Desarrollo de una aplicación para el turismo en Madrid

Trabajo Fin de Grado
Curso 2018-2019

Autor
Eduardo Vela Galindo

Director
Antonio Sarasa Cabezuelo

Departamento de Sistemas Informáticos y Computación

Universidad Complutense de Madrid

Madrid, 2018

AGRADECIMIENTOS

Agradezco enormemente a Antonio Sarasa el haber aceptado este proyecto y haberme ayudado tanto pese a las dificultades existentes.

A mi familia, especialmente a mis padres, por haber estado ahí todo el tiempo siempre que los necesitaba para lo que fuera.

A mis compañeros de grado Carlos, Carmen, Iván, Fernando, Francisco, Rodrigo y Tomás ya que sin ellos nunca habría podido acabar esta carrera.

RESUMEN

Las bases de datos abiertas constituyen una ingente fuente de datos públicos disponibles en cualquier momento a través de internet, lo cual supone una gran oportunidad de aprovechamiento a la hora de desarrollar aplicaciones en base a ellos, que de otro modo necesitarían un enorme esfuerzo de recopilación de datos.

La finalidad de este proyecto ha sido desarrollar una aplicación web que haga uso de fuentes de datos abiertas. La aplicación está destinada al turismo de la ciudad de Madrid y recopila la información de la API de datos abiertos del Ayuntamiento de Madrid. Para su implementación se ha utilizado tecnología Node.

Con el objetivo de mostrar el uso de estos de datos, la aplicación permite la creación de visitas a la ciudad de Madrid. Cada visita consta de una serie de lugares a visitar y las diferentes interacciones que pueden llevar a cabo los usuarios con las visitas.

ABSTRACT

Open data is a huge source of public data available at any time through the Internet. This is a great opportunity to take advantage when developing applications based on them, which otherwise would require a huge effort to compile the data.

The purpose of this project is to develop an application that uses these data sources assuming an example of the power of these applications. Specifically, a web application for tourism has been developed for the city of Madrid that collects data from the city Council open data API. This web application has been made under Node technology and using open data sources.

To show the application of open data, the tourism portal has focused on the creation of visits to the city of Madrid, which consist of a series of places to visit and the different interactions that can be carried out by users within their visits

PALABRAS CLAVE

Datos abiertos, Arquitectura multicapa, Node.js, Aplicación Web, Gestor de Contenido Web, API

ACRÓNIMOS

API: Application Programming Interface

EJS: (Embedded JavaScript templates)

BD: Base de datos

CSS: Cascading Sytle Sheets

HTML: HyperText Markup Lenguage

JS: Javascript

JSON: Javascript Object Notation

NPM: Node Package Manager

TFG: Trabajo de Fin de Grado

UCM: Universidad Complutense de Madrid

ÍNDICE

AGRADECIMIENTOS 3
RESUMEN 4
ABSTRACT 5
PALABRAS CLAVE 6
ACRÓNIMOS 7
1. INTRODUCCIÓN 1
1.1. MOTIVACIÓN 2
1.2. OBJETIVOS 3
2. ESTADO DEL ARTE 4
3. TECNOLOGÍA EMPLEADA 8
3.1. TECNOLOGÍA DEL SERVIDOR: NODE.JS 9
3.1.1. NODE PACKAGE MANAGER 9
3.1.2. EXPRESS.JS 10
3.1.3. MYSQL 11
3.1.4. OTROS MÓDULOS 12
3.2. TECNOLOGÍAS DEL CLIENTE 12
3.2.1. BOOTSTRAP 13
3.2.2. JAVASCRIPT 13
3.2.3. JQUERY Y AJAX 13
3.3. OTRAS TECNOLOGÍAS. 14
3.3.1. MARÍADB 14
3.4. APACHE 15
3.5. XAMPP 15
4. ESPECIFICACIÓN 16
4.1. ACTORES DEL SISTEMA 17
4.2. DIAGRAMA DE CASOS DE USO 18
4.3. CASOS DE USO 19
4.3.1. CASOS DE USO DEL VISITANTE 19
4.3.2. CASOS DE USO DEL USUARIO REGISTRADO 23
4.3.3. CASOS DE USO DEL ADMINISTRADOR 29
5. MODELO DE DATOS 36
5.1. MODELO ENTIDAD-RELACIÓN 37
5.2. ESTRUCTURA DE LA BASE DE DATOS. 38
5.3. ESTRUCTURA DE LAS TABLAS DE LA BD 39
5.3.1. TABLA USUARIOS 39

5.3.2. TABLA VISITAS 40
5.3.3. TABLA MONUMENTOS 41
5.3.4. TABLA VISITAS RECOMENDADAS 42
5.3.5. TABLA COMENTARIOS 42
5.3.6. TABLA SESSIONS 42
6. ARQUITECTURA DE LA APLICACIÓN 44
7. DISEÑO DE LA APLICACIÓN 48
8. CONCLUSIONES Y TRABAJO FUTURO 56
8.1. CONCLUSIONES 57
8.2. TRABAJO FUTURO 57
8. CONCLUSIONS AND FUTURE WORK 59
8.1. CONCLUSIONS 60
8.2. FUTURE WORK 60
ANEXO 62
ANEXO I: GUÍA DE INSTALACIÓN 63
BIBLIOGRAFÍA 72
ÍNDICE DE FIGURAS 75

1. INTRODUCCIÓN

1.1.MOTIVACIÓN

Las fuentes de datos abiertos (Auer Sören, 2014) consisten en repositorios de datos accesibles que pueden ser consultados libremente por cualquiera utilizando normalmente una API de servicios web.

Estos repositorios pertenecen a organizaciones tanto públicas (gobiernos de países) como privadas (Google[1]), y su propósito principal es el ofrecer los datos para ser utilizados por cualquiera que los necesite, siguiendo así una filosofía similar al del software libre.

Concretamente en España se pueden encontrar una gran cantidad de fuentes de datos abiertos públicos, algunas de ellas muy ricas y completas como las fuentes de datos abiertos del gobierno. En este proyecto, se ha utilizado la fuente de datos abiertos del Ayuntamiento de Madrid[2].

La fuente de datos abiertos del Ayuntamiento de Madrid ofrece una gran cantidad de datos de diferente índole tales como accidentes de tráfico, emisiones de CO_2, contaminación acústica, etc. Estos datos pueden ser recuperados utilizando una API REST[4] que permite enviar peticiones GET para obtener los datos deseados y devolver en base a las peticiones realizadas, conjuntos de datos en diversos formatos de datos.

Utilizando los datos abiertos se pueden crear nuevos servicios que no existían previamente. En particular, en este proyecto se han utilizado datos abiertos acerca de museos, templos y monumentos con el objetivo de crear un servicio de valor añadido destinado a turistas que gestione toda esta información.

[1] Enlace a web de una API de datos de Google: https://developers.google.com/gdata/
[2] Enlace al portal de datos abiertos del ayuntamiento de Madrid: https://datos.madrid.es/portal/site/egob/

1.2.OBJETIVOS

El objetivo principal de este trabajo es mostrar el potencial que ofrecen las fuentes de datos abiertas para crear nuevos contenidos a partir de las mismas. Partiendo de este objetivo general, se definen un conjunto de objetivos más específicos:

- Desarrollar una aplicación web destinada a crear visitas culturales en el contexto de la ciudad de Madrid.

- Usar fuentes de datos abiertas para recuperar información de terceros y en base a ella ofrecer un servicio nuevo con fines turísticos.

- Implementar un gestor de contenido web que permita acceder a la información recuperada y utilizarla de forma dinámica para crear y buscar visitas turísticas centradas en Madrid.

2. ESTADO DEL ARTE

No se han encontrado páginas que ofrezcan un servicio explícitamente similar al ofrecido por la aplicación desarrollada en este proyecto (permiten crear y guardar visitas personalizas por la ciudad), sin embargo, si existen numerosos portales web que ofrecen información turística sobre la ciudad de Madrid, aunque no es posible conocer si la información que almacenan ha sido obtenida de fuentes de datos abiertas o es propia.

Algunos de estos ejemplos se listan a continuación, proporcionando una breve descripción sobre que ofrecen y un enlace a los mismos:

- INSPIROCK[3]. Este sitio web ofrece la posibilidad de crear visitas para casi cualquier ciudad del mundo. Algunos de los puntos destacados de esta web son:
    - No ofrece la posibilidad de personalizar las visitas, sino que las genera en base a una serie de parámetros que se introducen al generar la visita.
    - La información que ofrece de los distintos lugares turísticos es bastante limitada.
    - También es destacable el carácter lucrativo de esta página, que ofrece planes económicos de terceros y se aleja del carácter meramente informativo.
    - El sitio web es bastante confuso y difícil de encontrar lo que se busca en general.
- NYCGO, The Official Guide[4]. Se trata de una página web que proporciona información sobre la ciudad de Nueva York, incluyendo museos, monumentos y toda clase de lugares relevantes para el turista. Del análisis de este sitio web se desprende:

[3] https://www.inspirock.com/
[4] https://es.nycgo.com/

- Aunque no ofrece la posibilidad de crear visitas, si ofrece información completa sobre los distintos monumentos que pueden ser visitados.

- No ofrece la posibilidad de valorar los monumentos visitables.

- Al igual que el ejemplo anterior el acceso a la información es confuso.

- CivitatisMadrid[5] En este caso tenemos una página centrada exclusivamente en el turismo en la ciudad de Madrid. Tras analizar el sitio web, estos son algunos de los puntos destacados:

  - No permite personalizar las visitas, sino que ofrece una serie de visitas estandarizadas.

  - La página tiene un marcado fin lucrativo, ya que las visitas propuestas están enfocadas a contratar a guías propuestos por el propio sitio.

  - Se centra más en aspectos económicos como hoteles, aviones y logística en general que en la propia oferta turística.

  - Ofrece un número muy limitado de monumentos, centrado en su mayoría en los cuatro o cinco más famosos de cada tipo.

- esmadrid[6]. Esmadrid es una web turística centrada solo en la ciudad de Madrid y en la oferta turística que oferta la ciudad. Esta guía proporciona una gran cantidad de información de todo tipo de eventos y monumentos a visitar, algunos de los puntos destacados son:

  - No permite crear visitas personalizadas.

[5] https://www.disfrutamadrid.com/
[6] http://www.esmadrid.com

- No ofrece un listado completo de los monumentos de la ciudad de Madrid, tan sólo algunos representativos.

- La información ofrecida va más encaminada a un enfoque de ocio más general, ofreciendo conciertos, partidos de fútbol y toda clase de eventos que tienen lugar en la ciudad de Madrid.

3. TECNOLOGÍA EMPLEADA

3.1. TECNOLOGÍA DEL SERVIDOR: NODE.JS

Node es un entorno de programación en tiempo de ejecución multiplataforma (Herron, 2011) principalmente para la capa del servidor basado en el lenguaje de programación ECMAScript y en el motor V8 de Google (entorno de ejecución JavaScript de Google Chrome[7]). Una de las principales cualidades de Node es su funcionamiento concurrente asíncrono, que utiliza un hilo de ejecución principal para la aplicación, del que se generan el resto de las llamadas concurrentes de entrada/salida[8](Mark Clements, 2014).

Otra de las cualidades más llamativas y apreciadas de Node que importa del motor V8, es el uso de módulos que extienden enormemente la funcionalidad de Node (Mark Clements, 2014) y le convierten en un entorno escalable muy configurable. Estos módulos pueden ser enormes y añadir una gran funcionalidad como Express[9] o añadir una funcionalidad más concreta y acotada como puede ser el módulo MySQL que permite lanzar y recibir consultas de una base de datos SQL.

El carácter asíncrono y "no bloqueante" de las llamadas de datos recibidas sumado a una alta tolerancia de concurrencia le convierten en un lenguaje perfecto para aplicaciones web ligeras y es ese el motivo por el que ha sido elegido para implementar la aplicación web. Además, se encuentra bajo licencia MIT de código abierto, por lo que puede ser utilizada sin problemas de derechos.

A continuación, se van a comentar con más profundidad algunos detalles sobre la tecnología Node empleada. En concreto se hará un repaso por algunos de los módulos más destacados que se han utilizado.

3.1.1. NODE PACKAGE MANAGER

Node Package Manager (de ahora en adelante npm) es el manejador de paquetes de Node, el cual se usa para instalar y mantener actualizados todos los módulos y requisitos de una aplicación Node (Herron, 2011). Este tiene un funcionamiento similar al de otros

[7] Proyecto V8 de Google https://v8.dev
[8] Página web oficial de Node https://nodejs.org/en/about/
[9] Página oficial de Express https://expressjs.com/es/

manejadores de paquetes tradicionales como puede ser Yum[10]. Aunque también se puede utilizar para instalar y/o actualizar aplicaciones Node enteras.[11]

npm se instala automáticamente junto con Node desde la versión 0.6.3 (Node documentation, 2018) y se accede a él mediante el comando "npm" lanzado desde una terminal con acceso a Node. Algunos ejemplos significativos de npm, y que serán necesarios más adelante durante la guía de instalación son npm install, que permite instalar todas las dependencias de un proyecto (Node documentation, 2018), o npm update, que actualiza a la última versión todas las dependencias del proyecto (Node documentation, 2018).

Algunas herramientas que se han utilizado para desarrollar la aplicación de forma más eficiente han sido Nodemon y Forever, que permiten más facilidades a la hora de trabajar con servidores Node, como por ejemplo que un script se esté ejecutando continuamente sin necesidad de lanzarlo manualmente cada vez que lo actualizas (Mark Clements , 2014).

