

Projet Webservice Nobitsgram

AUTEUR : EYRAM DOVI

PROFESSEUR SUPERVISEUR : OLIVIER LIECHTI

Table des matières

1-	Introduction	2
2-	Intérêt d'une application RESTful	3
2.1-	REST	3
2.2-	Avantages et inconvénients de REST	4
2.2.1-	Avantages	4
2.2.2-	Inconvénient.....	5
3-	Instagram	5
4-	Procédure d'intégration de nobits.ch aux services d'Instagram.....	6
4-1	Procédure d'enregistrement de l'application sur Instagram	6
4.2-	Procédure d'authentification sur Instagram	10
4.2.1-	Récupération de l'access token.....	10
5-	Cahier de charge de l'application	12
7-	Architecture de l'application nobitsgram	14
8-	Outils utilisés	16
9-	Implémentation de l'application nobitsgram	17
9.1-	Use Case	17
9.2-	Schéma de la base de données	19
9.3 –	Modèle MVC (Modèle – Vue – Contrôleur)	20
9.3.1-	Modèle	20
9.3.2-	Vue	20
9.3.3-	Contrôleur	20
9.4 –	Diagramme UML et schéma d'interaction entre les différents modules de l'architecture MVC	21
9.4.1-	Page d'accueil.....	21
9.4.2-	Enregistrement et page d'enregistrement.....	22
9.4.3-	Page de recherche et servlet gérant la recherche	24
9.4.4-	Servlet gérant la déconnexion	25
9.4.5-	Servlet gérant la modification des données et pages JSP correspondantes.....	26
9.4.5-	Classe entité USER.....	27

1- Introduction

Ce travail de diplôme intervient dans le cadre de la formation de bachelor. Il a pour objectif l'intégration entre la plate-forme nobits.ch et les services fournis par instagram. La plate-forme nobits est développée à la HEIG-VD dans le cadre du projet européen "Nostalgia Bits". L'objectif de ce projet est d'explorer de nouvelles formes d'interactions sociales, avec une attention particulière aux besoins des personnes âgées. Le but est de permettre à cette population d'utilisateurs d'avoir plus d'interactions avec leur famille, mais également avec leurs voisins. Ainsi, elle permet et encourage le partage d'artefacts personnels, tels que des photos, des documents ou des objets digitalisés. Les interactions entre les utilisateurs et ce contenu (par le biais de commentaires, d'appréciations, etc.)

Dans le cadre de ce projet de diplôme, nous serons amenés à développer une application qui aura une architecture de trois tiers (couche présentation, couche métier/business, couche accès aux données) :

- 1.- Une application Java EE, exposant un ensemble de ressources au travers d'une API RESTful
- 2.- Une application PHP, gérant une couche d'accès graphique aux ressources
- 3.- Un module d'intégration entre le système et instagr.am, grâce au protocole OAuth.

Notre application aura pour nom Nobitsgram.

2- Intérêt d'une application RESTful

Une application RESTful est une application qui utilise les principes et concepts REST.

2.1- REST

REST (**Representational State Transfer**) est un style de programmation d'architecture distribuée orientée hypermédia. En d'autres termes, c'est un style d'architecture pour les systèmes distribués comme le World Wide Web. Ce terme REST a été proposé par Roy Thomas Fielding en 2000 lors de sa thèse (**Architectural Style and the Design of Network-based Software Architectures**) informatique présentée à l'université de Californie. Notons qu'il est à l'origine des travaux sur le protocole HTTP et sur le serveur Web Apache dont il est membre fondateur. Selon Fielding

(<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>) REST fait référence à l'image d'un système informatique web bien conçu, à partir d'un réseau de page web, permettant à l'utilisateur de naviguer dans l'application en utilisant des liens. En d'autres termes, cela veut tout simplement dire qu'un utilisateur peut lire ou modifier une ressource de l'application en utilisant une représentation de cette ressource. Ainsi, le concept essentiel de REST est la notion de ressource. Chaque information nommée d'une application peut être considérée comme une ressource : un texte brut, un document HTML/XML, une image, une collection, etc ; bref, toute information pouvant être accédée par une référence hypertexte (lien ou formulaire).

Par contre REST n'est pas un standard ni un protocole ni encore moins un format d'échange. En effet, il n'existe pas de spécification du W3C (**World Wide Web Consortium**) pour décrire l'architecture REST.

REST ne se limite pas à la réalisation d'application pour les utilisateurs que nous sommes. Il est de plus en plus utilisé dans la communication entre systèmes hétérogènes et se pose comme alternative à SOAP (même s'il est possible de combiner les deux) et au style d'architecture RPC (**Remote Procedure Call**).

REST se base sur des standards déjà existant comme :

- REST part du principe que le protocole http est largement suffisant pour l'utilisation des services web si on utilise ses méthodes GET, POST, PUT et DELETE et autre en se basant sur des ressources.
- Les URI/URL comme syntaxe universelle d'accès aux ressources.
- Les types MIME pour la représentation des ressources (text/plain, application/xml, text/xml, application/json, images ...)

- Des liens hypermédias (liens HTML) dans les documents XML, XHTML et HTML pour la représentation de la ressource et l'accès à des informations supplémentaires liées.

À partir de la définition de REST, nous comprenons qu'une application (système) RESTful est différente d'un service SOAP dans la mesure où REST repose uniquement sur l'utilisation de protocole HTTP, d'URI/URL et du langage XML, alors que SOAP utilise l'API RPC (**Remote Procedure Call**). En effet, SOAP ne suit pas les spécifications HTTP car il utilise une couche d'abstraction sur le protocole plutôt que d'utiliser de base les contraintes qu'ils imposent. SOAP est un protocole de messagerie.

Le langage informatique utilisé pour implémenter notre application est JAVA. JAVA propose une API pour l'utilisation des services REST. Cette API est JAX-RS.

2.2- Avantages et inconvénients de REST

(Source http://fr.wikipedia.org/wiki/Representational_State_Transfer)

2.2.1- Avantages

La thèse de Roy Fielding précise les avantages de ce style architectural par rapport à d'autres styles d'architectures d'applications web. Citons entre autres :

- L'application est plus simple à entretenir, car les liens sont mieux structurés, et de façon universelle ; les liens sont le moteur de l'état de l'application.
- L'absence de gestion d'état du client sur le serveur conduit à une consommation de mémoire inférieure, une plus grande simplicité et donc à une capacité plus grande de répondre à un grand nombre de requêtes simultanées.
- L'absence de gestion d'état du client sur le serveur permet une répartition des requêtes sur plusieurs serveurs : une session client n'est pas à maintenir sur un serveur en particulier (via une *sticky session* d'un *loadbalancer*), ou à propager sur tous les serveurs (avec des problématiques de fraîcheur de session). Cela permet aussi une meilleure évolutivité et tolérance aux pannes (un serveur peut être ajouté facilement pour augmenter la capacité de traitement, ou pour en remplacer un autre).
- L'utilisation du protocole HTTP en tirant partie de son enveloppe et ses entêtes, à l'opposé de SOAP qui ne présuppose pas un protocole : SOAP réinvente un protocole via une enveloppe, des entêtes et un document, à l'intérieur du protocole réseau l'hébergeant (la plupart du temps HTTP). il ne bénéficie donc pas des caractéristiques HTTP, gérée par l'infrastructure réseau (notamment les proxys supportant le cache côté serveur pour des performances plus importantes).

- L'utilisation d'URI comme représentant d'une ressource, permet la mise en place de serveurs cache.

2.2.2- Inconvénient

Les principaux inconvénients de REST sont la nécessité pour le client de conserver localement toutes les données nécessaires au bon déroulement d'une requête, ce qui induit une consommation en bande passante réseau plus grande. S'il est possible de coupler une application web REST à un service extérieur assurant la permanence des données lors de la durée d'une session, par exemple une base de données ou un cookie, on pourrait cependant considérer que l'utilisation d'un tel service pour gérer des données relatives à une session ouverte par le client serait en violation de la philosophie de REST. REST préférera l'utilisation de tableaux codés en Javascript présents dans la mémoire du navigateur client.

L'utilisation d'un formulaire HTML envoie ses données avec une méthode comme DELETE ne peut être compris par la plupart des navigateurs. Pour pallier à ce problème on émule ce comportement avec un champ caché qui transmettra le type de méthode d'envoi des données à l'application.

Par ailleurs beaucoup d'applications, bien que ne respectant pas scrupuleusement toutes les contraintes de l'architecture REST, sont largement inspirées par elle.

3-Instagram



Instagram est un réseau social grand public de masse alimenté uniquement avec des outils mobiles (tablette iPad, smartphone iPhone) via une appli iOS gratuite. Média social de nouvelle génération s'affranchissant d'un Web de bureau (ordinateur fixe ou portable). Il permet de publier, "liker" ("aimer"), commenter et partager des images (effets de filtres ou non) avec des connaissances (amis) en mode public ou privé. De plus en plus utilisé par les médias et les personnalités, Instagram séduit également le monde de l'entreprise et des institutions en termes de communication et marketing. Ce réseau social est aussi marqué

par des codes et des pratiques des utilisateurs : instameets (rencontre d'utilisateurs d'Instagram), utilisation des mots-clés par centres d'intérêts, concours, API permettant la création de produits dérivés ou d'opportunités d'affaires...

