Universitat Oberta de Catalunya

uoc.edu

Gestor de copias de seguridad de Rsync para configurar en un Virtual Private Server

David Campos Magdaleno
Grado de Ingeniería Informática
Desarrollo Web

Consultor: Gregorio Robles Martínez
Profesor: Santi Caballé Llobet

Junio de 2020

Agradecimientos

Quiero agradecer a mi familia el apoyo mostrado durante esta etapa de estudios, en especial a María y a Marta, por sus valiosos consejos.

Sirva el presente trabajo como prueba de que esta vez he ganado yo.

La mejor manera de luchar contra la Esclerosis Múltiple es llevándole la contraria.

No te inclines ante la adversidad; más bien oponte audazmente a ella, tanto cuanto tu suerte te lo permita

— Horacio

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Gestor de copias de seguridad de Rsync para configurar en un VPS (Virtual Private Server)*
Nombre del autor:	*David Campos Magdaleno*
Nombre del consultor/a:	*Gregorio Robles Martínez*
Nombre del PRA:	*Santi Caballé Llobet*
Fecha de entrega:	06/2020
Titulación:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Desarrollo web*
Idioma del trabajo:	*Castellano*
Palabras clave	*Respaldo de datos* *Framework* *Diseño centrado en el usuario*

Resumen del Trabajo:

El presente trabajo aborda el desarrollo de un gestor de copias de seguridad. Debido a la implantación de los servidores *Virtual Private Server* en el ámbito del desarrollo web, se propuso crear una aplicación para aprovechar esta tecnología, ahorrar costes, y que sea además ligera y sencilla. Para ello, se decidió crear una aplicación web desde cero, prescindiendo de la dependencia de otras plataformas o librerías, y desarrollada en PHP y SQLite.

Para el desarrollo, se utilizó el patrón de arquitectura de *sofware* MVC. El núcleo del *framework* contiene clases para gestionar las peticiones y respuestas, rutas, gestión de la base de datos, renderización de vistas, así como modelos y controladores. A estos componentes se les añadió un sistema de autenticación, de paso de mensajes, un módulo de gestión de usuarios, herramientas de desarrollo y pruebas funcionales.

La infraestructura creada maneja un sistema de copias de seguridad sobre Rsync gestionado mediante Cron. Además, cuenta con un sistema de logs y de interacción con la Shell de GNU/Linux. Para la realización de copias remotas se apoya en un intercambio de claves públicas y privadas, con un sistema de rotación local para disponer de un historial de *backups*.

Se concluye que la realización de pruebas funcionales durante el desarrollo de software es una parte importante del mismo, así como el hecho de que la generación de documentación y un código limpio y ordenado, facilitan la revisión y modificación.

Abstract:

This project deals with the development of a backup manager. Due to the implementation of Virtual Private Servers in the area of web development, it was intended to create an application to take advantage of this technology, save costs, this system being also light and simple. To this end, it was decided to create a web application from scratch, regardless of dependence on other platforms or libraries, and developed in PHP and SQLite.

For the development, the MVC software architecture pattern was used. The core of the framework contains classes to manage requests and responses, routes, database management, rendering of views, as well as models and controllers. An authentication, message passing system, a user management module, development tools and functional tests were added to these components.

The infrastructure created manages a backup system over Rsync managed by Cron. In addition, it has a system of logs and interaction with the GNU / Linux Shell. To make remote copies, it relies on an exchange of public and private keys, with a local rotation system to have a history of backups.

It is concluded that the performance of functional tests during software development is an important part of it, together with the fact that the generation of documentation and clean and orderly code facilitate revision and modification.

Índice

Lista de figuras

Lista de tablas

1 Introducción

1.1 Contexto y justificación del Trabajo

Durante años, la forma más accesible de poder publicar un pequeño proyecto en Internet era mediante un **servidor compartido**. Esta opción es la más económica pero también es muy limitada. Esto es debido a que se comparten todos los recursos de la máquina con el resto de los usuarios (dirección IP, procesador y memoria). Este tipo de alojamiento también tiene limitaciones técnicas y carece de la posibilidad de configuración de muchos aspectos. La alternativa a los servidores compartidos eran los **servidores dedicados**, en los que el usuario dispone de una máquina completa para gestionar a su entera disposición. Esto significa tener un acceso total además de disponer de todos los recursos del equipo. Como contrapartida, este tipo de alojamiento tiene unos precios elevados y dificilmente asumibles para la población general.

En los últimos años, los proveedores de servicios de Internet ofrecen **VPS** (Servidores Privados Virtuales) a precios muy competitivos. Los VPS son un paso intermedio entre los servidores compartidos y los servidores dedicados, que ofrecen la flexibilidad y control de la que no disponen en un servidor compartido, pero sin la potencia de un servidor dedicado. Los proyectos web también han evolucionado a lo largo de los años, pasando de ser páginas estáticas a los proyectos dinámicos de hoy en día, con contenido en constante actualización, por lo que los VPS también han pasado a ser la mejor opción para publicar pequeños o medianos proyectos.

Como el contenido es dinámico, también es necesario contar con un **sistema de copias de seguridad** para respaldar toda la información que se genera. Para ello existen muchas opciones, incluidas las que ofrecen los propios proveedores, previo pago, a través de aplicaciones o paneles de control. De esta manera, si se quiere prescindir de este gasto, las opciones se reducen.

Por ello, la idea de este trabajo es la de desarrollar una pequeña aplicación para instalar en un VPS, desde la cual un desarrollador pueda **gestionar y automatizar** copias de seguridad de sus proyectos en otros servidores a través de la red en un punto centralizado. Con el fin de optimizar la aplicación y el control de ésta, he realizado el proyecto sin depender de otras plataformas o recursos, limitando al máximo la dependencia de librerías de terceros. Así pues, con la implementación de un **marco de trabajo propio**, se dispondría de una base sólida para poder ampliar el proyecto de forma rápida y sencilla.

1.2 Objetivos del Trabajo

El objetivo principal de este trabajo es la obtención de una plataforma, llamada **BATEA** (***B****ackup* ***A****dministrator* ***T****ool* ***E****nd-of-degree* ***A****pplication*) que permite realizar copias de seguridad programadas por el usuario de forma automatizada mediante el software Rsync. Para el desarrollo de esta herramienta se creó un *framework ad hoc* en PHP partiendo de cero.

A continuación, se detallan los subobjetivos:

- Estudiar las características de las herramientas que se van a utilizar para familiarizarse con esas tecnologías:
  - Estudiar las posibilidades que ofrece Rsync.
  - Realizar pruebas desde una Shell para explorar las distintas posibilidades a la hora de realizar la aplicación y así plasmarlo en la aplicación.
  - Estudiar las formas de acceso a través de SSH.

- Construir un micro *framework* en PHP para poder desarrollar la idea propuesta, con las características necesarias para poder construir el gestor de copias de seguridad. Para ello es necesario desarrollar los siguientes elementos:
  - Implementar la gestión de rutas.
  - Implementar un sistema de acceso a la base de datos.
  - Implementar las sesiones y la autenticación.
  - Implementar el renderizado de las plantillas.
  - Implementar la gestión de tareas programadas.

- Programar la aplicación web:
  - Implementar la creación de una nueva copia de seguridad.
  - Implementar la configuración de una copia de seguridad (configurar origen, regularidad, exclusión de archivos y otras opciones).
  - Implementar la restauración de la copia de seguridad.
  - Implementar un sistema de avisos al correo (errores, éxitos y otros eventos a considerar)
  - Documentar el código de la aplicación.

- Elaborar el diseño y plasmarlo en HTML y CSS.
  - Elegir un *framework* CSS que sea adaptable.
  - Implementar el diseño con el *framework* seleccionado.

1.3 Enfoque y método seguido

Puesto que el presente trabajo tiene un gran contenido de desarrollo, la estrategia seguida se llevó a cabo en tres pasos.

En primer lugar, la de documentarme acerca del software sobre el que se iba a interactuar (SSH, Rsync y Shell), y así evaluar las posibilidades que ofrecen. Puesto que la idea en un primer momento fue la de desarrollar un **producto nuevo**, valoré la utilización de *frameworks* de desarrollo o la utilización de sus componentes para la elaboración del producto final. Pese a que estos componentes o *frameworks* son muy potentes, fáciles de instalar y de desarrollar sobre ellos, decidí prescindir de ellos y concluí que la forma más apropiada de construir la plataforma era elaborar yo mismo el marco de desarrollo, ya que, en caso contrario, dependería de proyectos complejos, con infinidad de características y dependencia de otro software. De esta forma se pretende lograr un **producto sencillo**, con **características muy definidas** y así obtener un

programa mucho **más ligero**, cosa que no podría lograrse con el uso del software mencionado.

En segundo lugar, una vez superado el punto anterior, ya disponía de un *framework* sobre el cual implementar y desarrollar la aplicación de forma mucho más fácil y rápida. Así que se inició la implementación del sistema de **copias de seguridad**. Para ello se necesitó la creación de una red de máquinas virtuales para poder simular la transferencia de archivos a través de la red.

En tercer y último lugar, se trabajó en el diseño de la **interfaz** de usuario. Si bien se podría utilizar una plantilla HTML, decidí realizar un diseño propio buscando ante todo la **simplicidad** y pensando en **el usuario**.

1.4 Planificación del Trabajo

La siguiente figura muestra la planificación temporal del proyecto en un diagrama de Gantt. Los puntos rojos representan las entregas de las PEC.

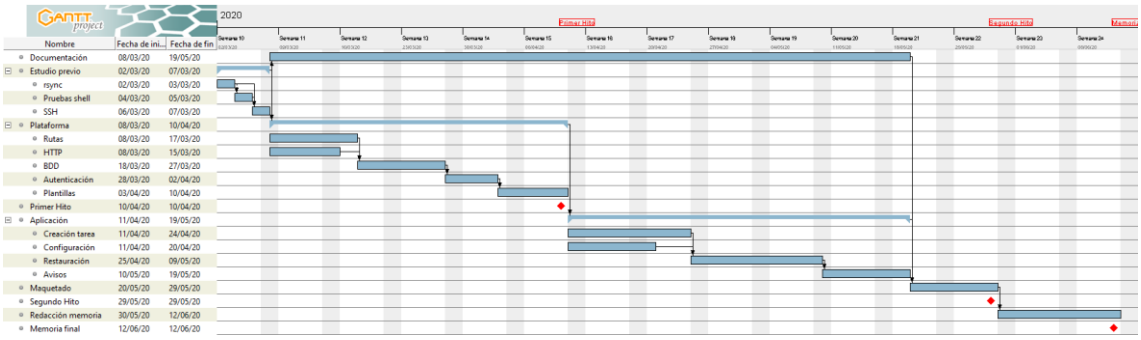

Fig. 1 Diagrama de Gantt

1.5 Breve sumario de productos obtenidos

BATEA como *framework* de desarrollo en PHP.

BATEA como gestor de copias de seguridad. Proyecto funcional disponible en la plataforma GitHub. El código está liberado con una licencia MIT[1]. Puede descargarse y consultar los pasos de instalación desde la dirección https://github.com/davidcmg/batea

Documentación de referencia sobre la funcionalidad del proyecto. Disponible para su consulta en línea en la web https://davidcmg.github.io/batea/

1.6 Breve descripción de los otros capítulos de la memoria

El **capítulo 2** contiene dos apartados diferenciados. En primer lugar, se introduce el tema del presente trabajo en el subapartado “Estado de la cuestión”. A continuación, se resumen las tecnologías que se van a utilizar para su

[1] La licencia puede consultarse en https://github.com/davidcmg/batea/blob/master/LICENSE.md

desarrollo. Este capítulo se utiliza para poner al lector en contexto acerca del producto a desarrollar.

En el **capítulo 3**, se aborda el desarrollo del proyecto, que abarca desde los casos de uso a la interfaz de usuario. Se deben diferenciar cuatro apartados: (1) los casos de uso; (2) el desarrollo del *framework*; (3) el gestor de copias de seguridad; (4) la interfaz de usuario. En este capítulo se explican las decisiones tomadas en la implementación del producto.

El **capítulo 4** incluye las conclusiones obtenidas tras la finalización del proyecto, así como futuras ampliaciones.

Los dos últimos capítulos se corresponden con el glosario y la bibliografía.

2 Estado de la cuestión y tecnologías utilizadas

Como norma general, la utilización de ordenadores conlleva la posibilidad de pérdidas de información. A raíz de este problema surgieron los gestores de copias de seguridad, que permiten recuperar la información resguardada en una fecha determinada.

2.1 Estado de la cuestión

El despliegue de una aplicación en un servidor de Internet no queda exento de posibles pérdidas de información, sino que el problema puede verse agravado, ya que se puede tratar de información que se genere o que cambie con mucha frecuencia, por lo que en caso de falla pueda producir daños irreparables. La utilización de máquinas en Internet conlleva que el servidor está **expuesto a todo el mundo**, por lo que no está exento de posibles **ataques** informáticos, a los que hay que añadirles **fallos en el software o en el hardware**. Pese a que es posible automatizar las copias de seguridad en el propio servidor en dónde se tenga alojado el proyecto, es recomendable realizarlas en otra máquina de forma externa, ya que de esta forma se **reduce la pérdida de información** en el caso de un fallo de disco o alguna otra incidencia.

2.1.1 Soluciones existentes

Hoy en día existen muchas opciones en el mercado para dar solución a este problema. Los propios proveedores de alojamiento ofrecen servicios adicionales para la realización de copias de seguridad externas, y suelen ser soluciones propias del proveedor. Los paneles de control también ofrecen la posibilidad de la realización de copias de seguridad, y por último está el software específico, que tendría que instalarse y configurarse por el propio administrador del VPS. A continuación, comentaré brevemente algunas de las opciones que se pueden encontrar o implementar en un VPS GNU/Linux:

- **Soluciones de los proveedores:** los proveedores de alojamiento utilizan su infraestructura para ofrecer soluciones de copias de seguridad. Si bien pueden ser ofertadas como un servicio gratuito, conviene reforzarlas, ya que estas no suelen ser configurables y se realizan dentro de la misma máquina contratada. Esta opción está generalmente integrada en sus propios paneles de control. Otra variante de este tipo de soluciones es su contratación como **servicio** aparte. Las copias se realizan de forma **externa**, lo que implica que se podrá disponer de la información en caso de falla. El precio varía según el tamaño del disco del que se quiera hacer uso.

- **Panel de control:** otro servicio disponible es el de la adquirir una licencia de un panel de control. Los más habituales son Plesk o CPanel que se utilizan para configurar y administrar los servidores mediante un panel de control web. Además, también existen alternativas *Open Source* como puede ser Webmin, Virtualmin, aunque con más limitaciones y no tan completas como

las anteriores. Este tipo de paneles contempla la opción de realizar copias de seguridad a un equipo remoto.

- **Aplicaciones de copias de seguridad**: Son una buena opción a la hora de gestionar las copias de seguridad sin que se vea incrementado el precio. A su vez, podemos categorizarlas de la siguiente manera:

    - Como **herramientas** potentes de respaldo *Open Source* y basados en la arquitectura **cliente/servidor** cabe destacar a Bacula y a Amanda. Cuentan con muchas características y opciones, además de un amplio abanico de módulos y clientes. Por la contra, su instalación y configuración puede llegar a ser compleja para usuarios poco experimentados.
    - De entre las herramientas de respaldo para **línea de comandos.** Habría que citar Rsync, Rsnapshot o Burp entre otros, que permiten, además, realizar la copia de ficheros **a través de la red**.

2.1.2 Tipos de copias de seguridad

Además de la posibilidad de realizar copias de seguridad en la máquina local o en un servidor remoto, existe la posibilidad de que estas copias sean completas, incrementales o diferenciales. En la copia **completa**, como su propio nombre indica, se realiza una copia directa del contenido a resguardar. Generalmente, requiere de mucho tiempo ya que tiene que copiar todo el contenido cada vez que se ejecute, y en el caso de que se quiera guardar copias antiguas, se incrementaría mucho el espacio en disco ya que siempre se copia todo el contenido. Como alternativa a este tipo de copias, existen las copias **diferenciales**, que, trabajando sobre una copia completa, en los siguientes respaldos solamente se copiaría el contenido modificado. Por último, las copias **incrementales** copian solamente el contenido modificado desde el último *backup*, ya sea incremental o completa, por lo que este tipo de copias son las más rápidas y no consumen demasiado el espacio en disco.