3.1.2. EXPRESS.JS

Express es un framework[12] (Yaapa, 2013) diseñado y enfocado para desarrollar aplicaciones web y APIS en Node (Herron, 2011). Su uso se ha popularizado mucho (solo en github se puede encontrar más de 120.000 proyectos que lo utilizan y que suponen aproximadamente un 30% del total de proyectos Node de Github[13] y además según npm-stats acumula más de medio millón de descargas[14] como muestra la figura 3.1). En este sentido, se considerado el "framework" por defecto si se va a desarollar aplicaciones web en Node (Yaapa, 2013).

[10] Enlace a página de Wikipedia de Yum: https://en.wikipedia.org/wiki/Yum_(software)
[11] Enlace al proyecto npm https://www.npmjs.com/
[12] Un framework es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia para enfrentar y resolver problemas de índole similar.
[13] Datos obtenidos directamente de Github https://github.com
[14] Datos obtenidos de NpmStats https://npm-stat.com/charts.html?package=express&from=2010-05-01&to=2019-01-01

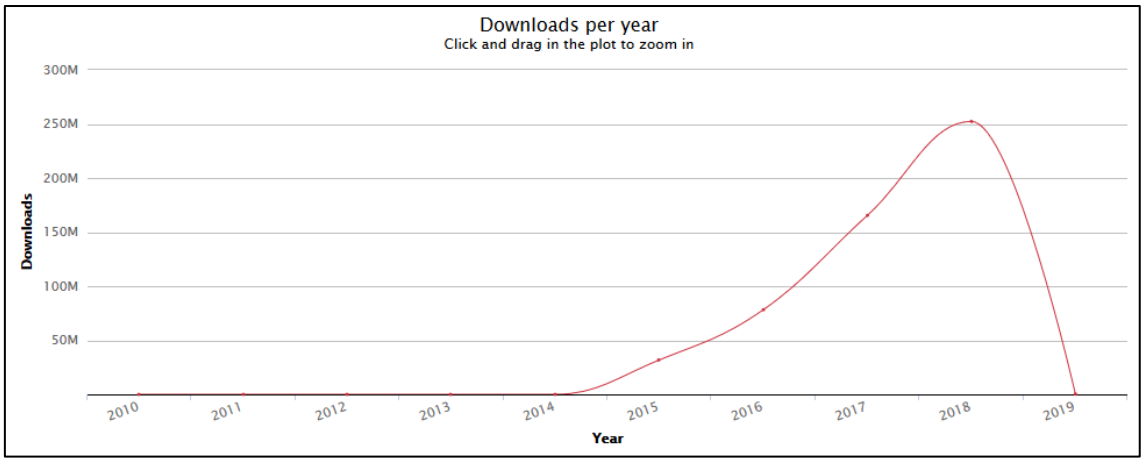


Figura 3-1. Descargas por año del framework Express.js. Fuente: https://npm-stat.com

Otro factor determinante para elegir este framwork es la gran cantidad de excelentes cualidades (Vlăduțu, 2014) que presenta tales como su versátil o la posibilidad de extender su funcionalidad medienta el uso del sistema de "middlewares" (Yaapa, 2013) o crear funcionalidad propia (Herron, 2011). A continuación, en la figura 3.2 se muestra el funcionamiento de los middleware en Express. Así, se puede observar que cuando Express recibe una petición del cliente en el servidor, este va pasando por distintas fases (cada una de estas fases es un middleware) y al concluir dicha fase se encadena con la siguiente hasta terminar (Yaapa, 2013).

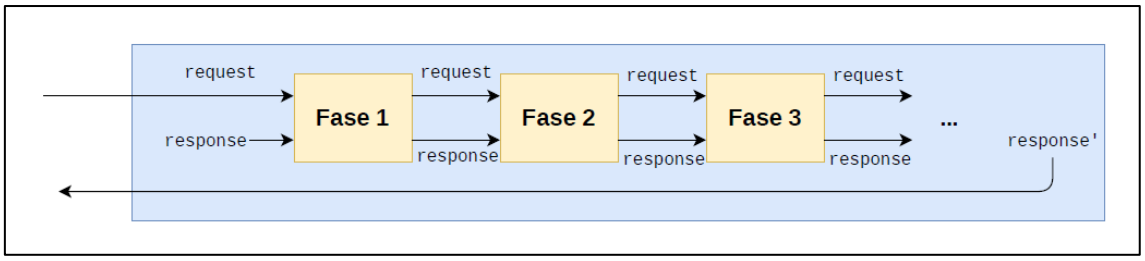


Figura 3.2.Funcionamiento de los middlewares en Express.js

3.1.3. MYSQL

Este módulo actúa como driver de Node para la utilización de MySQL (Wisborg Krogh, 2017). Se encuentra bajo licencia MIT y está escrito íntegramente en JavaScript. En este proyecto, se ha utilizado combinado con otros módulos menores que extienden la funcionalidad de este como MySQLStore y sesión-store. Este módulo permite establecer una conexión con una base de datos MySQL simplemente indicando el servidor que contiene la base de BD y el nombre de la BD en cuestión. Una vez realizada la conexión

se pueden lanzar consultas desde la aplicación, y el módulo se encarga de traducirlas y hacerlas llegar a la BD (Wisborg Krogh, 2017) bajo los parámetros establecidos.

3.1.4. OTROS MÓDULOS

Los dos módulos citados anteriormente son los más representativos del proyecto, el resto de módulos utilizados solo se van a citar indicando brevemente su funcionalidad:

- Request: Simplifica enormente el manejo de rutas http y https[15].
- CreateError: Permite generar errores personalizados[16].
- ExpressSession: Permite generar y gestionar sesiones de usuarios (Vlăduțu, 2014).
- ExpressMysqlSession: Permite insertar y mantener las sesiones en BD[17].
- Path: simplifica el uso de rutas internas para establecer la localización de archivos locales (Yaapa, 2013).
- CookieParser: Permite parsear el contenido de datos, por ejemplo procesar documentos en JSON cuando estos están contenidos en una cookie (Yaapa, 2013).
- Async: Añade una gran cantidad de funcionalidades para resolver la problemática de trabajar en un entorno asíncrono (Vlăduțu, 2014).
- Wdk: Simplica la conexión con la API de datos Wikidata[18].

3.2. TECNOLOGÍAS DEL CLIENTE

Con respecto a la tecnología del cliente, al tratarse de una aplicación web, se ha elegido como tecnología HTML5 y CSS3. En concreto, se ha utilizado como plantillas el formato EJS, Embedded JavaScript Templates (Mark Clements, 2014). En estas plantillas el código HTML se encarga de generar la estructura y el código CSS se encarga de los estilos (Grannell, 2007). La principal cualidad de las plantillas EJS es que permiten incrustar código JavaScript muy fácilmente en ellas sin necesidad de asociar un archivo JavaScript específico para ello y también permiten insertar código HTML direcamente en

[15] Documentación oficial del módulo request por sus autores: https://github.com/request/request
[16] Enlace a documentación de CreateError: https://gist.github.com/justmoon/15511f92e5216fa2624b
[17] Enlace a documentación de ExpressMySqlSession: https://www.npmjs.com/package/express-mysql-session
[18] Enlace a documentación de Wdk: https://www.npmjs.com/package/wikidata-sdk

el navegador generado dinámicamente desde un archivo JavaScript mediante una llamada AJAX al servidor (Garcia-Izquierdo, 2012).

3.2.1. BOOTSTRAP

Bootstrap es una tecnología web que establece una serie de diseños predefinidos (Krause, 2016) que facilitan la maquetación de una web. Básicamente consiste en una serie de documentos CSS que contienen clases para las etiquetas más comunes HTML, e incluyen tipografías, botones, paneles, menús y casi todos los objetos HTML (Krause, 2016). Así mismo, implementa animaciones y funcionalidades propias como revisores de formularios (Krause, 2016), y una amplia gama de personalización de estas.

3.2.2. JAVASCRIPT

Debido a la escasa versatilidad de programación ofrecida por HTML+CSS, que no permiten realizar tareas complejas, se ha visto necesario incluir JavaScript que así se puedan realizar determinadas acciones en el cliente y descargar de carga al servidor. Y es que la potencia que aporta el lenguaje JavaSript en el cliente es altísima y por ello su uso está altamente generalizado en el desarrollo web. Aunque es cierto que puede suponer algunos riesgos para la seguridad del cliente (Ritchie, 2007) sus capacidades claramente valen la pena, ya que nos van a permitir relegar una gran carga de operaciones del servidor al cliente, además de agilizar muchos procesos y de dar una sensación de mayor velocidad y rendimiento de la aplicación web.

3.2.3. JQUERY Y AJAX

Para incrementar las funcionalidades de Javascript se han utilizado como tecnologías complementarias JQuery (JQuery API documentation, 2018) y Ajax(Ajax API documentation, 2018).

JQuery es una librería de JavaScript que permite interactuar de manera muy simple y potente con el árbol DOM de las plantillas HTML asociadas a un script. Además, también es posible modificar dinámicamente elementos tanto del árbol DOM como sus estilos asociados (CSS). Un ejemplo sería la inserción de código HTML generado de forma

dinámica en base a la respuesta de una llamada al servidor, permitiendo así que solo se genere el código HTML que se desee.

AJAX (*Asynchronous JavaScript And XML*) es una tecnología web que permite a las aplicaciones interactuar de forma interactiva y dinámica con el servidor mediante una conexión asíncrona permitiendo así poder realizar peticiones con el servidor sin necesidad de recargar la página del navegador. Esta propiedad mejora la velocidad y usabilidad de las aplicaciones (Smith, 2015), además de ofrecer al usuario una mayor sensación de

Figura 3-3. Funcionamiento básico de AJAX. Fuente: Smith, 2015.

control de la aplicación. En la figura 3.3 se muestra un ejemplo simple de su funcionamiento:

3.3.OTRAS TECNOLOGÍAS.

3.3.1. MARÍADB

MySQL es un gestor de bases de datos relacionales que se encuentra bajo una licencia dual (código libre y otra comercial bajo Oracle). Concretamente, se ha utilizado el SGDB MariaDB, un sistema derivado de MySQL bajo licencia GPL (Wisborg Krogh, 2017).

Aunque MySQL y MariaDB son prácticamente iguales, se ha elegido MariaDB pues incorpora algunas diferencias en los mecanismos de almacenamiento y en prestaciones, que hacen más atractiva su uso.

PhpMyAdmin (PhpMyAdmin docs, 2018) es un programa utilizado para manejar y administrar bases de datos de tipo MySQL (Delisle, 2004) de una forma sencilla a través

de un navegador Web. En este sentido permite realizar las operaciones típicas sobre una base de datos (modificar campos, eliminar, crear o modificar tablas, eliminar o insertar tuplas entre otras muchas opciones) de forma interactiva sin necesidad de introducir manualmente los comandos SQL (Delisle, 2004).

3.4. APACHE

Apache HTTP Server es un servidor web (Vukotic, 2011) que implementa el protocolo HTTP, gratuito y de código abierto. La función principal de Apache es ejercer de servidor web ofreciendo contenido online cuando el cliente solicita un recurso al servidor. No se trata de un servidor físico, sino de un servidor virtual cuyo trabajo es hacer de controlador entre el cliente de la aplicación (un navegador) y el servidor de una aplicación. Cuando el cliente solicita información a través del navegador, es Apache el encargado de envíar toda la información requerida por el cliente para poder visualizar la página y que incluye código HTML, CSS, JavaScript, imágenes, etc.

3.5. XAMPP

Se trata de una herramienta de software libre que se utiliza para facilitar la gestión de PHP, MySQL y Apache (Gibbs, M). La herramienta ofrece integradas las tecnologías indicadas de manera que MySQL y Apache ya que se encuentran configurados para su ejecución y uso bajo diversos sistemas operativos. Por lo tanto su principal finalidad es la de ejercer de gestor de servidores web en un entorno local, permitiendo simular la conexión a un host remoto mediante un cliente local.

Xampp También permite configurar y personalizar los servidores a utilizar como si se utilizaran de forma nativa, proporcionando acceso a las consolas y un un completo log de errores en caso de fallo.

4. ESPECIFICACIÓN

4.1. ACTORES DEL SISTEMA

En la aplicación, se diferencian tres tipos de actores distintos:

- Visitante: Accede a la aplicación sin necesidad de autenticarse y puede ejecutar operaciones que no requieran persistencia de datos.

- Usuario registrado: Este actor debe estar registrado en el sistema. En este sentido, accede a la funcionalidad una vez que se ha autenticado satisfactoriamente. Puede llevar a cabo todas las acciones del visitante, además de las suyas propias que incluyen aquellas que necesiten persistencia en los datos.

- Administrador: Este actor debe estar registrado en el sistema (una diferencia con los usuarios registrados, es que el registro se realiza de manera manual en la BD). En este sentido, accede a la funcionalidad una vez que se ha autenticado satisfactoriamente. Lleva a cabo operaciones de mantenimiento del sistema, aunque también puede realizar las acciones propias de un visitante.

4.2.DIAGRAMA DE CASOS DE USO

A continuación se presenta el diagrama de casos de uso donde se representa el alcance de la aplicación.

