## HERRAMIENTAS DE DESARROLLO

Para la implementación del proyecto se ha hecho uso de las siguientes herramientas. Estas se han integrado mediante el entorno de programación detallado en el punto 2.

### LENGUAGES DE PROGRAMACIÓN

#### Java



Figura 1 - Java: <https://www.java.com/en/>

El principal lenguaje de programación ha sido Java, con el que se han implementado íntegramente todas las funcionalidades que se ejecutan en el servidor, como la lógica de negocio de la aplicación, las validaciones y acceso a datos, entre otros. Las motivaciones para elegir Java frente a otros lenguajes de programación se sintetizan de la siguiente manera:

* Familiaridad por parte del desarrollador: Se trata del lenguaje más utilizado en el Grado, por lo que el aprendizaje de nuevas herramientas es rápido e intuitivo. Además, no ha sido necesario invertir tiempo en el aprendizaje de un lenguaje nuevo.
* Programación orientada a objetos: Durante la preproducción del proyecto, y en la fase de análisis y diseño, se han identificado un número considerable de entidades, que se encuentran fuertemente relacionadas entre sí. Por ello, ha resultado más intuitivo emplear un lenguaje orientado a objetos, que transforma estas entidades en clases, además de las otras ventajas que acarrea la programación orientada a objetos, como modularidad y escalabilidad.
* Popularidad: Al tratarse de uno de los lenguajes más populares, no es difícil encontrar material de ayuda y aprendizaje.
* Formación: El índice TIOBE[[1]](#footnote-1), entre otros sistemas de medición de popularidad, declaran a Java como el lenguaje más popular de la industria, por lo que su dominio se considera una fortaleza fundamental para un egresa del Grado para obtener un prometedor futuro profesional.

#### JavaScript

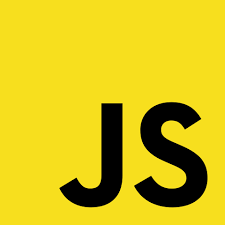


Figura 2 – JavaScript

El segundo lenguaje de programación más usado ha sido JavaScript. Se ha empleado principalmente para dotar de dinamismo a las páginas de la aplicación, así como para validar entradas.

Además, también se ha empleado para desarrollar la herramienta que calcula el balance de fuerzas en los enfrentamientos que ha sido la única parte de la lógica de negocio que se ha desarrollado en el cliente. El objetivo de esta decisión es que el Game Master disponga de un panel de mando donde pueda conocer el balance de fuerzas y los efectos de los cambios en este en tiempo real y dinámicamente.

Para completar el desarrollo en el cliente, se han empleado las siguientes librerías de JavaScript:

* jQuery[[2]](#footnote-2): Librería empleada extensivamente en toda la aplicación para facilitar y agilizar el scripting de las funcionalidades JavaScript. Entre otras utilidades, se ha empleado para cambiar dinámicamente el aspecto de la aplicación y realizar llamadas asíncronas mediante AJAX al servidor y manipular los datos recibidos en el cliente.
* jQueryUI[[3]](#footnote-3): Librería que contiene un conjunto de herramientas construidas sobre jQuery que mejoran la experiencia de usuario. Se ha empleado espontáneamente para incluir animaciones y arrastre de elementos.
* jQuery Validation[[4]](#footnote-4): Plugin del jQuery empleado en todos los formularios para validar sus entradas instantáneamente en navegador del cliente, lo que ahorra uso del servidor.

