|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Candidat/-e** |  | | | |
| Numéro de candidat/-e | | 122599 | | |
| Nom | | Dogan | Prénom | Onur Oktay |

Jour 1 22.05.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Lecture et Analyse du cahier des charges | 3h |
| Choix des technologies et architecture de l’application | 1h |
| Préparation, mise en page des documents | 1h |
| Planification | 2h |
| Rapport + Journal de Travail | 1h |

Choix des technologies et architecture de l’application

Les choix concernant les langages de programmation côté client et serveur ont été faciles à choisir (voir le rapport).

En ce qui concerne la base de données, je voulais utiliser *Couchbase*[[1]](#footnote-1) que je connaissais déjà. Suite à une discussion avec mon supérieur, je vais cependant utiliser *ElasticSearch*[[2]](#footnote-2), que je n’ai encore jamais utiliser, mais qui offre de bien meilleures performances pour les recherches d'articles car il s'agit d'une base de données de type *full-search[[3]](#footnote-3)*.

Jour 2 23.05.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Mockup interface graphique | 2h |
| User stories | 2h |
| Documentation des APIs | 1h |
| Diagramme UML de séquence | 2,5h |
| 1ère visite des experts | 0,5h |

Jour 3 24.05.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Installation de Elasticsearch | 1h |
| Prise en main de Elasticsearch (document Xpert) | 2h |
| Analyse du fichier JSON | 5h |

Etape 1 : contrôler le contenu du fichier

Visualiser le contenu du fichier (metadata.json) représente un premier défi car ce fichier à une taille de plus de 10 GB. J’ai bien essayé de l’ouvrir avec le programme *SublimeText* mais, après plus d’une heure, je l’ai arrêté.

Il faut trouver un moyen pour lire quelques lignes dans ce fichier et s’assurer que les données correspondent à ce qui est décrit sur le site (<http://jmcauley.ucsd.edu/data/amazon/)>.

J’ai finalement écrit un petit programme en Go pour lire les 3 premières lignes :

package main  
  
import (  
 "os"  
 "fmt"  
 "bufio"  
)  
  
func main() {  
 count := 0  
  
 file, err := os.Open("../metadata.json")  
 if err != nil {  
 fmt.Println(err)  
 }  
 defer file.Close()  
  
 scan := bufio.NewScanner(file)  
  
 for scan.Scan() {  
 fmt.Println(scan.Text())  
 count++  
  
 // Read 3 first items and stop  
 if count == 3 {  
 return  
 }  
 }  
}

Le résultat retourné est le suivant :

{**'**asin**'**: **'**0001048791**'**, 'salesRank': {'Books': 6334800}, 'imUrl': 'http://ecx.images-amazon.com/images/I/51MKP0T4DBL.jpg', 'categories': [['Books']], 'title': 'The Crucible: Performed by Stuart Pankin, Jerome Dempsey &amp; Cast'}

{'asin': '0000143561', 'categories': [['Movies & TV', 'Movies']], 'description': '3Pack DVD set - Italian Classics, Parties and Holidays.', 'title': 'Everyday Italian (with Giada de Laurentiis), Volume 1 (3 Pack): Italian Classics, Parties, Holidays', 'price': 12.99, 'salesRank': {'Movies & TV': 376041}, 'imUrl': 'http://g-ecx.images-amazon.com/images/G/01/x-site/icons/no-img-sm.\_CB192198896\_.gif', 'related': {'also\_viewed': ['B0036FO6SI', 'B000KL8ODE', '000014357X', 'B0037718RC', 'B002I5GNVU', 'B000RBU4BM'], 'buy\_after\_viewing': ['B0036FO6SI', 'B000KL8ODE', '000014357X', 'B0037718RC']}}

{'asin': '0000037214', 'related': {'also\_viewed': ['B00JO8II76', 'B00DGN4R1Q', 'B00E1YRI4C']}, 'title': 'Purple Sequin Tiny Dancer Tutu Ballet Dance Fairy Princess Costume Accessory', 'price': 6.99, 'salesRank': {'Clothing': 1233557}, 'imUrl': 'http://ecx.images-amazon.com/images/I/31mCncNuAZL.jpg', 'brand': 'Big Dreams', 'categories': [['Clothing, Shoes & Jewelry', 'Girls'], ['Clothing, Shoes & Jewelry', 'Novelty, Costumes & More', 'Costumes & Accessories', 'More Accessories', 'Kids & Baby']]}

Je récupère bien les données et constate qu’elles correspondent à la description sur le site.

Cependant, le format JSON n’est pas correct : les noms des champs et les valeurs de type string sont entourés par des apostrophes (‘) au lieu des guillemets ("). Il faudra en tenir compte dans l’implémentation.

**Remarque importante**

Lors de l’implémentation du décodage, il faudra remplacer les apostrophes par des guillemets.

**Remarque**

Dans la phase actuelle, le code me sert à analyser les données, c’est pourquoi il y aura un minimum de commentaires. Une fois l’analyse terminée, les commentaires seront ajoutés selon les consignes de mon employeur, à savoir en anglais. L’utilisation de GoLint m’indiquera également si j’en ai oublié (notamment pour la documentation).

J'utilise l'IDE Gogland de Jetbrains pour le développement Go :

* Complétion automatique du code

Etape 2 : Mapping

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

Les informations *related* sont trop détaillées. Je ne conserverai 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, il a été convenu de ne conserver que la première catégorie :

[

**[**

**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, deviendra 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 seront ignorées.

Etape 3 : Récupérer le nombre d’articles

J’adapte mon programme pour savoir combien d’articles se trouvent dans le fichier et connaître le temps nécessaire pour lire les données.

package main  
  
import (  
 "time"  
 "os"  
 "fmt"  
 "bufio"  
)  
  
func main() {  
 t0 := time.Now()  
 count := 0  
  
 file, err := os.Open("../metadata.json")  
 if err != nil {  
 fmt.Println(err)  
 }  
 defer file.Close()  
  
 scan := bufio.NewScanner(file)  
  
 for scan.Scan() {  
 count++  
 }  
  
 fmt.Println(count, "items parsed in :", time.Since(t0))  
 //245442 items parsed in : 489.786784ms  
}

Le résultat retourné est le suivant :

245442 items parsed in : 489.786784ms

Le chiffre est plutôt surprenant car il est indiqué sur le site qu’il y a 9,4 millions d’articles. Vu la taille du fichier, ce nombre semble beaucoup plus réaliste que le résultat obtenu.

En faisant quelque recherche j’ai constaté que la fonction de *Go*, **bufio.NewScanner**, avait une limite de 4096 octets par ligne. Comme certains articles possèdent une propriété *description* (non documentée sur le site) dont la taille dépasse largement cette limite, cette fonction n'est pas utilisable dans mon cas. De plus, elle a pour effet de planter le programme.

Heureusement, le package *bufio* propose une autre fonction, **bufio.NewReader(file)**.

Le code devient :

package main  
  
import (  
 "time"  
 "os"  
 "fmt"  
 "bufio"  
 "io"  
 "strings"  
)  
  
func main() {  
 t0 := time.Now()  
 count := 0  
 file, err := os.Open("../metadata.json")  
 if err != nil {  
 fmt.Println(err)  
 }  
 defer file.Close()  
  
 **r := bufio.NewReader(file)**  
  
 // Read first line  
 **s, err := r.ReadString('\n')**  
  
 **for err == nil {  
 count++**

**// Read next line  
 s, err = r.ReadString('\n')  
 }**  
  
 // Check Error -> only EOF is a normal error  
 **if err != io.EOF {  
 fmt.Println(err)  
 return  
 }**

fmt.Println(count, "items parsed in :", time.Since(t0))  
 // 9430088 items parsed in : 13.6s  
}

Le résultat retourné est le suivant :

9430088 items parsed in : 13.6s

Cette fois-ci, le compte est bon.

Etape 4 : Parsage des articles

L’étape suivante consiste à désérialiser les données brutes afin d'obtenir des structure Go. L'utilisation de la fonction *json.Unmarshal* ne fonctionne que si le format json est correct, c'est pourquoi je corrige au préalable la ligne en remplaçant les apostrophes par des guillemets et compte les erreurs s'il y en a.

package main  
  
import (  
 "bufio"  
 "bytes"  
 "encoding/json"  
 "fmt"  
 "io"  
 "os"  
 "time"  
)  
  
/\* Original Data Structure \*/  
  
type RELATED struct {  
 Also\_bought []string `json:"also\_bought"`  
}  
  
type ITEMREAD struct {  
 Asin string `json:"asin"`  
 Title string `json:"title"`  
 Price float64 `json:"price"`  
 ImUrl string `json:"imUrl"`  
 Related RELATED `json:"related"`  
 Brand string `json:"brand"`  
 Categories [][]string `json:"categories"`  
}  
  
// ---------------------------------  
  
func main() {  
 old := []byte("'")  
 new := []byte(`"`)  
 t0 := time.Now()  
 count := 0  
 countErrDec := 0  
  
 file, err := os.Open("../metadata.json")  
 if err != nil {  
 fmt.Println(err)  
 }  
 defer file.Close()  
  
 r := bufio.NewReader(file)  
  
 // Read first line  
 itemRaw, err := r.ReadBytes('\n')  
  
 for err == nil {  
 var item ITEMREAD  
  
 count++  
  
 // Correction (the json file is wrong formatted: ' instead of ")  
 itemRaw = bytes.Replace(itemRaw, old, new, -1)  
  
 // Decode json to original struct  
 errDec := json.Unmarshal(itemRaw, &item)  
 if errDec != nil {  
 countErrDec++  
 }  
  
 // Read next line  
 itemRaw, err = r.ReadBytes('\n')  
 }  
  
 // Check Error -> only EOF is a normal error  
 if err != io.EOF {  
 fmt.Println(err)  
 return  
 }  
  
 fmt.Println(count, "items parsed in :", time.Since(t0))  
 fmt.Println(countErrDec, "errors detected during decoding")  
}

Le résultat retourné est le suivant :

9430088 items parsed in : 2m31.373297971s

3401453 errors detected during decoding

Est-il possible que plus d’un tiers des données contiennent des erreurs ?

**Remarque**

Lors de l’implémentation définitive, je traiterai le décodage des articles et leur sauvegarde dans la base de données en parallélisant les tâches.

