Hakim AOUDIA
Nafis Basheer Ahamed

Semestre 1
Master 1 Informatique (2023 - 2024)

Université Gustave Eiffel

Projet GitClout - Projet du premier semestre

Documentation technique

Introduction

Ce type de projet est assez nouveau pour nous. C'est la première fois qu'on réalise un projet qui contient une backend avec un REST API, un frontend et une base de données. Le but de ce projet était de réaliser une application web pour pouvoir analyser la contribution de chaque personne pour chaque tag d'un repository.

Technologies demandés

Backend : Helidon 3 MP Reactive
Persistence : JPA
Base de données (version embedded) : HyperSQL
Frontend : Svelte
UI : Semantic

Le backend est ce qui nous a servi à produire/renvoyer un résultat lorsqu'on l'appelle depuis le frontend. Nous devions utiliser la version reactive de Helidon 3 MP mais nous avons eu des problèmes ne sachant pas correctement utiliser la réactivité.

Base de données

1 - Mise en place du schéma relationnel

Avant de commencer à coder des classes pour le projet, nous devions comprendre quel objet on nous demandait d'utiliser. Et pour cela nous avons décidé de partir sur la base de données suivante :

```
CREATE TABLE REPOSITORY
(
    repository_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    url VARCHAR(128) NOT NULL,
    date DATE DEFAULT CURRENT_DATE
);
```

```
CREATE TABLE TAG
(
    tag_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    repository_id INT,
    name VARCHAR(128),
    FOREIGN KEY (repository_id) REFERENCES REPOSITORY(repository_id)
);
```

```sql
CREATE TABLE CONTRIBUTOR
(
    contributor_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    tag_id INT,
    mail VARCHAR(128),
    name VARCHAR(128),

    codes INT DEFAULT 0,
    builds INT DEFAULT 0,
    configs INT DEFAULT 0,
    resources INT DEFAULT 0,
    docs INT DEFAULT 0,
    FOREIGN KEY (tag_id) REFERENCES TAG(tag_id)
);
```

Nous avons décidé de ne pas créer une 4 ème table Contribution car on a estimé que dans notre cas une seule table “Contributor” suffit à gérer les informations des contributeurs et aussi leurs contributions en 1 seul table.

JPA

Pour la création des entités en JPA il a fallu se familiariser avec le mapping. Car c’est en utilisant le mapping qu’on crée les liens entre les entités et la base de données.

Nous avons utilisé plusieurs annotations comme :

- @Entity, @Table,, @Column, @ManyToOne (Pour spécifier des clés étrangères), @JoinColumn.

- **@Id :** Pour spécifier que c’est une clé primaire
- **@NamedQueries :** pour stocker des requêtes SQL
- **@GeneratedValue :** Génère automatiquement les valeurs des id
- **@ManyToOne :** Pour spécifier des clés étrangères

Backend

Concernant le backend nous n’avons pas de partie réactive au vue de la lenteur d’analyse et d’affichage pour des gros repository. Au lieu de renvoyer des **List**<Tag> depuis notre backend par exemple, nous avons tenté d’utiliser **PublisherBuilder**<Tag>, **Multi**<Tag> et **ReactiveStreams**<Tag> mais sans réussite.

Frontend

Pour le frontend nous devions utiliser Svelte qui est un framework JavaScript. Pas de grande difficulté sur cette partie, seulement s’habituer à certaines notation du framework comme l’utilisation de **OnMount()**, **#await**, **$:**.

OnMount() : onMount permet d'exécuter une fonction lorsque le composant est chargé dans le DOM.

$: : Est une déclaration réactive de svelte permettant de relancer un bout de code lorsqu' une condition ou variable est mise à jour/modifiée.

#await : On va pouvoir bloquer et afficher un certain message tant qu'une promesse d'un async n'est pas arrivée. Quand le promise est reçu alors on affiche la partie de code dans **{then ..}** et si on reçoit rien on affiche un message dans **{catch ..}.**

Bibliothèque graphique

1 - UI

Pour notre groupe nous devions utiliser semantic pour améliorer la partie CSS. Or comme le projet n'est pas fonctionnel à 100% nous avons décidé de ne pas s'attarder sur une partie non essentielle à l'analyse des tags.

2 - Librairie

Pour la partie graphique nous avons utilisé ChartJs. On aurait pu utiliser D3js qui est tout aussi connu mais on a trouvé plus simple l'utilisation de ChartJs avec svelte.

"Vous pouvez de plus, utiliser une librairie spéciale pour la gestion de l'affichage du tableau pourvu qu'elle soit adaptée à votre framework (pas de react-grid si vous devez utiliser svelte)."

On a importer la librairie avec cette ligne de code en svelte :

import Chart from 'chart.js/auto'

Pour différencier les types de graphes on modifie la variable **type** dans nos variables **configs**.

Comme ceci :

type: 'bar'
type: 'radar'