Les développeurs d'Instagram ont mis à disposition une API à destination de tous les développeurs tiers intéressés par leur service. L'intégration de la plate-forme nobits.ch aux services fournis par Instagram se fera au travers de cette API. La documentation de l'API d'Instagram se trouve sur à l'adresse <http://instagram.com/developer/>.

4- Procédure d'intégration de nobits.ch aux services d'Instagram

Avant de pouvoir intégrer toute application aux services d'Instagram, la première des choses à faire est d'enregistrer la dite application sur le site d'Instagram à l'adresse <https://instagr.am/accounts/create/?next=/developer/register/> afin de recevoir le client_ID et le client_secret de l'OAuth. En effet, Instagram utilise le protocole OAuth qui permet l'authentification à une API sécurisée d'une façon simple et standard depuis son bureau ou une application web tout en protégeant les noms d'utilisateurs et leur mot de passe.

4-1 Procédure d'enregistrement de l'application sur Instagram

La procédure pour recevoir le client_ID et le client_secret de l'application est décrite ci-dessous.

Le premier enregistrement concerne la création d'un compte client Instagram.

<https://instagr.am/accounts/create/?next=/developer/register/>

INSTAGRAM Register

Prénom:

Nom:

E-mail:

Nom d'utilisateur:

Numéro de téléphone: facultatif

Sexe: facultatif

Date de naissance: facultatif

Mot de passe:

Password confirm:

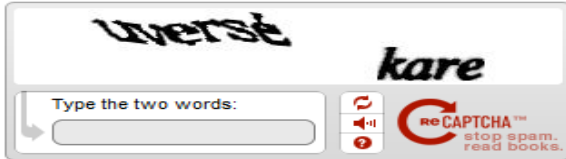
Recaptcha: 

Figure 1 Création d'un compte client Instagram

Après avoir validé cette première page d'enregistrement, nous accédons à un deuxième formulaire qui se présente comme ci :

instagr.am/developer/register/

INSTAGRAM · Developer Signup

Thanks for your interest. To get started, just a few things we need to know:

Your website:

Phone number:

Your plans:

What do you want to build with the API?

☐ I accept the API [Terms of Use](#) and [Brand Guidelines](#)

Figure 2 Formulaire d'enregistrement du développeur de l'application

Après avoir validé ce formulaire, l'utilisateur (développeur) accède à sa page client qui a cette apparence ci en sélectionnant le lien « Manage » :

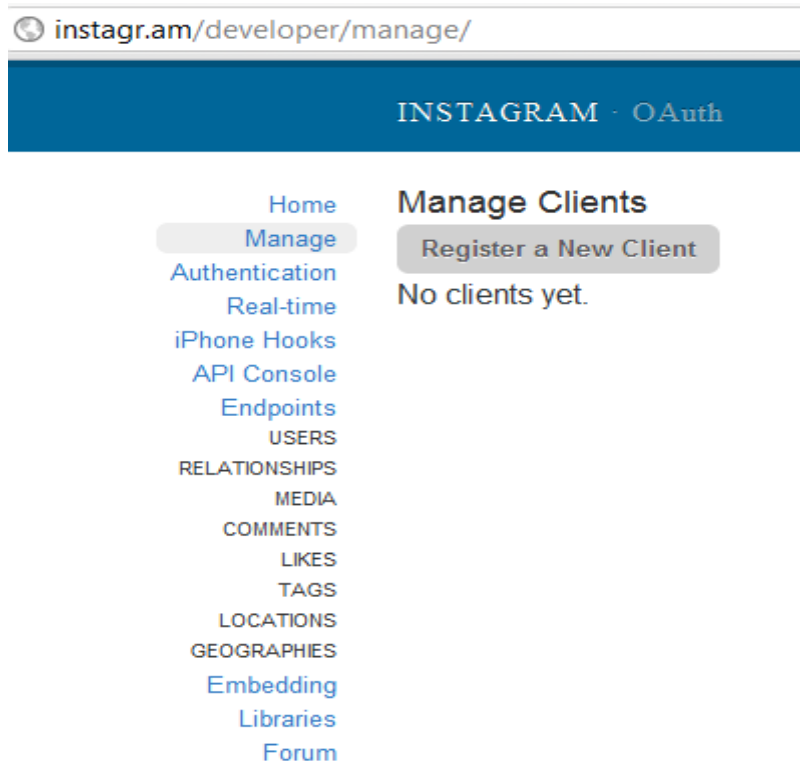


Figure 3 Page d'accueil client après enregistrement

Pour finalement recevoir son client_ID et son client_secret, l'utilisateur (développeur) doit cliquer sur le bouton « Register a New Client » pour accéder au dernier des trois (3) formulaires qui se présente comme ci :

INSTAGRAM

Register new OAuth Client

Application Name:

Description:

Website:

OAuth redirect_uri:

The redirect_uri specifies where we redirect users after they have chosen whether or not to authenticate your application.

[Register](#)

Figure 4 Formulaire d'enregistrement d'une application pour l'utilisation de l'OAuth d'Instagram

Après cette dernière étape, nous recevons finalement notre client_ID et notre client_secret. Les données clients se trouvent sur le lien « Manage » et se présente comme ci :

INSTAGRAM · OAuth

- Home
- Manage
- Authentication
- Real-time
- iPhone Hooks
- API Console
- Endpoints
- USERS
- RELATIONSHIPS
- MEDIA
- COMMENTS
- LIKES
- TAGS
- LOCATIONS
- GEOGRAPHIES
- Embedding
- Libraries
- Forum

Manage Clients

Register a New Client

YOUR CLIENTS:

Client_nobitsgr.am [Edit](#) [Delete](#)

CLIENT ID
client_ID reçu d'instagram

CLIENT SECRET
client_secret reçu d'instagram

CALLBACK URL
Adresse de redirection de notre application

WEBSITE
Adresse du site de notre application

DESCRIPTION
Description de l'application

Figure 5 Affichage des données à utiliser pour l'authentification OAuth d'instagram

4.2- Procédure d'authentification sur Instagram

L'étape suivante après la réception du client_ID et du client_secret, est l'authentification au travers du protocole OAuth de la version 2.0. Cette authentification voudrait tout simplement dire que tout client qui utilisera notre application ne confie pas son mot de passe Instagram à notre application, mais qu'il autorise notre application à se connecter sur le compte Instagram du client. Ainsi, notre application demande à l'utilisateur de s'authentifier chez Instagram et d'autoriser le site à utiliser ses données.

La procédure d'authentification sert à récupérer l'« access token » Instagram de l'utilisateur. En effet, tout utilisateur possédant un compte instagram, possède un « access token » unique. Cet access token définit chaque utilisateur. Il permet d'accéder aux données du client et si possible de modifier certaines de ses données. Notons que chaque client aura uniquement accès à son access token et qu'il ne sera pas partagé avec une tierce personne.

La récupération de l'access token est l'une des parties importantes du projet.

4.2.1- Récupération de l'access token

La récupération de l'access token nécessite deux (2) choses.

Une première chose est de rediriger l'utilisateur de notre application (celui qui ne s'est pas encore enregistré sur nobitsgram) vers l'url d'autorisation d'instagram à l'adresse suivante :

https://api.instagram.com/oauth/authorize/?client_id=CLIENT-ID&redirect_uri=REDIRECT-URI&response_type=code

- CLIENT-ID = client-id reçu par Instagram
- REDIRECT-URI = adresse de redirection de notre application
- Response_type = type de réponse que l'on souhaite recevoir d'Instagram. Nous avons 2 types de réponse, « code » et « access_token ».

La deuxième chose est la redirection du client selon le type de réponse choisie. Une réponse de type = « access_token » est qualifiée de « Implicit flow » et de type = « code » qualifiée de « Explicit flow ».

- **Implicite flow**

En choisissant ce type de réponse, Instagram ajoute directement l'access_token à la fin de notre adresse de redirection (dans notre cas à la fin de l'url de nobitsgram) et précédé du caractère « # ». Ce type de réponse est utilisé pour les applications ne possédant pas de serveur interne. Le choix de ce type de réponse ne permet pas la récupération directe de l'access token par le serveur. Dans ce cas précis, Instagram communique directement avec le « browser » du client en lui envoyant directement l'access token de l'utilisateur. Ce type de

réponse ne nous convient pas car nous voulons que notre application communique directement avec l'Instagram ce qui permettrait de récupérer l'access token. Pour ce genre de réponse, Instagram propose un autre type de réponse qui est le type « code » qualifié de « Explicit flow ».

