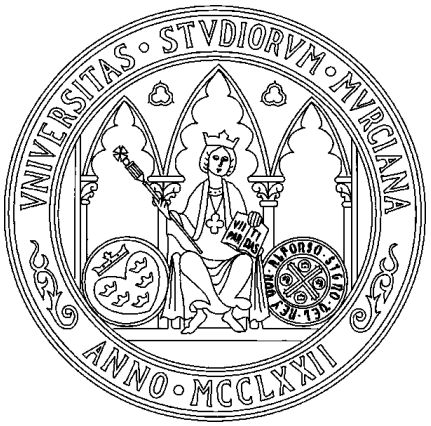
Trabajo Fin de Grado

Generación de datos para testing en Bases de Datos NoSQL

Alumno
Óscar Hernández Navarro

Tutores
Jesús J. García Molina
Diego Sevilla Ruiz



Enero de 2022

Agradecimientos

Agradecimientos...

Índice general

Extended Abstract

Nowadays, Git is by far the most widely used modern version control system in the world. Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Git is a mature and actively maintained open source project, with tons of developers behind it. Originated and launched in 2005, by the creator of the Kernel of Linux systems, Linus Torvalds, has become a crucial tool for the development teams, bringing them better performance, flexibility, and security for their projects.

Some of the reasons that make this tool so widely used may be, (i) Teamwork, allowing parallel development on a project with a shared access. (ii) Autonomy, each developer, has a full local copy of the project and the changes generated on it, which let developers work individually. (iii) Speed, Git need less processing requirements and management than other version control systems. (iv) Tree structure, teams can have different branches, on which they can make changes without modifying the principal base code, making the testing easier and making possible the creation of alternative solutions. (v) Scalability and fault tolerance, open source, multi platform implementations and community support.

The rise of this technology has been so great that multiple faculties related to software education integrate it into their teaching, using this tool and integrating it in the development of the projects to be accomplished by their students. When this happens, the students create repositories making the teacher a collaborator of them. In that way, the teacher has full access to the repository where the students work progressively, allowing cooperative work in real time.

On the other hand, the teaching staff can later analyze the progression and work of the students, checking the changes made in the repository, in such a way that they can observe what changes each student has made, how the work has been distributed, detect overload of work of some student with respect the rest, and much more. In this way, the teacher can offer a rating as fair as possible.

However, for teachers, at time of evaluation, it can be hard to check the data offered by their student's repositories, especially when the number of repositories increases. Imagine

a professor who teaches different subjects, and for each of them the students use different repositories.

Currently, the are some solutions for the problem, such as the stats panels of Git service providers, such as GitHub or GitLab. However in case of GitHub, these stats can only be consulted on public repositories, to be able of check them at private repositories, a premium account is needed under a monthly subscription. On the other hand, GitLab stats panel is offered for private repositories for free, but access to the changes made in the repository cannot be easily visualized together with the stats, having to open another page for viewing them, becoming somewhat tedious in the case of have many repositories.

These services providers, GitHub and GitLab, also offer payment plans dedicated to education, these plans include multiple advantages, being more interesting in the case of GitHub, offering a tool designed exclusively for education "GitHub ClassRoom", this tool allows the creation of virtual classrooms, where teachers can create templates of projects and share for the students, create homeworks for the students which can be self-assessed, assign work teams, in addition of have full control of the changed made by the students, with multiple statistics and facilities for students follow-up, allowing a fast and direct interaction between teachers and students.

As we can see, the solutions offered are not entirely effective, the one that probably best suits the problem posed is the GitHub Education plan, nevertheless, although many subsidies and grants can be requested, the service entails a cost for the institution and also the necessary management to form the agreement between the institution and GitHub must be taken in account. In case of not being able to apply this solution, it would have to be the teaching staff, who would have to either get a premium account or contract the GitHub services on their own. Or manually inspect all the repositories.

For the reasons mentioned previously, we want to develop a software for the computer science faculty of the University of Murcia, that allows teachers to manage the information of their students' repositories, obtaining statistics about them, simplifying their task. This would be a first implementation, with lots of room for improvement, with the possibility of implementing new features.

In addition, it is intended to offer a useful implementation for deployment in a cloud service, in such a way that the services are based on containers, which can be launched in a structure of cloud nodes, thus allowing their control and scaling.

In order to continue, we must first understand the Git jargon, and some relevant definitions related to the developed project. As a glossary, we see the following definitions:

- Repository: A Git repository tracks and saves the history of all changes made to the files in a Git project. It saves this data in a directory called "`.git`", also known as the repository folder. Git uses a version control system to track all changes made to the project and save them in the repository.

- Commit: In version control systems, a commit is an operation which sends the latest changes of the source code to the repository, making these changes part of the head revision of the repository. Commits in version control systems are kept in the repository indefinitely. Thus, when other users do an update or a checkout from the repository, they will receive the latest committed version.
- Branch: is the duplication of an object under version control (such as a source code file or a directory tree). Each object can thereafter be modified separately and in parallel so that the objects become different. In this context the objects are called branches. The users of the version control system can branch any branch. Branches allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository.
- REST API: A RESTful API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data. That data can be used to GET, PUT, POST and DELETE data types, which refers to the reading, updating, creating, and deleting of operations concerning resources.

Once these concepts are understood, we can begin by explaining the software requirements. The proposed solution is based on the development of a web application that allows the management of teacher's repositories. To do this, each teacher will make a registration in the application, in which they can later add their subjects, being able to filter their repositories by matching strings and manually add repositories to their subjects. For the registration process, each teacher must have or create an authentication token obtained on the official GitHub page,necessary for the developed application to access the private repositories of teachers. The tool will never make any changes to the repositories or add any information to git, it will only need read permissions.

Once the registration is complete, you can access the application and view all the repositories in which you participate as a collaborator on the main page. In addition, a search bar is offered, for filtering the repositories that match a string of characters and filtering by previously created subjects. You can click on any of those displayed to access the repository statistics. Obtaining information about the collaborators of the repository, the total number of commits made by each collaborator, the temporal distribution of the commits, that is, how many commits have been made in each month of the course, and the content of the changes made in the project on every commit. With this information, the teacher can evaluate the performance of the students during the development of the assigned work.

Turning now to the implementation of the software, there were different alternatives, as for the development of the web server, the possibility of using Python with the Flask framework, or using Node JS based on the Express framework. Finally, the decision was to use Node JS given the good integration of the JSON format used by the Github API through which we retrieve the data from the repositories, in addition to the many possibilities that the Express Framework offer for the development of simple API REST.

Regarding the persistence of the data, necessary in this case for the data of both the users and the subjects created, a MySQL implementation is used, since the data to be persisted perfectly adapts to the relational model, and NodeJS offers very good libraries for connection and integration with the database from the server.

Finally, for design reasons, a second server has been implemented, also developed in NodeJS Express, which is in charge of carrying out all the communication with the Github servers, that is, it will be in charge of obtaining all the necessary information from the Github services. In this way, we have:

The first server, which implements the entire web service, is intended only to receive and respond to web requests from users. This server will also be in charge of communicating with the database and managing it, carrying out registration queries and user creation, as well as managing the subjects. If data from the Github Rest API is necessary, this data is requested from the second server.