Etape 5 : contrôle du formatage des données

**Méthode**

* Remplacement des apostrophes par des guillemets
* Désérialisation : en cas d'erreur, affichage des lignes brutes et modifiées et arrêt du programme
* Copie de la ligne modifiée dans le fichier *test.json* et analyse des erreurs dans IntelliJ

…

var (

old = []byte("'")

new = []byte(`"`)

)

…

// The received file is wrong formatted (' instead of " in the json format)

**itemModif := bytes.Replace(itemRaw, old, new, -1)**

// Decode json to original struct

**errDec = json.Unmarshal(itemRaw, &item)**

**if errDec != nil {**

**fmt.Println(string(itemRaw))**

**fmt.Println(string(itemModif))**

**os.Exit(0)**

**}**

Première erreur rencontrée :

{"asin": "0000589012", "title": "Why Don**"**t They Just Quit? DVD Roundtable Discussion: What Families and Friends need to Know About Addiction and Recovery", "price": 15.95, "imUrl": "http://ecx.images-amazon.com/images/I/519%2B1kseM3L.\_SY300\_.jpg", "related": {"also\_bought": ["B000Z3N1HQ", "0578045427", "B007VI5AQ8", "B003AC98V2", "B004V4RW8O", "B000I0QL7I", "B000J10F8C", "B0007CEXYK", "B000ERVK4Y", "B000XSKDBA", "B002UNMWTC", "B00008MTXI", "B007TSV4GK", "B0052ADP6Y", "B00EUENWIY", "B003YKYX9M", "B004RD3YFE", "B007Y9F6RW", "B00004UEDQ", "B0039Y774Q", "B0006IIKRG", "B00JAGF9HE", "6305162026", "6305692572", "B001D7T460", "B0018QOIWG", "B002Y7ZELW", "B0045HCJ08", "0830907394", "B000LAZDPG", "B00A2H9QN8", "B001O5CLXY", "B000JBXXYK", "B003B3NGS6", "B0037SR3N4", "B00641Y2ZS", "0470903953", "0977977315", "B00049QQHI", "B000E6ESU8", "0470402741", "061565732X", "0615763146", "B000VZPTH8", "B003JO6OPO", "B00787BTEO", "B004R1Q7YQ", "B001GG6GKK", "B0015VQAZM", "1592854869", "B000QRIL08", "B000GQLA8O", "B000MPM3TE", "0979021804", "1608823407", "159285821X", "B00005Q4CS", "B0000549B1", "6305594333", "B00AFEXRME", "B004FN25AG", "0830906363", "0470402768", "1118414756", "B009SV4O2M", "1481106694", "1572306254", "B0013MOLPO", "B00009Y3QI", "B003NMOL2U", "B001AKBI8C", "0981708803", "1572306394", "B00B9LNPA6", "B005BYBZEK", "B004D7SBMU", "B00CQMADIO", "0470405511", "B00CHEHHT4", "B000ESUWY2", "0792838068", "B00AWE09Z0", "B00E4XZZEK", "0830914870", "B00GFZLEF4", "083090459X", "1402218443", "1893007170", "1893277046", "B005CKI7H6", "B0001LQL6K", "B000067S10", "0890425558", "B00114KYC8", "1466221224", "0943158508", "B00A7ID5BG", "0671765582", "B000B8IH10", "1568381395"], "buy\_after\_viewing": ["B003AC98V2", "B007VI5AQ8", "B000ERVK4Y", "B0007CEXYK"]}, "salesRank": {"Movies & TV": 1084845}, "categories": [["Movies & TV", "Movies"]]}

Ici, *Title* contient un guillemet. Il n'est donc pas possible de simplement remplacer toutes les apostrophes par des guillemets.

De plus, en affichant les 20 premières lignes, j'ai aussi remarqué le cas suivant où les guillemets ont été remplacés par des apostrophes … sauf lorsque la valeur contient une apostrophe :

{'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']]}

Etape 6.1 : Concept du programme de correction

Me voilà contraint à corriger le fichier reçu. Je décide de ne pas corriger à la volée car le format du prochain fichier reçu pourrait être correct mais plutôt d'écrire un programme qui lit les articles dans le premier fichier, les corrige et écrit le résultat dans un second fichier. Le fichier généré contiendra les mêmes informations que le fichier original.



**Méthode**

* 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

Jour 4 26.05.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Développement du parser | 6h |
| Création d'une librairie | 1h |
| Importation des données dans la base de données | 1h |

Etape 6.2 : Programme de correction

**Remarque**

Je ne vais pas optimiser cet outil, qui, je l'espère, sera à usage unique.

Code du programme de correction :

package main

import (

"bufio"

"log"

"io"

"os"

"time"

"fmt"

)

var (

counter = 0

mark1 = `'`

mark2 = `"`

backslash = `\`

t0 = time.Now()

)

/\*

JSON CORRECTOR -> GENERATE NEW FILE (amazon.json)

\*/

func main() {

s := ""

// Open file to read

fileR, err := os.Open("../../json/metadata.json")

if err != nil {

log.Fatalln(err)

}

defer fileR.Close()

// Open file for writing

fileW, err := os.Create("../../json/amazon.json")

if err != nil {

log.Fatal(err)

}

defer fileW.Close()

// Create a buffered reader from the file

r := bufio.NewReader(fileR)

// Create a buffered writer from the file

w := bufio.NewWriter(fileW)

// Read first line

s, err = r.ReadString('\n')

for err == nil {

counter++

// Write in buffer

w.WriteString(CorrectQuotes(s))

// Read next line

s, err = r.ReadString('\n')

}

// Write memory buffer to disk

w.Flush()

// Check Error -> only EOF is a normal error

if err != io.EOF {

log.Println(err)

return

}

log.Println(counter, "items parsed succesfully in :", time.Since(t0))

// 9430088 items parsed succesfully in : 41m23.702319783s

}

func CorrectQuotes(s1 string) (s2 string) {

inside := false

marker := ""

c := ""

// Display counter every 100'000 items

if (counter % 100000) == 0 {

fmt.Println(counter, "items saved in :", time.Since(t0))

}

for pos, char := range s1 {

c = string(char)

// Detect Start Marker

if !inside && (c == mark1 || c == mark2) {

inside = true

s2 += mark2

if c == mark1 {

marker = mark1

} else {

marker = mark2

}

continue

}

// Detect End Marker

if inside {

if c == marker && s1[pos-1:pos] != backslash {

inside = false

s2 += mark2

continue

}

}

// Ignore \ ' "

if inside && (c == backslash || c == mark1 || c == mark2) {

continue

}

// Transfer char

s2 += c

}

return

}

Étape 7 : Développement du parser en utilisant le fichier corrigé

Le code suivant permet de parser le fichier json et de convertir les éléments lus en objets compréhensibles pour Go.

package main

import (

"bufio"

"fmt"

"io"

"os"

"time"

"encoding/json"

)

/\* Own Data Structure \*/

type ITEM struct {

ID string `json:"id"`

Title string `json:"title"`

Price float64 `json:"price"`

ImURL string `json:"imUrl"`

Related []string `json:"related"`

Brand string `json:"brand"`

Category string `json:"category"`

}

// ---------------------------------

/\*

PARSE ALL ITEMS

\*/

func main() {

t0 := time.Now()

count := 0

file, err := os.Open("../../json/amazon.json")

if err != nil {

fmt.Println(err)

}

defer file.Close()

r := bufio.NewReader(file)

// Read first line

itemRaw, err := r.ReadBytes('\n')

if itemRaw == nil {

}

for err == nil {

count++

item := &ITEM{}

errDec \_= json.Unmarshal(itemRaw, &item)

iferrDec != nil {  
 fmt.Println(errDec)  
 }

// Read next line

itemRaw, err = r.ReadBytes('\n')

}

// Check Error -> only EOF is a normal error

if err != io.EOF {

fmt.Println(err)

return

}

fmt.Println(count, "items parsed in:", time.Since(t0))

// 9430088 items parsed in: **2m7.973722313s**

}

Étape 8 : Optimisation de la désérialisation

Je sais pour l'avoir déjà utilisée que, pour la désérialisation, la librairie *jsonParser*[[4]](#footnote-4) est plus performante que la fonction standard *json.Unmarshal*.

Installation de la bibliothèque

"go get github.com/buger/jsonparser"

Import de la bibliothèque

"github.com/buger/jsonparser"

Le code de l'étape précédente …

item := &ITEM{}

errDec \_= json.Unmarshal(itemRaw, &item)

iferrDec != nil {

fmt.Println(errDec)  
}

… est remplacé par l'appel de la nouvelle fonction de lecture :

readItem(itemRaw)

Code de la fonction *readItem* :

func readItem(raw []byte) \*ITEM {

item := &ITEM{}

// Read properties from json

item.ID, \_ = jsonparser.GetString(raw, "asin")

item.Title, \_ = jsonparser.GetString(raw, "title")

item.Price, \_ = jsonparser.GetFloat(raw, "price")

item.ImURL, \_ = jsonparser.GetString(raw, "imUrl")

item.Brand, \_ = jsonparser.GetString(raw, "brand")

jsonparser.ArrayEach(raw, func(value []byte, dataType jsonparser.ValueType, offset int, err error) {

item.Category = string(value)

}, "categories", "[0]")

jsonparser.ArrayEach(raw, func(value []byte, dataType jsonparser.ValueType, offset int, err error) {

item.Related = append(item.Related, string(value))

}, "related", "also\_viewed")

return item

}

Cette modification a permis de réduire le temps de lecture des articles de 40 secondes, soit de plus de 30%. Il faut maintenant 1 minute et 27 secondes pour lire les 9,4 millions d'articles.

Étape 9 : Optimisation grâce au parallélisme

Bien que les données soient prêtes à être enregistrées dans la base de données, je vais effectuer une dernière modification de notre code afin d’accélérer la lecture des données.

