 **Informaticien/-ne CFC**

Travail pratique individuel 2017 (TPI)

Travail Pratique Individuel 2017

Candidat N°: 122599

Amazon Reader



Nom du candidat : Onu Oktay Dogan

Classe du candidat : 304541

13 juin 2017

Table des matières

[1 Introduction 5](#_Toc485114431)

[2 Gestion du projet 5](#_Toc485114432)

[3 Phase d'initialisation 6](#_Toc485114433)

[3.1 Analyse de l’état actuel 6](#_Toc485114434)

[3.1.1 Analyse de l’état désiré 6](#_Toc485114435)

[3.2 Exigences de l’application 6](#_Toc485114436)

[3.2.1 Source de données 6](#_Toc485114437)

[3.2.2 Sauvegarde des articles dans une base de données 7](#_Toc485114438)

[3.2.3 Serveur Web 7](#_Toc485114439)

[3.2.4 La sécurité 7](#_Toc485114440)

[3.2.5 Client Web 8](#_Toc485114441)

[3.3 Architecture de l’application 8](#_Toc485114442)

[3.4 Choix des technologies 9](#_Toc485114443)

[3.4.1 Front-end 9](#_Toc485114444)

[3.4.2 Back-end 9](#_Toc485114445)

[3.4.3 Base de données 9](#_Toc485114446)

[3.5 Maquettes de l'interface graphique 10](#_Toc485114447)

[3.6 User stories 12](#_Toc485114448)

[3.7 APIs du serveur web 14](#_Toc485114449)

[3.8 Diagrammes 15](#_Toc485114450)

[3.8.1 Login 15](#_Toc485114451)

[3.8.2 Logout 15](#_Toc485114452)

[3.8.3 Récupération de la liste des utilisateurs 16](#_Toc485114453)

[3.8.4 Création d'un utilisateur 16](#_Toc485114454)

[3.8.5 Suppression d'un utilisateur 17](#_Toc485114455)

[3.8.6 Modification d'un utilisateur 17](#_Toc485114456)

[3.8.7 Recherche d'articles 18](#_Toc485114457)

[3.8.8 Import d'articles 18](#_Toc485114458)

[3.8.9 Modification d'un article 19](#_Toc485114459)

[3.8.10 Suppression d'un article 19](#_Toc485114460)

[4 Phase d'exécution 20](#_Toc485114461)

[4.1 Préparation de l'environnement 20](#_Toc485114462)

[4.1.1 Angular-CLI 20](#_Toc485114463)

[4.2 Import des données 21](#_Toc485114464)

[4.2.1 Analyse du contenu du fichier json 21](#_Toc485114465)

[4.2.2 Correction des erreurs 22](#_Toc485114466)

[4.2.3 Mapping 24](#_Toc485114467)

[4.2.4 Optimisations 25](#_Toc485114468)

[4.2.5 Persistance 25](#_Toc485114469)

[4.3 Utilitaire d'importation en ligne de commande 26](#_Toc485114470)

[4.4 Serveur RESTFul (API) 27](#_Toc485114471)

[4.4.1 Serveur https 27](#_Toc485114472)

[4.4.2 Routes 27](#_Toc485114473)

[4.4.3 Réponses 28](#_Toc485114474)

[4.4.4 Middleware 28](#_Toc485114475)

[4.5 Recherche des articles dans la base de données 29](#_Toc485114476)

[4.5.1 Indexation des données 29](#_Toc485114477)

[4.5.2 Ajout des articles 29](#_Toc485114478)

[4.5.3 Filtres 29](#_Toc485114479)

[4.5.4 Recherche des articles 30](#_Toc485114480)

[4.5.5 Limite 31](#_Toc485114481)

[4.6 Client Web 32](#_Toc485114482)

[4.6.1 Architecture 32](#_Toc485114483)

[4.6.2 Les composants 32](#_Toc485114484)

[4.6.3 Les services 33](#_Toc485114485)

[4.6.4 Affichage de messages 33](#_Toc485114486)

[4.6.5 Stratégie d'édition et de suppression 34](#_Toc485114487)

[4.6.6 Importation des données 34](#_Toc485114488)

[4.7 Sécurité 35](#_Toc485114489)

[4.7.1 Authentification 35](#_Toc485114490)

[4.7.2 Autorisations 36](#_Toc485114491)

[4.7.3 Protection des mots de passe 36](#_Toc485114492)

[4.7.4 Protection de l'accès à la base de données 36](#_Toc485114493)

[4.7.5 Attaques par injection SQL 37](#_Toc485114494)

[4.7.6 Minification 37](#_Toc485114495)

[4.8 Revue de code 38](#_Toc485114496)

[4.9 Test 38](#_Toc485114497)

[4.9.1 Tests manuels 38](#_Toc485114498)

[4.9.2 Tests unitaires 38](#_Toc485114499)

[4.9.3 Tests de bout-en-bout 40](#_Toc485114500)

[4.9.4 Tests de montée en charge 40](#_Toc485114501)

[5 Documentation 41](#_Toc485114502)

[5.1 Documentation des librairies 41](#_Toc485114503)

[6 Améliorations 42](#_Toc485114504)

[7 Conclusion 42](#_Toc485114505)

[8 Annexes 43](#_Toc485114506)

[9 Bibliographie 43](#_Toc485114507)

[10 Remerciements 43](#_Toc485114508)

# Introduction

Lors de ce travail pratique individuel (TPI) final, j'aurai à cœur de montrer mes connaissances en développement de logiciels acquises lors de ces 4 années d'apprentissage. Je vais donc m'appliquer à utiliser les meilleures pratiques.

# Gestion du projet

J'ai choisi de gérer ce projet en utilisant une variante simplifiée de la méthode *Agile Scrum* puisque je suis seul dans ce projet. Ma démarche sera la suivante :

* Phase d'initialisation
  + Analyse du cahier des charges
  + Choix technologiques
  + Architecture
  + Création des user stories pour l'établissement du *product backlog*
* Phase de planning
  + Préparation du planning du projet
  + Maquettes de l'application client
  + Diagrammes de séquences
  + Diagramme de cas d'utilisation
* Phase d'exécution
  + Préparation de 5 sprints pour la livraison du projet (le dernier sprint concernera la finition de la documentation et la programmation des tests end-to-end)
    - Import des données
    - API du serveur *RESTFul*
    - Client Web
    - Sécurité
    - Documentation et test end-to-end
  + Préparation du sprint *backlog* de chaque *sprint*
  + Utilisation du Kanban pour visualiser l'avancement de mon projet

Vu l'objectif et la durée du projet, je n'aurai pas de phase de "*monitoring*" ni de phase de "*closing*" durant ce projet.

# Phase d'initialisation

## Analyse de l’état actuel

Il s’agit d’un nouveau développement. Il n’y a pas d’état actuel.

### Analyse de l’état désiré

L’objectif de ce projet est de réaliser une application web permettant la recherche et la visualisation d'articles en se basant sur une source de données contenant plusieurs millions d'articles.

## Exigences de l’application

### Source de données

Pour réaliser ce projet j’ai besoin d’importer un fichier json (de 10 GB) qui contient quelques millions d’articles Amazon avec leur titre, leur prix, un lien vers une image, une liste de catégories.

L’obtention des données est conditionnelle à la citation des documents SIGIR et KDD ci-dessous.