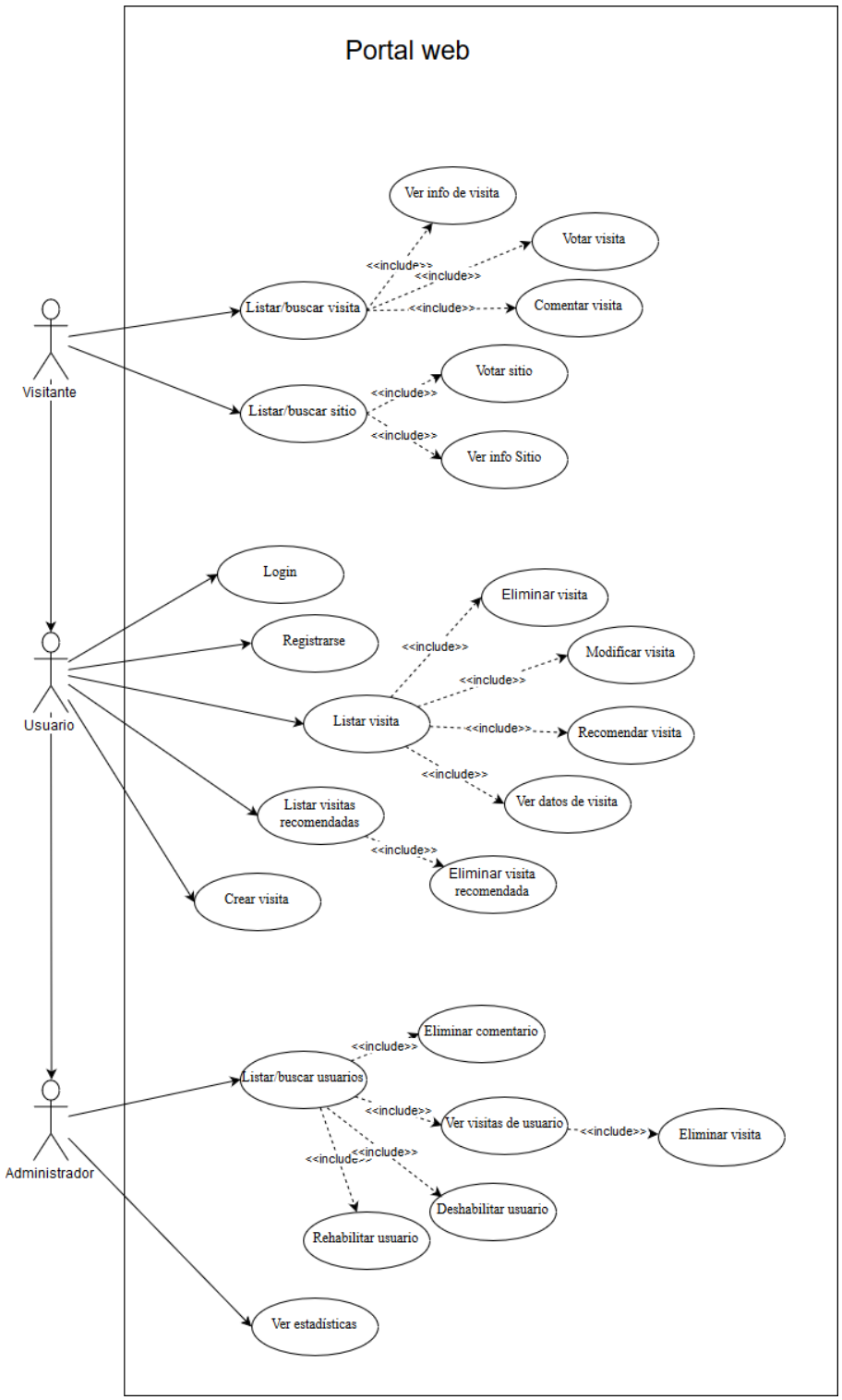


Figura 4-1. Diagrama de casos de uso

4.3.CASOS DE USO

A continuación, se van a describir los casos de uso representado en el diagrama. Los casos de uso se han clasificado según los roles que puede adoptar el usuario y que se han listado previamente. Para facilitar su comprensión de la descripción de los casos de uso, se van a definir los siguientes conceptos:

- Visita: Es una entidad compuesta por un título, una descripción, los lugares que se desean visitar y tiene siempre asociado un usuario creador.
- Monumento: Es un lugar concreto que se desea visitar como por ejemplo el Museo del Prado.
- Sistema: Hace referencia a la aplicación implementada

4.3.1. CASOS DE USO DEL VISITANTE

Buscar una visita (CU-V-01)	
Entrada	Texto
Descripción	Busca una visita por coincidencia del texto introducido en el título o la descripción de la visita.
Dependencias	n/a
Precondición	Texto introducido en el buscador.
Postcondición	Listados todos los usuarios que coincidan.
Actores	Visitante, usuario registrado, administrador
Flujo del evento	1. El visitante introduce la cadena de texto que desea buscar en un cuadro de texto habilitado para ello. 2. El sistema busca en la BD la palabra clave introducida y devuelve aquellas visitas que contienen en su título, descripción o en su lista de palabras claves una coincidencia. Puede haber dos posibles resultados: 2.1. Hay coincidencias: en cuyo caso se listan las 5 primeras visitas para las que se encontró una coincidencia. 2.2. No hay coincidencias: en cuyo caso se informa al usuario y no se lista nada.

Ver información de una visita (CU-V-02)	
Entrada	Visita seleccionada.
Descripción	Lista la información asociada a dicha visita, como su descripción, localización, mapa interactivo asociado, etc.
Dependencias	n/a
Precondición	Existe al menos una visita en el sistema.
Postcondición	Se muestra la información de la visita
Actores	Visitante, usuario registrado, administrador
Flujo del evento	1. El visitante selecciona el icono de ver información asociada a una visita, representado de forma universal por un 👁 en toda la aplicación. 2. El sistema busca la información correspondiente a esa visita y abre un panel sobre el que se muestra la información que ha encontrado.

Votar una visita (CU-V-03)	
Entrada	Visita seleccionada. Voto numérico (entre 1 y 5).
Descripción	Permite votar la visita especificada.
Dependencias	CU-V-02
Precondición	Existe una visita. Se ha seleccionado ver información de una visita.
Postcondición	Se inserta el voto en el sistema
Actores	Visitante, usuario registrado, administrador
Flujo del evento	1. El visitante selecciona la visita a la que desea calificar. 2. Se abre un panel con la información asociada a la visita, incluyendo un apartado indicando la puntuación actual de la visita y la posibilidad para votar la visita. 3. El visitante establece una puntuación entre 1 y 5 y selecciona la opción de enviar. 4. El sistema contabiliza el voto e informa de ello al visitante.

Comentar una visita (CU-V-04)	
Entrada	Visita seleccionada. Texto con el comentario.
Descripción	Permite comentar la visita especificada con el texto introducido en el apartado del comentario.
Dependencias	CU-V-02
Precondición	Existe una visita. Se ha seleccionado ver información de una visita. Se ha introducido un texto con el comentario.
Postcondición	Se inserta el comentario en la BD
Actores	Visitante, usuario registrado, administrador
Flujo del evento	1. El visitante selecciona la visita sobre la que desea comentar. 2. Se abre un panel con la información asociada a la visita, incluyendo un apartado indicando los comentarios asociados y la posibilidad de escribir un nuevo comentario. 3. El visitante escribe en un cuadro habilitado para ello el comentario deseado y selecciona la opción de enviar comentario. Puede haber dos posibles resultados: 3.1. Se contabiliza el comentario correctamente y se informa al visitante. 3.2. Se encuentra un error en el comentario (por ejemplo que esté vacío) y se informa de ello al visitante.

Listar Monumentos (CU-V-05)	
Entrada	N/a.
Descripción	Lista todos los monumentos disponibles en el sistema, que se corresponden con los monumentos turísticos proporcionados por la API de datos abiertos del ayuntamiento de Madrid.
Dependencias	N/a
Precondición	N/a
Postcondición	Se listan todos los monumentos disponibles

Actores	Visitante, usuario registrado, administrador
Flujo del evento	1. El visitante selecciona la opción de Monumentos en el menú de navegación del portal. 2. Se redirecciona a una nueva página del portal, donde se listan todos los monumentos asociados al sistema.

Buscar Monumento (CU-V-06)	
Entrada	Texto con la clave a buscar y/u otras opciones.
Descripción	Busca el monumento correspondiente en base al texto introducido.
Dependencias	CU-V-05
Precondición	Existe al menos un monumento en el sistema. Texto introducido
Postcondición	Se listan los monumentos con coincidencias
Actores	Visitante, usuario registrado, administrador
Flujo del evento	1. El visitante introduce la cadena de texto que desea buscar en un cuadro de texto habilitado para ello. También puede introducir otras opciones de filtrado. 2. El sistema busca entre todos los monumentos con dos posibles resultados: 2.1. Hay coincidencias: se muestran los monumentos con coincidencias encontradas. 2.2. No hay coincidencias: no se muestra nada.

Ver información de un monumento (CU-V-07)	
Entrada	Monumento seleccionado.
Descripción	Lista la información asociada a dicho monumento, como su descripción, localización, mapa interactivo asociado, etc.
Dependencias	CU-V-05
Precondición	Existe al menos un monumento.
Postcondición	Se muestra la información correspondiente al monumento seleccionado.

Actores	Visitante, usuario registrado, administrador
Flujo del evento	1. El visitante selecciona el monumento del cual quiere ver su información asociada. 2. El sistema busca la información correspondiente a ese monumento y abre un panel sobre el que se muestra la información que ha encontrado.

4.3.2. CASOS DE USO DEL USUARIO REGISTRADO

Login o autentificación (CU-UR-01)	
Entrada	Email del usuario que se desea autentificar. Contraseña del usuario que se desea autentificar.
Descripción	Permite a un usuario autenticarse en el sistema y pasar así de visitante a usuario registrado.
Dependencias	N/a
Precondición	Se introduce un email y una contraseña. No se ha iniciado sesión.
Postcondición	• Éxito: Si el email existe en la BD y la contraseña se corresponde con el email el usuario queda autenticado. • Fracaso: Si el email no existe y/o la contraseña no se corresponde con el email el usuario no queda autenticado.
Actores	Usuario registrado, administrador
Flujo del evento	1. El visitante selecciona la opción de “Login” en el menú de navegación del portal. 2. Se redirecciona a una nueva página del portal, donde se muestra un panel dando al visitante la opción de introducir su email, su contraseña y enviar datos. 3. El usuario introduce su email y su contraseña en los campos correspondientes habilitados para ello y selecciona la opción de enviar datos, con dos posibles resultados: 3.1. El email y su contraseña asociado son correctos y por tanto el usuario queda autenticado correctamente con su rol correspondiente.

	3.2. El email o su contraseña no son correctos, por lo que se informa al visitante y este no queda autenticado.

Registro (CU-UR-02)	
Entrada	Email. Nick. Nombre y apellidos. Contraseña. Otros datos.
Descripción	Permite a un usuario registrarse en el sistema con los datos especificados por el mismo.
Dependencias	N/a
Precondición	Se introducen todos los datos correspondientes al registro. No se ha iniciado sesión.
Postcondición	• Éxito: Si los datos son correctos el usuario se registra y queda añadido en la BD. • Fracaso: Si los datos no son correctos el usuario no queda registrado.
Actores	Visitante
Flujo del evento	1. El visitante selecciona la opción de "Registro" en el menú de navegación del portal. 2. Se redirecciona a una nueva página del portal, donde se muestra un panel dando al visitante la opción de introducir los datos con los que se desea registrar en la aplicación. 3. El usuario introduce sus datos y una vez ha concluido selecciona la opción de enviar con dos posibles resultados: 3.1. Los datos introducidos por el usuario son correctos y por tanto queda registrado, desde ese preciso momento ya puede autenticarse en el portal mediante la opción de Login. 3.2. Alguno de los datos no es correcto, el sistema informa al usuario de qué datos no son correctos y el motivo de ello. Hasta que todos los datos no sean correctos el visitante no podrá registrarse.

Listar visitas asociadas a usuario (CU-UR-03)	
Entrada	N/a.
Descripción	Lista todas las visitas asociadas al usuario autenticado en ese momento en el sistema.
Dependencias	CU-UR-01 con postcondición de éxito
Precondición	Existe al menos una visita asociada al usuario autenticado.
Postcondición	Se listan todas las visitas asociadas al usuario autenticado.
Actores	Usuario registrado
Flujo del evento	1. El usuario registrado selecciona la opción de "Mis Visitas" en el menú de navegación del portal. 2. El sistema comprueba los datos de sesión del usuario y si son correctos lista todas las visitas asociadas a ese usuario en una nueva página. 3. El usuario puede llevar a cabo diversas funciones sobre cada una de las visitas entre las que se incluyen: 3.1. Eliminar visita. 3.2. Modificar visita. 3.3. Recomendar visita. 3.4. Ver información de visita. A continuación se detallan estos casos de uso, para identificar que son dependientes del caso de uso de listar visita, se identificarán con un "Listar:" antes del nombre. Por lo que para todos los casos de uso que incluyan "Listar" antes se supondrá que ya se ha llevado a cabo dicha opción antes.

Listar : Eliminar visita (CU-UR-04)	
Entrada	Visita seleccionada.
Descripción	Elimina la visita especificada por el usuario registrado. Esta acción no se puede deshacer, por lo que se solicitará confirmación.
Dependencias	CU-UR-01 con postcondición de éxito CU-UR-03
Precondición	Existe al menos una visita asociada al usuario autenticado.

Postcondición	Se elimina la visita especificada de la BD.
Actores	Usuario registrado
Flujo del evento	1. El usuario registrado selecciona la opción de eliminar, representada por un de forma universal en toda la aplicación. 2. El sistema pide confirmación al usuario antes de eliminar la visita con dos posibles resultados: 2.1. El usuario confirma la operación, en cuyo caso la visita se elimina, y ya no estará disponible. 2.2. El usuario cancela la operación en cuyo caso no se verá afectada la visita.

Listar : Recomendar visita (CU-UR-05)	
Entrada	Visita seleccionada.
Descripción	El usuario autenticado puede recomendar una de sus visitas a otro usuario registrado del sistema, indicando su email.
Dependencias	CU-UR-01 con postcondición de éxito CU-UR-03
Precondición	Existe al menos una visita asociada al usuario autenticado.
Postcondición	La visita especificada queda asociada al usuario destino especificado como recomendada en la BD.
Actores	Usuario registrado
Flujo del evento	1. El usuario registrado selecciona la opción de recomendar, representada por un de forma universal en toda la aplicación. 2. Se abre un panel donde se da opción de especificar el email del usuario al que se quiere recomendar la visita. También ofrece la posibilidad de introducir un texto asociado a dicha recomendación. 3. El usuario introduce el email del destinatario de la recomendación con dos posibles resultados: 3.1. El email existe y se corresponde con el de un usuario registrado de la aplicación.

