Documentation Technique du projet de Java

Sommaire:

Introduction

Dans le cadre de notre projet de Master 1 en Informatique avec Java, nous avons développé une application Web destinée à analyser les répertoires Git, qui montre les contributions de chaque participant. Utilisant des technologies comme **Micronaut**, **Angular**, **JPA** (**Jakarta**), **H2** et **Tailwind**, cette application n'est pas seulement un outil d'analyse technique mais aussi un moyen de valoriser le travail collaboratif et de mieux comprendre la dynamique des contributions dans les projets de développement logiciel.

Spécifications Techniques

1) Descriptions des technologies de l'application

Pour l'architecture de notre application nous devions utiliser **Micronaut** qui va nous servir de **back-end**, gérant les services, **Angular**, utilisé pour le front-end ce qui va permettre l'interaction avec le l'utilisateur **JPA** qui va nous permettre de gérer la Base de données, alors que **H2** va nous permettre de stocker notre Base de données, et **Tailwind CSS** pour améliorer l'esthétique.

Architecture de l'Application

1) Structure générale et modularité

L'architecture de notre **application** se reflète dans la structure de ses dossiers et fichiers. Le dossier **api** c'est tout ce tout ce va nous permettre de la communication entre le back-end et le **front-end** ou notamment pour la communication via **Websocket** qui nous permet de savoir l'avancée de l'analyse.
Nous avons dans le dossier **Database** toutes les tables de notre **base de données**.
Sous le dossier **DTO** nous avons le fichier qui vont nous servir pour le transport de données car nous ne voulons pas envoyer des données inutiles au front.
sous le dossier **Gitanalyze** nous avons tout ce qui va nous permettre d'analyser le répertoires le clonage du répertoire, le nombre de tags, le nombre de contributeurs les contributions par contributeur et ce qui va aussi nous permettre de déterminer sur quelle type de fichier le contributeur a écrit et également si c'est un commentaire.
Sous le dossier **gitcloutexception** nous avons les exceptions que nous avons crée pour le projet afin de pouvoir gérer nous même nos les exceptions.
Il y a également une javadoc et de la documentation technique et utilisateur dans le dossier docs/doc qui est dans le dossier resources .
Pour accéder au swagger il faut aller **http://localhost:8080/swagger-ui#/**

2) Interaction entre les composants

L'interaction entre les composants de l'application se fait de manière fluide et intégrée. **Les contrôleurs** dans le dossier api gèrent **les requêtes HTTP** et communiquent avec les services. **Ces services** interagissent avec la couche de **database** pour la persistance des données via **JPA** et la base de données **H2**. Le module **Gitanalyze** prend en charge l'analyse des données du dépôt Git.

Développement du Back-end (Micronaut)

1) Configuration et mise en place de microservice

Dans notre application, nous avons implémenté une architecture basée sur des **microservices**, y compris un service dédié à la gestion **des interactions avec la base de données**. Un autre service est spécialisé dans l'exécution d'une fonction spécifique, permettant au contrôleur de déléguer des tâches en appelant cette fonction.

2) Gestion des requêtes et intégration avec JPA et H2

Pour la gestions des requêtes entre **jpa** et **h2** nous avons des des interfaces donné par **CrudRepository** qui nous permet de d'avoir des fonctions pré-faite tel que **save** pour l'ajout, **delete** pour la suppression.
et pour cela nous avons fait un service contenant les interfaces afin d'avoir toutes les fonctions de la base de données dans un seul service.

3) Analyse des répertoires

Le code commence par l'analyse des contributions en fonction des tags. Il itère sur les tags pour examiner les différences de code entre les deux tags consécutifs. Cela permet d'analyser quelles contributions ont été apportées entre ces versions du dépôt.
Reconnaissance du Langage: Le code reconnaît le langage dans lequel chaque fichier est écrit en fonction de son extension. Cette reconnaissance facilite la catégorisation des contributions par langage de programmation, ce qui peut être utile pour l'analyse.
Gestion des Commentaires: Le code gère les commentaires dans le code source en identifiant les commentaires mono-ligne et multi-lignes. Cela permet de s'assurer que seules les lignes de code effectives sont prises en compte dans l'analyse des contributions.
Suivi des modifications: Le code effectue un suivi des lignes de code insérées et supprimées pour chaque contribution. Cela permet de quantifier les changements apportés par chaque contributeur et de suivre l'évolution des contributions au fil du temps.
Analyse des différences: Le code utilise des bibliothèques **jGit** pour obtenir les différences entre les commits associés aux tags. Ces différences sont ensuite analysées pour déterminer les contributions spécifiques.