**Citation**

Please cite one or both of the following if you use the data in any way:

**Image-based recommendations on styles and substitutes**  
J. McAuley, C. Targett, J. Shi, A. van den Hengel  
SIGIR, 2015  
[pdf](http://cseweb.ucsd.edu/~jmcauley/pdfs/sigir15.pdf)

**Inferring networks of substitutable and complementary products**  
J. McAuley, R. Pandey, J. Leskovec  
Knowledge Discovery and Data Mining, 2015  
[pdf](http://cseweb.ucsd.edu/~jmcauley/pdfs/kdd15.pdf)

L’importation des données devra se faire de 2 manières différents :

* Directement depuis le client web (package GO)
* Depuis une application desktop (en ligne de commande, en mentionnant le chemin du fichier à importer).

### Sauvegarde des articles dans une base de données

Les données doivent être sauvegardée dans une base de données offrant :

* De très bonnes performances en écriture.
* D'excellentes performances en lecture.
* Le support de recherches complexes.

### Serveur Web

* Le serveur web doit offrir des APIs qui permettront au client de récupérer les données.
* Le trafic de données qui transitent entre le client et le serveur doit être crypté.

### La sécurité

* Les utilisateurs doivent s’authentifier
* L’application se compose de 3 niveaux d’autorisations prédéfinis (rôles).
  + Les utilisateurs peuvent effectuer des recherches et afficher les articles
  + Les rédacteurs disposent des droits des utilisateurs et peuvent gérer les articles (modification et suppression).
  + Les administrateurs disposent des droits des rédacteurs et peuvent gérer les comptes des utilisateurs (création, modification et suppression).
* Les mots de passe doivent être cryptés et salés.
* Le serveur génère un jeton jwt ne contenant que l’identifiant de l’utilisateur. Le jeton circule avec les requêtes, encapsulé dans un cookie qui ne devra pas pouvoir être visible par le client.
* Le cookie doit avoir une durée de vie limitée à :
  + 1 jour pour les administrateurs
  + 1 semaine pour les rédacteurs
  + 1 mois pour les utilisateurs
* Il faudra générer un certificat (auto-signé).
* La connexion entre le serveur et la base de données doit être également protégée.
* Une protection contre les attaques de type injection sql doit être implémentée.
* La minification du code javascript

### Client Web

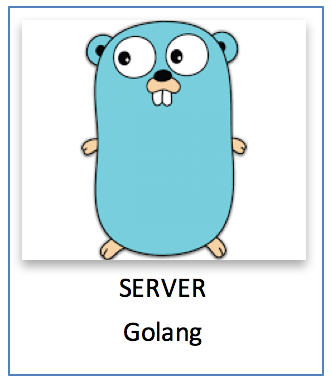
Une librairie permettant d’assurer une qualité à l’interface utilisateur.

* Si l’utilisateur n’a pas de compte, il peut en créer un lui-même. Par défaut, les nouveaux comptes ont un niveau d’autorisation « utilisateur ».
* Une fois authentifié, l’utilisateur peut effectuer des recherches pour afficher les articles :
  + Par mots-clés contenus dans le titre
  + Par catégorie
  + Par prix
  + Combinaison de plusieurs de ces choix
* S’il dispose du privilège « rédacteur » il peut également modifier ou supprimer des articles.
* L’administrateur doit pouvoir modifier les privilèges et supprimer des comptes utilisateurs.

## Architecture de l’application

L'application se compose des 3 éléments principaux suivants :

* Une interface utilisateur qui s'exécutera dans le navigateur de l'utilisateur et qui récupérera les données sur le serveur
* Un serveur *RESTFul* qui fournira les API permettant de transférer les données qu'il récupère dans la base de données. Il mettra également une adresse à disposition afin de charger l'application web dans le navigateur. Le serveur est également le garant de la sécurité de l'application.
* Une base de données qui traite les requêtes du serveur et lui fournit les données.



## Choix des technologies

### Front-end

J'ai passé une année à utiliser *Angular.js* et 6 mois à utiliser *Angular 2*. Ce sont des frameworks que je maitrise et que j'apprécie beaucoup. Le choix s'impose donc à moi naturellement.

La version actuelle de *Angular* est cependant la 4. Par chance, elle est identique à la version 2, je n'aurai donc pas à réapprendre de nouveaux concepts.

En ce qui concerne les éléments graphiques, je vais m'appuyer sur la bibliothèque *Angular Material 2* que j'ai déjà eu l'occasion d'utiliser. Je suis le seul chez Xpert à la connaître car elle est encore en version bêta mais je pense qu'elle est suffisamment stable pour ne pas rencontrer de problèmes insurmontables.

### Back-end

Mon choix s'est tout de suite porté sur Go car c'est le langage que je maitrise le mieux pour le back-end et également parce qu'il est très performant. Il possède d'autres qualités qui en font un choix idéal (voir annexe 1 🡪 Pourquoi Xpert Technologies utilise Go).

De plus, je vais pouvoir très rapidement développer mon serveur *RESTFul* sans devoir installer et configurer un serveur http.

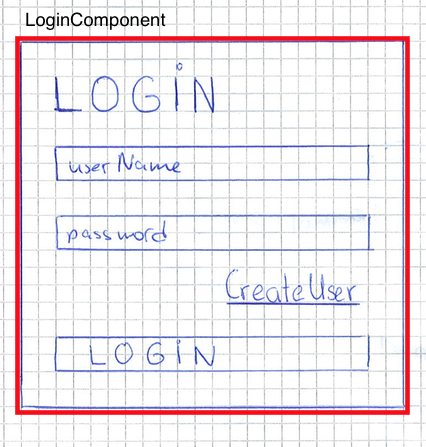
### Base de données

J'avais initialement opté pour *CouchBase* qui est une base de données *NoSQL* orientée document offrant d'excellentes performances en lecture et écriture.

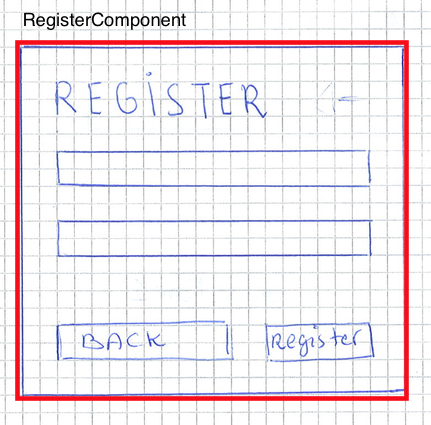
Après discussion avec mon maître de stage, j'ai finalement choisi d'utiliser *ElasticSearch* qui est également une base de données *NoSQL* orientée document mais offre un moteur de recherche *full-text* qui sera, m'a-t-on dit, très utile pour effectuer des recherches performantes dans les millions d'articles disponibles.

## Maquettes de l'interface graphique

Mon application va débuter par l'identification de l'utilisateur. En cas de succès, le serveur retournera le nom de l'utilisateur ainsi que son rôle.

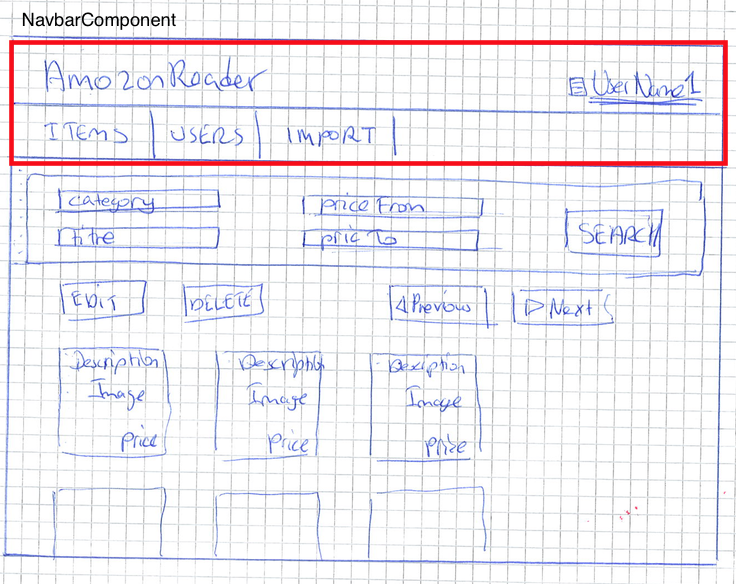


Si un utilisateur ne dispose pas encore de compte, il pourra en créer un lui-même. Initialement, il recevra le rôle d'utilisateur.

