Projet BDR - Cuisine

Départements : TIC

Unité d'enseignement BDR

Auteurs : **Graf Calvin**
Lucas Lattion
Sottile Alan

Professeur : **Nastaran Fatemi**
Assistant : **Christopher Meier**

Classe : **BDR-C**

Date : **21.01.2024**

1 Introduction

Dans le cadre du cours, BDR nous avons entrepris le projet d'une gestion d'une base de données d'un frigo contenant des aliments qui nous permettent d'effectuer des recettes en fonctions des goûts et préférences des utilisateurs.

2 Objectif

L'objectif de ce projet est d'entreprendre la réalisation complète d'une application de base de données. Elle sera réalisée en 5 phases. Après la réalisation du cahier des charges, nous allons devoir réaliser la modélisation conceptuelle qui consiste à faire un schéma conceptuel de la base de données au format UML. Ensuite, nous allons transformer ce schéma en un schéma relationnel. A ce moment, nous allons effectuer la création de la base de données donc tout ce qui est table, contrainte d'intégrités référentielles avec l'écriture des requêtes, la création des vues pour notre application. Pour terminer, il faudra réaliser une application qui permettra de manipuler la base de données.

3 Projet

Comme projet, nous avons décidé de faire une application qui présentera à l'utilisateur un grand choix de recettes de cuisine en fonction des aliments à sa disposition ainsi que d'éventuelles allergies ou intolérances alimentaires. Chaque plat aura une liste d'ingrédients requis, ainsi qu'une marche à suivre afin de le réaliser. Les ingrédients composant le plat auront chacun un apport énergétique en Kilocalories, ainsi que sa composition en protéines, glucides, lipides vitamines sel et sucre pour 100 grammes. Cela permettra à l'utilisateur d'être informé de l'apport d'un certain plat avant de le réaliser, afin de convenir à un potentiel régime ou besoin alimentaires requis. L'utilisateur aura aussi la possibilité de sauvegarder ou de cacher certains plats par rapport à ses goûts ou besoins. Le fait de sauvegarder un plat permettra à l'utilisateur de le retrouver plus facilement, même s'il lui manque des aliments requis.

4 Besoins en données

Les données suivantes seront incluses :

- un **aliment** a un nom, des quantités d'**éléments nutritifs**
- un **élément nutritif** a un nom
- un **allergène** a un nom
- un **utilisateur** a une adresse email, un nom et prénom, un mot de passe, liste de **plat** sauvegardés et cachés.
- Un **plat** a un nom, une marche à suivre et est composé d'une quantité en gramme d'aliments ainsi que des **allergènes**

5 Fonctionnalités

Cette application va contenir les fonctionnalités suivantes :

