20/05/2019

Basile Botebol, Arzur Catel Torres, Tommy Gerardi, Simon Mirkovitch, Pierrick Müller, Isaïa Spinelli

HEIG-VD2019

Gestionnaire de Budget Personnel

Rapport de projet PRO 2019

Table des matières

[1 Introduction 2](#_Toc8386785)

[2 Objectifs du projet 2](#_Toc8386786)

[2.1 Acquérir des nouvelles connaissances techniques / scientifiques 2](#_Toc8386787)

[2.2 Travail de groupe 2](#_Toc8386788)

[2.3 Mettre en pratique de notions acquises dans les cours précède 2](#_Toc8386789)

[2.4 Livrer un projet 2](#_Toc8386790)

[3 Concepts du projets 2](#_Toc8386791)

[4 Développent 2](#_Toc8386792)

[4.1 Langage 3](#_Toc8386793)

[4.2 Interface graphique 3](#_Toc8386794)

[4.3 Base de données 3](#_Toc8386795)

[4.4 Système de gestion de version 3](#_Toc8386796)

[5 Difficultés rencontrées 3](#_Toc8386797)

[5.1 Repo Git 3](#_Toc8386798)

[5.2 Environnement stable 3](#_Toc8386799)

[5.3 Familiarisation des nouvelles technologies 3](#_Toc8386800)

[5.4 Communication 3](#_Toc8386801)

[6 Tests 3](#_Toc8386802)

[6.1 Tests effectues 3](#_Toc8386803)

[6.2 Bugs restants 3](#_Toc8386804)

[7 Améliorations 3](#_Toc8386805)

[8 Remerciements 3](#_Toc8386806)

[9 Conclusion 3](#_Toc8386807)

[9.1 Situation finale du projet 3](#_Toc8386808)

[9.2 Fonctionnement du groupe 3](#_Toc8386809)

[9.3 Avis personnels 3](#_Toc8386810)

[10 Bibliographie 3](#_Toc8386811)

[11 Tables d’illustrations 3](#_Toc8386812)

[12 Annexes 4](#_Toc8386813)

# Introduction

Dans le cadre du cours PRO du 4eme semestres à la HEIG-VD, un projet nous a été confié afin d’apprendre différentes choses, dont le mise en pratique du travail en groupe.

Le projet que nous avons choisi, parmi ceux proposé, consiste à mettre en œuvre une application Web permettant de gérer un budget personnel.

# Objectifs du projet

Le cours PRO permet d’atteindre plusieurs objectifs :

## Acquérir des nouvelles connaissances techniques / scientifiques

## Travail de groupe

Prendre conscience des difficultés liées au travail en groupe (organisation, planification, communication)

## Mettre en pratique des notions acquises dans les cours précède

## Livrer un projet

# Concepts du projets

Une application qui permet de gérer un budget personnel. L'application permet de saisir les dépenses ainsi que de planifier le budget. Avec la saisie des dépenses l'application permet de saisir dépenses quotidiennes (supermarché, essence, ...) et de les catégoriser. L'application offre des catégories prédéfinies, mais l'utilisateur peut aussi définir ses propres catégories. L'utilisateur peut aussi saisir les revenus et l'application lui montre à tout instant son cash flow. À la fin du mois l'application montre à l'utilisateur les dépenses dans chaque catégorie et s'il est dans le rouge ou s'il reste quelque chose qu'il peut mettre de côté.

L'application permet aussi de planifier le budget. En utilisant les dépenses des mois passés l'application fait une projection des dépenses. L'utilisateur peut définir des quotas maximaux pour certaines catégories. Quand l'utilisateur saisit une dépense, l'application lui montre combien il reste de quota ou donne un avertissement si le quota est dépassé.

# Développement

## Langage

## Interface graphique

## Base de données

Afin de stocker les données et d’y avoir accès rapidement de manière simple et efficace, nous avons utilisé une base de données. Comme recommandé, nous avons utilisé PostGreSQL. PostGreSQL possède une interface graphique (pgAdmin) qui permet (après avoir appris à l’utiliser) de facilement modifier/faire des ajouts dans une base de données.

