarrollo, bases de datos… que permiten a los desarrolladores generar sus propias aplicaciones en la nube. Ejemplos: Google App Engine, Heroku, Microsoft Azure.

- Infrastructure as a Service (IaaS): capa inferior en la que se basan las dos anteriores que provee el almacenamiento y capacidades de procesamiento. Ejemplos: Amazon Web Services EC2, Joyent, Rackspace Cloud, Eucalyptus.

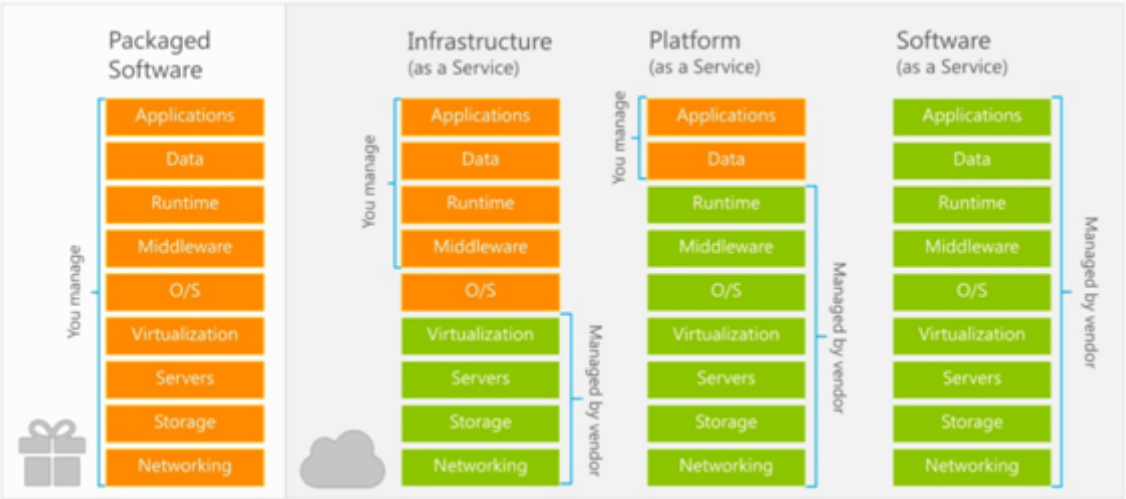


Figura 3. Jumpstart - Windows Azure [45]

Atendiendo a la curva de expectación creada por la consultora Gartner para cloud computing (una gráfica que mide la madurez, el nivel de adopción y la aplicación de una tecnología específica) en el año 2013 (ver Figura 4) , la capa lógica PaaS se encuentra en el Pico de expectativas sobredimensionadas (*Peak of Inflated Expectations*). Esta fase se caracteriza por las grandes expectativas que genera la tecnología analizada en ese momento y por la escasa implementación existente.

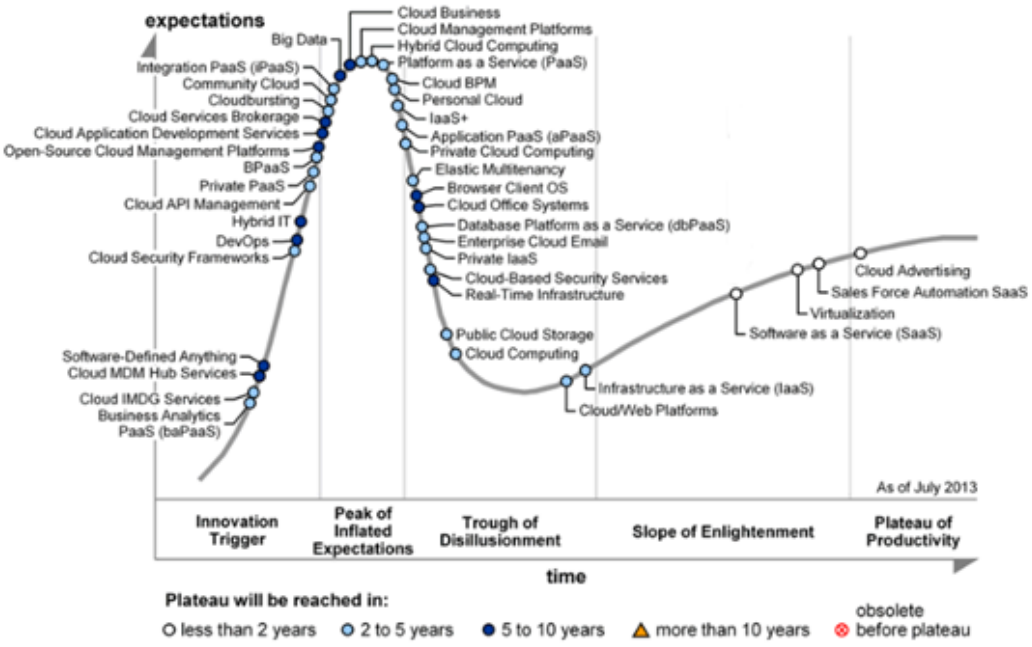


Figura 4. Curva de expectación para cloud computing 2013 [46]

Desde el punto de vista del desarrollador es imprescindible centrarse en el PaaS, donde hoy en día encontramos una gran variedad de soluciones: Amazon Elastic Beanstalk, Aneka, AppFog, CloudBees, Cloudify, DotCloud, Engine Yard, Force.com, Google App Engine, Heroku, IBM SmartCloud Application Services, Jelastic, Microsoft Azure, Nodejitsu, Red Hat OpenShift, VMWare Cloud Foundry…

Ante esta ingente oferta de proveedores existente, surge la necesidad de establecer una comparativa que permita al desarrollador elegir qué plataforma es la más adecuada para publicar su aplicación en la nube. En los siguientes apartados se describen las plataformas más conocidas de PaaS así como una comparativa entre ellas.

Las plataformas han sido clasificadas según su portabilidad [47]. Se definen como PaaS no portables aquellas plataformas en las que la aplicación debe estar escrita de acuerdo a una API específica del proveedor, dificultando así la movilidad de la aplicación entre diferentes plataformas. Este tipo de PaaS proporcionan poca flexibilidad en favor de una mayor robustez y seguridad. Por otro lado, se definen como PaaS portables, las plataformas que permiten ejecutar aplicaciones sin apenas hacer cambios en el código

2.5.1 PaaS No Portables

Google App Engine

Google App Engine [48], también conocido como GAE o App Engine, es la solución PaaS que ofrece Google desde abril del 2011. Permite a los desarrolladores utilizar la infraestructura de Google para el desarrollo y alojamiento de aplicaciones web escalables.

Soporta lenguajes de programación como Java, Python y PHP de manera estable y Go [49] de forma experimental. Este último es el lenguaje de programación de código abierto lanzado por Google, soportado por Linux, Mac OS X y Windows entre otros.

GAE cuenta con un SDK (Software Development Kits), un entorno de desarrollo para los lenguajes de programación soportados. Este SDK se compone de las APIs de App Engine, un entorno denominado sandbox que simula de manera local un entorno de ejecución para las aplicaciones que desarrollemos y un conjunto de herramientas para desplegar las aplicaciones en la nube y gestionar diferentes versiones de esta.

Los principales entornos de desarrollo (frameworks) que facilitan el desarrollo de las aplicaciones web que ofrece Google App Engine son Django y webapp (propio de Google). Ambos frameworks están escritos en Python y permiten implementar el patrón de diseño MVC (Modelo-Vista-Controlador).

Para el almacenamiento persistente de los datos existe la posibilidad de emplear bases de datos relacionales con Google Cloud SQL. GAE utiliza fundamentalmente

Google Cloud Datastore, un servicio de almacenamiento de datos no relacionales en la nube. Este servicio proporciona una base de datos NoSQL basada en la infraestructura BigTable de Google, con la ya conocida ventaja de escalabilidad que supone el uso de este tipo de base de datos.

GAE ofrece de manera gratuita un 1 GB de almacenamiento así como suficiente CPU y ancho de banda para dar soporte a una cantidad de 5 millones de visitas al mes. El resto de detalles sobre cuotas y límites pueden consultarse a través de Google Developers.

Microsoft Azure

Microsoft Azure [50] es la oferta de PaaS de Microsoft desde el año 2010. Inicialmente estaba basada únicamente en el framework .NET. Por ello, ofrece una pila .NET (compuesta por C#, VB.NET, ASP.NET) a los desarrolladores así como otros servicios basados en .NET [51]. La solución de Microsoft está compuesta por varios elementos:

- Fabric Controller se encarga de monitorizar y gestionar los servidores y conectar los distintos recursos para ser dinámicamente asignados. Proporciona auto escalabilidad, seguridad, gestión de los recursos de memoria y balanceo de carga.

- .NET Service Bus registra y conecta las aplicaciones de manera conjunta proporcionando una infraestructura de almacenamiento segura y accesible para las comunicaciones, distribuciones de eventos, servicios de nombre y de publicación.

- .NET Access Control es una infraestructura para gestionar el control de acceso. Utiliza diferentes métodos de autenticación: usuario y contraseña, autorización a través de peticiones HTTP, tokens a través de entidades de confianza, etc.

- .NET Workflow proporciona un entorno para la orquestación de servicios mediante control de flujo basado en reglas, invocaciones de servicios así como procesamiento y correlación de mensajes.

Microsoft ha ampliado la gama de lenguajes de programación y tecnologías que ofrece incluyendo Node.js, Java, PHP, Python y Ruby. Esto ha supuesto para Microsoft Azure una mejora en la portabilidad de las aplicaciones web, empezando a alejarse así de de ser un PaaS no portable.

En cuanto a la persistencia de los datos [52], Microsoft Azure posee una API dedicada para el almacenamiento y recuperación de datos denominada SQL Services,

bajo la cual se encuentra un sistema gestor de base de datos Microsoft SQL Server. También existen otras alternativas en Azure como MySQL o MongoDB.

Los costes sobre el procesamiento, almacenamiento de datos, redes y otros servicios pueden consultarse a través de la página web de Microsoft Azure en la sección de precios. Actualmente ofrece una versión de prueba gratuita durante un mes. También permite ejecutar hasta 10 sitios web por región gratis en un entorno multiempresa.

Force.com

Force.com [47][53] es la plataforma de Salesforce lanzada en 2008 que permite construir aplicaciones extendiendo la funcionalidad de Salesforce, una herramienta de ventas CRM (*Customer Relationship Management*) muy popular. Fue una de las primeras PaaS como conocemos hoy en día.

Esta plataforma proporciona APIs y herramientas para servicios web, una interfaz de usuario para construir aplicaciones, base de datos para la persistencia de los datos y una funcionalidad básica de alojamiento de páginas web.

La principal desventaja de utilizar Force.com es que no se pueden crear aplicaciones utilizando el lenguaje que deseemos. Las aplicaciones se construyen utilizando Apex, un lenguaje de programación orientado a objetos similar a Java. Por otro lado su gran ventaja es la creación de nuevas aplicaciones, una tarea fácil y rápida a través de su interfaz web.

Una de las razones por las que esta plataforma es bastante popular es que el desarrollador no tiene que preocuparse sobre la gestión o escalabilidad de las aplicaciones, solo debe centrarse en la creación y el diseño de las aplicaciones.

Visualforce es el framework de la plataforma Force.com. Incluye componentes AJAX y un lenguaje de marcado basado en etiquetas similar a HTML, en el que cada etiqueta se corresponde con componente de la interfaz de usuario. Implementa el patrón de diseño MVC.

Esta plataforma utiliza la propia base de datos de Salesforce, denominada Database.com. Esta base de datos es objeto relacional.

El precio de desplegar aplicaciones en esta plataforma (consultar en la página web de Salesforce) depende del número de aplicaciones a desplegar, el número de objetos por usuario y el número de llamadas a las APIs.