Développement du Front-end (Angular, Tailwind)

1) Conception de l'interface utilisateur

L'utilisation des fonctionnalités de **Angular**, comme les composants dynamiques et les services injectables, a été cruciale pour structurer notre front-end, permettant une interaction efficace et cohérente avec le **back-end.** Nous avons adopté une **architecture modulaire**, où chaque fonctionnalité est isolée dans des composants distincts, facilitant ainsi la maintenance. Notre application intègre un composant 'Home' pour la page principale, qui inclut des éléments tels que la gestion de formulaires, les retours utilisateurs, et une liste de répertoires avec des options de filtrage. Cette liste mène également à une seconde page qui affiche les analyses effectuées par le back-end sous forme de graphiques à barres ou radars, offrant une visualisation claire des données traitées.
Et pour ce qui est de l'affichage des graphes nous avons opté une barre ou un axe par language
Tailwind CSS est également utilisé pour améliorer l'esthétique du front-end.

2) Gestion des données (JPA H2)

Dans votre application, la gestion des données est optimisée grâce à l'utilisation de **JPA** et **H2**. Les **Data Transfer Objects (DTO**) récupèrent les informations directement de la base de données. Cela rend le processus plus efficace, car seules les données nécessaires sont envoyées au front-end. Ainsi, lors de l'affichage des dépôts Git ou des contributions par tags, le front-end reçoit exactement ce dont il a besoin, permettant une manipulation efficace et structurée des données dans votre application.

Gestion des Données (JPA et H2)

1) Modélisation de la base de données

Dans notre modèle de données, chaque entité représente une table dans notre base de données, et les relations entre ces entités permettent de représenter les contributions des utilisateurs aux dépôts Git. L'entité **Contribution** est au cœur de notre modèle, représentant chaque contribution individuelle. Elle est associée à deux autres entités essentielles : **Contributor** et **Tag**. La relation **ManyToOne** entre **Contribution** et **Contributor** signifie qu'une contribution est effectuée par un seul **contributeur**, tandis que la relation **ManyToOne** avec **Tag** catégorise chaque **contribution** avec un **tag** spécifique.

De plus, l'entité Contributor représente les utilisateurs de l'application et est liée à d'autres entités par des relations **ManyToMany** et **OneToMany**. Les relations **ManyToMany** avec Repository permettent de suivre la collaboration entre les contributeurs et les dépôts, tandis que la relation **OneToMany** avec **Contribution** permet de garder une trace de toutes les **contributions** d'un **contributeur**.

D'autre part, l'entité **Repository** représente les dépôts Git et est associée à Tag et **Contributor**. La relation **OneToMany** avec Tag indique que chaque dépôt peut avoir plusieurs tags pour catégoriser les contributions. Enfin, la relation **ManyToMany** avec **Contributor** permet de savoir quels contributeurs sont associés à chaque dépôt.

Enfin, l'entité **Tag** représente les tags attribués aux contributions et est liée à Repository et Contribution. La relation **ManyToOne** avec **Repository** signifie qu'un **tag** est associé à un dépôt particulier, tandis que la relation OneToMany avec Contribution indique que plusieurs **contributions** peuvent être associées à un même **tag**.

Dans l'ensemble, ce schéma entité-association forme la base de notre structure de données, permettant des requêtes complexes et des analyses approfondies des contributions par langage, utilisateur et dépôt.

Tests et Validation

Pour garantir la qualité et la fiabilité de notre application, nous avons mis en place une stratégie de test complète. Nous avons effectué des tests pour un large éventail de fonctions, y compris les fonctions publiques et privées de l'application. Notre approche de test inclut l'utilisation de **l'invocation de méthodes (invoke**) pour accéder aux fonctions privées lors des tests unitaires.

Utilisation de l'invocation de méthodes

Pour tester les fonctions privées de notre application, nous avons employé la technique de l'invocation de méthodes. Cela nous a donné la possibilité de vérifier le comportement interne de ces fonctions, garantissant ainsi leur bon fonctionnement même dans des scénarios complexes.

Actions en cas d'échec

Lorsque des cas de test ont échoué, nous avons immédiatement entrepris des actions correctives. Nous avons suivi une approche itérative pour résoudre les problèmes, en les documentant soigneusement et en appliquant des correctifs appropriés. Cette démarche nous a permis d'améliorer constamment la qualité de l'application au fil du développement.