### Schéma UML

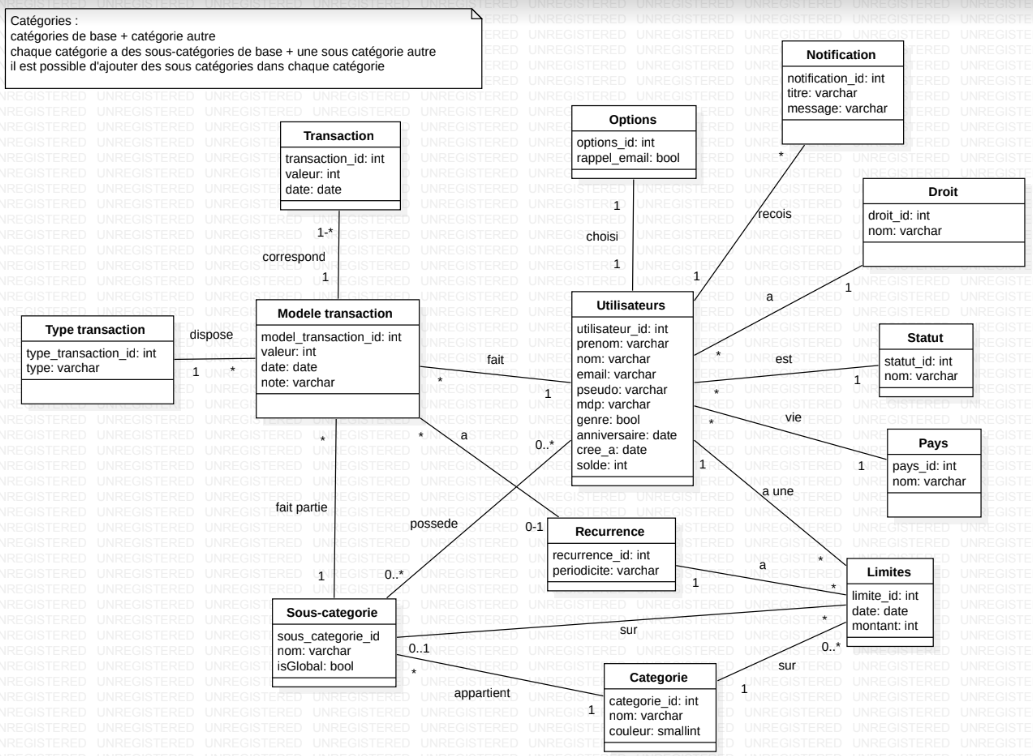


Figure 1 : Schéma UML de la base de données

### Résumé des tables

Transaction :

Stocke toutes les transactions effectuées par les utilisateurs, elles sont forcément reliées à un modèle de transaction. C’est un peu le “log” de toutes les transactions qui ont eu lieu sur le site, elle peut permettre de faires des statistiques et observer le comportement général des inscrits.

Modèle transaction :

Contient les modèles sur lesquels sont basées une ou plusieurs transactions. C’est ici qu’on saura si une transaction est récurrente ou encore si c’est une dépense ou un revenu. C’est une classe assez centrale car elle fait le lien entre un grand nombre de tables et contient donc toutes les informations pour faire les liens entre tout ce qui se passe lors d’une transaction.

Type transaction :

Permet de différencier les dépenses et les revenus, nous avons créé cette classe en pensant qu’il pourrait être possible d’avoir d’autres types de transaction dans le futur. Elle pourrait permettre par exemple d’ajouter “emprunt” par exemple, ainsi on saurait que cet argent devra être rendu, ce qui pourrait permettre de le traiter différemment pour les statistiques par exemple.

Récurrence :

Cette table contient les différentes récurrences disponibles, il y en a pour l’instant 6, mais on pourrait y ajouter des nouvelles.

Catégorie :

Ici sont stockées les catégories “de base” du site, qu’on ne pourra pas modifier en tant qu’utilisateur. Afin d’en ajouter il faudrait l’insérer soit en étant admin (cette fonctionnalité n’existe cependant pas encore), soit en le faisant à la main dans pgAdmin avec un nouveau backup.

Sous-catégorie :

Contient les sous-catégories “de base” ainsi que les sous-catégories créées par les utilisateurs. L’attribut isGlobal permet de savoir si c’est une classe globale (de base) ou personnelle à un utilisateur.

Limites :

Contient les limites fixées par les utilisateurs ainsi que la période durant laquelle elle s’applique (mensuelle, annuelle,…).

Options :

Contient les options choisies pour chaque utilisateur.

Notification :

Contient les notifications des utilisateurs, chaque notification se compose d’un titre et d’un message et appartient à un utilisateur.

Droit :

Contient les différents droits existants. Actuellement il y en a 3 (admin, utilisateur et conseiller) mais ils ne sont pas utilisés.

Statut :

Contient les différents statuts qu’un utilisateur peut avoir (étudiant, employé et autre).