### LENGUAGES DE MARCADO

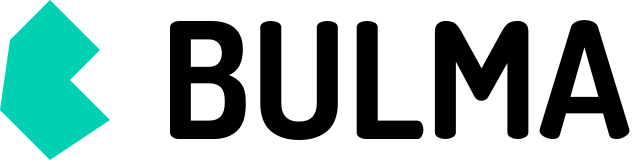


Figura 3 - Bulma: <https://bulma.io/>

Para el marcado y la estructuración de la aplicación, se han empleado HTML5 y CSS3, como es habitual en las aplicaciones que cuentan con páginas web. Para agilizar el proceso de aplicar estilo a la aplicación, se ha empleado el framework Bulma, un sustituto de Bootstrap que permite aplicar un estilo *responsive* y homogéneo a las páginas mediante un solo fichero CSS. Se ha optado por este framework frente a otros por la sencillez y modularidad que ofrece, además de contar con una documentación completa y ejemplos ya preparados. No obstante, ya que el estilo no era una prioridad en el desarrollo, se ha optado por emplear las configuraciones por defecto de Bulma.

### SPRING



Figura 4 - Spring Boot: <https://spring.io/projects/spring-boot>

El framwork Spring es la columna vertebral de toda la aplicación. Mediante este, se implementa la arquitectura MVC, el acceso a datos, y los *endpoints* con los que interacciona el usuario, así como la lógica de negocio que corre en el servidor. En concreto, la aplicación se ha construido con Spring Boot 2.0. Existen dos principales motivos por los que se ha decidido emplear Spring Boot frente a otros frameworks: su popularidad y su sencillez.

* Popularidad: Spring Boot es uno de los frameworks de Java más populares del mercado[[5]](#footnote-5). Esta popularidad trae consigo dos ventajas: por un lado, existe abundante material didáctico para su aprendizaje, y de ayuda para resolver dudas. Por otro, su dominio y familiarización se consideran una fortaleza a la hora de entrar en el mercado laboral de la industria. Así, se completa la formación proporcionada por el Grado, donde se trabaja con Struts2, otro framework similar.
* Sencillez: La principal propuesta de Spring Boot es el *just run[[6]](#footnote-6)*. Al crear una aplicación con Spring Boot, se compila como una aplicación Java tradicional, empaquetada en un JAR atuónomo, que se puede lanzar mediante el terminal. Esta trae consigo el servidor Tomcat embebido, por lo que, una vez lanzada, basta acceder a *localhost:8080* para empezar a usarla.

#### Propiedades de una aplicación Spring Boot

La sencillez de Spring se puede descomponer en varios aspectos ventajosos que trabajan en conjunción para facilitar el trabajo al desarrollador. Todos ellos se han aprovechado al máximo para invertir el mayor tiempo posible en el desarrollo de la aplicación en sí.

##### Aplicación independiente o *stand alone*

Al lanzar el proyecto con Maven, se empaqueta todo en un fichero JAR, en lugar de un WAR que luego haya que desplegar. Este fichero incluye todo lo necesario para ejecutar la aplicación, como si se tratase de una aplicación de escritorio. Entre otros elementos, se incluyen las páginas HTML, el código JavaScript y CSS y las imágenes estáticas, además de un servidor Tomcat embebido. La aplicación se lanza mediante el terminal con el comando *java -jar*, y se utiliza mediante el navegador. Esto permite que, sin necesidad de trabajo extra, se puedan exportar tres versiones distintas para acomodarse a las necesidades del cliente:

1. Versión en un hosting: La aplicación se lanza en un servidor remoto, donde también se ha instalado una instancia de MariaDB, y a cuya IP se le asigna un dominio, y se accede y utiliza como una aplicación web tradicional.
2. Versión local permanente: La aplicación se lanza en un equipo local, en conjunción con una base de datos MariaDB que se encarga de la permanencia. De este modo, los datos son permanentes en ese equipo.
3. Versión local portátil: La aplicación se lanza en un equipo local, empleando la base de datos *in memory* H2. No es necesario instalar nada, ya que H2 viene por defecto en Spring Boot, pero el contenido de esta base de datos se borrará cuando se cierre la aplicación.

Además, la versión remota es multiplataforma y las locales funcionan desde cualquier sistema operativo y navegador, como las aplicaciones Java tradicionales.

