

# PharmApp

*Université de Bordeaux*

Jérémy Morin  
Louis Laine

# Contents

A propos du rapport . . . . .	2
Licence . . . . .	2
A propos des auteurs . . . . .	2
A l'attention . . . . .	2
Github link . . . . .	2
Cahier des charges . . . . .	3
Chargement des données . . . . .	3
Création d'une fonction d'usage . . . . .	3
Validité de la base de données . . . . .	3
Histogramme . . . . .	3
Recherche d'informations . . . . .	3
Bruit et précision . . . . .	3
Ajout de nouveau médicament . . . . .	4
Interface graphique . . . . .	4
Identification des ressources . . . . .	4
Fonctionnement . . . . .	5
Conception . . . . .	5
Logique métier . . . . .	5

## **A propos du rapport**

### **Licence**

Copyright © 2014 Jérémy MORIN & Louis Lainé

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

The Software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the Software or the use or other dealings in the Software.

### **A propos des auteurs**

Jérémy Morin : **Etudiant en Licence 3 MIAGe** : *jer.morin@free.fr*

Louis Lainé : **Etudiant en Licence 3 MIAGe**

### **A l'attention**

A l'attention de Madame Penois-Pinault

### **Github link**

## Cahier des charges

A la lecture du cahier des charges, nous avons déterminés quelles étaient les spécifications et les attentes. L'application devra couvrir la gestion d'une base de données minimaliste de médicaments et effets correspondants. Elle devra répondre aux spécifications suivantes :

### Chargement des données

- La persistance des données sera assuré par un fichier **texte**.
- Les données devront être chargés, une fois parsées dans l'application durant toute la durée de vie de l'application.

### Création d'une fonction d'usage

Une fonction d'usage devra être implémenté au cas ou l'utilisateur, n'a pas appelé le programme avec les bons paramètres. Une fonctionnalité d'aide, accessible via un -h ou -help.

### Validité de la base de données

Une exception devra être lancé, au cas ou la base de données n'est plus accessible.

### Histogramme

Nécessité de créer, un histogramme des effets secondaires existant correspondant à un médicament et de pouvoir visualiser cet histogramme.

### Recherche d'informations

L'application devra être en mesure, de faire plusieurs recherches sur la base de données.

- La liste des médicaments ayant un effet secondaire, en commun. L'utilisateur saisira un effet secondaire, l'application lui renverra l'ensemble des médicaments contenant cet effet.
- La liste des médicaments partageant le plus d'effet secondaires possibles avec un médicament saisi par l'utilisateur.
- L'utilisateur aura accès à tous les médicaments partageant un ou plusieurs effets secondaires. Pour ce faire l'utilisation de deux algorithmes, qui devront être développés :
  - Parcoure tous les médicaments et sélectionne ceux qui ont le plus d'effet.
  - Renvoie les deux premiers résultats trouvés avec au moins un effet secondaire identique.

### Bruit et précision

Calculer l'efficacité de l'algorithme de recherche par le biais de : nombre de Médicament trouvés / nombre de Médicaments normalement attribués ↔

### **Ajout de nouveau médicament**

L'utilisateur doit pouvoir ajouter un nouveau médicament

### **Interface graphique**

L'application sera livrée avec une interface graphique en plus de l'interface classique.

### **Identification des ressources**

Une fois le cahier des charges bien déchiffré, nous avons dû déterminer le moyen par lequel les contraintes allaient être mise en oeuvre.

## Fonctionnement

### Conception

Lors de la phase de conception nous avons décidé de mettre en place une architecture trois-tiers.

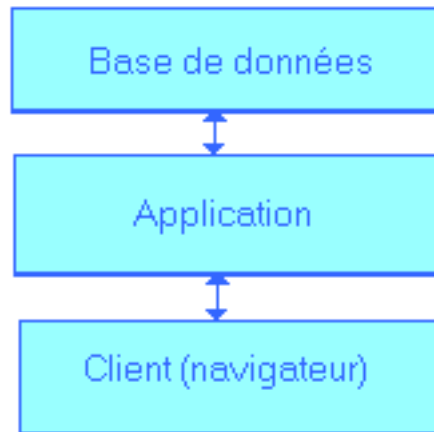


Figure 1: Schema de l'architecture trois-tiers

- **Présentation correspondant** à l'affichage, le dialogue avec l'utilisateur.
- **Traitement correspondant** à la couche métier, la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative
- **Accès aux données** correspondant aux données qui sont destinées à être conservées sur la durée, voire de manière définitive.