	3.1.1. En este caso se registra la nueva visita dentro de la tabla de visitas recomendadas del usuario especificado. 3.2. El email no existe, en cuyo caso se informa al usuario de que no hay coincidencias encontradas y no se recomendará la visita.

Listar : Ver información de visita (CU-UR-06)	
Entrada	Visita seleccionada.
Descripción	Permite ver toda la información asociada a la visita seleccionada.
Dependencias	CU-UR-01 con postcondición de éxito CU-UR-03
Precondición	Existe al menos una visita asociada al usuario autenticado.
Postcondición	Se muestra la información asociada a la visita seleccionada.
Actores	Usuario registrado
Flujo del evento	1. El usuario registrado selecciona la opción de ver información de visita. 2. Se abre un panel donde con toda la información asociada a dicha visita encontrada.

Listar : Modificar visita (CU-UR-07)	
Entrada	Visita seleccionada.
Descripción	Permite modificar determinados aspectos de la visita, como añadir o eliminar monumentos de esta.
Dependencias	CU-UR-01 con postcondición de éxito CU-UR-03
Precondición	Existe al menos una visita asociada al usuario autenticado.
Postcondición	Se modifica la visita seleccionada en la BD.
Actores	Usuario registrado
Flujo del evento	1. El usuario registrado selecciona la opción de ver información de visita. 2. Se abre un panel donde con toda la información asociada a dicha visita encontrada.

	3. El usuario puede entonces modificar ciertos aspectos de la visita, aunque no todos. El usuario, concretamente, puede: 3.1. Eliminar un lugar asociado a dicha visita, siempre que no se quede sin ningún lugar. 3.2. Añadir un nuevo lugar, aunque una visita no puede contener más de 10 lugares.

Crear una visita (CU-UR-08)	
Entrada	Datos asociados a la visita: • Título de la visita • Lugares que visitar • Descripción de la visita • Etiquetas asociadas a la visita • Fecha de la visita
Descripción	Permite crear una visita en el sistema con los datos proporcionados por el usuario registrado.
Dependencias	CU-UR-01 con postcondición de éxito
Precondición	Datos introducidos correctamente
Postcondición	Se inserta la nueva visita en la BD asociada al usuario autenticado.
Actores	Usuario registrado
Flujo del evento	1. El usuario registrado selecciona la opción crear una nueva visita, en la ventana de “Mis visitas” del portal. 2. Se abre un panel donde se solicita toda la información asociada a la visita que es necesaria para crearla. 3. El usuario introduce todos los datos asociados a la nueva visita, con dos posibles resultados: 3.1. Los datos introducidos son todos correctos, con lo que la visita queda guardada y se recargará la página, donde ya aparecerá la nueva visita. 3.2. Alguno o varios de los datos introducidos no son correctos, por lo que se indicará cuál o cuáles de los mismos no son correctos, y se dará opción de reintentar.

4.3.3. CASOS DE USO DEL ADMINISTRADOR

Buscar usuario (CU-A-01)	
Entrada	Texto con la clave a buscar y/u otras opciones.
Descripción	Permite buscar a un usuario en base al texto proporcionado, la búsqueda se hará por coincidencia del texto con el email o el Nick del usuario.
Dependencias	CU-UR-01 con postcondición de éxito
Precondición	Texto introducido
Postcondición	Se listan todos los usuarios cuyo Nick o email se corresponde con el texto introducido.
Actores	Administrador
Flujo del evento	1. El administrador accede a la ventana de "Administración" del menú de navegación del portal. 2. El administrador introduce la cadena de texto que desea buscar en un cuadro de texto habilitado para ello. 3. El sistema busca entre todos los usuarios con dos posibles resultados: 3.1. Hay coincidencias: se muestran los usuarios con coincidencias encontradas. 3.2. No hay coincidencias: no se muestra nada y se informa de ello.

Listar usuarios (CU-A-02)	
Entrada	N/a.
Descripción	Lista todos los usuarios registrados en el sistema.
Dependencias	CU-UR-01 con postcondición de éxito
Precondición	Existe al menos un usuario registrado.
Postcondición	Se listan todos los usuarios registrados en la BD.
Actores	Administrador
Flujo del evento	1. El administrador accede a la ventana de "Administración" del menú de navegación del portal.

	2. El administrador selecciona la opción de listar usuarios. 3. El sistema entonces listará en pantalla todos los usuarios asociados al mismo.

Ver información de un usuario (CU-A-03)	
Entrada	Usuario seleccionado
Descripción	Permite ver la información asociada a un usuario seleccionado.
Dependencias	CU-UR-01 con postcondición de éxito CU-A-01 o CU-A-02
Precondición	Se ha seleccionado un usuario.
Postcondición	Se muestra la información asociada a ese usuario.
Actores	Administrador
Flujo del evento	1. El administrador accede a la ventana de "Administración" del menú de navegación del portal. 2. El administrador busca un usuario manualmente o bien lista todos los usuarios. 3. El administrador selecciona entonces el usuario que desea que se rehabilite y lo selecciona. 4. Se abre un panel con información del usuario, incluyendo toda la información relevante relativa a dicho usuario.

Deshabilitar usuarios (CU-A-04)	
Entrada	Usuario seleccionado
Descripción	Permite deshabilitar del sistema a un usuario seleccionado por el administrador.
Dependencias	CU-UR-01 con postcondición de éxito CU-A-01 o CU-A-02 CU-A-03
Precondición	Se ha seleccionado un usuario. El usuario no está ya deshabilitado
Postcondición	Se deshabilita ese usuario de la BD.
Actores	Administrador

Flujo del evento	1. El administrador accede a la ventana de "Administración" del menú de navegación del portal. 2. El administrador busca un usuario manualmente o bien lista todos los usuarios. 3. El administrador selecciona entonces el usuario que desea que se elimine y lo selecciona. 4. Se abre un panel con información del usuario, incluyendo una opción para eliminar/deshabilitar usuarios. 5. El administrador pulsa la opción eliminar/deshabilitar usuario. 6. El sistema solicita entonces confirmación al administrador con dos posibles resultados: 6.1. El administrador confirma la eliminación, con lo que se establece como "deshabilitado" al usuario seleccionado, y a todos los efectos quedará excluido del sistema, aunque permanecerá en él. (en la lista de usuarios, este aparecerá diferenciado de los demás). 6.2. El administrador cancela la operación, sin afectar al sistema ni a los usuarios.

Rehabilitar usuarios (CU-A-05)	
Entrada	Usuario seleccionado
Descripción	Permite rehabilitar en el sistema a un usuario previamente deshabilitado en el sistema.
Dependencias	CU-UR-01 con postcondición de éxito CU-A-01 ó CU-A-02 CU-A-03
Precondición	Se ha seleccionado un usuario. El usuario no está ya habilitado
Postcondición	Se rehabilita el usuario seleccionado en la BD.
Actores	Administrador
Flujo del evento	1. El administrador accede a la ventana de "Administración" del menú de navegación del portal.

	2. El administrador busca un usuario manualmente o bien lista todos los usuarios. 3. El administrador selecciona entonces el usuario que desea que se rehabilite y lo selecciona. 4. Se abre un panel con información del usuario, incluyendo una opción para rehabilitar usuarios. 5. El administrador pulsa la opción eliminar/deshabilitar usuario. 6. El sistema solicita entonces confirmación al administrador con dos posibles resultados: 6.1. El administrador confirma la rehabilitación, con lo que se establece como "habilitado" al usuario seleccionado, y a todos los efectos quedará incluido en el sistema, como si nunca hubiera sido eliminado. 6.2. El administrador cancela la operación, sin afectar al sistema ni a los usuarios.

Eliminar comentario de visita (CU-A-06)	
Entrada	Usuario seleccionado
Descripción	Permite eliminar un comentario asociado a una visita.
Dependencias	
Precondición	
Postcondición	
Actores	Administrador
Flujo del evento	1. El administrador accede a la ventana de "Administración" del menú de navegación del portal. 2. El administrador busca un usuario manualmente o bien lista todos los usuarios. 3. El administrador selecciona entonces el usuario que desea que se rehabilite y lo selecciona. 4. Se abre un panel con información del usuario, incluyendo una opción para rehabilitar usuarios.

	5. El administrador pulsa la opción eliminar/deshabilitar usuario. 6. El sistema solicita entonces confirmación al administrador con dos posibles resultados: 6.1. El administrador confirma la rehabilitación, con lo que se establece como "habilitado" al usuario seleccionado, y a todos los efectos quedará incluido en el sistema, como si nunca hubiera sido eliminado. 6.2. El administrador cancela la operación, sin afectar al sistema ni a los usuarios.

Eliminar visita de usuario (CU-A-07)	
Entrada	Usuario seleccionado
Descripción	Permite eliminar una visita concreta de un usuario concreto del sistema.
Dependencias	
Precondición	
Postcondición	
Actores	Administrador
Flujo del evento	1. El administrador accede a la ventana de "Administración" del menú de navegación del portal. 2. El administrador busca un usuario manualmente o bien lista todos los usuarios. 3. El administrador selecciona entonces el usuario que desea que se rehabilite y lo selecciona. 4. Se abre un panel con información del usuario, incluyendo una opción para rehabilitar usuarios. 5. El administrador pulsa la opción eliminar/deshabilitar usuario. 6. El sistema solicita entonces confirmación al administrador con dos posibles resultados:

	6.1. El administrador confirma la rehabilitación, con lo que se establece como "habilitado" al usuario seleccionado, y a todos los efectos quedará incluido en el sistema, como si nunca hubiera sido eliminado. 6.2. El administrador cancela la operación, sin afectar al sistema ni a los usuarios.

Ver estadísticas (CU-A-08)	
Entrada	Usuario seleccionado
Dependencias	
Precondición	
Postcondición	
Actores	Administrador
Flujo del evento	1. El administrador accede a la ventana de "Administración" del menú de navegación del portal. 2. El administrador busca un usuario manualmente o bien lista todos los usuarios. 3. El administrador selecciona entonces el usuario que desea que se rehabilite y lo selecciona. 4. Se abre un panel con información del usuario, incluyendo una opción para rehabilitar usuarios. 5. El administrador pulsa la opción eliminar/deshabilitar usuario. 6. El sistema solicita entonces confirmación al administrador con dos posibles resultados: 6.1. El administrador confirma la rehabilitación, con lo que se establece como "habilitado" al usuario seleccionado, y a todos los efectos quedará incluido en el sistema, como si nunca hubiera sido eliminado. 6.2. El administrador cancela la operación, sin afectar al sistema ni a los usuarios.

5. MODELO DE DATOS

A continuación, se detallan los distintos elementos que componen la base de datos. Para ello se mostrará:

- El modelo entidad-relación de la información que se necesita almacenar. En este diagrama se muestran las principales entidades de información que son necesarias gestionar y las relaciones que existen entre ellas.

- La estructura de la BD obtenida a partir del diagrama de entidad-relación.

- El modelo detallado de cada una de las tablas en la BD, donde se describen los diferentes atributos y las relaciones entre las tablas.

5.1. MODELO ENTIDAD-RELACIÓN

En la figura 5.1 se muestra el modelo entidad-relación donde se representan las entidades de las que es necesario almacenar información y cómo se encuentran relacionadas:

- Usuarios: se trata de los usuarios del sistema. Estos pueden tener asociadas visitas.

- Visitas: están compuestas por un conjunto de monumentos, y se relacionan con un único usuario.

- Monumentos: representan los monumentos de interés de los que se quiere gestionar información.

- Comentarios: son textos escritos por usuarios registrados o visitantes que están asociados a una única visita.

- Sesiones: representan las sesiones de los usuarios. No mantienen relación con ninguna otra entidad.

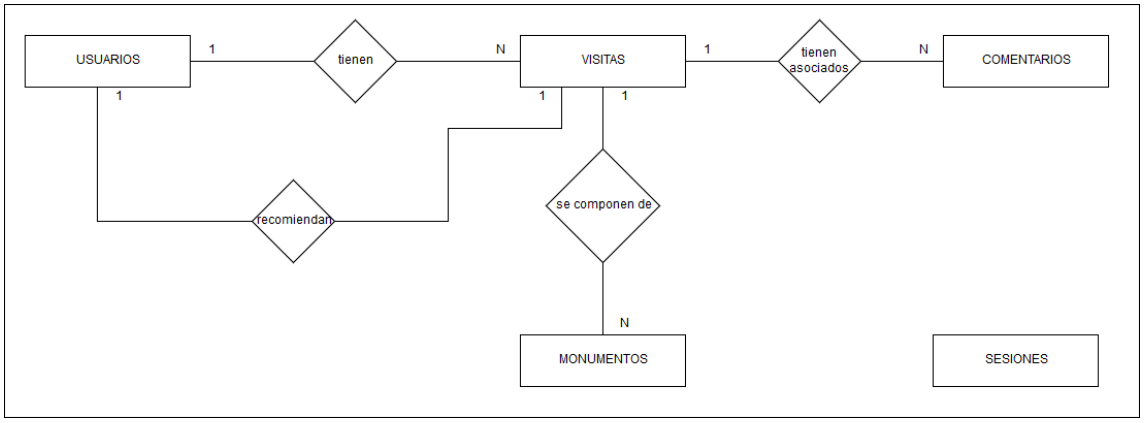


Figura 5.1. Modelo entidad-relación de la información.

5.2.ESTRUCTURA DE LA BASE DE DATOS.