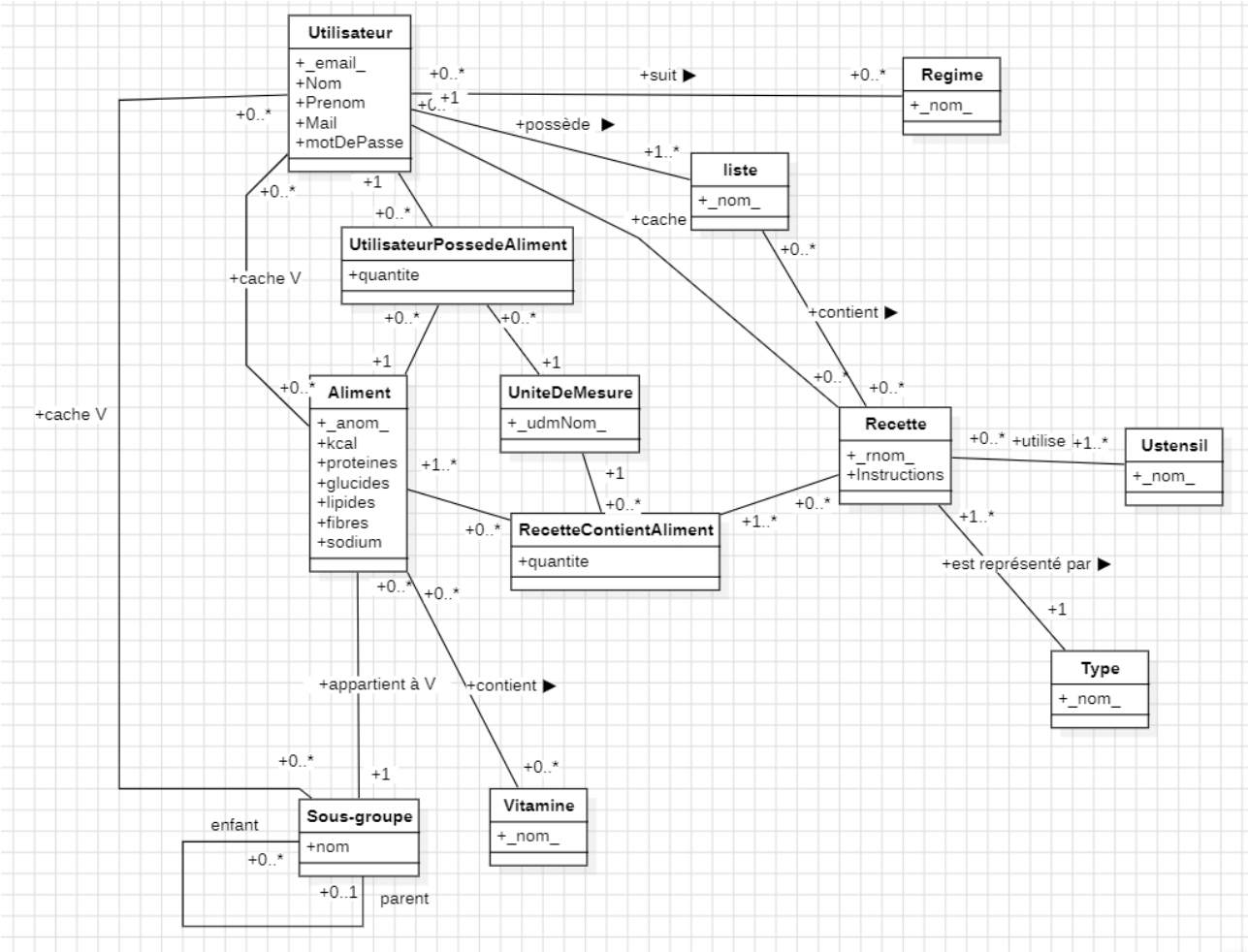
- Afficher ce que contient un aliment ou un plat
- Afficher un/des plats selon des filtres de recherches tels que le taux d'un certain aliment, allergènes, ou encore taux en certains éléments nutritifs
- Afficher des plats/ingrédients convenant à un certain régime
- Informer quels aliments manque à la conception d'un plat.
- Permettre la création d'un compte utilisateur
- Permettre à un utilisateur d'ajouter un plat en favoris
- Permettre à un utilisateur d'indiquer ses allergènes
- Permettre à un utilisateur de cacher un plat