The second server implements a service as a REST API in charge of attending to the requests of the first server and intercommunicating with the GithuB API to obtain the requested data. Therefore, it acts as a proxy for the GitHub service.

The reasons for implementing a second server are justified by assigning only the web server to users and not overloading it with requests to another external service, thus reducing its workload and improving response times.

On the other hand, for the communication of the two servers, the second server is implemented as a REST API, where it receives HTTP requests, in this case it only accepts Get requests, the information about what data you want to recover, is received through the url parameters. Once the request is received and processed, the second server communicates with the GIthub API, obtains the corresponding data and responds back to the first server.

As a second objective for the project, we want to give support to the services developed in order to facilitate deployment in a cloud service.

To do this, all services must offer a container based implementation, in this case using Docker's container technology, currently, one of the most relevant technologies for this task.

The structure of the application is divided into 3 containers, a dedicated container for the database, which must configure a volume to give persistence to the data in the container, and two different containers for each of the servers implemented in NodeJS.

To make the deployment of these containers easier,a Docker Compose configuration file is provided, which allows us to automatically launch the 3 services, with the specific configurations for each one. It is also prepared to configure an internal subnet used to resolve the communications between the services, that is, the communications between the main server and the MySQL database and the communications between the first and second servers for obtaining Git data.

Regarding the work methodology followed, multiple meetings have been held with the tutor Diego Sevilla, in which the idea of the project has been raised and evolved.

In the first meeting, the development of a tool that tries to solve the problem of analyzing student repositories is planned. Different forms of implementation and technologies to be used are also proposed.

Subsequently, after a period of research on how to retrieve statistics from the

Git servers, in this case Github, a second meeting took place, where it is concisely specified what the architecture of the application will be and the technologies that will be used.

After a period of work, other meetings have been held where the progress made has been reviewed, new application requirements have been proposed and some aspects have been improved.

Successively, there have been multiple consultations throughout the development of the project, both by email and by shorter face-to-face meetings.

Finally, it should be noted that, as it could not be missing, a repository has been used

Private GitHub to share work with tutor and version control.

To conclude, comment that the development of the project has entailed a lot of work, but it has been a very rewarding experience, and key to consolidating technologies that are currently on rise, such as the development of microservices and the collaboration between them to provide solutions for multiple problems, and its subsequent deployment on cloud architectures.

Resumen

1 Introducción

Un sistema de control de versiones(VCS, por sus siglas en inglés), es una herramienta o software, que monitoriza y gestiona cambios en un sistema de archivos, es decir, se encarga de la monitorización de la adición, eliminación y modificación aplicadas a archvivos y directorios. El primer sistema de control de versiones, surge en 1990, con el nombre de CVS(Concurrent Version System), capaz de gestionar multiples versiones desarrolladas de forma concurrente en diferentes máquinas y almacenadas en un servidor central. En la actualidad, el sistema de control de versiones mas usado es Git, desarrollado por Linux Torvalds(creador de linux) y lanzado en el 2005, su creador, buscaba mejorar la eficiencia, confiabilidad y compatibilidad respecto las alternativas existentes, que el mismo criticaba. Este controlador de versiones esta basado en un grafo acíclico dirigido, donde cada nodo representa una entidad susceptible a modificaciones.

Para Git, un *repositorio* constituye el proyecto que esta bajo el control de versiones, los *commits* constituyen los cambios que van surgiendo sobre el repositorio, y además ofrece la posibilidad de la creación de *ramas*, siendo estas diferentes caminos que puede ir tomando el proyecto, pudiendo combinarlas o descartarlas.

1.1 Motivación

Para comprender el impacto que tuvo Git a partir de su creación y los mótivos por los cuales a dia de hoy es la primera opción de casi el 90 % de los desarrolladores, destacaremos algunos de sus principales puntos fuertes, (i) Trabajo en equipo, permite el desarrollo en paralelo sobre un proyecto con un acceso compartido (ii)Autonomía, cada trabajador cuenta con una copia en local de todo el proyecto y los cambios generados, permitiendo el trabajo de forma individual, el cualquier momento. (iii) Velocidad, necesita menos capacidad de procesamiento y gestión al realizar las operaciones en local (iv) Estructura en árbol, el equipo de trabajo puede contar con diferentes ramas, sobre las que se realizan cambios sin modificar el código base principal, facilitando así las pruebas y caminos alternativos en el desarrollo. (v) Escalabilidad, código libre, multiplataforma y apoyo de la comunidad.

Es por ello, que Git es una de las herramientas que los alumnos de la facultad de informática de la Universidad de Murcia deben adquirir, conocer y trabajar. Usandose en multiples asignatuaras, para el control de versiones del trabajo de los alumnos, permitiendo la colaboración en los equipos de trabajo. En estos proyectos, el profesorado indica que se debe crear un repositorio, donde tanto los alumnos del equipo de trabajo como el profesor deben ser colaboradores, de esta forma, el profesor tiene acceso pleno al repositorio pudiendo, así valorar el trabajo de los alumnos y analizando como ha sido su progresión durante el desarrollo del proyecto, tanto tiempo dedicado, número de commits por alumno, tiempo que se ha dedicado a la realización del proyecto etc.

Como se puede apreciar, esta herramienta también es útil de cara a la evaluación por parte del profesorado, sin embargo, surge un problema de gestión de cara al tutor a la hora de gestionar los múltiples repositorios de tdos sus alumnos. Esto es, imaginemos un profesor el cual imparte diferentes asignaturas y sobre las cuales quiere usar repositorios Git para el control de versiones de los proyectos de sus alumnos, suponiendo que cada equipo de alumnos genere su propio repositorio, el tutor se encuentra con una cantidad abrumadora de repositorios sobre la que es colaborador. De cara a cuantificar el trabajo realizado, debe consultar los commits realizados por cada uno, el contenido de los mismos, las fechas sobre las que se han realizado etc, sin embargo, no tiene una forma sencilla con la que poder ver toda esta información simultáneamente y rápidamente.

Se pretende por tanto, proveer al profesorado de una herramienta con la que intentar paliar el problema planteado, es decir, el acceso a estadísticas de los repositorios Git de los alumnos, permitiendo además, gestionar los repositorios sobre los que es colaborador, añadir nuevos repositorios y filtrado de los repositorios. En cuanto a las estadísticas, es de interés el número de commits realizados por cada alumno, la distribución temporal de dichos commits, y además del contenido de los commits (cambios realizados en cada archivo del proyecto, también conocido como *Patch*).

1.2 Objetivos

El desarrollo del trabajo esta enfocado principalmente:

- El desarrollo de un software capaz de gestionar estadísticas sobre repositorios Git en los que se participa como colaborador, a. Para su implementación se lleva a cabo el desarrollo de una aplicación web, accediendo el usuario mediante el navegador de preferencia.
- Implementación de un esquema de desarrollo basado en una arquitectura de microservicios, es decir, implementación de servicios de no gran tamaño, intercomunicados por API bien definidas, ofrecidos en forma de contendores Docker, ofreciendo una capa de automatización para el despliegue en multiples sistemas operativos, que posteriormente puedan ser llevados a un sistema cloud permitiendo su escalado.