Pays :

Contient les différents pays existants.

Utilisateur :

Cette table contient tous les utilisateurs existants, c’est aussi une classe “centrale” de la base de données, beaucoup d’éléments en dépendent.

### Triggers :

transactionCreation :

Lorsqu’on crée un modèle transaction (en ajoutant une dépense ou un revenu), ce trigger va entraîner la création de la transaction qui en découle. Attention toutefois, la transaction n’est créée que si le modèle transaction a pour date le jour actuel (ce qui ne pose pas de problème comme les utilisateurs ne peuvent actuellement pas mettre en place de paiements autres qu’instantanés).

modifSoldeExpense :

Ce trigger entraîne la modification du solde lorsqu’une transaction de type “dépense” est créée.

modifSoldeIncome :

Ce trigger entraîne la modification du solde lorsqu’une transaction de type “revenu” est créée.

check\_limits :

Quand une transaction est créée, on va aller voir si l’utilisateur à qui elle est reliée possède des limites. Si c’est le cas, on regarde si cette/ces limite(s) est/sont pour la catégorie à laquelle la transaction est liée. Ensuite, pour chaque limite pertinente, on va regarder si le total dépensé dans cette même catégorie ne dépasse pas la limite désirée pour le temps choisi pour la limite. Si une limite est dépassée, on va créer une notification pour l’utilisateur afin de la notifier qu’il a dépassé une limite dans la catégorie en question.

### Procédures :

add\_sous\_cat\_perso(user\_id integer, sous\_cat\_id integer) :

Cette procédure est utilisée lors de la création d’une sous-catégorie personnelle par un utilisateur. Elle permet d’insérer dans la table sous\_catégories\_personnelles l’id du user et celui de la sous\_catégorie qui vient d’être créée. On l’appelle depuis le code java avec les deux paramètres requis pour l’insertion dans la table.

check\_recurrences(user\_id integer) :

Cette procédure est appelée à chaque connexion d’un utilisateur. Elle est appelée dans le code java avec l’id de l’utilisateur qui se connecte comme paramètre. Elle permet de mettre à jour les transactions récurrentes de l’utilisateur en question.

## Système de gestion de version

Git blablabla

## Déploiement

# Difficultés rencontrées

## Repo Git

## Environnement stable

## Familiarisation des nouvelles technologies

## Communication

# Tests

## Tests effectues

### Test unitaire

Afin de vérifier le bon fonctionnement de chaque partie du logiciel, nous avons appris en GEN (Génie Logiciel) comment mettre en place un système de tests unitaires. Nous avons donc appliqué ce principe sur notre projet avec JUnit. Etant donnée que la partie Java de notre projet est fondamentalement composée sur une base de données, et que nous n’avons pas encore vu comment utiliser les mocks, nous n’avons donc malheureusement pas utilisé ce principe.

Voici une image qui illustre le bon fonctionnement du système de tests unitaires.

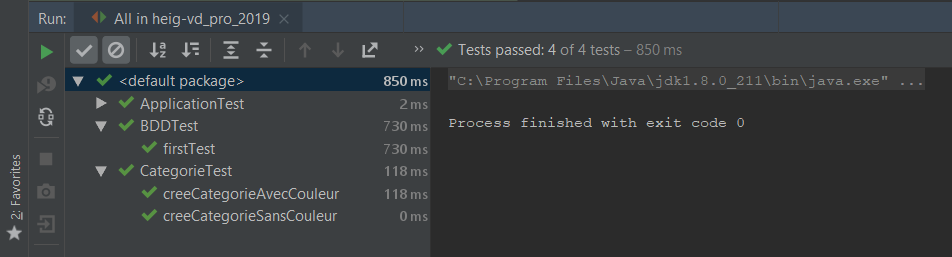
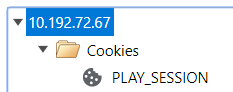


Figure 2 Tests unitaires

### Tests de la session

Avant de déployer le projet sur le serveur de l’HEIG-VD, nous avons pu tester le principe de session en local avec plusieurs navigateurs. Sans le principe de session, lorsqu’un utilisateur se connectait à son compte, toutes les autres fenêtres et navigateurs était aussi connecté sur le même compte. Alors que bien évidemment en ajoutant le principe de session de Play, chaque fenêtre et navigateurs avait sa propre session. Voici comment nous avons pu nous en assurer au tant en local que sur le serveur de l’HEIG-VD.