2.5.2 PaaS Portables

Heroku

Heroku [54], plataforma en desarrollo desde el 2007, es una de las primeras soluciones PaaS existentes. En 2008 fue adquirida por Salesforce, constituyendo junto con Force.com las ofertas de PaaS en la nube de dicho proveedor. El principal punto que diferencia a esta plataforma del resto es su madurez y las posibilidades que presenta con su asombrosa cantidad de add-ons.

Inicialmente fue concebida para el despliegue instantáneo de aplicaciones web sobre Ruby on Rails. El gran éxito que tuvo entre los desarrolladores provocó que proporcionase soporte también para Node.js, Python, Java, Clojure y Scala. A estos lenguajes hay que añadirle diversos frameworks como Django (Python); Tapestry, Spring, Grails, Play (Java); Rails, Sinatra, Ramaze, Camping (Ruby); Node.js, Express (JavaScript).

Un aspecto destacable de este PaaS es que proporciona un despliegue rápido de las aplicaciones a través de la interfaz de línea de comandos. Con un simple comando, 'git push heroku', nuestra aplicación se desplegará en Heroku.

La propuesta de Heroku en el apartado de almacenamiento es una de las más completas que podemos encontrar entre los diferentes PaaS analizados. Por defecto, Heroku ofrece una base de datos PostgreSQL, aunque también podemos elegir entre otras bases de datos relacionales como MySQL y un gran número de soluciones NoSQL como Hadoop, MongoDB, CouchDB, Redis, Memcache, etc. La infraestructura que hay tras esta plataforma es EC2 de Amazon. Los servidores son gestionados por la plataforma y nunca se exponen a los usuarios.

Ofrece de manera gratuita un máximo 5 MB de espacio en disco para el almacenamiento de la base de datos y 50 MB para todos los ficheros de la aplicación, incluyendo los repositorios Git. Los precios establecidos en Heroku están basados en los recursos que se utilicen. Dichos precios están en función de dynos, la unidad de potencia de cálculo en Heroku.

OpenShift

Openshift [55] es la plataforma de alojamiento de aplicaciones en la nube de Red Hat lanzada en junio de 2013. Es una solución PaaS que hay que tener en cuenta por la experiencia de Red Hat en productos de código abierto, aunque la plataforma sea reciente.

Esta plataforma proporciona un amplio rango de lenguajes como Java, PHP, Python, Perl, Ruby y populares frameworks de desarrollo: JBoss (Java); Zend, Laravel, Symfony, CakePHP y Codelgniter (PHP); Django, Bottle, Pylons, Zope y TurboGears

(Python); Rails, Sinatra (Ruby); Node.js. También dispone de consola web, interfaz de línea de comandos, entorno de desarrollo integrado o repositorios de código que facilitarán el desarrollo de las aplicaciones web.

El almacenamiento de los datos puede realizarse con bases de datos relacionales empleando MySQL o PostgreSQL y con bases de datos NoSQL haciendo uso de MongoDB. OpenShift provee de una consola web para la administración de las bases de datos MySQL y MongoDB. Red Hat ofrece tres posibilidades distintas para utilizar la plataforma OpenShift:

- OpenShift Online: un servicio de alojamiento gratuito en una nube pública para desarrolladores de aplicaciones.
- OpenShift Enterprise: una plataforma PaaS diseñada para ejecutarse dentro de los centros de datos de las empresas o en la nube privada.
- OpenShift Origin: la plataforma de alojamiento de aplicaciones de código abierto que se encuentra bajo OpenShift Online y OpenShift Enterprise.

El modelo de facturación es similar al de otras plataformas que se basan en instancias o dynos (en el caso de Heroku). OpenShift se fundamenta en engranajes (gears), contenedores de recursos donde se ejecutan una o más pilas de software definidas por el usuario, denominadas cartuchos (cartridges). Cada engranaje corre en una máquina virtual que tiene una cantidad de RAM y espacio de disco asociado.

OpenShift otorga de manera gratuita tres engranajes, lo que supone 1.5 GB de memoria RAM (512 MB por engranaje) y 3 GB de almacenamiento (1 GB por engranaje). Los precios de los otros dos planes disponibles (bronce y plata) pueden ser consultados a través de la página web de OpenShift en el apartado de precios.

AppFog

AppFog [47][56] es una plataforma que empezó como una startup independiente y fue adquirida en junio de 2013 por CenturyLink. Esta plataforma incorpora y extiende la tecnología de Cloud Foundry [57], el PaaS de código abierto de VMware. Cloud Foundry tiene dos componentes: una librería de código abierto llamada CloudFoundry.org (en lo que se basa AppFog) y una plataforma de alojamiento propietaria llamada CloudFoundry.com que utiliza el código CloudFoundry.org.

Una característica diferenciadora de AppFog es la flexibilidad y la portabilidad que tiene ya que las aplicaciones desarrolladas pueden de ser desplegadas sobre múltiples infraestructuras y sistemas de nube pública (Amazon Web Services, Rackspace, HP Cloud, ect.). Esto permite elegir la infraestructura que sea más adecuada para nuestra aplicación.

AppFog da soporte para múltiples lenguajes de programación como Ruby, Java, PHP, Python, Scala y Erlang. Además, ofrece varios frameworks para los lenguajes anteriormente citados: Rails, Sinatra (Ruby); Grails, Spring (Java); Drupal (PHP); Django, Flask (Python); Node.js, Express (JavaScript). También cuenta con repositorios de código como Git, SVN y Mercurial.

Este PaaS proporciona esencialmente tres opciones para el almacenamiento de datos: MySQL, PostgreSQL y Redis. AppFog permite add-ons de terceros, lo que amplía la oferta de bases de datos, incluyendo MongoDB, CloudDB y Xeround.

El modelo de servicio de esta plataforma es un poco diferente del resto de PaaS examinados, los cuales se basan en instancias de servidor, dynos (Heroku) o gears (OpenShift). AppFog factura principalmente por la utilización de un servidor, donde se dispone de una cierta cantidad de memoria RAM y espacio de almacenamiento en disco, logrando así alojar el número de aplicaciones que se desee. Los planes que AppFog ofrece están disponibles en su página web, en la sección de precios (no tiene ningún plan gratuito).

AWS Elastic Beanstalk

AWS (Amazon Web Services) Elastic Beanstalk [58] es la oferta de Amazon que permite el alojamiento de aplicaciones web utilizando diversos servicios de AWS: Amazon Elastic Cloud Compute (Amazon EC2), Amazon Simple Storage Service (Amazon S3), Amazon Simple Notification Service (Amazon SNS), Elastic Load Balancing y Auto Scaling. Estos servicios proveen escalabilidad, balanceo de carga, alta fiabilidad, flexibilidad y rentabilidad.

La plataforma se caracteriza principalmente por permitir el control total sobre los recursos AWS que subyacen en las aplicaciones, la propia infraestructura. Esta propiedad le diferencia del resto de PaaS, donde el desarrollador apenas es consciente de lo que se encuentra bajo la aplicación.

La propuesta de Amazon soporta Python, Ruby, Java y PHP como lenguajes de programación sobre los que desarrollar las aplicaciones. En contraposición con otras plataformas que tienen un gran número de frameworks para cada lenguaje, este PaaS dispone de los frameworks más utilizados: .NET; Django, Flask (Python); Rails, Sinatra (Ruby); CakePHP, Symfony2 (PHP); Node.js, Express, Geddy (JavaScript).

Elastic Beanstalk aporta un conjunto de herramientas para simplificar el trabajo a los desarrolladores, que incorpora AWS Management Console (interfaz web de usuario), AWS Toolkit para Visual Studio y Eclipse, interfaz de línea de comandos y repositorios Git.

Este PaaS permite elegir entre diferentes opciones de persistencia como MySQL, MongoDB, soluciones de Amazon (Amazon RDS, Amazon DynamoDB, Amazon SimpleDB) o bases de datos ampliamente utilizadas como Microsoft SQL Server, IBM DB2, Oracle, Informix.

Amazon afirma que no existe una facturación adicional por el uso de esta plataforma, únicamente se paga por los recursos que se necesiten para almacenar y ejecutar las aplicaciones. Ofrece una capa de uso gratuito durante 12 meses para implementar una aplicación de poco tráfico.

2.5.3 Comparativa de PaaS

Una vez analizados las soluciones de los principales proveedores de PaaS del mercado para el despliegue de aplicaciones en la nube, el siguiente paso es preguntarse qué plataforma elegir de entre todas ellas. Existen diversos trabajos [43],[51],[52],[59] donde se realizan comparativas entre algunas de estas plataformas, teniendo en cuenta diversas métricas o criterios: coste del servicio, escalabilidad, lenguajes de programación, seguridad, frameworks, herramientas de desarrollo, opciones de persistencia, etc.

Se ha elaborado las tablas 5 y 6, que establece una comparativa entre las plataformas de los apartados anteriores, teniendo en cuenta los criterios más relevantes desde el punto de vista del desarrollador, que son los siguientes:

- Lenguajes de programación: por lo general, los proveedores PaaS soportan múltiples lenguajes de programación, lo que proporciona una mayor comodidad al desarrollador. Los lenguajes más comunes utilizados en las plataformas son Python, Java, Ruby, PHP y JavaScript. El caso de la plataforma Force.com es especial, ya que únicamente soporta su propio lenguaje, Apex. Para realizar la elección del lenguaje de programación (de entre los ofrecidos por la plataforma) se recomienda acudir a la comparativa realizada en apartados anteriores.

- Frameworks: las plataformas disponen de una gran cantidad de frameworks que faciliten el desarrollo de las aplicaciones. Estos están orientados, lógicamente, a los lenguajes de programación soportados por dichas plataformas. A pesar de de haber una gran variedad de frameworks, podemos observar que existe un conjunto de ellos que se repiten en todas las plataformas debido a su popularidad y extendido uso. Es el caso de Django (Python), Spring (Java), Rails o Sinatra (Ruby) y Node.js (JavaScript). Se sugiere tener en cuenta la revisión de frameworks realizada en apartados anteriores.

- Código abierto: indica si la plataforma es de código abierto o no. Con esta característica el desarrollador puede acceder al código fuente subyacente y editarlo al gusto. Solo OpenShift y AppFog poseen esta propiedad.

- Repositorios de código: contiene el software ofrecido por la plataforma donde poder realizar control de versiones y almacenar el código de la aplicación. Casi todos los proveedores PaaS cuenta con esta funcionalidad incluyendo por lo menos la opción de Git.

- Opciones de persistencia: tradicionalmente las aplicaciones web y las aplicaciones empresariales han optado por las bases de datos relacionales como método de persistencia, aunque actualmente las soluciones NoSQL están adquiriendo mayor protagonismo. El catálogo de bases de datos del que disponen las plataformas es amplio (en cierta parte debido a los add-ons), destacando MySQL, PostgreSQL (relacionales) y MongoDB (NoSQL) como principales preferencias en los proveedores PaaS. Se recomienda repasar el apartado dedicado a las base de datos.

- Precio: un aspecto esencial en la elección de la plataforma. Los proveedores PaaS analizados no proporcionan un modelo de facturación común del servicio. Mientras que unos fijan sus precios en función de la utilización de los recursos, otros lo hacen según componentes particulares de la arquitectura de la plataforma, por ejemplo dynos en el caso de Heroku y gears en OpenShift. Por esta razón, no pueden compararse directamente unos precios con otros y se aconseja visitar la página web de cada proveedor. En su lugar, se indican las opciones gratuitas que se proporcionan como alternativa.

- Uso: define el objeto por el cual son utilizadas las plataformas de los proveedores PaaS. En este punto coinciden la gran mayoría de las plataformas, que principalmente se utilizan para aplicaciones web y aplicaciones empresariales.