1.3 Metodología

Para la realización del trabajo, se han tenido múltiples reuniones con el tutor Diego Sevilla, en las cuales se han ido planteando y evolucionando la idea del proyecto.

En primer lugar, el 11 de febrero de 2022 se tuvo una reunión donde el tutor me plantea el problema del análisis de los repositorios de los alumnos, concluyendo para ello el desarrollo de una herramienta que de una solución al problema y tratamos documentación relevante de cara al posterior desarrollo del trabajo.

Posteriormente, tras un periodo de investigación sobre como recuperar estadísticas de los servidores de Git, en este caso Github, se tiene una segunda reunión el día 16 de febrero, donde se plantea, tanto la arquitectura completa y como tecnologías con las que se podría implementar, destacando principalmente un esquema de desarrollo web basado en Phyton con Flask o el uso del framework Express sobre NodeJS con Javascript como lenguaje de principal, finalmente veremos que la alternativa elegida es la segunda, principalmente por la facilidad de integración de javascript con el formato de información JSON, y la fuerte demanda de dicho formato en la aplicación. Además en esta reunión se hace la primera recolecta de requisitos del software a producir. Los detalles de la arquitectura se comentarán en los capítulos siguientes.

A mediados del mes de marzo, con avances realizados y una previa maquetación a modo de prototipo ya desarrollada se vuelve a tener una reunión, en la que se analizan los avances realizados, se plantea la adición de las estadísticas del contenido de los commits, y principalmente se trata el desarrollo de la aplicación sobre contenedores Docker.

Además de estas reuniones claramente diferenciadas y de larga duración, se han tenido multiples consultas y breves tutorías a lo largo del desarrollo del proyecto, tanto mediante correo electrónico como reuniones presenciales de menor duración.

Por último, cabe destacar que, como no podía faltar, se ha utilizado un repositorio GitHub privado para compartir el trabajo con el tutor y el control de versiones.

1.4 Estructura del documento

Este documento se organiza en los siguientes capítulos. El presente capítulo 1 ha presentado el contexto, motivación, objetivos y la metodología seguida. El capítulo 2 presenta fundamentos de las bases de datos NoSQL y una Introducción al sistema de gestión de bases de datos NoSQL MongoDB, el capítulo 3 hace un recorrido por las soluciones vigentes que podemos encontrar a día de hoy para motivar la implementación de este trabajo, el capítulo 4 tratará los objetivos del trabajo, tecnologías a utilizar y el marco en el que se encuentra, el capítulo 5 trata de la implementación del generador, su uso, funcionalidad y cómo ha sido construido y por último el capítulo 8 recoge las conclusiones obtenibles del trabajo realizado, así como una serie de vías futuras con el propósito de ampliar y extender el trabajo realizado.

2 Fundamentos

2.1 Bases de Datos NoSQL

Para entender las bases de datos NoSQL primero deberíamos conocer en qué contexto surgieron y la necesidad de estas. A principios de los años 2000 surgió el concepto de 'Web 2.0' o 'Web social' en la cual el usuario adquiría mayor relevancia en la web, en este momento la World Wide Web paso de ser un contenedor de información a una plataforma en constante evolución gracias a los usuarios que entraban en ella y la hacían evolucionar. Esto hizo que el volumen de tráfico de datos incrementara exponencialmente debido a que los usuarios cada vez eran más y cada vez introducían mayor cantidad de datos hacia redes sociales, plataformas de autopublicaciones como blogs o wikis.

Debido a esto los servidores empezaron a quedarse faltos de recursos. Hasta entonces la mayoría de páginas permanecían en potentes servidores únicos que gestionaban todas las solicitudes y al necesitar más recursos se mejoraban añadiendo mejores tarjetas de red, procesadores, discos de almacenamiento..., lo que llamamos escalabilidad vertical. Debido a que estos servidores suponían un coste muy alto seguir mejorándolos y el aumento de rendimiento no era suficiente, las empresas se vieron en la necesidad de añadir escalabilidad horizontal, esto es, añadir nuevos nodos (más servidores) que se repartieran el tráfico y el procesamiento de estos.

Con esto se solucionaron varios problemas como es el procesamiento de los datos, el tiempo de respuesta, balanceo de cargas, etc. Sin embargo, las bases de datos relacionales, que eran las bases de datos más utilizadas entonces no disponían de escalabilidad horizontal. Esto es debido a que estas bases utilizan el modelo ACID (Atomic Consistent Isolated Durable), este modelo aporta una gran fiabilidad a la hora de manejar datos en una SGDB controlando los datos y revirtiendo los cambios debido a fallos, sin embargo este modelo no casa con la escalabilidad horizontal y por ello nació el modelo BASE (Basically Available Soft state Eventually consistent) el cual prioriza la disponibilidad de los datos a su consistencia delegando en los desarrolladores la responsabilidad de la consistencia de los datos, aunque no todos los sistemas de bases de datos utilizan este modelo base, otras se apoyan en la disponibilidad y la tolerancia a fallos como podría ser DynamoDB, CouchDB, Cassandra, etc.

NoSQL significa no sólo SQL (Not Only SQL) pero como podemos ver la definición de una base de datos va mas allá de SQL el cual es simplemente un lenguaje de consulta. Para definir una base de datos como NoSQL además de no basarse en SQL, deben estar orientadas a la escalabilidad, debe ser flexible con el modelo de datos (Schemaless), suelen ser libres u Open Source y seguir algunos principios como el procesamiento distribuido o MapReduce.

Actualmente las bases de datos NoSQL se categorizan entre cuatro tipos: key-value (Clave-valor), documentales, columnares, de grafos. Estos son los modelos que siguen a la hora de almacenar sus datos y lo que las diferencia principalmente de las bases de datos relacionales.

- *Key-value*: Las bases de datos *key-value* proveen una forma sencilla de almacenamiento de datos pares *clave-valor*. Esta estructura es similar a un mapa de cualquier lenguaje de propósito general, una colección que relaciona una clave única con un valor que puede tomar cualquier tipo, y la base de datos no asume ninguna estructura en los valores de esas claves.
- *Documentales*: Las bases de datos basadas en documentos almacenan su información en forma de objetos o documentos que son similares a objetos JSON[1]: un conjunto de pares clave-valor y los valores pueden ser de un tipo primitivo, un array de valores u otro objeto embebido con la misma estructura.
- *Columnares*: Las bases de datos basadas en familias de columnas están organizadas como una colección de filas, cada una de ellas con una clave y una serie de familias de columnas que conforman a su vez un conjunto de pares *clave-valor*.
- *De grafos*: Las bases de datos basadas en grafos se organizan en entidades, que se almacenan como nodos con propiedades, y relaciones entre entidades, que lo hacen como aristas con propiedades y dirección. Estas relaciones con dirección son las que permiten interpretar los datos almacenados.

2.2 Introducción a MongoDB

MongoDB es un sistema de base de datos NoSQL documental y open source. Según DB-Engines es el quinto SGDB más popular hasta enero de 2022 y el primero entre los sistemas NoSQL.