Difficultés rencontrées

Pendant la phase de développement de notre projet, nous avons été confrontés à plusieurs défis techniques

L'intégration de la librairie **JGit** s'est révélée être une étape exigeante de notre projet. Cette librairie, bien puissante pour interagir avec des dépôts Git, présente une courbe d'apprentissage abrupte. Comprendre son architecture interne et compliqué d'utilisation et pas très optimiser .

La gestion de **la base de données** a été une autre difficulté à laquelle nous avons dû faire face. Nous nous sommes interrogés sur la manière optimale de stocker des données complexes telles que la carte des langages dans la base de données. Nous avons dû concevoir une approche qui garantit des performances optimales même lorsque la base de données était soumise à une grande quantité de données.

Assurer une interaction rapide entre le back-end et **l'interface utilisateur** a été une préoccupation constante. Nous avons dû nous assurer que les données étaient transmises de manière précise et qu'elles étaient correctement affichées dans l'interface utilisateur.

Les erreurs et **les exceptions** sont inévitables dans le développement logiciel. Nous avons été confrontés à des situations où des erreurs se sont produites, nécessitant une gestion appropriée. La création d'une stratégie de gestion des erreurs robuste a été essentielle pour garantir la stabilité de l'application.

Nous avons pas réussi à régler certains warnings tels que
[WARNING] **No processor claimed** any of these annotations:
et [WARNING] **Supported source version 'RELEASE_20'** from annotation processor 'io.micronaut.annotation.processing.PackageConfigurationInjectProcessor' less than -source '21' lors du package du projet

Choix de développement

Durant la conception du projet nous avons fait plusieur choix d Implémentation tels que pour les calculs des contributions nous comparons uniquement avec le tag précédent et qui nous paraissait un peu plus intuitif pour le moyenne des contributions .

Pour le calcul d'une moyenne par tag nous avons fait les calculs dans le front évité de faire plein de appel api

Conclusion

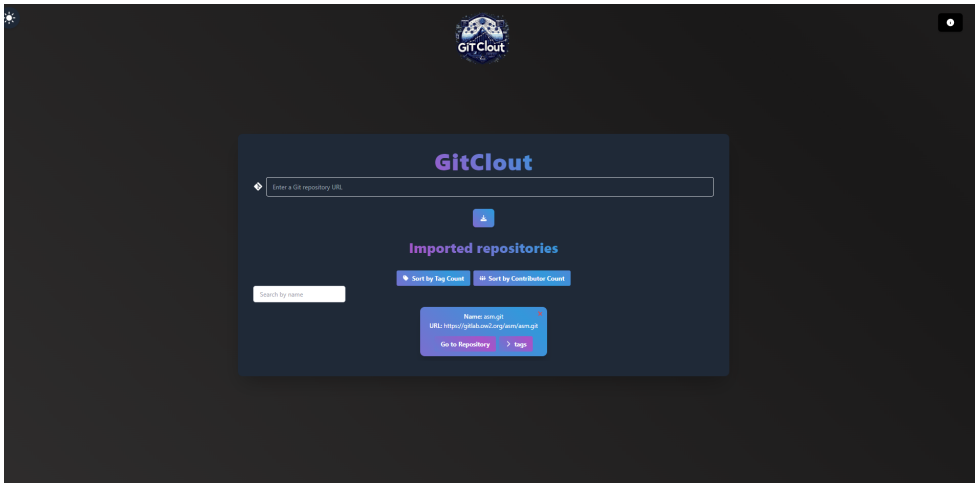
Le projet que nous avons mené dans le cadre de notre cursus en M1 Informatique a été une bonne expérience riche en enseignements. Notre objectif principal était de développer une application Web en utilisant diverses technologies modernes. Nous sommes fiers du travail accompli et des connaissances acquises au cours de ce projet.
Au fil de ce projet, nous avons eu l'opportunité d'explorer en détail des technologies telles que Micronaut, Angular, JPA (Jakarta Persistence API), H2 et Tailwind CSS. Nous avons réussi à créer une application fonctionnelle qui met en œuvre ces technologies de manière cohérente.
L'architecture modulaire que nous avons adoptée nous a permis de structurer notre application de manière organisée, simplifiant ainsi sa maintenance. La gestion des données a été réalisée efficacement grâce à l'utilisation de Data Transfer Objects (DTO) et à une interaction fluide entre le back-end et le front-end.
Bien que nous ayons rencontré des défis techniques tout au long du projet, notamment en ce qui concerne l'intégration complexe de la librairie JGit, nous avons réussi à les surmonter avec persévérance et détermination.
En conclusion, ce projet a été une étape importante de notre formation en informatique. Nous sommes conscients qu'il y a encore beaucoup à apprendre et à améliorer, mais cette expérience nous a permis de progresser dans notre parcours académique et professionnel.