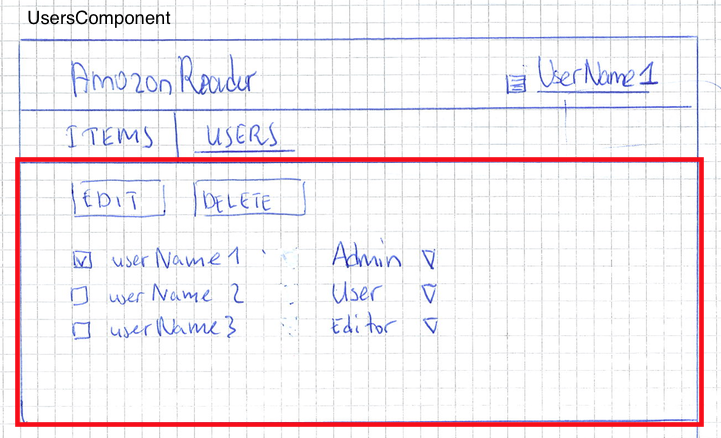


Lorsque l'utilisateur a été identifié avec succès, il accèdera à l'écran principal de l'application où trois onglets seront disponibles en fonction de son privilège :

* **Articles**  
  permet de rechercher et d'afficher les articles par page de 15 éléments. En outre, les rédacteurs et les administrateurs peuvent modifier les articles.



* **Utilisateurs** (uniquement pour les administrateurs)  
  permet d'accéder à la liste des utilisateurs et de modifier leur rôle.



* **Import** (uniquement pour les administrateurs)  
  permet d'importer un nouveau fichier json pour ajouter ou mettre à jour des articles.

## User stories

Après lecture attentive du cahier des charges, j'ai identifié les fonctionnalités principales que j'ai saisie dans des *users stories*.

Les *user stories* sont composée de 2 parties :

* La description d'une fonctionnalité sous la forme "en tant que <*personne*>, je peux <*quoi*> afin de <*pourquoi*>", où le pourquoi est optionnel.
* La liste des critères d'acceptation qui, une fois validée durant la phase de test, me permettront de considérer la tâche comme effectuée.

|  |  |
| --- | --- |
| **Titre**: Création d’un compte utilisateur | **N°**: 1 |
| **Description**:  En tant qu’**utilisateur** de l’application, je peux créer mon propre compte.  **Critères d’acceptation :**   * Le compte créé reçoit un rôle minimal, c’est-à-dire celui d’utilisateur. | |

|  |  |
| --- | --- |
| **Titre**: Accès à l’application | **N°**: 2 |
| **Description**:  En tant qu’**utilisateur**, **rédacteur** ou **administrateur** je dois m’authentifier afin d’accéder à l’application.  **Critères d’acceptation :**   * L’application n’est disponible que pour les personnes authentifiées, sinon l’application redirige l'utilisateur vers la fenêtre du login. * Si l’authentification est incorrecte, l’application affiche un message d’erreur. * Une fois l'authentification effectuée avec succès, le nom de l’utilisateur est affiché de manière visible dans l’application. | |

|  |  |
| --- | --- |
| **Titre**: Gestion des utilisateurs | **N°**: 3 |
| **Description**:  En tant qu’**administrateur** je peux accéder à la gestion des utilisateurs.  **Critères d’acceptation :**   * Le menu de gestion des utilisateurs ne doit être visible que pour les administrateurs. * Les administrateurs, et eux uniquement, peuvent modifier le rôle des utilisateurs. * Les administrateurs, et eux uniquement, peuvent supprimer un utilisateur. * La modification ou la suppression n’est possible que si au minimum un administrateur reste actif. | |

|  |  |
| --- | --- |
| **Titre**: Import des articles | **N°**: 4 |
| **Description**:  En tant qu’administrateur, je peux importer les articles d’un fichier json.  **Critères d’acceptation :**   * Le menu d’import ne doit être visible que pour les administrateurs. * Le fichier à importer doit être du type json. | |

|  |  |
| --- | --- |
| **Titre**: Recherche des articles | **N°**: 5 |
| **Description**:  En tant qu’**utilisateur**, **rédacteur** ou **administrateur** de l’application, je peux effectuer des recherches sur les articles.  **Critères d’acceptation :**   * Par mots-clés sur le titre. * Par catégorie. * Par prix (plage de – à). * Combinaison de plusieurs de ces choix. * L’affichage contient 15 articles par page. * Une pagination permet d’afficher les articles suivants ou précédents. | |

|  |  |
| --- | --- |
| **Titre**: Gestion des articles | **N°**: 6 |
| **Description**:  En tant que **rédacteur** ou **administrateur** je peux éditer et supprimer un article.  **Critères d’acceptation :**   * Seuls les rédacteurs et administrateurs ont accès à la gestion des articles. | |

## APIs du serveur web

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | Accès | URI | Action |
| POST | Utilisateur | /user | Créer un utilisateur |
| GET | Administrateur | /user/id | Obtenir un utilisateur |
| PUT | Administrateur | /user/id/role | Mettre à jour un utilisateur |
| DELETE | Administrateur | /user/id | Supprimer un utilisateur |
| GET | Administrateur Rédacteur Utilisateur | /users | Récupérer tous les utilisateurs |
| GET | Administrateur Rédacteur Utilisateur | /item/id | Obtenir un item |
| PUT | Administrateur Rédacteur | /item/id | Mettre à jour un article |
| DELETE | Administrateur Rédacteur | /item/id | Supprimer un article |
| GET | Administrateur Rédacteur Utilisateur | /items | Récupérer les articles en fonction des critères de recherche données dans le body de la requête |
| GET | Administrateur | /import | Importation des articles |
| POST | Administrateur Rédacteur Utilisateur | /login | Authentification |
| GET | Administrateur Rédacteur Utilisateur | /logout | Déconnexion et suppression du cookie |

## Diagrammes

### Login



### Logout



### Récupération de la liste des utilisateurs



### Création d'un utilisateur



### Suppression d'un utilisateur



### Modification d'un utilisateur



### Recherche d'articles



### Import d'articles



### Modification d'un article



### Suppression d'un article



# Phase d'exécution

## Préparation de l'environnement

Il n'est pas prévu que je déploie l'application sur un vrai serveur. La présentation s'effectuera sur mon ordinateur.

La préparation consiste donc à mettre à jour Angular-CLI (V4.1.3) et Go (V1.8.3) qui sont déjà installés sur ma machine.

La base de données est le seul élément nouveau de mon environnement. J'ai été impressionné par la facilité d'installation d *ElasticSearch* puisqu'il m'a fallu 10 minutes, téléchargement compris, pour être en mesure de l'utiliser.

### Angular-CLI

Angular-CLI est un outil très pratique qui simplifie le développement de l'application client en transpilant le code TypeScript en javascript, en rechargeant les pages dans le navigateur lorsqu'une modification a été enregistrée, et bien d'autres choses encore. Il me sera également très utile à la fin du projet pour générer le code minifié.

Toutefois, pour pouvoir profiter de ces avantages, je vais travailler avec 2 serveurs durant la phase de développement : l'un pour télécharger l'application web au démarrage de l'application et l'autre pour récupérer les données.