En suivant cette logique, nous avons donc conçu l'architecture suivante concernant l'application.

- La couche de présentation est représenté par les modules **mainTerm** et **mainGui**
- La couche de traitement est représenté par les packages **Abstract Parser** et **Dll**, représentant la logique applicative, ainsi que l'accès à la base de données.
- Les données, correspondent au fichier **liste.txt**

### Logique métier

#### ParserFichier

```
34. typedef std::map<string, std::vector<string> > mapParse;  
35. typedef std::map<string, std::vector<string> >::iterator mapParseIterator;  
36. typedef std::vector<string> leFichier;
```

C'est la classe qui va permettre la dialogue entre le fichier texte et les objets.

Cette classe va charger les données issue du fichier texte, dans une *mapParse*.

Cette map sera constitué en clé, d'un *string* associé avec un *vector*. Cette conception rappelle la base de données qui à la forme suivante :

```
nom du médicament : effet1, effet2 et effet3.
```

## ObjectFactory

```
34. typedef std::map<int, Medicament> dataMap;  
35. typedef std::map<string, std::vector<string> > parserMap;
```

ObjectFactory, est la classe permettant la transition des données en objet.

Cette couche, abstraite permettre de remonter les données en objet vers la couche supérieure.

La fonction *transformToObject()*, qui se base sur l'utilisation du *ParserFichier*, va parcourir chaque ligne du fichier et créer les objets correspondant, à l'aide des fonctions de *ParserFichier* et de la Dll pour créer les objets *Médicament* et *Effet*

## DataProvider

```
20. typedef std::map<int, Medicament> dataMap;  
...  
24. typedef std::map<string, std::vector<string> > parserMap;  
...  
28. typedef std::map<string, double> dataEffet;
```

La couche **DataProvider**, héritant directement de la couche inférieure *ObjectFactory*, sera considéré comme l'objet **base de données** de l'application.

Les données sont durant toute la durée de vie de l'appliacion chargé dans cet objet via un attribut privée *DataProvider::dataStorage*.

Ainsi toutes les fonctions de traitements concernant les données utiliseront le *DataProvider* comme source de données.

**Utilisation** Lors de l'instanciation de l'objet (équivalent à la création de la base de données) la méthode mère *TransformToObject()* sera appelé pour pour récupérer dans le scope les données et hydrater l'attribut *DataProvider::dataStorage*.

```
try{  
    this->dataStorage = this->transformToObject();  
}
```

```

} catch (ObjectFactory::FactoryException e){
    e.what();
}

```

Toutes les “requêtes”, utiliseront cet attribut comme source de données.

## Controller

C’est la couche d’abstraction pour la gestion des événements. Pour éviter de créer une fonction *main()* à rallonge et illisible, cette classe se chargera de renvoyer en fonction d’un potentiel choix utilisateur, les données mise en forme qui pourront être affichés dans la fonction *main()*, par la suite.

Cette classe se voulant polyvalente pour une utilisation graphique ou cli, renverra en fonction d’attribut passé en paramètre dans les méthodes de, renvoyer un affichage de données ou de renvoyer les données via un **dataMap**.

```

//Exemple d'une fonction
template <typename T>
T Controller::dispEffetFromMedoc(...) {
    //Fonction d'usage de la base de données
    try{
        this->tabTemp = this->database.getMapMedocFromEffet(theNomEffet);
    } catch (DataProvider::DBException e){
        e.getMessage();
    }

    ...

    //Le renvoie utile pour l'utilisation dans un main graphique par exemple.
    if (asReturn){
        return this->tabTemp;
    }

    ...

    return T();
}

```

Ainsi, qu’on développe un main graphique ou cli, la classe controller sera toujours celle qui sera utilisé pour la gestion des événements, grâce à l’utilisation de *template*.