2.2 Tecnologías utilizadas

Las tecnologías que se utilizaron para la realización de este proyecto fueron las siguientes:

2.2.1 PHP

Pese a que existen otros lenguajes para el desarrollo de aplicaciones web, me decanté por PHP, -un lenguaje de scripting que se ejecuta del lado del servidor-, por su amplia utilización e implementación en los servidores web de todo el mundo. Además, es software libre, estable, tiene una buena comunicación entre la web y el servidor, y cuenta con una buena documentación [1] que fue clave para el desarrollo del presente trabajo. Se utilizó la versión 7.4, la última estable publicada.

Paralelamente, la gestión de paquetes se realizó con *Composer*, un gestor de las dependencias del proyecto. Dichas dependencias son las siguientes:

- phpseclib/phpseclib [2]: es una librería criptográfica para PHP utilizada para realizar conexiones seguras entre equipos y para la gestión de claves criptográficas.
- phpmailer/phpmailer [3]: clase para la creación y transferencia de correo electrónico en PHP.

Para el desarrollo de las pruebas se utilizó Codeception [4] y alguno de sus módulos para añadir funcionalidades:

- codeception/Codeception
- codeception/module-phpbrowser
- codeception/module-webdriver

2.2.2 Máquina GNU/Linux y aplicaciones

Los servidores GNU/Linux son el objetivo de la aplicación de este proyecto. Se utilizó una máquina virtual con las aplicaciones a implementar, en este caso Rsync y OpenSSH. Además, sirvió para controlar la configuración del entorno de desarrollo.

Rsync [5] es una herramienta para copiar ficheros, con múltiples opciones, y que permite trabajar de forma local y remota. Su algoritmo reduce la cantidad de datos que se envían a través de la red. Además, utilizando los parámetros correctos podemos reducir considerablemente el tamaño de las copias en disco utilizando **enlaces duros**. Esto significa que dos o más archivos comparten el mismo **inodo**, por lo que no ocupará espacio en disco. En la siguiente Figura (Fig. 2) se muestra un ejemplo de esta característica. Puede observarse que el tamaño del directorio de la última copia de seguridad (*task_5/last/*) es de 2.1Gb, el mismo que el de la copia anterior en el tiempo (*task_5/files_-1/*). Además, el tamaño del directorio que la contiene a ambas copias (*task_5/*), al tratarse de enlaces duros, se observa que su tamaño no es la suma de ambos, sino que es el tamaño real: 2.1GB. En la parte inferior de la Figura también puede observarse que los archivos tienen el mismo inodo.

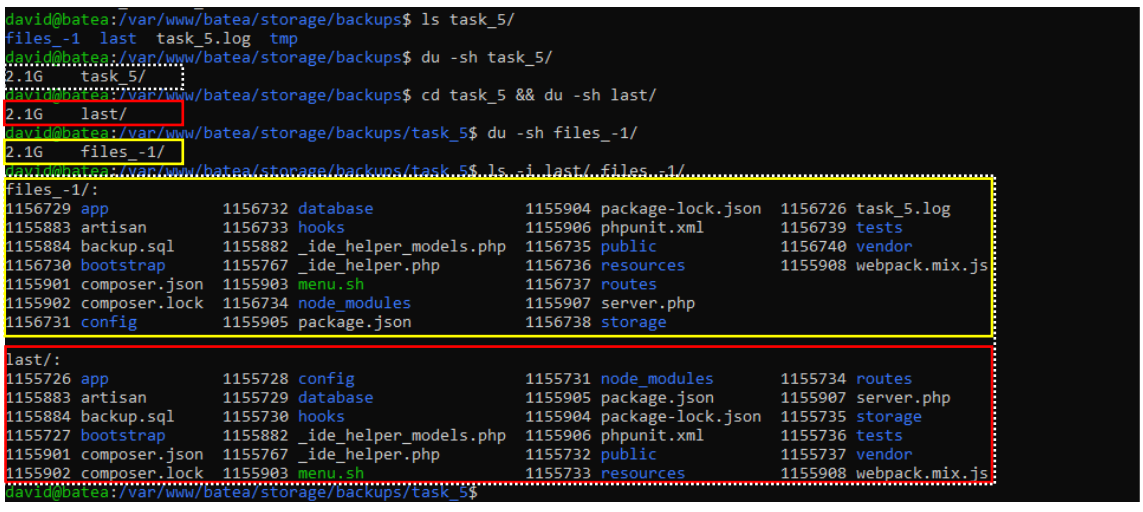

Fig. 2 Tamaño de las copias

Shell es el intérprete de comandos para GNU/Linux. Procesa comandos en modo texto y permite la ejecución de *scripts* para automatizar tareas. Un ejemplo es el que se muestra en la captura anterior (Fig. 2).

OpenSSH es un conjunto de aplicaciones entorno al algoritmo SSH que permite el acceso a una Shell de forma remota y segura. Esta comunicación se realiza a través de un canal seguro y toda la información que viaja a través de él está cifrada. También se utiliza para la generación de claves asimétricas.

2.2.3 SQLite

El gestor de bases de datos relacional SQLite es un sistema de gestión pequeño y que no utiliza un proceso para hacer de servidor, sino que se utiliza un fichero enlazado por un programa, buscando la sencillez del proyecto y para no depender de grandes gestores de bases de datos. De esta forma se logra poder **instalar el software sin depender ni configurar un servidor de bases de datos**.

2.2.4 HTML y CSS

La interfaz de usuario está maquetada con HTML, CSS y JavaScript. Para facilitar esta tarea se utilizó *Bootstrap* [1], un *framework* que se emplea como herramienta de prototipado rápido. Para facilitar la creación de la interfaz de usuario, se utilizó Npm[2] como herramienta para **procesar** las fuentes CSS y JavaScript.

2.2.5 phpDocumentor

Es un **generador de documentación** a partir de los comentarios del código fuente. Analiza el proyecto en busca de fragmentos de la documentación que se añade antes de cada clase o de cada función. A medida que desarrollaba el código iba documentando cada método y exportando esa documentación con phpDocumentor[3] a formato HTML. La documentación generada está disponible para su consulta en la red.

---

[2] Página oficial del proyecto: https://www.npmjs.com/

[3] Proyecto phpDocumentor en GitHub: https://github.com/phpDocumentor/phpDocumentor

3 Desarrollo del proyecto

Desde el inicio del proyecto he realizado la estructura de la aplicación sobre la que funcionará y se mostrará la interfaz para gestionar las copias de seguridad. Esta estructura será un micro *framework*, con las funcionalidades básicas para la realización del presente trabajo.

Pero antes de realizar la programación, me documenté acerca del funcionamiento de Rsync, que es el software principal sobre el que se realizarán las copias de seguridad. Para ello he tomado nota de los principales parámetros que permiten hacer diferentes tipos de copias de ficheros. También necesitaba documentarme sobre como automatizar procesos desde la terminal de GNU/Linux, así como el acceso remoto mediante el protocolo SSH generando claves, para así no depender del usuario para que efectúe un inicio de sesión cada vez que se realiza una conexión programada. Tras el estudio previo de estas tecnologías, realicé pruebas creando *scripts* para testear sus posibilidades y su correcto funcionamiento. Una vez documentadas las características de estos programas, inicié la programación de la estructura de la aplicación. Para ello me apoyé en Composer para cargar los *namespaces* siguiendo el estándar PSR-4 y realicé la estructura de directorios. Avanzado el proyecto, necesité añadir programas externos para realizar la documentación y procesar test y estilos:

- **Codeception**: en un primer momento empecé a realizar las pruebas en PHPUnit, pero se quedaban cortos y se complicaba el trabajo. Ante este problema, y teniendo en cuenta que necesitaba test funcionales en lugar de unitarios, decidí utilizar Codeception.
- **Npm**: se trata de un proyecto que procesa y empaca recursos JavaScript y CSS utilizando el software Webpack.

3.1 Casos de uso

Caso de uso	Acceso usuario registrado
Resumen de la funcionalidad	Para poder acceder a la plataforma, el usuario tendrá que acceder como usuario registrado. Al acceder tendrá acceso a la gestión de las copias de seguridad.
Actores	Usuario.
Precondición	El usuario debe tener una cuenta.
Postcondición	El usuario obtiene una sesión en el sistema. El usuario accede al panel de control de la plataforma.
Escenario principal de éxito	1. El usuario accede al formulario de acceso 2. El usuario indica los datos de acceso 3. El sistema valida los datos 4. El sistema redirige al usuario a la página principal
Escenarios alternativos	1. El usuario no indica los datos de acceso a. El sistema indica al usuario los campos que debe completar 2. Los datos indicados son incorrectos

	a. El sistema indica al usuario que los datos no son correctos

Tabla 1 Caso de uso Acceso usuario registrado

Caso de uso	Cerrar sesión
Resumen de la funcionalidad	El usuario cierra sesión en el panel de control.
Actores	Usuario registrado.
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión.
Postcondición	El sistema cierra la sesión de usuario. El sistema redirige al formulario de acceso.
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de cerrar sesión 2. El sistema cierra la sesión 3. El sistema redirige al usuario al formulario de acceso
Escenarios alternativos	1. Se produce un error. a. El sistema muestra un error *403 – Forbidden* por caducidad de la sesión

Tabla 2 Caso de uso Cerrar sesión

Caso de uso	Crear usuario
Resumen de la funcionalidad	El usuario registrado crea una cuenta de usuario.
Actores	Usuario registrado.
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. El nuevo usuario que se quiere crear no debe existir en el sistema.
Postcondición	Se crea una nueva cuenta de usuario.
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de crear un usuario 2. El usuario registrado rellena los datos del nuevo usuario 3. El usuario registrado hace clic en el botón de crear nuevo usuario 4. El sistema crea el usuario a. El sistema encripta la contraseña que se indica b. El sistema redirige al usuario al listado de usuarios c. El sistema muestra un mensaje de éxito
Escenarios alternativos	1. Se produce un error. a. El sistema muestra un error indicando que la nueva contraseña no se ajusta a los parámetros de seguridad

	b. El sistema muestra un error indicando que el usuario que se quiere crear ya existe en el sistema

Tabla 3 Caso de uso Crear usuario

Caso de uso	Eliminar usuario
Resumen de la funcionalidad	El usuario registrado puede eliminar otra cuenta de usuario.
Actores	Usuario registrado.
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. La cuenta de usuario que se desea borrar debe ser una cuenta distinta a la del usuario registrado. El sistema debe contar con al menos dos cuentas de usuario antes de eliminar una cuenta de usuario.
Postcondición	Se elimina la cuenta de usuario. Las copias de seguridad e información del usuario que se borra se mantienen.
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de borrar un usuario 2. El sistema pide una confirmación al usuario registrado 3. El usuario registrado confirma el usuario que desea borrar 4. El sistema borra la información de la cuenta a. El sistema redirige al usuario registrado al listado de usuarios b. El sistema muestra un mensaje de éxito
Escenarios alternativos	1. Se produce un error. a. El sistema muestra un error indicando que el usuario a borrar es el activo

Tabla 4 Caso de uso Eliminar usuario

Caso de uso	Editar usuario
Resumen de la funcionalidad	El usuario registrado edita los datos de las cuentas de usuario.
Actores	Usuario registrado.
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión.
Postcondición	Se cambia la contraseña del usuario.
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de editar un usuario 2. El usuario registrado edita la contraseña de una cuenta de usuario 3. El usuario registrado hace clic en el botón de cambiar contraseña 4. El sistema redirige al usuario al listado de usuarios

	5. El sistema muestra un mensaje de éxito
Escenarios alternativos	1. Se produce un error. a. El sistema muestra un error indicando que la nueva contraseña no se ajusta a los parámetros de seguridad

Tabla 5 Caso de uso Editar usuario

Caso de uso	Añadir un servidor
Resumen de la funcionalidad	El usuario registrado añade una máquina al sistema.
Actores	Usuario registrado.
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. El usuario tiene acceso remoto a la máquina que se desea añadir.
Postcondición	Se añade un servidor remoto al sistema. Se guarda el estado del servidor.
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de crear un servidor 2. El usuario registrado indica los datos de acceso al servidor 3. El usuario registrado hace clic sobre el botón de crear un servidor 4. El sistema da de alta el servidor a. El sistema accede al servidor remoto con el usuario y la contraseña indicados por el usuario registrado b. El sistema copia la clave pública del sistema al archivo de claves autorizadas en el servidor remoto c. El sistema crea un nuevo servidor remoto d. El sistema añade un estatus activo al nuevo servidor e. El sistema redirige a la página del listado de servidores remotos f. El sistema muestra un mensaje de éxito
Escenarios alternativos	1. El servidor remoto no es accesible a. El sistema redirige al usuario registrado al formulario de añadir servidor b. El sistema muestra un error indicando que el servidor remoto no es accesible 2. No se logra establecer una conexión a. El sistema redirige al usuario registrado al formulario de añadir servidor b. El sistema muestra el error de conexión 3. El sistema no logra copiar la clave pública en el servidor remoto

	a. El sistema redirige al usuario registrado al formulario de añadir servidor b. El sistema muestra un error indicando el problema

Tabla 6 Caso de uso Añadir un servidor

Caso de uso	Editar un servidor
Resumen de la funcionalidad	El usuario registrado edita una máquina remota.
Actores	Usuario registrado.
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. Se debe haber realizado el caso de uso Añadir un servidor.
Postcondición	Se guardan los nuevos datos de acceso al servidor remoto.
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de editar un servidor 2. El usuario registrado edita los datos de acceso al servidor 3. El usuario registrado hace clic sobre el botón de guardar 4. El sistema guarda los nuevos datos del servidor remoto a. El sistema redirige a la página de servidores b. El sistema muestra un mensaje de éxito
Escenarios alternativos	1. Se produce un error a. El sistema redirige al usuario registrado al formulario de editar servidor b. El sistema muestra un error indicando que los nuevos valores no son válidos

Tabla 7 Caso de uso Editar un servidor

Caso de uso	Eliminar un servidor
Resumen de la funcionalidad	El usuario registrado borra una máquina remota
Actores	Usuario registrado
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. Se debe haber realizado el caso de uso Añadir un servidor.
Postcondición	Se elimina el servidor remoto del sistema. Las copias de seguridad realizadas sobre ese servidor se mantienen en el equipo.
Escenario principal de éxito	2. El usuario registrado hace clic en el botón de borrar un servidor 3. El sistema pide una confirmación al usuario registrado 4. El usuario confirma el servidor que desea borrar del sistema

	5. El sistema redirige al listado de servidores 6. El sistema muestra un mensaje de éxito.
Escenarios alternativos	

Tabla 8 Caso de uso Eliminar un servidor

Caso de uso	Crear una tarea
Resumen de la funcionalidad	El usuario registrado añade una tarea para programar las copias de seguridad.
Actores	Usuario registrado.
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. Se debe haber realizado el caso de uso Añadir un servidor.
Postcondición	1. Se añade una tarea programada al sistema.
Escenario principal de éxito	2. El usuario registrado hace clic en el botón de crear una tarea 3. El usuario registrado elige un servidor sobre el que programar una copia de seguridad 4. El usuario registrado indica los datos adicionales para programar la tarea 5. El usuario registrado hace clic sobre el botón de crear una tarea 6. El sistema guarda la tarea a. El sistema redirige a la página del listado de tareas b. El sistema muestra un mensaje de éxito
Escenarios alternativos	1. El sistema muestra un mensaje de error a. El sistema redirige al usuario registrado al formulario de añadir una tarea b. El sistema muestra un error indicando que los datos proporcionados no son correctos

Tabla 9 Caso de uso Crear una tarea

Caso de uso	Editar una tarea
Resumen de la funcionalidad	El usuario registrado modifica una tarea programada
Actores	Usuario registrado.
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. Se debe haber realizado el caso de uso Añadir una tarea.
Postcondición	Se guardan los nuevos datos de la tarea.
Escenario principal de éxito	2. El usuario registrado hace clic en el botón de editar una tarea 3. El usuario registrado edita los datos de la tarea 4. El usuario registrado hace clic sobre el botón de guardar

	5. El sistema guarda los nuevos datos a. El sistema redirige a la página de tareas b. El sistema muestra un mensaje de éxito
Escenarios alternativos	1. Se produce un error a. El sistema redirige al usuario registrado al formulario de editar la tarea b. El sistema muestra un error indicando que los nuevos valores no son válidos

Tabla 10 Caso de uso Editar una tarea

Caso de uso	Eliminar una tarea
Resumen de la funcionalidad	El usuario registrado borra una tarea
Actores	Usuario registrado
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. Se debe haber realizado el caso de uso Añadir una tarea.
Postcondición	Se elimina la tarea programada del sistema. Las copias de seguridad realizadas sobre esa tarea se mantienen en el equipo.
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de borrar una tarea. 2. El sistema pide una confirmación al usuario registrado. 3. El usuario confirma la tarea que desea borrar del sistema. 4. El sistema redirige al listado de tareas. 5. El sistema muestra un mensaje de éxito.
Escenarios alternativos	

Tabla 11 Caso de uso Eliminar una tarea

Caso de uso	Descargar log
Resumen de la funcionalidad	El usuario descarga el log de la ejecución de una copia de seguridad
Actores	Usuario registrado
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. Se debe haber realizado el caso de uso Añadir una tarea. El sistema debe haber ejecutado una tarea
Postcondición	El usuario descarga un archivo .txt con el log seleccionado
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de descargar un log 2. El sistema devuelve una respuesta con el archivo seleccionado.
Escenarios alternativos	

Tabla 12 Caso de uso Descargar el log de un respaldo

Caso de uso	Descargar una copia de seguridad
Resumen de la funcionalidad	El usuario descarga una copia de seguridad
Actores	Usuario registrado
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. Se debe haber realizado el caso de uso Añadir una tarea. El sistema debe haber ejecutado una tarea
Postcondición	El usuario descarga un archivo .zip con la copia de seguridad seleccionada
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de descargar una copia de seguridad 2. El sistema devuelve una respuesta con la copia de seguridad seleccionada. a. El sistema crea un archivo comprimido. b. El sistema recorre de forma recursiva los datos resguardados y los añade al archivo. c. El sistema devuelve una respuesta con el archivo. d. El sistema borra el archivo del servidor.
Escenarios alternativos	

Tabla 13 Caso de uso Descargar una copia de seguridad

Caso de uso	Restaurar copia de seguridad
Resumen de la funcionalidad	El usuario restaura una copia de seguridad
Actores	Usuario registrado
Precondición	El usuario es un usuario registrado. El usuario accedió previamente como usuario registrado e inició una sesión. Se debe haber realizado el caso de uso Añadir una tarea. El sistema debe haber ejecutado una tarea
Postcondición	El sistema muestra un resumen con los datos copiados
Escenario principal de éxito	1. El usuario registrado hace clic en el botón de restaurar una copia de seguridad 2. El sistema pide confirmación para la restauración de la copia. 3. El sistema muestra un mensaje de confirmación.
Escenarios alternativos	1. El sistema muestra un mensaje de error.