Les navigateurs modernes considèrent ce fonctionnement comme dangereux. Ils y voient une faille potentielle de sécurité connue sous le nom de CORS (Cross-Origin Resource Sharing) et bloque les requêtes effectuées sur le second serveur. Pour le résoudre, il faut indiquer dans les entêtes des réponses de mon serveur Go que ce cas de figure est permis. **Cette permission devra cependant être retirée dans la version productive**.



## Import des données

### Analyse du contenu du fichier json

L'importation des données a été un défi coriace.

Afin d'analyser le contenu du fichier de 10,54 GB, j'ai écrit quelques lignes de code en Go permettant de limiter la lecture à quelques lignes.

J'ai ensuite pu constater que le fichier était mal formaté. A la place des guillemets, les délimiteurs utilisés étaient des apostrophes … sauf lorsque l'information contenait des apostrophes.

{'asin': '0000401048', 'title': "The rogue of publishers**'** row;: Confessions of a publisher (A Banner Book)", 'imUrl': 'http://ecx.images-amazon.com/images/I/41bchvIfgaL.jpg', 'related': {'also\_viewed': ['068240103X']}, 'salesRank': {'Books': 6448843}, 'categories': [['Books']]}

Une correction des données s'est avéré nécessaire avant de pouvoir procéder à l'importation.



Imaginant qu'il faudra plusieurs imports, notamment pour retrouver des données correctes après mes tests, j'ai cependant opté pour une autre approche qui consiste à générer un fichier corrigé qui sera réutilisé pour les imports suivants afin de perdre moins de temps.



Le fichier contient 9'430'088 lignes mais je ne sais pas encore s'il y a des doublons ou non.

### Correction des erreurs

Sur la base des constatations faites durant l'analyse, j'ai établi mon concept de correction des données :

* Parcours du fichier, ligne par ligne.
* Parcours de la ligne, caractère par caractère.
* Recherche du délimiteur de départ (soit un guillemet, soit une apostrophe).
* Recherche du délimiteur de fin (le même que celui de départ non précédé d'un back slash).
* Remplacement des guillemets trouvés entre les délimiteurs par des apostrophes.
* Ignore les back slashs (\) trouvés entre les délimiteurs.
* Remplacement des délimiteurs par des guillemets dans la nouvelle chaine.
* Copie des informations dans la nouvelle chaine.
* Sauvegarde de la nouvelle chaine corrigée.

Le code *Go* correspondant au concept ci-dessus est présenté à la page suivante.

func CorrectQuotes(s1 string) (s2 string) {  
 inside := *false* marker := ""  
 c := ""  
  
 for pos, char := range s1 {  
 c = string(char)  
  
 **// Detect Start Marker**  
 if !inside && (c == singleQuote || c == doubleQuote) {  
 inside = *true* s2 += doubleQuote  
  
 if c == singleQuote {  
 marker = singleQuote  
 } else {  
 marker = doubleQuote  
 }  
  
 continue  
 }  
  
 **// Detect End Marker**  
 if inside {  
 if c == marker && s1[pos-1:pos] != backslash {  
 inside = *false* s2 += doubleQuote  
  
 continue  
 }  
 }  
  
 **// Replace " by '**  
 if inside && (c == doubleQuote) {  
 s2 += singleQuote  
 continue  
 }  
  
 **// Ignore \**  
 if inside && (c == backslash) {  
 continue  
 }  
  
 **// Transfer char in new string**  
 s2 += c  
 }  
  
 return  
}

### Mapping

Toutes les informations du fichier json ne sont pas intéressantes. De plus, je vais utiliser mes propres noms : *Asin* est spécifique à Amazon, je préfère utiliser *id*.

Les informations *related* sont trop détaillées. J'ai décidé de ne conserver que les liens avec les articles achetés (*also\_bought*). Il n’est plus donc plus nécessaire que *related* soit un objet, un tableau de *string* convient tout à fait.

En ce qui concerne les catégories, je ne sais pas à quoi sert un tableau de tableau chez Amazon mais, après discussion avec mon maître d'apprentissage, nous avons convenu de garder uniquement la première chaine de catégories :

[

**[**

**Clothing, Shoes & Jewelry Girls**

**],**

[

Clothing, Shoes & Jewelry Novelty,

Costumes & More Costumes & Accessories More Accessories Kids & Baby

]

]

Ainsi le champ *categories*, qui est de type [][]*string* chez Amazon, devient un simple tableau de *string* dans mon application.

|  |  |
| --- | --- |
| Structure Amazon | Structure de mon application |
| asin string | Id string |
| title string | Title string |
| price float64 | Price float64 |
| imURL string | ImURL string |
| related {  **alsoBought** []string  alsoViewed []string  boughtTogether []string  } | Related []string |
| brand string | Brand string |
| categories [][]string | Categories []string |

Les autres informations sont ignorées.

### Optimisations

Il est spécifié dans le cahier des charges que l'optimisation du code est importante. Une fois mon programme d'importation fonctionnel, j'ai donc pris soin de l'améliorer de la façon suivante :

* **Optimisation de la désérialisation**

Utilisation d'une librairie externe plus performante que celle de la librairie standard

* **Optimisation par parallélisme**

La lecture des données dans le fichier json est très rapide et ne peut de toute manière pas être optimisée. Le processus de correction et désérialisation peut, par contre, être accéléré en le traitant en parallèle.

Ces optimisations m'ont permis de réduire le temps de lecture des articles qui était initialement de 128 secondes à 41 secondes, soit près de 70% !



### Persistance

La phase finale consiste à sauvegarder les articles dans la base de données. Chaque article est enregistré sous la forme clé-valeur dans *ElasticSearch* :

* La clé est son n° d'article provenant du fichier json.
* La valeur est un document json contenant toutes les données de l'article.

Grâce au document fourni par mon maître de stage, j'ai pu rapidement me connecter à *ElasticSearch* et connaître les requêtes de base. Pour la sauvegarde des articles, j'utilise une librairie qui me simplifie la tâche.

L'import a été effectué avec succès en 2h15'. Au final, 9'430'088 articles ont été ajoutés, ce qui correspond au nombre de lignes du fichier json.

## Utilitaire d'importation en ligne de commande

Mon utilitaire en ligne de commande s'appuie sur le code que j'ai développé pour importer les données et que j'ai transformé en librairie afin qu'il soit réutilisable.

Du coup, il ne faut que quelques lignes pour le rendre opérationnel :

package main

import (

"flag"

"fmt"

"github.com/onuroktay/Amazon-Reader/AmzR-pkg-import"

)

func main() {

// Read filename given by the user

fileName := flag.String("file", "amazon.json", "name of the json file to read")

flag.Parse()

fmt.Println("Import data from ", \*fileName)

// Import data

err := **OnurTPIjsonImporter.ImportJSON(\*fileName)**

if err != nil {

fmt.Println(err.Error())

}

}

**Utilisation de l'application en ligne de commande**

Afficher l'aide en utilisant les options –h ou –help ou --h ou --help :

AmzR-import-cli -h

Résultat :

Usage of ./AmzR-import-cli:

-file string

name of the json file to read (default "amazon.json")

Sans indication de l'option *-file*, le fichier *amazon.json* se trouvant dans le même répertoire que l'application est importé (s'il existe) :

./AmzR-import-cli

Il est possible d'indiquer le nom du fichier json à importer en utilisant l'option *–file* ou --*file* suivi d'un espace ou de *=*

./AmzR-import-cli –file=../json/amazon.json

## Serveur RESTFul (API)

### Serveur https

En Go, il n'est pas nécessaire d'installer et de configurer un serveur http, tels que *Apache Tomcat* ou *Microsoft IIS*. Le serveur est intégré dans la librairie *net/http* :

err = http.ListenAndServe(":8080", nil)

if err != nil {

fmt.Println(err.Error())

}

Le serveur étant le principal garant de la sécurité, il est nécessaire de crypter les communications entre le client et le serveur. A cette fin, j'ai généré un certificat auto-signé avec *openssl*. Comme l'adresse *ip* de mon ordinateur n'est pas fixe, j'ai choisi de le faire pour *localhost*.

