Documentation de DIVA'NUM

RANDRIANARIMANANA Mamisoa Nandrianina - Zhu Chenrui - ZHANG Xianxiang

1 Installation et configuration de l'environnement de développement et de mise en production

1.1 Développement

1.1.1 Développement du frontend

Nous avons utilisé **VScode** pour développer angular frontend.

Installer les dépendances spécifiées:

```
npm install
```

Pour lancer le frontend localement:

```
ng serve
```

1.1.2 Développement du backend

Nous avons utilisé **IntelliJ IDEA** pour développer springboot backend.

Pour lancer le backend localement, il faut accéder à application.java et l'exécuter.

Dans le cas local, assurez-vous que le fichier **application.properties** obtient correctement

- l'url de la base de données / nom de schéma
- le nom d'utilisateur et le mot de passe correspondants

```
spring.datasource.url= jdbc:mysql://localhost:3306/demo?useSSL=false
spring.datasource.username= root
spring.datasource.password= divanum2023
```

1.1.3 Développement de la base de données

Nous avons utilisé **MySQL Workbench** pour développer la base de données.

Lors de la première exécution, vous devez créer un schéma et le nommer (par exemple demo). Après avoir exécuté le backend, vous devez utiliser les commandes sql suivantes pour ajouter des pré-données.

```
use demo;
INSERT INTO roles(name) VALUES('ROLE_USER');
INSERT INTO roles(name) VALUES('ROLE_MODERATOR');
INSERT INTO roles(name) VALUES('ROLE_ADMIN');
```

Une fois ce qui précède terminé, il ne sera plus nécessaire d'effectuer d'autres opérations côté base de données.

1.2 Déploiement et mise en production

Nous avons utilisé Google Cloud Console pour déployer notre projet. Le montant offert par Google Cloud Console nous permet de faire fonctionner le serveur pendant 90 jours. Mais gardez à l'esprit que les étapes suivantes s'appliquent à tous les serveurs.

1.2.1 Création du serveur

Tout d'abord, vous devez créer une instance de machine virtuelle sous ubuntu ou debian. Et modifiez les paramètres de pare-feu de l'instance de machine virtuelle pour autoriser le trafic HTTP réseau et le trafic HTTPS.

Dans l'instance de machine virtuelle, utilisez la commande **git clone** pour récupérer les codes de projet frontend et backend de notre projet sur le serveur. De cette façon, le code du projet peut être rapidement mis à jour sur le serveur, et il est également pratique pour nous de le modifier ultérieurement.

Étant donné que le projet doit également utiliser une base de données, vous devez également créer une nouvelle instance de base de données dans le SQL de Google Cloud Console. Si vous voulez créer une base de données sur le serveur, vous pouvez suivre les commandes suivantes.

```
mysql -u root -p
```

Après connexion avec un bon mot de passe,

```
CREATE DATABASE IF NOT EXISTS database_name;
```

Une fois le serveur créé, assurez-vous que **localhost:8080** dans tous les codes du frontend et du backend doit être remplacé par l'adresse IP externe du serveur(Dans notre cas c'est 34.155.164.205).

Comme les étapes locales, assurez-vous que le fichier **application.properties** obtient correctement le bonne url de la base de données.

Une fois les étapes ci-dessus terminées, vous pouvez commencer à déployer le frontend et le backend.

1.2.2 Installation des packages

- NodeJs

  ```
  sudo apt install nodejs
  ```

- Angular

  ```
  npm install -g @angular/cli
  ```

- OpenJDK

  ```
  sudo apt install openjdk-19-jdk
  ```

- MySQL

  ```
  sudo apt install mysql-server mysql-client
  ```

- Nginx

  ```
  sudo apt install nginx
  ```

1.2.3 Deploiement du frontend

Nous devons d'abord utiliser

```
ng build
```

sur la ligne de commande du code frontal pour générer des fichiers statiques pour le déploiement.

Ensuite, nous devons modifier le fichier de configuration Nginx afin qu'il puisse lire nos fichiers statiques.

