 **Informaticien/-ne CFC**

Travail pratique individuel 2017 (TPI)

Travail Pratique Individuel 2017

Candidat N°: 122599

Amazon Reader



Nom du candidat : Onur Oktay Dogan

Classe du candidat : 304541

12 juin 2017

Table des matières

[1 Introduction 5](#_Toc484800269)

[2 Gestion du projet 5](#_Toc484800270)

[3 Phase d'initialisation 6](#_Toc484800271)

[3.1 Analyse de l’état actuel 6](#_Toc484800272)

[3.1.1 Analyse de l’état désiré 6](#_Toc484800273)

[3.2 Exigences de l’application 6](#_Toc484800274)

[3.2.1 Source de données 6](#_Toc484800275)

[3.2.2 Sauvegarde des articles dans une base de données 7](#_Toc484800276)

[3.2.3 Serveur Web 7](#_Toc484800277)

[3.2.4 La sécurité 7](#_Toc484800278)

[3.2.5 Client Web 8](#_Toc484800279)

[3.3 Architecture de l’application 9](#_Toc484800280)

[3.4 Choix des technologies 9](#_Toc484800281)

[3.4.1 Front-end 9](#_Toc484800282)

[3.4.2 Back-end 9](#_Toc484800283)

[3.4.3 Base de données 10](#_Toc484800284)

[3.5 User stories 10](#_Toc484800285)

[3.6 Maquette de l'interface graphique 12](#_Toc484800286)

[4 Phase d'exécution 13](#_Toc484800287)

[4.1 Préparation de l'environnement 13](#_Toc484800288)

[4.1.1 Angular-CLI 13](#_Toc484800289)

[4.2 Import des données 14](#_Toc484800290)

[4.2.1 Analyse du contenu du fichier json 14](#_Toc484800291)

[4.2.2 Correction des erreurs 15](#_Toc484800292)

[4.2.3 Mapping 17](#_Toc484800293)

[4.2.4 Optimisations 18](#_Toc484800294)

[4.2.5 Persistance 18](#_Toc484800295)

[4.3 Utilitaire d'importation en ligne de commande 19](#_Toc484800296)

[4.4 Server RESTFul (API) 20](#_Toc484800297)

[4.4.1 Server http 20](#_Toc484800298)

[4.4.2 Routes 20](#_Toc484800299)

[4.4.3 Middleware 20](#_Toc484800300)

[4.4.4 Contrôle des données contenues dans les requêtes 20](#_Toc484800301)

[4.4.5 Réponses 20](#_Toc484800302)

[4.5 Recherche des articles dans la base de données 21](#_Toc484800303)

[4.5.1 Indexation des données 21](#_Toc484800304)

[4.5.2 Librairie vs RESTFul 21](#_Toc484800305)

[4.6 Client Web 22](#_Toc484800306)

[4.6.1 Architecture 22](#_Toc484800307)

[4.6.2 Composants 22](#_Toc484800308)

[4.6.3 Services 22](#_Toc484800309)

[4.6.4 Echange d'informations entre les composants 22](#_Toc484800310)

[4.6.5 Affichage de messages 22](#_Toc484800311)

[4.6.6 Stratégie d'édition 22](#_Toc484800312)

[4.6.7 Importation des données 22](#_Toc484800313)

[4.7 Sécurité 23](#_Toc484800314)

[4.7.1 Jwt, cookie et https 23](#_Toc484800315)

[4.7.2 Authentification 23](#_Toc484800316)

[4.7.3 Autorisations 23](#_Toc484800317)

[4.7.4 Protection des mots de passe 23](#_Toc484800318)

[4.7.5 Protection des données 23](#_Toc484800319)

[4.7.6 Attaques par injection SQL 23](#_Toc484800320)

[4.7.7 Minification et uglification 23](#_Toc484800321)

[4.8 Test 24](#_Toc484800322)

[4.8.1 Revue de code 24](#_Toc484800323)

[4.8.2 Tests manuels 24](#_Toc484800324)

[4.8.3 Tests unitaires 24](#_Toc484800325)

[4.8.4 Tests de bout-en-bout 25](#_Toc484800326)

[4.8.5 Tests de charge 25](#_Toc484800327)

[5 Documentation 26](#_Toc484800328)

[5.1 APIs de l’application 26](#_Toc484800329)

[5.2 Documentation des librairies 26](#_Toc484800330)

[6 Améliorations 27](#_Toc484800331)

[7 Conclusion 27](#_Toc484800332)

[8 Bibliographie 27](#_Toc484800333)

# 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 (je suis seul dans le 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
  + Maquette de l'application client
  + Diagramme 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 server RESTful
    - Client Web
    - Sécurité
    - Documentation
  + 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’éléments.

## 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.

* Les performances en écriture doivent être très bonne (importation en minutes).
* Les performances en lecture qui devront excellentes afin de supporter des recherches complexes (lecture en millisecondes).
* Les données devront être sauvegardées dans la base de données.

### Serveur Web

Le serveur web doit offrir des APIs sur lesquelles le client pourra s’appuyer pour récupérer les données.

Il faut s’assurer que les données qui transitent entre le client et la base de données soient cryptées.

Écrire les tests unitaires.