##### Código 100% Java

Spring Boot viene completamente autoconfigurado, por lo que el tiempo que es necesario invertir en configuración es mínimo. Además, esta configuración se realiza mediante anotaciones en las clases Java donde son relevantes, sin necesidad de ficheros XML. Por ejemplo, la anotación *@SpringBootApplication* marca la clase Main como una aplicación Spring Boot, y pone en marcha todos los mecanismos de configuración y arranque de manera transparente para el desarrollador.

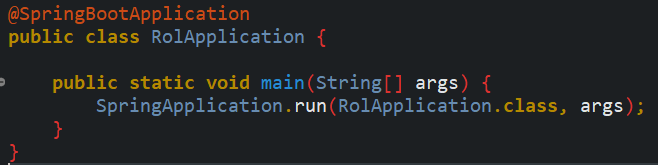


Figura 5 - Contenido del método main

##### Bajo acoplamiento o *loose coupling*

El framework Spring funciona empleado el patrón de inyección de dependencia[[7]](#footnote-7), también llamado inversión de control. Cuando una clase A depende de otra clase B, tradicionalmente es necesario instanciar B en A, explícitamente designando la implementación de B. En su lugar, el contenedor de A inyecta la dependencia B cuando esta sea necesaria, tomando la implementación de B que se haya marcado como tal con una anotación. En el ejemplo de la Figura 6, mediante la anotación *@Service* marcamos a la clase *ScenarioSerciveImpl* como implementación de *ScenarioService*. Así, esta será la clase inyectada cuando otra clase tenga una dependencia con *ScenarioService*. Para marcar una dependencia que debe ser inyectada, se emplea la anotación *@Autowired*.

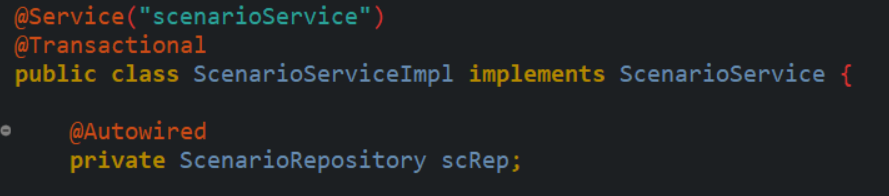


Figura 6 - Inyección de dependencia

Así, el acoplamiento se produce solo con la interfaz, y no con la clase que la implementa, y todo este proceso es transparente para el desarrollador.

#### Arquitectura de una aplicación Spring Boot

La arquitectura MVC de la aplicación se ha implementado usando las herramientas de Spring Boot. Como se ha visto en el documento de Diseño, la capa de Vista se ha implementado usando Thymeleaf (que no incluye código Java), la capa del Controlador incluye la capa de Controladores y la de Servicio, y el Modelo incluye los POJO y los Repositorios.

En general, para cada entidad, existe un servicio y un repositorio, y las entidades más relevantes disponen de un controlador. No se ha creado un controlador para cada entidad por dos motivos. En primer lugar, no todas las entidades tienen cuatro operaciones CRUD distinguidas, y en segundo, las operaciones que involucran a algunas entidades, como Turno e Intervención, están fuertemente relacionados con otras entidades, como Partida y Enfrentamiento, por lo tanto, los procedimientos de las primeras se han incluido en los controladores de las segundas.

##### Capa de controladores

Los controladores de Spring se marcan con la anotación @Controller y sus métodos proporcionan *endpoints* que se disparan cuando se accede a la URL que los mapea. Este mapeo se lleva a cabo con las anotaciones como *@GetMapping* y *@PostMapping*, que asignan estos métodos a dichas operaciones GET o POST, y es completamente transparente, como puede verse en la Figura 7:

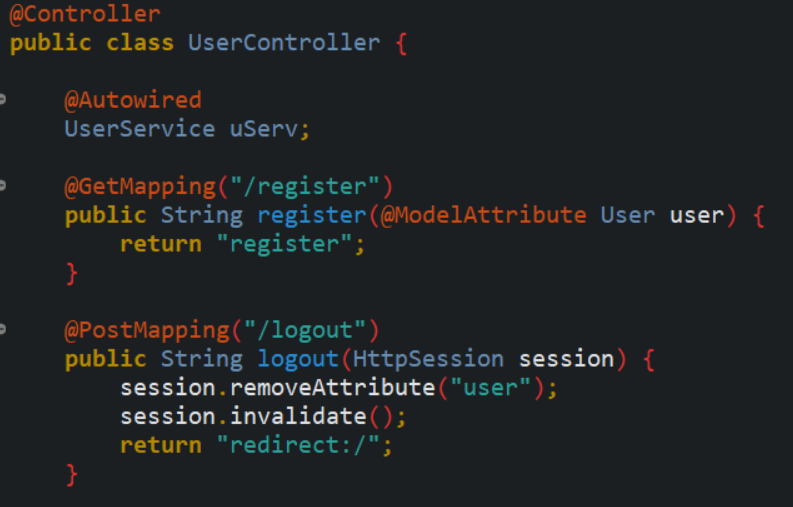


Figura 7 – Ejemplo de Controlador

Además, esta capa se encarga de validar los datos de los formularios cuando sea necesario, y de preparar el contenido que se mostrará por la capa de vista al usuario.

##### Capa de Servicio

Incorpora la lógica de negocio de la aplicación. Todos los servicios se componen de una interfaz con su implementación. Las clases se marcan con la anotación *@Service*:



Figura 8 – Ejemplo de Servicio

##### POJOs

Al marcar una clase como *@Entity*, la JPA de Spring (en este caso, Hibernate), las mapea como tablas en la base de datos. Es decir, que las tablas de la base de datos se crean y modifican automáticamente en función de la configuración especificada en las clases POJO: los atributos de la clase se traducen de tipos Java a tipos SQL y se convierten en columnas, las condiciones en restricciones, y las relaciones con otros POJO en claves foráneas. Esta configuración se lleva a cabo mediante anotaciones, de manera automatizada y transparente para el desarrollador.

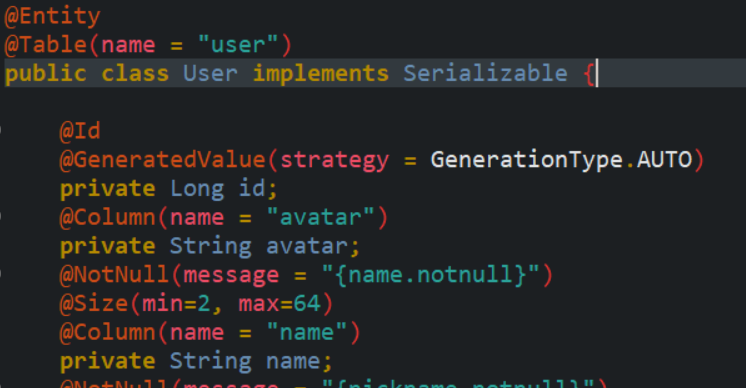


Figura 9 - Ejemplo de POJO

##### Capa de Repositorios

Los repositorios son interfaces que heredan de *CrudRepository* (proporcionada por Spring) y se encargan del acceso a datos. Estos se construyen de manera complemente desacoplada de la implementación de JPA y el tipo de base de datos que se esté empleando. Además, no es necesario implementar los métodos de consultas. En su lugar, basta con denominar al método como *findByAttribute* para que Spring lo interprete para buscar la entidad por el atributo especificado. En caso de consultas más complejas, estas se pueden proporcionar mediante una anotación. En el ejemplo de la Figura 10, el primer método devuelve la lista de Partidas cuyo atributo *master* coincida con el usuario proporcionado; y en el segundo, devuelve la lista de Partidas que cumplan la consulta.