Le démarrage du serveur n'est guère plus compliqué :

err = http.ListenAndServeTLS(":8080", certPath + "server.pem", certPath + "server.key", nil)

if err != nil {

fmt.Println(err.Error())

}

Cerise sur le gâteau, mon serveur prend également en charge le protocole HTTP/2 lorsque le navigateur le supporte.

### Routes

Le rôle du routeur est, en se basant sur l'URL reçu, de diriger une requête vers une fonction à exécuter afin de retourner la réponse attendue.

La ligne suivante, par exemple, exécute la fonction startPage lorsque l'URL racine est reçu, qui retourne la page index.html au navigateur :

r.HandleFunc(**"/"**, **startPage**).Methods("GET")

Dans le cas des API *RESTFul*, la réponse contiendra les données désirées au format json. J'ai prévu les API suivantes :

// USERS

r.HandleFunc("/users", getUsers).Methods("GET")

r.HandleFunc("/user", addUser).Methods("POST")

r.HandleFunc(""/user/{id}/{role}"", updateUser).Methods("PUT")

r.HandleFunc(""/user/{id}"", deleteUser).Methods("DELETE")

// ITEMS

r.HandleFunc("/item/{id}", getItem))).Methods("GET")

r.HandleFunc("/items", getItems))).Methods("POST")

r.HandleFunc("/item/{id}", updateItem))).Methods("PUT")

r.HandleFunc("/item/{id}", deleteItem))).Methods("DELETE")

### Réponses

Si l'URL demandée n'existe pas, la réponse aura un statut **404**, ressource non trouvée.

Je gère également les cas d'erreurs suivants :

* **401**, non autorisée, lorsque l'information d'authentification est manquante, erronée ou échue.
* **406**, pas acceptable, lorsque la taille des informations dans le corps de la requête dépasse la limite acceptée.

Dans les autres cas, mon serveur renvoie une réponse au format json avec un statut **200** contenant *data* et *success* :

* **success** est faux en cas de problème, par exemple si les données reçues dans la requête n'ont pas le format désiré ou que la base de données a retourné un message d'erreur, et vrai si la réponse est utilisable.
* **data** contient le résultat de la requête si *success* est à vrai ou un message d'erreur dans le cas contraire.

### Middleware

Le middleware permet d'emballer (wrap) les fonctions afin d'effectuer des actions ou contrôles avant et/ou après leur exécution. J'ai prévu de les utiliser pour :

* **Journaliser des requêtes** : cette tâche n'était pas demandée dans le cahier des charges et n'a pas été faite, faute de temps.
* **Authentifier les utilisateurs :**  Vérification de l'existence de l'utilisateur.
* **Contrôler les requêtes demandées :** Autoriser ou bloquer les requêtes en fonction du rôle de l'utilisateur.

Exemple d'ajout de middleware aux routes :

// Delete user  
r.Handle("/user/{id}", **checkAuth(*ADMIN***,  
 http.HandlerFunc(deleteUser))).Methods("DELETE")

// Update item  
r.Handle("/item/{id}", **checkAuth(*EDITOR***,  
 http.HandlerFunc(updateItem))).Methods("PUT") // update item

Dans mon implémentation, je passe au middleware le droit minimum dont il faut disposer pour être autorisé à utiliser l'API.

## Recherche des articles dans la base de données

### Indexation des données

*ElasticSearch* offre un moteur de recherche dit *full-text* qui permet de rechercher un ou une suite de mots. Chaque mot présent dans les enregistrements est indexé de manière à rendre sa recherche très rapide.

Il est cependant important de noter que seuls les mots complets seront trouvés.

### Ajout des articles

J'ai utilisé la librairie "*github.com/olivere/elastic"* pour sauvegarder les données dans la base.

Comme on peut le voir, son utilisation rend l'enregistrement d'un item est très simple :

func (es \*ELASTICSEARCH) SaveItem(item \*ITEM) (err error) {  
 \_, err = es.client.Index().  
 Index(es.\_indexName).  
 Type(es.\_type).  
 Id(item.ID).  
 BodyJson(item).  
 Refresh("false").  
 Do(context.TODO())  
  
 return  
}

### Filtres

Parmi toutes ces fonctions, *ElasticSearch* permet d'effectuer des recherches en spécifiant les critères suivants :

* **match**  
  permet de rechercher les éléments contenant au moins l'un des mots de la liste (OR)
* **match\_phrase**  
  permet de rechercher les éléments contenant tous les mots de la liste (AND)
* **size**  
  permet de récupérer *n* éléments
* **from**  
  permet d'indiquer un décalage à partir duquel on veut récupérer les éléments
* **sort**  
  permet de trier les éléments

Il est bien sûr possible de combiner ces critères, y compris les "*match*" et "*match\_phrase*".

J'ai choisi de ne pas utiliser match car je veux que le résultat de la requête ne contienne que des éléments répondant à tous les mots-clés que l'utilisateur a indiqués.

En me basant sur ces fonctionnalités, j'ai décidé d'offrir 4 critères de recherche dans mon application web :

* **Titre**  
  Liste de mots-clés à rechercher dans le titre de l'article
* **Catégorie**  
  Liste de mots-clés à rechercher dans la catégorie de l'article
* **Prix, à partir de**  
  Prix minimum de l'article à rechercher
* **Prix, jusqu'à**   
  Prix maximum de l'article à rechercher

### Recherche des articles

Malheureusement, j'ai rencontré quelques soucis lors de l'utilisation de cette librairie pour effectuer des requêtes : aucun résultat n'était retourné dès que la requête devenait plus complexe.

Heureusement, *ElasticSearch* dispose d'une interface RESTFul, ce qui m'a permis d'envoyer les requêtes sous une autre forme et d'obtenir le résultat souhaité. Il a cependant fallu pour cela que je construise les requêtes moi-même, ce que j'ai fait en me référant à la requête suivante :

GET /amazonreader/item/\_search

{

"query": {

"bool": {

"must": [

{

"match\_phrase": {

"title": {

"query": "lazy dog"

}

}

}

],

"filter": {

"range": {

"price": {

"gte": 10,

"lte": 20

}

}

}

}

},

"sort" : [

{"price" : {"order" : "asc"}}

],

"size": 2,

"from": 10

}

### Limite

J'ai constaté durant mes tests que je ne pouvais pas dépasser le 10'000ème enregistrement de ma requête. Si, par exemple, je veux afficher 15 articles par page, je ne pourrai afficher que les 666 premières pages. La page suivante me retournera un résultat vide.

Après recherche sur Internet, j'ai pu constater qu'il s'agit d'une limite imposée par *ElasticSearch*[[1]](#footnote-1).

J'imagine que, pour une application comme la mienne, ça n'est pas très important car il faut avoir de la patience et du temps pour consulter autant de produits en passant en revue des centaines de pages.

J'ai implémenté un contrôle pour vérifier le n° de page dans l'URL : si celle-ci dépasse la limite, je renvoie à la dernière page possible.

## Client Web

### Architecture

Le client web a une architecture Modèle – Vue – Contrôleur (MVC).