- **Explicit flow**

L'explicit flow, permet au serveur d'Instagram de communiquer directement avec le serveur de notre application. Avec ce type de réponse, Instagram envoie un code à notre application. Ce code sera utilisé pour récupérer les données du client, données dans lequel se trouve l'access token. Le code reçu doit être échangé au près d'Instagram pour récupérer les données du client. L'échange se fait au travers de la requête POST. La requête POST doit être effectuée avec les paramètres de l'application à l'adresse https://api.instagram.com/oauth/access_token. Voici les paramètres de la requête :

client_id = CLIENT-ID (celui de l'application)

client_secret = CLIENT-SECRET (celui de l'application)

grant_type = authorization_code (cette valeur reste telle quelle)

redirect_uri = Adresse_de_redirection (l'adresse de redirection de notre application)

code = CODE (Le code reçu d'instagram)

Il est à noter que l'access token peut expirer. Si la requête est acceptée du côté d'Instagram, ce dernier renvoie une réponse dont le format ressemble à ceci :

```
{
  "access_token": "fb2e77d.47a0479900504cb3ab4a1f626d174d2d",
  "user": {
    "id": "1574083",
    "username": "snoopdogg",
    "full_name": "Snoop Dogg",
    "profile_picture":
"http://distillery.s3.amazonaws.com/profiles/profile_1574083_75sq_12
95469061.jpg"
  }
}
```

Après avoir reçu cette réponse, il suffit juste de faire un « parsing » pour extraire les données utiles de l'utilisateur.

Après la procédure d'authentification, l'utilisateur peut donc effectuer des actions sur l'application en utilisant les ressources stockées sur Instagram.

5- Cahier de charge de l'application

Comme nous l'avons souligné plus, le but de notre application est de réaliser une intégration d'une plate-forme nobits.ch et des services REST fournis par Instagram. En partant sur cette base, il a été défini les spécification suivante de notre application nobitsgram :

Fonctions de base

(US1) En tant qu'utilisateur anonyme, quand je me connecte au site, je vois la page d'accueil du site nobitsgr.am:

- 1.1. J'ai la possibilité de me logger en tapant mon user id et mon mot de passe et en cliquant sur un bouton "login"
- 1.2. J'ai la possibilité de créer un compte en cliquant sur un lien "register"
- 1.3. Je vois une photo venant de instagr.am, choisie au hasard parmi les photos taggées avec le mot "nobits"
- 1.4. La photo est mise à jour toutes les 5 secondes

(US2) En tant qu'utilisateur anonyme, j'ai la possibilité de créer un compte:

- 2.1. Je dois fournir mes informations personnelles
- 2.2. Je peux saisir mon adresse (et grâce au service de geocoding de Google, la latitude et la longitude sont automatiquement ajoutées à mon profil)
- 2.3. Je peux saisir mes centres d'intérêt en saisissant des mots clés (correspondant aux tags dans instagr.am)
- 2.4. Je dois suivre la procédure d'autorisation OAuth supportée par instagr.am (pour que l'API puisse être utilisée avec mon compte instagr.am)

(US3) En tant qu'utilisateur, après m'être connecté avec mon user id et mon mot de passe, j'arrive sur ma page personnelle; je vois plusieurs onglets (tabs):

- 3.1. Je peux cliquer sur un lien pour modifier mes données personnelles (y.c. mes centres d'intérêt).
- 3.2. Je peux cliquer sur un lien pour me déconnecter.
- 3.3. Dans un onglet ("gallery"), je vois une grille de 3x3 images; sur la première ligne, je vois 3 photos choisies au hasard parmi les photos de instagr.am taggées avec un de mes centres d'intérêt; sur la deuxième ligne, je vois 3 photos choisies au hasard parmi celles de mes followers; sur la troisième ligne, je vois 3 photos choisies au hasard parmi toutes les photos que j'ai "aimé" (like)
- 3.4. Dans un onglet ("search"), je peux taper un mot clé dans un champs de texte et cliquer sur un bouton "find"; à chaque fois que je clique, une grille de 3x3 images et mise à jour; chaque image est choisie au hasard parmi celles retrouvées dans instagr.am en fonction du mot clé saisi; je vois aussi le nombre total de photos qui existent dans instagr.am et qui satisfont au critère de recherche.

Fonctions sociales

(US4) En tant qu'utilisateur, je peux cliquer sur un onglet ("friends") dans ma page personnelle. Je vois des informations sur mes contacts/amis:

- 4.1. Je vois la liste de tous mes contacts instagr.am qui ont un compte nobitsgr.am
- 4.2. Je vois la liste des utilisateurs nobitsgr.am qui ont des centres d'intérêt en commun avec moi.

- 4.3. Je vois la liste des utilisateurs nobitsgr.am qui sont ACTUELLEMENT connectés.
- 4.4. Je vois la liste des personnes qui me suivent (mes followers) et que je ne suis pas; en face de chacune de ces personnes, j'ai les 3 dernières photos postées; je peux "liker" ces photos (en cliquant sur un bouton); je peux aussi cliquer sur un bouton pour devenir un follower de la personne en question.

Fonctions géographiques

- (US5) En tant qu'utilisateur, je peux cliquer sur un onglet ("map") dans ma page personnelle.
- 5.1. Je vois une carte géographique (google maps); je peux cliquer sur la carte, ce qui met à jour une grille de 3x3 images; les images affichées sont choisies au hasard et ont été prises dans un rayon de 1 km;
- 5.2. Je vois une autre carte géographique, où sont représentés les différents utilisateurs nobitsgr.am

Fonctions d'administration

- (US6) En tant qu'administrateur du site nobitsgr.am, je peux accéder à une console d'administration (en utilisant une URL spéciale, en donnant mon user id et mon mot de passe).
- 6.1. C'est uniquement parce que j'ai le ROLE d'administrateur que je peux y accéder (un utilisateur normal ne pourra pas y accéder).
- (US7) En tant qu'administrateur, je vois le nombre de comptes utilisateurs créés;
- (US8) En tant qu'administrateur, je peux naviguer dans une liste d'utilisateurs (avec pagination, c'est à dire avec n utilisateurs par page, boutons "p")
- (US9) En tant qu'administrateur, je peux "bloquer" un compte (l'utilisateur ne pourra pas se logger et verra un message d'erreur adéquat)
- (US10) En tant qu'administrateur, je peux savoir pour chaque utilisateur:
- quand le compte a été créé
 - combien de fois l'utilisateur s'est connecté depuis la création du compte
 - combien de fois l'utilisateur s'est connecté pendant le mois courant
 - combien de recherches ont été faites depuis la création du compte
 - combien de recherches ont été faites pendant le mois courant
- (US11) En tant qu'administrateur, je peux taper un mot clé et afficher la liste de tous les utilisateurs qui ont utilisé ce mot clé dans leurs centres d'intérêt
- (US12) En tant qu'administrateur, je peux afficher l'historique de toutes les actions effectuées par un utilisateur donné (création du compte; connexion/déconnexion; recherches; accès aux différentes listes et cartes;)

Fonctions développeur

- (US13) En tant que développeur, je peux utiliser une API REST exposée par nobitsgr.am:
- je peux récupérer la liste des actions réalisées par un utilisateur donné.
 - je peux récupérer la liste des utilisateurs, avec à chaque fois: la data de création du compte; le nombre de connexions total; le nombre de connexions dans le mois courant; le nombre de recherches faites depuis la création du compte; le nombre de recherches faites dans le mois courant.

7- Architecture de l'application nobitsgram

Notre application devra utiliser les ressources mis en service par Instagram. Nous aurons le schéma suivant pour décrire l'interaction entre nobitsgram et Instagram.