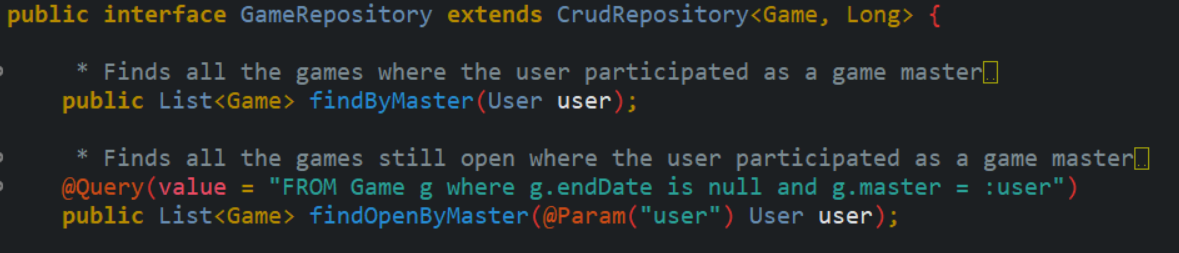


Figura 10 - Ejemplo de Repositorio

De este modo, se aplica el patrón DAO sin necesidad de implementar el acceso a datos en sí, en su lugar, Spring lo implementa de manera transparente según la configuración especificada en el fichero *properties*. Por defecto, como implementación de JPA se usa Hibernate, y esta es la que se ha empleado en el proyecto.

### HIBERNATE



Figura 11 - Hibernate: <http://hibernate.org/>

Hibernate se ha usado como herramienta para el mapeo objeto-relación entre la aplicación Java y la base de datos SQL. Además, también se ha usado como implementación de JPA para gestionar la persistencia desde Spring. Los motivos para usar Hibernate frente a otras herramientas son similares a los motivos para usar Spring Boot y Java: popularidad, sencillez y familiaridad, con las ventajas que estos motivos suponen mencionadas anteriormente. Finalmente, el uso de esta utilidad también aporta otros beneficios:

Por otro lado, ya que Hibernate es el proveedor de JPA por defecto en Spring Boot, puede ponerse en marcha con una mínima configuración mediante un fichero *properties*, sin necesidad de gestionar ficheros XML y generación de código mediante el IDE, como puede verse en la Figura 12.

Por otro, editando este fichero *properties*, se puede cambiar la fuente de datos con la que Spring Boot va a trabajar, permitiendo cambiar entre H2 y MariaDB (u otra fuente de datos que se desee en el futuro) sin esfuerzo extra para el desarrollador.

