Integration project

1. Description

The objective of the project is to develop a Git repository mining tool to load, process and analyze data from Git projects hosted on different platforms. To this end, an application will be developed with a three microservices architecture, shown in Figure 1. The *BitbucketMiner* and *GitHubMiner* services will aim to read data from Bitbucket and GitHub, respectively, and send it to *GitMiner* for storage using a common data model. *GitMiner*, on the other hand, will store the data and offer it to the outside world via a REST API so that other applications can query and analyze it.

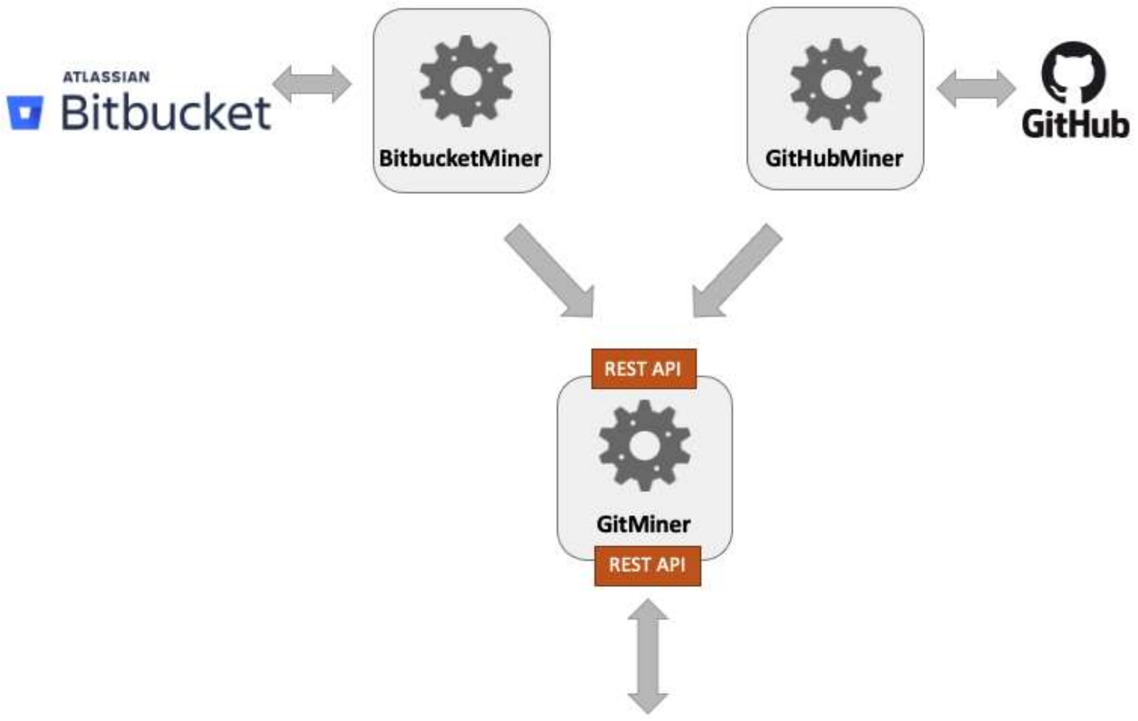

Figure 1. General view of the system architecture

In what follows, we detail the operation of each service:

GitHubMiner. This is an adapter service that will read data from the GitHub REST API and send it to GitMiner using the appropriate data model (Figure 2). In a real data engineering scenario this process could be performed periodically (e.g. every 24 hours). In this case, the service will implement a RESTful service with an operation to get the data from GitHub and send it to GitMiner each time it is invoked. Specifically, the operation shall be able to be invoked as follows, providing mandatory parameters for name of the owner and name of the repository:

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

Optional parameters:

- sinceCommits: The operation will return the commits sent in the last X days, where X is the value entered as parameter. Default value: 2.

- sinceIssues: The operation will return the issues updated in the last X days, where X is the value entered as parameter. Default value: 20.
- maxPages: Maximum number of pages to be iterated in all cases. Default value: 2.

It is recommended to also implement an equivalent read-only operation for testing purposes, i.e. displaying the search results without sending them to GitMiner.

BitbucketMiner. Its operation will be analogous to the previous service, but in this case the information will be taken from Bitbucket, through its REST API. In this case, the operation to be implemented varies slightly, receiving as mandatory parameters the name of the workspace and the identifier of the repository:

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

Optional parameters:

- nCommits: The operation will return X commits per page, where X is the value entered as parameter. Default value: 5.
- nIssues: The operation will return X issues per page, where X is the value entered as parameter. Default value: 5.
- maxPages: Maximum number of pages to be iterated in all cases. Default value: 2.

GitMiner. It will be the service responsible for integrating and storing all the data in an H2 database using the data model shown in Figure 2. To do so, it will implement a REST API that will allow manipulating the following resources:

- Projects. You will implement several read and write operations to add and list projects. Among others, you will need to implement the appropriate POST operation so that adapters can add data from new projects.
- Commits. It will implement, at least, several read operations to list all commits, and search for commits by id.
- Issues. It will implement, at least, several read operations to list all issues, search for issues by id and by status.
- Comments. It will implement, at least, several read operations to list all comments and search for comments by id.