Tabla 14 Caso de uso Restauración de una copia de seguridad

Caso de uso	Ejecución de una tarea

Resumen de la funcionalidad	El sistema ejecuta una tarea
Actores	Sistema
Precondición	Se debe haber realizado el caso de uso Añadir una tarea.
Postcondición	Se crea una tarea en el sistema. Las copias de seguridad realizadas sobre ese servidor se mantienen en el equipo.
Escenario principal de éxito	1. El sistema genera una lista de todas las tareas que existen 2. El sistema comprueba que la programación horaria de la tarea coincide con la fecha actual a. El sistema comprueba que el servidor remoto está en línea b. El sistema obtiene una conexión remota c. El sistema comprueba que existe el directorio sobre el que se quiere realizar la copia de seguridad d. El sistema prepara los directorios para las copias de seguridad e. El sistema rota las copias de seguridad 3. El sistema ejecuta la sincronización a. El sistema guarda el estado de la copia de seguridad b. El sistema envía un correo electrónico si se indicó en la programación de la tarea 4. En caso de que no coincida con la fecha actual pasa a la siguiente tarea
Escenarios alternativos	1. El servidor remoto no es accesible a. El sistema guarda el estado de la copia de seguridad como fallida 2. El sistema no logra obtener una conexión con el servidor remoto a. El sistema deja constancia de la incidencia en el log del sistema 3. El sistema no logra crear los directorios para realizar las copias de seguridad a. El sistema registra el error en el log del sistema 4. El sistema no logra realizar la copia de seguridad a. El sistema registra el error en el log del sistema b. El sistema guarda el estado de la copia de seguridad como fallida

Tabla 15 Caso de uso Ejecución de una tarea

Caso de uso	Comprobación del estado de los servidores remotos
Resumen de la funcionalidad	El sistema comprueba que los servidores remotos están en línea
Actores	Sistema
Precondición	Se debe haber realizado el caso de uso Añadir un servidor.
Postcondición	El sistema registra un cambio de estado

Escenario principal de éxito	1. El sistema genera una lista de todos los servidores 2. El sistema comprueba que el servidor está en línea a. En caso de que el estado anterior del servidor sea offline, cambia el estado b. En caso de que el estado anterior del servidor sea online, continúa 3. El sistema pasa al siguiente servidor
Escenarios alternativos	1. El sistema comprueba que el servidor está fuera de línea a. En el caso de que el estado anterior del servidor sea offline, continúa b. En caso de que el estado anterior del servidor sea online, cambia el estado.

Tabla 16 Caso de uso Comprobación del estado de los servidores remotos

3.2 Arquitectura detallada del *framework* Batea

El desarrollo de la aplicación se realizó siguiendo el patrón **Modelo-vista-controlador**. Este patrón nos propone el desarrollo sobre tres componentes diferenciados: (1) el **Modelo** representa la información con la que se va a operar; (2) el **Controlador** define la lógica de los eventos e interactúa con los Modelos y las Vistas; (3) la **Vista** representa gráficamente el Modelo sobre el que se actúa, y será la que se devuelva al usuario.

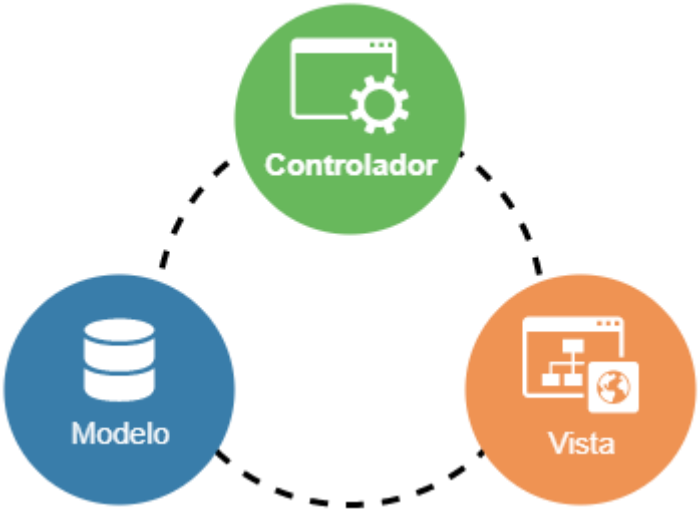

Fig. 3 Patrón MVC

Además, diferencio en directorios la programación de componentes, estando en src/Core/ el núcleo de la aplicación, que contiene los elementos básicos para que funcione el sistema. En src/Controllers/ se encuentran los controladores de la aplicación y en src/Models/ los modelos. Dicho esto, para tener acceso a la aplicación, se define **un punto de partida** que en todo caso va a ser el archivo index.php (Fig. 4). Situado en un directorio público del servidor web, sirve como punto de entrada y puente al resto de la aplicación. Es además

el único archivo del sistema accesible directamente desde el navegador web (directorio public/), de esta forma, el resto de los archivos de la aplicación no están expuestos en Internet, incrementando la seguridad del programa. Su única función es la de llamar al archivo bootstrap.php (Fig. 6) que no es accesible por el navegador. El archivo bootstrap.php contiene una serie de instrucciones para **iniciar la aplicación**, como la instancia de *Request* y *Response*, así como del *Router*.

```php
<?php

require_once __DIR__ . '/../src/bootstrap.php';

```

Fig. 4 Contenido del archivo index.php

El funcionamiento, de forma simplificada, se puede ver en un esquema de la figura Fig. 5 y la figura Fig. 6. Bootstrap crea objetos *Request* y *Response* para manejar las solicitudes y las respuestas y por último se instancia *Router*, que es la clase que va a manejar las rutas definidas de la aplicación. Router se encarga de llamar al controlador con su respectiva acción, y aquí es en dónde se desarrolla la lógica de cada apartado de la web. SQLite consulta los datos en la base de datos y los procesa a la clase View, que renderizará y mostrará la página resultante.

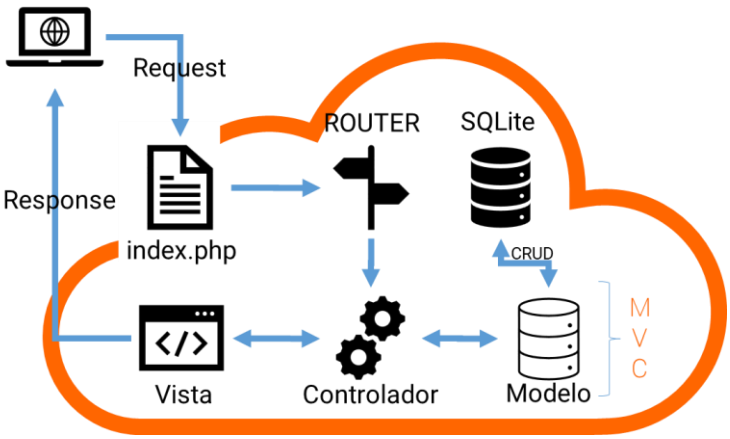

Fig. 5 Esquema de funcionamiento básico

Los directorios src/Controllers y src/Models contienen los controladores y los modelos de la aplicación. El directorio templates/ contiene las plantillas que se renderizarán para mostrar los resultados.

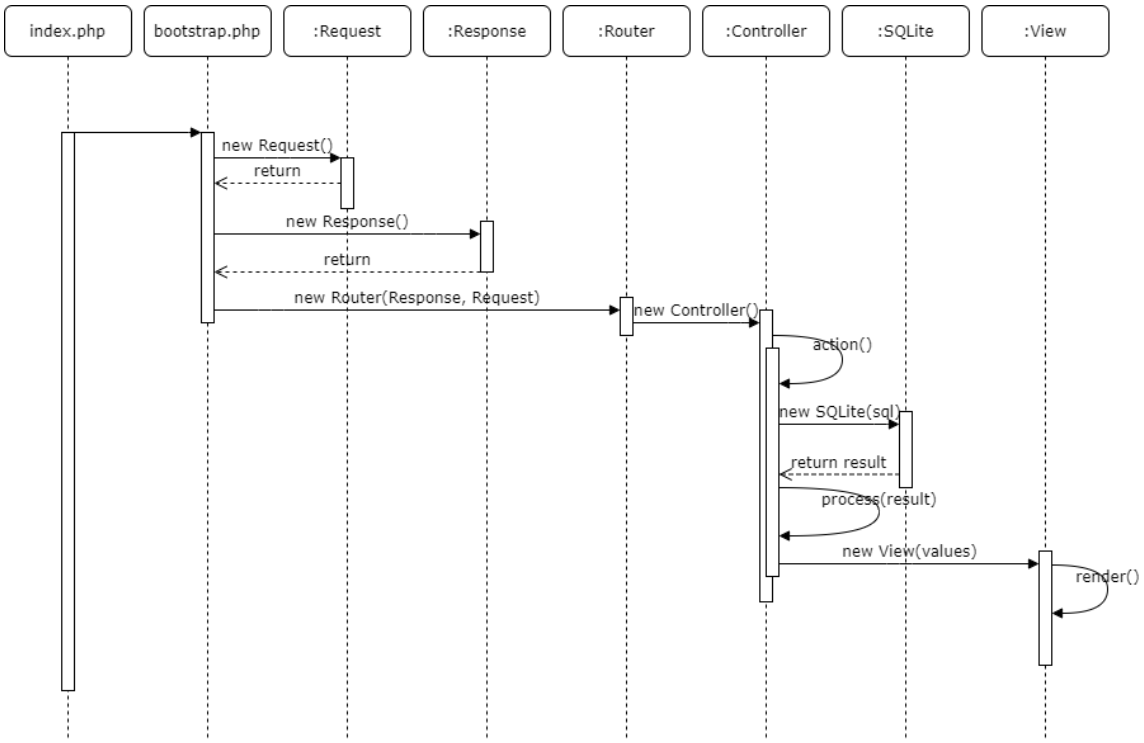

Fig. 6 Diagrama de secuencia

3.2.1 Solicitudes y respuestas

Para desarrollar el proyecto empecé implementando las clases *Http*[4] *Request* y *Response*, las cuales se podrían considerar los **cimientos del sistema**, ya que es la base para procesar la información de lo que se solicita a través del navegador, y por lo tanto es el **primer paso para desarrollar un sistema que pueda interactuar con un usuario**. La clase Request implementa HTTP Request [6] y se utiliza para gestionar las peticiones que se realizan al servidor. En esta clase se analiza la URL con la que se accede para así obtener la funcionalidad que se necesita ejecutar, o también calcular, los parámetros que se envían a la aplicación. Por otra parte, la clase Response, que implementa HTTP Response [6], se utiliza para gestionar las respuestas del servidor hacia el cliente. En esta clase se definen los códigos de respuesta, cabeceras y el propio contenido que se muestra al usuario.

Uno de los aspectos clave en el desarrollo del *framework* es la gestión de rutas, ya que las URL son el mecanismo que tiene un navegador web para interactuar con la aplicación. Así que para realizar esa comunicación fue necesario el desarrollo de un sistema para que gestione las rutas de la aplicación.

[4] Para más información consultar
https://davidcmg.github.io/batea/namespaces/davidcmg-batea-core-http.html

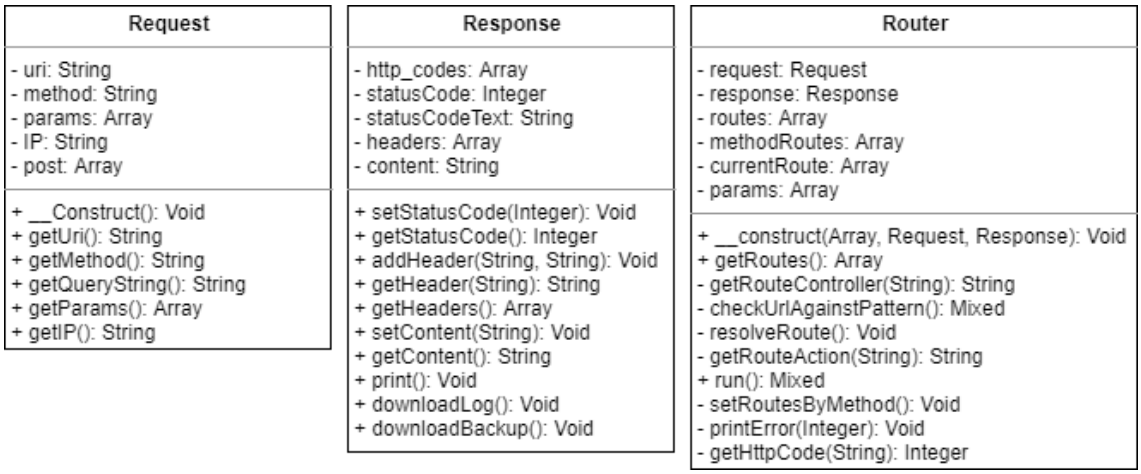

Fig. 7 Clases Request, Response y Router

3.2.2 Gestión de rutas

Ahora que tenemos una comunicación entre la web y el servidor, es necesario procesar las rutas que llegan a la aplicación. Para ello se implementa la clase Router.php[5] que es la encargada de gestionarlas. En primer lugar, es necesario definir las rutas con las que se quiere trabajar y para ello se define el archivo Routes.php.

En el archivo Routes.php se define un *array*, clasificando las rutas que gestionará la aplicación según el método HTTP. En el caso de esta aplicación, las agrupo entre el método GET, para solicitar datos y el método POST para enviar, crear o actualizar datos. De esta forma se logra tener un fichero fácil de gestionar por el desarrollador y a su vez fácil de manejar por el enrutador. Cada ruta la represento con una URI a la que **se asigna un controlador y un método** de éste para que sea el que gestione la lógica. En el caso de la figura siguiente (Fig. 8) la petición a '/admin/users' implica una llamada al controlador de usuarios (Users) y al método que contiene que muestra todos los usuarios del sistema (all). También se debe de tener en cuenta que, al ser una aplicación dinámica, era necesario implementar rutas dinámicas. Para ello añadí una etiqueta en la propia ruta para tener en cuenta estas variables (Fig. 9). De esta forma, los controladores pueden saber el valor de las variables que se pasan a través de la URL. En el caso del ejemplo, si se accede a la dirección /admin/users/1, la variable $id tendría el valor 1. El método *show*, sabe que tiene que mostrar la información relativa al usuario con id = 1.

```
'/admin/users' => ['davidcmg\batea\Controllers\Users', 'all'],
```

Fig. 8 Ejemplo de definición de ruta

[5] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Core-Router.html

```
'/admin/users/:id' => ['davidcmg\batea\Controllers\Users', 'show'],
```

Fig. 9 Ejemplo de definición de ruta con parámetro

A partir de esta lista de rutas, la clase Router se encargará de resolver la ruta que llega como una petición (Request) y la compara con el *array* de rutas. Si la ruta y el método existe, el Router llama al método del controlador correspondiente, con los parámetros -si los hubiere- con un código 200, que indica que la solicitud ha tenido éxito. En caso de que no exista la ruta, utiliza Response para devolver el código de error. Si la ruta no existe en el *array* de rutas, devuelve un error 404 (Fig. 10), que indica que no se encuentra el recurso solicitado; si no existe el controlador o el método del controlador, devuelve un error 500 (error interno del servidor) y 501 (no implementado), respectivamente. La clase Router, además, contiene otros métodos para su gestión, como pueden ser la obtención de variables de la ruta al compararla con la de su definición a través de una expresión regular.