### 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

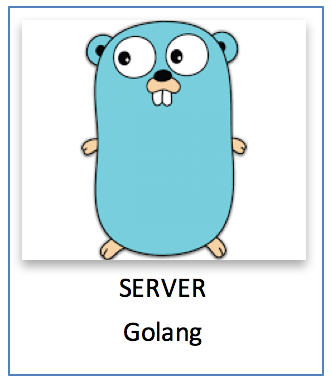
Une librairie permettant d’assurer une qualité à l’interface utilisateur.

* 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 ou phrases 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 ou supprimer des comptes utilisateurs.
* Écrire les tests unitaires et end-to-end

## 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 utiliser la bibliothèque Angular Material 2 que je connais également déjà. Je suis le seul chez Xpert à la connaître car elle est encore en version bêta mais je sais qu'elle est suffisamment stable pour ne pas rencontrer de problèmes.

### 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 server 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 me sera, m'a-t-on dit, très utile pour effectuer des recherches très rapidement dans les millions de produits disponibles.

## 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 n’affiche que la fenêtre du login * Si l’authentification est incorrecte, l’application affiche la fenêtre du login ainsi qu’un message d’erreur * 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 * Éditer un utilisateur existant afin de lui changer son rôle * Supprimer un utilisateur existant * 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.  **Critères d’acceptation :**   * Par mot ou phrases 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 | |

## Maquette de l'interface graphique

# Phase d'exécution

## Préparation de l'environnement

Il n'est pas prévu que je déploie l'application sur un serveur. La présentation s'effectuera également 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 consiste donc le seul élément nouveau de mon environnement. J'ai été impressionné par la facilité d'installation puisqu'il m'a fallu 10 minutes, téléchargement compris, pour être en mesure d'utiliser *ElasticSearch*.

### 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 considère ce fonctionnement comme une faille potentielle de sécurité connue sous le nom de CORS (Cross-Origin Resource Sharing) et bloque les requêtes effectuées sur le second server. Pour le résoudre, il faut indiquer dans les entêtes des réponses de mon server Go que ce cas de figure est permis. **Cette permission devra cependant être retirée dans la version définitive**.



## Import des données

### Analyse du contenu du fichier json

L'importation des données a été un défi plus 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'avère nécessaire avant de pouvoir procéder à l'importation.



Comme j'image qu'il faudra plusieurs imports, notamment pour retrouver des données correctes après mes tests, j'opte cependant 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)
* Remplacement des guillemets trouvés entre les délimiteurs par des apostrophes
* Ignore les backslashs (\) 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

Code correspondant au concept :

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 conserver 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 une simple tableau de *string* dans mon application.

|  |  |
| --- | --- |
| Structure Amazon | Structure de notre 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 – désérialisation peut par contre être accéléré en les 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

## Server RESTFul (API)

### Server http

ListenAndServe

### Routes

Appel des fonctions en parallèle

### Middleware

Implémentation de la partie authentification et contrôle d'autorisations

### Contrôle des données contenues dans les requêtes

Limite de taille

### Réponses

Format des réponses (data + success)

Status 200, 401, …

## 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.

### Filtres

### Librairie vs RESTFul

J'ai pu utiliser la librairie *github.com/olivere/elastic* sans problème pour sauvegarder les données. Par contre, j'ai rencontré quelques difficultés pour la lecture des données (aucun résultat retourné dès que la requête devenait plus complexe).

Heureusement, ElasticSearch dispose d'une interface RESTFul.

### 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.

## Client Web

### Architecture

Schémas

### Composants

Composants

### Services

Services

### Echange d'informations entre les composants

Via les services

### Affichage de messages

Messages

### Stratégie d'édition

Edition multiple, empêcher de quitter sans sauver, confirmation de suppression

### Importation des données

Pas eu beaucoup de temps

## Sécurité

### Jwt, cookie et https

La sécurité est principalement assurée par le serveur. Lors de la demande d

### Authentification

La sécurité est principalement assurée par le serveur. Lors de la demande d

### Autorisations

fsdafda

### Protection des mots de passe

Hashage et salage

### 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[[2]](#footnote-2).

### Attaques par injection SQL

Injection SQL

### Minification

Angular-CLI permet la minification. 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 TSLint y veille). De plus, les noms des variables sont remplacés par une version plus courte.

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[[3]](#footnote-3) 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 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 refactoring 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 :

import (

"testing"

)

func **Test**XY (t **\*testing.T**) {

**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

[Todo]

### Tests de bout-en-bout

Dans cette phase, on va dérouler les processus métier de bout en bout (vision utilisateur) et s'assurer qu'ils sont conformes aux besoins du Client

### 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[[4]](#footnote-4)*, très utilisé dans le monde Java. J'avais prévu d'utiliser *Vegeta*[[5]](#footnote-5) qui est un outil écrit en Go, malheureusement, je n'ai pas eu le temps.

# Documentation

## APIs de l’application

|  |  |  |  |
| --- | --- | --- | --- |
| 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  Administrateur | /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  Administrateur | /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  Administrateur | /login | Authentification |

## Documentation des librairies

Godoc. Voir en annexe.

# Améliorations

# Conclusion

Projet très conséquent qui ne m'a pas laissé assez de temps pour soigner les détails

# Bibliographie

<https://www.quora.com/What-is-golang-good-for>

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/X.509 [↑](#footnote-ref-2)
3. ou EDI en français : Environnement de Développement Intégré [↑](#footnote-ref-3)
4. https://httpd.apache.org/docs/2.4/fr/programs/ab.html [↑](#footnote-ref-4)
5. https://github.com/tsenart/vegeta [↑](#footnote-ref-5)