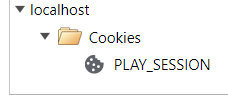


Figure 3 Cookie sur le serveur

Figure 4 Cookie en local

### Tests du pool de connexions

Afin de tester l’utilité et le fonctionnement du pool nous avons faits quelques tests.

Voici les activités faites sur la base de données lors de l’utilisation du projet sans le principe du pool :

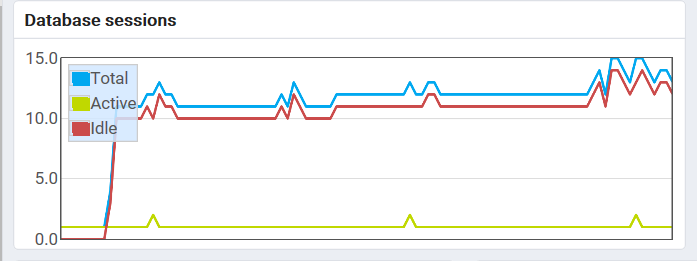


Figure 5 Connexions sans pool

On peut voir que les connections ne sont pas stable et ne cesse d’augmenter jusqu’à atteindre un maximum et bloquer. Ensuite, nous avons relancé le projet avec le système de pooling :

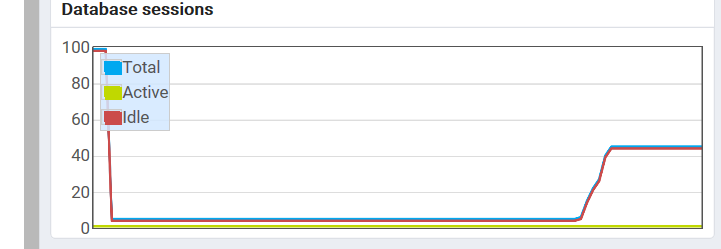


Figure 6 Connexion avec pooling

On peut voir qu’une de nos configurations est d’avoir un minimum de 40 connections en idle afin de toujours être prêt à la demande d’une connexion. Finalement, nous avons simulé beaucoup d’activité afin de voir si l’activité sur la base de données se comportait de façon stable et régulière :

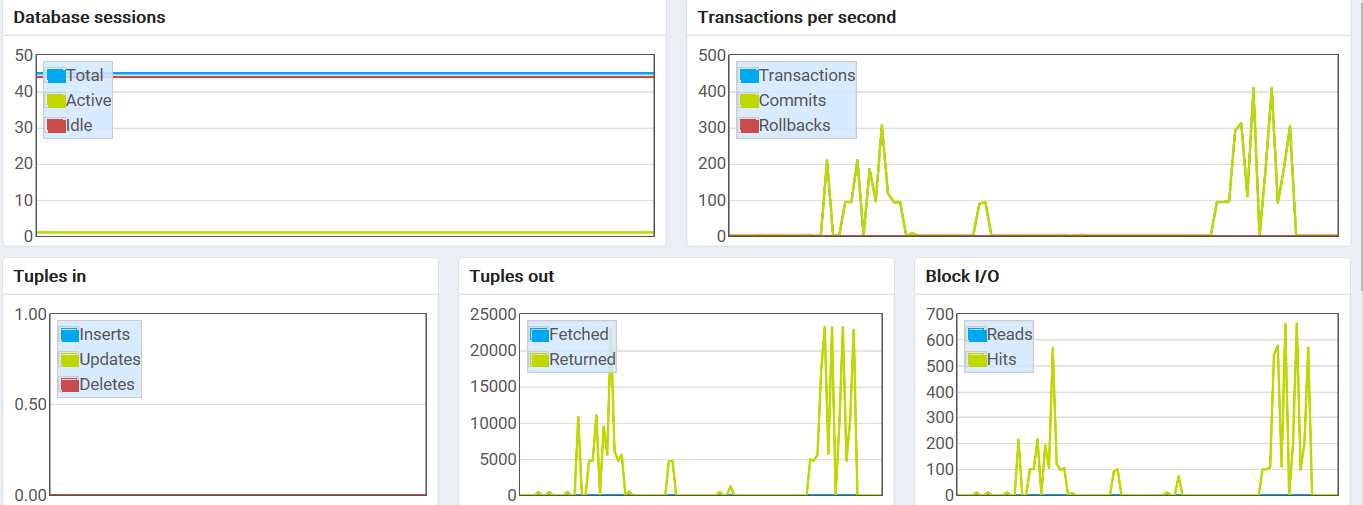


Figure 7 Test de la connexion avec pooling

Ici on peut voir que nous faisons pleins de passage sur la base de données et que l’activité reste stable et régulière.