- Herramientas de desarrollo: recoge el conjunto de facilidades disponibles en la plataforma que contribuyen a simplificar el desarrollo de las aplicaciones. Predominan herramientas como la consola web, la interfaz de línea de comandos (CLI) o entornos de desarrollo integrado (IDE).

Tabla 5. Comparativa entre plataformas (Parte 1)

	Lenguajes de programación	Frameworks	Código abierto	Repositorios de código
Google App Engine	Python, Java, PHP, Go Múltiples lenguajes JVM: Groovy, Scala, JRuby…	Django, webapp, webaap2, GAE Framework, CherryPy, web.py, web2py (Python) Restlet, Vaadin, JSF, Struts 2, Spring, Sinatra, Wicket, Tapestry, Grails, Slim3 (Java)	No	Git
Microsoft Azure	C#, Python, Java, Ruby, PHP, JavaScript	.NET, Spring (Java), Node.js (JavaScript)	No	Git, Team Foundation Server, GitHub
Force.com	Apex	Visualforce	No	-
Heroku	Python, Java, Ruby, PHP, JavaScript, Scala, Clojure	Django (Python) Tapestry, Spring, Grails, Play (Java) Rails, Sinatra, Ramaze, Camping (Ruby) Node.js, Express (JavaScript)	No	Git
OpenShift	Python, Java, Ruby, PHP, JavaScript, Perl	Django , TurboGears, Bottle, Pylons, Zope (Python) JavaEE6, Spring (Java) Rails, Sinatra (Ruby) Zend, Laravel, Codeigniter, CakePHP, Symfony (PHP) Node.js (JavaScript)	Sí	Git, Maven
AppFog	Python, Java, Ruby, PHP, JavaScript, Scala, Erlang	Django, Flask (Python) Grails, Spring (Java) Sinatra, Rails (Ruby) Drupal (PHP) Node.js, Express (JavaScript)	Sí	Git, SVN, Mercurial
AWS Elastic Beanstalk	Python, Java, Ruby, PHP, JavaScript	.NET Django, Flask (Python) Rails, Sinatra (Ruby) CakePHP, Symfony2 (PHP) Node.js, Express, Geddy (JavaScript)	No	Git

Tabla 6. Comparativa entre plataformas (Parte 2)

	Opciones de persistencia	Precio (planes gratuitos)	Herramientas de desarrollo	Uso
Google App Engine	Big Table, Google Cloud SQL	1 GB de almacenamiento	Google App Engine SDK, Eclipse IDE, Codenvy	Aplicaciones web
Microsoft Azure	SQL Server Express, MySQL, MongoDB	1 mes de prueba	Microsoft Visual Studio, Eclipse IDE, CLI	Aplicaciones web, aplicaciones empresariales
Force.com	Force Database	1 mes de prueba	Consola web, Eclipse IDE	Aplicaciones empresariales (especialmente CRM)
Heroku	MySQL, MongoDB, PostgreSQL, Redis, Hadoop, CouchDB, ClearDB, Memcache,	5MB para base de datos y 50 MB para todos los ficheros (incluyendo repositorios GIT)	CLI	Aplicaciones web
OpenShift	MySQL, MongoDB, PostgreSQL	3 gears = 1.5 GB de memoria y 3 GB de almacenamiento	Consola web, Eclipse IDE, CLI	Aplicaciones web
AppFog	MySQL, MongoDB, PostgreSQL, Redis , ClearDB, Xeround	-	Exo IDE	Aplicaciones web
AWS Elastic Beanstalk	MySQL, MongoDB, Amazon RDS, Amazon DynamoDB, Amazon SimpleDB, Microsoft SQL Server, IBM DB2, Oracle, Informix	12 meses acceso a la capa de uso gratuito de AWS	Microsoft Visual Studio, Eclipse IDE, CLI	Aplicaciones web

3 Sistema web complejo para uso didáctico

3.1 Descripción general

Cuadernos de viaje es un sistema web complejo basado en un sistema de información para gestionar datos y fotos sobre los viajes realizados por los usuarios.

El sistema web puede tener muchos usuarios y cada usuario puede tener tantos cuadernos de viaje como viajes haya añadido en este sistema. El viaje contiene la lista de lugares (sitios) que se han visitado, y cada lugar contiene una galería de fotos. La conceptualización de este sistema considera los siguientes aspectos adicionales:

- Al sistema web solo pueden acceder los usuarios registrados.
- El **usuario** registrado puede insertar en el sistema de información múltiples cuadernos de viaje.
- El **cuaderno** se corresponde con cada viaje realizado por el usuario.
- El **sitio** está definido por su nombre, descripción y geoposición (coordenadas: latitud y longitud).
- La **imagen fotografiada** pertenece al sitio y posee desde donde ha sido tomada.
- La ruta de cada viaje, compuesto por múltiples sitios, queda descrita por la secuencia temporal de los sitios visitados.
- El usuario puede realizar **comentarios** sobre las fotos de los sitios.

Este sistema web presenta una arquitectura web de tres capas (ver Figura 5) que permitirá organizar y desarrollar el sistema de información de manera escalable así como separar funcionalidades.

Figura 5. Arquitectura web de tres capas.

La capa de presentación, también conocida como *front-end*, se encarga de mostrar la interfaz gráfica y de interactuar con el usuario. La capa de negocio o *middleware* es la que contiene toda la lógica de la aplicación. Se función es recibir las peticiones realizadas por el usuario y enviarle las respuestas tras procesar dichas peticiones. La capa de datos o *back-end* está formado por las bases de datos que contienen el almacenamiento de los datos de la aplicación, es decir, donde persiste la información.

Tras evaluar las diferentes tecnologías en los diferentes niveles de implementación en la sección Estado del arte, la conclusión más evidente que se obtiene es que hay que diferenciar y especificar meticulosamente el escenario que se desea implementar para poder elegir bien las tecnologías con las que lo vamos a implementar, por ello son tan importantes las fases de especificación de requisitos en los proyectos software.

Además, el abanico de tecnologías existentes es tan grande que el no conocer (o investigar) todas las opciones posibles puede llevar a cometer errores graves en la implementación (tanto como para volver al principio por ser inviable seguir con las tecnologías elegidas) o a realizar una inversión que realmente no era necesaria. Por ello se proponen las siguientes tecnologías para el desarrollo e implementación del sistema web Cuadernos de viaje:

Lenguaje de programación: JavaScript

Lenguaje muy conocido que puede ofrecer el nivel de sencillez (sin perder funcionalidad) que este caso requiere.

Motor y framework: EJS y Node.js

Node.js es el framework de JavaScript por excelencia de los existentes en la actualidad, y en combinación con el motor de plantillas EJS se conseguirían las funcionalidades necesarias para este caso.

Base de datos: MySQL (entorno local de pruebas) y PostgreSQL (entorno de producción)

Son gratuitas, fiables y puede implementarse sobre sistemas operativos gratuitos. Cumple bien con los requerimientos de la aplicación a desarrollar, y es compatible con el resto de tecnologías elegidas.

Despliegue: Heroku

Destaca por la sencillez y rapidez que ofrece al desplegar las aplicaciones (un solo comando desde la interfaz de línea de comandos). Además dispone de una cantidad ingente de add-ons con los que ampliar las funcionalidades de las aplicaciones.

3.2 Modelo de datos

Se ha diseñado un diagrama Entidad-Relación (E/R) con el fin de representar de manera comprensible la información que va a ser almacenada en la base de datos. Este diagrama (ver Figura 6) modela la información que se ha presentado en el apartado anterior.

El diagrama está compuesto por cinco entidades básicas:

- **User:** representa al usuario registrado en el sistema que crea cuadernos de viaje por cada viaje que haya realizado. Un usuario también puede comentar las fotos de los cuadernos de otros usuarios.
- **Notebook:** entidad que refleja un cuaderno de viaje de un usuario. Se compone de múltiples sitios.
- **Site:** lugar visitado en un viaje que pertenece a un cuaderno de viaje. Un sitio puede adjuntar las fotos tomadas en ese lugar.
- **Picture:** imagen fotografiada de un sitio visitado durante un viaje. Una imagen puede tener comentarios realizados por los usuarios.
- **Comment:** entidad que contiene un comentario escrito por un usuario sobre una foto de un determinado sitio.

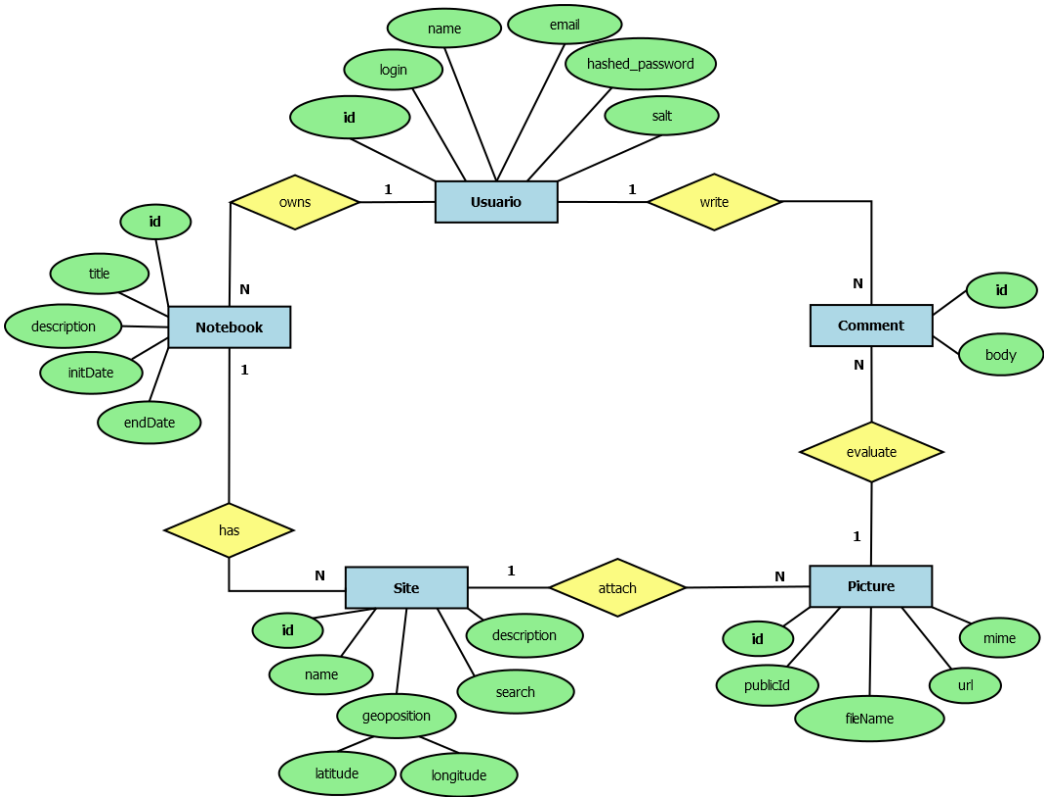


Figura 6. Diagrama E/R

3.3 Instalación de componentes

3.3.1 Node.js, Express.js y otros paquetes

En primer lugar, es necesario realizar la instalación de **Node.js** (http://nodejs.org). Una vez instalado, el resto de paquetes que completan al sistema web se han instalado a través del comando npm (*node package manager*).

Después se instalado **Expressjs** (http://expressjs.com), un paquete que permite generar los ficheros de la aplicación. Dicha instalación se ha realizado de manera global con el siguiente comando:

$ npm install express -g

Utilizando el paquete anterior, se ha creado la aplicación indicando que se utiliza EJS como motor de plantillas (opción --ejs) e incluyendo el soporte para las sesiones (opción --sessions).

