Subproyecto de Integración

1. Descripción

El objetivo del proyecto es desarrollar una herramienta de minería de repositorios Git que permita cargar, procesar y analizar datos de proyectos Git alojados en distintas plataformas. Para ello, se desarrollará una aplicación con una arquitectura de tres microservicios, mostrada en la Figura 1. Los servicios *BitbucketMiner* y *GitHubMiner* tendrán como objetivo leer datos de las APIs de Bitbucket y GitHub, respectivamente, y enviarlos a *GitMiner* para su almacenamiento usando un modelo de datos común. Por su parte, *GitMiner* almacenará los datos y los ofrecerá al exterior a través de una API REST para que otras aplicaciones puedan consultarlos y analizarlos.

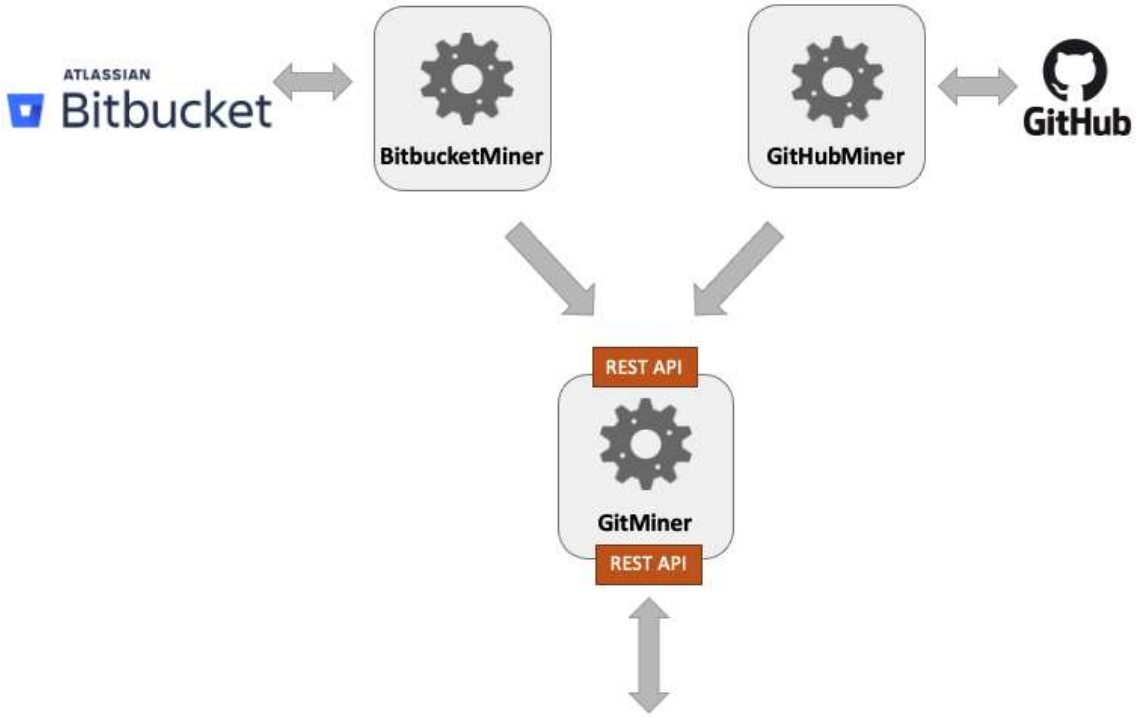

Figura 1. Vista general de la arquitectura

A continuación, se detalla el funcionamiento de cada servicio:

GitHubMiner. Se trata de un servicio adaptador que leerá los datos desde la API REST de GitHub y los enviará a GitMiner usando el modelo de datos adecuado (Figura 2). En un escenario de ingeniería de datos real este proceso podría realizarse periódicamente (ej. cada 24 horas). En este caso, el servicio implementará un servicio RESTful con una operación para obtener los datos de GitHub y enviarlos a GitMiner cada vez que sea invocada. En concreto, la operación deberá poder ser invocada de la siguiente manera, recibiendo como parámetros obligatorios el nombre del dueño y el nombre del repositorio:

POST *apipath*/{owner}/{repoName}[?sinceCommits=5&sinceIssues=30&maxPages=2]

Parámetros opcionales:

- sinceCommits: La operación devolverá los commits enviados en los últimos X días, siendo X el valor introducido como parámetro. Valor por defecto: 2.
- sinceIssues: La operación devolverá los issues actualizados en los últimos X días, siendo X el valor introducido como parámetro. Valor por defecto: 20.
- maxPages: Número máximo de páginas en los que se iterará en todos los casos. Valor por defecto: 2.

Se recomienda implementar también una operación equivalente de sólo lectura para hacer pruebas, es decir, que muestre los resultados de la búsqueda sin enviarlos a GitMiner.

BitbucketMiner. Su funcionamiento será análogo al del servicio anterior, pero en este caso la información se tomará de Bitbucket, a través de su API REST. En este caso, la operación que debe implementar varía ligeramente, recibiendo como parámetros obligatorios el nombre del espacio de trabajo y el identificador del repositorio:

POST *apipath*/{workspace}/{repo_slug}[?nCommits=5&nIssues=5&maxPages=2]

Parámetros opcionales:

- nCommits: La operación devolverá X commits por página, siendo X el valor introducido como parámetro. Valor por defecto: 5.
- nIssues: La operación devolverá X issues por página, siendo X el valor introducido como parámetro. Valor por defecto: 5.
- maxPages: Número máximo de páginas en los que se iterará en todos los casos. Valor por defecto: 2.

GitMiner. Será el servicio responsable de integrar y almacenar todos los datos en una base de datos H2 usando el modelo de datos mostrado en la Figura 2. Para ello, implementará una API REST que permitirá manipular los siguientes recursos:

- Projects. Implementará varias operaciones de lectura y escritura para añadir y listar proyectos. Entre otras, tendrá que implementar la operación POST adecuada para que los adaptadores puedan añadir datos de nuevos proyectos.
- Commits. Implementará, como mínimo, varias operaciones de lectura para listar todos los commits, y buscar commits por id.
- Issues. Implementará, como mínimo, varias operaciones de lectura para listar todos los issues, buscar issues por id y por estado.
- Comments. Implementará, como mínimo, varias operaciones de lectura para listar todos los comentarios y buscar comentarios por id.

2. Descripción

Como material de apoyo, se proporciona:

1. Proyecto plantilla con las clases del modelo de datos de GitMiner, necesarias para la creación de la base de datos H2 y la manipulación de los datos recibidos desde los servicios adaptadores.
2. Conjunto de pruebas de Postman para validar el funcionamiento básico de GitMiner (descargar en Enseñanza Virtual).
3. Video demo ilustrando el comportamiento esperado de la aplicación y cómo probarla.