**Méthode**

En rendant notre application multi-thread, ce qui est relativement simple à faire en Go, nous devrions accélérer la vitesse d’importation des données.

package main

import (

"bufio"

"fmt"

"io"

"os"

"time"

"github.com/buger/jsonparser"

"sync"

)

/\* Own Data Structure \*/

type ITEM struct {

ID string `json:"id"`

Title string `json:"title"`

Price float64 `json:"price"`

ImURL string `json:"imUrl"`

Related []string `json:"related"`

Brand string `json:"brand"`

Categories []string `json:"categories"`

//Rank int `json:"rank"`

}

// ---------------------------------

var (

**concurrency = 8**

**sem = make(chan bool, concurrency)**

**wg sync.WaitGroup**

)

/\*

PARSE ALL ITEMS JSONPARSER IN PARALLEL

\*/

func main() {

t0 := time.Now()

count := 0

file, err := os.Open("../../json/amazon.json")

if err != nil {

fmt.Println(err)

}

defer file.Close()

r := bufio.NewReader(file)

// Read first line

itemRaw, err := r.ReadBytes('\n')

for err == nil {

count++

// Wait as long the channel is full and then send a new message

**sem <- true**

// Increment the waiting group

**wg.Add(1)**

// Import next item

**go** importItem(itemRaw)

// Read next line

itemRaw, err = r.ReadBytes('\n')

}

// Wait until all processes are done

**wg.Wait()**

// Check Error -> only EOF is a normal error

if err != io.EOF {

fmt.Println(err)

return

}

fmt.Println(count, "items parsed in :", time.Since(t0))

// 9430088 items parsed in : 41.37453216s

}

func importItem(itemRaw []byte) {

**defer wg.Done()**

item := readItem(itemRaw)

if item == nil {

}

// Read in semaphore channel to free the process

**<-sem**

}

func readItem(raw []byte) \*ITEM {

item := &ITEM{}

// Read properties from json

item.ID, \_ = jsonparser.GetString(raw, "asin")

item.Title, \_ = jsonparser.GetString(raw, "title")

item.Price, \_ = jsonparser.GetFloat(raw, "price")

item.ImURL, \_ = jsonparser.GetString(raw, "imUrl")

item.Brand, \_ = jsonparser.GetString(raw, "brand")

jsonparser.ArrayEach(raw, func(value []byte, dataType jsonparser.ValueType, offset int, err error) {

item.Categories = append(item.Categories, string(value))

}, "categories", "[0]")

jsonparser.ArrayEach(raw, func(value []byte, dataType jsonparser.ValueType, offset int, err error) {

item.Related = append(item.Related, string(value))

}, "related", "also\_viewed")

return item

}

En moins de 10 lignes, nous avons rendu l'application multi-thread et réduit le temps de lecture des articles de plus de 50%. La lecture des 9,4 millions d'articles s'effectue à présent en 41 secondes.

Étape 10 : Sauvegarde des données dans la base de données

L'enregistrement des données dans la base de données est effectué en ajoutant la ligne *SaveItem* après la lecture de l'article :

func importItem(itemRaw []byte, db \*DATABASE) {

defer wg.Done()

item := readItem(itemRaw)

**// Save in database**

**err := db.accesser.SaveItem(item)**

if err == nil {

fmt.Println(err)

}

// Read in semaphore channel to free the process

<-sem

}

J'ai utilisé ici le pattern en cours chez Xpert afin de faire abstraction du type de base de données dans le code principal.

Tout d'abord, j'ai créé une interface qui contient la liste des méthodes à implémenter (en l'occurrence, il n'y en a qu'une) :

**type DATABASE struct {**

// List of Methods to be implemented (mysql, elasticsearch, couchbase, ...)

**accesser interface {**

**SaveItem(\*ITEM) error**

**}**

**}**

Implémentation de la méthode et des autres fonctions utiles :

package main

import (

"errors"

"fmt"

elastic "gopkg.in/olivere/elastic.v5"

"golang.org/x/net/context"

)

type ELASTICSEARCH struct {

client \*elastic.Client

\_indexName string

\_type string

}

// NewElasticSearch open a connection to the ElasticSearch database

func NewElasticSearch(indexName, typeName string) (es \*ELASTICSEARCH, err error) {

if indexName == "" {

err = errors.New("Index Name is mandatory")

return

}

if typeName == "" {

err = errors.New("Type is mandatory")

return

}

es = new(ELASTICSEARCH)

es.\_indexName = indexName

es.\_type = typeName

es.client, err = elastic.NewClient()

if err != nil {

return

}

err = es.createIndexIfNotExist()

return

}

func (es \*ELASTICSEARCH) createIndexIfNotExist() error {

exists, err := es.client.IndexExists(es.\_indexName).Do(context.TODO())

if err != nil {

return err

}

if exists {

return nil

}

createIndex, err := es.client.CreateIndex(es.\_indexName).Do(context.TODO())

if err != nil {

return err

}

if !createIndex.Acknowledged {

err = errors.New(fmt.Sprintf("expected IndicesCreateResult.Acknowledged %v; got %v", true, createIndex.Acknowledged))

}

return err

}

// SaveItem save an item in ElasticSearch

**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**

**}**

En Go, une interface est implémentée implicitement. Il n'est pas nécessaire de la préciser avec *implements* comme en Java.

L'accès à la base de données s'effectue comme suit dans la partie principale du programme :

// Connect to ElasticSearch  
**es, err := NewElasticSearch("amazonreader", "item")**  
if err != nil {  
 log.Fatalln("ElasticSearch connection error:", err.Error())  
}  
  
// Set Database  
database = &DATABASE{accesser: es}

En cas d'utilisation d'une autre base de données, il suffira de changer uniquement la première ligne.

Étape 11 : Création d'une librairie pour l'import

Selon le cahier des charges, l'importation des données doit pouvoir s'effectuer aussi bien avec un petit utilitaire en ligne de commande que depuis l'application web.

Pour que mon code puisse être utilisé deux fois, je vais le transformer en librairie en remplaçant *package main* par :

package OnurTPIjsonImporter

Finalement, je remplace la fonction *main* par *ImportJSON* (avec I majuscule pour la rendre publique) et ajoute un paramètre pour indiquer le nom complet du fichier json à importer :

func ImportJSON(jsonFileName string) error {

Étape 12 : Création d'un utilitaire en ligne de commande

Le code du programme en ligne de commande se limite à lire le nom du fichier donné par l'utilisateur (ou d'utiliser *amazon.json* s'il est omis) et de lancer la procédure d'importation présente dans ma nouvelle librairie:

package main

import (

"flag"

"fmt"

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

)

func main() {

// Read filename

**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

Sur Linux ou OS X, la commande doit être précédée des caractères *./* :

./AmzR-import-cli -h

Sur Windows, l'extension *exe* est optionnelle :

AmzR-import-cli.exe -h

Résultat :

Usage of ./AmzR-import-cli:

-file string

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

Sans option, le fichier *amazon.json* se trouvant dans le même répertoire que l'application est importé :

./AmzR-import-cli

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

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

Jour 5 29.05.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Préparation du serveur et des routes | 1h |
| Connexion à la base de données | 2h |
| Création API pour l'ajout d'utilisateur | 1h |
| Création API pour la récupération d'un utilisateur | 1h |
| Création API pour modifier le rôle d'un utilisateur | 1h |
| Création API pour supprimer un utilisateur | ½h |
| Création API pour récupérer tous les utilisateurs | ½h |
| Création API pour accès à l'application (login) | 1h |

A partir d'aujourd'hui, je m'attaque au cœur de mon application, c'est-à-dire au serveur *Restful* qui offre les *APIs* nécessaires à l'accès des données :

* Utilisateurs
* Articles

La partie sécurité sera implémentée ultérieurement.

Etape 1 : Création d'un server http

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

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

if err != nil {

fmt.Println(err.Error())

}

Etape 2 : Préparation des routes

func routes() {  
 // Here we are instantiating the gorilla/mux router  
 r := mux.NewRouter()

// get index.http  
 **r.HandleFunc("/", startPage).Methods("GET")**  
  
 // CRUD user ---->

// save user and password in db  
 **r.HandleFunc("/user", addUser).Methods("POST")**

// get users in db  
 **r.HandleFunc("/users", getUsers).Methods("GET")**

// get user in db  
 **r.HandleFunc("/user/{id}", getUser).Methods("GET")**

// get users in db  
 **r.HandleFunc("/users", getUsers).Methods("GET")**

// update rolevalue in db  
 **r.HandleFunc("/user/{id}/{role}", updateUser).Methods("PUT")**

// delete user

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

// Check login Validity  
 **r.HandleFunc("/login", checkLoginValidity).Methods("POST")**  
 // import data from json file

**r.HandleFunc("/import", Import).Methods("POST")**

// Allow to get resources (like http, js, css, png, ...) in the given path only  
 **r.PathPrefix("/").Handler(http.FileServer(http.Dir(path)))**

**http.Handle("/", r)**  
}

Etape 3 : Développement des fonctions

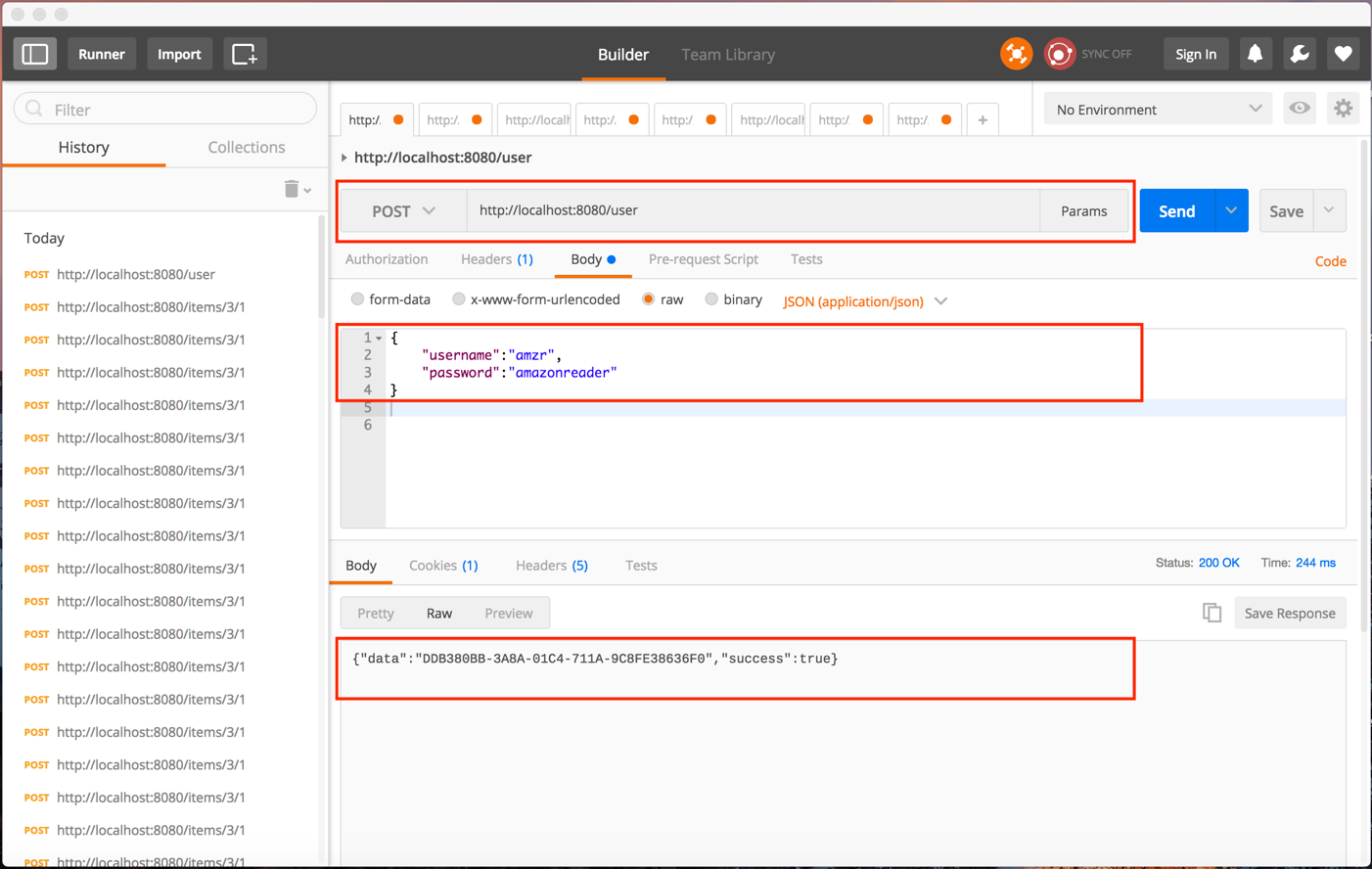
Le code des fonctions est disponible sur *github* à l'adresse :