L'utilisation du router Angular simplifie le développement et permet le contrôle d'accès aux URL internes (*CanActivate*) et permet d'éviter de quitter un composant dans lequel des données doivent être validées (*CanDeactivate*).



### Les composants

Les composants *Angular* contiennent un template (la vue) et une classe (le contrôleur) dans lequel le code métier est implémenté.

### Les services

L'accès aux données s'effectue par l'intermédiaire des services (modèle).



J'ai choisi de sauvegarder les données des articles (maximum 15) et des utilisateurs (tous) dans les services. Lors de leur utilisation dans les composants, le tableau d'objet local pointera vers celui se trouvant dans le service. En cas de suppression d'un élément dans le service, le composant supprime donc automatiquement l'élément dans la vue.

### Affichage de messages

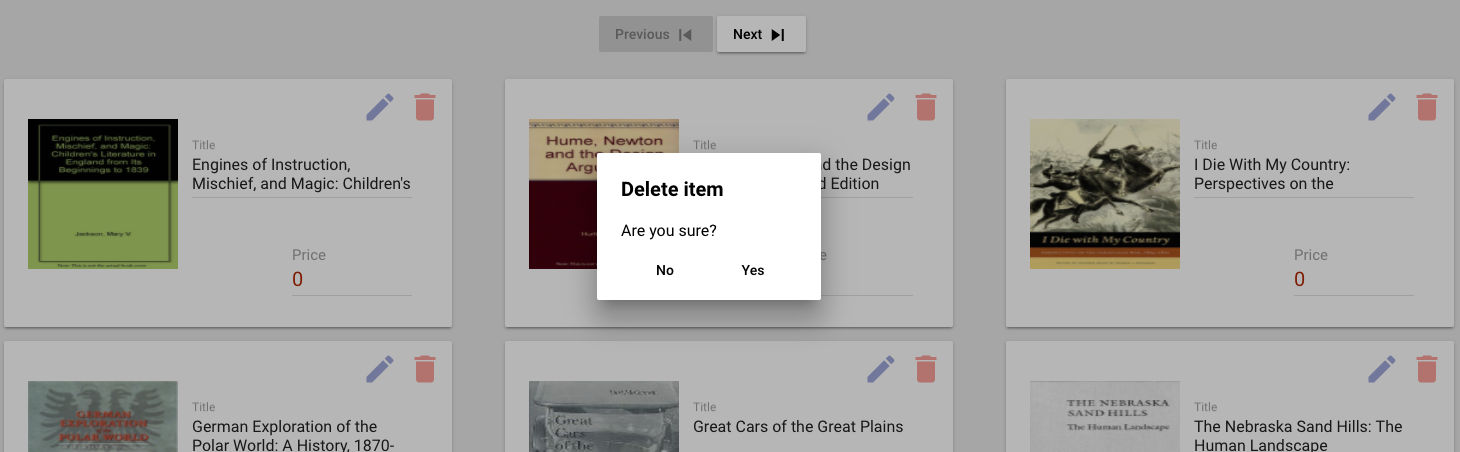
J'ai prévu l'affichage de messages afin que l'utilisateur sache si son action a été effectuée ou non. Ceci ne concerne bien sûr que les actions "invisibles" : par exemple, lors de la modification d'un article dans la base de données. L'affichage de messages n'auraient pas de sens pour valider une authentification correcte, par exemple, car l'utilisateur peut voir le résultat correct lorsque l'application affiche la page suivante.

### Stratégie d'édition et de suppression

Pour ce projet, j'ai décidé de permettre l'édition et la suppression directement dans la liste affichée. Si la personne est autorisée, les boutons correspondants à ces actions s'afficheront sur la ligne (utilisateur) ou dans la carte (article) de l'élément.

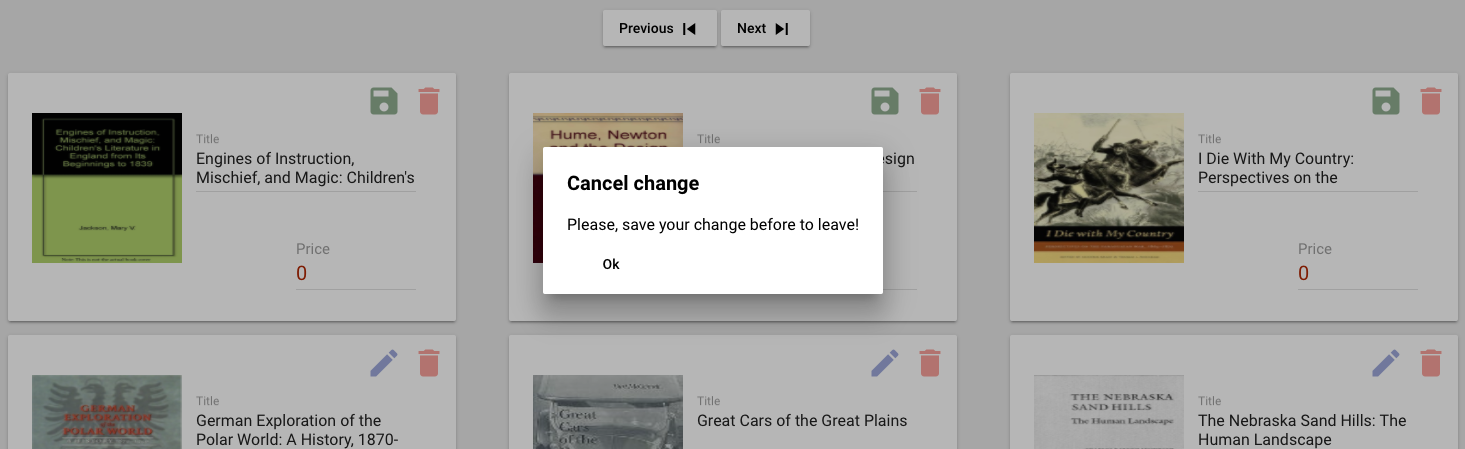
#### Suppression

Lors de la suppression d'un élément (article ou utilisateur), un message de confirmation apparaît. Une fois validé, l'élément supprimé disparaît de la liste. Aucun nouvel élément vient le remplacer dans le cas des articles.



#### Modification

Il est possible de modifier tous les éléments présents à l'écran. Cependant, si des éléments sont toujours en édition, l'application affichera un message indiquant que tous les changements d'avoir avoir été validés avant de quitter la page.



### Importation des données

En raison du manque de temps, j'ai dû simplifier l'importation des données. J'avais prévu un onglet permettant de sélectionner un fichier et affichant l'avancement dans une barre de progression. J'aurais également interdit de quitter la page tant que la tâche était en cours.

En lieu et place, j'ai juste ajouté un bouton de sélection de fichier dans l'onglet des articles : aucun onglet dédié, aucune barre de progression et aucun contrôle de fin de tâche.