6 Schéma conceptuel

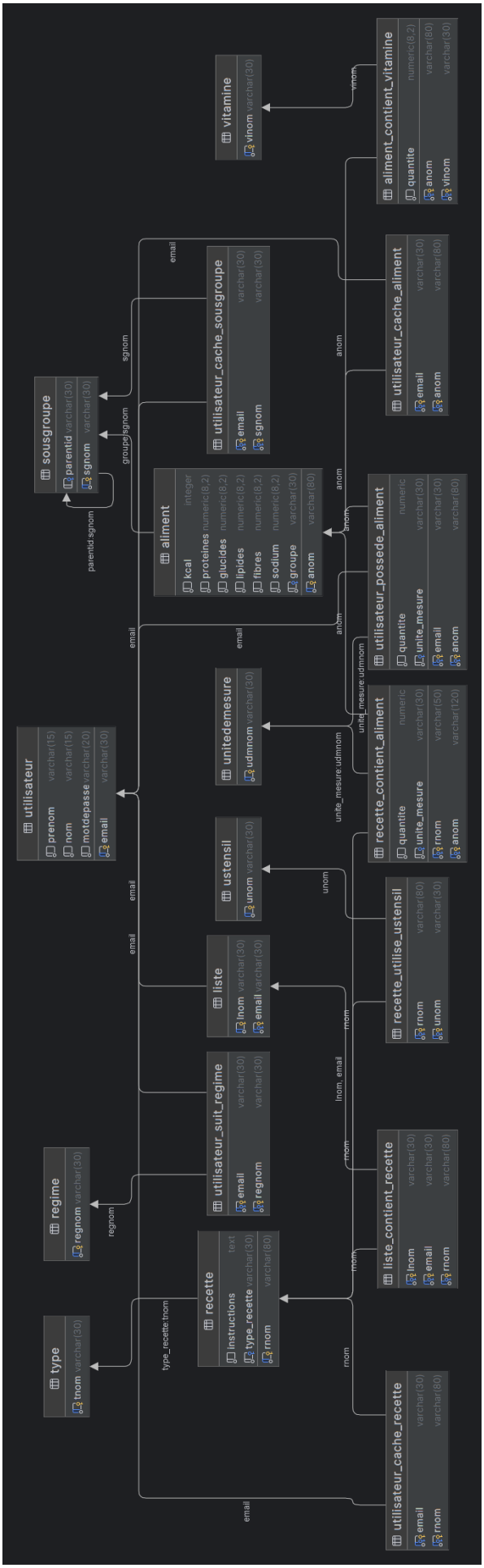
Nous avons effectué plusieurs modifications conceptuelles au cours du projet. Dont certaines recommandées par l'assistant et le professeur. Par exemple, l'entité Sous-groupe est elle-même parent et enfant.

Le schéma conceptuel contient comme entités suivantes :

- Utilisateur
- Type
- Aliment
- Sous-Groupe
- Recette
- Liste
- RecetteContientAliment
- UtilisateurPossedeAliment
- Regime
- Ustensile
- UniteDeMesure
- Vitamine



7 Schéma relationnel



8 Les vues

Nous avons créé plusieurs vues :

8.1 La vue calories_recette

La vue calories_recette permet d'estimer les calories que contient une recette.
Elle vas multiplier la quantité par les calories de chaque aliments pour ensuite additionner au total de calorie de la recette.

8.2 Les vues pour les groupes d'aliments

Chaque aliment appartient à un groupe d'aliment. Afin d'utiliser au mieux les groupes, voici les vues existantes :

- top_groupe_type: Contient la liste des groupes principaux/racine.
- sousgroupe_viande: Contient la liste des groupes de type viande.
- sousgroupe_Légumes: Contient la liste des groupes de type légume.
- sousgroupe_Fruits: Contient la liste des groupes de type fruit.
- sousgroupe_Poissons: Contient la liste des groupes de type poisson.
- sousgroupe_Laitiers: Contient la liste des groupes de type produits laitiers.
- sousgroupe_animal: Contient la liste des groupes de type animal (utilisé par opposition cas 'végétarien').
- sousgroupe_provenance_animal: Contient la liste des groupes de type provenance (utilisé par opposition cas 'végétalien').

8.3 La vue recettes_non_vegetarien et recettes_vegetarien

Pour trouver la vue des recettes non végétariennes, on récupère toutes les recettes qui sont dans la vue sous-groupe animal (donc les sous-groupes comme viandes et poissons).
Puis, par exclusion de toutes les recettes qui sont dans la vue recettes_non_vegetarien, on trouve la vue des recettes végétariennes.

8.4 La vue recette_avec_gluten et recette_sans_gluten

Pour trouver la vue des recettes avec gluten, on récupère toutes les recettes qui sont dans le sous-groupe « Céréales et légumineuses ».
Puis, par exclusion de toutes les recettes qui sont dans la vue recettes_avec_gluten, on trouve la vue des recettes sans gluten.

9 Les déclencheurs automatiques – triggers

9.1 Cacher un aliment et les recettes le contenant

Lorsqu'un utilisateur cache un aliment dans la table utilisateur_cache_aliment on souhaite que les recettes contenant l'aliment soient aussi cachées et donc mettre à jour la table utilisateur_cache_recette. Le même mécanisme est existant pour les sous-groupe.