El modelo entidad-relación descrito se ha implementado mediante una base de datos relacional. A continuación, en la figura 5.2 se muestran las tablas y las relaciones que existen entre ellas.

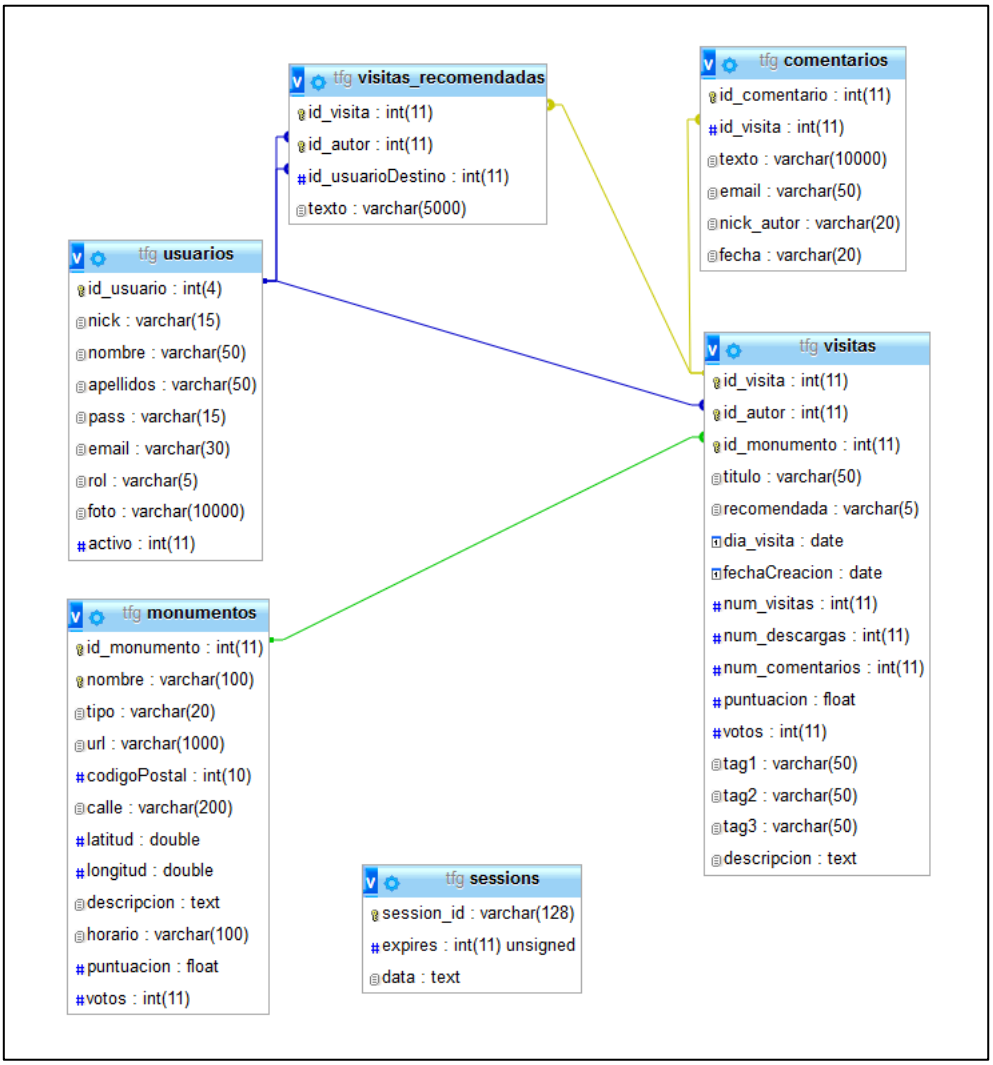


Figura 5.2. Tablas y relaciones de la base de datos relacional.

5.3.ESTRUCTURA DE LAS TABLAS DE LA BD

En esta sección se van a describir la estructura de las tablas que existen en la base de datos y las relaciones que mantienen.

5.3.1. TABLA USUARIOS

Esta tabla contiene toda la información relevante de los usuarios del sistema.

Usuarios

Id_usuario	int	Identificador del usuario.
Nick	Varchar (50)	Nick con el que aparecerá visible el usuario.
Nombre	Varchar (50)	Nombre del usuario.
Apellidos	Varchar (50)	Apellidos del usuario.
Email	Varchar (50)	Email del usuario.
Pass	Varchar (15)	Contraseña del usuario.
Rol	Varchar (5)	Establece el rol del usuario. Puede ser usuario o administrador.
Activo	int	Valor binario que determina si un usuario está activo o no en el sistema. • 0: El usuario no está activo • 1: El usuario está activo

5.3.2. TABLA VISITAS

Esta tabla contiene toda la información relevante a las visitas del sistema. Se encuentra relacionada con la tabla Usuarios mediante el atributo id_autor que relaciona los identificadores de los usuarios. También se relaciona con la tabla Monumentos a través del atributo id_monumento que hace referencia a los ids de los monumentos.

Visitas		
Id_visita	Int	Identificador de las visitas.
Id_autor	Int	Identificador del usuario.
Id_monumento	Int	Identificador del monumento.
titulo	Varchar (50)	Título de la visita.
Dia_visita	date	Día establecido para la realización de la visita, en caso de que se desee especificar uno.
fechaCreacion	date	Fecha en que se creó la visita.
Num_visitas	Int	Cantidad de veces que se ha visualizado la visita.
Num_descargas	Int	Cantidad de veces que se ha descargado la visita.
Num_comentarios	Int	Cantidad de veces que se ha comentado la visita.
Puntuación	Float	Puntuación actual de la visita.
Votos	Int	Cantidad de veces que se ha votado la visita.

Tag1	Varchar (50)	Etiqueta 1, se utiliza para las búsquedas.
Tag2	Varchar (50)	Etiqueta 2, se utiliza para las búsquedas.
Tag3	Varchar (50)	Etiqueta 3, se utiliza para las búsquedas.
Descripción	Text	Descripción de la visita. Se utilizará para hacer más efectivas las búsquedas de una visita concreta.

5.3.3. TABLA MONUMENTOS

Esta tabla contiene toda la información sobre los monumentos gestionados por el sistema. Los datos de esta tabla se obtienen a partir de las llamadas a la API de datos abiertos del Ayuntamiento de Madrid.

Monumentos		
Id_monumento	Int	Identificador del monumento.
Nombre	Varchar(50)	Título del monumento.
Tipo	date	Tipo del monumento. Se utilizará para aplicar ciertos tipos de filtros. Puede ser temploC, edificioM o Museo.
url	date	Url a la web que contiene información adicional sobre el monumento.
codigoPostal	Int	Código postal del lugar donde se encuentra el monumento.
Calle	Int	Calle y número donde se encuentra el monumento.
latitud	Int	Latitud del punto donde se encuentra el monumento.
longitud	Float	Longitud del punto donde se encuentra el monumento.
horario	Int	Horario de apertura del monumento, en caso de haberlo.
Puntuación	Varchar(50)	Puntuación actual del monumento.
Votos	Varchar(50)	Cantidad de veces que se ha votado el monumento.
Descripción	Text	Descripción del monumento.

5.3.4. TABLA VISITAS RECOMENDADAS

Esta tabla relaciona las tablas de usuarios y visitas de manera que se tiene identificada qué visita ha sido recomendada a qué usuario y por quién. Se utilizan los identificadores para establecer las relaciones existentes.

Visitas_recomendadas		
Id_visita	Int	Identificador de la visita.
Id_autor	Int	Identificador del usuario que recomienda la visita.
Id_usuarioDestino	Int	Identificador del usuario que recibe la recomendación.
Texto	Text	Texto que adjunta el usuario que recomienda una visita al destino de la misma.

5.3.5. TABLA COMENTARIOS

Esta tabla contiene los comentarios que se han realizado sobre una visita concreta. Los comentarios están relacionados con las visitas mediante los ids de la visitas.

Visitas_recomendadas		
Id_comentario	Int	Identificador del comentario.
Id_visita	Int	Identificador de la visita a la que está referida el comentario.
Texto	Text	Texto del usuario donde recomienda una visita a un destino concreto.
Email	Varchar (50)	Email del usuario o visitante que comenta la visita. Es opcional para visitantes.
Nick_autor	Varchar (20)	Nick del usuario o visitante que comenta la visita. Es opcional para visitantes
Fecha	Varchar (20)	Fecha en que se creó el comentario

5.3.6. TABLA SESSIONS

Esta tabla contiene información asociada a las sesiones activas en cada momento en el servidor. Su contenido es generado automáticamente por el servidor:

Visitas_recomendadas		
session_id	Varchar(128)	Identificador de la sesión.
expires	Int	Indica cuando expirará la sesión
data	Text	Contenido de datos asociado a la sesión.

6. ARQUITECTURA DE LA APLICACIÓN

La aplicación se ha desarrollado utilizando el marco de aplicaciones Express, y se ha modelado como una arquitectura cliente/servidor (Gackenheimer, 2013) donde se distinguen principalmente tres componentes:

- Cliente: lleva a cabo las peticiones al servidor.
- Servidor: recibe y procesa las peticiones del cliente.
- Base de datos: recibe consultas para gestionar la información del sistema.

En la figura 6.1 se muestra un esquema de la arquitectura empleada:

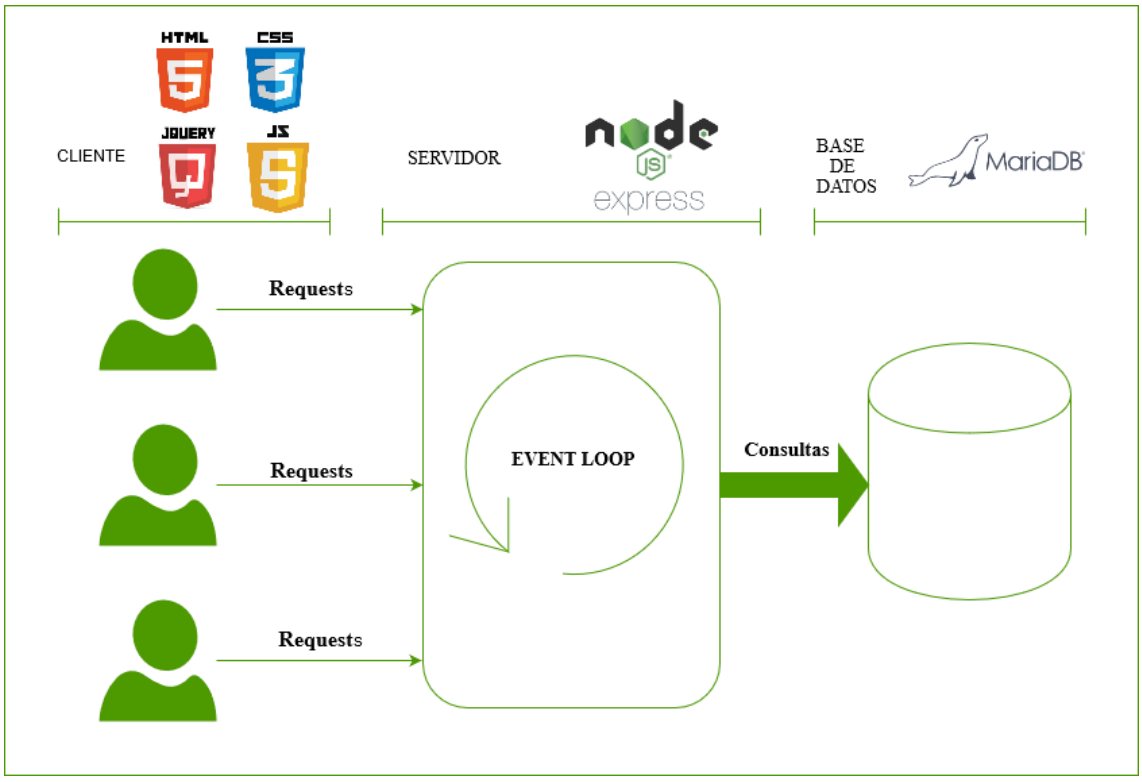


Figura 6.1. Arquitectura de la aplicación

El cliente de la aplicación se ha diseñado en base a plantillas ejs[19] cuya única diferencia con las plantillas tradicionales HTML es la posibilidad de trabajar de forma más eficiente

[19]Página web al portal web oficial del módulo de plantillas ejs https://ejs.co/

con Node, además de permitir insertar código JavaScript de forma mucho más sencilla bajo la sintaxis <% %> y otras variaciones.

El código utilizado para las plantillas ejs es HTML5, que se encarga de definir las estructuras de la página y código CSS3, que se encarga de dar estilos a dichas estructuras. En este sentido para facilitar el desarrollo del código CSS3 se ha incluido Bootstrap, cuyos estilos se utilizan prácticamente en todas las plantillas, de modo que el css propio tan sólo se utiliza para correcciones o pequeñas modificaciones del código de Bootstrap.

Para facilitar la labor del servidor se ha incluido Javascript (junto con JQuery y Ajax) en el cliente, liberando así al servidor de gran parte de la carga de la aplicación y delegando en el cliente algunas funciones tales como:

- Validaciones de formularios: Se utilizan JavaScript y JQuery para validar los formularios que aparecen a lo largo de la aplicación.

- Llamadas al servidor: Se utiliza Ajax para realizar llamadas al servidor de forma dinámica e interactiva sin necesidad de recargar el cliente y permitiendo así una mayor sensación de fluidez y rapidez en el cliente.

- Construcción de código dinámica: En gran parte del diseño de la aplicación se utiliza este tipo de codificación donde mediante una llamada Ajax al servidor se obtienen los datos. Para evitar hacer una recarga de la página, se insertan de forma dinámica los elementos HTML junto con sus estilos asociados a la página utilizando para ello JQuery.