<https://github.com/onuroktay/amazon-reader/tree/master/AmzR-Server>

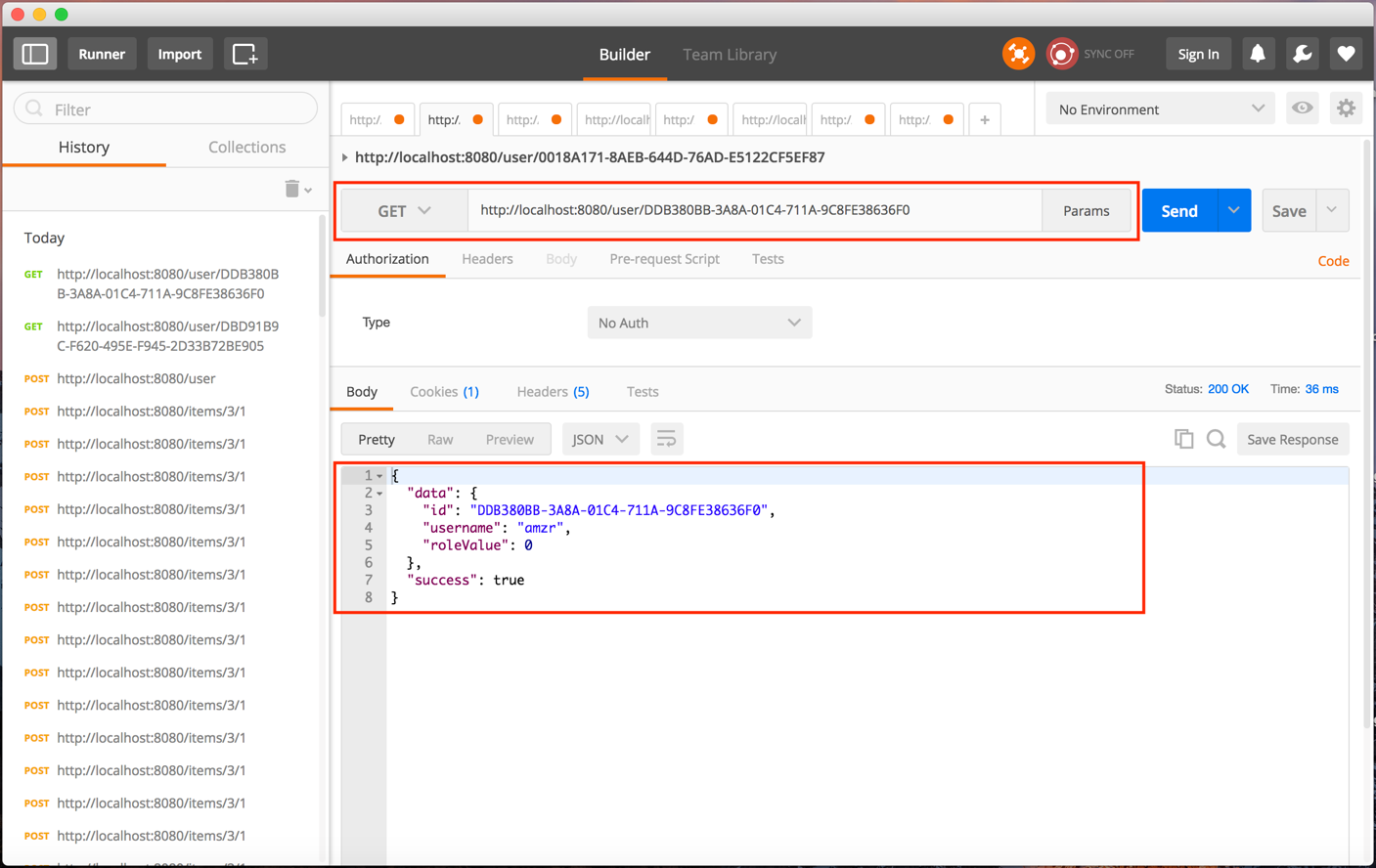
Etape 4 : Contrôle des API

Comme je n'ai pas encore de client, je vais envoyer les requêtes au server en utilisant *Postman*[[5]](#footnote-5), qui est une extension installable dans *Chrome* et utilisable comme une application.

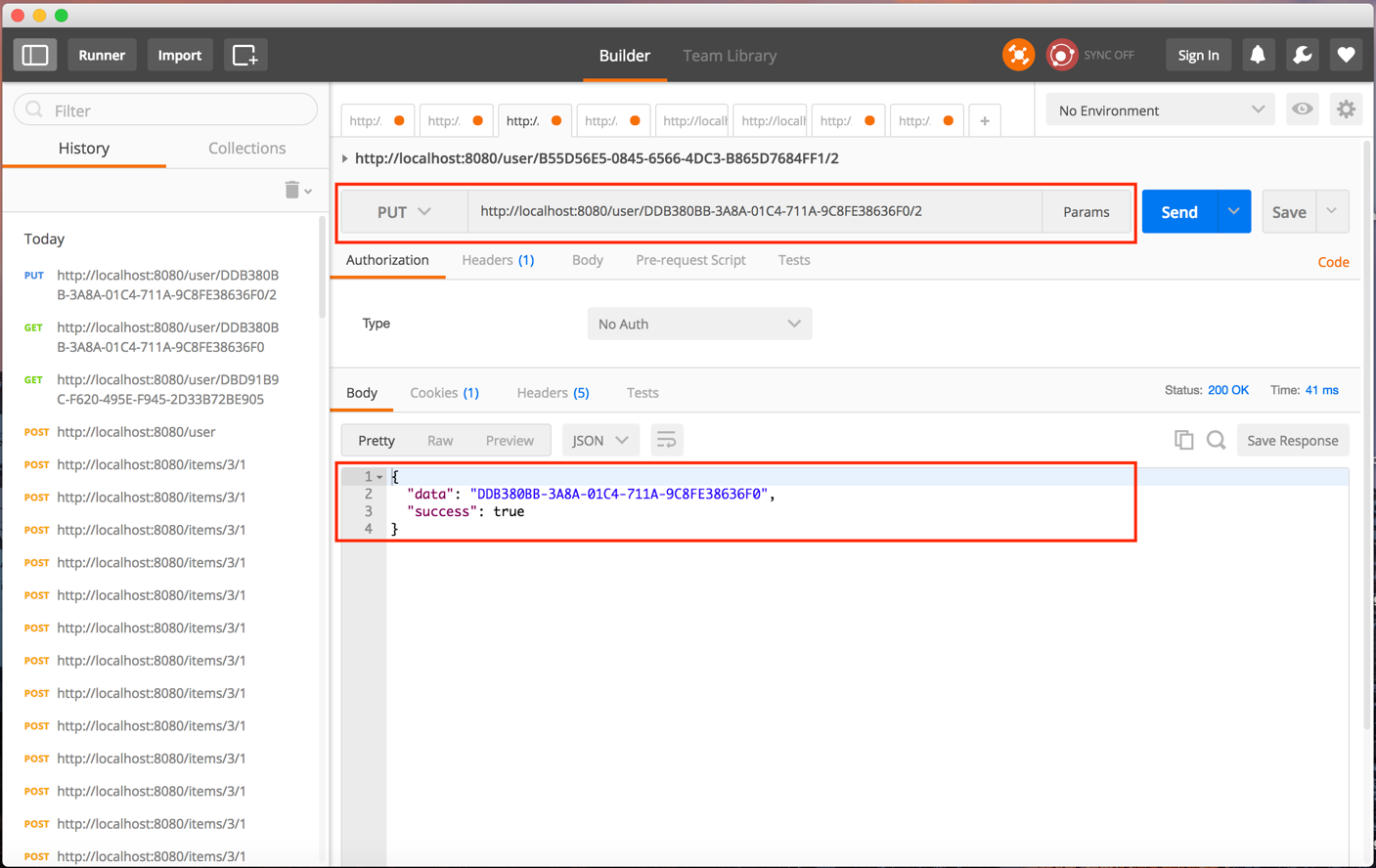
Contrôle de l'ajout d'utilisateurs :