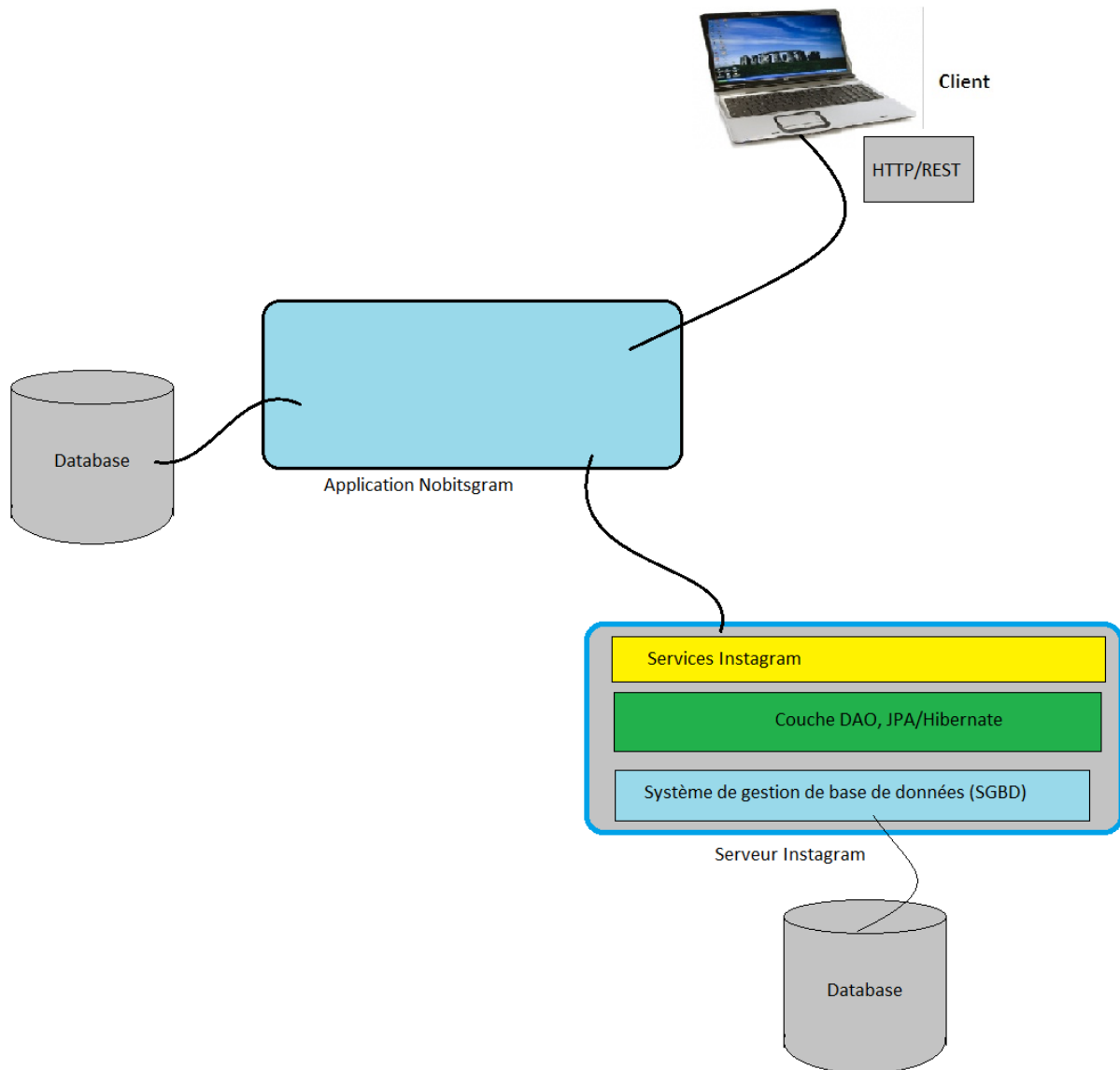


Figure 6 Schéma montrant l'interaction entre Nobitsgram et Instagram

L'architecture de notre application aura l'apparence suivante :

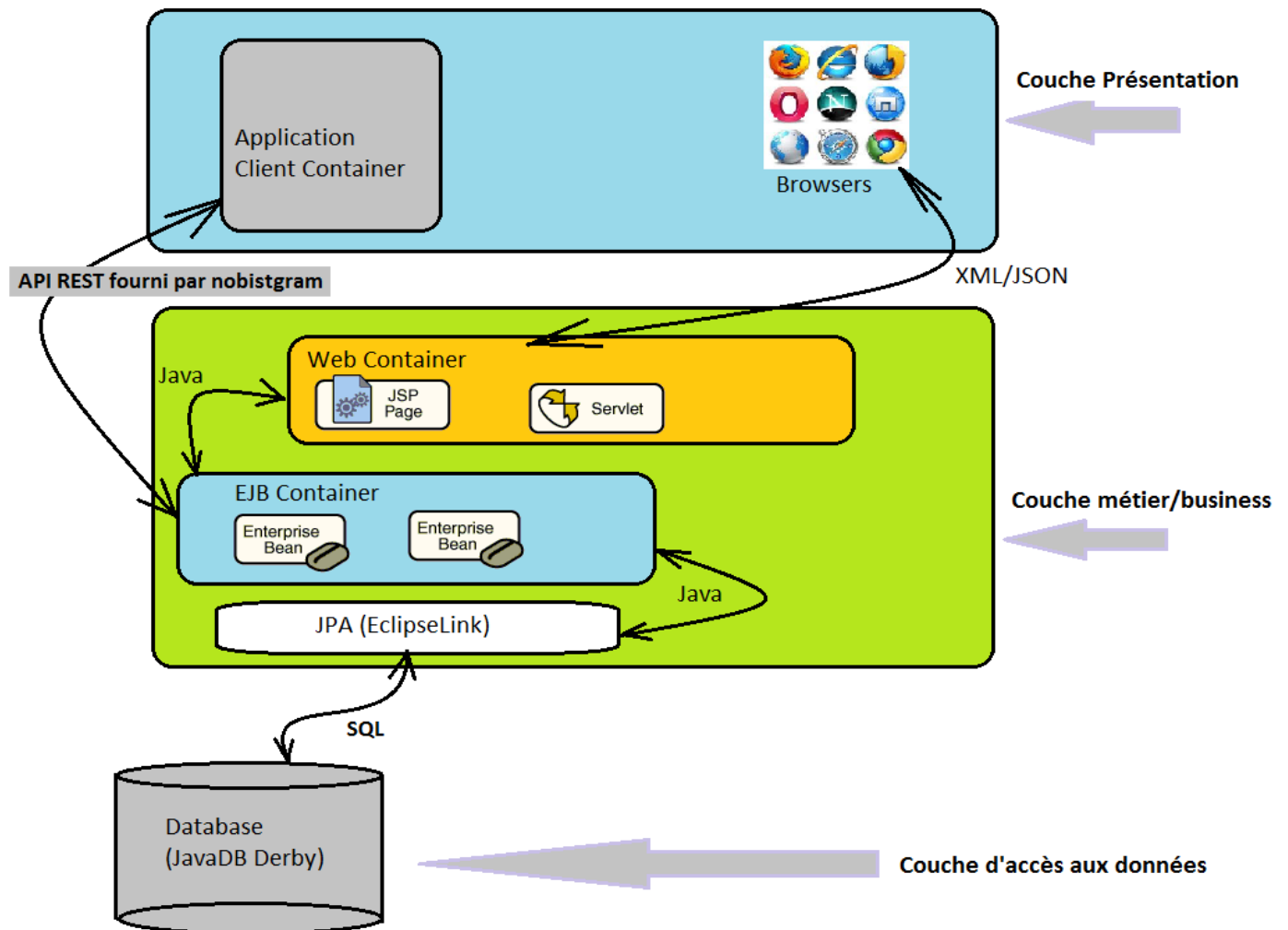


Figure 7 Architecture de l'application nobitsgram

Dans le cadre de ce projet, nous avons utilisé les outils NetBeans, Glassfish et Github.

8.1-NetBeans



NetBeans est un IDE (integrated development environment) développé par SUN Microsystems. Il supporte plusieurs environnements (JSP, EJB, Java EE, JFS, JPA, Java Servlet API, Hibernate). Il est donc l'IDE qui correspond bien à notre projet, puisque nous aurons à utiliser des servlets, des pages JSP, des EJB. Entre autre, il comprend un explorateur de bases de données qui supporte toutes les bases relationnelles pour lesquelles un connecteur JDBC existe exemple de JavaDB (Derby) MySQL, PostgreSQL, Oracle, Microsoft SQL, pour ne citer que ceux-là.

Il intègre les serveurs Glassfish et Apache. Dans le cadre de ce projet, nous utiliserons le serveur Glassfish. Une description plus approfondie suivra.



8.2 – Glassfish

Glassfish est le serveur que notre application utilisera. La version utilisée est la 3.1. Comme pour le cas netbeans et celui de Github, une ample description viendra par la suite.



8.3 – Github

Github est un service web d'hébergement et de gestion de développement de logiciels, utilisant le programme Git. Une description suivra pour compléter cette section.

9- Implémentation de l'application nobitsgram

Nous avons implémenté notre application sous le modèle MVC (Modèle – Vue – Contrôleur). Dans le cadre de ce projet, nous avons décidé de travailler avec l'IDE Netbeans. La version utilisée est le 7.0.1. Pour réaliser cette application, nous avons utilisé le langage Java (Servlet et classe java), du HTML (pages JSP) éventuellement du javascript (pages jsp) et du JPQL (Java Persistence Query Language) pour interroger la base de données.

Pour mieux cerner le travail à faire, nous avons réalisé un diagramme « Use Case » de l'application, ainsi qu'un schéma UML de la base de données. Après ces deux schémas, nous avons établi un diagramme UML des classes et de leurs interactions.