Ouvrez le fichier de configuration Nginx:

```
sudo vim ~/etc/nginx/sites-available/default
```

Modifiez le fichier de configuration de Nginx comme suit:

```
server{
    listen 80 default_server;
    listen [::]:80 default_server;

    root /home/zhangxianxiang99/ptrans/angular-frontend/dist/angular-fronted;

    index index.html index.htm index.nginx-debian.html;

    server_name _;

    location / {
        try_files $uri $uri/ /index,html =404;
        proxy_http_version 1.1;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Port $server_port;
    }
    location /api
    {
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' 'http://34.155.164.205/';
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
            add_header 'Access-Control-Allow-Headers'
                'DNT,User-Agent,X-Requested-With,If-Modified-Since,
                Cache-Control,Content-Type,Range';
            add_header 'Access-Control-Max-Age' 3600;
            add_header 'Access-Control-Allow-Credentials' 'true';
            add_header 'Content-Length' 0;
            add_header 'Content-Type' 'text/plain charset=UTF-8';
            return 204;
        }

        # Handle other requests
        add_header 'Access-Control-Allow-Origin' 'http://34.155.164.205/';
        add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
        add_header 'Access-Control-Allow-Headers'
            'DNT,User-Agent,X-Requested-With,If-Modified-Since,
            Cache-Control,Content-Type,Range';
        add_header 'Access-Control-Expose-Headers'
            'Content-Length,Content-Range';
        add_header 'Access-Control-Max-Age' 3600;
        add_header 'Access-Control-Allow-Credentials' 'true';
        proxy_pass    http://localhost:8080;
        proxy_set_header    Host                $host;
        proxy_set_header    X-Real-IP           $remote_addr;
        proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Host    $host;
        proxy_set_header    X-Forwarded-Server $host;
        proxy_set_header    X-Forwarded-Port    $server_port;
        proxy_set_header    X-Forwarded-Proto   $scheme;
    }
}
```

Dans le fichier de configuration, le chemin racine est le chemin absolu du fichier **index.html** généré après l'utilisation de **ng build**. **34.155.164.205** est l'adresse IP externe de l'instance de machine virtuelle que nous avons créée.

Démarrez ensuite le serveur Nginx:

```
sudo service nginx start
```

Le déploiement du front-end est terminé!

1.2.4 Deploiement du backend et Configuration de la base de donnée

Modifier le fichier **application.properties**:

```
spring.datasource.url= jdbc:mysql://34.163.200.201/demo?useSSL=false
spring.datasource.username= root
spring.datasource.password= divanum2023

spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto= update

spring.servlet.multipart.max-file-size=200MB
spring.servlet.multipart.max-request-size=200MB
```

34.163.200.201 est l'adresse IP externe de l'instance de base de données et **demo** est le nom de l'instance de base de données.

Exécutez la ligne de commande dans le répertoire du code backend. Utilisez **mvn package** pour empaqueter le code backend.

Exécutez le programme backend avec

```
cd springboot/target/
java -jar spring-boot-login-example-0.0.1-SNAPSHOT.jar
```

,**spring-boot-login-example-0.0.1-SNAPSHOT.jar** est un fichier généré après le package du projet backend, il créera une nouvelle table dans la base de données.

Ouvrir la base de données avec

```
sudo mysql -h 34.163.200.201 -u root -p
```

Passez au schéma "demo", et ajouter trois rôles:

```
use demo;
INSERT INTO roles(name) VALUES('ROLE_USER');
INSERT INTO roles(name) VALUES('ROLE_MODERATOR');
INSERT INTO roles(name) VALUES('ROLE_ADMIN');
```

Exécutez à nouveau le code backend et maintenez-le en cours d'exécution:

```
cd springboot/target/
java -jar spring-boot-login-example-0.0.1-SNAPSHOT.jar > log.file 2>&1 &
```

2 Description de l'architecture technique de DIVA'NUM

2.1 Couche de presentation (FrontEnd)

Cette couche est responsable de l'interface utilisateur de l'application et de la gestion des interactions avec l'utilisateur. Elle est constitué de la partie affichage où on sépare les différents entités en composant et les services.

Les composants (component) qui sont des entités regroupant du code en TypeScript, du HTML et du CSS pour fournir une fonctionnalité ou une partie d'interface utilisateur réutilisable. Elles permettent de diviser une application en éléments plus petits, plus faciles à gérer et à développer, ce qui permet de créer des applications plus modulaires et évolutives. On a mis dans différents components les différents fonctionnalités et entites du projet notamment la connexion, le questionnaire, la page d'accueil, la plateforme direct, etc.