$ express --ejs --sessions cuadernosviaje

Tras la ejecución de este comando se crea la siguiente estructura de ficheros y directorios bajo el directorio cuadernosviaje:

- **app.js**: contiene el programa principal a ejecutar. En este fichero se cargan los módulos necesarios para el funcionamiento de la aplicación, se crea el servidor http y se configura la aplicación (vistas, rutas, *middlewares*, etc.).
- **package.json**: contiene información de la aplicación creada (nombre, versión, *script* de inicio). La parte *dependencies* contiene todos los paquetes de los que va a depender la aplicación. Para instalar un paquete nuevo basta con añadir una línea a esta sección indicando el nombre del paquete y la versión, y posteriormente instalar con npm install.
- **public**: este directorio contiene los ficheros estáticos de la aplicación. Estos ficheros son hojas de estilo CSS, *scripts* escritos en javascript e imágenes.
- **routes**: este directorio contiene los módulos que proporcionan los métodos asociados a las diferentes rutas de la aplicación.
- **views**: este directorio contiene los ficheros con las vistas y los *layouts* de la aplicación, es decir, lo que se mostrará al usuario.

Adicionalmente, se ha instalado los paquetes express-partials y sequelize como dependencias en el fichero package.json. El paquete **express-partials** permite integrar el uso de *layouts* y *partials* en el motor de vistas EJS. El paquete **sequelize**

proporciona un ORM para ocultar el acceso a la base de datos y operar únicamente con objetos.

3.3.2 Servidor MySQL

Se ha instalado un servidor de bases de datos MySQL local sobre el que se realizará el desarrollo y las pruebas antes de desplegar la aplicación en la nube. Se puede descargar un servidor MySQL del siguiente enlace:

http://dev.mysql.com/downloads/mysql/

Se ha creado una base de datos denominada cuadernosviaje y un usuario al que se le han otorgado todos los permisos sobre la base de datos creada.

$ mysql -u root -p

mysql> create database cuadernosviaje;

mysql> create user 'cuadernosviaje'@'%' identified by 'cuadernosviaje';

mysql> create user 'cuadernosviaje'@'localhost' identified by 'cuadernosviaje';

mysql> grant all privileges on *.* to 'cuadernosviaje'@'%' identified by 'cuadernosviaje' with grant option;

mysql> grant all privileges on *.* to 'cuadernosviaje'@'localhost' identified by 'cuadernosviaje' with grant option;

Además, es necesario instalar manualmente el paquete mysql en la aplicación para poder realizar la conexión a la base de datos:

$ npm install mysql

3.4 Implementación del *back-end*

La creación de las tablas definidas en el modelo de datos se ha realizado utilizando el mecanismo de migración de sequelize. Este mecanismo permite alterar el estado de la base de datos mediante unos ficheros de migraciones (escritos en JavaScript) donde se especifica cómo llegar a un nuevo estado de la base de datos y cómo revertirlo. En este caso se ha creado un fichero para cada tabla.

Con el siguiente comando se crea el directorio **migrations** que almacena los ficheros de migración que sean creados:

$ ./node_modules/sequelize/bin/sequelize -i

También se crea el fichero **config/config.json** donde hay que especificar los datos de acceso a la base de datos: usuario, contraseña, base de datos, *host*…

Para aplicar las migraciones a la base de datos hay que ejecutar el siguiente comando:

$ ./node_modules/sequelize/bin/sequelize -m

En los siguientes apartados se exponen los esquemas que tienen las diferentes tablas que componen la base de datos (información que aparecerá en su fichero de migración correspondiente). Es necesario resaltar que todas las tablas contienen las columnas createdAt y updatedAt debido a que sequelize añade automáticamente estos atributos a los modelos.

3.4.1 Notebooks

La tabla Notebooks almacena los cuadernos de viaje publicados por los diferentes usuarios. Se compone de las siguientes columnas:

- **id**: clave primaria de la tabla. Es un entero que se autoincrementa de manera automática.
- **userId**: id del usuario al que pertenece el cuaderno. Clave ajena a la tabla Users.
- **title**: cadena de texto con el título del cuaderno de viaje.
- **description**: texto que describe brevemente el cuaderno de viaje.
- **initDate**: fecha de inicio del viaje.
- **endDate**: fecha en la que finalizó el viaje.
- **createdAt**: fecha de creación del cuaderno.

- **updatedAt**: fecha de actualización del cuaderno.

3.4.2 Users

La tabla Users contiene los usuarios registrados en el sistema web. Se compone de las siguientes columnas:

- **id**: clave primaria de la tabla. Es un entero que se autoincrementa de manera automática.
- **login**: cadena de texto con el *login* del usuario. Debe ser único.
- **name:** cadena de texto con el nombre completo del usuario.
- **email:** cadena de texto con el correo electrónico del usuario.
- **hashed_password:** cadena de texto con el *hash* de la contraseña del usuario.
- **salt:** cadena de texto con el *salt* utilizado para crear el *hash* del usuario.
- **createdAt**: fecha de creación del usuario.
- **updatedAt**: fecha de actualización del usuario.

3.4.3 Sites

La tabla Sites contiene los sitios visitados durante el viaje registrado en el correspondiente cuaderno de viaje. Se compone de las siguientes columnas:

- **id**: clave primaria de la tabla. Es un entero que se autoincrementa de manera automática.
- **notebookId**: id del cuaderno al que pertenece el sitio. Clave ajena a la tabla Notebooks.
- **name:** cadena de texto con el nombre del sitio visitado.
- **search:** cadena de texto con la búsqueda de la ubicación en Google Maps.
- **description:** texto que describe brevemente el sitio.
- **latitude:** número (*float*) que contiene la latitud del sitio.
- **longitude:** número (*float*) que contiene la longitud del sitio.
- **createdAt**: fecha de creación del sitio.
- **updatedAt**: fecha de actualización del sitio.

3.4.4 Pictures

La tabla Pictures contiene las imágenes de los sitios visitados en el viaje. Se compone de las siguientes columnas:

- **id**: clave primaria de la tabla. Es un entero que se autoincrementa de manera automática.
- **siteId**: id del sitio al que pertenece la imagen. Clave ajena a la tabla Sites.
- **publicId**: atributo public_id proporcionado por Cloudinary para la imagen.
- **url**: dirección URL de la imagen en Cloudinary.
- **fileName:** cadena de texto con el nombre de la imagen.
- **mime:** cadena de texto que indica el tipo de contenido almacenado.
- **createdAt**: fecha de creación de la imagen.
- **updatedAt**: fecha de actualización de la imagen.

3.4.5 Comments

La tabla Comments contiene los comentarios que los usuarios realizan sobre las imágenes de los sitios. Se compone de las siguientes columnas:

- **id**: clave primaria de la tabla. Es un entero que se autoincrementa de manera automática.
- **authorId**: id del autor al que pertenece el comentario. Clave ajena a la tabla Users.
- **pictureId:** id de la imagen comentada. Clave ajena a la tabla Pictures.
- **body:** texto que contiene el cuerpo del comentario.
- **createdAt**: fecha de creación del comentario.
- **updatedAt**: fecha de actualización del comentario.

3.5 Recursos REST

REST (*REpresentational State Transfer*) es un estilo de arquitectura utilizado para construir aplicaciones y servicios en la web. Se basa en la utilización del protocolo HTTP y en sus métodos para la manipulación de los recursos:

- GET: lee/lista un recurso.
- POST: actualiza un recurso.
- PUT: crea un nuevo recurso.
- DELETE: elimina un recurso.

En los siguientes apartados se definen las rutas REST implementadas en el sistema web y se comentan algunos aspectos sobre los controladores de los recursos, que contienen los métodos utilizados para crear esas rutas.

3.5.1 Notebooks

Las rutas que contienen las acciones new, create, edit, update y delete son accesibles únicamente para el usuario propietario del cuaderno. La acción delete eliminará el cuaderno, los sitios que lo componen y sus respectivas imágenes y comentarios asociados.

Tabla 7. Rutas REST para Notebooks

Método HTTP	URL	Acción
GET	/notebooks	notebooksController.index
GET	/notebooks/new	notebooksController.new
GET	/notebooks/:notebookid	notebooksController.show
POST	/notebooks	notebooksController.create
GET	/notebooks/:notebookid/edit	notebooksController.edit
PUT	/notebooks/:notebookid	notebooksController.update
DELETE	/notebooks/:notebookid	notebooksController.delete

El controlador de este recurso, definido en el fichero **notebook_controller.js**, contiene además otros métodos que no tienen ruta asociada:

- **load:** *middleware* que auto carga un objeto Notebook a partir de su id que se pasará como parámetro de entrada. Si se encuentra el objeto, se añadirá a **req.notebook**. Este método evita la repetición de código porque esta operación se realizará cada vez que se llame a la acción show, edit, update o delete.

- **loggedUserIsAuthor:** comprueba si el usuario que está logueado es el autor del cuaderno. Este método se utiliza para determinar las operaciones que el usuario puede realizar sobre el cuaderno (si no es el autor solo puede ver el cuaderno).

3.5.2 Users

Las rutas que contienen las acciones edit, update son accesibles únicamente para el usuario logueado. No es posible eliminar a los usuarios desde ninguna ruta.

Tabla 8. Rutas REST para Users

Método HTTP	URL	Acción
GET	/users	userController.index
GET	/users/new	userController.new
GET	/users/:userid	userController.show
POST	/users	userController.create
GET	/users/:userid/edit	userController.edit
PUT	/users/:userid	userController.update

El fichero **user_controller.js** (controlador de este recurso) contiene además los siguientes métodos:

- **load**: auto carga un objeto User a partir de su id.

- **loggedUserIsUser**: comprueba que el usuario logueado es al que se refiere en la ruta a la que se está intentando acceder.

- **createNewSalt**: genera una cadena de texto aleatoria que será utilizada en una función *hash* para encriptar la contraseña del usuario.

- **encriptarPassword**: encripta la contraseña del usuario con el *salt* que devuelve la función anterior haciendo uso de una función SHA-1. El

resultado son 20 bytes (40 caracteres hexadecimales) que será lo que se almacene en la base de datos como contraseña.

- **autenticar:** autentica al usuario contra la base de datos comprobando que el login y la contraseña proporcionados son los adecuados.

3.5.3 Sites

Las rutas que contienen las acciones new, create, edit, update y delete son accesibles únicamente para el usuario propietario del sitio (el mismo que es propietario del cuaderno). La acción delete eliminará de la base de datos el sitio que corresponda y todas sus imágenes y comentarios realizados sobre estas.

Tabla 9. Rutas REST para Sites

Método HTTP	URL	Acción
GET	/notebooks/:notebookid/sites	siteController.index
GET	/notebooks/:notebookid/sites /new	siteController.new
GET	/notebooks/:notebookid/:notebookid/sites /:siteid	siteController.show
POST	/notebooks/:notebookid/sites	siteController.create
GET	/notebooks/:notebookid/sites/:siteid/edit	siteController.edit
PUT	/notebooks/:notebookid/sites/:siteid	siteController.update
DELETE	/notebooks/:notebookid/sites/:siteid	siteController.delete

El controlador de este recurso (**site_controller.js**) posee algunos métodos adicionales:

- **load**: auto carga un objeto Site a partir de su id.
- **loggedUserIsAuthor:** comprueba si el usuario que está logueado es el autor del sitio. Este método se utiliza para determinar las operaciones que el usuario puede realizar sobre el sitio (si no es el autor solo puede ver el sitio).

3.5.4 Pictures

Las rutas que contienen las acciones new, create, edit, son accesibles únicamente para el usuario propietario de la imagen (el mismo que es propietario del sitio y el cuaderno). Este recurso no es editable o actualizable. La acción delete eliminará la imagen y todos los comentarios que tenga.