9.1- Use Case

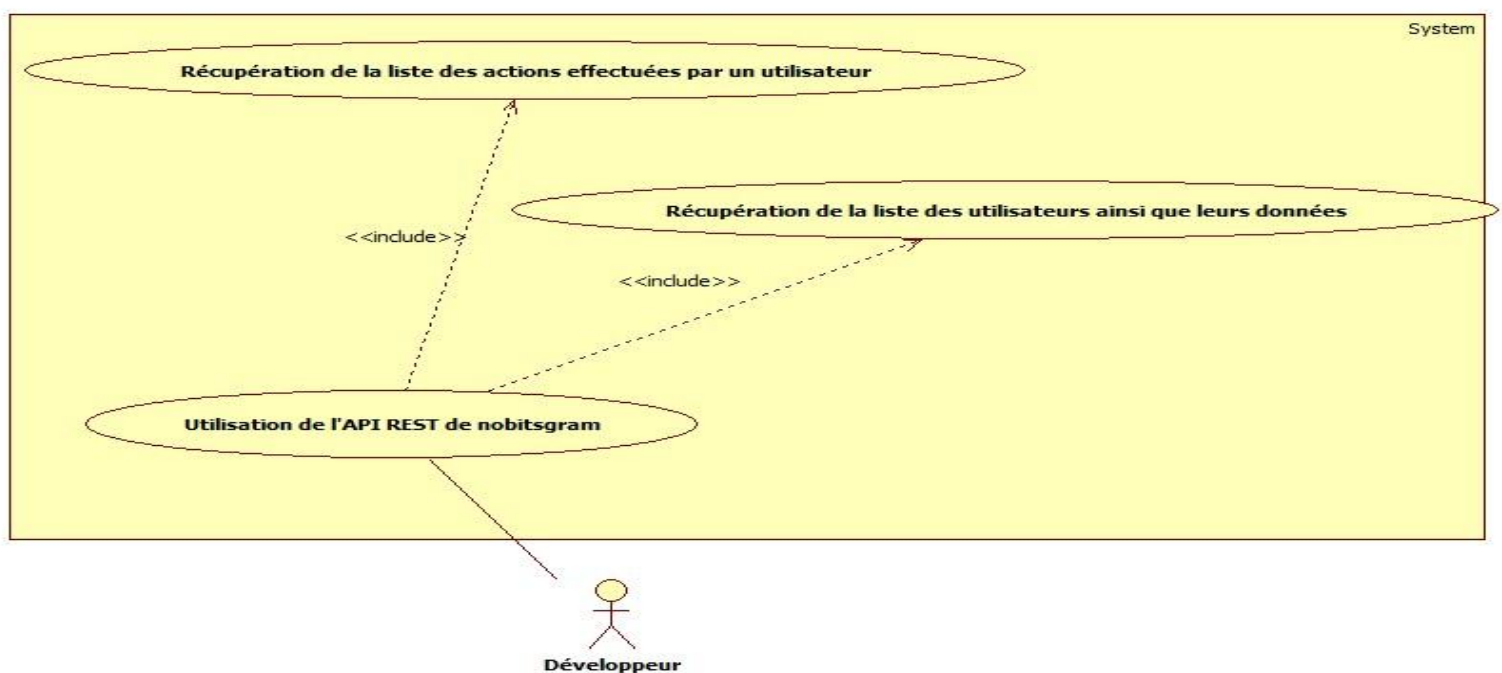


Figure 8 Use case développeur

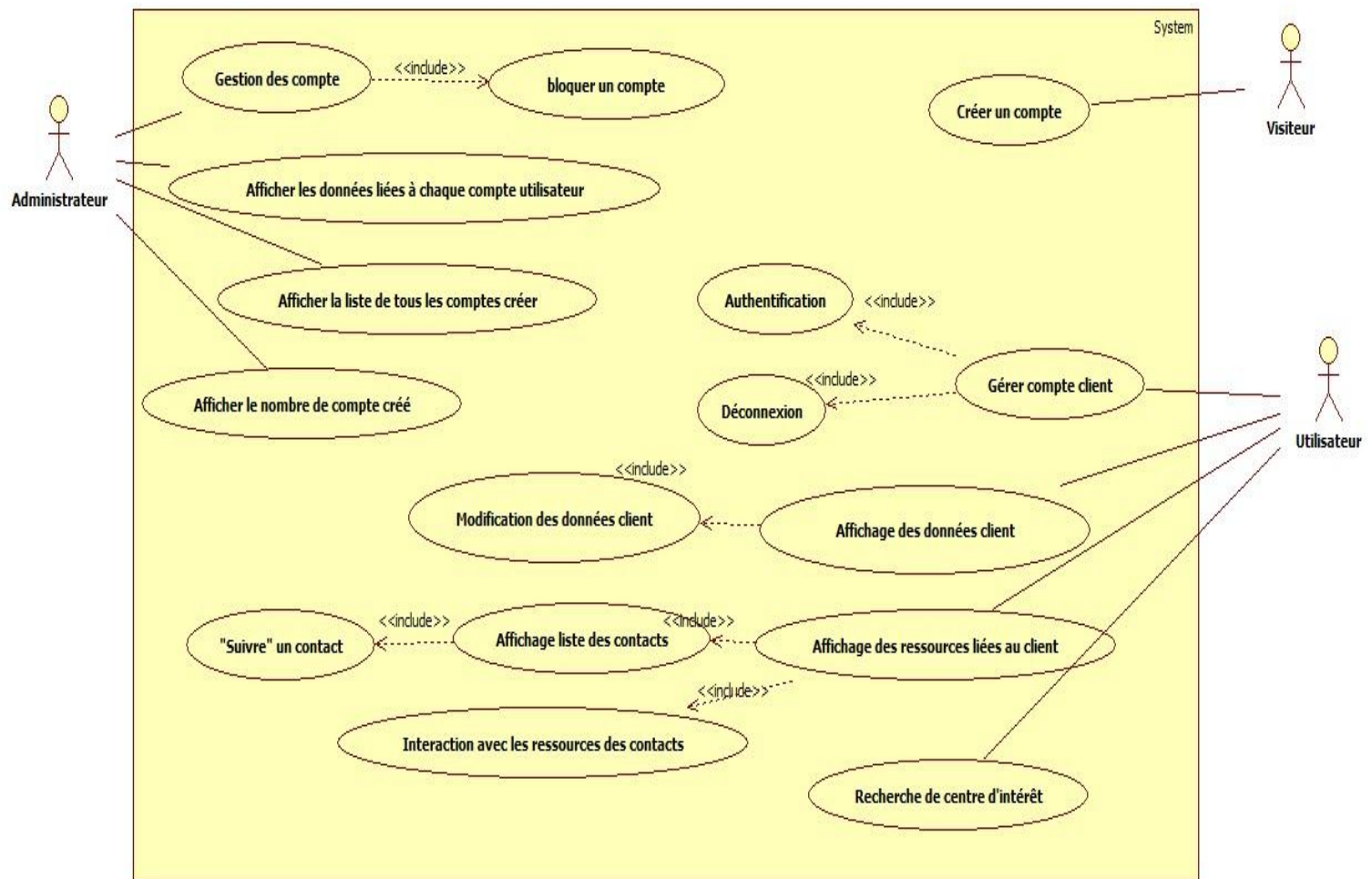


Figure 9 Use case 1 (Administrateur, utilisateur, visiteur)

Les diagrammes « Use Case » ont été construits sur la base des spécifications du projet. Ainsi, dans les cas d'utilisations de l'utilisateur, nous remarquons qu'il peut gérer un compte (s'authentifier et se déconnecter du contact). Ce cas d'utilisation, nous devons mettre en place une interface permettant à l'utilisateur de s'authentifier et de se déconnecter de son compte.

Un autre cas d'utilisation important est l'affichage des ressources liées au client (utilisateur). Dans ce cas précis nous parlons de ressources provenant d'Instagram, c'est-à-dire les photos taggées, les photos que le client aime « like », les « followers » et les personnes qui ont pour « follower » le client (l'utilisateur). Comme précédemment, l'application doit fournir des interfaces afin que le client puisse interagir avec ses données se trouvant sur Instagram.

Les cas d'utilisations de l'administrateur résument à la gestion des données des clients de nobitsgram et à la gestion de leur compte. Ainsi, l'administrateur a la possibilité de bloquer un compte si le propriétaire du compte ne respecte pas les clauses du contrat de

nobitsgram. Il a aussi la possibilité de voir la liste de tous les utilisateurs ainsi que toutes les actions qu'ils ont effectuée depuis la création de leur compte.

La troisième personne intervenant dans les cas d'utilisations est le développeur. Au travers d'une API REST, notre application devrait fournir des services aux développeurs voulant accéder à nos ressources. Les services qui devront être proposés sont la récupération des utilisateurs de nobitsgram ainsi que leurs données et la liste des actions qu'ils ont effectuées.

Notre implémentation, prévoit un cas d'utilisation pour un quelconque visiteur de nobitsgram. Le visiteur est considéré comme un utilisateur anonyme et il a juste la possibilité de créer un compte sur nobitsgram, dans le cas où il n'en possède pas encore un.

Vu que nous aurons à manipuler des données, il s'avère nécessaire de penser à l'allure de notre base de données.