Fig. 10 Error 404

3.2.3 El conector a la base de datos

Una vez resuelta la gestión de rutas, fue necesario implementar un conector a la base de datos. La clase SQLite[6] es la encargada de crear esta conexión, procesar y ejecutar las consultas. Para ello se hizo uso de la clase PDO de PHP, que representa la conexión entre el código y la base de datos.

[6] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Core-SQLite.html

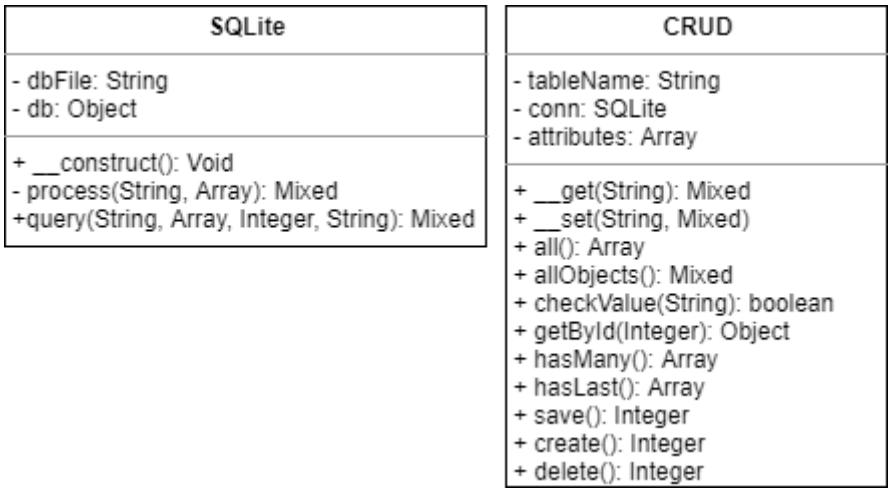

Fig. 11 Las clases SQLite y CRUD

La clase SQLite

Por lo tanto, una vez que se instancia la clase SQLite nos proporciona una **conexión a la base de datos**, y una vez que recibe una consulta, la procesa y la ejecuta. Las consultas se ejecutan llamando al método ***query***. Este método llama a *process* para preparar la consulta y vincular los parámetros que se le pasan, retornando un objeto PDOStament que contiene la consulta preparada. Dependiendo del tipo de consulta y de los argumentos que se indiquen en la llamada, devuelve un tipo de información distinta. Es decir, si se realiza una consulta tipo SELECT, esta clase va a devolver una colección de resultados o un objeto, dependiendo de los parámetros que se indiquen. En el caso de realizar una consulta INSERT, la consulta devuelve el último ID, para tener un control de la fila de la tabla sobre la que se realizó esta consulta. En el caso de realizar una sentencia UPDATE o DELETE, devolverá el número de filas que contiene la tabla. También se podría indicar si se desea obtener la información como objetos o estructurada en un *array*. Hay que tener en cuenta que SQLite3 tiene las restricciones de clave externa deshabilitadas y no permitía realizar borrados en cascada, por lo que en el constructor de la clase se añade una instrucción para habilitar esta característica.

La clase CRUD

Si bien la clase SQLite facilita mucho la realización de consultas a la base de datos, se implementa la clase CRUD[7] (*Create, Read, Update and Delete*) para implementar las funciones básicas que se necesitan implementar contra la base de datos. Es una clase abstracta de la que podrán extender los modelos de la aplicación. Así, una vez definidos los métodos básicos que deberían tener los modelos (creación, lectura, actualización y borrado en la base de datos), no es necesaria su implementación en cada uno de ellos.

Por otra parte, también se definen otros métodos de uso generalizado, como son los *getters* y *setters* para obtener y establecer atributos a las instancias

[7] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Core-CRUD.html

de los modelos, o el método *all(),* que devuelve un *array* con todas las entradas de la tabla asociada al modelo. El modelo de datos contiene relaciones 1 a n sobre las que se necesita realizar consultas. La clase CRUD contempla esta casuística con los métodos *hasMany()*, que retorna un array de los elementos en la relación, y *hasLast()*, que retorna el último elemento añadido en la relación.

3.2.4 Las vistas

Una vez procesada la información, es necesario mostrarla al usuario. Para ello se implementa la clase View[8]. Ésta es la clase que se encarga de renderizar las plantillas. Permite utilizar archivos parciales y asignar valores.

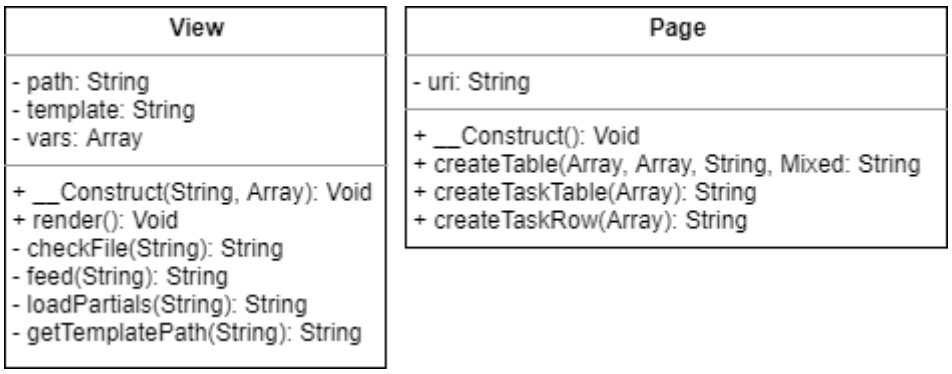

Fig. 12 Clase View y Page

Para ello establecí unas etiquetas para que la clase sepa cuando tiene que agregar el valor de una variable o un parcial. Así, cuando se instancia la clase View, se le pasa como parámetro los valores de las variables que se quieren mostrar. Por lo tanto, cuando se renderice la plantilla, se mostrará en ella el contenido de las variables previamente procesadas en el controlador.

Las plantillas, por tanto, contienen dos etiquetas que sirven como marcas para incluir contenido y parciales. Para el contenido de las variables se define la etiqueta **[@variable]**, la cual imprimirá el contenido de dicha variable en la plantilla. La otra etiqueta es **[#plantilla]**, la cual imprimirá el contenido de otra plantilla. Esta última resulta útil para fragmentos de la plantilla que se repiten varias veces, como pueden ser las cabeceras, menús, etc. La figura Fig. 13 es un ejemplo de la utilización de estas etiquetas. En las líneas 1 y 24 se incluye el contenido de un parcial para la cabecera de la web y para el pie. En las líneas 12, 13 y 15 se incluye el contenido de la variable *título*, *subtítulo* y *content*.

[8] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Core-View.html

```
[#partials/head.html]
    <div class="container-fluid landing" id="landing1">
        <div class="row">
            <div class="col-md-2"></div>
            <div class="col-md-8">
                <div class="jumbotron align-middle">
                    <div class="row">
                        <div class="col-md-4">
                            <div class="text-center"><img class="img-fluid" src="/img/logo.png"/></div>
                        </div>
                        <div class="col-md-8">
                            <h1 class="display-6">[@title]</h1>
                            <p class="lead">[@subtitle]</p>
                            <hr class="my-4">
                            <p>[@content]</p>
                            <a href="/login" type="button" class="btn btn-outline-secondary">Panel de control</a>
                        </div>
                    </div>
                </div>
            </div>
            <div class="col-md-2"></div>
        </div>
    </div>
[#partials/footer.html]
```

Fig. 13 Código de una plantilla

Pese a implementar estas etiquetas, en algunos casos necesité representar estructuras complejas, como tablas de datos, en las que se necesita un bucle para representar cada fila de ésta. Para ello se implementó la clase Page[9] como clase abstracta y que sirve como ayuda para la representación de estas estructuras.

3.2.5 Autenticación y seguridad

Por último, aunque quedan más clases en el Core de la aplicación, que pasaré a explicar más adelante, se implementó la clase Auth[10] para gestionar la autenticación y las sesiones del usuario.

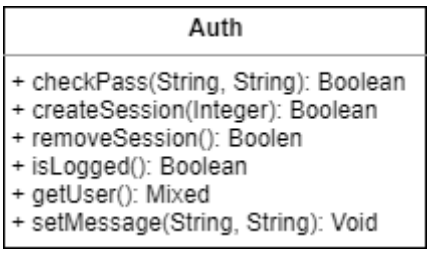

Fig. 14 Clase Auth

Vuelvo a utilizar la clase Page, vista en el apartado 3.2.4, para llamar a la clase Auth y así comprobar si el usuario está autenticado, evitando que un usuario ajeno pueda acceder a la información. De modo que todos los controladores que pertenezcan a la zona privada extienden de Page, así simplifico el proceso evitando comprobar si el usuario está autenticado en todos los controladores. Además, a la clase Auth se le añade la funcionalidad de establecer mensajes entre las páginas utilizando las sesiones, de forma que a las redirecciones entre páginas se pueda pasar información que será mostrada

[9] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Core-Page.html

[10] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Core-Auth.html

en la página destino. Un ejemplo de ello es ilustrado por la siguiente figura, en la que se muestra un mensaje tras la comprobación de las credenciales de un usuario.

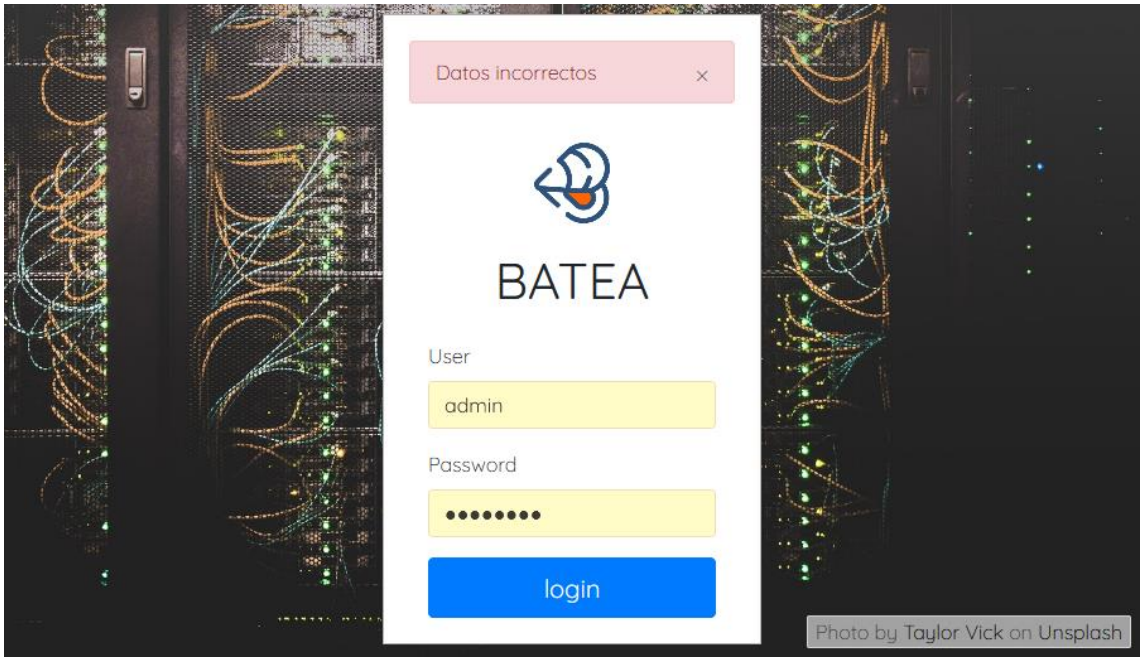

Fig. 15 Formulario de inicio de sesión con error

3.2.6 Los Modelos

Una forma de representar la información es mediante los Modelos, que se instanciarán para poder crear objetos y operar con ellos. Los Modelos[11] están en el directorio *src/Models* y todos ellos siguen una estructura similar. Tomando como ejemplo el modelo *User*[12], podemos observar que se definió la propiedad *tableName*. Esta propiedad se utiliza para definir el nombre de la tabla en la base de datos, y de esta forma, cuando se quiera realizar una consulta sobre la misma, la clase CRUD sepa en qué tabla buscar y poder construir las consultas a la base de datos de forma correcta. Además, en el constructor, se instancia una conexión a la base de datos con la clase SQLite.

A partir de esta estructura simple, hay modelos que tienen más especificaciones que las que puedan extender de CRUD. Un buen ejemplo de esto es el modelo que define un *Server* o una *Task*, que se verán más adelante.

[11] Para más información consultar https://davidcmg.github.io/batea/namespaces/davidcmg-batea-models.html

[12] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Models-User.html

3.2.7 Los controladores

Los Controladores[13] son llamados por las rutas e invocan a los Modelos. Así, para cada Modelo deseamos realizar tareas como la de mostrar, recibir y procesar formularios. Toda esta lógica se realiza desde estos controladores. Las siguientes funciones son comunes a la mayoría de los controladores:

- ***all()***: genera una tabla con todos los recursos solicitados y renderiza la plantilla.
- ***new()***: muestra un formulario para agregar información relativa al modelo sobre el que se esté trabajando.
- ***create()***: tras cumplimentar el formulario, éste se envía con el método Post a esta función, para procesar los datos y guardarlos en la base de datos.
- ***show()***: recupera un registro en la base de datos y se muestra.
- ***getEdit()***: recupera un registro de la base de datos y lo muestra en un formulario de edición.
- ***postEdit()***: modifica el registro en la base de datos de acuerdo con los cambios que se realizasen en el formulario de edición.
- ***delete()***: borra un registro de la base de datos.

3.2.8 Modelo físico de datos

Toda la implementación descrita en este apartado necesita un sistema para almacenar la información que va a almacenar la aplicación. En la figura Fig. 16 se muestra el modelo Entidad Relación que se desarrolló para esta aplicación.

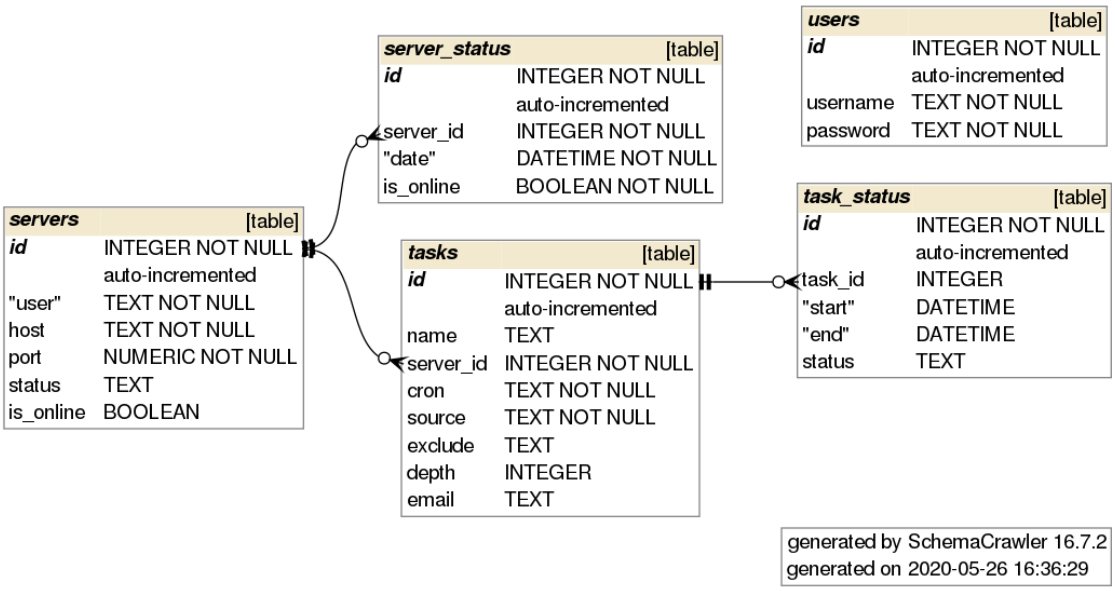

Fig. 16 Modelo Entidad Relación

Aparte de las credenciales del usuario, que se necesitan para poder iniciar sesión y que se guardan en la tabla *users*, se requieren de otras acciones. Es

[13] Para más información consultar https://davidcmg.github.io/batea/namespaces/davidcmg-batea-controllers.html

necesario guardar los datos del servidor sobre el que se quiere operar. Para cada servidor, se permite realizar tareas que tendrán una periodicidad definida por el usuario, las cuales ejecutan la tarea programada. Tanto los servidores como las tareas llevan asociada otra tabla (*servers* y *tasks*). En el caso de *servers*, está la tabla *server_status* para guardar el momento en el que un servidor deja de estar disponible o vuelve a estar en línea. Para *tasks* está la tabla *task_status*, que guarda el estado y otros parámetros de la ejecución de la tarea. *Task_status* representa, en definitiva, a una copia de seguridad.