9.2 Le triger cache_aliment_trigger

Le trigger cache_aliment_trigger appel la fonction after_update_utilisateur_cache_aliment() après chaque mise à jour de la table utilisateur_cache_aliment.

9.3 La fonction after_udpate_utilisateur_cache_aliment

La fonction after_udpate_utilisateur_cache_aliment mets à jour la table utilisateur_cache_recette en fonction de la table rectte_contien_aliment.

10 API et les routes

10.1 La sécurité

Nous sommes conscients que la gestion des jetons, des autorisations et des mots de passe manque de sécurité. Si le temps le permet, nous ajouterons une meilleure sécurité.

10.2 Routes d'authentification

10.2.1 POST /login

La requête POST /login est utilisée pour s'authentifier auprès du serveur.

Elle fournit un jeton qui sera utilisé dans les futures requêtes.

Pour l'utiliser, vous devez connaître un nom d'utilisateur ou en créer un : POST /user.

Envoyez un JSON avec un exemple similaire de valeurs formulées :

```
{
   "email" : "test.user1@email.com",
   "password" : "newPassword2"
}
```

Le serveur renvoie un jeton utilisateur avec l'adresse électronique :

```
user=test.user1@email.com ; Path=/ ;
```

Le jeton admin suivant a tous les privilèges (certaines requêtes basées sur l'utilisateur peuvent ne pas fonctionner) :

```
user=admin ; Path=/ ;
```

10.2.2 POST /logout

La requête POST /logout supprime le jeton utilisateur reçu lors de la demande de connexion.

10.3 Routes des utilisateurs

10.3.1 GET /profil

La requête GET /profile demande au serveur le profil de l'utilisateur actuellement connecté.

Le serveur répond avec un JSON contenant les informations du profil :

```
{
   "firstName" : "Test",
   "lastName" : "User1",
   "email" : "test.user1@email.com",
   "password" : "newPassword2"
}
```

10.3.2 POST /user

La requête POST /user crée un utilisateur.

Elle envoie un JSON avec un exemple similaire de valeurs formatées :

```
{
   "firstName" : "Test",
   "lastName" : "User1",
   "email" : "test.user1@email.com",
   "password" : "newPassword2"
```

}
Le serveur répond avec le code de statut "201 : Created" lorsque la création est réussie. Si l'adresse électronique de l'utilisateur existe déjà, le serveur renvoie un code d'état "409 : Conflict" : "409 : Conflict".

Une fois qu'un utilisateur a été créé, vous pouvez vous connecter à l'aide d'une requête POST /login.

10.3.3 PUT /user/{email}

La requête PUT /user/{email} met à jour l'utilisateur avec l'adresse électronique : {email}.

Envoyez la requête et un JSON avec un exemple similaire de valeurs formatées :

PUT /user/test.user1@email.com
{
  "firstName" : "Test",
  "lastName" : "NewLastName",
  "email" : "test.user1@email.com",
  "password" : "newPassword3"
}
Le serveur répond avec le code d'état "204 : No Content" lorsque la mise à jour est réussie.

10.3.4 DELETE /user/{email}

La requête DELETE /user/{email} supprime l'utilisateur dont l'adresse électronique est {email} : {email}.

Envoyer un JSON avec un exemple similaire de valeurs formatées :

DELETE /user/test.user1@email.com
Le serveur répond avec le code d'état "204 : No Content" lorsque la suppression est réussie.

10.4 CRUD pour une table de la base de données

Pour la table suivante de la base de données postgre, des requêtes CRUD (Create, Read, Update, Delete) sont disponibles comme décrit.

La requête doit être effectuée avec un utilisateur connecté POST /login.