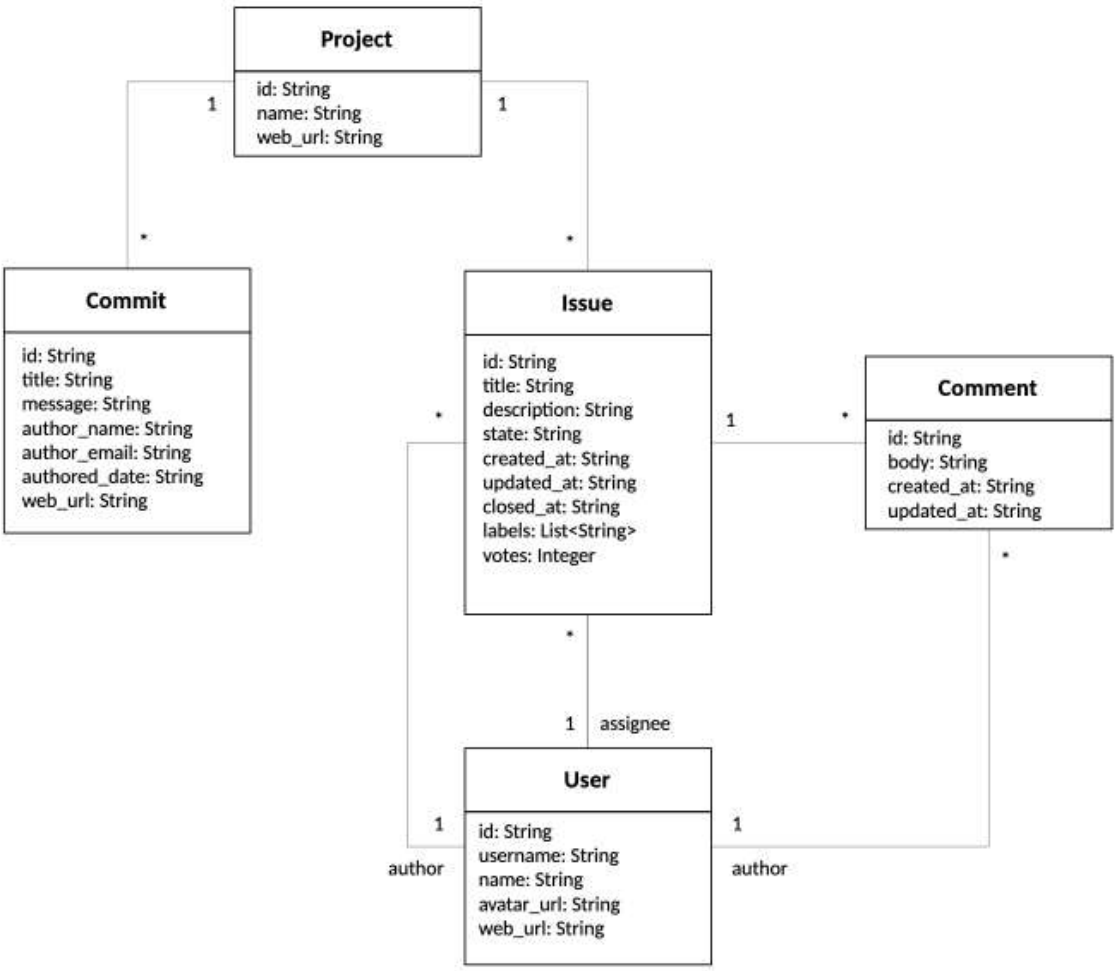

Figura 2. Modelo de datos

3. Criterios de evaluación

3.1. Criterios mínimos (IMPRESDINCIDIBLES PARA APROBAR)

Los tres servicios deberán funcionar correctamente de acuerdo con las instrucciones indicadas. Es imprescindible que todas las pruebas de Postman facilitadas como parte del material de apoyo pasen con éxito. No se aceptarán aplicaciones que no se ajusten a la arquitectura descrita anteriormente.

3.2. Criterios estándar (necesarios para obtener una calificación de notable)

Además de los criterios mínimos, se valorarán los aspectos de diseño de la aplicación, así como el nivel de detalle de las pruebas (unitarias con Junit y de sistema con Postman) y la documentación interactiva de la API (OAS). También se tendrá en cuenta el nivel de participación y la claridad de la exposición durante la defensa.

3.3. Criterios avanzados (necesarios para obtener una calificación de sobresaliente)

Además de los criterios estándar, aquellos grupos que quieran optar a sobresaliente deberán implementar unas o varias mejoras acordadas con su tutor/a. Algunos ejemplos:

- Extensión del modelo de datos.
- Extensión de la API proporcionada por GitMiner (ej. paginación, filtros, ordenación).
- Extraer datos de más plataformas (ej. GitLab).
- Uso de APIs GraphQL.