9.2- Schéma de la base de données

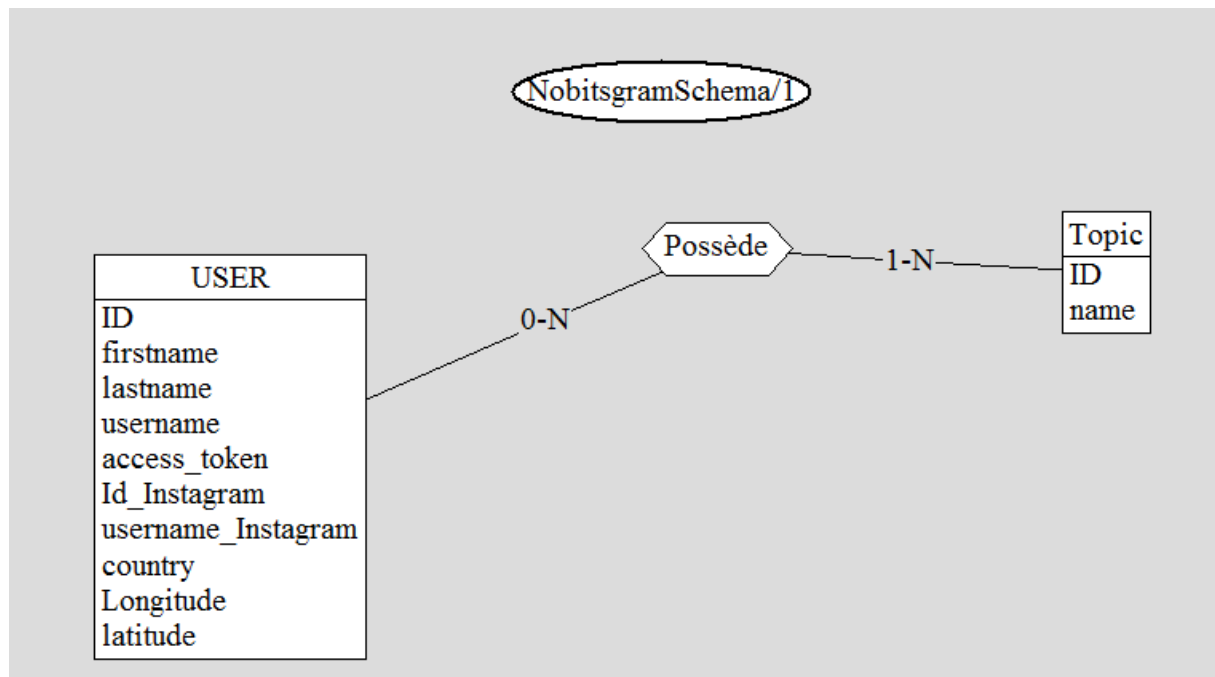


Figure 10 Schéma de la base de données de nobitsgram

La première partie de l'application consiste à mettre sur pieds une interface sur laquelle un utilisateur de nobitsgram pourra interagir avec ses données se trouvant sur le serveur d'Instagram. Un utilisateur est défini par son prénom (firstname), son nom (lastname), son pseudo (username), éventuellement son adresse (adresse converti en coordonnées géographiques – longitude et latitude –), son pays (country), ainsi que quelques données (access token, ID_Instagram, Username_Instagram) reçues d'Instagram lors de son enregistrement sur nobitsgram.

L'utilisateur a la possibilité de définir ses centres d'intérêts (topic). Ainsi, un utilisateur peut avoir plusieurs centres d'intérêts (topics) et un centre d'intérêt pourrait être partagé par plusieurs utilisateurs. Nous aurons entre autre alors une entité pour représenter un centre d'intérêt.

La relation qui lie l'entité Utilisateur (User) à l'entité centre d'intérêt (Topic) est de cardinalité 0-N. En effet, un utilisateur peut avoir 0 ou plusieurs centres d'intérêts. Par contre, la cardinalité de « Topic » à « User » est de 1-N. Un topic doit nécessairement être rattaché à un User, sinon il n'a pas sa raison d'être.

Notons qu'au fil de l'avancement du projet, notre base de données serait amenée à évoluer et à changer (possibilité d'ajout de nouveaux attributs sur les entités déjà existantes et création de nouvelles entités).

Nous avons spécifié plus haut, que nous optons pour le modèle MVC (Modèle – Vue – Contrôleur) comme architecture pour ce projet. Il convient dès lors d'expliquer comment cette architecture sera mise en place

9.3 – Modèle MVC (Modèle – Vue – Contrôleur)

9.3.1- Modèle

Le rôle du modèle de représenter les données du domaine et de fournir les méthodes permettant l'accès et la modification. Il est indépendant de la logique applicative. Le modèle dans notre cas sera les EJB (Enterprise Java Beans). En effet, les EJBs que nous implémenterons devront mettre à disposition des méthodes permettant le CRUD (Create Read Update Delete). Ainsi, les EJBs devront permettre de créer de nouvelles données (User et Topic – pour le moment -), de modifier leurs attributs, d'éditer leurs attributs, de mettre à jour l'élément après une modification et de supprimer l'élément.

9.3.2- Vue

La vue est ce qui apparaît dans le « browser » de l'utilisateur. Dans notre cas, c'est du code HTML interprété par le navigateur du client. Dans notre cas, la technologie de production de contenu utilisée est la page JSP (Java Server Page). La vue utilise le langage HTML éventuellement Javascript et JQuery. Il est à noter qu'à aucun moment, la vue ne pourra modifier les données affichées. Elle joue juste le rôle de processus de fabrication de page Web en fonction des données qui lui sont fournies.

9.3.3- Contrôleur

Le contrôleur est le module qui fait le lien entre le modèle et la vue. Il communique avec le modèle pour traiter des informations reçues à partir de la vue ou soit pour aller chercher les informations afin de les envoyer à la vue pour qu'elle les affiche. Le contrôleur décide de la vue à présenter en fonction des informations reçues du modèle.

Le contrôleur transforme les requêtes utilisateurs en requêtes métiers. Ainsi, il aura pour tâche de vérifier les données entrantes avant de les soumettre au modèle. La logique veut que pour chaque cas d'utilisation, nous ayons un contrôleur pour le gérer. Ainsi, nous aurons un contrôleur pour gérer la création de compte utilisateur (RegistrationServlet), un autre pour gérer la connexion (LoginServlet), encore un autre pour la modification des données du client (ServletPersonnalPage), ainsi de suite.

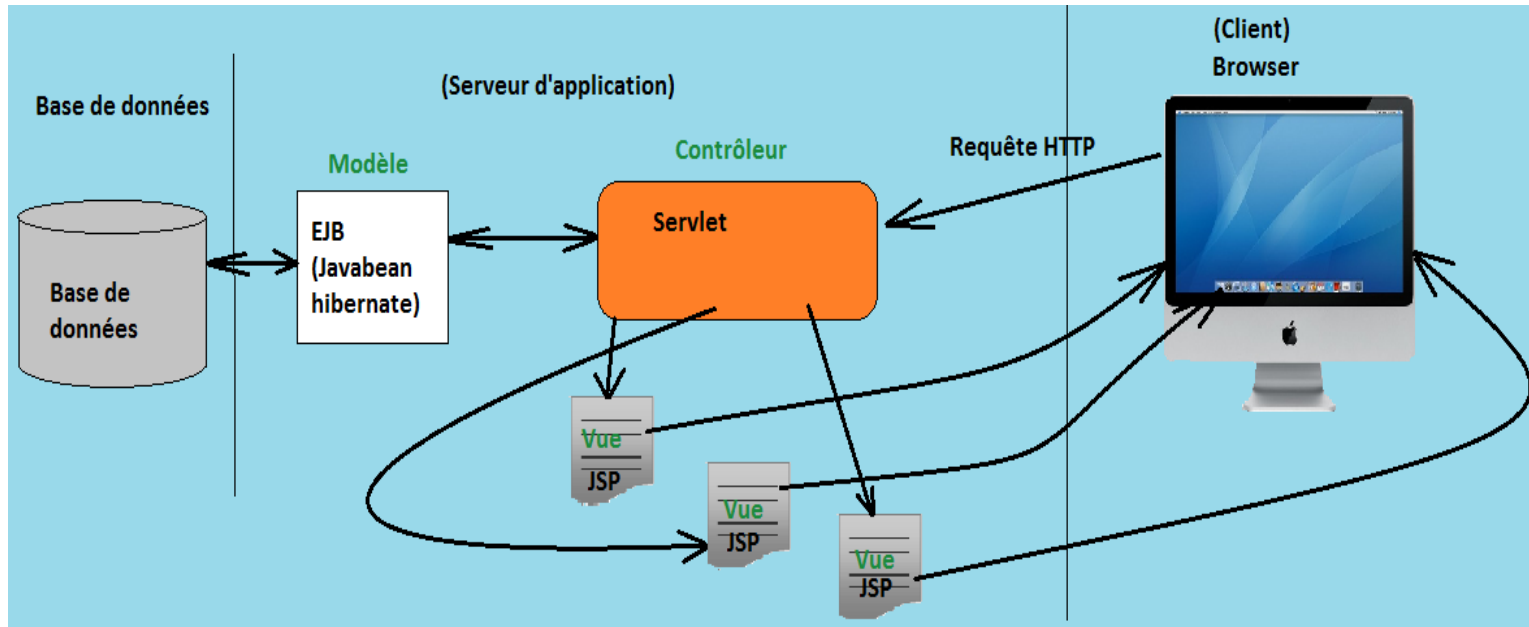


Figure 11 Schéma représentant l'architecture MVC (Modèle - Vue - Contrôleur)

L'étape suivante avant le début de l'implémentation proprement dite est l'établissement d'un diagramme UML.