MongoDB debe su éxito al desarrollo en una época donde las bases de datos NoSQL comenzaban a expandirse, con una sintaxis sencilla para la gestión de los datos y un gran soporte de drivers para lenguajes de programación de uso general. Además de la replica de servidores que provee una escalabilidad horizontal y que ha llevado a muchas empresas a migrar a MongoDB sus sistemas relacionales para algunos tipos de aplicaciones que requerían mayor eficiencia y escalabilidad.

Un gran ayuda para su expansión es el hecho de que guarda sus datos en estructuras *BSON*, la cual es una especificación similar a *JSON*[1] y esto provoca que sea muy amigable para desarrolladores que ya están familiarizados con esta. MongoDB provee además distintos servicios de almacenamiento en la nube, software para empresa, gratuitos, etc. En este proyecto se ha utilizado el driver oficial de MongoDB para Java, con él utilizaremos las funciones de leer y escribir en una base de datos local o remota.

```
[{
  "name": "Peter",
  "surname": "McGregor",
  "address": "Fifth street avenue",
  "bornYear": 1992,
  "Friends": [ "Emma", "George" ]
},
{
  "name": "Peter",
  "surname": "Cresswell",
  "address": "London",
  "bornYear": 1970,
  "Friends": [ "Harry", "Willy", "Paul" ]
}]
```

Figura 2.1: Ejemplo de objeto JSON[1].

Uno de los drivers que proporciona MongoDB de manera oficial es su software de conexión para el lenguaje de programación Java, capaz de configurar y mantener cualquier base de datos local o en red. Para este trabajo se utilizará esta API para realizar conexiones, realizar peticiones de consulta e insertar datos en colecciones. El driver de MongoDB está disponible en el repositorio Maven de Apache o Gradle.

```
String stringConnection = "mongodb://root:1234@localhost:27017";
MongoClientURI uri = new MongoClientURI(stringConnection);
mongoClient = new MongoClient(uri);
MongoCollection<Document> collection = mongoClient.getDatabase("People")
    .getCollection(uri.getCollection());
Bson queryJSON = BasicDBObject.parse("{ name: Peter }");
MongoCursor<Document> mongoCursor = collection.find(queryJSON).iterator();
```

El ejemplo anterior realiza una conexión a la base de datos ubicada en el puerto 27017 de localhost con usuario root y contraseña 1234, recupera la base de datos 'People' y realiza una petición que devuelve las entidades con el atributo name igual a 'Peter'. Esta petición devuelve un objeto BSON que puede devolver un iterador de los resultados.

3 Estado del arte.

3.1 Análisis de estadísticas sobre repositorios Git.

Actualmente, podemos encontrar multiples vías y herramientas con las que poder visualizar los datos acerca de los repositorios Git, en este caso, vamos a analizar tanto las herramientas ofrecidas para el control de estadísticas, por los proveedores de servicio, teniendo en cuenta dos de los proveedores mas reconocidos, como son, **GitHub**: y **GitLab** y subscripciones a planes de pago, los cuales se adaptan a un entorno educativo, donde profesores y alumnos cuentan con herramientas para el control y planificación del curso. En la actualidad podemos encontrar generadores de datos por la web que trabajan a través de una API para generar objetos con atributos predefinidos y grandes conjuntos de datos. Estos objetos además pueden ser exportados a una gran variedad de formatos de objetos como XML, JSON, CSV, TXT o directamente se pueden almacenar en algunas bases de datos muy populares como MySQL u Oracle DB.

Vemos a continuación, los aspectos más relevantes de cada una de ellas:

- **Estadisticas de GitHub (Insights)**, desde la propia página web de GitHub, o alternativas como GitHub Desktop o terminal, accediendo a un repositorio propio o del cual se es colaborador, en el apartado Insights, se ofrece un amplio catálogo de estadísticas para consultar, contando entre otros, estadísticas sobre contribuidores, tráfico de datos en el repositorio, información acerca de los commits realizados, frecuencia de código en el proyecto, ramas etc.

  Es una muy buena herramienta, con una apariencia clara y limpia, con una interfaz muy intuitiva, sin embargo, esta característica únicamente es accesible a repositorios públicos, o mediante la obtención de una cuenta GitHub Premium, obtenida mediante el pago de una cuota mensual.

  En cuanto al uso del profesorado, principalmente encontramos la desventaja de que los repositorios de los alumnos no pueden ser públicos, además del requerimiento de los profesores de tener una cuenta premium. Por otra parte, no todos los alumnos

usan GitHub como servicio para repositorios Git, también se ofrece la posibilidad de usar GitLab o otras alternativas, lo cual haría que esta solución deje de ser válida para estos casos.

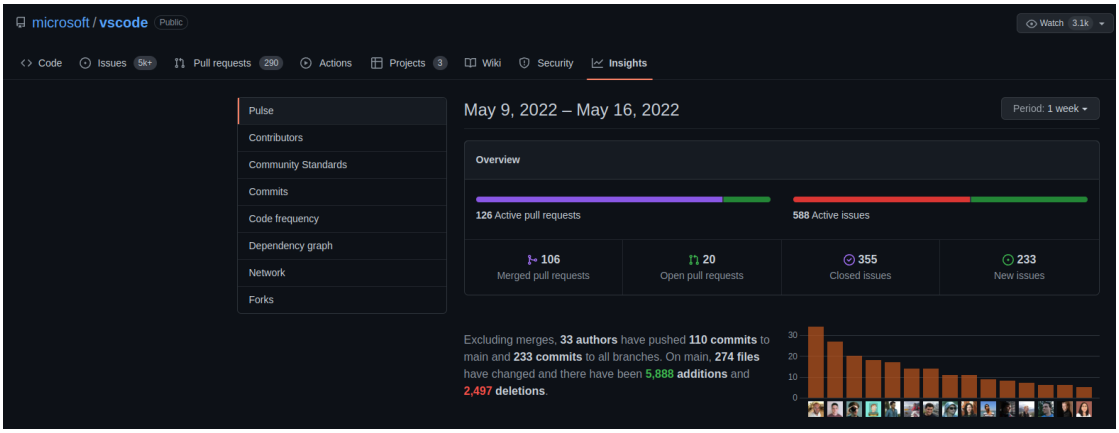


Figura 3.1: Ejemplo de la web de estadísticas en la página oficial de GitHub.

En la figura 3.1 se muestra un ejemplo de un ejemplo de un panel de estadísticas de los múltiples que ofrece GitHub, como vemos, tienen una apariencia limpia y muy clara, con una interfaz sencilla de usar y muy cómoda. Para el ejemplo se ha tomado el repositorio de ”vscode”de microsoft, como vemos es un repositorio público lo que permite ver las estadísticas completas, como ya se ha comentado, en caso de querer visualizarlas para un repositorio privado al cual tengamos acceso, ya sea siendo propietarios del mismo o como colaborador, necesitaríamos contar con una cuenta premium para poder visualizarlas, teniendo que realizar un pago mensual para ello.

En la parte izquierda, podemos visualizar un menú seleccionable, en el cual elegimos que estadísiticas queremos visualizar, ofreciendo distintas posibilidades, siendo para el caso tratado las mas interesantes las relacionadas con los commits, contribuidores y la frecuencia con la que se ha programado.

- **GitHub Education**: De la mano de GitHub, se ofrecen múltiples servicios de pago destinados a la educación. Estos servicios son aplicables a distintas instituciones educativas y de distintos tamaños. Estas pueden llegar a un convenio con GitHub, contando con múltiples tipos de ayudas para la contratación de servicios.