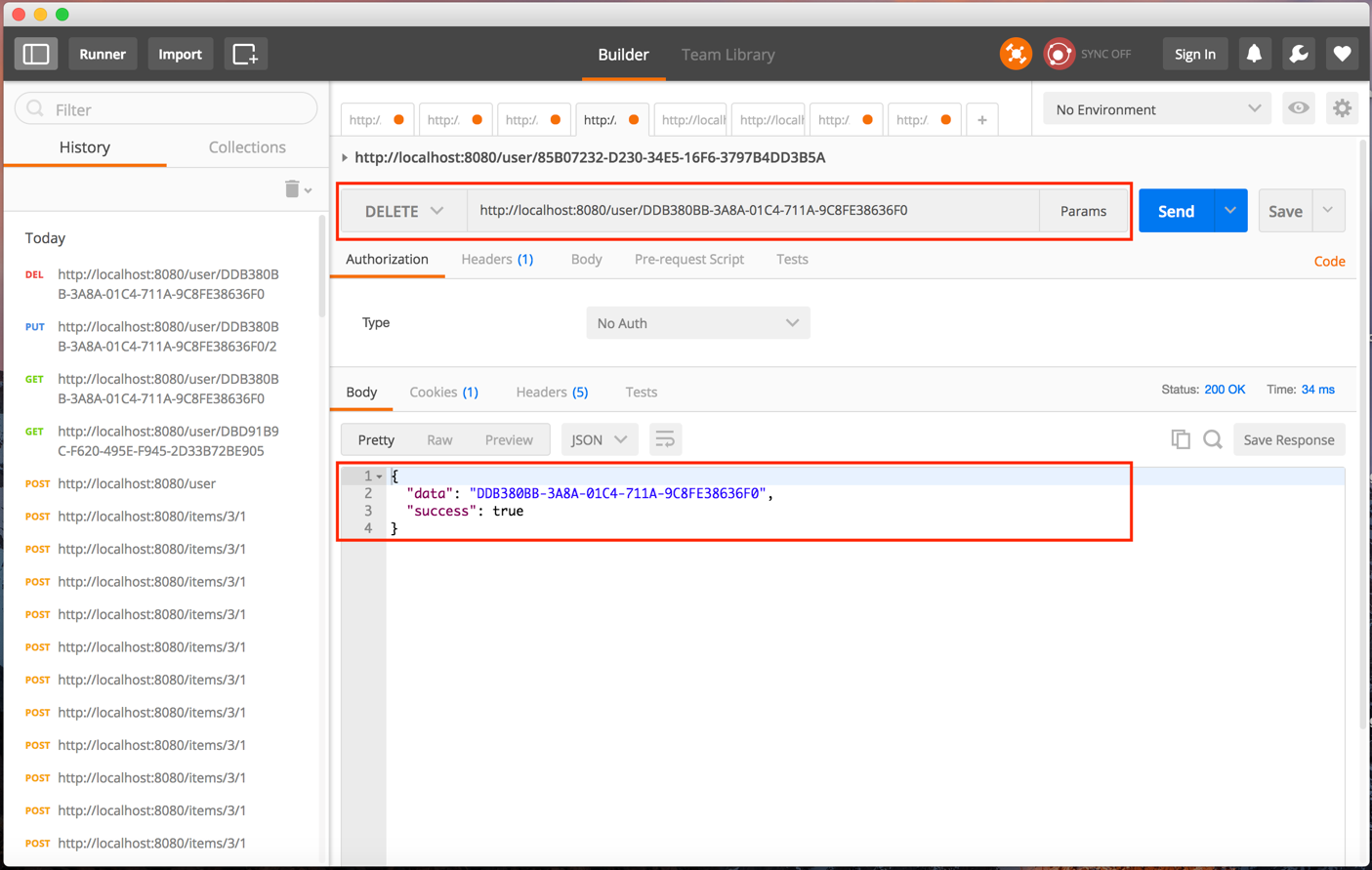
Récupération des données d'un utilisateur :



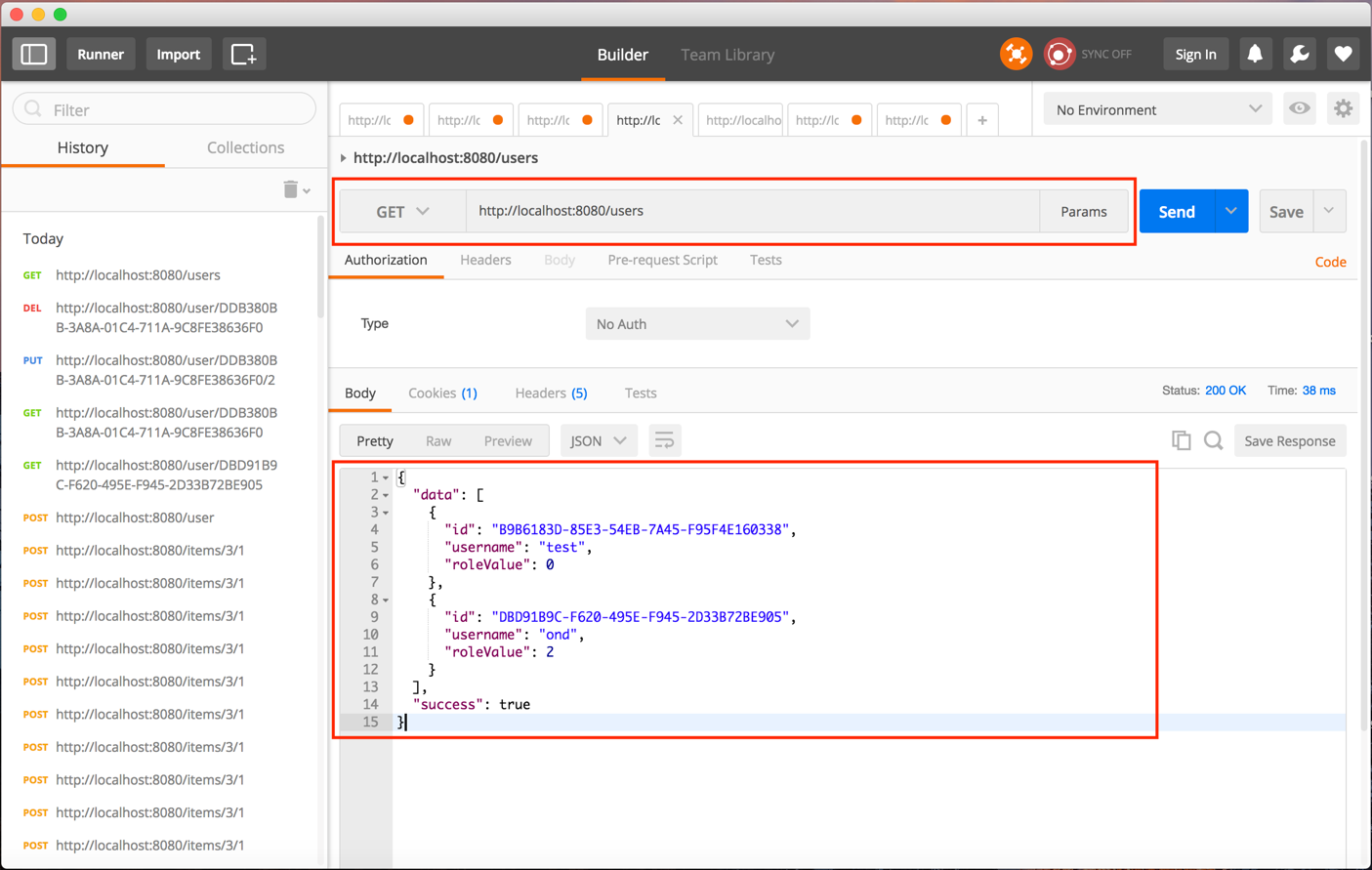
Modification du rôle d'un utilisateur :



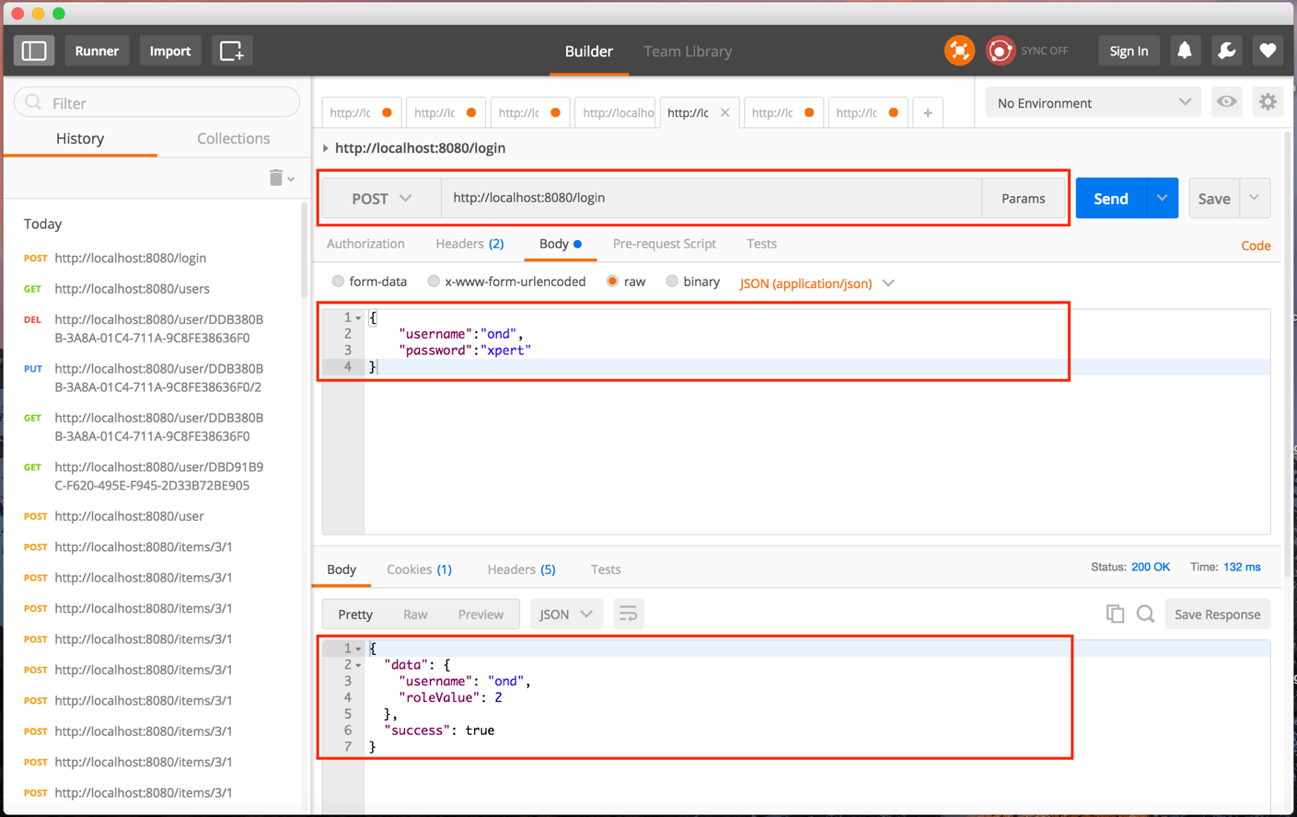
Suppression d'un utilisateur :