9.4 – Diagramme UML et schéma d'interaction entre les différents modules de l'architecture MVC

Pour une bonne mise en œuvre de nobitsgram, il s'avère nécessaire d'établir un diagramme présentant les interactions entre les différentes parties de l'application. D'où le diagramme UML. Notons que principalement dans notre cas, nous nous sommes permis d'ajouter quelques modifications aux différents diagrammes afin de bien saisir l'interaction entre les différents modules de l'architecture MVC.

9.4.1- Page d'accueil

L'élément qui se charge de présenter la page d'accueil est le fichier « **pagelogin.jsp** ». C'est la page d'accueil de nobitsgram. Tout personne arrivant sur cette page est considérée comme utilisateur anonyme à moins qu'il se connecte sur son compte. Cette page sera assez simple, et présentera juste un champ pour le login et un lien vers l'adresse d'authentification OAuth d'Instagram. Outre ces deux composantes, selon la spécification, la page devra pouvoir afficher dynamiquement et aléatoirement chaque 5 secondes, une

image provenant d'Instagram et taggé avec le mot « nobits ». Cette page sera attribuée comme page de démarrage à l'application Instagram.

9.4.2- Enregistrement et page d'enregistrement

Avant tout enregistrement, l'utilisateur doit posséder un compte Instagram. Ainsi, lorsqu'il sera dirigé vers l'adresse d'authentification OAuth d'Instagram, il pourra recevoir le code qui permettra à nobitsgram de récupérer ses données (access token, ID, Username) sur Instagram. Lorsque l'utilisateur se sera authentifié sur Instagram, il sera redirigé vers la servlet « RegistrationServlet » qui s'occupera de recevoir le code et de communiquer avec le serveur d'Instagram afin de recevoir les données du client. Ensuite la servlet redirigera le client vers la page JSP d'enregistrement sur laquelle se trouve un formulaire.

L'enregistrement se fait suivant les étapes suivantes :

➤ Etape 1 : Réception du code à partir d'Instagram

Lorsque le client a pu s'authentifier sur Instagram, le serveur de ce dernier envoie une requête GET au serveur de nobitsgram et dans laquelle le code est enveloppé dans le quering. Il suffit donc de faire appel à la méthode « request.getQuering() » pour récupérer la chaîne de caractère du code et faire un parsing pour l'y extraire. Le parsing se fait à l'aide d'une méthode static de la classe « MyParser ». Notons que la réception du code ne peut se faire que dans la méthode **doGet(HttpServletRequest , HttpServletResponse)**. En effet, puisque le serveur d'Instagram fait une requête « GET » en direction de notre serveur, le seul moyen de voir un flux de données c'est se trouver dans la méthode invoquée.

➤ Etapes 2, 3 et 6' : Redirection vers la page d'enregistrement

La particularité d'une servlet est de partager ses attributs globaux avec toutes les instances qui y font appel. L'idée de déclarer une variable globale et ensuite de l'affecter au code pour le réutiliser comprend le risque que le code soit le même pour tous les utilisateurs. Ce qui est absurde. Une façon d'y remédier est d'envoyer le d'assimiler le code à la session, à travers la méthode « **request.getSession.setAttribute("code", code) ;** » , créer par celui qui s'enregistre. Ainsi, lorsqu'il aura fini de remplir le formulaire et qu'il le soumettra à la servlet, cette dernière va récupérer le code à la même occasion que les valeurs des champs qu'il aura rempli.

➤ Etape 4 : Communication avec le serveur Instagram

La communication avec le serveur Instagram se fait par l'intermédiaire de la requête Post. Le serveur de nobitsgram envoie une requête Post dans laquelle il aura ajouté les paramètres de requêtes suivants :

client_id = CLIENT-ID (celui de l'application)

client_secret = CLIENT-SECRET (celui de l'application)

grant_type = authorization_code (cette valeur reste telle quelle)

redirect_uri = Adresse_de_redirection (l'adresse de redirection de notre application)

code = CODE (Le code reçu d'instagram)

Si le serveur accepte la requête, il renverra une réponse au travers d'une méthode Post, réponse dans laquelle se trouveront les informations (access token, ID, Username,...) concernant le compte Instagram du client.

➤ Etape 5' : Redirection vers la page d'erreur d'Instagram

Lorsqu'une erreur est trouvée dans les valeurs des champs soumis à la servlet, l'utilisateur est redirigé vers la page d'erreur du formulaire.

➤ Etape 6 : Redirection vers la page client

Si le formulaire soumis par le client qui s'enregistre est accepté, le client sera automatiquement redirigé vers sa page client personnel. Sur cette page, le client pourra interagir avec ses données récupérer sur le serveur d'Instagram et avec les ressources se trouvant sur le dit serveur.

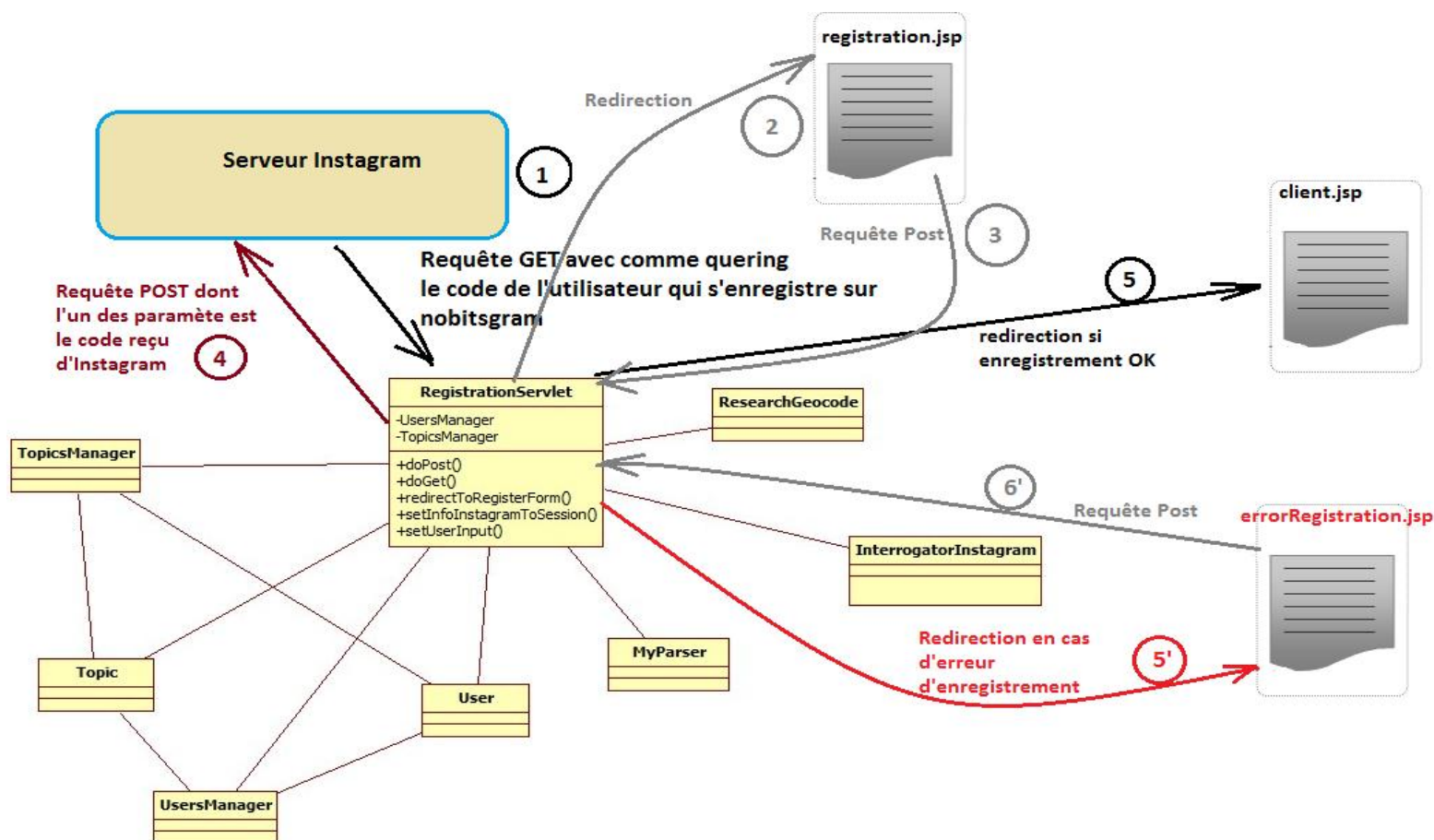


Figure 12 Diagramme UML de la Servlet gérant l'enregistrement et interactions avec les pages JSP correspondantes