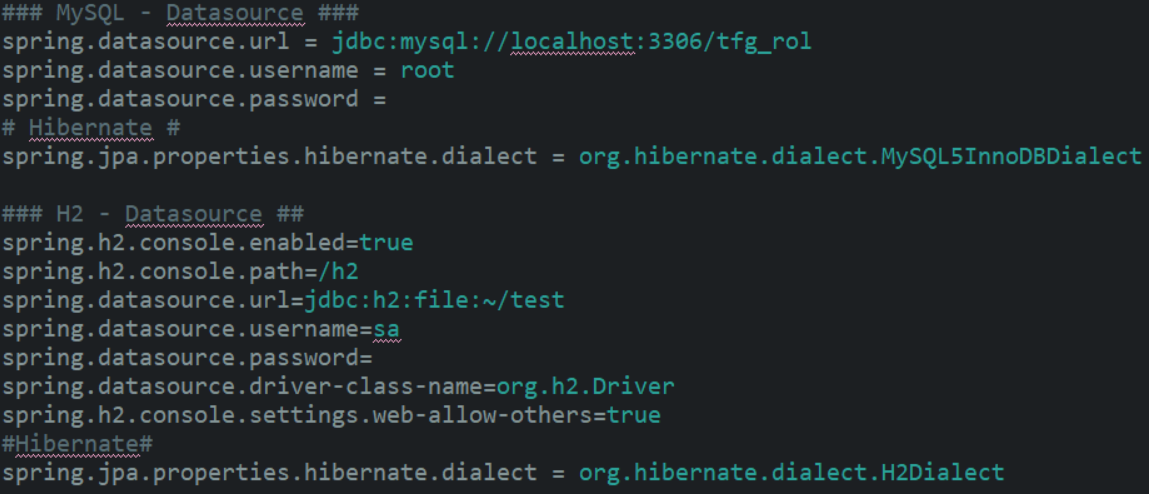


Figura 12 - Ejemplo de configuración de Hibernate

### MARIADB



Figura 13 - MariaDB: <https://mariadb.com/>

Como principal gestor de la persistencia se ha empleado MariaDB, una solución de código abierto completamente compatible con MySQL. Se ha optado por una base de datos SQL en lugar otro sistema, como MongoDB[[8]](#footnote-8) por los siguientes beneficios:

* Entidades fuertemente relacionadas: El proceso de negocio necesita varias entidades fuertemente relacionadas, es decir, que, para caso de uso, es necesario crear, consultar o modificar varias entidades relacionadas a la vez. Esto convierte en ideal a una base de datos relacional frente a una no relacional, cuya ventaja es la inserción y lectura de entidades independientes.
* Integridad de la información: Las bases de datos SQL garantizan atomicidad en las transacciones, algo fundamental cuando las operaciones incluyen varias entidades relacionadas, para evitar dejar entidades huérfanas. Además, el tipado fuerte de datos de Java no necesita la flexibilidad que ofrece un sistema no SQL.
* Poco volumen de datos: El proceso de negocio no genera un gran volumen de datos, y tampoco se espera un gran volumen de usuarios, por lo que la escalabilidad ofrecida por una base de datos no SQL no es necesaria.

Una vez tomada la decisión de emplear una base de datos SQL, se ha optado por MariaDB por su popularidad y familiaridad, y las ventajas laborales y productivas que esto conlleva. Para la versión portátil, se ha empleado la base de datos *in memory* H2[[9]](#footnote-9), principalmente debido a su sincronía con Spring Boot, y a otras ventajas como su baja huella en el peso de la aplicación y aplicación de gestión en el navegador.

### THYMELEAF



Figura 14 - Thymeleaf: <https://www.thymeleaf.org/>

Thymeleaf es un motor de templates que permite ofrecer contenido dinámico HTML5 desde el servidor. Se ha empleado para implementar la capa de la vista de la aplicación. Thyemleaf emplea un lenguaje llamado *Standard Dialect* para procesar la lógica de la vista (por ejemplo, listas, condicionales, entre otros), el cual puede acceder a ciertos atributos que se proporcionan desde el Controlador, así como a los almacenados en la sesión.

### MAVEN

## ENTORNO DE PROGRAMACIÓN

Para desarrollar el Proyecto mediante las utilidades y las tecnologías mencionadas anteriormente, se han empleado las siguientes herramientas:

### ECLIPSE



Figura 15 - Eclipse: <https://www.eclipse.org/>

La principal herramienta de desarrollo para el proyecto ha sido el IDE Eclipse. En concreto, se ha empleado Eclipse Java EE para Desarrollo Web, versión 4.7 Oxygen. Se ha optado por esta versión en concreto ya que es la más completa, ofreciendo, por un lado, integración con otros útiles como Maven y Git, y por otro, multitud de herramientas para el desarrollo de aplicaciones web, como el servidor Apache Tomcat, o editores de HTML y JavaScript, entre otras.

Las justificaciones a la hora de elegir Eclipse frente a otros IDE son las siguientes:

* Al tratarse de uno de los IDE más populares, existe una amplia gama de material consultivo, tutoriales y ayudas para poner en marcha cualquier tupo de proyecto, y resolver dudas durante su desarrollo.
* Buena coordinación con el Spring Framework.
* Software libre y de código abierto, de descarga y uso gratuito.
* Se trata de una de las herramientas más usadas en la industria, por lo que su conocimiento y familiarización son fortalezas clave para el futuro. Esta formación se complementa con el conocimiento adquirido durante el Grado, donde NetBeans es el IDE más utilizado.