  Entre los servicios distinguimos entre herramientas para el profesorado y herramientas para el alumnado. Ofreciendo diferentes ventajas destinadas a los requerimientos de cada uno. En este caso, el análisis está más enfocado en cuanto a las ventajas que puede obtener el profesorado, a modo de comparativa con la herramienta desarrollada, cuyo fín es el soporte a los profesores para llevar el control de los alumnos.

Desde la página web de GitHub, encontramos una guía donde se indican claramente los pasos que se han de seguir para darse de alta como profesor, indicando que el proceso conlleva alrededor de unos 15 minutos, una vez realizado el registro, se cuentan con las siguientes herramientas:

- Acceso a "Education Community", un lugar donde los educadores pueden comentar sus ideas acerca de las tendencias de la educación tecnológica. Pudiendo así comentar, investigar y aprender nuevas formas de comunicación, enseñanza y esquemas de trabajo para sus alumnos. Además, se pueden consultar dudas acerca de cualquier problema relacionado con el uso de las herramientas ofrecidas, contando con una amplia comunidad activa, que da solución a múltiples consultas y problemas.
- Posibilidad de solicitar un "botín de GitHub", incluyendo este beneficios educativos y material destinado a los estudiantes. En caso de ser aprobada la solicitud, se reciben múltiples tutoriales y guías sobre el uso de Git, posters, stickers, y algunas tarjetas de regalo de camisetas de GitHub, canjeables por los alumnos en la web, destinados a ofrecer a modo de premio a los estudiantes con mejor desempeño.
- Acceso a "GitHub Teams", permitiendo tener un número ilimitado de usuarios y repositorios privados.
- El software completo de "GitHub classroom", una aplicación web tanto para docentes como alumnos, que permite entre otras, (i) la creación de aulas virtuales, donde los alumnos y maestros interactúan a lo largo de la duración del curso. (ii) Creación de tareas, tanto de manera individual como grupales, (iii) Crear plantillas a partir de un repositorio inicial de tal forma que el código sea sencillamente distribuible a los alumnos. (iv) Programar tareas con una calificación automática, es decir, una vez los alumnos entregan la tarea, siguiendo las indicaciones asignadas a la tarea, esta se autoevalúan, teniendo acceso en tiempo real de parte de los alumnos de las calificaciones y haciendo más eficiente el trabajo del profesor. (v) Control total de los repositorios creados sobre el aula de trabajo, permitiendo, tanto comunicación con los alumnos, como un amplio catálogo de estadísticas y herramientas para la evaluación.
- Solicitud de servicios cloud, aplicados a la enseñanza de nuevas tecnologías, contando con colaboradores de alta influencia en la actualidad de los servicios cloud, como lo son Digitalocean o Azure.

- **Estadisticas de GitLab**: En el caso del proveedor GitLab, si se ofrece soporte para la consulta de estadísticas de los repositorios, permitiendo ver gráficas sobre los commits realizados, la distribución temporal de los mismos, además se ofrecen datos acerca del "coverage" del código del repositorio, y porcentajes de uso en caso de utilizarse más de un lenguaje de programación sobre el proyecto.

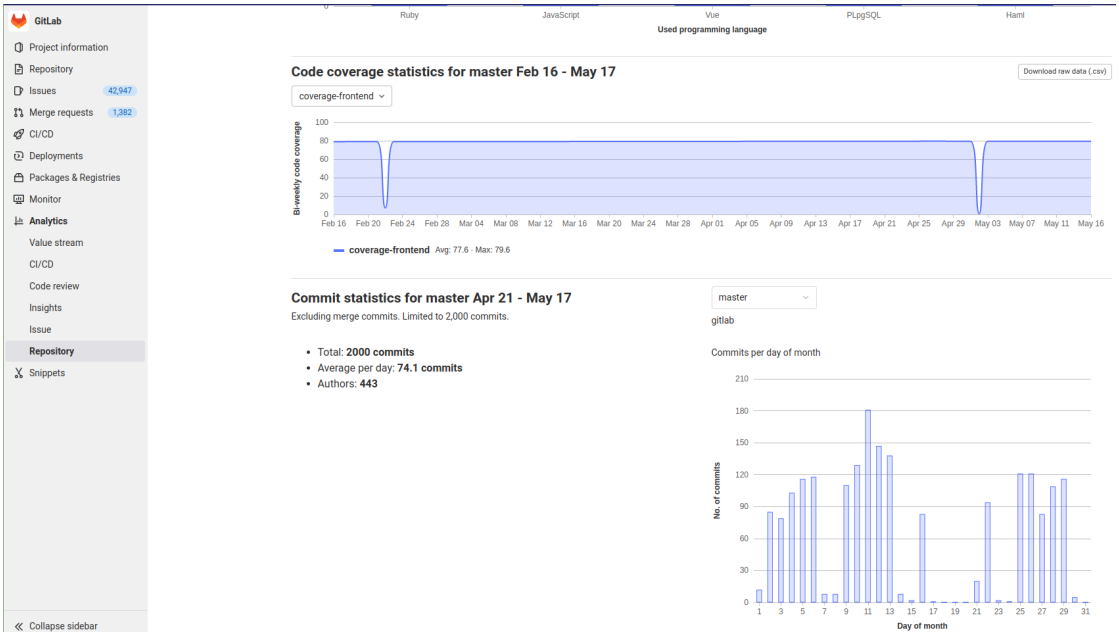


Figura 3.2: Ejemplo de la web de estadísticas en la página oficial de GitLab.

En la figura 3.2, se muestra una captura de la web de GitLab, y mostrando información relacionada con un repositorio público, podemos encontrar a la izquierda una sección en la que seleccionar las diferentes analiticas ofrecidas para el repositorio.

Respecto a la accesibilidad de los datos, en el caso de la web de GitHub, se muestran de forma más clara y accesible, conteniendo además datos de mayor relevancia para el propósito que se está tratando y más cantidad de información. En cuanto el posible uso del profesorado de GitLab para la consulta de estadísticas, se destaca que el acceso a los repositorios, no es el más rápido, en mayor instancia en caso de ser un número alto de repositorios, pudiéndose hacer difícil y tedioso la consulta de todos ellos.

- **GitLab for education**: Al igual que GitHub, GitLab también ofrece servicios destinados a la educación, en este caso también son servicios de pago y como en el otro caso, con posibilidad de solicitar múltiples tipos de subvenciones para la institución donde se quiera aplicar el programa para la educación de GitLab.

  En cuanto a lo ofrecido, los principales objetivos del programa de educación son: (i) Creación de cuentas para los usuarios de la institución educativa sin límites. (ii) Colaboración entre profesorado y alumno. (iii) Acceso al foro de GitLab con categorías específicas para la educación y un amplio catálogo de miembros. (iv) Guias para el alumnado sobre las tecnologías ofrecidas (v) Acceso a los eventos "Hackaton" donde los alumnos pueden competir en jornadas de programación,

colaborando con problemas de la comunidad, y accediendo a un amplio número de premios para los mejores contribuidores.