9.4.3- Page de recherche et servlet gérant la recherche

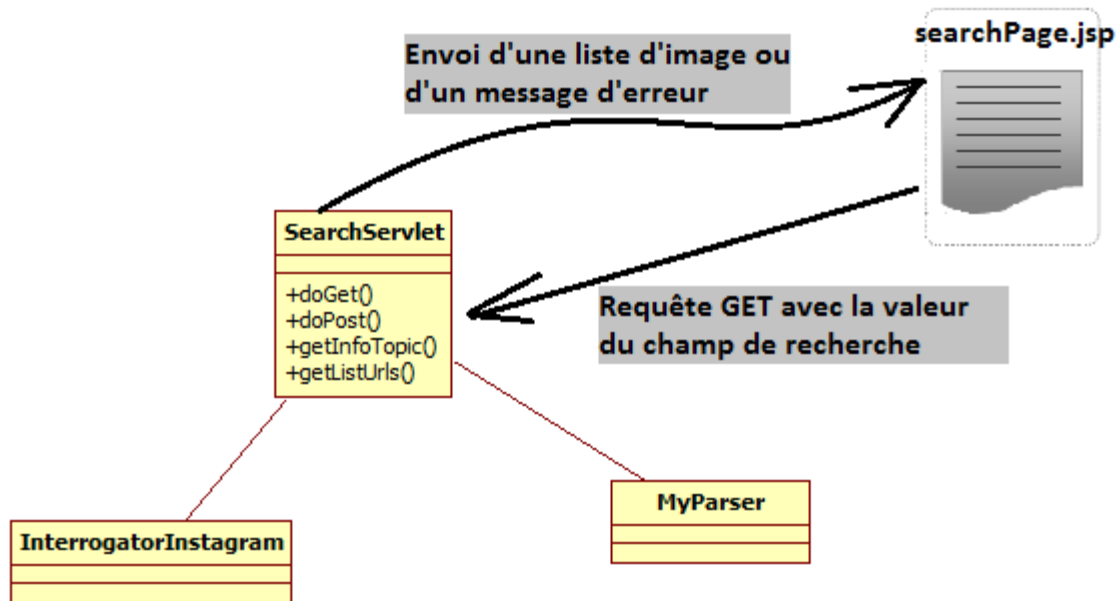


Figure 13 Diagramme UML de la servlet gérant la recherche et de son interaction avec sa page JSP correspondante

Lors d'une recherche, la page JPS de recherche envoie une requête GET à la servlet, requête dans laquelle se trouve la valeur du champ de recherche. Si le champ est vide, la servlet va automatiquement envoyer un message d'erreur à sa page JSP. La page JSP, va consulter les éléments se trouvant dans la réponse de la servlet. Si elle trouve que l'élément « error » n'est pas « null » alors elle va afficher le message d'erreur envoyé par la page JSP.

Dans le cas où la valeur du champ de saisie n'est pas nulle, la servlet va construire l'URL de recherche selon la documentation de l'API d'Instagram à savoir insérer le terme de la requête ainsi que l'access token de l'utilisateur selon le format suivant :

```
https://api.instagram.com/v1/tags/Terme_recherché/media/recent?access_token=ACCESS-TOKEN
```

Le serveur va envoyer une réponse dans laquelle se trouveront les liens vers les images ayant un rapport avec le terme recherché. La communication avec le serveur d'Instagram se fait au travers d'une instance de la classe **InterrogatorInstagram**. Cette instance reçoit une réponse sous forme de chaîne de caractères qu'il faudra parser afin d'en extraire les liens des images. Ces liens seront enregistrés dans une liste qui sera envoyée à la page JSP « searchPage » afin qu'elle puisse les afficher.

Si la recherche s'avère infructueuse du côté d'Instagram, son serveur retourne juste une chaîne de caractères dans laquelle ne se trouve aucun lien.

Pour connaître les informations concernant le terme recherché en l'occurrence le nombre d'image ayant un rapport avec le dit thème, il suffit d'envoyer une requête au serveur d'Instagram selon le format suivant :

```
https://api.instagram.com/v1/Terme_recherché/nofilter?access_token=ACCESS-TOKEN
```

Le serveur retournera une réponse qui aura le format suivant :

```
{
  "data": {
    "media_count": 472,
    "name": "nofilter",
  }
}
```

Il suffit donc de parser la réponse reçu du serveur pour en extraire la donnée voulue à savoir le nombre de média (images) ayant un rapport avec le terme recherché.

9.4.4- Servlet gérant la déconnexion



Figure 14 Diagramme de la servlet gérant la déconnexion et son interaction avec la page JSP correspondante

Lorsqu'un client se déconnecte, il effectue une requête GET à la servlet à la servlet LogoutServlet. La méthode doGet() invoque la méthode toDisconnect() pour fermer la session. La fermeture de la session se fait au travers la méthode « **request.getSession..invalidate();** ». Après l'exécution de cette instruction, le client est redirigé vers la page d'accueil de l'application.

9.4.5- Servlet gérant la modification des données et pages JSP correspondantes

Pour modifier ses données, l'utilisateur doit aller sur la page « My account ». Arrivé sur cette page il a la possibilité de voir toutes ses données et de les modifier. Les seules données qu'il ne peut pas modifier est son Identifiant (nombre) nobitsgram, son pseudo nobitsgram et ses données Instagram (access token, username, id).

L'utilisateur a la possibilité d'ajouter ou de supprimer un centre d'intérêt. Lorsque l'utilisateur demande à afficher sa page personnelle, il fait une requête GET à la servlet « ServletPersonnalPage ». Cette servlet récupère les données du client et les envoie à la page JSP « settingAccount.jsp ». La servlet insère la liste des topics dans la requête envoyée à la page « settingAccount.jsp ». La page JSP extrait les données se trouvant dans la requête et les affiche. Lorsque, l'utilisateur modifie une ou plusieurs de ses données, il envoie une requête Post à la servlet ainsi que la valeur des champs à modifier. Si aucun champ n'est modifié, la servlet valide le formulaire. Par contre si l'utilisateur modifie une ou plusieurs données et que l'une au moins d'entre elles n'est pas correcte, la servlet redirige le client vers la page « My account » avec un message d'erreur correspondant à l'erreur trouvée.

Dans le cas où le formulaire est accepté par la servlet et que la modification a été effectuée, le client est automatiquement redirigé vers la page client.

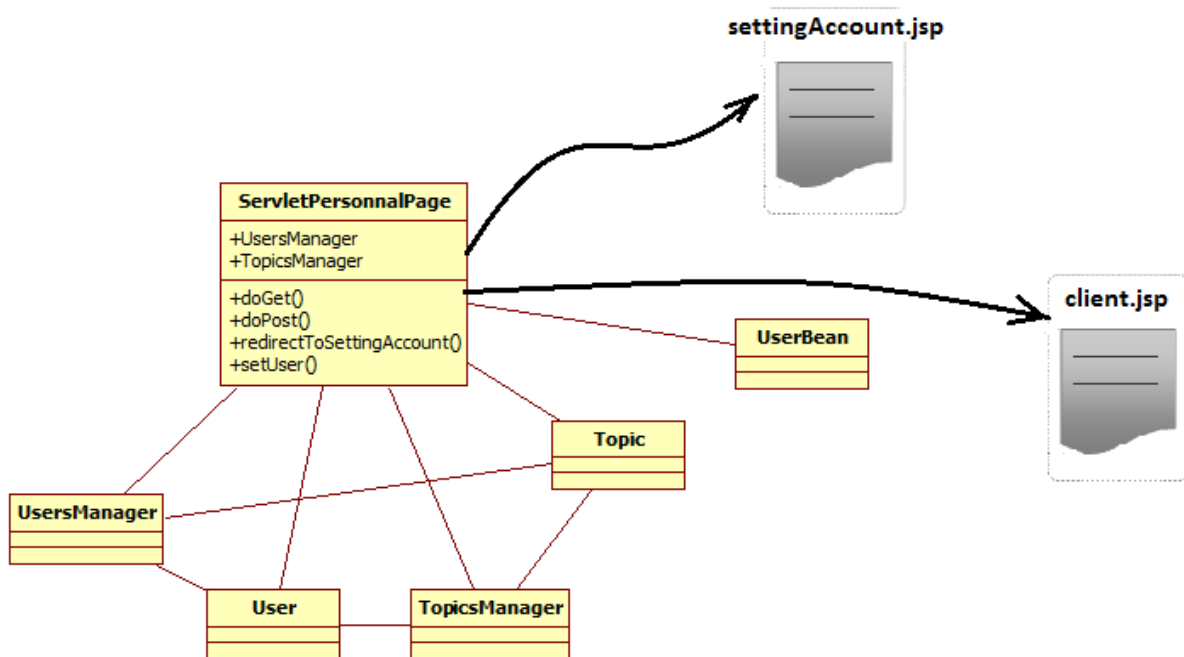


Figure 15 Diagramme de UML de la servlet gérant la modification des données client et interaction avec ses pages correspondantes

9.4.5- Classe entité **USER**

La classe entité User représente l'entité User. Elle met à disposition des méthodes d'accès (getters) et de modification (setters) à disposition.