Les services sont des classes qui fournissent une fonctionnalité spécifique à l'ensemble de l'application notamment pour le partager des données, les appels de données du Backend comme les appels d'API par exemple le service pour récuperer les fichiers d'un aidant, etc.

Dans notre projet, on aura les principaux composants:

- **board-admin:** qui correspond à la partie de l'évaluateur notamment à l'interface et les différents actions qu'il pourra faire, il sera relié à d'autres composants comme "caregiver-moderator"
- **board-user:** qui correspond à l'interface de l'aidant notamment à la page d'accueil, et l'ajout de fichier "file-upload"
- **caregiver-moderator:** correspond à la création d'un nouveau compte aidant relié avec le composant "register" ainsi que la liste des aidants existant avec leur informations.
- **components/file-upload:** correspond à l'ajout de fichier pour l'aidant
- **home:** correspond au point d'entrée de l'application Web avec les bouton connexion et plateforme direct
- **login:** correspond à la connexion et l'authentification
- **profile:** permet d'avoir les détails de l'utilisateur : identifiant, rôle, code..
- **quick-access:** correspond au plateforme directe d'ajout de fichier
- **register:** correspond à l'inscription d'un nouvel utilisateur
- **register_admin:** correspond à l'inscription d'un nouvel évaluateur
- **survey:** correspond au gestion des questionnaires
- **survey-creator:** correspond à la création de nouveau questionnaire

2.2 La couche applicative (Backend)

Cette couche est responsable de la logique métier de l'application , de la gestion des requêtes entrantes provenant de la couche de présentation et de l'intéraction avec la base de donnée.

L'architecture général du backend de notre application est structuré sous forme de package selon leur role:

- **Le package security** contient les classes de configuration pour le projet, telles que la configuration de sécurité, la authentification et la configuration Web.
- **Le package controllers** contient les contrôleurs de l'API, qui sont responsables de l'écoute des requêtes HTTP et de la gestion des réponses HTTP.
  - AuthController s'agit d'un contrôleur Spring MVC chargé de gérer les requêtes HTTP pour toutes les opérations liées à l'enregistrement et à la connexion des utilisateurs et des administrateurs
  - FileController permet de gérer les chargements et téléchargements de fichiers
  - InfoController inclue les informations utilisateur et les codes
  - SurveyController implique le stockage des questionnaires et des réponses ainsi que la récupération des questionnaires
- **Le package repository** contient les objets de transfert de données, qui sont importe dans controllers et sont utilisés pour transférer des données entre les couches de l'application. S'il implique l'acquisition de certaines colonnes de données dans la base de données, il contiendra des requêtes SQL associées, sinon la plupart des appels de données sont directement appelés entités de données via JpaRepositoy (qui peut être compris comme une ligne de données dans la base de données).
- **Le package exception** contient les exceptions personnalisées, qui sont levées lorsque des erreurs se produisent dans l'application.