En cuanto a las diferencias con GitHub education, ofrecen servicios que no solo son válidos en la educación, ofreciendo distintos planes tanto para empresas e instituciones de cualquier tipo. El servicio de GitHub está plenamente enfocado en la educación, y ofrece un control al profesorado más amplio que GitLab, permitiendo la creación de aula, tareas de distintos tipos etc.

3.1.1 Limitaciones de las herramientas existentes y ventajas de la herramienta implementada

Las herramientas de analisis de repositorios en múltiples casos, como hemos visto, requieren una suscripción de pago a un servicio, excepto en el caso de GitLab en la que su panel de estadísticas si se ofrece de manera abierta tanto para repositorios públicos y privados. Sin embargo, en los paneles de estadísticas ofrecidos por ambos proveedores, no se muestra de forma rápida el contenido de los commits, teniendo que acceder previamente a los commits y visualizarlos uno por uno, esto es un aspecto bastante negativo ya que para la correción de los alumnos, es necesario visualizar el interior de los commits, ya que en muchos casos estos pueden contener simples modificaciones que realmente no conllevan un gran trabajo por parte de dicho alumno, esto indica que únicamente con visualizar el numero de commits de cada alumno y la distribución temporal de la misma no es suficiente y es necesario visualizar el contenido de los mismos.Encontramos por tanto, que el profesorado debe acceder por separado a los commits y las estadísticas en las soluciones propuestas, esto repetido para cada uno de sus alumnos.

Por otra parte, sobre los servicios ofrecidos especificamente para las instituciones, son una gran oportunidad de integrar en la institución la herramienta de Git, ofreciendo múltiples ventajas tanto para la misma institución como para sus integrantes. Sin embargo, estas soluciones requieren de un convenio entre la institución y el proveedor de servicios, esto en muchos casos no es sencillo, conllevando múltiples gestiones y requerimientos, además teniendo de tener en cuenta el coste que conlleva para la institución la integración de dichos servicios.

Como conclusión, tras conocer diferentes alternativas válidas para el control del trabajo realizado por parte del alumnado, podemos destacar los siguientes beneficios de la herramienta desarrollada en este trabajo:

- Una herramienta libre de pagos, tanto por parte del profesorado como de la institución.
- Control de los datos del profesorado, estos datos se manejan de forma interna en la institución de la universidad de Murcia, teniendo el control total de dichos datos, y no dependiendo de un servicio externo.

- Herramienta desarrollada por la propia Facultad, con un margen de mejora muy amplio y posibilidad de implementar nuevas funcionalidades, que se adapten específicamente a las necesidades de la institución. Pudiendo además extrapolar a distintas facultades para la gestión de distintos proyectos.
- Es un proyecto plenamente escalable, mediante el uso de contenedores Docker pudiendo ofrecer un buen rendimiento incluso bajo un nivel de requerimiento alto
- Un acceso rápido y eficiente a las estadísticas de los repositorios de los alumnos, obteniendo la información necesaria con el menor número de çlicks"posible.

4 Objetivos del proyecto

Este proyecto pretende ofrecer una herramienta desarrollada en la facultad de Informática de la Universidad de Murcia, que permita analizar los repositorios de Git en múltiples casos usados por sus alumnos, facilitando la tarea de la corrección, y aportando la información que refleja el trabajo por parte de los alumnos al trabajar en sus repositorios. Pudiendo obtener de dichos datos, la frecuencia de trabajo de los alumnos, casos en los que puede que algún alumno haya tenido un mayor desempeño y carga de trabajo que otro.

Por otra parte, se tiene como objetivo poder llevar la herramienta desarrollada a un servicio real, pudiendo hacer accesible el servicio para la facultad, por lo que el servicio debe contar con una arquitectura que permita su despliegue y escalado.

Como requisitos de la aplicación encontramos:

- Gestión de cuentas para el profesorado, contando con la posibilidad de realizar un registro, y posteriormente un login con la cuenta creada.
- El profesorado debe contar con una cuenta válida de GitHub, cuyo nombre de usuario debe ser el mismo que el creado en la cuenta para el servicio implementado.
- Cada tutor debe contar con un token de GitHub que se introducirá junto con el registro, de tal forma que el software pueda acceder a los repositorios privados del tutor.
- El software debe facilitar la gestión de asignaturas, donde los tutores puedan representar sus asignaturas, utilizando una cadena de coincidencia, que los alumnos introducirán en en nombre del repositorio, de tal forma que se realice el filtrado de los repositorios de cada asignatura.
- Se debe ofrecer la posibilidad de añadir un repositorio concreto en una asignatura, de forma persistente, para aquellos casos en los que la cadena de coincidencia no sea correctamente introducida por los alumnos.
- Se ofrece una barra de búsqueda por cadena de caracteres para el filtrado de los repositorios.

- Se muestran estadísticas sobre el número de colaboradores del proyecto, y su identificación.
- Se muestran estadísticas sobre el número de commits totales realizados por cada contribuyente y la distribución en el curso de los mismos.
- Se muestra de forma simple y clara el contenido de los commits (Patch), pudiendo visualizar los cambios realizados sobre cada archivo del repositorio.

5 Diseño e implementación

El proyecto, como ya se comentó en la aplicación está desarrollado principalmente con NodeJS sirviéndose del framework Express, dicha elección tuvo relevancia frente al uso de Python con el framework express dada la buena integración del lenguaje Javascript con el formato JSON, y la facilidad que ofrece para realizar peticiones a un API REST que proporciona los datos en un formato JSON. En este caso, todos los datos de los repositorios, deben ser recuperados a través del proveedor de servicios Git, en este caso GitHub.

GitHub ofrece un completo API REST mediante el cual se puede realizar un control total de la herramienta Git sobre la cuenta de un usuario. Para ello, es necesario ofrecer un token de acceso, el cual se puede solicitar a través de la misma página web de GitHub y el cual es necesario para el funcionamiento y el acceso a los datos de los repositorios por parte de la aplicación desarrollada.

Como veremos en más detalle a continuación, la arquitectura de la aplicación se conforma con dos servidores implementados con NodeJS utilizando el framework Express, y una base de datos para ofrecer persistencia, en este caso una implementación de MySQL.

5.1 Arquitectura base de la aplicación.

En primer lugar, contamos con el servidor web, implementado con NodeJS-Express, este se encargará de servir las peticiones de los usuarios a la página principal de la aplicación web. Implementando en esta todo la interfaz para el usuario, donde se podrá realizar tanto login como usuarios, y donde se visualizará todo el panel de repositorios, asignaturas, y estadísticas.

Dicho servidor, necesita ofrecer persistencia, tanto para el registro de los usuarios, en este caso el profesorado, como para el posterior inicio de sesión en la aplicación. Además, tanto las asignaturas que los profesores puedan crear, y los repositorios que manualmente pueden añadir a dichas asignaturas deben ser persistidas sobre una base de datos para su persistencia. Para ofrecer dicha persistencia, se hace uso de una implementación de MySQL, debido a la fácil integración de los datos a persistir con el modelo relacional,

con una estructura basada en tablas. Para la integración del servidor web, con la base de datos, se utiliza la librería mysql, ofrecida de manera oficial por el la misma. A través de la misma, podremos mantener y actualizar la información procedente de las acciones de los usuarios en la página web.