2. Description

As support material, we provide:

1. Template project with the classes of the GitMiner data model, necessary for the creation of the H2 database and the manipulation of the data received from the adapter services.
2. Postman Test suite to validate the basic operation of GitMiner (see Virtual Teaching).
3. Video demo illustrating the expected behavior of the application and how to test it.

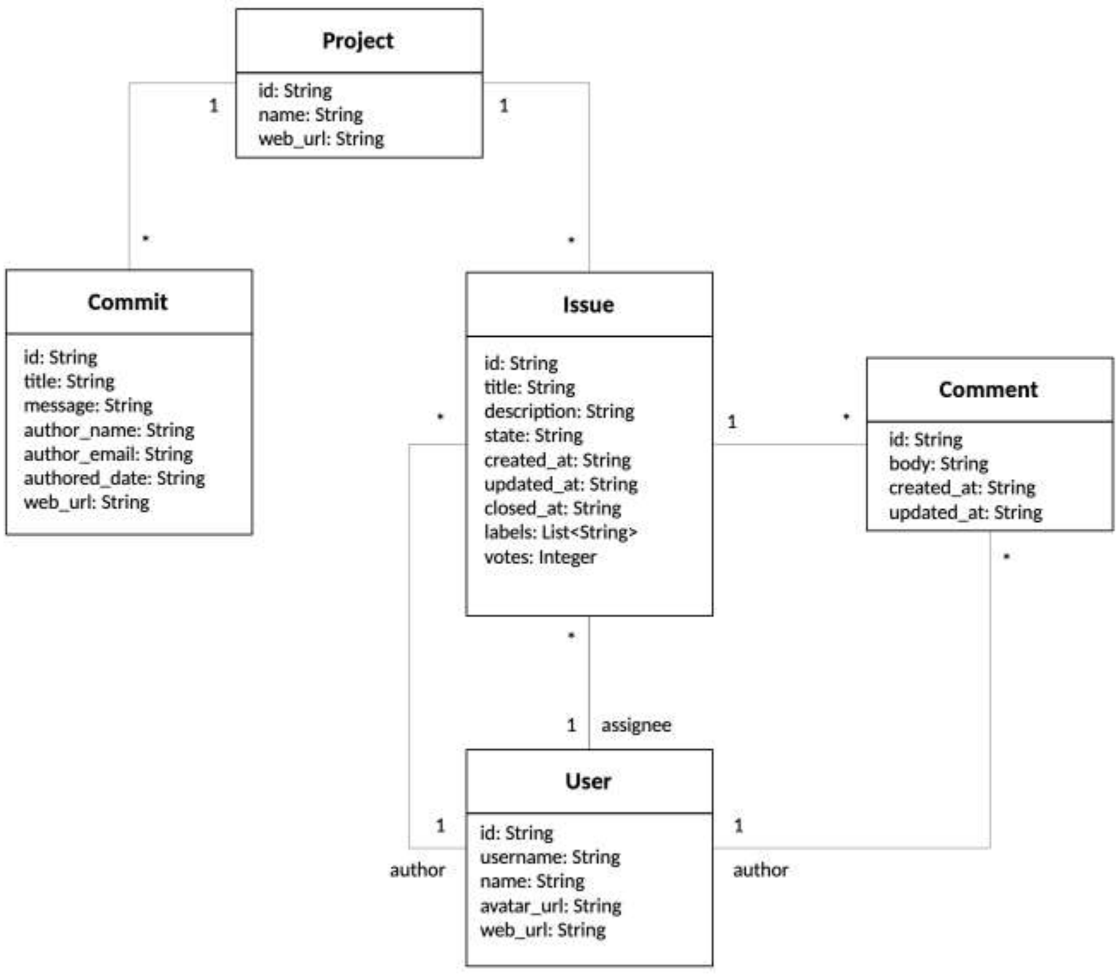

Figura 2. Data model

3. Evaluation criteria

3.1. Minimum Criteria (REQUIRED FOR PASSING)

All three services must function correctly according to the instructions provided. It is imperative that all Postman tests provided as part of the supporting material pass successfully. Applications that do not conform to the architecture described above will not be accepted.

3.2. Standard criteria (required to obtain a "B" grade)

In addition to the minimum criteria, the design aspects of the application will be evaluated, as well as the level of detail of the tests (unit tests with Junit and system tests with Postman) and the interactive documentation of the API (OAS). The level of participation and the clarity of the presentation during the defense will also be taken into account.

3.3. Advanced criteria (necessary to obtain an "A" grade)

In addition to the standard criteria, those groups that want to qualify for an "A" must implement one or more improvements agreed upon with their tutor. Some examples:

- Extension of the data model.

- Extension of the API provided by GitMiner (e.g. pagination, filters, sorting).
- Extracting data from more platforms (e.g. GitLab).
- Use of GraphQL APIs.