DOCUMENT DE CONCEPTION

KORALLIA FRENETTE  
WILLIAM LEMIRE

TRAVAIL PRÉSENTÉ À JEAN-CHRISTOPHE DEMERS  
DANS LE CADRE DU COURS **420-C61-IN – PROJET SYNTHÈSE**

TECHNIQUES DE L’INFORMATIQUE  
CÉGEP DU VIEUX-MONTRÉAL  
08/09/2022

# Maquettes de l’interface graphique

Voici les maquettes pour les interfaces graphiques avec lesquelles l’usager va interagir dans les modules du logiciel trOqc.

Figure - Module de connexion

**Graphical user interface

Description automatically generated**

Figure 2 - Module de connexion – Création de compte  
  
**Graphical user interface

Description automatically generated**

Figure 3 - Module de Lobby

**Graphical user interface

Description automatically generated**

Figure 4 - Module de Lobby (Bloc-Notes)

**A picture containing graphical user interface

Description automatically generated**

Figure 5 - Module d'inventaire

**Graphical user interface, text

Description automatically generated**

*Figure 6 - Module d'inventaire (Ajouter un produit)* **Graphical user interface

Description automatically generated**

Figure - Module de finances

**Graphical user interface, text, application

Description automatically generated**

Figure 8 - Module de finances (Extrapolation)

**Graphical user interface

Description automatically generated**

Figure - Module de Configurations

**Graphical user interface, application

Description automatically generated**

# Conception UML

Voici différents diagrammes qui détaillent les cas d’usage du logiciel, ainsi que le diagramme de classes en UML.

## Digramme(s) des cas d’usage – détaillé(s)

Figure 10 - Cas d'usage du Module de Connexion

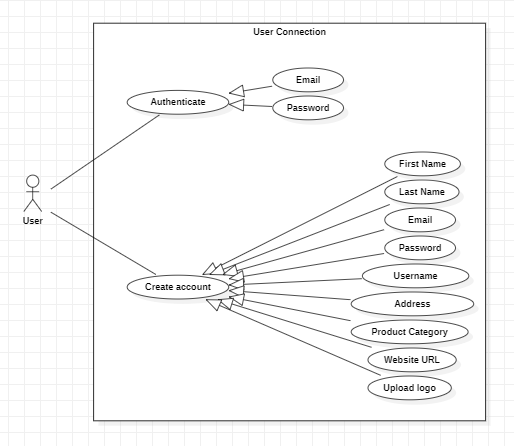
****

Figure 11 - Cas d'usage du Module de Lobby

**Diagram

Description automatically generated**

Figure 12 - Cas d'usage du Module de Finances

**Diagram

Description automatically generated**

Figure 13 - Cas d'usage du Module d'Inventaire

**Diagram

Description automatically generated**

Figure 14 - Cas d'usage du Module de Configurations

Diagram

Description automatically generated

## Diagramme(s) de classes – détaillé(s)

# Schéma(s) de la structure de données externe – détaillés

Figure 15 - Schéma de la base de données MySQL

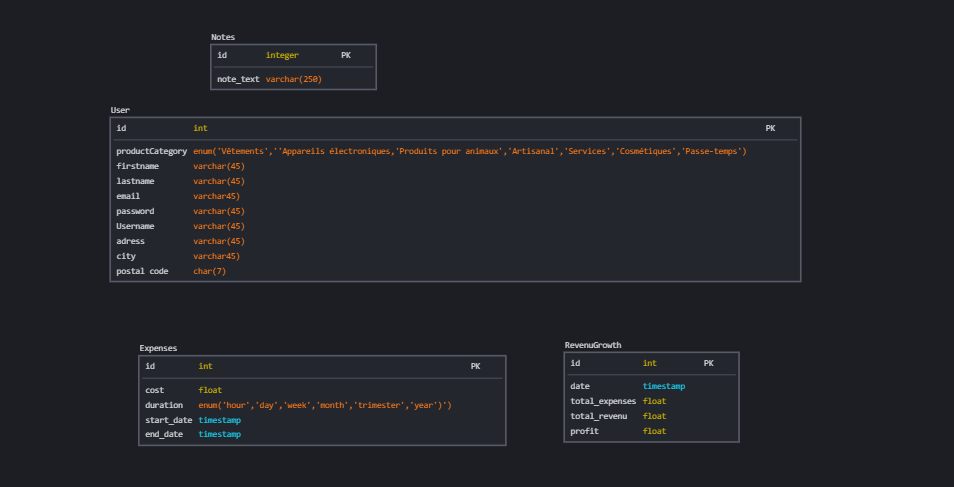
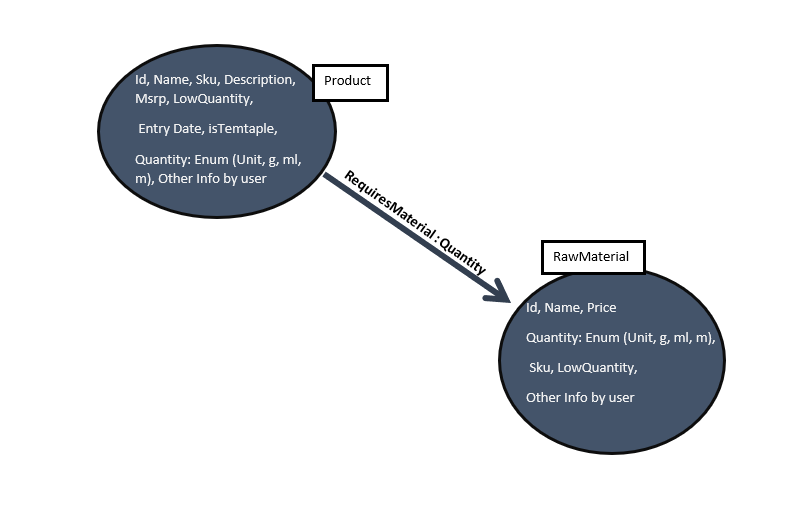


Figure - Base de données Neo4J



Nous allons utilisé Ne4j pour storer les produits de l’utilisateur , car nous voulons laissé celui-ci créant ces propres champs descriptifs. Pour accomplir ceci , nous avons besoin des propriété no-sql c’est-à-dire avoir la liberté de ne pas avoir exactement les mêmes informations à chaque entrée de donnée. De plus . neo4j permet de représenter des liens entre les différentes données avec facilité. Nous pouvons donc représenter le lien 1 vers N de produit et matérieau tout en ayant la flexibilié d’avoir des données variables.