- Llamadas a APIS: Para mostrar cierta información adicional referente a algunos monumentos o visitas, se realizan llamadas a APIS de datos externas. En este sentido, para evitar recargar excesivamente al servidor es el cliente quien las realiza directamente.

En la figura 6.2 se resume la estructura del cliente, donde se ven todos los actores que intervienen en la construcción de las plantillas ejs.

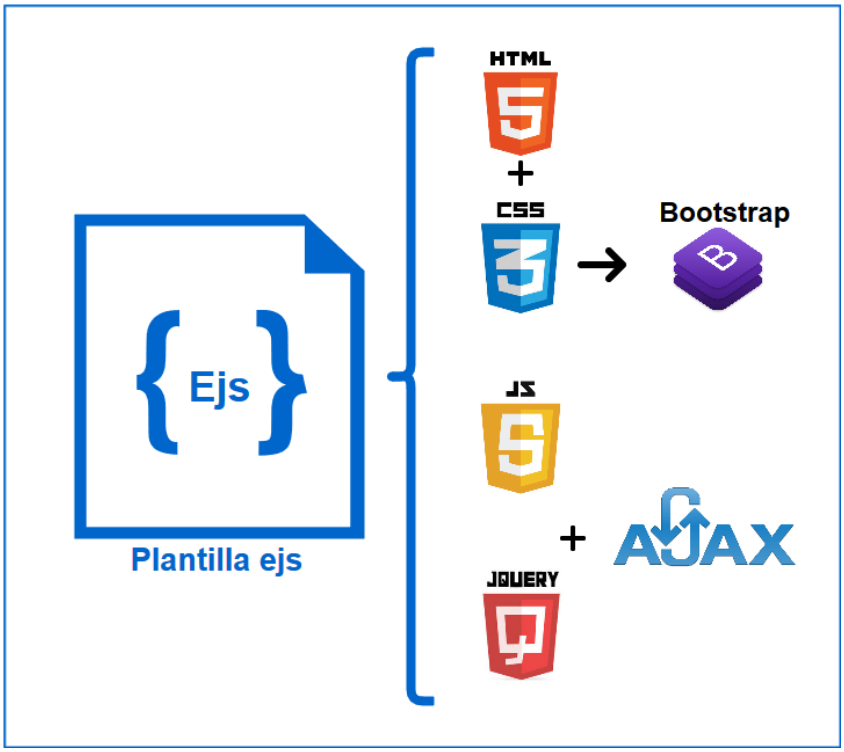


Figura 6.2. Arquitectura de las plantillas ejs utilizadas en el cliente

7. DISEÑO DE LA APLICACIÓN

En este capítulo, se muestra el diseño e implementación de la aplicación. No se va a mostrar la aplicación entera, sino que se ha elegido un caso de uso representativo. Concretamente se va a analizar el caso de uso crear visita.

El cliente solicita acceso para crear una visita desde el portal de la aplicación y se le muestra un formulario (Figura 7.1).

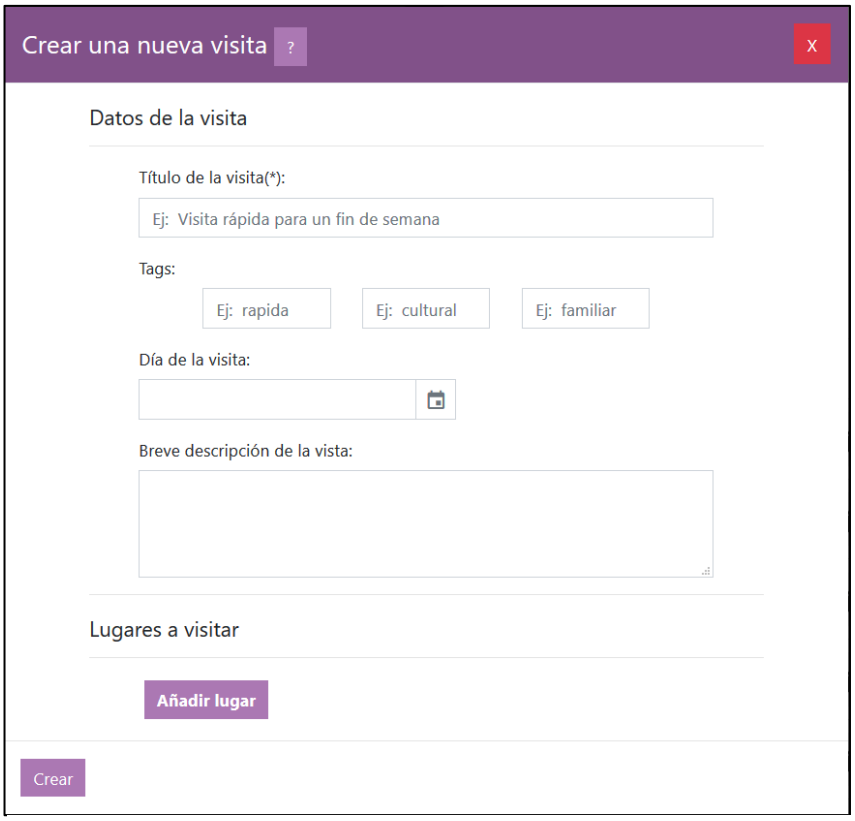


Figura 7.1. Formulario para crear una nueva visita.

Una vez que el usuario ha rellenado todos los campos y ha seleccionado la opción de crear, entonces se realizan las comprobaciones en el lado del cliente para asegurar que los campos han sido cumplimentados debidamente tal como se muestra en la Figura 7.2.

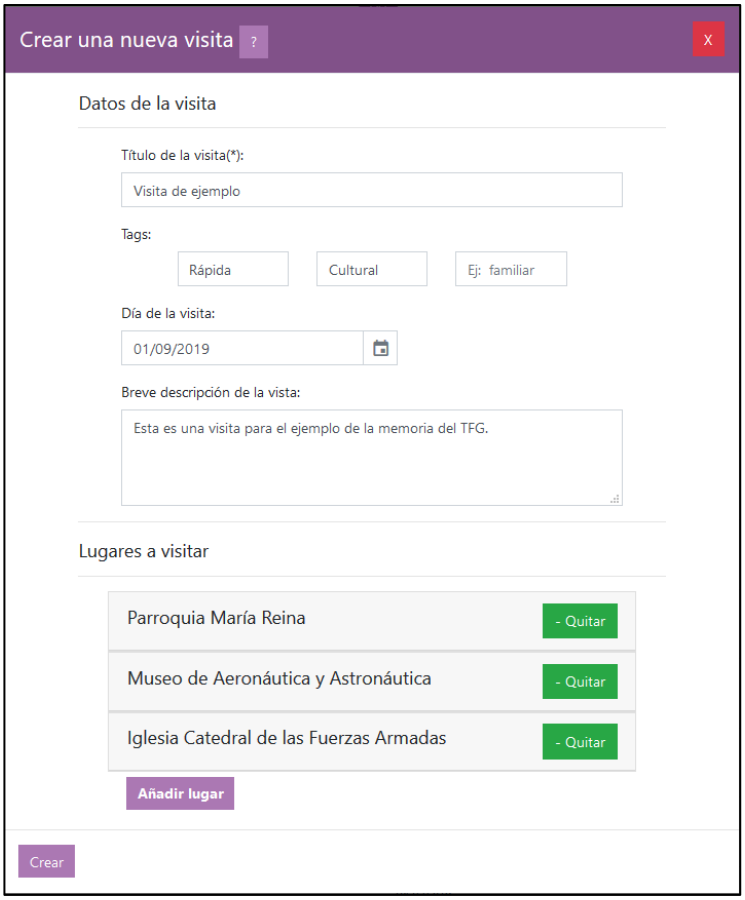


Figura 7.2. Creación de una nueva visita.

Las comprobaciones se hacen primero en el cliente usando la función validarVisita() que se muestra en la Figura 7.3 . La función se encuentra definida en el script misVisitas.js asociado a la plantilla para la ventana Mis Visitas del portal.

```
function validarVisita(){

	var titulo = document.getElementsByName("tituloVisita")[0].value;
	var descripcion = document.getElementsByName("breveDesc")[0].value;
	var validar = true;

	$('.alert').remove();

	//Validaciones del título:
	if(titulo == ""){
		$('#tituloVisita').after('<p class="alert alert-danger">Debes
introducir un título válido.</p>');
		validar = false;
	} else if (titulo.length > 40){
		$('#tituloVisita').after('<p class="alert alert-danger">El
título es demasiado largo.</p>');
		validar = false;
	};

	//Validaciones de lugares:
	if (lugares.length < 1){
```

```
            $('#accordion2').after('<p class="alert alert-danger">Debes
introducir al menos un lugar a visitar.</p>');
            validar = false;
      } else if (lugares.length > 9){
            $('#accordion2').after('<p class="alert alert-danger">Has
introducido demasiados lugares.</p>');
            validar = false;
      }

      //Validaciones de descripcion
      if (descripcion.length < 20){
            $('#breveDesc').after('<p class="alert alert-danger">Introduce
una descripción.</p>');
            validar = false;
      }

      if(!validar){
            $('#botonCrear').after('<p class="alert alert-danger">No has
rellenado alguno de los campos clave</p>');
      } else{
            crearVisita();
      }

}
```

Figura 7.3. Función validarVisita()

Si se cumplen todos los requisitos exigidos para poder crear la visita, se accede a la función crearVisita(), dentro del mismo script:

```
function crearVisita(){
      var museos = [];

      $('.listaMuseos').each(function () {
        museos.push($(this).attr('id'));
    });
      var tituloVisita =
document.getElementsByName("tituloVisita")[0].value;
      var tag1 = document.getElementsByName("tag1")[0].value;
      var tag2 = document.getElementsByName("tag2")[0].value;
      var tag3 = document.getElementsByName("tag3")[0].value;
      var desc = document.getElementsByName("breveDesc")[0].value;

      $.ajax({
        type: "POST",
        url:  "/crearVisita",
        data: {
            museos : museos,
            tituloVisita : tituloVisita,
            tag1 : tag1,
            tag2 : tag2,
            tag3 : tag3,
            desc : desc
        },

        success: function (data, textStatus, jqXHR) {
```

```
                window.location.reload();
            },
            error: function(data, textStatus, jqXHR) {
                alert("Error al crear visita");
            }

        });

}
```

Figura 7.4. –Función crearVisita()

La función realiza una llamada Ajax tipo POST al servidor para insertar la visita en la BD. Esta llamada será recibida en el servidor por el manejador de ruta correspondiente, que se corresponde con el código mostrado en la Figura 7.5.

```
/*
    Crea una visita asociada al usuario logeado en ese momento.
    Los datos de la visita se envía a través de una petición ajax.
*/
app.post("/crearVisita", function(request, response) {
    if(typeof request.session.logueado != 'undefined'){
        var date = new Date();
        var current_hour = date.getHours();

        var museos = request.body.museos;
        var datosVisita = {
            titulo: request.body.tituloVisita,
            id_visita:0,
            tag1: request.body.tag1,
            tag2: request.body.tag2,
            tag3: request.body.tag3,
            breveDesc : request.body.desc,
            recomendada: "no",
            diaVisita: date,
            fechaCreacion: date,
            id_autor: request.session.userData.id_usuario
        };

        //Primero consultamos que el nombre de la visita a crear no exista
ya:
        nCon.buscarTituloVisita(datosVisita.titulo, function(err, result){
            if (!result) {
                nCon.obtenerIDvalidoVisita(function(err, result){
                    //Para cada museo, hacemos un insert a la BBDD:

                    if(result[0].idvalido != undefined)
                        datosVisita.id_visita = result[0].idvalido;

                    museos.forEach(function(museo){
                        nCon.insertarVisita(datosVisita, museo,
function(err, result){
                            if(err){
                                console.log(err);
                            }
```

```
                        });
                    });
                });
                response.redirect("misVisitas");
            } else {
                response.redirect("misVisitas");
                response.end();
            }
        });
    } else{
        response.status(200);
        response.render("index");
    }

});
```

Figura 7.5. Archivo app.js.

El código del servidor llevará a cabo una serie de comprobaciones adicionales, y si todas ellas se cumplen, se llamará al script encargado de realizar las consultas contra la BD tal como se muestra en la Figura 7.6:

```
insertarVisita(data, museo, callback) {
      this.pool.getConnection((err, connection) => {
            if (err) {
                  //connection.release();
                  callback(err);
            } else {
                  connection.query("INSERT INTO visitas2(id_visita,
id_autor, id_monumento, titulo, recomendada, dia_visita, fechaCreacion,
num_visitas, num_descargas, tag1, tag2, tag3, descripcion) VALUES
(?,?,?,?,?,?,?,0,0,?,?,?,?)",
                  [data.id_visita, data.id_autor, museo, data.titulo,
data.recomendada, data.diaVisita, data.fechaCreacion, data.tag1, data.tag2,
data.tag3, data.breveDesc], (err, result) => {
                        connection.release();
                        if (err) {
                              callback(err);
                        } else {
                              if(result.length === 0) {
                                    callback(null);
                              } else {
                                    callback(null, result);
                              }
                        }
                  });
            }
      });

}
```

Figura 7.6. Función insertarVisita()

La consulta insertará en la BD la información recibida de forma parametrizada, evitando así posibles riesgos de seguridad como ataques por inyección SQL entre otros. Una vez realizada la inserción si no se ha producido ningún fallo, se informará al servidor tal como se muestra en la captura de código de la Figura 7.7:

```
response.status(200);
response.redirect("misVisitas");
response.end();
```

Figura 7.7. Respuesta en caso de éxito de la operación.

La respuesta será recibida por el callback de la llamada Ajax ejecutándose success o error según el resultado de la llamada. Se muestra en la Figura 7.8.

```
success: function (data, textStatus, jqXHR) {
       window.location.reload();
},
error: function(data, textStatus, jqXHR) {
       alert("Error al crear visita");
}
```

Figura 7.6. Manejador de respuestas de la llamada AJAX al servidor.