Tabla 10. Rutas REST para Pictures

Método HTTP	URL	Acción
GET	/notebooks/:notebookid/sites/:siteid/pictures	picController.index
GET	/notebooks/:notebookid/sites/:siteid/pictures/new	picController.new
GET	/notebooks/:notebookid/:notebookid/sites /:siteid/pictures/:pictureid	picController.show
POST	/notebooks/:notebookid/sites/:siteid/pictures	picController.create
DELETE	/notebooks/:notebookid/:notebookid/sites /:siteid/pictures/:pictureid	picController.delete

El controlador **picture_controller.js** contiene además los siguientes métodos:

- **load**: auto carga un objeto Picture a través de su id.
- **loggedUserIsAuthor:** comprueba si el usuario que está logueado es el autor del cuaderno. En caso afirmativo será quien pueda subir fotos a los sitios que componen el cuaderno en cuestión.

3.5.5 Comments

Las rutas que contienen las acciones edit, update y delete son accesibles únicamente para el usuario que ha escrito el comentario.

Tabla 11. Rutas REST para Comments

Método HTTP	URL	Acción
GET	/notebooks/:notebookid/sites/:siteid/pictures /:pictureid/comments	commentController.index
GET	/notebooks/:notebookid/sites/:siteid/pictures /:pictureid/comments/new	commentController.new
GET	/notebooks/:notebookid/:notebookid/sites /:siteid/pictures/:pictureid/comments/:comm entid	commentController.show

POST	/notebooks/:notebookid/sites/:siteid/pictures/:pictureid/comments	commentController.create
GET	/notebooks/:notebookid/:notebookid/sites/:siteid/pictures/:pictureid/comments/:commentid/edit	commentController.edit
PUT	/notebooks/:notebookid/:notebookid/sites/:siteid/pictures/:pictureid/comments/:commentid/edit	commentController.update
DELETE	/notebooks/:notebookid/:notebookid/sites/:siteid/pictures/:pictureid/comments/:commentid/edit	commentController.delete

El controlador de este recurso, definido en el fichero **comment_controller.js**, tiene otros métodos complementarios:

- **load**: auto carga un objeto Comment a través de su id.
- **loggedUserIsAuthor:** comprueba si el usuario que está logueado es el autor del comentario. En caso afirmativo será quien pueda editar o eliminar el correspondiente comentario.

3.6 Modelos

Un modelo en **sequelize** representa un mapeo entre una tabla de la base de datos relacional y un objeto definido por el desarrollador. Esto permite trabajar con objetos, lo cual resulta mucho más sencillo y evita tener que realizar *queries* en SQL para consultar y actualizar la base de datos.

Los modelos de este sistema web están definidos en el fichero **models.js**. Cada modelo se corresponde con una tabla almacenada en la base de datos y sus atributos constituyen las columnas de dichas tablas. Es necesario comentar que en estos modelos no están incluidos los atributos id, createdAt y updatedAt que sí aparecen en las tablas de la base de datos. Esto es así porque sequelize añade automáticamente estos atributos.

Un aspecto importante a tener en cuenta en la definición de los modelos son las asociaciones, es decir, las relaciones entre los múltiples objetos. Estas asociaciones ayudan a acceder con mayor facilidad a los objetos. Sequelize permite definir asociaciones uno a uno, uno a muchos y muchos a muchos.

Se han creado las relaciones 1-N (uno a muchos) existentes en el diagrama Entidad-Relación (descrito en apartados anteriores) mediante las asociaciones de sequelize en el fichero models.js:

```
//Relación 1-N entre User y Notebook
User.hasMany(Notebook, {foreignKey: 'userId'});
Notebook.belongsTo(User, {as: 'Author', foreignKey: 'userId'});

//Relación 1-N entre Notebook y Site
Notebook.hasMany(Site, {foreignKey: 'notebookId'});
Site.belongsTo(Notebook, {as: 'Notebook', foreignKey: 'notebookId'});

//Relación 1-N entre Site y Picture
Site.hasMany(Picture, {foreignKey: 'siteId'});
Picture.belongsTo(Site, {as: 'Site', foreignKey: 'siteId'});

//Relación 1-N entre Picture y Comment
Picture.hasMany(Comment, {foreignKey: 'pictureId'});
Comment.belongsTo(Picture, {as: 'Picture', foreignKey: 'pictureId'});

//Relación 1-N entre User y Comment
User.hasMany(Comment, {foreignKey: 'authorId'});
Comment.belongsTo(User, {as: 'Author', foreignKey: 'authorId'});
```

3.7 Vistas

Este sistema web utiliza EJS como motor de vistas, combinando los datos y plantillas para generar las páginas HTML que formarán la interfaz de usuario.

La plantilla utilizada, definida en el fichero **views/layout.ejs** está compuesta por tres secciones:

- La primera sección se sitúa en la parte superior (ver Figura 7). Muestra el nombre del sistema web (Cuadernos de viaje) a la izquierda. A la derecha se encuentra un botón para hacer login y otro para registrarse si no se está logueado ningún usuario. En caso contrario aparecerá el nombre del usuario seguido de un botón de logout. Además contiene una barra de navegación para volver a la página de inicio, acceder al listado de cuadernos o consultar los usuarios registrados en el sistema.

Figura 7. Parte superior de la plantilla

- La segunda sección contiene la expresión **<%- body %>**. Esto será reemplazado por el resultado de renderizar la vista que corresponda mostrar.

- La tercera sección contiene un pie de página con el texto "Universidad Politécnica de Madrid".

Cada recurso (Notebook, User, Site, Picture y Comment) que compone el sistema web contiene cinco vistas que se renderizan en función de la ruta REST con la que se acceda:

- **index.ejs**: lista todos los elementos de un determinado recurso que el controlador le pasa en un array.

- **show.ejs**: muestra todo el contenido de un elemento concreto de un recurso que el controlador le pasa en una variable con el nombre del recurso.

- **new.ejs**: presenta un formulario para crear un nuevo elemento de ese tipo de recurso.
- **edit.ejs**: contiene un formulario para editar un elemento de un determinado recurso que el controlador le pasa en una variable con el nombre del recurso.
- **_form.ejs**: vista parcial que utilizan new.ejs y edit.ejs con los campos del formulario.

A continuación se exponen algunas características de varias vistas de los recursos del sistema web.

Notebooks

La vista show.ejs incluye la vista index.ejs de los Sites. Al acceder a los detalles de un cuaderno se presentarán todos sus datos y un listado de los sitios que componen ese cuaderno.

Sites

La vista index.ejs contendrá un botón denominado Mostrar ruta si el cuaderno tiene dos o más sitios. Al pulsar el botón se mostrará un mapa de Google Maps con la ubicación de todos los sitios en el orden en el que han sido insertado, constituyendo así la ruta del viaje.

La vista show.ejs incluye la vista index.ejs, que es una galería con las imágenes fotografiadas en ese sitio. Además, se mostrará la ubicación del sitio en un mapa estático de Google Maps. Este mapa también se enseña en las vistas new.ejs y edit.ejs.

Pictures

La vista show.ejs incluye las vistas new.ejs e index.ejs de los Comments. Debajo de la imagen se podrá añadir un nuevo comentario así como ver un listado de los comentarios existentes.

La vista new.ejs contiene una pre visualización de la imagen que se quiere subir al sitio una vez se ha seleccionado.

No dispone de la vista edit.ejs porque no son editables las imágenes, ni de la vista _form.ejs, ya que al haber un único formulario (el perteneciente a la vista new.ejs) no es necesaria.

3.8 Despliegue

Una vez se ha desarrollado todo el código de la aplicación y se han realizado pruebas de su correcto funcionamiento en local, el siguiente paso es desplegar la aplicación en la nube.

Se ha elegido Heroku (https://www.heroku.com) como plataforma de despliegue para la aplicación de entre todas las alternativas examinadas en el apartado Estado del arte. Se han seguido los siguientes pasos:

1. Registrarse en Heroku. La cuenta creada en esta plataforma es totalmente gratuita.

2. Descargar e instalar Heroku Toolbelt (https://toolbelt.heroku.com), una herramienta que proporciona la interfaz de línea de comandos de Heroku, acceso a respositorios git y foreman, un gestor de aplicaciones basadas en Procfile.

3. Loguearse en Heroku desde un terminal. Se deben introducir los credenciales (correo electrónico y contraseña) que utilizamos al crear la cuenta.

   $ heroku login

4. Especificar las versiones de node y npm que son compatibles con la aplicación desarrollada añadiendo el campo **engines** al fichero **package.json** con tal información.

   ```
   "engines": {
     "node": "0.10.x",
     "npm": "1.3.x"
   }
   ```

5. Crear el fichero **Procfile** en el directorio raíz de la aplicación. Este fichero contiene el comando que debe ser ejecutado para lanzar la aplicación:

   web: node app

   En este momento podemos ejecutar la aplicación de manera local con el siguiente comando:

   $ foreman start

6. Almacenar la aplicación en un repositorio local git. Esto es necesario porque para poder subir la aplicación a Heroku esta debe estar en un repositorio git. Ejecutar los siguientes comandos (situado en el directorio raíz de la aplicación):

$ git init
$ git add .
$ git commit -m "versión inicial"

7. Crear la aplicación en Heroku. Con el siguiente comando se genera en la plataforma la aplicación y devuelve la URL a través de la cual podemos acceder a la aplicación (puede renombrarse después) y la URL del repositorio git remoto donde se almacenará el código de la aplicación.

   $ heroku create
   Creating shrouded-anchorage-7961... done, stack is cedar http://shrouded-anchorage-7961.herokuapp.com/ | git@heroku.com:shrouded-anchorage-7961.git
   Git remote heroku added

   Cambiamos el nombre de la aplicación por uno más fácil de recordar:

   $ heroku apps:rename cuadernosviaje
   Renaming shrouded-anchorage-7961 to cuadernosviaje... done
   http://cuadernosviaje.herokuapp.com/ | git@heroku.com:cuadernosviaje.git
   Git remote heroku updated

8. Desplegar el código de la aplicación en el repositorio git que creado en el paso anterior con un simple comando:

   $ git push heroku master

9. Instalar una base de datos para el almacenamiento persistente de los datos. Heroku ofrece una gran variedad de bases de datos, tanto relacionales como NoSQL, a través de sus add-ons. Se ha optado por utilizar PostgreSQL. Lanzar el siguiente comando para instalar el add-on:

   $ heroku addons:add heroku-postgresql:dev

   Añadir el paquete **pg** como dependencia en el fichero **package.json**.

   ```
   "pg": "3.x"
   ```

10. Obtener los datos de acceso a la base de datos:

    $ heroku config
    === cuadernosviaje Config Vars
    DATABASE_URL: postgres://**user**:**password**@**hostname**:**port**/**database**
    HEROKU_POSTGRESQL_CHARCOAL_URL: postgres:// **user**: **password** @ **hostname**:**port**/**database**

11. Crear variables de entorno en Heroku con los datos de acceso a la base de datos para no tener que almacenarlos en los ficheros de configuración.

    **$ heroku config: set **
    > DATABASE_NAME= **database** \
    > DATABASE_USER= **user** \
    > DATABASE_PASSWORD= **password** \
    > DATABASE_DIALECT=postgres \
    > DATABASE_PROTOCOL=postgres \
    > DATABASE_PORT= **port** \
    > DATABASE_HOST= **hostname**

    Donde **user**, **password**, **hostname**, **port** y **database** son los valores que devuelve el comando del paso anterior.