Aussi, nous allons utilisé Beurkley, une base de donée nosql style « bucket » qui va nous permettre de storer les images de l’utilisateur , car Beurkley est plus performant pour gérer les images que sql et neo4j.

# Éléments de conception

## Structures de données

La première structure de données que nous avons choisi avec une attention particulière, est un dictionnaire pour stocker les templates de produits, ainsi que les champs supplémentaires que les usagers peuvent ajouter lors de la création d’un produit. Ce qui importe le plus dans ces cas-ci, est l’accès aux données, peu importe l’ordre, donc la complexité de cette structure (ordre de (1)) va être optimale pour ce besoin.

La deuxième est un set qui servira à stocker le nom des matériaux qui servent à créer des produits qui ont une basse quantité (spécifiée par l’usager), afin de les afficher d’une autre couleur pour avertir l’usager. Dans cette situation, un set est plus pertinent qu’un dictionnaire, puisque nous ne sommes intéressés que par le nom du produit et non la valeur associée. De plus, la complexité d’ordre O(1) en accès rend cette structure de données très pertinente pour cet usage.

Finalement, la troisième structure de données, et celle dont nous allons implémenter de toutes pièces, est une liste chaînée. Cette structure de donnée servira à contenir une liste des points qui se trouvent sur la droite de profits, en fonction du temps spécifié par l’usager. Cette liste sera utilisée principalement pour ajouter des points, donc puisque nous allons toujours ajouter les points un après l’autre à la fin de la liste, nous bénéficions d’une complexité d’ordre O(1). De plus, les listes chaînées sont plus appropriées que des listes classiques lorsque les données doivent être manipulées. Nous allons faire des calculs de régression sur ces points afin de faire une extrapolation du profit qui pourrait être généré dans le temps spécifié par l’usager.

## Patrons de conception

Le premier patron de conception que nous avons décidé d’utiliser est le patron Builder. Nous l’utiliserons afin de faciliter la création de Produits, puisqu’ils peuvent avoir différentes combinaisons de variables d’instance. Ceci nous évite d’avoir à coder d’innombrables constructeurs.

Nous avons aussi utilisé le patron DAO (Data Access Object) parce qu’il se prête bien au framework Spring, qui est fondé dans le concept de l’encapsulation de données grâce à l’inversion de contrôle (Inversion of Control). Le DAO était un choix naturel qui nous permet pousser l’encapsulation en séparant le traitement des données dans les différents servlets (modules) et l’accès aux données. Les modules peuvent donc que ne se concentrer sur leur logique. Dans le module Lobby, en cliquant sur le bloc-notes, le DAO accède à l’information de la base de données, y apporte des changements au besoin, et la transmet au module de Lobby. Chacun de nos modules, soit des aspects (finances, inventaire, connection, etc. ) en ce qui a trait à *l’Aspect-Oriented Programming*, a son propre DAO qui accéder à l’information qui le concerne.

Notre logiciel est une application web, donc un patron de conception approprié est le MVC, soit le Modèle-Vue-Contrôleur où la vue (l’affichage) communique avec le modèle (la logique) via le contrôleur. Spring vient avec des fonctionnalités (en fait, Spring MVC est un framework en soit) qui utilisent le patron, en faisant un choix logique dans notre situation. Ce patron permet une encapsulation de ces trois (MVC) aspects fondamentaux à toute application web. Par exemple, lors de l’ajout de produits dans l’inventaire, la vue capte les informations à ajouter, puis le contrôleur les prend et les donne au modèle, qui finalement les insère dans la base de données par l’entremise du DAO du module d’inventaire.

## Expressions régulières

Nous allons utiliser une expression régulière à deux endroits afin de générer un code SKU (Stock Keeping Unit) qui permet d’identifier chaque produit de l’inventaire. Un patron sera spécifié par l’usager dans le module de configurations, qui s’appliquera aux produits créés dont le champs SKU est laissé nul. Toutefois, si l’usager veut générer un SKU à partir de colonnes supplémentaires, il pourra le faire durant la création du produit.

## Algorithme et mathématiques

L’algorithme que nous allons implémenter dans notre projet servira à extrapoler la tendance des ventes pour une période de temps spécifiée par l’utilisateur. Tout d’abord, nous devons déterminer le modèle de régression approprié grâce au calcul du coefficient de détermination pour les modèles linéaires, logarithmiques et exponentiels. Cet algorithme permettra de déterminer le modèle qui décrit le nuage de points (le profit) avec le plus de fiabilité. Quand nous connaissons le modèle approprié, nous allons pouvoir extrapoler des valeurs et les afficher à l’usager pour faire une projection des revenus pour la période spécifiée, c’est-à-dire, nous allons trouver les valeurs Y (le profit) associé à des valeurs X (le temps) futures avec le modèle que nous avons trouvé grâce à notre algorithme. Nous pouvons aussi afficher la droite, le type, et le taux de fiabilité de la régression pour offrir un maximum d’informations pertinentes à l’usager.