Mediante la implementación del servidor web y el servicio de persistencia, podríamos obtener una solución viable y funcional, sin embargo, es conveniente, en este caso, separar el servidor web de la consulta de datos al servicio de GitHub. El servidor web está destinado al múltiple acceso de los usuarios, sobre el cual requerimos de un buen rendimiento y tiempo de respuesta, lo que puede verse afectado por la gestión de múltiples peticiones a otro servicio.

Por consiguiente, se ha implementado un segundo servidor, también desarrollado en NodeJS, el cual está encargado de realizar toda la comunicación con los servidores de Github, es decir, se encargará de obtener toda la información necesaria de los servicios de GitHub. De esta forma, contaremos con:

- **El primer servidor,**servidor web, el cual implementa todo el servicio web, destinado únicamente a recibir y responder las peticiones web de los usuarios. Este servidor será por tanto el encargado de comunicarse con la base de datos y gestionarla, realizando las consultas de registro y creación de usuarios, así como gestionar las asignaturas. En caso de necesitar datos provenientes del API Rest de Github, estos datos se solicitan al segundo servidor.

- **El segundo servidor,**, servidor proxy Rest sobre GitHub, implementa un servicio encargado de atender las peticiones del primer servidor e intercomunicarse con el API de GithuB para obtener los datos solicitados. Por tanto, actúa a modo de proxy para el servicio de GitHub.

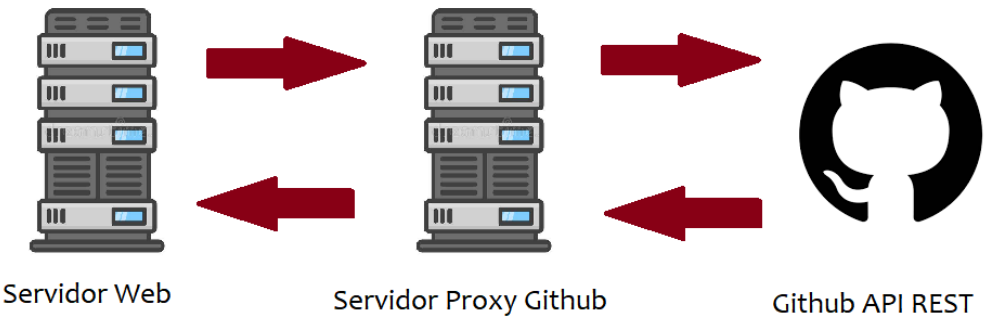


Figura 5.1: Grafico de funcionamiento de los servidores implementados y el servicio API de GitHub.

En la figura 5.1, se representa mediante un gráfico el flujo de comunicación entre los servidores implementados y el API que ofrece GitHub.

Respecto la comunicación de los dos servidores, el segundo servidor está implementado a modo de API REST, donde recibe peticiones HTTP, en este caso únicamente recibe peticiones de tipo Get, recibiendo por los parámetros de la url los datos que se desean obtener, vez recibida la petición y procesada, se comunica con el API de GIthub, obtiene los datos correspondientes y responde de nuevo al primer servidor.

```
//RUTAS DE ACCESO PARA PETICIONES A LA API REST.

app.get('/repos', function (request, response) {

        getReposUsuario(request.query.user, request.body.token).then(resu => response.send(resu));

});

app.get('/commits', function (request, response) {
        console.log("imprimo el token recibido : "+request.body.token)
         getCommitsRepo(request.query.repo, request.query.owner, request.body.token)
                .then(resu => response.send(resu))
                .catch(err => console.log(err))

})

app.get('/collaborators', function (request, response) {

        getColaboradoresRepo(request.query.repo, request.query.user, request.body.token).then(resu => response.send(resu))

})
```

Figura 5.2: Configuración del servidor proxy GitHub, implementación en NodeJS con Express.

En la figura 5.2, se muestran los diferentes metodos get y la forma de especificar que información se desea recuperar mediante el uso de parametros en la url de la petición.

Como hemos comentado, las causas de la implementación del segundo servidor, es destinar únicamente el servidor web para los usuarios y no sobrecargarlo con peticiones a otro servicio externo, reduciendo así su carga de trabajo y mejorando los tiempos de respuesta. Además, conseguimos de esta forma, separar la arquitectura de la aplicación, teniendo por una parte la implementación del frontend, a la cual tienen acceso los clientes , y el backend, encargado de proveer la información al frontend, de tal forma que ambos servicios son independientes y obteniendo un buen rendimiento en ambos, en aquellos casos en los que puedan ocurrir múltiples peticiones. Posteriormente, veremos esta separación, también permite distribuir mejor la aplicación desarrollada sobre un servicio cloud, permitiendo escalar cada servicio en medida de la sobrecarga que se produzca.

5.2 Consulta de datos al API de GitHub y autenticación.

Para la implementación de la aplicación, se utiliza la librería "Octokit", una librería ofrecida de forma oficial por el proveedor de servicios, y cual se integra con el lenguaje javascript, mediante ella, podemos realizar las consultas al API de Github de una manera sencilla y teniendo acceso a operaciones que requieren acceso mediante token. Con esta librería, las peticiones realizadas, se configuran con las cabeceras adecuadas para garantizar la seguridad necesaria.Vemos a continuación, un fragmento de código que representa en javascript la obtención de una instancia de la librería y una función utilizada para recuperar la información de un usuario a través del API proporcionada.

```
var octokit = new Octokit({ auth: "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" })


async function getUsuario(usuario) {
        var res = await octokit.rest.users.getByUsername({ username: usuario });
        return res;
}
```

Figura 5.3: Ejemplo de uso de la librería Ocktokit, ofrecida por GitHub.

En la figura 5.3, se muestra la creación e inicialización de un objeto "Octokit", objeto ofrecido por la librería, mediante el cual posteriormente se realizarán las peticiones HTTP, para su creación le brindamos el token del usuario, necesario para poder leer la información de sus repositorios.

5.3 Maquetado de la aplicación web.

La funcionalidad de la web debe ser, en un primer lugar, permitir registro e inicio de sesión. Y en segundo lugar, mostrar los repositorios, permitiendo el filtrado o bien por cadenas de caracteres, o bien por filtrado en base a asignaturas previamente creadas por el usuario de la sesión.

Para la implementación de la web, se basa en cuatro vistas diferentes, dos de ellas para registro e inicio de sesión, una para la página de inicio, donde se muestran los repositorios y asignaturas del usuario, y por último la página donde se visualizan las estadísticas del repositorio, esta última accesible al hacer click sobre algún repositorio en la página principal.

Para el diseño de las vistas, se ha utilizado tecnologías específicas para la tarea, dando estructura a las páginas con HTML, para posteriormente añadir con css y bootstrap estilos, y por último implementar animaciones y eventos en javascript.

En el caso de la página de las páginas de inicio de sesión y estadísticas, al solicitar las webs al servidor, este nos envía la estructura y diseño de la web, además del código javascript asociado a la página, mediante el cual, se solicitarán de manera asíncrona, la información al servidor para completar las vistas. Por ejemplo, en el caso de acceder a la página principal, el servidor nos responde con una tabla vacía, que justo cuando se recibe, se realiza otra petición asíncrona al servidor que una vez resuelta, hará que los datos se muestren en el interior de la tabla.