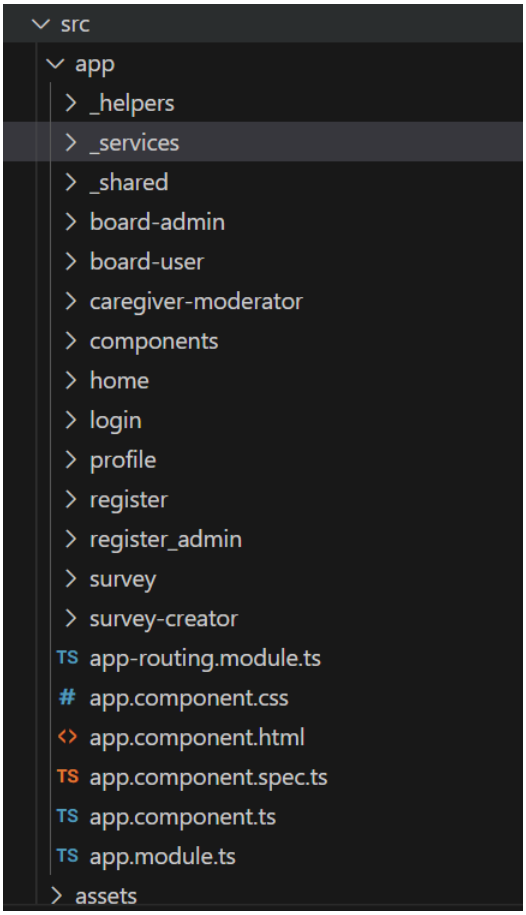

Figure 1: Architecture du frontend

- **Le package models** contient les entités de la base de données, qui sont utilisées pour représenter les données dans la base de données. Un model est une java classe qui est définie par nombreux de attributs et fonctions.
- **Le package service** contient les services métier, qui contiennent la logique métier de l'application et communiquent avec les repositories pour accéder aux données. Cela peut être compris comme la modification et l'obtention de données via Repository.
  - CodeService est responsable de la génération, de l'acquisition et de la modification du code
  - FileStorageService permet de récupérer les fichiers par utilisateur ou par code
  - SurveyService s'agit d'ajouter, de supprimer et de modifier des questionnaires
  - UserService concern des recherches d'informations sur l'utilisateur
- **Le package payload** définit les classes pour les objets Request et Response.

2.3 La couche de persistance des données

Cette couche est responsable de la gestion de l'interaction avec la base de données, où sont stockées les données de l'application. On a utilisé MySQL qui est une base de données relationnelle open source très populaire. Elle est largement utilisée pour stocker des données structurées telles que des informations d'utilisateur, des transactions et des rapports, ainsi que pour les applications web qui nécessitent des interactions en temps réel avec une base de données.

On va décrire via la figure 3 qui suit le modèle de base de donnée.