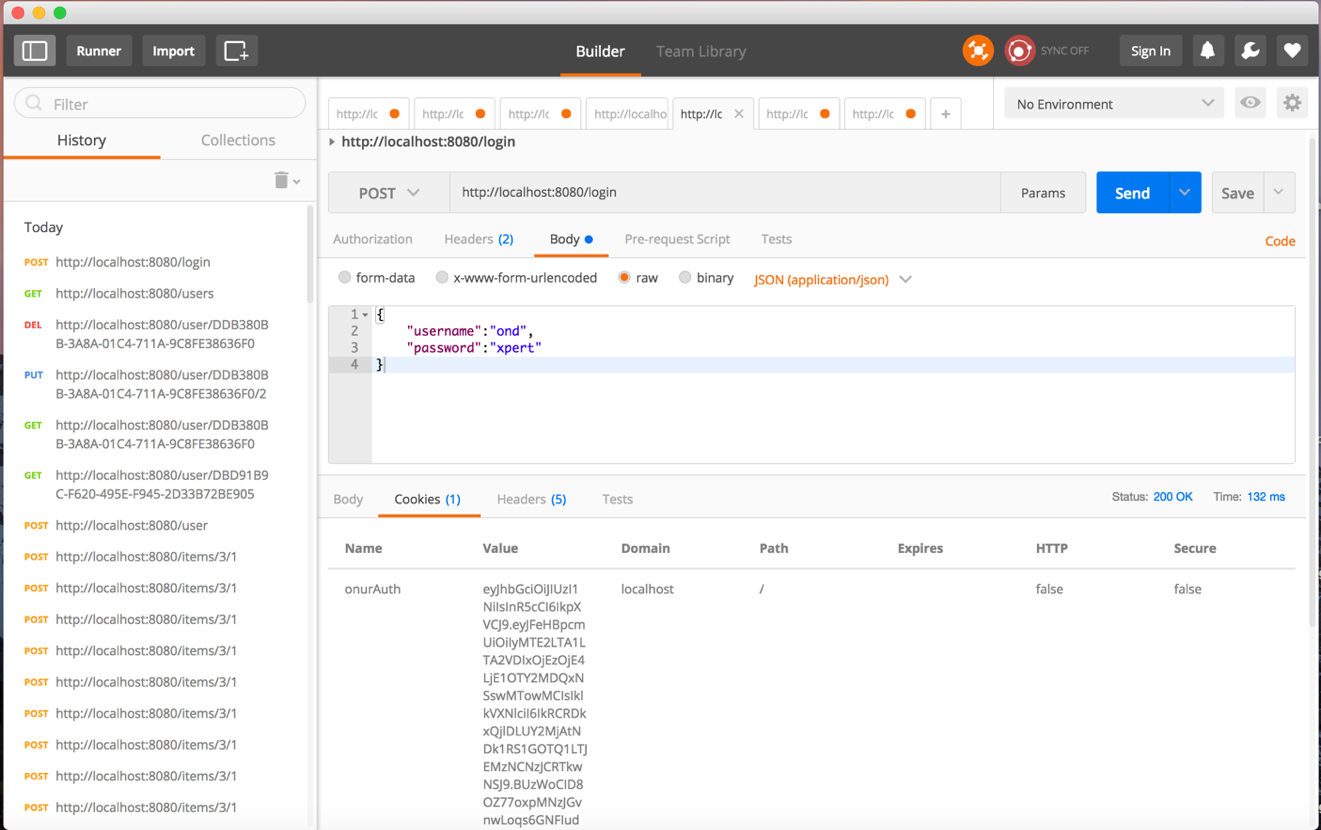
Récupération des données de tous les utilisateurs :



Contrôle du login. Si les données sont correctes, le server retourne le nom de l'utilisateur ainsi que son rôle :



Le serveur retourne également un cookie qui sera utilisé plus tard pour la sécurité. Comme je n'ai pas encore crypté la communication avec le serveur, j'ai provisoirement désactivé les options http et Secure.



Jour 6 30.05.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Création API pour récupérer un article | 1h |
| Création API pour modifier un article | 1h |
| Création API pour supprimer un article | ½h |
| Création API pour rechercher des articles | 3,5h |
| Tests des API | ½h |
| Tests unitaires | 1h |
| Réimportation des données | ½h |

Etape 5 : Préparation des routes

La fonction *routes* est complété avec les API suivantes :

// RUD item ---->

// get item in db  
 **r.HandleFunc("/item/{id}", getItem).Methods("GET")**

// get items in db  
 **r.HandleFunc("/items/{size}/{from}", getItems).Methods("POST")**

// update item

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

// delete item  
 **r.HandleFunc("/item/{id}", deleteItem).Methods("DELETE")**  
 // <----

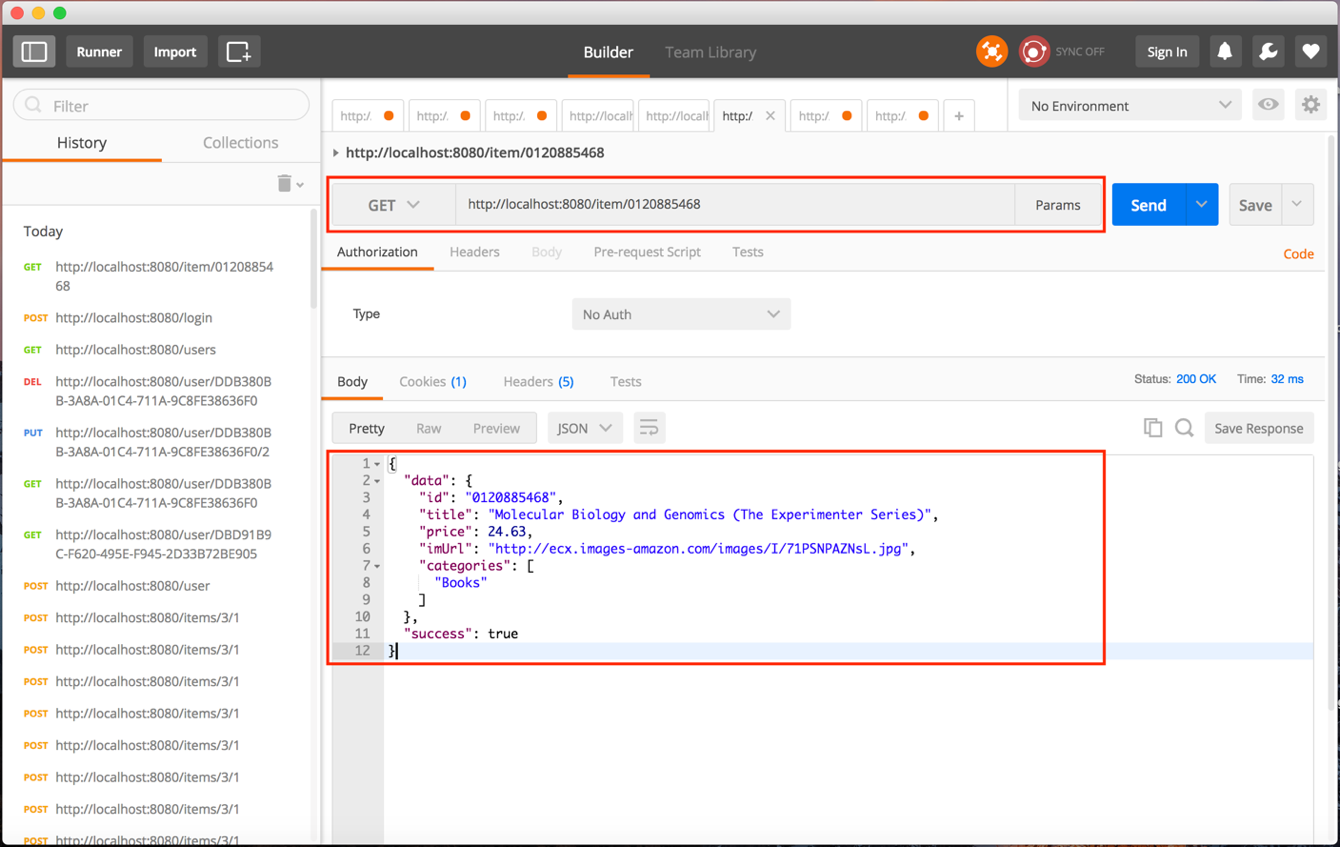
Etape 6 : Développement des fonctions

Le code des fonctions est disponible sur *github* à l'adresse :

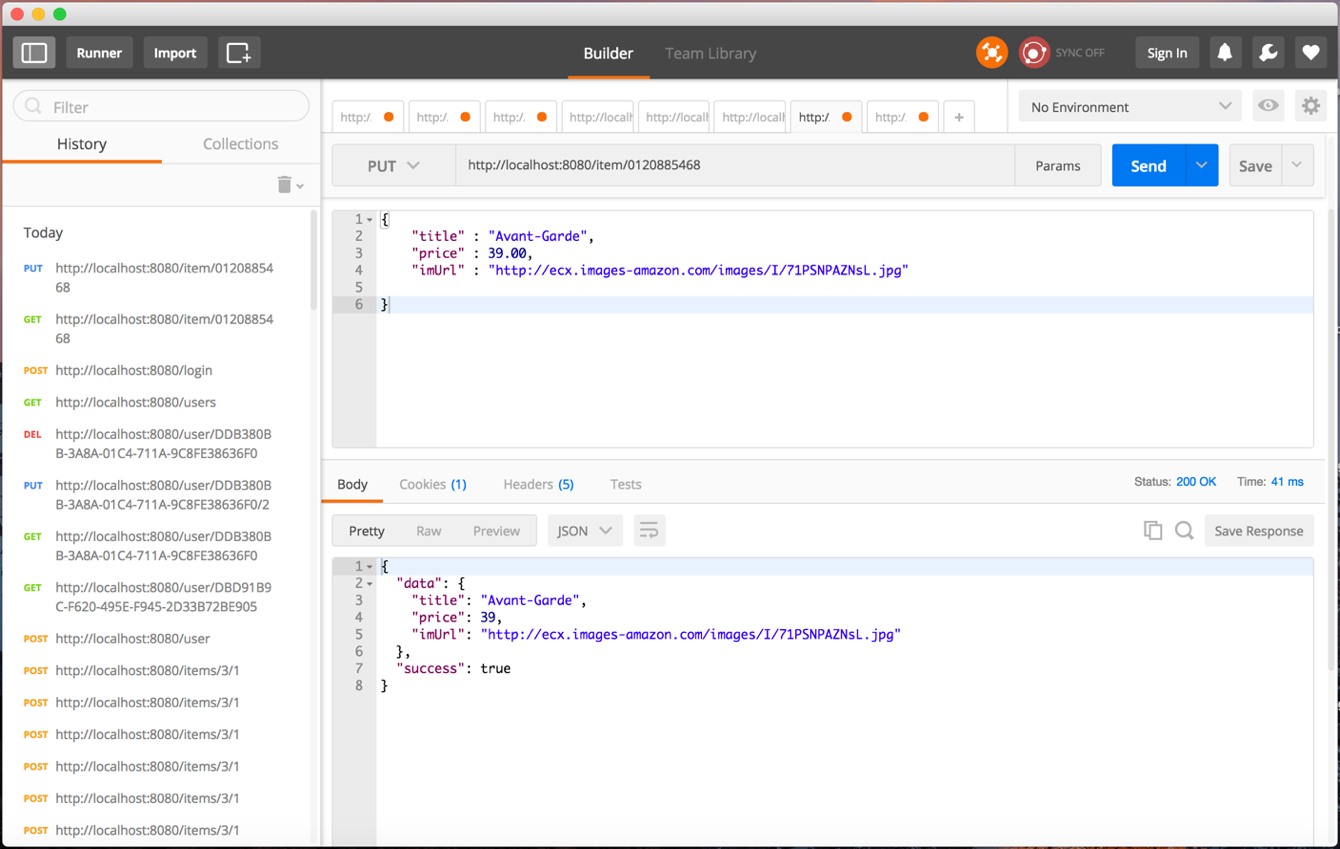
<https://github.com/onuroktay/amazon-reader/tree/master/AmzR-Server>