Liste du nom de la table :

- Aliment
- Aliment_contient_vitamine
- Liste
- Liste_contient_recette

- Recette
- Recette_contient_aliment
- Recette_utilise_ustensil
- Regime
- Sousgroupe
- Type
- Ustensil
- Utilisateur_cache_aliment
- Utilisateur_cache_recette
- Utilisateur_cache_sousgroupe
- Utilisateur_possède_aliment
- Utilisateur_suit_regime
- Vitamine

10.4.1 GET /nom-de-la-table

La requête GET /nom-table (en minuscules) demande au serveur de récupérer toutes les entrées de la table.

Le serveur répond avec un JSON contenant les informations de la table (exemple avec aliment) :


```
[
   {
      "anom" : "Ananas",
      "kcal" : 50,
      "proteines" : 0.5,
      "glucides" : 13.0,
      "lipides" : 0.2,
      "fibres" : 1.4,
      "sodium" : 1.0,
      "groupe" : "Fruits Tropicaux"
   },
   {
      "anom" : "Banane",
      "kcal" : 105,
      "proteines" : 1.3,
      "glucides" : 27.0,
      "lipides" : 0.3,
      "fibres" : 3.1,
      "sodium" : 1.0,
      "groupe" : "Fruits"
   }
]
```


10.4.2 GET /nom-de-la-table/limite

La requête GET /nom-table/limite (en minuscules) demande au serveur d'obtenir des entrées pour la table.

{
"limit" : "0", // limite le nombre d'éléments renvoyés, par défaut 0 signifie tous les éléments.
"offset" : "0", // sauter un nombre x d'entrées, par défaut 0 signifie qu'aucun élément n'est sauté.
anom" : "0", // limite le nombre d'éléments renvoyés, par défaut 0 signifie tous les éléments : "TestAliment", // filtre spécifique à l'aliment
"groupe" : "Fruits Tropicaux" // filtre spécifique à l'aliment
}
Le serveur répond avec un JSON contenant les informations de la table (exemple avec aliment) :


```
[
  {
    "anom" : "TestAliment",
    "kcal" : 50,
    "proteines" : 0.5,
    "glucides" : 13.0,
    "lipides" : 0.2,
    "fibres" : 1.4,
    "sodium" : 1.0,
    "groupe" : "Fruits Tropicaux"
  }
]
```


10.4.3 POST /nom-de-la-table

La requête POST /nom-table crée une entrée.

Envoyer un JSON avec les valeurs de la table, exemple avec des valeurs formatées :


```
  {
    "anom" : "TestAliment",
    "kcal" : 50,
    "proteines" : 0.5,
    "glucides" : 13.0,
    "lipides" : 0.2,
    "fibres" : 1.4,
    "sodium" : 1.0,
    "groupe" : "Fruits Tropicaux"
  }
```


Le serveur répond avec le code de statut "201 : Created" lorsque la création est réussie. Si l'entrée existe déjà, le serveur renvoie le code d'état "409 : Conflict" : "409 : Conflict".

10.4.4 PUT /nom-table/{identifiant}

La requête PUT /table-name/{email} met à jour l'entrée avec l'identifiant : {identifiant}.

Envoyez la requête et un JSON contenant un exemple similaire de valeurs formatées :

PUT /aliment/TestAliment1

```
{
    "anom" : "TestAlimentUpdated",
    "kcal" : 50,
    "proteines" : 999,
    "glucides" : 13.0,
    "lipides" : 0.2,
    "fibres" : 1.4,
    "sodium" : 1.0,
    "groupe" : "Fruits Tropicaux"
}
```

Le serveur répond avec le code de statut "204 : No Content" lorsque la mise à jour est réussie.

10.4.5 DELETE /nom-table/{identifiant}

La requête DELETE /nom-table/{identifiant} supprime l'entrée avec l'identifiant de la table : {identifiant}.

Envoyer la requête de suppression comme par exemple :

```
DELETE /aliment/TestAliment1
```

Le serveur répond avec le code d'état "204 : No Content" lorsque la suppression est réussie.

11 Conclusion

Pour conclure, il a été intéressant d'effectuer un projet complet utilisant une base de données. On réalise l'importance de la conception du schéma de départ et la facilité ou non que cela peut apporter pour la création d'une API.

Date : 21.01.2024

Noms des étudiants : *Graf Calvin*
Lucas Lattion
Sottile Alan