Fig. 17 Jerarquía de operaciones

3.2.9 Otros aspectos del desarrollo

Para que la aplicación funcione correctamente en distintos equipos, se creó un **archivo de configuración**[14], en el que se pueden definir varios aspectos de éste, como la duración de las sesiones, la zona horaria o la localización de los directorios entre otros. Una de las constantes definidas es **APP_DEBUG**, que si se define como false no mostrará códigos de error ni se podrán utilizar las herramientas de desarrollo.

Pese a que la documentación de PHP es detallada y está bien estructurada, en momentos del desarrollo necesité utilizar software complementario para poder avanzar. *Xdebug* es una extensión de PHP para depurar el código y permite analizar el contenido de las variables y el flujo de ejecución. En determinadas ocasiones, para saber el contenido de cierta variable, estructura o el *backtrace* de determinado fragmento de código no me resultaba ni cómoda ni rápida la utilización de esta herramienta. Por eso, inspirándome en Laravel y otros frameworks de desarrollo, creé varios métodos para la ayuda en la codificación. El método *dd()*[15] muestra el contenido de una estructura y el método *tt()*[16] el backtrace en un punto determinado del código.

```
56 dd($servers, $ip, 'Hola mundo!');
```

Fig. 18 Llamada a dd()

[14] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Config.html

[15] Para más información consultar https://davidcmg.github.io/batea/namespaces/default.html#function_dd

[16] Para más información consultar https://davidcmg.github.io/batea/namespaces/default.html#function_tt

```
dd(🔑)
<?php
array (
  0 =>
  davidcmg\batea\Models\Server::__set_state(array(
     'tableName' => 'servers',
     'conn' =>
    davidcmg\batea\Core\SQLite::__set_state(array(
       'dbFile' => '/home/vagrant/code/batea/storage/db.dev.sqlite',
       'db' =>
      PDO::__set_state(array(
      )),
    )),
     'attributes' =>
    array (
      'id' => '1',
      'user' => 'usuario',
      'host' => 'localhost',
      'port' => '22',
      'status' => 'online',
      'is_online' => NULL,
    ),
     'connError' => NULL,
  )),
)
dd(🔑)
<?php
'127.0.1.1'
dd(🔑)
<?php
'Hola mundo!'


tt(📜)
#3 Admin.php(56): function dd
#2 Router.php(117): class davidcmg\batea\Controllers\Admin ⟹ function show
#1 bootstrap.php(36): class davidcmg\batea\Core\Router ⟹ function run
#0 index.php(3): function require_once
```

Fig. 19 Resultado de dd()

A medida que avanzaba con la escritura del código y **en paralelo**, iba escribiendo pruebas funcionales. Con este tipo de pruebas evaluaba y comprobaba el correcto funcionamiento del código. Este aspecto fue de decisivo, ya que tras cambios en el código o después de la escritura de una nueva funcionalidad, podía comprobar que seguía funcionando de forma correcta todo lo desarrollado anteriormente. Para la realización de estas baterías de pruebas utilicé Codeception [4], que permite una escritura descriptiva y cuyo aspecto hace que, a la hora de ejecutar una prueba, los resultados sean fáciles de interpretar incluso por una persona que no sepa programar (ver Fig. 21). Pese a la potencia del software, realicé las pruebas sobre el navegador sin interfaz gráfica *PhantomJS*, lo que me permitió, además, probar otras características de la web como las ventanas modales lanzadas en JavaScript. Las pruebas funcionales que se codificaron comprueban el correcto funcionamiento de:

- **FrontendCest**: se comprueba que la aplicación está funcionando.

```
- FrontendCest » View frontend 0.44s
I am on page "/"
```

I see "Trabajo Fin de Grado"

Tabla 17 Prueba de la página principal

- **LoginCest**: se comprueba que se accede con un usuario correcto, que gestiona un usuario incorrecto y muestra un error en las páginas protegidas.

- LoginCest » Login works 5.27s

I am going to restore database
I restore database
I am going to login with default user
I am on page "/"
I click "Panel de control"
I see "User"
I see "Password"
I am going to start a session
I don't see cookie "login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"
I see "Espacio libre"
I see cookie "BATEA"
I see current url equals "/admin"
I click "admin"
I click "Logout"
I see "User"
I see current url equals "/login"

- LoginCest » Login with incorrect user 3.02s

I am going to restore database
I restore database
I am on page "/login"
I fill field "//input[@type='text']","foo"
I fill field "//input[@type='password']","bar"
I click "login"
I see "Datos incorrectos"

- LoginCest » Access to protected pages 0.95s

I am going to restore database
I restore database
I am on page "/admin"
I see "403 - Forbidden"
I don't see cookie "login"

Tabla 18 Pruebas de acceso

- **UsersCest**: correcto funcionamiento de los formularios. Listar, añadir, ver, editar y borrar un usuario.

- UsersCest » View users page 3.57s

I login as "FunctionalTester","admin","secret"

I login as "FunctionalTester","admin","secret"

```
I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"
```

I am on page "/admin/users"

I see "Usuarios"

I see "admin","td"

- UsersCest » Add user 5.52s

I login as "FunctionalTester","admin","secret"

I login as "FunctionalTester","admin","secret"

```
I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"
```

I am on page "/admin/users"

I click "Añadir usuario"

I see current url equals "/admin/users/create"

I fill field {"name":"username"},"david"

I fill field {"name":"password"},"qwerty"

I click "Crear"

I see current url equals "/admin/users"

I see "david","td"

- UsersCest » View user 3.59s

I login as "FunctionalTester","admin","secret"

I login as "FunctionalTester","admin","secret"

```
I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"
```

I am on page "/admin/users"

I click "#btn-view-2"

I see element "input",{"value":"david"}

- UsersCest » Edit user 7.52s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

I am on page "/login" I fill field "//input[@type='text']","admin" I fill field "//input[@type='password']","secret" I click "login"

I am on page "/admin/users"
I click "#btn-edit-2"
I see "Cambiar contraseña del usuario 2"
I see element "input",{"value":"david"}
I fill field {"name":"password"},"123456"
I click "Cambiar contraseña"
I see "Contraseña modificada correctamente"
I login as "FunctionalTester","david","123456"
I login as "FunctionalTester","david","123456"

I am on page "/login" I fill field "//input[@type='text']","david" I fill field "//input[@type='password']","123456" I click "login"

I am on page "/"
I see "david"

- UsersCest » Delete user 5.08s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

I am on page "/login" I fill field "//input[@type='text']","admin" I fill field "//input[@type='password']","secret" I click "login"

I am on page "/admin/users"
I click "#btn-delete-2"
I wait 1
I see "¿Deseas borrar esta fila"
I click "Borrar"
I see "Correctamente"

Tabla 19 Pruebas de User

- **ServersCest**: se comprueba que se puede listar, añadir, ver, editar y borrar un servidor.

- ServersCest » View servers page 3.14s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"

I am on page "/admin/servers"
I see "Añadir servidor"

- ServersCest » Add server 8.46s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"

I am on page "/admin/servers"
I click "Añadir servidor"
I see current url equals "/admin/servers/create"
I fill field {"name":"host"},"localhost"
I fill field {"name":"port"},"22"
I fill field {"name":"user"},"root"
I fill field {"name":"password"},"1234"
I click "Crear"
I see current url equals "/admin/servers/create"
I see "Credenciales incorrectas"
I am on page "/admin/servers"
I see "localhost"
I see "online"

- ServersCest » View server 3.93s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"

I am on page "/admin/servers"
I click "#btn-view-1"
I see element "input",{"value":"22"}

- ServersCest » Edit server 4.89s

I login as "FunctionalTester","admin","secret"

I login as "FunctionalTester","admin","secret"

I am on page "/login" I fill field "//input[@type='text']","admin" I fill field "//input[@type='password']","secret" I click "login"

I am on page "/admin/servers"
I click "#btn-edit-1"
I see "Editar server"
I see element "input",{"value":"localhost"}
I fill field {"name":"host"},"127.0.0.1"
I click "Guardar cambios"
I see "correctamente"

- ServersCest » Delete server 5.21s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

I am on page "/login" I fill field "//input[@type='text']","admin" I fill field "//input[@type='password']","secret" I click "login"

I am on page "/admin/servers"
I click "#btn-delete-1"
I wait 1
I see "¿Deseas borrar este servidor"
I click "Borrar"
I see "Correctamente"

Tabla 20 Pruebas de Server

- **TasksCest**: se comprueba que se puede listar, añadir, ver, editar y borrar una tarea.

- TasksCest » View tasks page 3.06s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

I am on page "/login" I fill field "//input[@type='text']","admin" I fill field "//input[@type='password']","secret" I click "login"

I am on page "/admin/tasks"
I see "Añadir tarea"

- TasksCest » Add task 6.02s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

```
I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"
```

I am on page "/admin/tasks"
I click "Añadir tarea"
I see current url equals "/admin/tasks/create"
I fill field {"name":"name"},"Tarea de prueba"
I fill field {"name":"email"},"admin@localhost"
I select option {"name":"server"},"1"
I fill field {"name":"source"},"/origen/backup/"
I fill field {"name":"depth"},"0"
I fill field {"name":"cron"},"* * * * *"
I fill field {"name":"exclude"},".gitignore"
I click "Crear"
I see current url equals "/admin/tasks"
I see "Tarea Tarea de prueba creada correctamente"

- TasksCest » View task 3.82s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

```
I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"
```

I am on page "/admin/tasks"
I click "#btn-view-1"
I see element "input",{"value":"Tarea de prueba"}

- TasksCest » Edit task 5.04s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

```
I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"
```

I am on page "/admin/tasks"
I click "#btn-edit-1"
I see "Editar tarea"
I see element "input",{"value":"Tarea de prueba"}

I fill field {"name":"email"},"mail@localhost"
I click "Guardar"
I see "correctamente"

- TasksCest » Delete task 5.28s

I login as "FunctionalTester","admin","secret"
I login as "FunctionalTester","admin","secret"

I am on page "/login"
I fill field "//input[@type='text']","admin"
I fill field "//input[@type='password']","secret"
I click "login"

I am on page "/admin/tasks"
I click "#btn-delete-1"
I wait 1
I see "¿Deseas borrar esta tarea"
I click "Borrar"
I see "Correctamente"

Tabla 21 Pruebas de Task

Para facilitar la ejecución de estas pruebas, se programó un *helper*, que copia y a continuación borra el contenido de la base de datos. De esta forma se tiene controlado el entorno y se asegura que siempre sea el mismo. Al finalizar las pruebas se renombra la copia que se hizo antes de las mismas para poder seguir con el desarrollo de la aplicación.

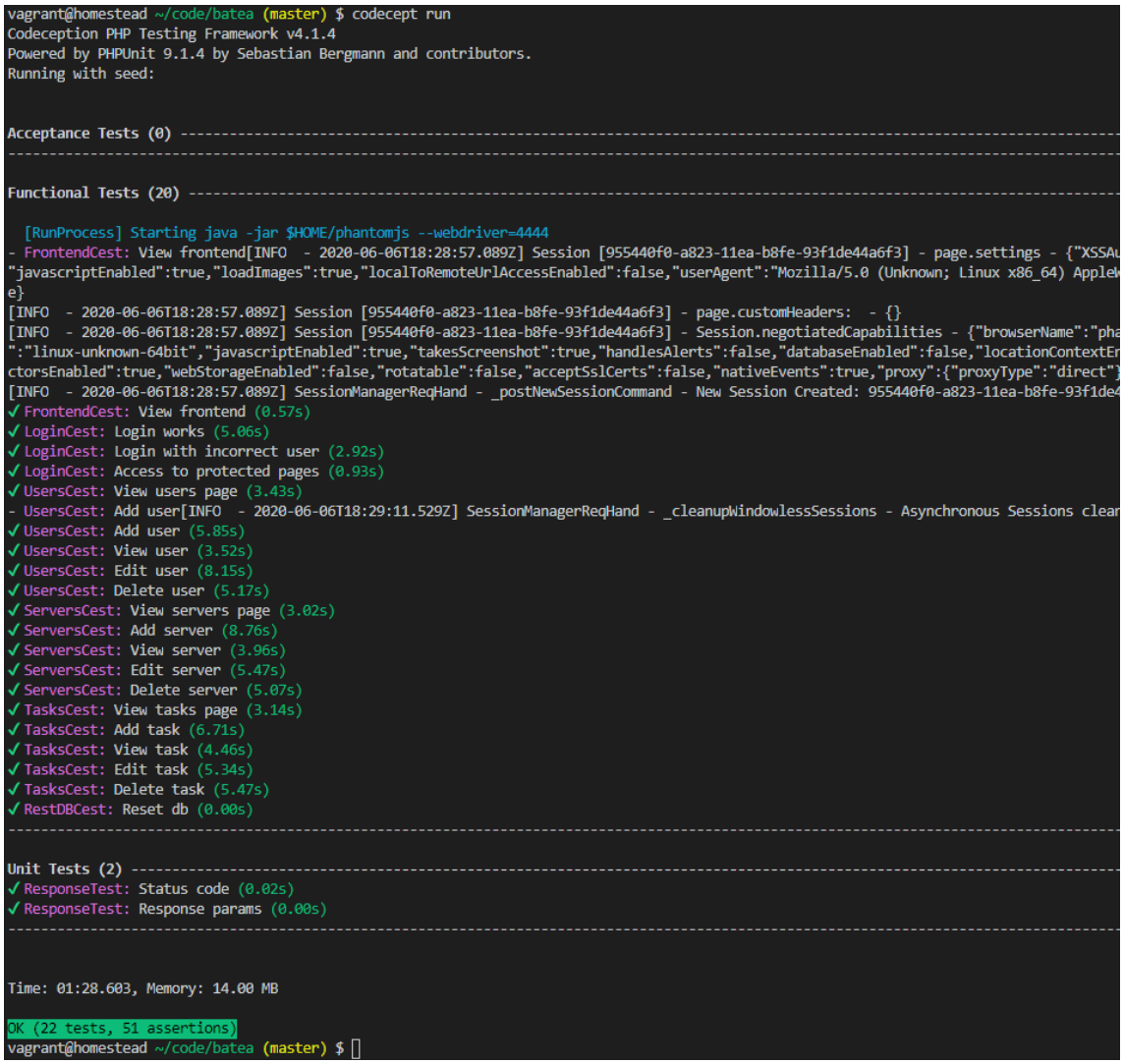

Fig. 20 Ejecución de batería de pruebas

```php
    public function addUser(FunctionalTester $I)
    {
        $I->loginAs($I, 'admin', 'secret');
        $I->amOnPage('/admin/users');
        $I->click('Añadir usuario');
        $I->seeCurrentUrlEquals('/admin/users/create');
        $I->fillField(['name' => 'username'], 'david');
        $I->fillField(['name' => 'password'], 'qwerty');
        $I->click('Crear');
        $I->seeCurrentUrlEquals('/admin/users');
        $I->see('david', 'td');
    }
```

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS

UsersCest: Add user
Signature: UsersCest:addUser
Test: tests/functional/C_UsersCest.php:addUser
Scenario --
 I login as "FunctionalTester","admin","secret"
 I login as "FunctionalTester","admin","secret"
   I am on page "/login"
   I fill field "//input[@type='text']","admin"
   I fill field "//input[@type='password']","secret"
   I click "login"
 I am on page "/admin/users"
 I click "Añadir usuario"
 I see current url equals "/admin/users/create"
 I fill field {"name":"username"},"david"
 I fill field {"name":"password"},"qwerty"
 I click "Crear"
 I see current url equals "/admin/users"
 I see "david","td"
 PASSED
```

Fig. 21 Extracto de ejecución de una prueba con su código fuente

Para completar la ayuda, inserté en cada clase y en cada método un bloque de texto comentado. El texto es un resumen de lo que implementa el código que le sigue en formato *PHPDoc*. Utilizando este formato, además de definir el código, se puede complementar con una serie de etiquetas, como pueden ser los parámetros o el valor de retorno. Una vez escrita esta documentación, con el programa *phpDocumentor* se exporta a HTML[17]. El material generado, además de ayudarme en el desarrollo, lo utilizo para escribir y documentar esta memoria.

[17] Contenido publicado en https://davidcmg.github.io/batea/