Etape 7 : Contrôle des API

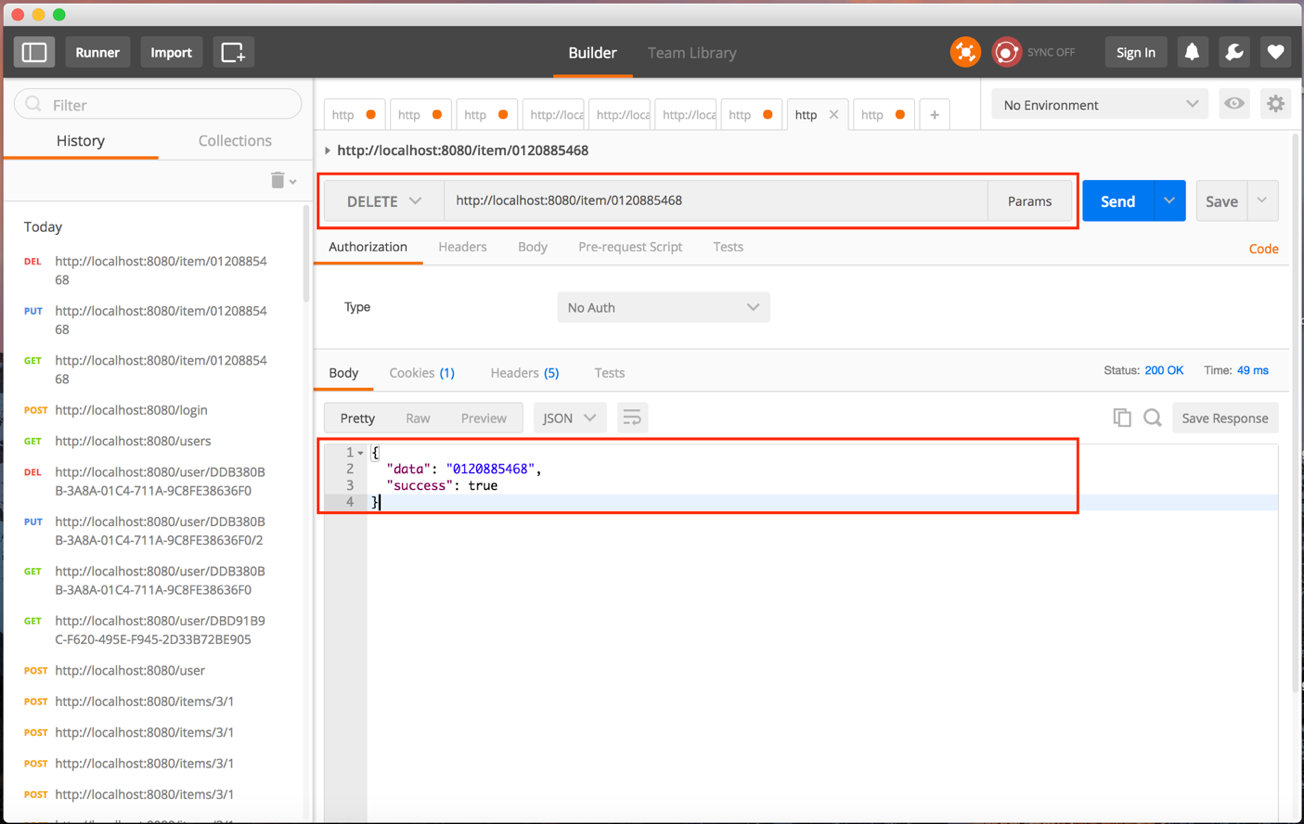
Contrôle API pour récupérer un article :



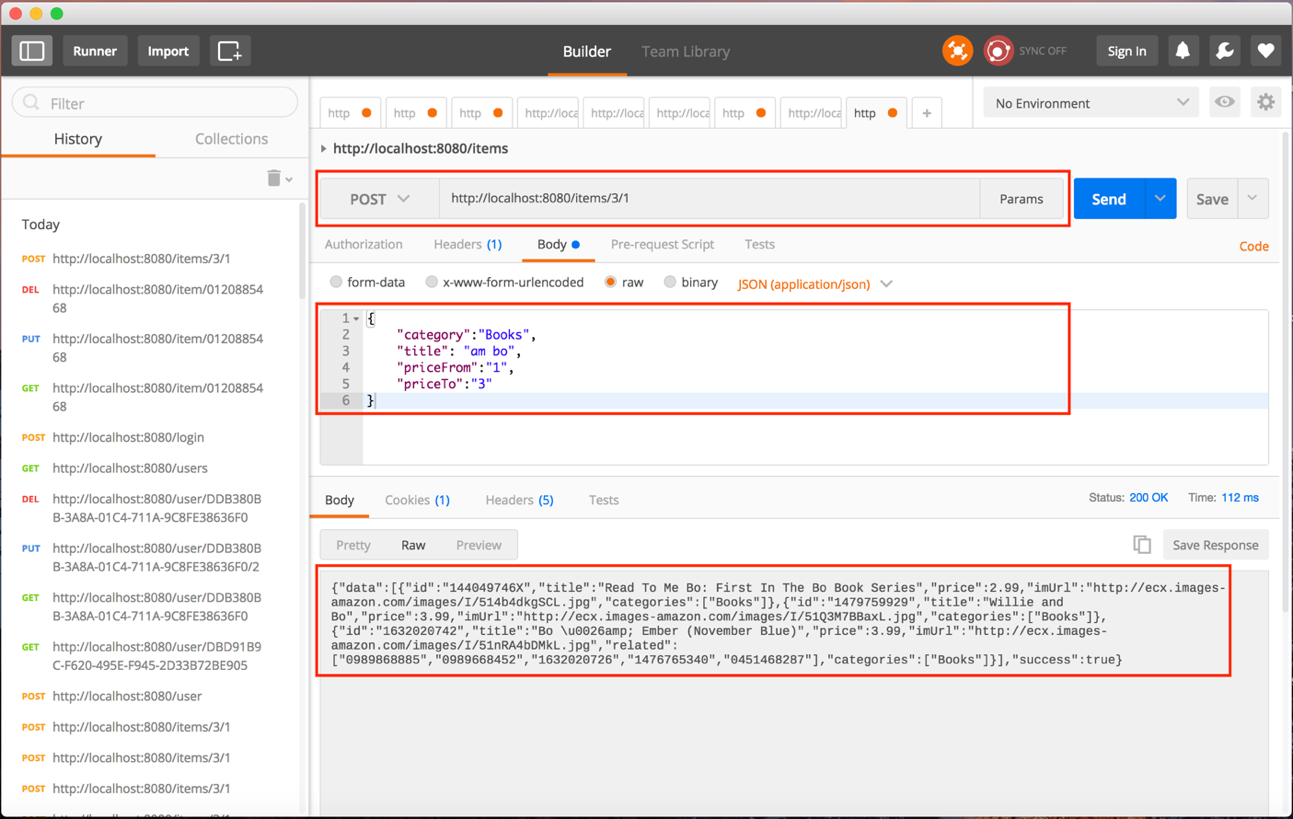
Contrôle API pour modifier un article :



Contrôle API pour supprimer un article :



Contrôle API pour rechercher un article :



Etape 8 : Tests unitaires

J’ai écrit deux tests unitaires, un qui permet de tester les fonctions des utilisateurs l’autre me permet de tester les fonctions des articles.

**Méthode TestUserFunctions**

* Création d’un account dans la base de données
* Récupérer l’account dans la base de données
* Supprimer l’account dans la base de données

**Méthode TestItemFunctions**

* Sauvegarde un article dans la base de données
* Récupérer l’article dans la base de données
* Supprimer l’item dans la base de données

Voici l’exécution du test unitaire avec la commande :

**go test**

PASS

ok github.com/onuroktay/amazon-reader/AmzR-Server 0.094s

Etape 9 : Réimportation des données

J'ai constaté un problème lors de l'affichage des articles triés par prix. L'ordre est exact sur les francs mais ne prend pas en compte les centimes.

La commande suivante de *ElasticSearch* m'a permis d'afficher le schéma généré automatiquement lors de l'import :

GET /amazonreader/\_mapping

J'ai pu constater que le champ *price* était défini comme entier long bien que l'affichage des centimes était correct :

…

"price": {

"type": "**long**"

},

…

Comme le prix de la 1ère position est 0, *ElasticSearch* en a sans doute déduit qu'il s'agissait d'un entier.

Il est malheureusement impossible de corriger le schéma des données déjà importées. Il faut donc recommencer l'opération d'import en fixant le schéma à l'avance.