### XAMPP

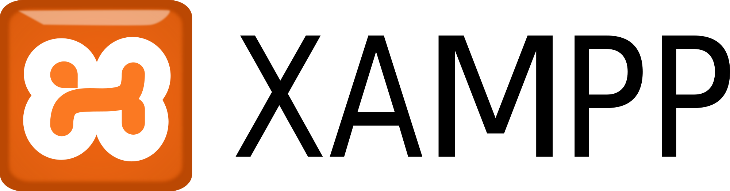


Figura 16 - XAMPP: <https://www.apachefriends.org/index.html>

La principal herramienta para la persistencia y el acceso a datos ha sido XAMPP, que permite desplegar rápidamente una base de datos de tipo MySQL para el entorno de desarrollo desde un sistema Windows, y manipularla mediante la GUI proporcionada por PhpMyAdmin. Además, esta manipulación también se puede realizar con la misma facilidad sobre una base de datos remota en el entorno de producción.

### OVH VIRTUAL PRIVATE SERVER



Figura 17 - OVH: <https://www.ovh.com/world/vps/>

Contratar un hosting para una aplicación web Java no se ha considerado económicamente viable para el alcance de este proyecto. En su lugar, se ha contratado un servidor virtual privado (referido como VPS, del ingés, *Virtual Private Server*). Por una décima parte de lo que cuesta un hosting para una aplicación Java, se obtiene una máquina virtual remota Ubuntu 16.04, con una dirección IPv4 accesible desde el exterior, donde se ha procedido a instalar MariaDB y a lanzar el fichero JAR generado por Spring Boot (que ya viene con Tomcat embebido). Además, se nos proporciona un panel de control donde se puede monitorizar el tráfico, uso de disco y memoria, entre otros. Todo ello alojado en el centro de datos de la compañía en Estrasburgo, Francia.

### FREENOM



Figura 18 - Freenom: <https://www.freenom.com/en/index.html>

Freenom ofrece dominios gratuitos durante doce meses. Se ha registrado un dominio *.tk* (perteneciente al territorio de Tokelau, administrado por Nueva Zelanda), y se ha redirigido a la IP proporcionada para la VPS de OVH mediante la herramienta de gestión de DNS proporcionada por Freenom.

### HERRAMIENTAS DE SSH

Ya que Windows no cuenta con una herramienta de SSH nativa, se ha optado por emplear dos herramientas de software libre y código abierto para gestionar la VPS remotamente:

#### PuTTY



Figura 19 PuTTY: <https://www.putty.org/>

Permite conectarse a través de SSH al usuario de Ubuntu desde Windows y acceder a la terminal para ejecutar los comandos. Además, permite copiar y pegar entre el host y la terminal.

#### WinSCP



Figura 20 – WinSCP: <https://winscp.net/eng/index.php>

Permite conectarse desde un host Windows de manera segura a la VPS y gestionar, mediante una GUI, el sistema de archivos, como si se tratase del mismo explorador de archivos de Windows. Además, permite copiar ficheros desde el host al la VPS remota y viceversa.

### GITHUB

1. <https://www.tiobe.com/tiobe-index/> [↑](#footnote-ref-1)
2. <https://jquery.com/> [↑](#footnote-ref-2)
3. <https://jqueryui.com/> [↑](#footnote-ref-3)
4. <https://jqueryvalidation.org/> [↑](#footnote-ref-4)
5. <https://redmonk.com/fryan/2017/06/22/language-framework-popularity-a-look-at-java-june-2017/> [↑](#footnote-ref-5)
6. <https://spring.io/projects/spring-boot> [↑](#footnote-ref-6)
7. <https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-spring-beans-and-dependency-injection.html> [↑](#footnote-ref-7)
8. <https://www.mongodb.com/> [↑](#footnote-ref-8)
9. <http://www.h2database.com/html/main.html> [↑](#footnote-ref-9)