```
/**
 * Comprueba que una tarea se puede ejecutar según el cron.
 * Para cada tarea guardo la programación en variables separadas, representando
 * cada una, una parte del cron, con el siguiente formato:.
 *
 *      .--------------- minuto (0-59)
 *      |  .------------ hora (0-23)
 *      |  |  .--------- día del mes (1-31)
 *      |  |  |  .------ mes (1-12) o jan,feb,mar,apr,may,jun,jul... (meses en inglés)
 *      |  |  |  |  .--- día de la semana (0-6) (domingo=0 ó 7) o sun,mon,tue,wed,thu,fri,sat (días en inglés)
 *      |  |  |  |  |
 *      *  *  *  *  *
 *
 * A continuación se define un array con el formato de cada variable, para
 * extraerlo de la hora actual y poder hacer la comparación.
 *
 * @author David Campos <davidcmg@uoc.edu>
 *
 * @param array $cron Array con la fecha programada
 * @param int   $time Fecha actual
 *
 * @return boolean
 */
```

Fig. 22 Ejemplo de bloque de comentario

3.3 El gestor de copias de seguridad Batea

En el apartado anterior hablé de la construcción de la plataforma Batea como un *framework*. Una vez realizado ese trabajo, ya disponía de una plataforma y de las herramientas para el desarrollo de la idea final del presente proyecto. Así, después de la implementación anterior, me sirvió como base para un desarrollo más rápido y cómodo.

Hecho esto, continuaré con los engranajes que quedaban por analizar dentro del *Core* del *framework*. Éstas son clases para gestionar el tiempo de las tareas, los logs del sistema, la interacción con la máquina que hospeda la plataforma y, por último, el almacenamiento.

3.3.1 Cron

Los sistemas operativos basados en GNU/Linux cuentan con un administrador de procesos en segundo plano llamado *Cron*, que se ejecuta como un servicio y permite ejecutar tareas cada cierto tiempo previa definición del usuario. La forma de especificar las tareas sigue el siguiente esquema:

```
┌──────────── minuto          (0 - 59)
│ ┌────────── hora            (0 - 23)
│ │ ┌──────── día             (1 - 31)
│ │ │ ┌────── mes             (1 - 12)
│ │ │ │ ┌──── día semana      (0 - 6)
│ │ │ │ │
* * * * * comando a ejecutar
```

Utiliza el asterisco * para indicar que toma todos los valores posibles, siendo la traducción para “* * * * *” una ejecución cada segundo. Otro ejemplo es “20 12 4 3 3” siendo la traducción que una tarea se ejecute a las 12:20 del día 4 de marzo y que sea miércoles. Dichos parámetros también pueden tomar una lista de valores separados por comas (1,30 * * * * se ejecutaría en el minuto 1 y 30), un rango de valores (1-10 * * * * se ejecutaría cada minuto entre el minuto 1

y el minuto 10), separados por una barra para mostrar fracciones (*/15 * * * * se ejecutaría cada 15 minutos). Para resolver esta casuística implementé la clase Cron[18]. Con esta clase se puede comprobar si la programación definida en una tarea de *backup* (clase Task) coincide con la hora actual del sistema, y así saber si se debe o no lanzar un proceso para realizar la copia de seguridad.

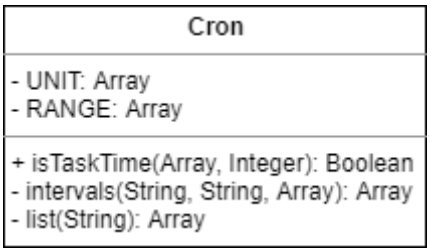

Fig. 23 Clase Cron

El sistema por sí solo no se ejecuta y es necesario integrarlo en el Cron del sistema en el que se instale. Como interesa que compruebe las tareas cada segundo, la configuración es similar a la de la siguiente figura (Fig. 24), en la que se especifica el intervalo de ejecución y el código a ejecutar. También se redirige la salida que se pueda obtener de la ejecución al dispositivo null, de esta forma la ejecución quedaría silenciada.

```
sudo crontab -u www-data -e
* * * * *  /usr/bin/php /full/path/to/project/batea.cron.php >> /dev/null 2>&1
```

Fig. 24 Ejemplo de configuración de Cron

3.3.2 Log

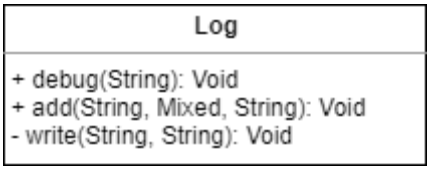

Fig. 25 Clase Log

La ejecución de las tareas para las copias de seguridad genera mucha información que no conviene descartar. Ante posibles problemas es necesario disponer de un diario con información para su revisión. Para manejar esta información, implementé la clase Log[19], que guarda y clasifica un texto en un archivo, con la hora del sistema y una etiqueta para su clasificación. Además, para facilitar la localización de la información, el fichero cambia cada día, por lo que cada uno de ellos contiene la información relativa a ese día. Para el desarrollo también fue necesario contar con los logs, ya que las tareas programadas no cuentan con una interfaz en la cual pueda verificar la correcta

[18] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Core-Cron.html

[19] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Core-Log.html

ejecución. Para ello está el método *debug*, que utiliza un solo fichero y no requiere una etiqueta como parámetro.

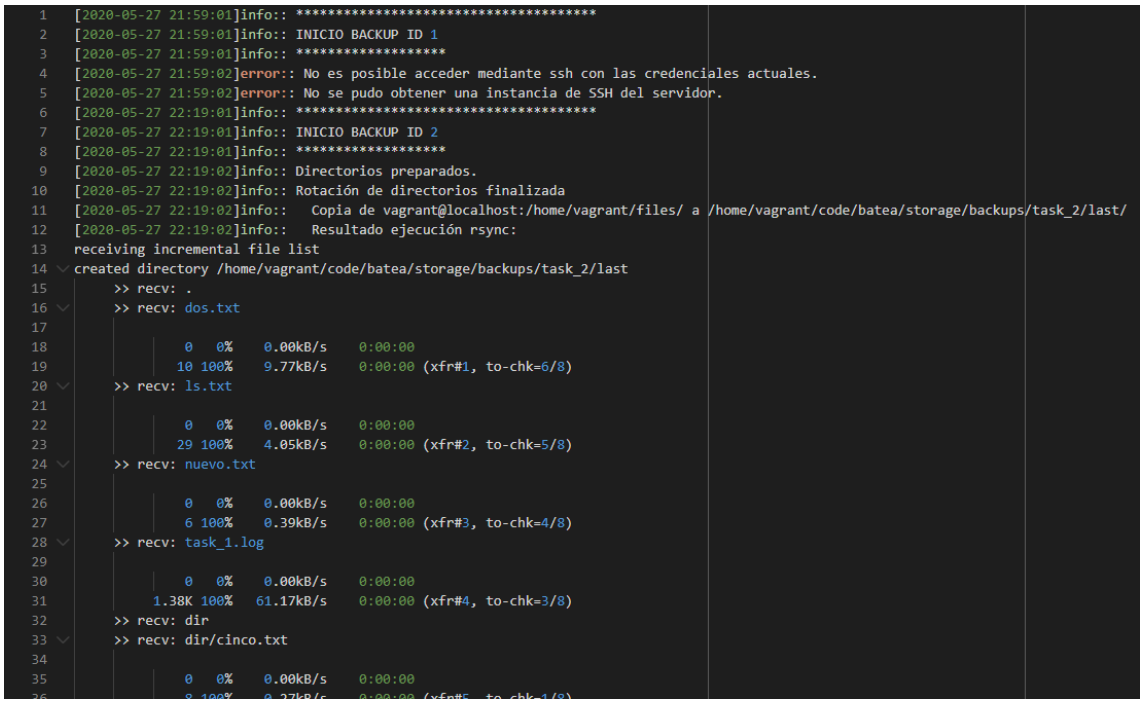

Fig. 26 Extracto de un archivo log

3.3.3 Shell

Un aspecto importante en esta aplicación es la unión del software desarrollado y la máquina en sí. Con esta clase logramos poder acceder a la Shell del sistema e interactuar con ella ejecutando comandos. Para lograrlo se establece una tubería con punteros de lectura y escritura y al final se retorna la salida de la ejecución. Es clave en este proyecto para poder ejecutar Rsync y realizar así las copias de seguridad.

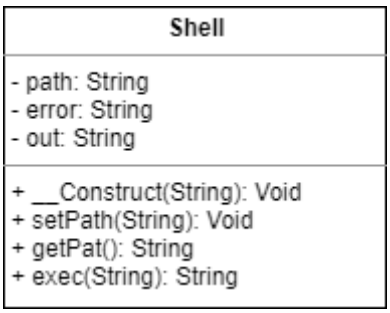

Fig. 27 Clase Shell

3.3.4 Storage

Para concluir con el núcleo de la aplicación, está la clase Storage, creada para manejar las descargas de contenido almacenado, y que así no esté expuesto en Internet. Para lograrlo se controla que el usuario que solicita un

recurso esté autenticado, y que además la petición esté bien formada, evitando así exponer contenido que pueda ser susceptible de un ataque.

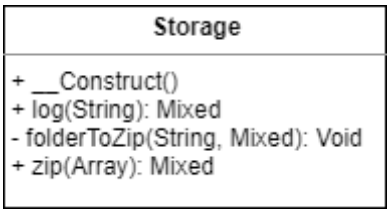

Fig. 28 Clase Storage

La gestión de estas descargas comienza con la definición de las rutas para ello, y que así puedan gestionarse en esta clase. Con el constructor se añade una llamada a Auth, para comprobar que la petición viene de un usuario autenticado. En los métodos log y zip se comprueba que la petición solamente contiene caracteres, incrementando la seguridad, evitando así ataques *SQL injection*. Una vez la descarga está disponible se hace uso de la clase Response, en la que se definen métodos para las descargas, añadiendo cabeceras específicas para ello asegurando una correcta descarga en el navegador.

3.3.5 Modelos y Controladores