12. Establecer el valor de las variables de entorno de Heroku en el fichero **models.js**.

```
var sequelize = new Sequelize(
        process.env.DATABASE_NAME,
        process.env.DATABASE_USER,
        process.env.DATABASE_PASSWORD,
        { dialect:  process.env.DATABASE_DIALECT,
          protocol: process.env.DATABASE_PROTOCOL,
          port:     process.env.DATABASE_PORT,
          host:     process.env.DATABASE_HOST,
          storage: process.env.DATABASE_STORAGE,
          omitNull: true
        });
```

    Adicionalmente, se ha añadido **DATABASE_STORAGE** como variable de entorno para poder utilizar la base de datos MySQL en el entorno local de pruebas. Esta variable no tendrá ningún valor asignado en Heroku.

13. Preparar las migraciones de sequelize en Heroku para crear las tablas en el servidor PostgreSQL. Sequelize utiliza el fichero config/config.json para este propósito y en él deben constar los datos de acceso a la base de datos. Al tener dos entornos, uno local de pruebas y otro de producción (en Heroku), generaremos un shell script que creará el fichero config.json con las variables de entorno configuradas en apartados anteriores.

    Eliminamos el fichero config.json del repositorio git y añadimos una entrada al fichero **.gitignore** (donde se especifican qué ficheros se ignorarán en el repositorio):

    $ git rm -- cached config/config.json

El contenido del shell script, que nombraremos **mkconfig.sh**, es el siguiente:

```
#! /bin/sh
cat > config.json <<EOF
{
    "username": "$DATABASE_USER",
    "password": "$DATABASE_PASSWORD",
    "database": "$DATABASE_NAME",
    "host":     "$DATABASE_HOST",
    "dialect":  "$DATABASE_DIALECT",
    "port":     "$DATABASE_PORT",
    "protocol": "$DATABASE_PROTOCOL",
    "storage":  "$DATABASE_STORAGE",
    "omitNull": true
}
EOF
```

Validar todos los cambios realizados sobre el repositorio git y subir a Heroku:

$ git commit -a
$ git push heroku master

14. Aplicar las migraciones en Heroku desde su consola:

    $ heroku run bash
    Running `bash` attached to terminal... up, run.1018
    ~ $ cd config
    ~/config $ sh mkconfig.sh
    ~/config $ cd ..
    ~ $ ./node_modules/sequelize/bin/sequelize -m

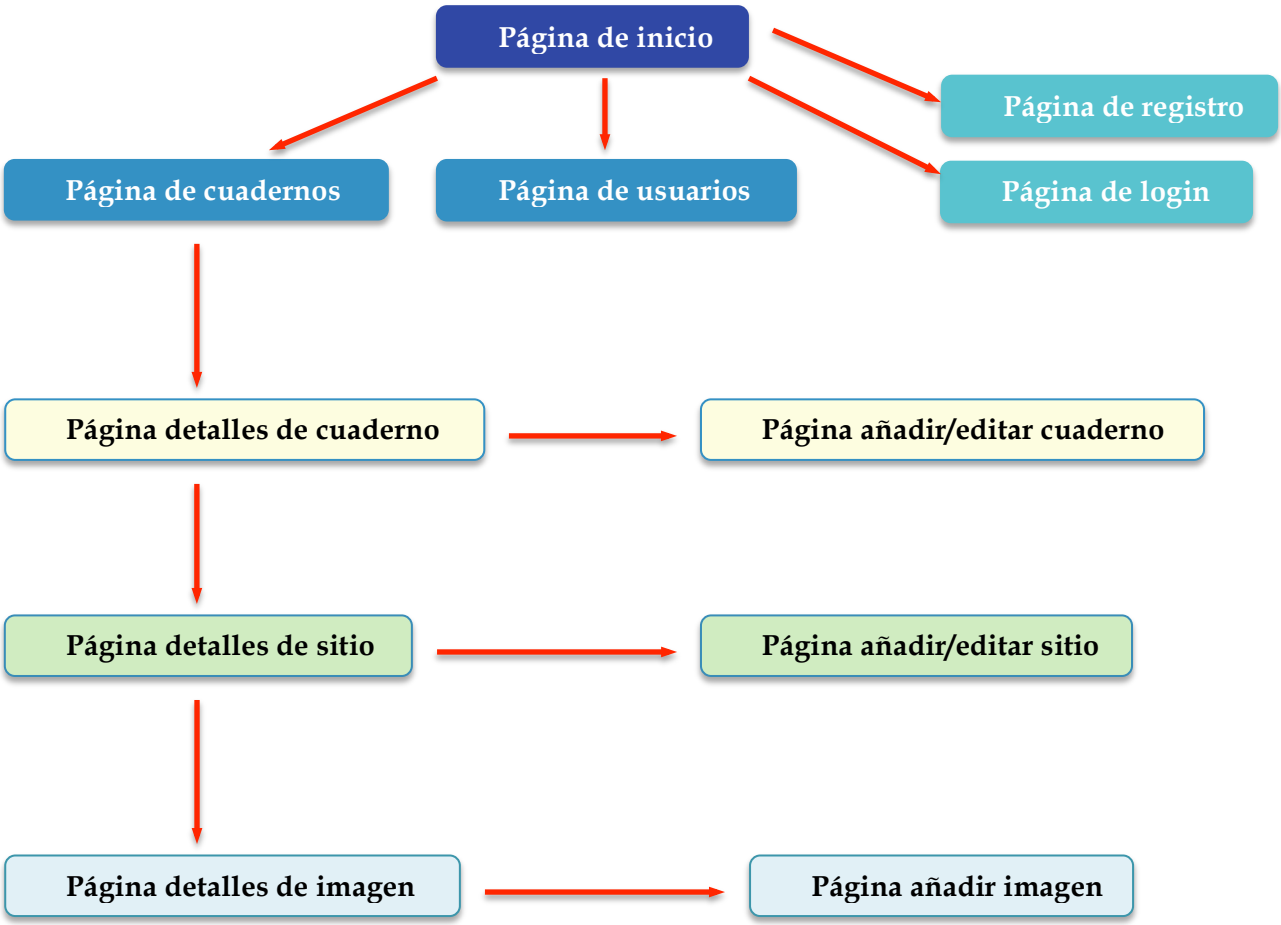
15. La aplicación está desplegada y lista para funcionar en Heroku. Acceder a ella a través de la siguiente URL:

    http://cuadernosviaje.herokuapp.com

4 Resultados

En este punto el sistema web Cuadernos de viaje se encuentra desarrollado, desplegado y completamente operativo en la plataforma Heroku. En este apartado se explica el funcionamiento del sistema web a través de las diferentes pantallas que se muestran en la interfaz de usuario con el fin de ofrecer una mejor comprensión del uso de este sistema.

Antes de comentar el contenido de las páginas, se expone un diagrama conceptual de la navegabilidad entre las diferentes páginas del sistema web. Con esto se pretende ayudar al lector a entender cómo navegar en las páginas.



Página de inicio

Esta página (ver Figura 8) únicamente muestra un mensaje de bienvenida al usuario. La parte superior, como ya se comentó en el apartado Vistas, es común a todas las páginas del sistema, por lo que se puede acceder a las funcionalidades que ahí se proporcionan en todo momento. Estas funciones son hacer login (se redirigirá a la Página de login) o logout en el sistema, registrarse (redirección a Página de registro). También a través de las pestañas se puede volver a la Página de inicio, ver los cuadernos almacenados (Página de cuadernos) o los usuarios registrados en el sistema web (Página de usuarios).

Figura 8. Página de inicio

Página de registro

En esta página (ver Figura 9) se presenta un formulario para que el usuario se dé de alta en el sistema y pueda empezar a añadir sus propios cuadernos de viaje. El usuario debe proporcionar su login (con el que identificarse al sistema), nombre, correo electrónico y una contraseña que deberá repetir para asegurarse de haberla escrito correctamente. Tras cumplimentar todos los datos debe pulsar el botón **Guardar**.

Figura 9. Página de registro

Página de login

A esta página (ver Figura 10) se accede pulsando el botón **Login** que se encuentra en la parte superior derecha de cualquier página. Se muestra un formulario de acceso donde el usuario debe introducir sus credenciales (login y password) para autenticarse en el sistema (pulsando el botón bajo el campo Password). Si los datos introducidos son correctos se redirigirá a la Página de inicio.

Página de usuarios

Esta página (ver Figura 11) muestra todos los usuarios registrados en el sistema web en forma de tabla. Se pueden consultar los detalles de un usuario en concreto pulsando el botón **Ver** situado junto a dicho usuario. El botón **Nuevo** permitirá añadir un nuevo usuario al sistema, redirigiendo a la Página de registro ya comentada. Si el usuario se encuentra logeado aparecerá un botón con el texto **Editar** (en la entrada de la tabla que le corresponda) que le permitirá editar toda la información relativa a su usuario salvo el login.

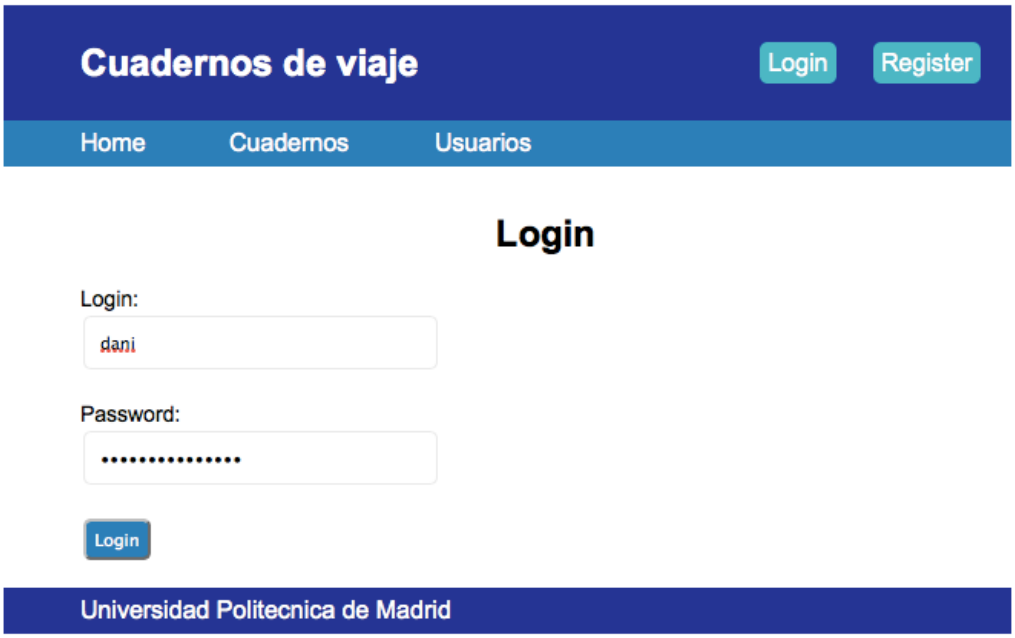


Figura 10. Página de login

Figura 11. Página de usuarios

Página de cuadernos

La Página de cuadernos (ver Figura 12) lista todos los Cuadernos de viaje que han publicado los usuarios en el sistema web. Si el usuario no está logueado solo podrá **ver**, pulsando el botón homónimo, los detalles de los cuadernos existentes (Página detalles de cuaderno). Si por el contrario, el usuario está logueado podrá visualizar el botón **Nuevo** para añadir un nuevo cuaderno en la parte superior izquierda. Además al usuario le aparecerán los botones **Editar** y **Borrar** para modificar o eliminar los cuadernos de los que es propietario. Tanto el botón Nuevo como el botón Editar redirigirán a la Página añadir/editar cuaderno.