## Bugs restants

# Améliorations

## Tests

Il est possible de faire des tests unitaires avec des bases de données (BDUnit, DbFit, Mock, ... ). Nous devrions apprendre ceci en GEN plus tard.

## Base de données

La base de données dans son état actuel a une structure acceptable, mais il serait tout de même possible d’améliorer certains points pour rendre son utilisation plus agréable et qu’elle possède plus d’informations utiles, par exemple :

* La table notification pourrait posséder un champ supplémentaire contenant la date à laquelle la notification est apparue afin de ne pas avoir de doute quant au moment d’apparition de cette dernière.
* La table notification pourrait posséder un autre champ supplémentaire qui serait simplement un boolean qui servirait à savoir si la notification à été vue ou non (afin de signaler les nouvelles notifications à l’utilisateur par exemple).
* La plupart des procédures et des triggers ont été écrits avec une connaissance minime du langage, il serait donc possible de les optimiser et de les rendre plus lisibles.

## Expérience utilisateur / features

* On pourrait faire savoir aux utilisateurs qu’une notification n’a pas été vue en l’affichant en rouge par-dessus l’onglet Notifications (ou autre).
* Étant donné qu’il n’est actuellement pas possible de supprimer les transactions récurrentes et les limites, c’est une option qui serait la bienvenue.
* Il serait possible d’ajouter une messagerie afin que les utilisateurs puissent communiquer entre eux.
* Il serait possible de permettre aux utilisateurs de visiter le profil des autres et de voir dans quelle catégorie ils dépensent le plus (sans forcément montrer de chiffres, juste des graphiques qui montrent la répartition des dépenses).
* Il n’est actuellement pas possible de se désinscrire ou de désactiver son compte, on pourrait ajouter cette option afin que les utilisateurs n’aient pas peur que certaines de leurs informations soient disponibles à jamais aux yeux de tous.
* Il serait possible d’ajouter d’autres types de statistiques ou d’augmenter les choix de l’utilisateur quant à ce qu’il veut voir (statistiques par mois, comparaisons entre périodes précises, autre…).
* Les limites se calculent actuellement par rapport à la date actuelle, ainsi si on est le 15 mai et qu’on a une récurrence mensuelle, on va regarder si la limite n’a pas été dépassée par rapport au 15 avril. Il serait mieux de checker les limites par rapport au mois courant, du 1er au 31 mai dans notre exemple.

# Remerciements

# Conclusion

## Situation finale du projet

## Fonctionnement du groupe

## Apprentissage et mise en pratique

Mise en pratique : BDR (Base de donnée), POO(Java), RES(Docker, Protocol), SER (Ecriture d’un document PDF), ADS/SOS (Gestion d’un serveur en ligne de commande), GEN (Cahier des charges, diagrammes UML, Planification du développement et GIT), TIB (Compréhension des adresse IP et ports), SLO (Utilisation d’une fonction de hashage non cassé «BCrypt» ).

Apprentissage : HTML, Javascript, CSS, SBT, Play, Organisation et communication avec un groupe, Livraison d’un grand projet, Rédaction d’un rapport/présentation d’un grand projet en groupe.

## Avis personnels

# Bibliographie

*A Simple Guide to Connection Pooling in Java*. (2019, 04 30). Récupéré sur Baeldung : https://www.baeldung.com/java-connection-pooling

*OpenClassRooms*. (2019, 03 04). Récupéré sur OpenClassRooms: https://openclassrooms.com/fr/

Play. (2019, 03 04). Récupéré sur Play Framework: https://www.playframework.com/

*PostGreSQL*. (2019, 03 04). Récupéré sur PostGreSQL: https://www.postgresql.fr/

*PostgreSQL JDBC*. (2019, 03 04). Récupéré sur postgresqltutorial: http://www.postgresqltutorial.com/postgresql-jdbc/

*Stackoverflow*. (2019, 03 04). Récupéré sur Stackoverflow: https://stackoverflow.com/

**(Remarque : ajouté les référence via Références -> Gérer les sources -> Nouveau)**

# Tables d’illustrations

Figure 1 Tests unitaires 5

Figure 2 Cookie sur le serveur 5

Figure 3 Cookie en local 5

Figure 4 Connexions sans pool 6

Figure 5 Connexion avec pooling 6

Figure 6 Test de la connexion avec pooling 6

**(Remarque : ajouté les référence via Références -> Insérer une légende)**

**(Insérer la table à la fin : Références -> Insérer une table…)**

# Annexes