## Sécurité

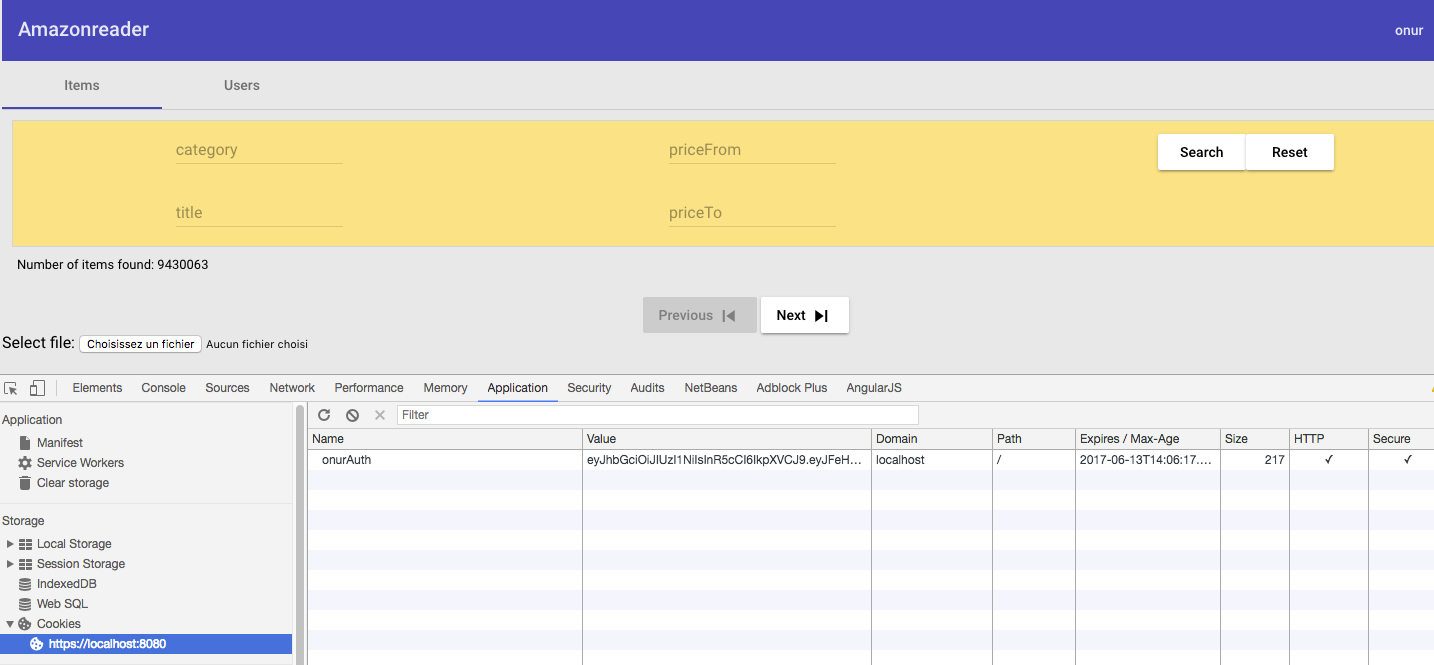
### Authentification

La sécurité est principalement assurée par le serveur. Lors de la requête d'authentification, le serveur génère un token jwt contenant l'identifiant de l'utilisateur en cours. J'utilise jwt pour m'assurer que le token n'a pas été modifié, bien que ça ne soit guère probable.

Le serveur génère également un cookie afin de pouvoir retrouver ce token à chaque nouvelle requête. De cette manière, je n'ai rien à développer au niveau du client.

Pour que la sécurité par cookie fasse sens, il est impératif que la communication soit sécurisé car il faut éviter son vol lors du transport. Ce point a déjà été traité dans le chapitre 4.4.1 (*Serveur https*). Cela ne suffit cependant pas. J'ai également configuré le cookie en activant les 2 paramètres suivants :

* **Secure = true**  
  Le cookie n'est transféré que si la connexion est sécurisée (https) de bout en bout.
* **HttpOnly = true**Afin d'éviter le vol du cookie dans le navigateur, ce flag spécifie qu'il n'est pas lisible par Javascript. C'est le navigateur qui se charge de le renvoyer à chaque requête effectuée sur l'URL lié.

****

La durée de vie du token n'est pas importante. Je l'ai toutefois limitée à 32 jours (1 jour de plus que la durée de vie maximum du cookie). La durée de vie du cookie a été configurée comme spécifiée dans le cahier des charges :

* 1 jour pour les administrateurs
* 1 semaine pour les rédacteurs
* 1 mois pour les utilisateurs

Au-delà de cette période, l'utilisateur devra à nouveau s'authentifier.

### Autorisations

Grâce à l'identifiant de l'utilisateur qui se trouve dans le token qui, lui, se trouve dans le cookie, le serveur sait qui effectue la requête. Avant d'y accéder, il vérifie dans la base de données le rôle de cet utilisateur et le compare au privilège minimum transmis au middleware.

Si l'utilisateur est autorisé, le serveur exécute la fonction et retourne le résultat. Autrement, il renvoie un statut 401, signifiant que la requête n'est pas autorisée.

### Protection des mots de passe

Afin de vérifier l'authentification des utilisateurs, il est nécessaire de disposer de leur mot de passe. Il faut donc les sauvegarder dans la base de données, ce qui est dangereux et fortement déconseillé.

La meilleure pratique consiste à ne pas sauvegarder le mot de passe directement mais une version hachée de ce mot de passe. C'est ce que j'ai fait en utilisant une librairie implémentant *bcrypt[[2]](#footnote-2)*.

En raison des performances des ordinateurs, le hachage n'est aujourd'hui plus suffisant pour protéger les mots de passe. C'est pourquoi, *bcrypt* effectue également un salage. Les avantages de bcrypt sont :

* Les mots de passe identiques apparaissent différents dans la base de données.
* Les attaques par force brute prennent plus de temps et coûte plus cher en capacité de calcul.

### Protection de l'accès à la base de données

J'ai trouvé les informations nécessaires afin de protéger l'accès à la base de données. La sécurité s'effectue en installant un utilitaire supplémentaire qui s'appelle X-Pack et qui permet de protéger l'accès par mot de passe et de crypter le transfert de données en utilisant un certificat à la norme X.509[[3]](#footnote-3).

L'usage de cet utilitaire est payant mais il est possible de l'installer pour des tests. Toutefois, le temps m'a manqué dans le projet et j'ai choisi de ne pas appliquer cet outil comme le cahier des charges m'y autorisait.

### Attaques par injection SQL

Les attaques par injection SQL ne représentent pas de danger avec *ElasticSearch* qui utilise un autre langage pour les requêtes. J'ai cependant contrôlé ce qui se passe lors de l'utilisation des apostrophes, guillemets et back slashs.

Lors de l'écriture, la librairie que j'utilise tient compte de ces caractères et leur enregistrement est effectué correctement dans la base de données.

Lors de la lecture, la requête que je créé peut effectivement être affectée et je dois les gérer. Il ne peut pas y avoir de conséquences autres qu'une réponse vide due à une mauvaise requête mais cela peut être gênant pour l'utilisateur. C'est pourquoi j'ai décidé de filtrer ces caractères. Comme ils ne sont pas indexés, la recherche s'effectue correctement et le résultat est celui attendu.

### Minification

Angular-CLI prend en charge la minification[[4]](#footnote-4). Cela rend le code plus difficile à lire côté client pour d'éventuels hackers car tous les espaces inutiles et les retours à la ligne sont supprimés (c'est pourquoi il ne faut pas oublier les ; à la fin des lignes – heureusement Tint y veille). De plus, les noms des variables sont remplacés par un nom court.

Lorsque le développement est terminé, la commande suivante permet de créer un répertoire *dist* (pour distribution) qui contiendra tous les fichiers nécessaires après concaténation et minification.

ng build

## Revue de code

Le contrôle de la qualité du code est une bonne pratique. Il existe des outils (Lint) pour Go comme pour *TypeScript*.

Pour *TypeScript*, il est intégré à *Angular-CLI*. En utilisant un IDE[[5]](#footnote-5) moderne, les erreurs ou propositions d'amélioration sont ajoutées lors du codage.

Il doit être démarré manuellement pour Go et indique également les commentaires manquants lorsqu'une fonction est publique :

golint

## Test

### Tests manuels

Les tests manuels sont effectués par une personne sur la base de protocoles décrivant précisément les tâches à effectuer.