* **Suppression des données dans ElasticSearch**

DELETE amazonreader

* **Créer le schéma avant l'import en fixant le type du champ *price***

PUT /amazonreader/\_mapping/item  
{  
 "item" : {  
 "properties" : {  
 "brand": {  
 "type": "text",  
 "fields": {  
 "keyword": {  
 "type": "keyword", "ignore\_above": 256  
 } }  
 },  
 "categories": {  
 "type": "text",  
 "fields": {  
 "keyword": {  
 "type": "keyword", "ignore\_above": 256  
 } }  
 }, "id": {  
 "type": "text",  
 "fields": {  
 "keyword": {  
 "type": "keyword", "ignore\_above": 256  
 } }  
 }, "imUrl": {  
 "type": "text",  
 "fields": {  
 "keyword": {  
 "type": "keyword", "ignore\_above": 256  
 } }  
 }, "price": {  
 "type": "**float**"  
 },  
 "related": {  
 "type": "text",  
 "fields": {  
 "keyword": {  
 "type": "keyword", "ignore\_above": 256  
 } }  
 }, "title": {  
 "type": "text",  
 "fields": {  
 "keyword": {  
 "type": "keyword", "ignore\_above": 256  
 }  
 }  
 }  
 }  
 }  
}

* **Relancer l’importation**

Jour 7 31.05.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Préparation du projet Angular 4 | 1h |
| Mise en place de l’architecture client | 7h |

Préparation du projet Angular 4

J'utilise l'outil Angular-cli afin de simplifier le développement :

* Contrôle de la qualité du code avec TSLint

Vérification d’Angular CLI :

ng –v

Mise en place d’architecture client :

Mise à jour de Angular CLI vers une nouvelle version :

**Global package:**

sudo npm uninstall -g @angular/cli

npm cache clean

npm install -g @angular/cli@latest

Voici la commande pour voir la version d’Angular CLI :

ng –v

Je vais créer le projet (client) avec Angular CLI :

ng new AmzR-Client --prefix amzr

cd Amzr-Client

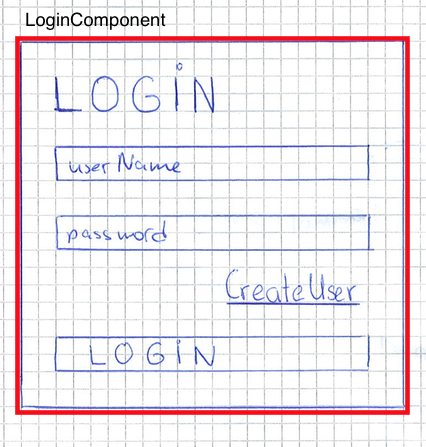
npm install

Je vais intégrer dans mon projet la bibliothèque Angular Material 2 :

npm install --save @angular/material @angular/animations

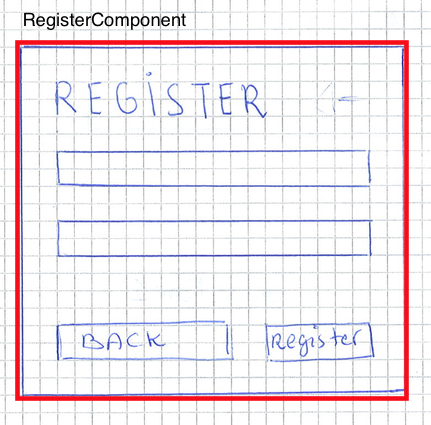
Création du LoginComponent avec l’outil d’Angular-cli avec la commande :

ng g component login



Création du RegisterComponent avec l’outil d’Angular-cli avec la commande :

ng g component register



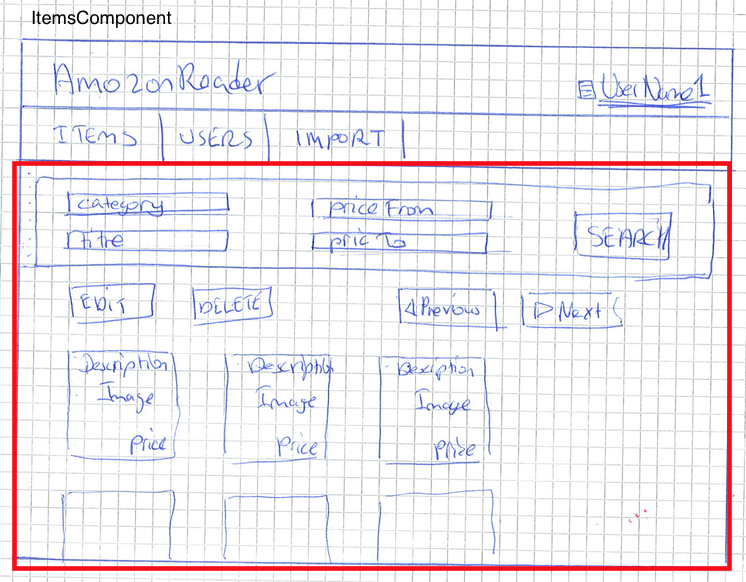
Création du NavbarComponent avec l’outil d’Angular-cli avec la commande :

ng g component navbar



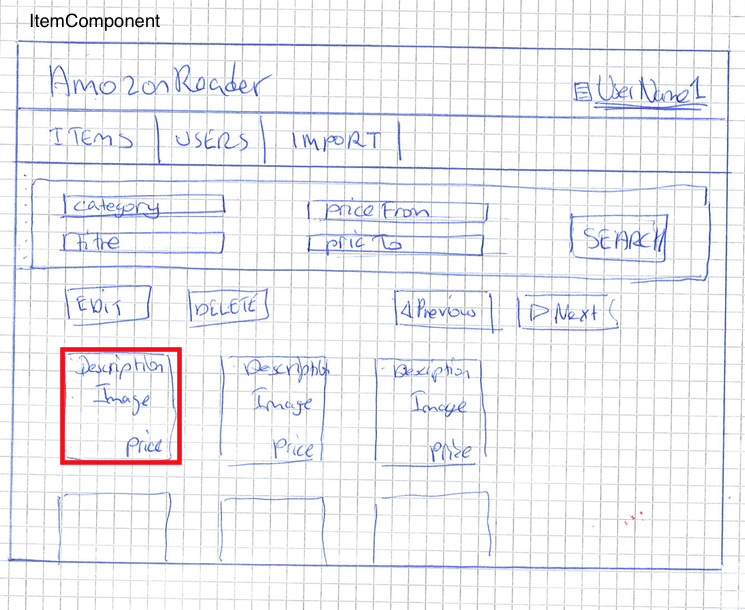
Création du ItemsComponent avec l’outil d’Angular-cli avec la commande :

ng g component items



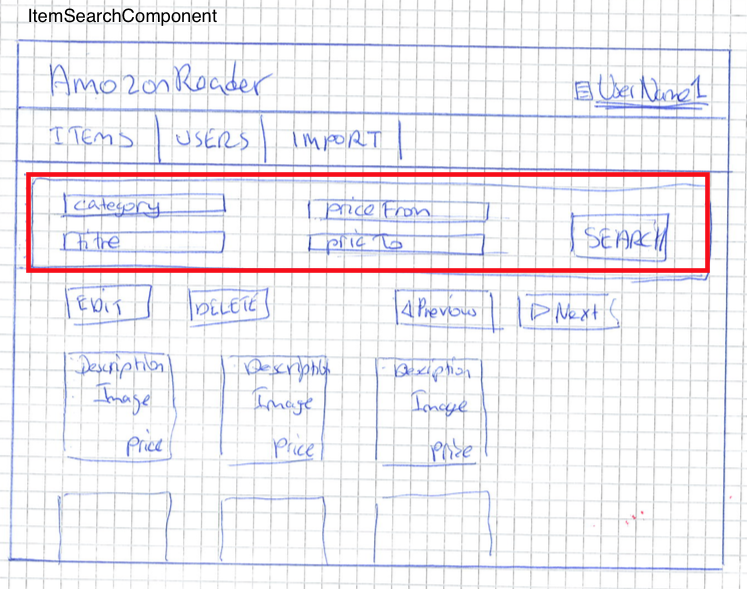
Création du ItemComponent avec l’outil d’Angular-cli avec la commande :

ng g component item



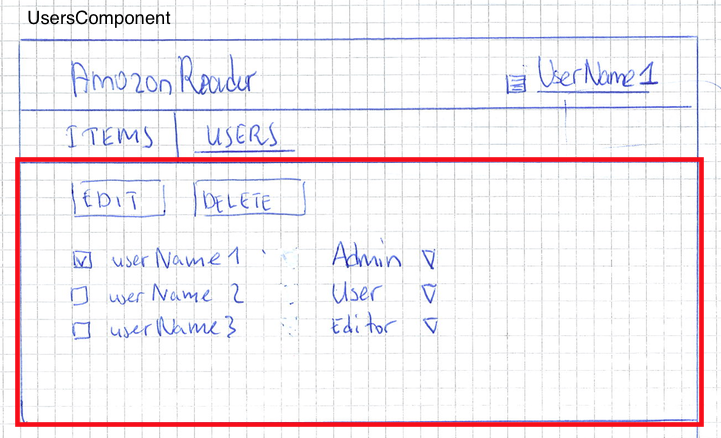
Création du ItemSearchComponent avec l’outil d’Angular-cli avec la commande :

ng g component itemSearch



Création du UsersComponent avec l’outil d’Angular-cli avec la commande :

ng g component users



Jour 8 01.06.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Création composent login | 2h |
| Création composent register | 2h |
| Gestion des articles (modification et suppression) | 4h |
| preflightRequest |  |

Jour 9 02.06.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Recherche des articles | 2h |

Jour 10 06.06.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Recherche des articles | 2h |
| 2ème visite des experts | 1h30 |

Jour 11 07.06.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Recherche des articles | 2h |
| Confirmation du suppression | 1h |
| Implémentation https | 0,5h |
| Cookie | 0,5h |
| Snackbar | 1,5h |

Jour 12 08.06.2017

|  |  |
| --- | --- |
| Travail effectué | Temps d'exécution |
| Recherche des articles | 2h |

1. <https://www.couchbase.com> [↑](#footnote-ref-1)
2. <https://www.elastic.co> [↑](#footnote-ref-2)
3. <https://fr.wikipedia.org/wiki/Recherche_plein_texte> [↑](#footnote-ref-3)
4. <https://github.com/buger/jsonparser> [↑](#footnote-ref-4)
5. <https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop> [↑](#footnote-ref-5)