Para ello, implementamos mediante javascript funciones que se ejecutarán una vez la página se haya cargado en el navegador que realicen el completado de la vista con la información obtenida. De esta forma, logramos que la respuesta a la petición sea inmediata, y posteriormente cuando los datos sean recuperados se muestran, así, el usuario no percibe un retraso al acceder al sitio web, como ocurriría si esperamos a enviar la web desde el servidor una vez los datos de GitHub se hayan recuperado.

5.4 Contenerización y despliegue de aplicación.

El segundo objetivo del proyecto, es ofrecer una implementación útil para el despliegue en un servicio cloud, de tal forma que los servicios, estén basados en contenedores, que puedan ser lanzados en una estructura de nodos cloud, permitiendo así su control y escalado. Por consiguiente, se ofrecen junto el proyecto, las herramientas necesarias para formar contenedores con cada uno de los servicios implementados, en primer lugar, una imagen Docker para cada uno de los servidores Node, y un fichero docker-compose como herramienta para simplificar el despliegue, de tanto, ambos servidores en Node, como un contenedor para la base de datos MySQL, configurado e inicializado para funcionar de forma integrada con los otros servicios.

Estos contenedores, en caso de contar con un proveedor cloud, o el hardware correspondiente, se pueden desplegar y escalar, por ejemplo, usando para el propósito un orquestador como podría ser Kubernetes, uno de los gestores de contenedores más popular a día de hoy. Permitiéndonos establecer configuraciones donde podríamos lanzar los servicios y en caso de caída de alguno de ellos levantarse de forma automática. Además, ofrece posibilidades como el escalado automático en momentos donde la carga de trabajo está creciendo de manera paulatina, repartiendo así la carga de trabajo y evitando el colapso.

En caso de querer realizar un despliegue normal, bien sea, en un ordenador personal o en un servidor, se ofrecen dos alternativas de despliegue:

1. Desplegar los servidores de manera manual, iniciándose mediante consola. Usando para ello los comandos:

   - *"npm install"*,para instalar las dependencias necesarias, en caso de no estas previamente instaladas.
   - *"node app.js"*, para lanzar el servidor, quedando en escucha en el puerto 3000.

   Este proceso debemos repetirlo también para el segundo servidor contenido en la carpeta "restjs", cambiando el segundo comando por "node app2.js".

   Con esto lanzaremos ambos servidores, pero tendríamos que tener instalada y configurada una instancia de MySQL, la cual podríamos inicializar utilizando el fichero "schema.sql" situado en la carpeta "scripts".

2. La segunda opción, por otro lado, sería utilizar el Docker-compose ofrecido junto el proyecto, para el cual únicamente deberíamos ejecutar "docker compose up".

   Este comando se encargaría de construir las imágenes de ambos servidores, recuperar una imagen oficial de MySQL desde el repositorio Docker Hub, configurar los volúmenes necesarios, y crear y configurar la subred necesaria para la intercomunicación de los mismos.

```
version: '2'

services:
  nodejs:
    build:
      context: .
      dockerfile: Dockerfile
    image: nodejs
    container_name: nodejs
    restart: unless-stopped
    environment:
      - MYSQL_HOST=db
      - PROXY_REST=restjs
    ports:
      - "3000:3000"
    volumes:
      - .:/home/node/app

  restjs:
    build: ./restjs
    restart: unless-stopped
    ports:
      - "3001:3001"
    volumes:
    - .:/home/node/restjs
    expose:
      # Opens port 3001 on the container
      - '3001'


  db:
    image: mysql
    restart: always
```

```
34      command: --default-authentication-plugin=mysql_native_password
35      environment:
36        MYSQL_DATABASE: 'nodelogin'
37        # So you don't have to use root, but you can if you like
38        MYSQL_USER: 'user'
39        # You can use whatever password you like
40        MYSQL_PASSWORD: 'root'
41        # Password for root access
42        MYSQL_ROOT_PASSWORD: 'root'
43      ports:
44        # <Port exposed> : < MySQL Port running inside container>
45        - '3005:3306'
46      expose:
47        # Opens port 3306 on the container
48        - '3005'
49        # Where our data will be persisted
50
51      volumes:
52        - "./scripts/schema.sql:/docker-entrypoint-initdb.d/1.sql"
```

Vemos la configuración del .yml construido para el arranque de los servicios mediante Docker-Compose, es de destacar, la configuración de los servicios, introduciendo como variables de entorno los nombres asignados, de esta forma se resuelve el direccionamiento de forma automática, mediante una subred que crea Docker-Compose y mediante dichas variables posteriormente en el programa podemos acceder al resto de servicios configurados. Además, destacamos la configuración de los volúmenes, y la inicialización de la base de datos, para contruir el schema con un fichero de inicialización para MySQL.

6 Conclusiones y vías futuras

6.1 Conclusiones

Conclusiones...

6.2 Futuros trabajos

Futuros trabajos...

Bibliografía

[1] IBM docs. Formato json. www.ibm.com/docs/.

A Clases Java Bean

Las clases Java Bean es un estándar que hace referencia a la definición de clases de negocio con unos requisitos en concreto 3:

1. **Getters y Setters de atributos privados**: Las clases java Bean deben almacenar todos sus atributos de forma privada. Para poder acceder a sus datos y modificarlos contendrán funciones Getters y Setters. Las funciones Getters y Setters son funciones que devuelven o modifican el valor de un atributo. Además estas funciones se deben construir con la nomenclatura de get o set seguido del atributo que se desee en camelCase, es decir con la primera letra del atributo en mayúscula. Por ejemplo, para un atributo llamado nombre sus métodos get y set se llamarían 'getNombre' y 'setNombre'.
2. **Constructor por defecto**: Es necesario que exista un constructor sin parámetros y con visibilidad pública para poder instanciar el objeto vacío.
3. **Implementar Serializable**: Las clases Bean deben implementar la clase Serializable. El interfaz Serializable es un interfaz de marca que no contiene ningún método pero que permite que los objetos sean serializables a disco o a red.

```
1  public class Mueble implements Serializable {
2    private Color color;
3    private String nombre;
4    private int patas;
5
6    public Mueble() {
7
8    }
9
10   public Color getColor() {
11     return color;
12   }
13
14   public void setColor(Color color) {
15     this.color = color;
16   }
17
18   public String getNombre() {
19     return nombre;
20   }
21
```

```
22      public void setNombre(String nombre) {
23        this.nombre = nombre;
24      }
25
26      public int getPatas() {
27        return patas;
28      }
29
30      public void setPatas(int patas) {
31        this.patas = patas;
32      }
33    }
```

En esta clase de ejemplo se crea una clase Java Bean llamada Mesa con tres atributos. El código escrito anteriormente es el mínimo necesario pero sería posible seguir completando la clase con nuevos atributos, funciones o constructores.

Gracias a esto las clases pueden ser gestionadas por diferentes librerías que gestionan la creación, gestión y guardado de objetos por medio de sus clases Bean. Un ejemplo de ellos es jackson una popular librería capaz de almacenar objetos de clases Bean y serializarlos en formato XML, CSV o JSON.