En caso de éxito se recargará la página mostrando ya la visita creada y en caso de error se informará de ello. En la Figura 7.9 se muestra el resultado de una llamada con éxito donde se puede ver que aparece la visita que se había creado con el nombre “Visita de ejemplo”.

Figura 7.7. Pantalla de Mis visitas mostrando el resultado exitoso de la operación.

.

8. CONCLUSIONES Y TRABAJO FUTURO

A continuación, se muestran las conclusiones extraídas del TFG y las líneas de trabajo futuro que permitirían extender la funcionalidad del trabajo presentado.

8.1. CONCLUSIONES

El uso de fuentes de datos abiertos proporciona una forma muy útil de crear aplicaciones que ofrezcan servicios exclusivos simplemente utilizando los datos de forma creativa. De este modo se puede crear una aplicación con una extensa BD cuyo mantenimiento y actualización corre a cargo de un tercero, permitiendo así que todo el esfuerzo se focalice en el desarrollo en sí.

El trabajo llevado a cabo en este TFG es un buen ejemplo de ello, ya que recuperando información de una base de datos abierta se ha podido crear una aplicación alrededor de dicha información. Lo cual da una idea del gran potencial de esta tecnología y sus muchas aplicaciones.

Para poder realizar este trabajo se ha implementado una aplicación web usando la tecnología Node, por lo que se ha tenido que investigar y aprender sobre esta tecnología. De la misma manera han sido analizadas las tecnologías de Bootstrap, Github o Javascript.

Una vez desarrollada la aplicación se ha podido comprobar el potencial y alcance de utilizar datos abiertos, ya que sin haber tenido que introducir ni un solo dato en una base de datos se tiene acceso a una inmensa cantidad de datos sobre los distintos monumentos turísticos de la ciudad de Madrid, que de otro modo habría sido casi imposible conseguir.

8.2. TRABAJO FUTURO

Las principales líneas de trabajo futuro que podrían seguirse para amplicar el alcance del presente trabajo son las siguientes:

- Desarrollar el cliente utilizando un framework destinado expresamente a ello como son Angular o React. Esto facilitaría el mantenimiento del sitio web.

- Incluir fuentes de datos abiertos adicionales que proporcionen aún más información sobre los monumentos y complementen así las visitas.

- Incluir seguridad adicional en la aplicación como cambiar al protocolo HTTPS o incluir librerias adicionales que mejoren el cifrado de datos y así garantizar mayor seguridad de los datos que se almacenan de los usuarios.

- Realizar una evaluación de prueba con potenciales usuarios de la aplicación, para detectar posibles fallos de diseño o ampliar su alcance en base a las conclusiones de la evaluación.

8. CONCLUSIONS AND FUTURE WORK

Below is shown the conclusions drawn from the Final Degree Project (FDP) and the lines of future work that would allow extending the functionality of the presented work.

8.1. CONCLUSIONS

The use of open data sources provides a very useful way to create applications that offer exclusive services simply by using the data creatively. This way you can create applications with an extensive database whose maintenance and updating is done by a third party, thus allowing all the effort to focus on the development itself.

The work carried out in this FDP is a good example of this, since by retrieving information from an open database it has been possible to create an application around said information. Which gives an idea of the great potential of this technology and its many applications.

In order to carry out this work, a web application using Node technology has been implemented, so it has been necessary to investigate and learn about this technology. In the same way the Bootstrap, Github or Javascript technologies were analyzed.

Once the application has been developed, it is possible to verify the potential and scope of using open data, since without having to introduce a single data in a database, there is access to an immense amount of data on the different tourist monuments of the city of Madrid, which otherwise would have been almost impossible to achieve.

8.2. FUTURE WORK

Throughout the development of the project for various reasons, mainly time and scope, there were discarded various technologies and fields that would have been interesting to introduce such as:

- Developing the client using a framework specifically designed for it, such as Angular or React, would facilitate the maintenance of the website for the future.

- Include additional open data sources that provide even more information about the places and thus complement the visits.

- Include additional security in the application such as switching to the HTTPS protocol or include additional libraries that improve data encryption and thus ensure greater security of the data stored by users.

- Carry out a test trial with potential users of the application, to detect possible design flaws or expand its scope based on the conclusions of the trial.

ANEXO

ANEXO I: GUÍA DE INSTALACIÓN

Para poder instalar la aplicación de forma local en el sistema deseado, este debe cumplir con los siguientes requisitos:

- Node.js versión 8.12.0 o superior. La aplicación ha sido desarrollada íntegramente en la versión 8.12.0 de Node, aunque no obstante debería ser compatible con cualquier versión superior.

- XAMPP versión 3.2.2 o superior. La aplicación se ha desarrollado utilizando XAMPP para la gestión de servidores, en concreto del servidor Apache y del servidor MySQL que se encargan de la gestión de la BD.

- Navegador web compatible con JQuery y JavaScript.

Los pasos que hay que seguir para llevar a cabo la instalación son los siguientes:

1. Descargar el código del proyecto, que se encuentra en el enlace de GitHub de la aplicación: https://github.com/eduUcm/tfg. A continuación se descomprime el archivo en una carpeta.

2. Lanzar los servidores de Apache y MySQL. Para ello se abre el panel de control de XAMPP y se selecciona la opción “Start” en los módulos Apache y MySQL como se muestra en la figura 9.1

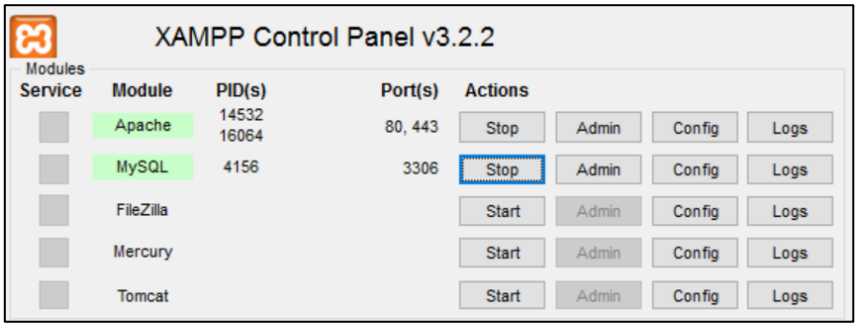


Figura 9.1. Panel de control de XAMPP

3. Instalar la base de datos SQL. Para ello se abre la aplicación PhpMyadmin que viene con XAMPP desde un navegador en la dirección: http://127.0.0.1/phpmyadmin/index.php.

   3.1. Se localiza el archivo correspondiente a la base de datos llamado tfg.sql.

   3.2. Se crea una base de datos llamada tfg y se importa el contenido del script anteriormente localizado.

4. Abrir la consola de comandos de Node (Node.js command prompt), la cual se genera automáticamente al instalar Node en la ubicación:

```
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Nodej.js
```

   Para acceder más fácilmente a ella simplemente se escribe node en el buscador de Windows:

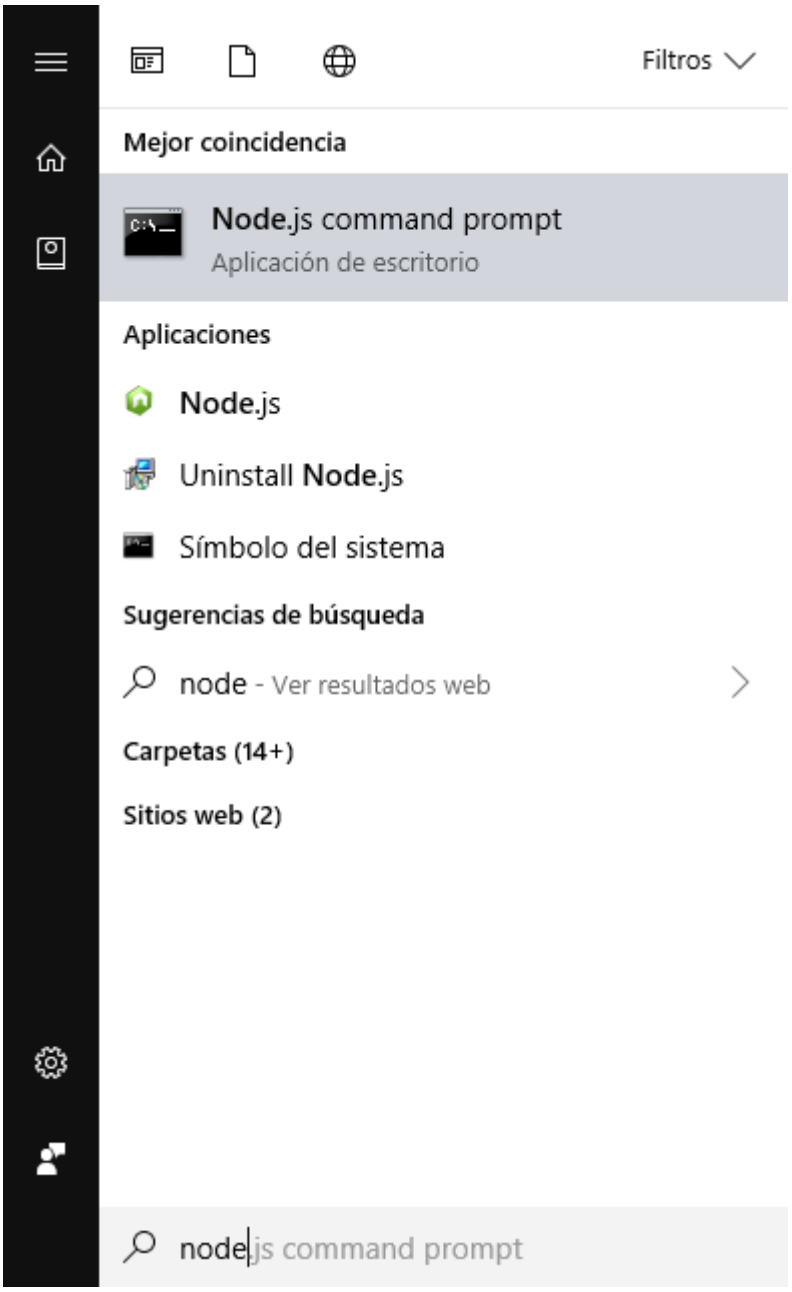


Figura 9.2. Buscando la consola de Node

5. Se localiza el proyecto que se ha descomprimido en el paso 1 desde la consola de Node.

6. Se ejecuta el script de instalación de dependencias, que instalará:

   - Todas las dependencias requeridas para el uso de la aplicación.

   - Nodemon, herramienta utilizada para lanzar y mantener el servidor lanzado.

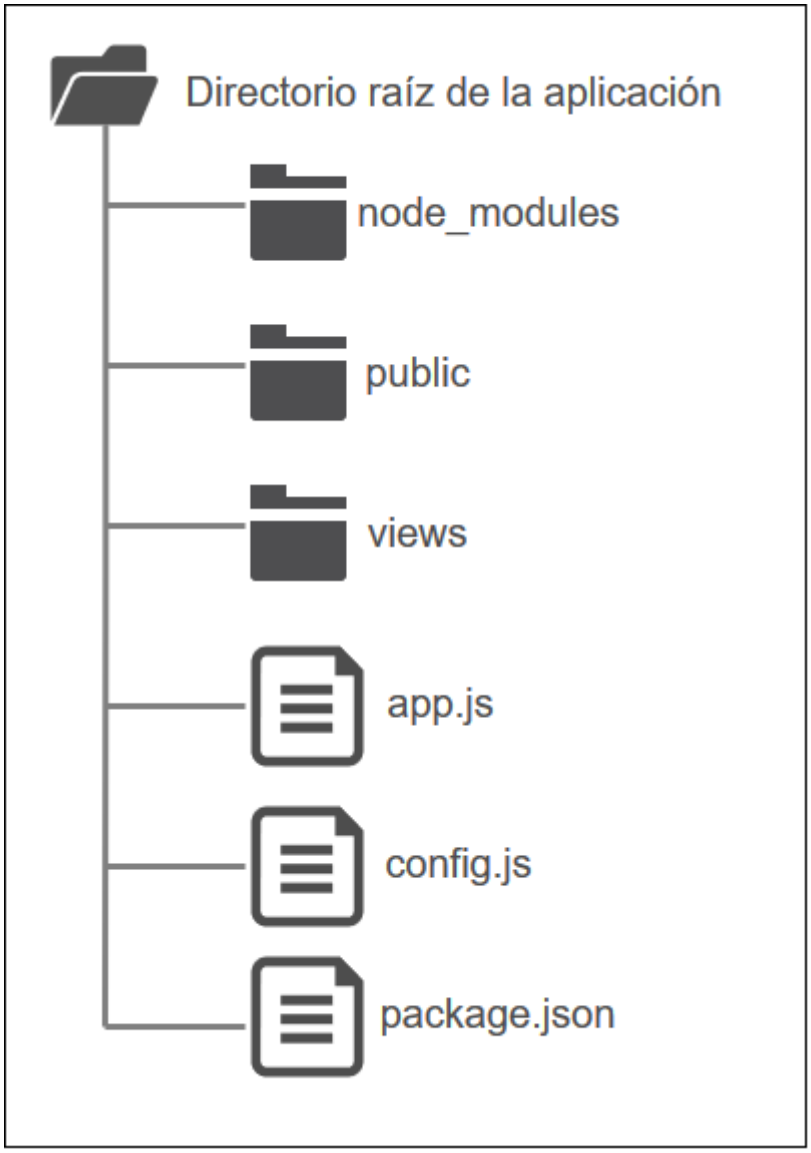
```
npm run pre-install
```

7. Se ejecuta el script de inicio de la aplicación:

```
npm run start
```

8. Ya se puede acceder a la aplicación localmente desde un navegador, accediendo a la dirección: http://localhost:3000/

La estructura utilizada por Node para crear la aplicación es una arquitectura más o menos fija que permite poca libertad a la hora de elegir dónde colocar los distintos componentes, y que obliga a la existencia de algunos elementos imprescindibles, que en la siguiente figura se muestran:



- **Node_modules**: En este directorio se guardan todos los módulos que utiliza nuestra aplicación y todos sus archivos correspondientes. No hace falta insertarlos manualmente ya que la instrucción `npm install --save` ejecutada en la raíz de nuestra aplicación ya los guarda directamente en su lugar correspondiente.