* Les protocoles de tests des APIs du serveur se trouvent dans l'annexe 2 du rapport.
* Les protocoles de tests des fonctionnalités du client se trouvent dans l'annexe 3.

### Tests unitaires

Ecrits pendant la phase de développement (ou alors commencé avant dans le cas de la TDD), ils permettent de s'assurer que notre code fonctionne après une phase de refactorisation ou suite à l'ajout de fonctionnalités.

#### En Go

En Go, les tests unitaires font partie du langage et sont très simples à écrire. Il suffit de créer un fichier dont le nom se termine par "*\_test.go"* et d'importer la librairie *testing*. Ces fichiers ne seront pas pris en compte par le compilateur lors de la création de l'exécutable.

Ensuite, chaque test consiste à écrire une fonction dont le nom commence pat "*Test"* et qui reçoit un paramètre de type *\*testing.T* :

func TestUserFunctions(t \*testing.T) {

// Create account

account := &OnurTPIUser.Account{}

account.UserName = "test1"

account.RoleValue = 2

account.Password = "test"

database.accesser.CreateAccount(account)

// Read account in DB

cred := &OnurTPIUser.CredentialsClient{}

cred.UserName = "test1"

cred.Password = "test"

readAccount, err := database.accesser.GetAccountByUserNameInDB(cred)

if err != nil {

t.Error()

}

// Check account

if readAccount == nil {

t.Error()

}

if readAccount.UserName != account.UserName ||

readAccount.Password != account.Password {

t.Error()

}

// SaveUser sets rolevalue to 0

if readAccount.RoleValue != 0 {

t.Error()

}

// Delete Account

err = database.accesser.DeleteAccount(readAccount.ID)

if err != nil {

t.Error()

}

}

La commande *t.Error()* permet de faire échouer le test lorsque la valeur contrôlée n'est pas celle attendue.

Tous les tests sont exécutés avec la commande suivante :

go test

Sur le même principe, il est également possible de créer des exemples qui seront utilisés pour la documentation et pour les tests, ainsi que des benchmarks qui permettront de vérifier si l'on a dégradé ou amélioré les performances de notre fonction en la modifiant ou en installant une nouvelle version de Go.

Les exemples et benchmarks sont de bonnes pratiques que je n'ai cependant pas pu implémenter car le temps m'a manqué.

#### En Angular

Angular-CLI prend également nativement les tests unitaires en charge. Lors de la création d'un composant ou d'un service, un fichier avec une extension ".spec.ts" est créé.

J'ai malheureusement très peu d'expérience avec les tests unitaires en Angular. En raison de la grande quantité de travail à fournir et en accord avec mon maître d'apprentissage, il a été décidé de les retirer du cahier des charges afin de libérer ce temps au profit d'autres tâches.

### Tests de bout-en-bout

Les tests de bout-en-bout (ou end-to-end), permettent d'automatiser des tâches comme si un utilisateur les exécutait, et s'assurer que le résultat est conforme aux besoins du client.

J'ai malheureusement très peu d'expérience dans ce domaine car ils ne font pas partie de notre programme scolaire. En raison de la grande quantité de travail à fournir et en accord avec mon maître d'apprentissage, il a été décidé de les retirer du cahier des charges afin de libérer ce temps au profit d'autres tâches.

### Tests de montée en charge

Les tests de charge permettent de contrôler les performances de notre serveur en lui envoyant un grand nombre de requêtes en parallèle pendant quelques minutes et en analysant le temps de réponse.

L'outil le plus connu pour les réaliser est *ab[[6]](#footnote-6)*, très utilisé dans le monde Java. J'avais prévu d'utiliser *Vegeta*[[7]](#footnote-7) qui est un outil écrit en Go, malheureusement, le temps m'a manqué. Comme cette tâche ne figure pas dans le cahier des charges,

# Documentation

## Documentation des librairies

Le langage Go offre un outil permettant de générer automatiquement une documentation de qualité pour les librairies. Pour cela, il faut respecter les prescriptions du langage et utiliser GoLint pour s'en assurer.

Les documents générés se trouvent dans l'annexe 4 du rapport.

# Améliorations

Voici la liste des points que je n'ai pas eu le temps de développer :

* Trier les articles selon l'ordre alphabétique du titre.
* Proposer aux utilisateurs des catégories lors de la saisie.
* Offrir à l'utilisateur de choisir le nombre d'articles à afficher par page.
* Importation des données comme expliqué au chapitre *4.6.6*.
* Ajouter un choix permettant de sauvegarder tous les éléments édités (valable pour les articles et les utilisateurs) ou d'annuler toutes les modifications.
* Ajouter lors de l'édition un bouton pour annuler une modification (valable pour les articles et les utilisateurs).
* Apporter plus de soins à l'interface graphique.

# Conclusion

Ce projet a été passionnant car il couvre la plupart des connaissances que j'ai acquises dans le développement d'application web durant mon apprentissage. Il m'a également permis de me familiariser avec une nouvelle technologie très intéressante que je ne connaissais pas, *ElasticSearch*.

Dans l'ensemble, le projet s'est assez bien déroulé. J'ai été confronté à quelques soucis que j'ai pu régler seul la plupart du temps ou avec l'aide de collègues.

J'éprouve cependant une petite frustration car je n'ai pas pu apporter autant de soin à mon application que je l'aurais voulu en raison de la grande quantité de travail à fournir. Au final, j'aurais aimé disposer de 2 jours supplémentaires, notamment pour les tests dans Angular. J'ai très peu d'expérience dans ce domaine, c'est pourquoi nous avons décidé avec mon maître d'apprentissage, de soulager la charge de travail en supprimant les tests unitaires et end-to-end dans Angular du cahier des charges.

C'est la première fois que j'avais à gérer autant de données dans une application. En utilisant les technologies adéquates, j'ai pu constater que la quantité n'était pas un problème : en utilisant la pagination, l'affichage des articles s'effectue à la même vitesse si ma base de données contient 1'000 ou 10'000'000 d'éléments.

Je remercie mon maître d'apprentissage pour ce projet et me réjouis d'avoir l'occasion

# Annexes

Annexe 1 : Pourquoi Xpert Technologies utilise Go.

Annexe 2 : Tests manuels des APIs du serveur.

Annexe 3 : Tests manuels des fonctionnalités du client.

Annexe 4 : Documentation des librairies que j'ai créées.

Annexe 5 : Guide d’utilisation.

# Bibliographie

<https://blog.meshup.net/chrome-58-corriger-lerreur-missing_subjectaltname-avec-openssl/>

<https://material.angular.io/>

<https://github.com/elastic/elasticsearch>

# Remerciements

Je tiens à remercier M. Laurent Brülhart, maître d'apprentissage, ainsi que M. Cédric Ackermann pour leur soutien technique durant le projet.

Un énorme merci également à M. Ibrahim Nimaga et M. Cédric Ackermann pour la correction de mes documents. Ils ont vécu avec moi les derniers moments de stress du projet et ont été d'un soutien sans faille.

1. https://www.elastic.co/guide/en/elasticsearch/reference/5.2/index-modules.html#dynamic-index-settings [↑](#footnote-ref-1)
2. https://fr.wikipedia.org/wiki/Bcrypt [↑](#footnote-ref-2)
3. https://fr.wikipedia.org/wiki/X.509 [↑](#footnote-ref-3)
4. https://fr.wikipedia.org/wiki/Minification [↑](#footnote-ref-4)
5. EDI en français pour **E**nvironnement de **D**éveloppement **I**ntégré [↑](#footnote-ref-5)
6. https://httpd.apache.org/docs/2.4/fr/programs/ab.html [↑](#footnote-ref-6)
7. https://github.com/tsenart/vegeta [↑](#footnote-ref-7)