GitClout (developer documentation)

Table of Contents

Introduction

GitClout is a web application that allows users to analyze tags from a public repository (Github or Gitlab for example) and display information about the contributors.

The application is built using the following technologies:

- Micronaut (Java framework)
- JPA (Java Persistence API)
- SQLite (Database)
- SolidJS (Javascript framework)
- Bootstrap (CSS framework)

See the README.md file for more information about the application.

Architecture

Backend (Java)

- The entry point of the application is the `GitCloutController` class in `app` package.
- The controller is defined in the `GitCloutController` class in `app` package.

The application uses the following packages:

- `app`: contains the controller and the entry point of the application.
- `contributions`: contains the classes that represent the contributions and the service that handles the analysis of the contributions.
- `repositories`: contains the classes that represent the repositories and the service that handles fetching the repositories.
- `tags`: contains the classes that represent the tags and the service that handles fetching the tags.
- `languages`: contains the classes that represent the languages and the service that handles getting the supported languages.

Frontend (Javascript)

- The entry point of the application is the `index.tsx` file in `src` folder.
- All of the pages are defined in the `pages` folder.

- The components are defined in the `components` folder.

Database

The database is created at the startup of the application at the root of the project in the `gitclout.db` file.

The database contains the following tables:

- `repository`: contains the repositories that have been analyzed.
- `tag`: contains the tags that have been analyzed.
- `contribution`: contains the contributions that have been analyzed.
- `contribution_details`: contains the details of the contributions that have been analyzed.

API

The API is defined in the `GitCloutController` class in `app` package. A Swagger UI is available at the `/swagger-ui` endpoint.

Analysis

For the analysis of the contributions, the application works as follows:

- Check if the tags of the repository have already been analyzed.
- Analyze the tags that have not been analyzed yet.

The analysis of the tags works as follows:

- Get every files of the tag.
- Add to a list of `Callable` the analysis of each file.
- Execute the `Callable` in a thread pool.
- Wait for the `Callable` to finish.
- Save the results in the database.

The analysis of a file works as follows:

- Get the content of the file.
- If the file is CODE, get the comments of the file.
- Blame every line of the file.
- For each line, get the author of the line.

`pom.xml`

The `pom.xml` file contains the dependencies of the application.

We use `com.github.eirslett:frontend-maven-plugin` to build the frontend of the application. That's why the first `mvn package` command will take a long time to execute.

Warnings

Some warnings are displayed when the application is building : `No processor claimed any of these annotations`. We could not find a way to remove them.

There are also some warnings displayed when the application starts `WARN  org.hibernate.orm.incubating - HHH90006001: Encountered incubating setting [hibernate.id.db_structure_naming_strategy]`. We could not find a way to remove them.

Improvements and difficulties

- The application could be improved by adding more tests.
- Better error handling could be added.
- The application could be improved by adding more features in the frontend.
- Better database management could be added.
- Remove the warnings.

The main difficulty of the project was to find a way to analyze the contributions of a tag. The solution that we found was to blame every line of the file and get the author of the line. This solution is very slow, that's why we use a thread pool to analyze the files in parallel.