Figura 12. Página de cuadernos

Página añadir/editar cuaderno

Si el usuario ha realizado login en el sistema web será capaz de acceder a esta página. Dependiendo desde dónde se haya accedido el contenido a mostrar será distinto:

- Mostrará un formulario en blanco si previamente se pulsó el botón Nuevo en la Página de cuadernos. El usuario deberá rellenar el formulario con el título, la descripción y la ubicación del cuaderno así como las fechas de inicio y fin del viaje en cuestión.

- Mostrará un formulario con los datos correspondientes a un cuaderno de viaje si anteriormente se pulsó el botón Editar sobre uno de los cuadernos. Los campos del formulario serán exactamente los mismos que en caso anterior pero esta vez tendrán el contenido adecuado del cuaderno seleccionado.

Página detalles de cuaderno

Esta página (ver Figura 13) muestra todos los detalles del cuaderno seleccionado. Esto incluye un listado con todos los sitios que componen el cuaderno de viaje. Además, si el cuaderno consta de dos o más sitios, aparecerá el botón **Mostrar ruta** en la parte inferior de la página que al pulsarlo generará un mapa de Google Maps con marcadores de los diferentes sitios del cuaderno enumerados en orden de inserción. Este mapa es interactivo y si se pulsa sobre alguno de los marcadores emergerá un diálogo con información adicional del sitio.

Si el usuario logueado que ha accedido a esta página no es el autor del cuaderno tiene las siguientes opciones: pulsar el botón **Volver** para regresar de vuelta al listado de cuadernos o pulsar el botón **Ver** sobre alguno de los sitios existentes para consultarlo en detalle (accederá a la Página detalles de sitio).

Si el usuario es el autor del cuaderno se le presentarán un conjunto de botones adicionales: **Editar**, en la parte superior para modificar el cuaderno (a Página añadir/editar cuaderno); **Editar**, junto a cada uno de los sitios para cambiar alguna característica de un sitio (a Página añadir/editar sitio); y **Borrar**, a la izquierda del botón anterior, para eliminar un sitio del cuaderno.

Figura 13. Página detalles de cuaderno

- **Página añadir/editar sitio**

Desde esta página el usuario propietario del cuaderno, y en consecuencia del sitio, puede añadir o modificar un sitio. Al igual que en el caso de los cuadernos se presentará un único formulario con contenido diferente, en blanco si es para añadir un nuevo sitio o cumplimentado si es para modificar un sitio. Los campos de este formulario son el nombre, una breve descripción y la ubicación del sitio. Una vez se haya seleccionado una ubicación en su respectivo campo se mostrará un mapa estático de Google Maps con un marcador en el sitio que se está añadiendo o modificando.

- **Página detalles de sitio**

Esta página (ver Figura 14) muestra la información de un sitio en concreto, esto es los datos que se introdujeron al crear o editar el sitio, incluido el mapa estático de Google Maps con la posición. Además contiene una galería en la parte inferior con todas las imágenes del sitio en cuestión. Pulsando sobre la imagen que se esté en un tamaño mayor se accederán a todos sus comentarios. Si el usuario logueado es el autor del sitio tendrá la posibilidad de modificar el sitio utilizando el botón **Editar** (a Página añadir/editar sitio), que se presentará en la parte superior izquierda, y agregar imágenes al sitio con el botón **Añadir imagen** (a Página añadir imagen) que estará sobre la galería.

- **Página añadir imagen**

En esta página el usuario puede añadir una imagen fotografiada de un determinado sitio. Solamente tiene que elegir el fichero que contiene la imagen pulsando el botón **Seleccionar archivo**. Entonces se mostrará una pre visualización de la imagen seleccionada en miniatura. A continuación debe pulsar el botón **Guardar** para añadir la imagen al sitio.

- **Página detalles de imagen**

Esta página (ver Figura 15) contiene la imagen seleccionada previamente de la galería de la Página detalles de sitio y un listado con todos los comentarios existentes sobre dicha imagen. Cualquier usuario registrado en el sistema web puede realizar un comentario en esta página. El propietario de la imagen (aquel que la subió en el sitio que le pertenece) podrá adicionalmente eliminar la imagen con el botón **Borrar imagen** que encontrará bajo ella. Si el usuario logueado hizo algún comentario sobre la imagen mostrada tendrá un botón de **Borrar** para suprimir el comentario y otro botón de **Editar** para modificarlo.

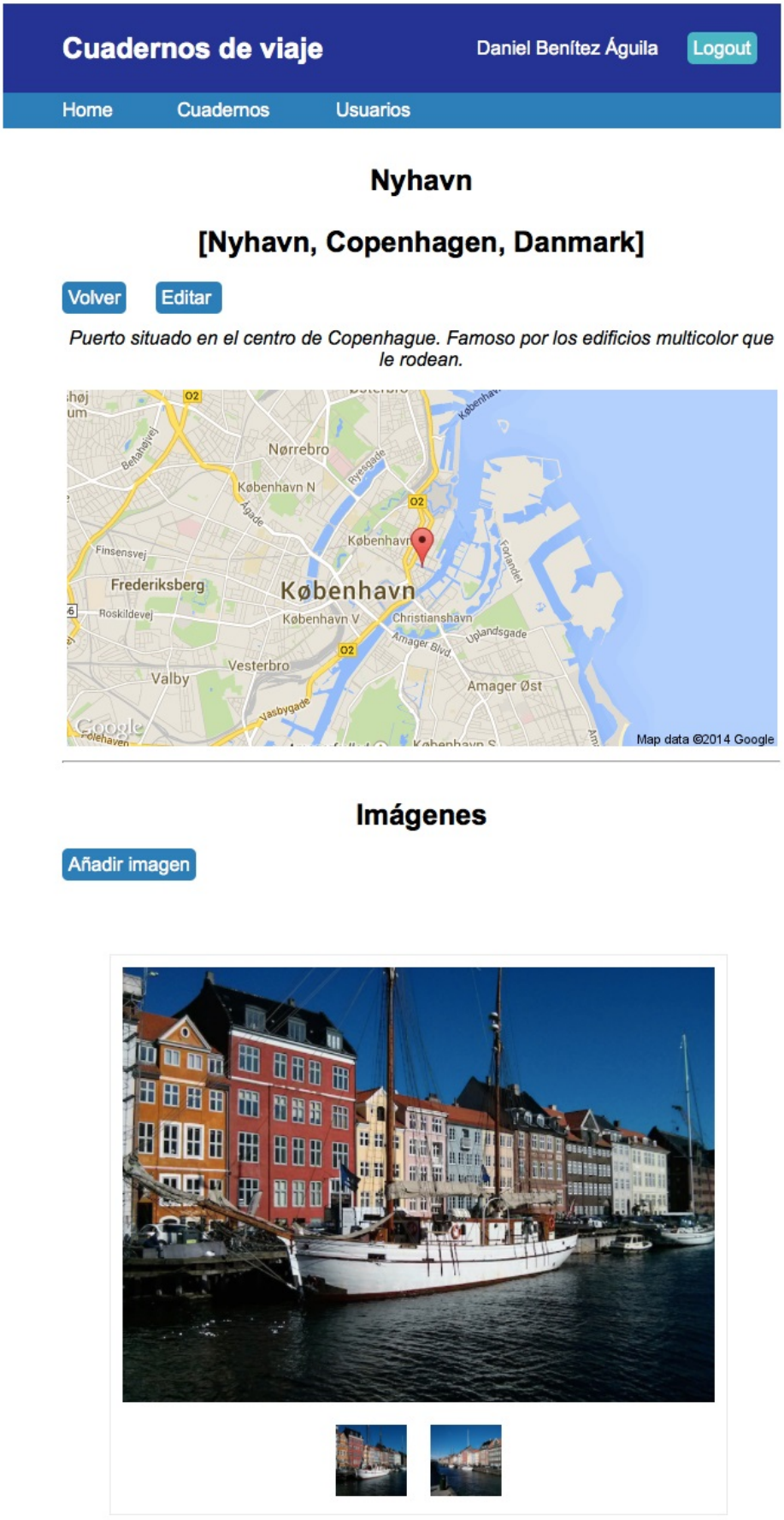


Figura 14. Página detalles de sitio

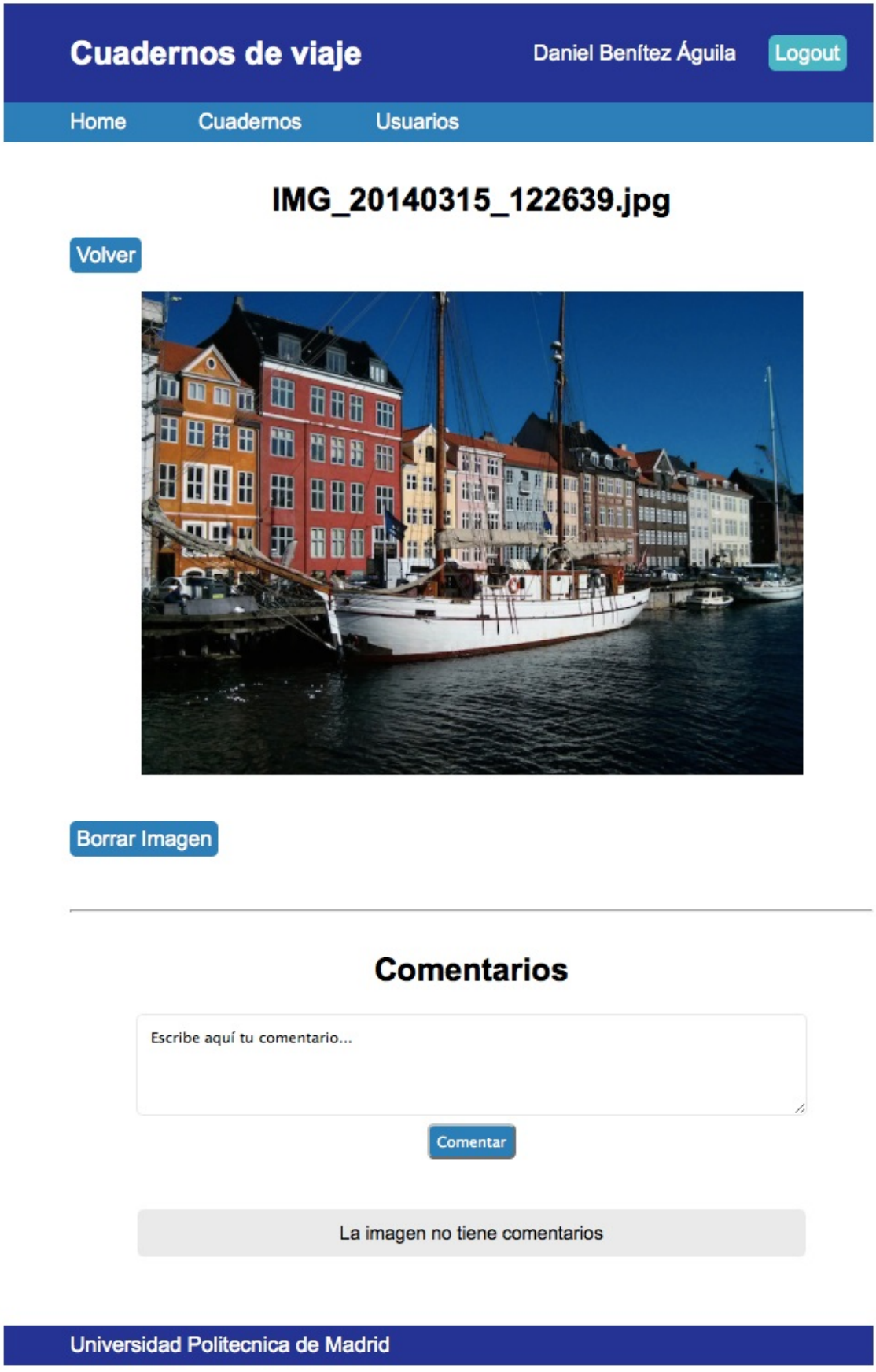


Figura 15. Página detalles de imagen

5 Conclusiones y trabajos futuros

5.1 Conclusiones

En el presente trabajo se ha diseñado, desarrollado e implementado un sistema web complejo para uso didáctico. Se ha conseguido un sistema web completo que consta de una arquitectura de tres capas (presentación, negocio y datos), donde cada una de las capas está claramente definida y diferenciada. Además, haciendo uso de un motor de vistas entre otras tecnologías, el sistema sigue el patrón Modelo-Vista-Controlador. Este ha sido el objetivo principal del trabajo, el cual se ha conseguido, pues el sistema Cuadernos de viaje es perfectamente replicable debido a la extensa documentación existente de las tecnologías utilizadas y a que puede ser descompuesto en pequeñas partes para reproducirlo paso a paso. Esto último se alcanza de varias formas:

- Implementando de manera gradual el modelo de datos: primero los cuadernos (Notebooks), después los usuarios (Users) y sitios (Sites), finalmente las imágenes (Pictures) y comentarios (Comments).