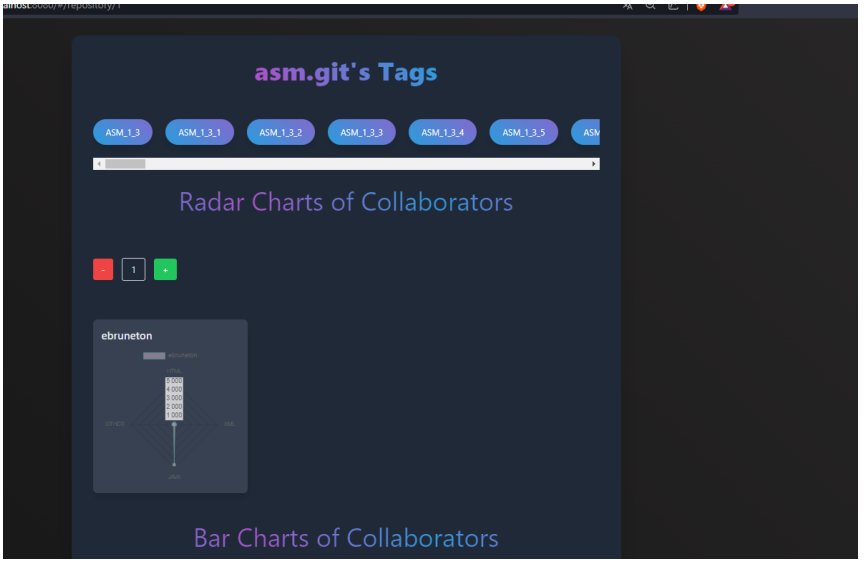
Références

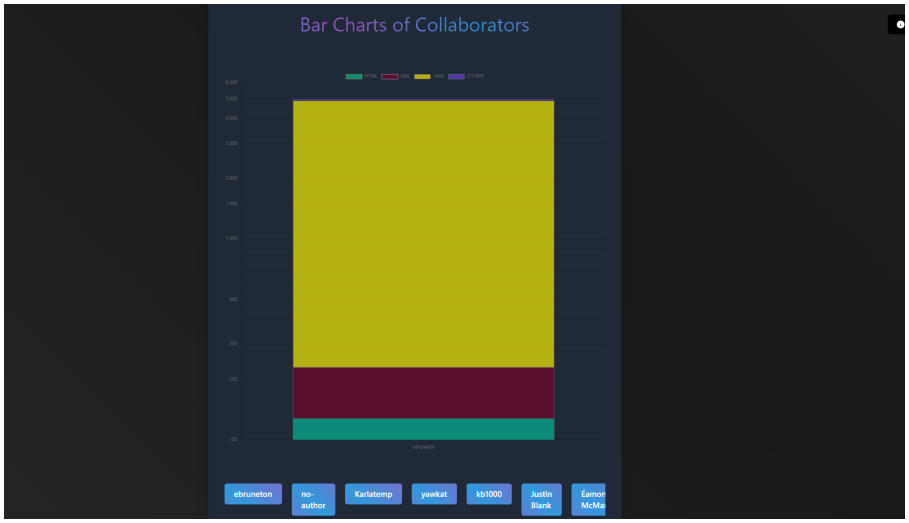
Pour la réalisation de ce projet, nous avons bénéficié de l'aide précieuse de diverses ressources en ligne, notamment des forums de développeurs et des documentations techniques:

https://micronaut-projects.github.io/micronaut-openapi/latest/guide/index.html
https://micronaut.io/launch/
https://github.com/centic9/jgit-cookbook
https://tw-elements.com/docs/standard/data/charts/
https://micronaut-projects.github.io/micronaut-data/latest/guide/
Access a database with JPA and Hibernate
https://mvnrepository.com/
Micronaut Framework
https://github.com/Vlt09/gitCloutTest/

Annexe


GitClout
GitClout
Enter a Git repository URL
Imported repositories
Sort by Tag Count
Sort by Contributor Count
Search by name
Name: asm.git
URL: https://gitlab.ow2.org/asm/asm.git
Go to Repository
tags


asm.git's Tags
ASM_1_3
ASM_1_3_1
ASM_1_3_2
ASM_1_3_3
ASM_1_3_4
ASM_1_3_5
Radar Charts of Collaborators
1
ebruneton
Bar Charts of Collaborators


Bar Charts of Collaborators
HTML
XML
JAVA
OTHER
ebruneton
no-author
Karlatemp
yawkat
kb1000
Justin Blank