```php
<?php

return [
    'GET' => [
        '/' => ['davidcmg\batea\Controllers\Index', 'show'],
        '/login' => ['davidcmg\batea\Controllers\Login', 'show'],
        '/logout' => ['davidcmg\batea\Controllers\Login', 'logout'],
        '/admin' => ['davidcmg\batea\Controllers\Admin', 'show'],
        /*============ DESCARGAS ============*/
        '/files/log/:folder/:file' => ['davidcmg\batea\Core\Storage', 'log'],
        '/files/log/:folder/:subfolder/:file' => ['davidcmg\batea\Core\Storage', 'log'],
        '/files/backup/:folder/:subfolder' => ['davidcmg\batea\Core\Storage', 'zip'],
        /*============== USERS ==============*/
        '/admin/users' => ['davidcmg\batea\Controllers\Users', 'all'],
        '/admin/users/:id' => ['davidcmg\batea\Controllers\Users', 'show'],
        '/admin/users/create' => ['davidcmg\batea\Controllers\Users', 'new'],
        '/admin/users/:id/edit' => ['davidcmg\batea\Controllers\Users', 'getEdit'],
        /*============= SERVERS =============*/
        '/admin/servers' => ['davidcmg\batea\Controllers\Servers', 'all'],
        '/admin/servers/:id' => ['davidcmg\batea\Controllers\Servers', 'show'],
        '/admin/servers/create' => ['davidcmg\batea\Controllers\Servers', 'new'],
        '/admin/servers/:id/edit' => ['davidcmg\batea\Controllers\Servers', 'getEdit'],
        /*============== TASKS ==============*/
        '/admin/tasks' => ['davidcmg\batea\Controllers\Tasks', 'all'],
        '/admin/tasks/:id' => ['davidcmg\batea\Controllers\Tasks', 'show'],
        '/admin/tasks/create' => ['davidcmg\batea\Controllers\Tasks', 'new'],
        '/admin/tasks/:id/edit' => ['davidcmg\batea\Controllers\Tasks', 'getEdit'],
    ],
    'POST' => [
        '/login' => ['davidcmg\batea\Controllers\Login', 'check'],
        /*============== USERS ==============*/
        '/admin/users/create' => ['davidcmg\batea\Controllers\Users', 'create'],
        '/admin/users/:id/edit' => ['davidcmg\batea\Controllers\Users', 'postEdit'],
        '/admin/users/:id/delete' => ['davidcmg\batea\Controllers\Users', 'delete'],
        /*============= SERVERS =============*/
        '/admin/servers/create' => ['davidcmg\batea\Controllers\Servers', 'create'],
        '/admin/servers/:id/edit' => ['davidcmg\batea\Controllers\Servers', 'postEdit'],
        '/admin/servers/:id/delete' => ['davidcmg\batea\Controllers\Servers', 'delete'],
        /*============== TASKS ==============*/
        '/admin/tasks/create' => ['davidcmg\batea\Controllers\Tasks', 'create'],
        '/admin/tasks/:id/edit' => ['davidcmg\batea\Controllers\Tasks', 'postEdit'],
        '/admin/tasks/:id/delete' => ['davidcmg\batea\Controllers\Tasks', 'delete'],
        '/admin/tasks/:id/:depth/restore' => ['davidcmg\batea\Controllers\Tasks', 'restore'],
    ],
];
```

Fig. 29 Archivo de rutas

Desde el panel de control podemos gestionar los datos. Como norma general, para hacer que un sitio sea dinámico, se tiene que permitir la creación, edición, actualización y borrado de los datos. Esta funcionalidad general se detalló en la clase CRUD (ver Fig. 11) más arriba, siendo ahora necesario hacer que el usuario pueda interactuar con estas funcionalidades. Entran aquí en juego el Router (ver Fig. 7) y los Controladores[20]: Cuando el usuario solicita una URI, el Router busca en el archivo de rutas (Fig. 29) el recurso solicitado para así llamar al Controlador y al método correspondiente. Aquí es en dónde se realiza la lógica del programa, interactuando con los Modelos, que tienen definida la

[20] Para más información consultar https://davidcmg.github.io/batea/namespaces/davidcmg-batea-controllers.html

gestión de los datos sobre los que se opera. En el caso de las copias de seguridad se implementaron las siguientes características que son de relevancia:

Server

Para acceder a las máquinas remotas se utiliza un sistema de criptografía asimétrica. Este método criptográfico utiliza un par de claves, una privada, que solo conoce el propietario, y otra pública, que se puede entregar. Si bien se puede utilizar el par de claves del equipo en el que se instale la aplicación, se recomienda crear otro par para uso exclusivo de este programa. Previendo esto se crea un directorio *keys/* para guardar estas claves o bien se puede configurar su ubicación desde el archivo de configuración. Con este sistema podemos acceder a los servidores que tengamos configurados sin necesidad de utilizar una contraseña, así que se implementa una función para copiar la clave pública del servidor al servidor remoto para así obtener acceso mediante la clave privada (Fig. 30). Esta función está implementada como una característica del modelo Server[21] en el método *copyPublicKey()*. Para ello fue necesario utilizar la librería criptográfica *phpseclib* que solucionó dos problemas: la gestión de claves RSA y las conexiones SSH2. Aunque el objetivo del proyecto es no depender de proyectos externos, el desarrollo de estas características sería inabarcable para realizar durante la duración del presente trabajo. Además, en la clase server se implementan funcionalidades para acceder con contraseña, necesario para una primera conexión y copiar la clave; comprobar que el servidor está en línea; obtener una instancia SSH; y añadir un estado a la base de datos con el que podemos llevar un seguimiento de las caídas del servidor.

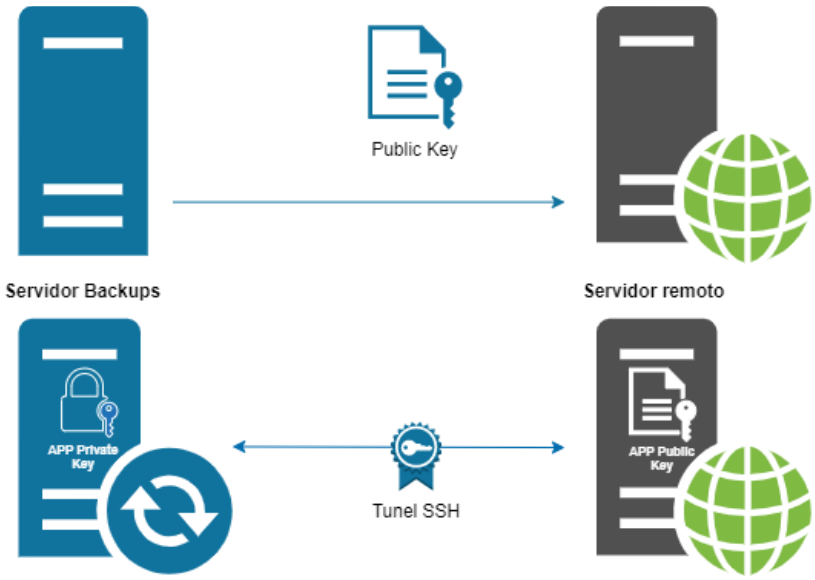

Fig. 30 Acceso mediante claves

[21] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Models-Server.html

Task

La segunda característica a destacar es la realización de copias de seguridad del modelo Task[22]. En este modelo adquieren relevancia las clases Log y Shell, ya que es necesario acceder al sistema para rotar las copias de seguridad, ejecutar Rsync y dejar constancia en los archivos .log. En el caso de que la tarea se ejecute según lo previsto, se actualiza el estatus con los datos de inicio y final. Este estatus además representa una copia de seguridad realizada con éxito. En la figura Fig. 31 se muestra un diagrama de secuencia con la ejecución de una tarea. Como en el caso anterior, utilicé la librería PHPMailer, ya que, aunque se pueden enviar correos directamente desde PHP, era necesario otro sistema porque por defecto, al no tener configurado un SMTP, proveedores como Gmail, directamente no reciben los correos. Intentar solventar este problema excedía el tiempo de desarrollo del proyecto y no era viable.

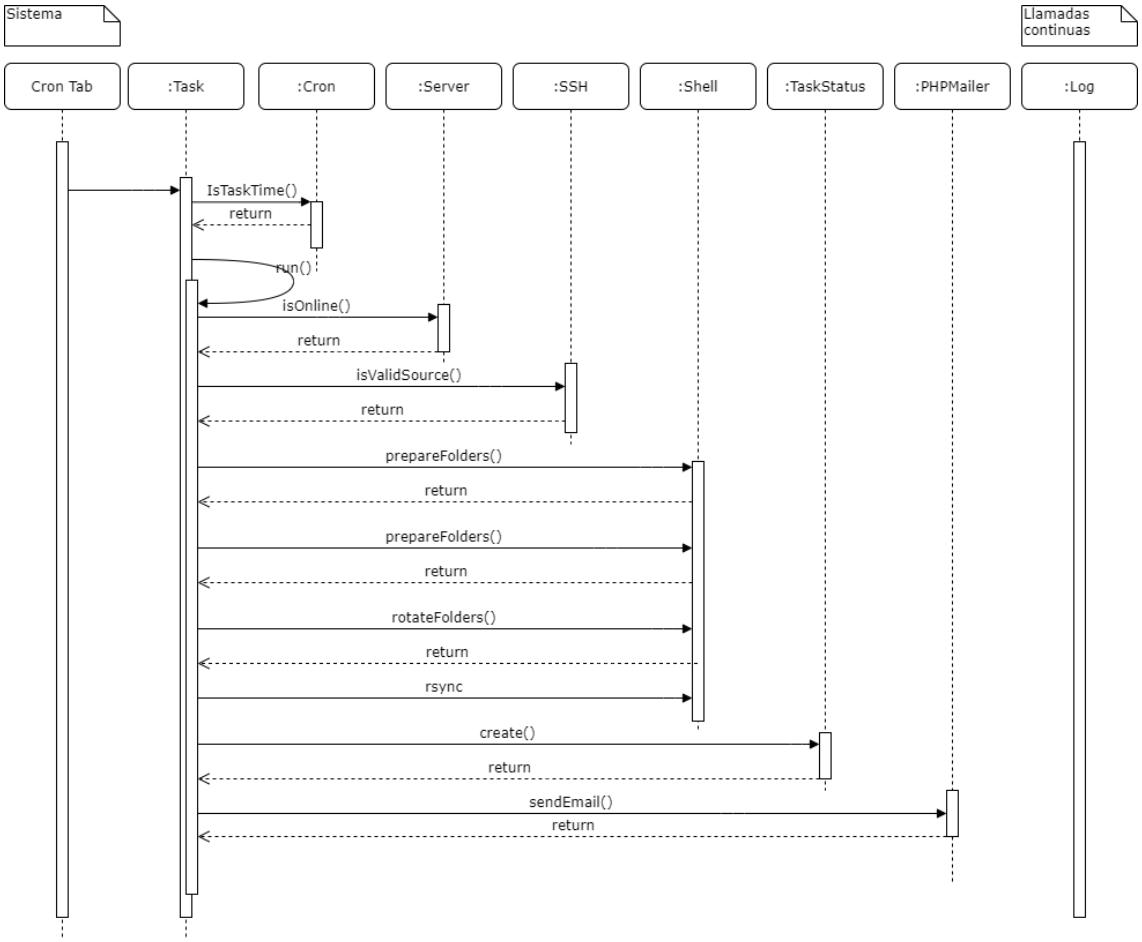

Fig. 31 Diagrama secuencia Task

[22] Para más información consultar https://davidcmg.github.io/batea/classes/davidcmg-batea-Models-Task.html

3.4 Interfaz de usuario

3.4.1 Herramientas

Para el maquetado y diseño de la web utilicé Bootstrap [7]. Éste *framework* se emplea para desarrollar páginas adaptativas y utiliza "(…) una estrategia en la que *primero* optimizamos el código para dispositivos móviles y luego escalamos los componentes según sea necesario mediante consultas de medios CSS." Cuenta con componentes, opciones y un sistema de cuadrículas flexibles para el diseño del proyecto. Algunos de los componentes que integra requieren el uso de JavaScript para funcionar, como pueden ser las ventanas modales.

También se añade código JavaScript, así como la librería **jqCron**[23] para convertir el formato que utiliza Cron a uno más amigable. Para procesar los archivos **CSS** y **JavaScript**, hice uso de Npm para gestionarlos mediante Webpack y otras dependencias. Me inspiré en Laravel Mix para crear los scripts, de modo que con el comando npm run production se comprime el código para la web en producción, o con npm run watch, se detectan los cambios en CSS o JavasCript y se actualiza automáticamente el navegador con los cambios. Además, para evitar la caché del navegador con los estilos, una vez que éstos se actualizan, se guardan con un nombre distinto que la propia aplicación sabe cómo buscar.

No quería olvidarme de otras herramientas que me ayudaron además en el desarrollo del código. **Git** es un gestor de control de versiones que se utiliza para registrar cambios en el código. **Visual Studio Code** es un editor de código con soporte a muchos lenguajes de programación con características como el resaltado de sintaxis o la depuración. Además, instalé las siguientes extensiones: Bracket Pair Colorizer 2, Colonize, PHP CS Fixer, PHP Debug, PHP DocBlocker, PHP IntelliSense, Remote SSH. Para la realización de los diagramas y demás gráficos utilicé la herramienta **draw.io**. Por último, InkScape es un editor de gráficos vectoriales Open Source, que se utilizó para la creación del logo.

3.4.2 Diseño

Para la realización de la interfaz de la web me decanté por un diseño simple siguiendo la filosofía del proyecto. En la interfaz se pueden distinguir tres partes claramente diferenciadas. (1) El menú superior en el que figura el logo y el usuario contiene un reloj que indica la hora local del servidor. Aunque la aplicación se puede configurar con el huso horario deseado, es interesante tener presente la hora del sistema, ya que será la que se utilice para las tareas programadas. (2) A la izquierda hay una barra lateral en la que se indican los apartados de la web. La última parte es la central, en la que se muestra el contenido de cada apartado. Para obtener un resultado más visual e intuitivo, utilizo el pack de iconos Boxicons[24]. Además, todas las acciones que conlleven

[23] Para más información consultar https://github.com/arnapou/jqcron
[24] Para más información consultar https://boxicons.com/

un cambio en la base de datos, como son añadir, modificar o borrar, se completan con un mensaje de confirmación o de error en caso de que surgiera un impedimento (Fig. 32).

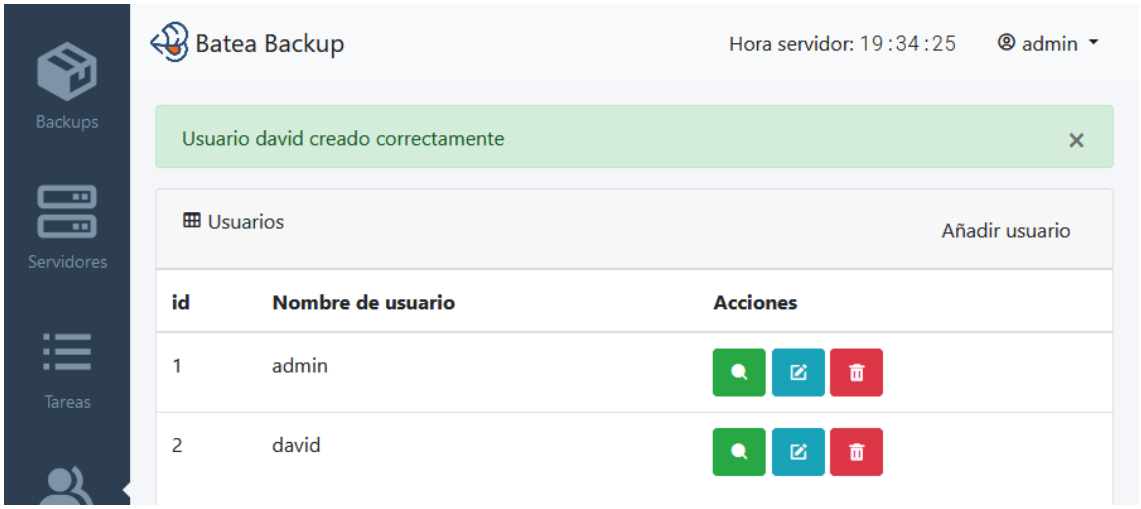

Fig. 32 Mensaje de confirmación

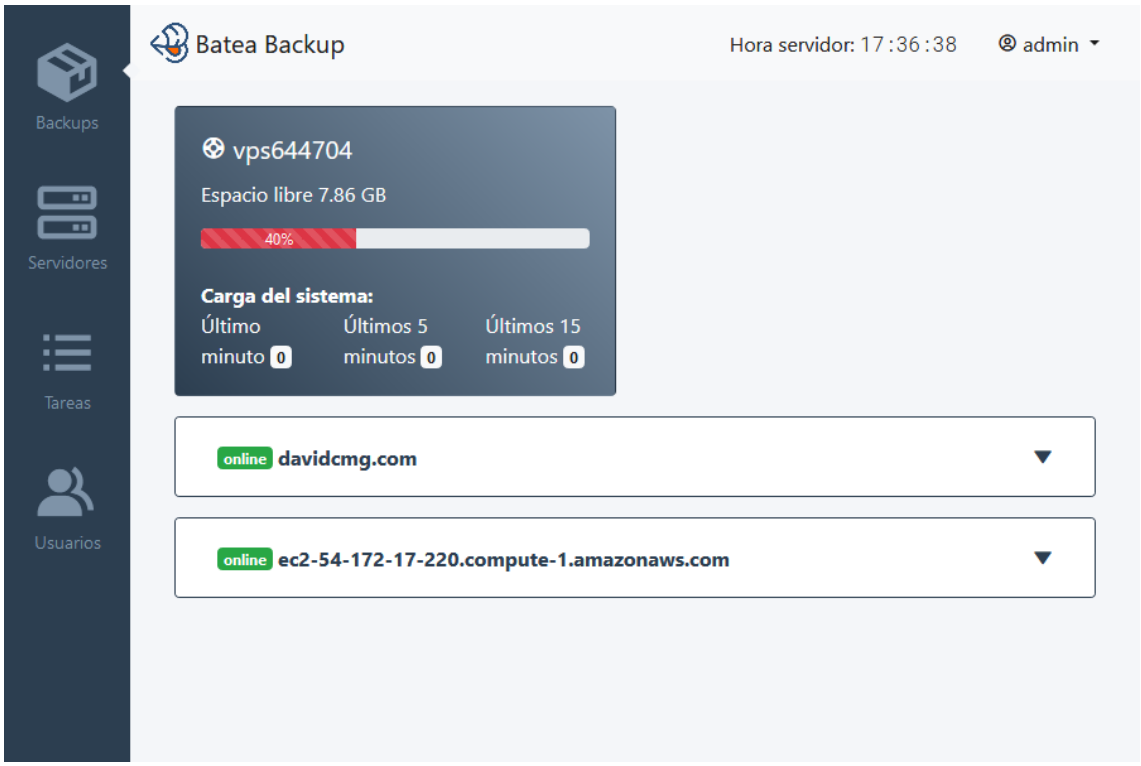

Fig. 33 Página principal del panel de control

La página principal sirve de panel de control de la aplicación. En la parte superior se muestra el nombre del servidor que aloja la aplicación, el espacio libre de la máquina acompañado por un gráfico y la carga del sistema, como se observa en la figura Fig. 33.

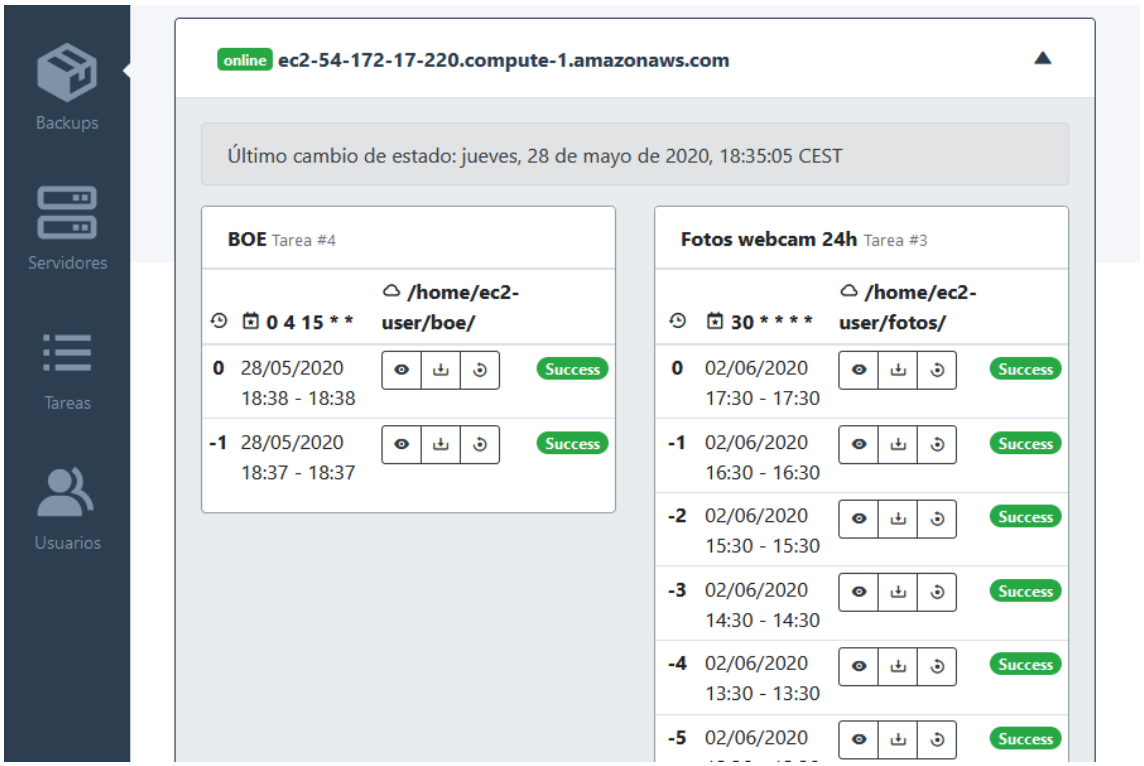

Fig. 34 Copias realizadas en un servidor

A continuación, se listan todos los servidores remotos que hay configurados en la aplicación con información del último estado (el servidor remoto está o deja de estar disponible), y seguidamente las tareas que se realizan en esa máquina con un listado de todas las copias y su estado (Fig. 34). Además de consultar la fecha de la copia de seguridad, su hora de inicio y de finalización, podemos interactuar mediante tres botones. 👁 Botón de descarga del log de la copia de seguridad seleccionada. 🡻 Descarga del contenido de la copia de seguridad en un archivo comprimido. ↻ Restaura la copia de seguridad (Fig. 35). Si se realiza una restauración de una copia de seguridad, la aplicación pide una confirmación ya que se sobreescribirán los datos. Una vez terminada esta acción, se muestra un resumen de los datos copiados en caso de que la restauración sea exitosa (Fig. 36) o un mensaje de error en caso de que sea errónea.

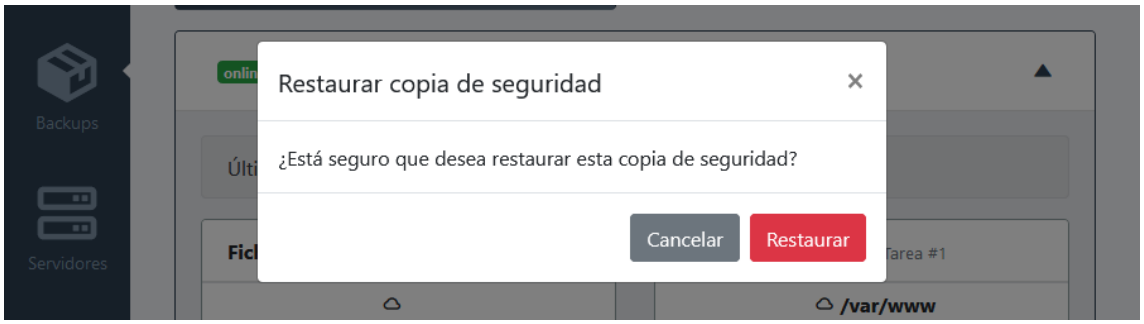

Fig. 35 Ventana de confirmación

Fig. 36 Resultado de restaurar una copia

El resto de los apartados gestiona los usuarios del sistema, los servidores remotos sobre los que se quiere hacer un respaldo y las tareas para las copias de seguridad. Todos siguen el mismo formato, siendo la página principal de la entidad (servidores, tareas o usuarios) un listado de éstos. Cada fila tiene cuatro acciones disponibles. La primera acción es la de añadir, que lleva a un formulario de ingreso de datos para su posterior inserción en la base de datos. Por cada fila tenemos las otras tres acciones disponibles. La segunda acción muestra el contenido completo de la entrada. La tercera acción muestra un formulario para editar los datos. La cuarta acción borra la fila de la base de datos previa confirmación con una ventana modal como la que se muestra en la figura Fig. 35.

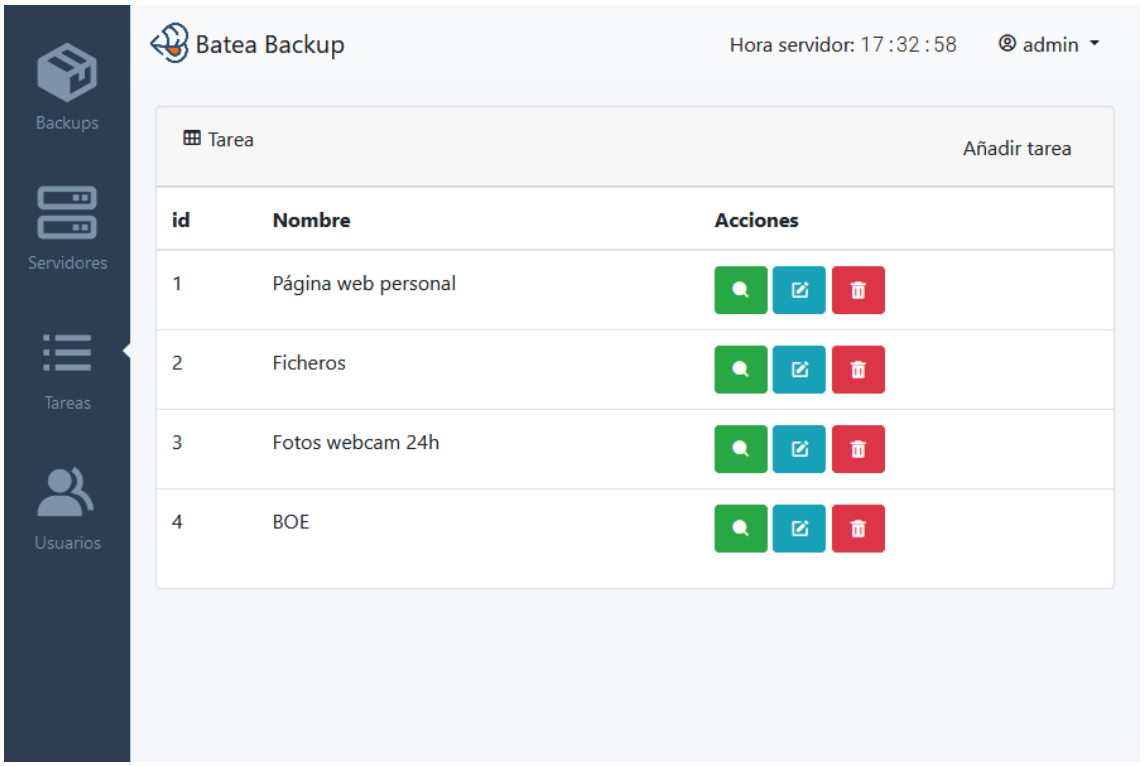

Fig. 37 Listado de tareas

El formato de añadir o de modificar una tarea difiere levemente de los demás. En estas páginas se utiliza el plugin **jqCron** que transforma el formato Cron a un formato más amigable para poder trabajar de una forma mucho más cómoda y que no dé lugar a equívocos. Este plugin actualiza automáticamente el campo del formulario y viceversa. También permite la selección de varios valores y rangos.

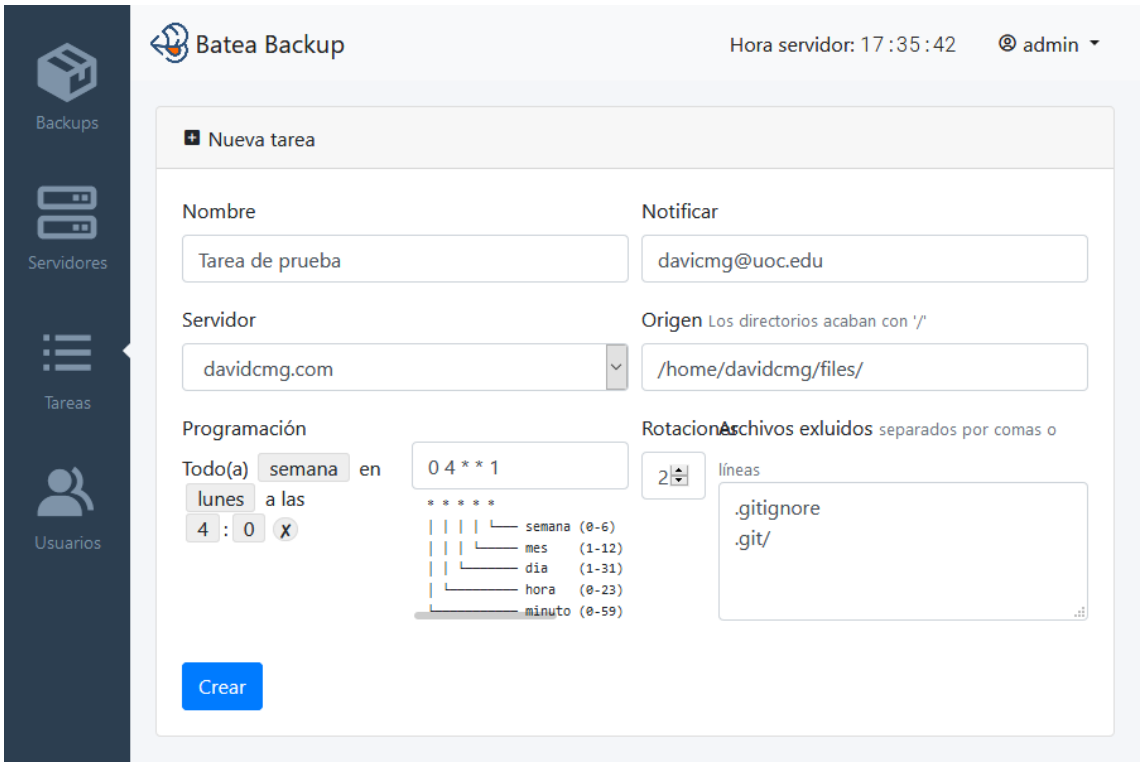

Fig. 38 Formulario nueva tarea

Para representar el producto creado, diseñé un logo con el juego de palabras “Batea”, que representa un mejillón. Para su realización empleé la herramienta de diseño vectorial InkScape, adaptando la imagen de la figura publicada en svgrepo.com[25] con licencia de dominio público. Las imágenes de la página principal[26] y del formulario de ingreso[27] fueron descargadas de la web unsplash.com con una licencia Unsplash.

Fig. 39 Logo del proyecto

[25] Vector disponible en la web https://www.svgrepo.com/svg/223509/mussel
[26] La imagen original está disponible en https://unsplash.com/photos/QURU8IY-RaI
[27] La imagen original está disponible en https://unsplash.com/photos/M5tzZtFCOfs

4 Conclusiones

Realizar el trabajo de fin de grado supone una experiencia en la que confluyen los aspectos educativos con los profesionales, ofreciendo la oportunidad de llevar a la práctica los aprendizajes adquiridos durante el grado. Si bien es cierto que la situación de estado de alarma derivada de la crisis del COVID-19 ha generado problemáticas a nivel discente, familiar y personal; la gestión ha sido satisfactoria y los resultados positivos, gracias a la comunicación continua y apoyo de mi tutor Gregorio Robles Martínez.

Lecciones aprendidas:

- La realización de pruebas automáticas es un factor esencial para el desarrollo de software.
- La documentación y estructuración del código.

Se han logrado todos objetivos planteados al inicio del trabajo y el resultado ha sido satisfactorio. El desarrollo de algunos componentes, como puede ser el renderizado de las vistas o los métodos de autenticación, necesitarían más desarrollo, pero es inabarcable poder implementar todas las características necesarias para una funcionalidad completa en el tiempo dispuesto para la realización del trabajo. Aun así, los resultados han sido satisfactorios y se han alcanzado los objetivos.

Se ha podido seguir la planificación con una adaptación. El diseño de la interfaz de usuario estaba planeado para realizar como última tarea, pero a la hora de realizar pruebas durante el desarrollo de parte de la aplicación, se necesita contar con parte de la interfaz. Se ha realizado un prototipo simple de la interfaz para poder solventar este inconveniente y en la última fase se profundiza en el diseño.

Una vez finalizado el producto, me he dado cuenta de que sería interesante continuar añadiendo funcionalidades, que serían fáciles de implementar sobre lo desarrollado. Estas funcionalidades pueden ser el seguimiento de los servidores remotos (carga, espacio, etc.), un explorador de archivos web que permita la navegación entre los archivos de los respaldos, o un visor de logs integrado en el sistema. Y como objetivo a más largo plazo, sería más interesante poder gestionar almacenamientos remotos en lugar de ceñirse únicamente al propio servidor. Esto haría la aplicación mucho más escalable.

5 Glosario

Array

Estructura de datos.

Backtrace

Una pila completa de llamadas a funciones.

CSS [8]

"Cascading Style Sheets, traducido como «hojas de estilos en cascada». Facilidad del HTML que permite crear plantillas de estilo gráfico y aplicarlas a las diferentes páginas de un proyecto, manteniendo elementos comunes en la maquetación como colores, tipografías, tamaños de letras, fondos…"

Framework [8]

"Conjunto de tecnologías que ayudan a otra para un mejor desarrollo."

Helper

Archivos con métodos de uso común, que se agrupan para su reutilización.

HTML

Lenguaje de Marcas de HiperTexto (*HyperText Markup Language*).

Interfaz [8]

"La cara visible de un proyecto, lo que nos muestra la pantalla. Además puede referirse a la forma de interrelación entre la máquina y el hombre, actualmente esa interrelación es gráfica, pero se puede tener una interface sonora, por poner un ejemplo."

Log

Listado secuencial en formato de texto de los eventos realizados en la ejecución de un evento.

Namespace

Espacio de nombres. Contenedor abstracto que garantiza que los contenedores sean únicos.

RSA

Sistema criptográfico de clave pública.

Rsync

Aplicación para la transmisión de datos incrementales.

Script

Programa simple o guión que contiene órdenes que interactúan con el sistema operativo.

Shell

Interprete de comandos del sistema operativo.

SQL Injection

Tipo de ataque informático que consiste en la inserción de un código SQL con la finalidad de comprometer la base de datos y el sistema.

SSH

Protocolo de acceso remoto a una Shell de forma segura.

URI

Identificador de Recursos Uniforme (*Uniform Resource Identifier*)

VPS

Servidor Privado Virtual (*Virtual Private Server*)

6 Bibliografía

[1] M. Achour et al, «PHP Manual,» 2020. [En línea]. Disponible en: https://www.php.net/manual/en/. [Último acceso: 9 Abril 2020].

[2] «phpseclib 1.0,» [En línea]. Disponible en: http://phpseclib.sourceforge.net/. [Último acceso: 2 Mayo 2020].

[3] «PHPMailer - A full-featured email creation and transfer class for PHP,» [En línea]. Disponible en: https://github.com/PHPMailer/PHPMailer. [Último acceso: 28 Junio 2020].

[4] Codeception, «Guides,» 2020. [En línea]. Disponible en: https://codeception.com/docs/. [Último acceso: 9 Abril 2020].

[5] A. Tridgell y P. Mackerras, «Rsync(1),» 2020. [En línea]. Disponible en: https://rsync.samba.org/ftp/rsync/nightly/rsync.html. [Último acceso: 23 Abril 2020].

[6] Mozilla Developer Network, «HTTP,» 2020. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP. [Último acceso: 28 Marzo 2020].

[7] Bootstrap, «Documentation,» 2020. [En línea]. Disponible en: https://getbootstrap.com/docs/4.4/getting-started/introduction/. [Último acceso: 5 Marzo 2020].

[8] F. BBVA, «Fundéu BBVA,» 27 Octubre 2011. [En línea]. Disponible en: https://www.fundeu.es/escribireninternet/entendiendo-a-los-disenadores-las-claves-de-su-vocabulario/. [Último acceso: 02 Junio 2020].

7 Anexo

El proyecto está liberado en la plataforma GitHub. En la misma página se encuentra un manual de instalación, que reproduzco a continuación.

Batea

Backup Administrator Tool End-of-degree Application

PHPDocs »

Trabajo Final de Grado:
Gestor de copias de seguridad de rsync para configurar en un VPS
David Campos · Universitat Oberta de Catalunya

Tabla de contenidos

Sobre el proyecto

El presente proyecto, fue realizado con fines académicos como parte del Trabajo Fin de Grado titulado "Gestor de copias de seguridad de Rsync para configurar en un VPS".

Realizado con

- PHP
- SQLite
- rsync
- arnapou/jqcron
- phpseclib/phpseclib
- phpmailer/phpmailer
- codeception/codeception
- phpDocumentor
- Bootstrap
- Boxicons

Empezando

Puedes conseguir una copia de este proyecto siguiendo los siguientes pasos.

Prerrequisitos

Este proyecto utiliza Composer para administrar dependencias.
Se pueden consultar los pasos a seguir para su instalación en el sitio de composer.

Instalación

1. Clonar el proyecto