- Desarrollando el código localmente en primera instancia y en una fase más avanzada desplegándolo en la nube.

El resultado del resto de objetivos propuestos en el capítulo Introducción es el siguiente:

- Utilizar las tecnologías del momento que existen en el ámbito del desarrollo de aplicaciones web. Este objetivo se ha conseguido de forma satisfactoria, ya que tecnologías como JavaScript y Node.js están a la vanguardia del desarrollo en la web así como las plataformas de despliegue en la nube. Hacer uso de bases de datos NoSQL en lugar de bases de datos relacionales hubiese sido más innovador, sin embargo, se considera más didáctico emplear bases de datos relacionales.

- Mejorar mi formación así como profundizar en conceptos y competencias de otras asignaturas del Máster relacionadas con el entorno web como "Aplicaciones y Sistemas Colaborativos en la Web 2.0" y "Sistemas de Información y Bases de Datos Web". Este objetivo se ha alcanzado plenamente porque he conseguido combinar los conocimientos adquiridos en ambas asignaturas y ampliarlos adquiriendo una mayor destreza en el desarrollo de aplicaciones web.

5.2 Trabajos futuros

En este apartado se presentan las futuras líneas de trabajo propuestas para continuar trabajando sobre este sistema web. Todas ellas están orientadas hacia mejoras y ampliaciones de dicho sistema.

Seguridad

Actualmente el sistema dispone de un mecanismo de autenticación básico basado en *login* y *password* sin ningún tipo de verificación. Podría implementarse algún mecanismo de verificación, por ejemplo, al crearse un usuario en el sistema que este envíe un correo electrónico a la dirección que se cumplimentó en el campo email del formulario para confirmar su identidad. Otra alternativa es integrar en el sistema el inicio de sesión con alguna red social como Facebook o Twitter a través de sus APIs.

Adaptación a diferentes dispositivos

Cuadernos de viaje se ha desarrollado para ser accesible a través de un navegador. Sin embargo, puede utilizarse en dispositivos móviles como *smartphones* o *tablets* sin problemas (utilizando un navegador). Resultaría interesante crear una aplicación móvil aprovechando la infraestructura desplegada con el fin de adaptar algunos contenidos y el formato de presentación en función del dispositivo. Esta aplicación podría crearse para diferentes sistemas operativos como Android, iOS o Windows Phone.

Portabilidad

El sistema web se encuentra desplegado en la plataforma Heroku. Este PaaS es clasificado como portable al igual que Openshift o AWS Elastic Beanstalk entre otros. Se propone migrar el sistema a otra plataforma, preferiblemente portable, para analizar si realmente el coste de portar aplicaciones web entre plataformas de este tipo es relativamente bajo.

Bibliografía

[1] "JavaEE at a glance", http://www.oracle.com/technetwork/java/javaee/overview/index.html. Consultada el 10/03/2014.

[2] "Your First Cup: An Introduction to the Java EE Platform", http://docs.oracle.com/javaee/5/firstcup/doc/firstcup.pdf. Consultada el 10/03/2014.

[3] "Java Platform, Enterprise Edition", http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition. Consultada el 10/03/2014.

[4] "About Ruby", https://www.ruby-lang.org/en/about/. Consultada el 10/03/2014.

[5] "Ruby (programming language)", http://en.wikipedia.org/wiki/Ruby_(programming_language). Consultada el 10/03/2014.

[6] "JavaScript", http://en.wikipedia.org/wiki/Javascript. Consultada el 10/03/2014.

[7] "JavaScript", https://developer.mozilla.org/en-US/docs/Web/JavaScript. Consultada el 10/03/2014.

[8] "Php", http://en.wikipedia.org/wiki/Php. Consultada el 18/03/2014.

[9] "PHP Manual", obtenido de http://www.php.net/download-docs.php. Consultada el 18/03/2014.

[10] "The Zen of Python", http://legacy.python.org/dev/peps/pep-0020/. Consultada el 18/03/2014.

[11] "Python (programming language)", http://en.wikipedia.org/wiki/Python_(programming_language). Consultada el 21/03/2014.

[12] "About Python", https://www.python.org/about/. Consultada el 21/03/2014.

[13] "TIOBE Software: TIOBE Index", http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. Consultada el 18/03/2014.

[14] "The Transparent Language Popularity Index", http://lang-index.sourceforge.net/. Consultada el 18/03/2014.

[15] "PYPL PopularitY of Programming Language index", https://sites.google.com/site/pydatalog/pypl/PyPL-PopularitY-of-Programming-Language. Consultada el 18/03/2014.

[16] "The RedMonk Programming Language Rankings", http://redmonk.com/sogrady/2013/02/28/language-rankings-1-13/. Consultada el 18/03/2014.

[17] Shan, T.C.; Hua, W.W., "Taxonomy of Java Web Application Frameworks," e-*Business Engineering, 2006. ICEBE '06. IEEE International Conference on* , vol., no., pp.378,385, Oct. 2006 doi: 10.1109/ICEBE.2006.98

[18] "Node.JS", http://nodejs.org/. Consultada el 21/03/2014.

[19] "Meteor", http://docs.meteor.com/. Consultada el 21/03/2014.

[20] "Symfony2: construye tu aplicación, no tus herramientas", http://librosweb.es/symfony_2_3/capitulo_1/symfony2_construye_tu_aplicacion_no_tus_herramientas.html. Consultada el 21/03/2014.

[21] "Java Server Pages Technology", http://www.oracle.com/technetwork/java/javaee/jsp/index.html. Consultada el 10/05/2014.

[22] "String Template", http://www.stringtemplate.org. Consultada el 10/05/2014.
[23] "Haml", http://haml.info. Consultada el 10/05/2014.
[24] "Slim, a fast, lightweight template engine for Ruby", http://slim-lang.com/index.html. Consultada el 10/05/2014.
[25] "Rails template engine performance", http://blog.crowdint.com/2013/07/17/view-engines-performance.html. Consultada el 11/05/2014.
[26] "Liquid Templating Language", http://liquidmarkup.org. Consultada el 11/05/2014.
[27] "Mustache(5). Logic-less template engine", http://mustache.github.io/mustache.5.html. Consultada el 10/05/2014.
[28] "Smarty official website", http://www.smarty.net. Consultada el 10/05/2014.
[29] "Smarty vs Twig", http://umumble.com/blogs/php/249/. Consultada el 10/05/2014.
[30] "Benchmark: Haanga, Mustache, RainTPL, Smarty, Twig", https://github.com/fabpot/Twig/issues/1089. Consultada el 10/05/2014.
[31] "Twig", http://en.wikipedia.org/wiki/Twig_(template_engine). Consultada el 10/05/2014.
[32] "Oracle Documentation", http://docs.oracle.com/. Consultada el 18/03/2014.
[33] D. J. Haderle and C. M. Saracco, "The History and Growth of IBM's DB2," IEEE Ann. Hist. Comput., vol. 35, pp. 54-66, APR-JUN, 2013.
[34] "IBM", http://www.ibm.com. Consultada el 18/03/2014.
[35] "MySQL", http://www.mysql.com. Consultada el 18/03/2014.
[36] X. Pan, W. Wu and Y. Gu, "Study and Optimization Based on MySQL Storage Engine," *Advances in Multimedia, Software Engineering and Computing, Vol 2,* vol. 129, pp. 185-189, 2011.
[37] "SQLite", https://sqlite.org/ Consultada el 18/03/2014.
[38] H. Liu and Y. Gong, "Analysis and Design on Security of SQLite" Proceedings of the International Conference on Computer, Networks and Communication Engineering (Iccnce 2013), vol. 30, pp. 585-588, 2013.
[39] Jing Han, E. Haihong, Guan Le and Jian Du, "Survey on NoSQL database," *Proceedings 2011 6th International Conference on Pervasive Computing and Applications (ICPCA 2011),* pp. 363-6, 2011, 2011.
[40] Popescu , Alex: Presentation: NoSQL at CodeMash – An Interesting NoSQL categorization. February 2010. – Blog post of 2010-02-188. http://nosql.mypopescu.com/post/396337069/presentation-nosql-codemash-an-interesting-nosql
[41] "NoSQL Database" http://nosql-database.org. Consultada el 21/03/2014.
[42] "DB-Engines", http://db-engines.com. Consultada el 21/03/2014.
[43] C. Teixeira, J. S. Pinto, R. Azevedo, T. Batista and A. Monteiro, "The Building Blocks of a PaaS," *Journal of Network and Systems Management,* vol. 22, pp. 75-99, JAN 2014, 2014.
[44] A. Jula, E. Sundararajan and Z. Othman, "Cloud computing service composition: A systematic literature review," *Expert Syst. Appl.,* vol. 41, pp. 3809-24, 15 June 2014, 2014.
[45] C. Edmonson, "Jumpstart: Windows Azure", 06 August 2012, http://www.slideshare.net/clintedmonson/windows-azure-jumpstart. Consultada el 14/04/2014.
[46] D.M. Smith, "*Hype cycle for cloud computing*", Gartner, 2013.
[47] L. Carlson, *Programming for PaaS*. O'Reilly Media, Inc., 2013.
[48] "Google Developers - Google App Engine", https://cloud.google.com/products/app-engine/. Consultada el 19/03/2014.

[49] The Go Programming Language", http://golang.org/. Consultada el 19/03/2014.
[50] "Microsoft Azure", http://azure.microsoft.com/es-es/. Consultada el 14/04/2014.
[51] R. Buyya, J. Broberg and A. Gościńnski, "*Cloud Computing Principles and Paradigms*". Hoboken, N.J.: John Wiley & Sons, 2011.
[52] M. Sala and L. Colombo, "CLOUD COMPUTING: A REVIEW OF PAAS, IAAS, SAAS SERVICES AND PROVIDERS," *Lámpsakos,* vol. 7, pp. 47-57, january-june 2012, 2012.
[53] "Salesforce1 platform", http://www.salesforce.com/platform/overview/. Consultada el 24/03/2014.
[54] "Heroku", https://www.heroku.com/features. Consultada el 24/03/2014.
[55] "OpenShift", https://www.openshift.com/. Consultada el 30/03/2014.
[56] "AppFog", https://www.appfog.com/. Consultada el 30/03/2014.
[57] "Cloud Foundry Community", http://cloudfoundry.org/index.html. Consultada el 30/03/2014.
[58] "AWS Elastic Beanstalk", http://aws.amazon.com/es/elasticbeanstalk/. Consultada el 11/04/2014.
[59] B. Cohen, "PaaS: New Opportunities for Cloud Application Development," *Computer,* vol. 46, pp. 97-100, SEP 2013, 2013.