- **Public**: Este directorio contiene casi todos los archivos y recursos estáticos de la aplicación y que serán descargados cuando sean necesarios por el navegador del cliente. Aquí se guardan imágenes, archivos css, archivos js, plantillas y cualquier recurso estático en general. También pueden guardarse las librerías externas a la aplicación y que por razones de eficiencia queramos que estén físicamente en nuestro servidor.

- **Views**: Esta carpeta es típica del modelo ejs de express. Aquí se guardan todas las plantillas ejs de la aplicación.

- **App.js**: Este es el archivo principal del servidor, es al que se llama para iniciar la aplicación y contiene todas las llamadas necesarias al resto de recursos para hacer funcionar la aplicación.

- **Config.js**: Este es un archivo de configuración importante que la contiene empaquetada y es importado por App.js para configurar el servidor correctamente.

- **Package.json**: Este archivo es vital para el funcionamiento de una aplicación Node y contiene información muy importante para que este puede instalarse y funcionar, como por ejemplo, scripts de instalación o lanzamiento de la aplicación, dependencias de la aplicación, configuración con repositorios, etc.

ANEXO II: GUÍA DE UTILIZACIÓN

A continuación se dictan una serie de pautas para el correcto uso de la aplicación una vez instalada.

- El menú de navegación, situado en la parte superior de la pantalla, nos permite ir de una pestaña a otra:

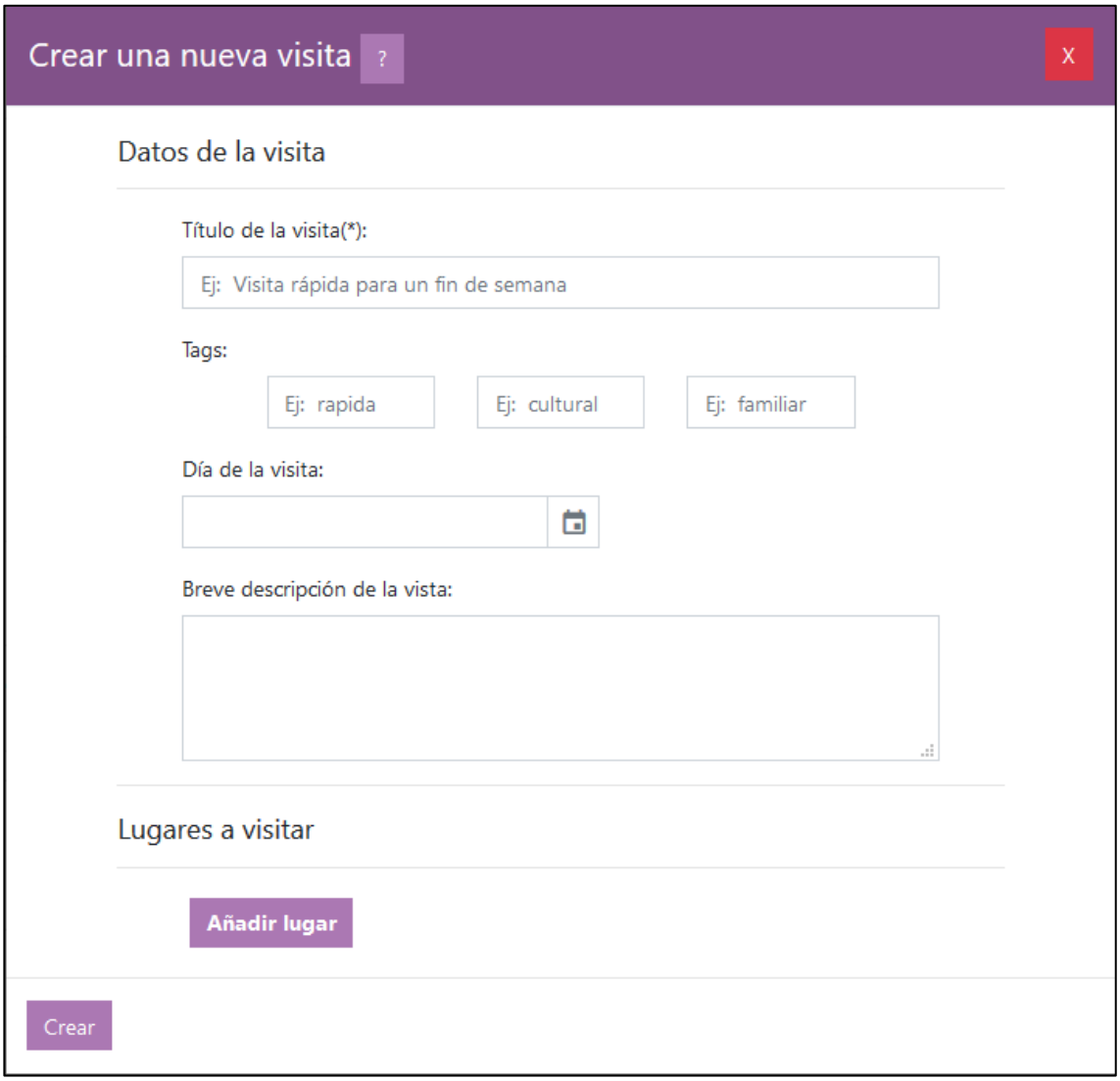
- Para poder acceder a los funciones de usuario registrado o adminsitrador debemos hacer login, por defecto en la entrega se distinguen los siguientes usuarios para acceder:

  Usuario registrado:
  - Email: user@ucm.es
  - Pass: user

  Administrador:
  - Email: admin@ucm.es
  - Pass: admin

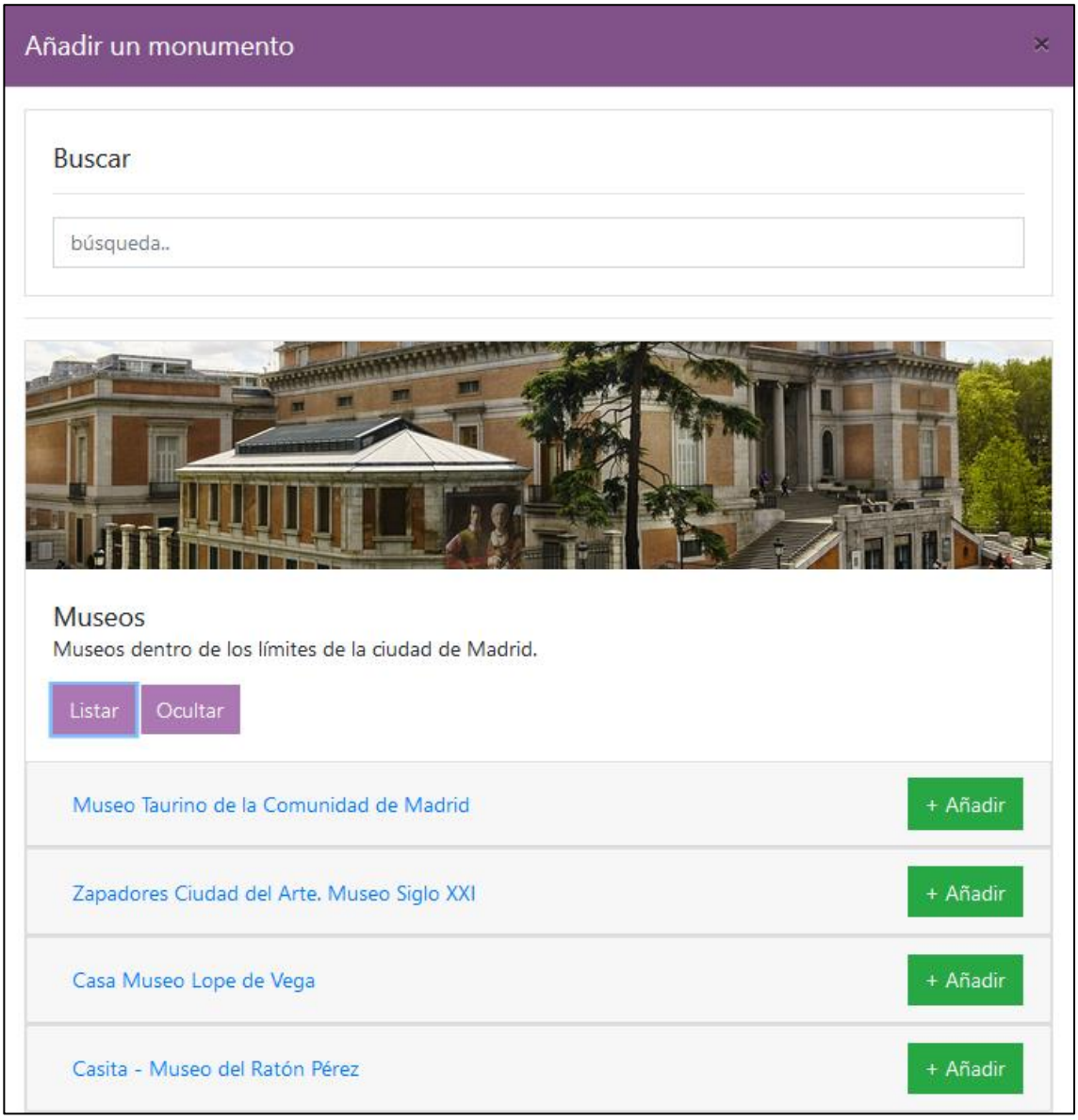
- Para crear una visita debemos acceder a la sección de mis visitas como usuario registrado y seleccionar la opción crear visita:

Una vez accedamos a crear nueva visita, se abrirá un panel nuevo con la información relevante de la visita a rellenar:



Para poder crear una visita se deben rellenar los campos de título, descripción y añadir al menos un lugar a visitar. Para añadir un lugar a visitar, se debe seleccionar la opción de añadir lugar y posteriormente seleccionar el lugar

deseado.


Añadir un monumento
Buscar
búsqueda..
Museos
Museos dentro de los límites de la ciudad de Madrid.
Listar
Ocultar
Museo Taurino de la Comunidad de Madrid
+ Añadir
Zapadores Ciudad del Arte. Museo Siglo XXI
+ Añadir
Casa Museo Lope de Vega
+ Añadir
Casita - Museo del Ratón Pérez
+ Añadir

BIBLIOGRAFÍA

Herron, D. (2011). Node Web Development (Second Edition). Packt Publishing, 2011.

Mark Clements, D. (2014). Node Cookbook (Second Edition). Packt Publishing, 2014.

Yaapa, H. (2013). Express Web Application Development. Packt Publishing, 2013.

Vlăduțu, A. (2014). Mastering Web Application Development with Express, 2014.

Wisborg Krogh, J.; Okuno, M. (2017). Pro MySQL NDB Cluster. Apress, 2017.

Node documentation, 2018. https://nodejs.org/api/index.html

Vukotic, A.; Goodwill, J (2011). Apache Tomcat 7. Apress, 2011.

Auer Sören, Bryl, V., Tramp, S., Eds.; Linked Open Data – Creating Knowledge Out of Interlinked Data. Springer Open 2014.

Gackenheimer, C (2013). Node.js Recipes. Apress, 2013.

Grannell, C. (2007). The Essential Guide to Css and Html Web Design. Friends of ED, 2007.

Smith, B. (2015). Beginning JSON (Learn the preferred data format of web). Apress, 2015.

Ritchie, Paul (2007). The security risks of AJAX/web 2.0 applications. Elsevier, 2007.

Gibbs, M. Php, Wamp and Xampp, oh My. Networld World, 2007, 26-26.

JQuery API documentation, 2018. https://api.jquery.com/

JQuery AJAX API documentation, 2018. https://api.jquery.com/category/ajax/

Krause, J. (2016). Introducing Bootstrap 4. Apress, 2016.

Bootstrap 4 documentation guide, 2018. https://getbootstrap.com/docs/4.1/getting-started/introduction/

MariaDB documentation, 2018. https://mariadb.com/kb/en/library/documentation/

Delisle, M. 2004. Dominar Phpmyadmin Para Una Administración Efectiva De Mysql, 1ª actualización.; Packt, 2004.

PhpMyAdmin documentation, 2018. https://www.phpmyadmin.net/docs/

Garcia-Izquierdo, F. J.; Izquierdo, R. Is the Browser the Side for Templating? Ieee Internet Computing 2012, 16 (1) DOI: 10.1109/MIC.2011.81.

ÍNDICE DE FIGURAS

Capítulo 3

Figura 3-1. Descargas por año del framework Express.js. 11
Figura 3.2.Funcionamiento de los middlewares en Express.js 11
Figura 3-3. Funcionamiento básico de AJAX. Fuente: Smith, 2015. 14

Capítulo 4

Figura 4-1. Diagrama de casos de uso 18

Capítulo 7

Figura 7.1. Formulario para crear una nueva visita. 49
Figura 7.2. Creación de una nueva visita. 50
Figura 7.3. Función validarVisita() 51
Figura 7.4. –Función crearVisita() 52
Figura 7.5. Archivo app.js. 53
Figura 7.8. Manejador de respuestas de la llamada AJAX al servidor. 54
Figura 7.9. Pantalla de Mis visitas mostrando el resultado exitoso de la operación. 54