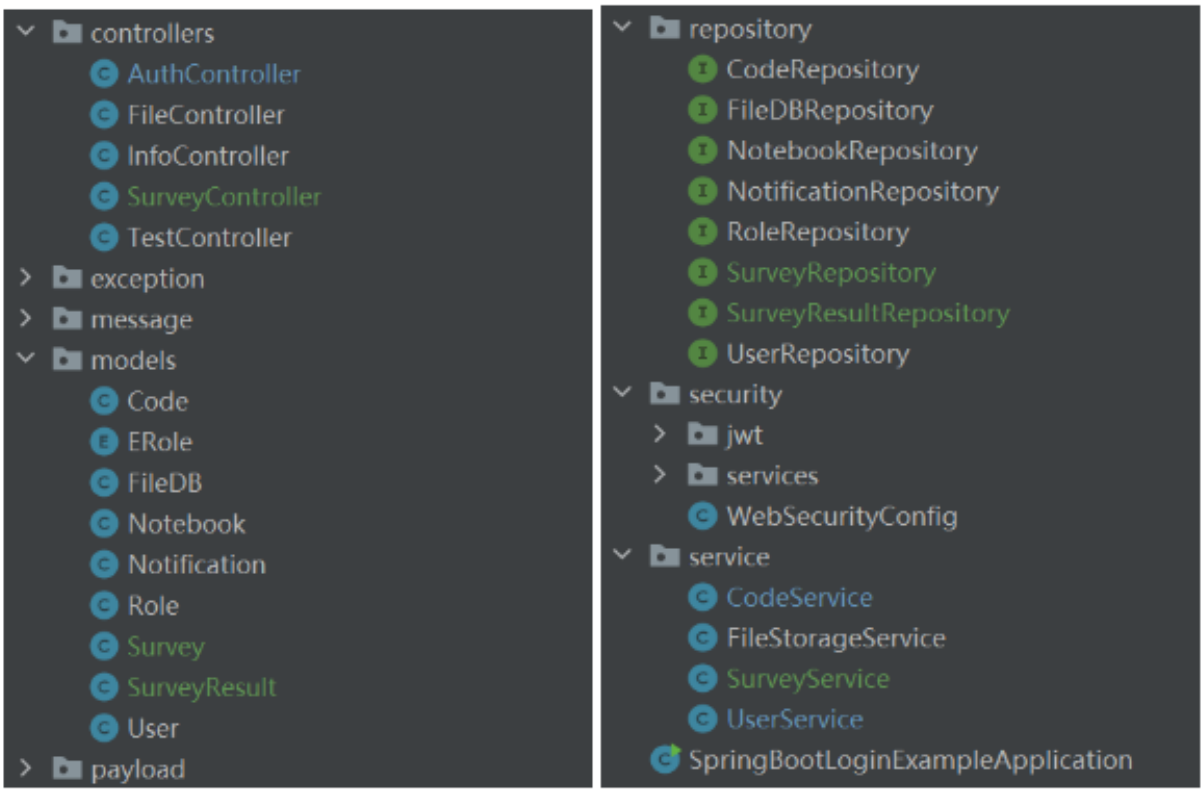

Figure 2: Architecture du backend

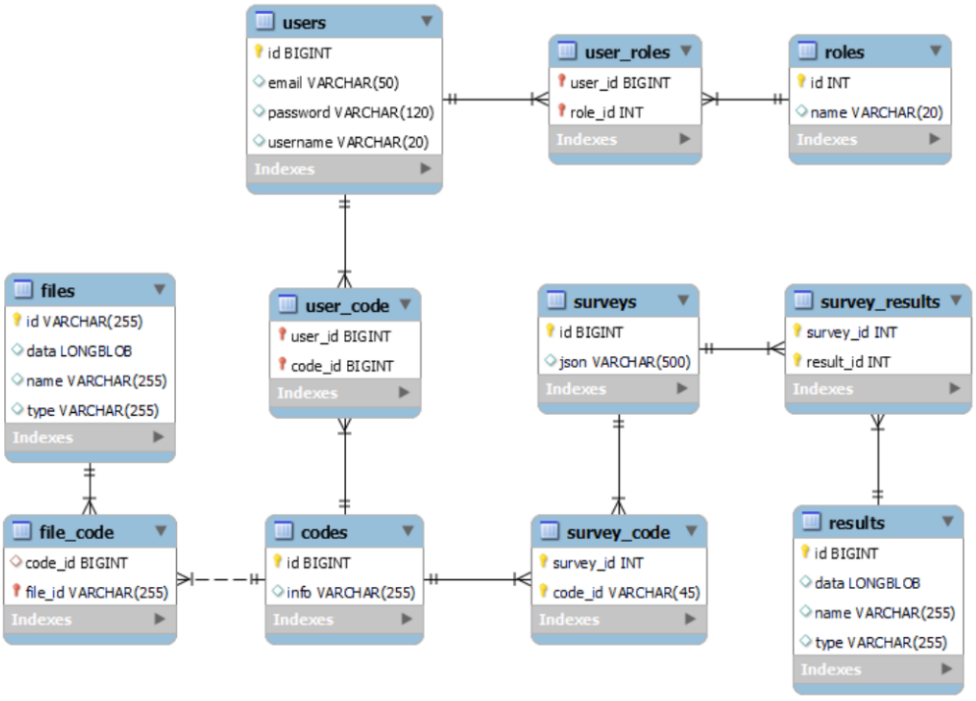

Figure 3: Modèle de base de donnée

Dans ce diagramme de relation, on peut voir que l'aidant est relié à un fichier ou un questionnaire via le code d'ajout et les fichiers sont reliés aux évaluateurs qui vont les recueillir.

2.4 API

Voici les API 1 qui peuvent être utilisées à ce stade, y compris les opérations de connexion et d'enregistrement des utilisateurs, les opérations sur les comptes d'utilisateurs et les opérations sur les transferts de fichiers.

Table 1: API

Méthode	**URL**	**Actions**
POST	/api/auth/signup	créer un nouveau compte pour l'utilisateur
POST	/api/auth/signup_admin	créer un nouveau compte pour l'évaluateur
POST	/api/auth/signin	se connecter à un compte
POST	/api/auth/signout	déconnecter le compte
GET	/api/test/user	accéder au contenu de l'utilisateur
POST	/api/auth/code/upload	déposer un fichier par code
GET	/api/auth/user_id/files	obtenir la liste des fichiers (nom et URL) d'un utilisateur
GET	/api/auth/download/files	télécharger un zip de tous les fichiers
DELETE	/api/auth/files	supprimer tous les fichiers
DELETE	/api/auth/users/id	supprimer un utilisateur
DELETE	/api/auth/admin/id	supprimer un administrateur
GET	/api/info/user_id/codes	obtenir la liste des codes d'un utilisateur
GET	/api/info/users	obtenir la liste des utilisateurs
POST	/api/auth/survey/save	enregistrer un questionnaire
GET	/api/auth/survey/survey_id	obtenir un questionnaire
GET	/api/auth/survey/code/surveycode	obtenir un questionnaire par un code
POST	/api/auth/survey/results	enregistrer les résultats du questionnaire
GET	/api/auth/survey/survey_result	télécharger un zip de tous les résultats du questionnaire
DELETE	/api/auth/survey/survey_result	supprimer les questionnaires