```
git clone https://github.com/davidcmg/batea.git
```

2. Instalar las dependencias

```
cd batea/
composer install --no-dev
```

3. Renombrar el archivo de config/Config.sample.php a config/Config.php y establecer los parámetros correspondientes.

```
mv config/Config.sample.php config/Config.php
```

4. Configurar el servidor web para que apunte al directorio public/

5. Sería recomendable crear un par de claves para la aplicación.

```
cd keys/
ssh-keygen -t rsa
Enter file in which to save the key
(/home/youruser/.ssh/id_rsa): id_rsa
Enter passphrase (empty for no passphrase): [en blanco]
```

6. Configurar como propietario del proyecto al usuario del servidor web

```
cd ../../
chown www-data:www-data batea/ -R
```

7. Establecer permisos de lectura y escritura a `storage/log/` y a `storage/backups/` así como al archivo `storage/db.sqlite`

8. Añadir una entrada al cron del sistema, tiene que ejecuarse como usuario del servidor web, por ejemplo:

```
sudo crontab -u www-data -e
* * * * *  /usr/bin/php /full/path/to/project/batea.cron.php
>> /dev/null 2>&1
```

El usuario por defecto es `admin` con contraseña `secret`.

Licencia

Distribuido bajo la licencia MIT. Ver https://github.com/davidcmg/batea/blob/master/LICENSE.md para más información.