

# Etude technique Application web Gask



Réalisé par : Ilias IRHBOULA (F2) & Mohammed BOUFFOUSS(F2)  
Encadré par : M. Alexis PLANTIN

## Table des matières

<b>1. Contexte .....</b>	<b>3</b>
<b>2. Fonctions de service .....</b>	<b>3</b>
<b>2.1. Fonctionnalité 1 .....</b>	<b>3</b>
<b>2.2. Ajouter une Question (exemple) .....</b>	<b>3</b>
<b>3. Description des données .....</b>	<b>3</b>
<b>3.1. Diagramme de classe de notre application .....</b>	<b>3</b>
<b>4. Architecture technique.....</b>	<b>5</b>
<b>4.1. Architecture applicative .....</b>	<b>5</b>
<b>4.1.1 Les domaines .....</b>	<b>5</b>
<b>4.1.2 Couche contrôleur .....</b>	<b>5</b>
<b>4.1.3 La couche vue.....</b>	<b>6</b>
<b>5- Tests unitaires.....</b>	<b>6</b>
<b>6- Règles du codage .....</b>	<b>7</b>
<b>7- Quelques interfaces de l'application .....</b>	<b>7</b>

## Table des figures

Figure 1: Diagramme de classes de l'application .....	4
Figure 2: Architecture de l'application .....	5
Figure 3: Test unitaire de UserController .....	6
Figure 4: Interface d'inscription .....	7
Figure 5: Liste des tags .....	8
Figure 6: Interface d'authentification .....	8
Figure 7: Liste des questions .....	8
Figure 8: Poser une question.....	8
Figure 9: Question & Réponses .....	8

## 1. Contexte

L'objectif de notre projet est de mettre en place une application web des questions/réponses destiné aux informaticiens. Le périmètre de ce projet s'inscrit dans un projet universitaire, cependant dans les envies personnelles on veut l'étendre dans une échelle un petit peu évolué. Ce projet a été développé en **grails 2.4.4** et **groovy/grails tool suite** come IDE.

## 2. Fonctions de service

### 2.1. Fonctionnalité 1

La fonctionnalité de notre application accomplit des données en entrées des différents utilisateurs soit des questions, réponses à ces questions des commentaires ect. Et comme sortie l'utilisateur qui pose une question peut recevoir une notification si sa question a eu une réponse, afin qu'il donne son avis sur les différentes réponses.

### 2.2. Ajouter une Question (exemple)

L'utilisateur saisit son contenu via un formulaire de saisie. Quand il clique sur le bouton Ask, le contenu est ajouté dans la base de données après tout un tas de validation. Le formulaire permet la saisie des informations suivantes :

- le contenu.
- les tags.

## 3. Description des données

### 3.1. Diagramme de classe de notre application

Le diagramme de classes dans notre application va représenter le schéma utilisé en génie logiciel pour présenter les classes du système ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques.

Une classe décrit les responsabilités, le comportement et le type d'un ensemble d'objets.

Une classe est un ensemble de fonctions et de données (attributs) qui sont liées ensemble par un champ sémantique dans notre cas on a mis juste un diagramme de classe non détaillé. Elles permettent de modéliser un programme et ainsi de découper une tâche complexe en plusieurs petits travaux simples.

Elles sont finalement instanciées pour créer des objets (une classe est un domaine à objet : elle décrit les caractéristiques des objets, les objets contiennent leurs valeurs propres pour chacune de ses caractéristiques lorsqu'ils sont instanciés).

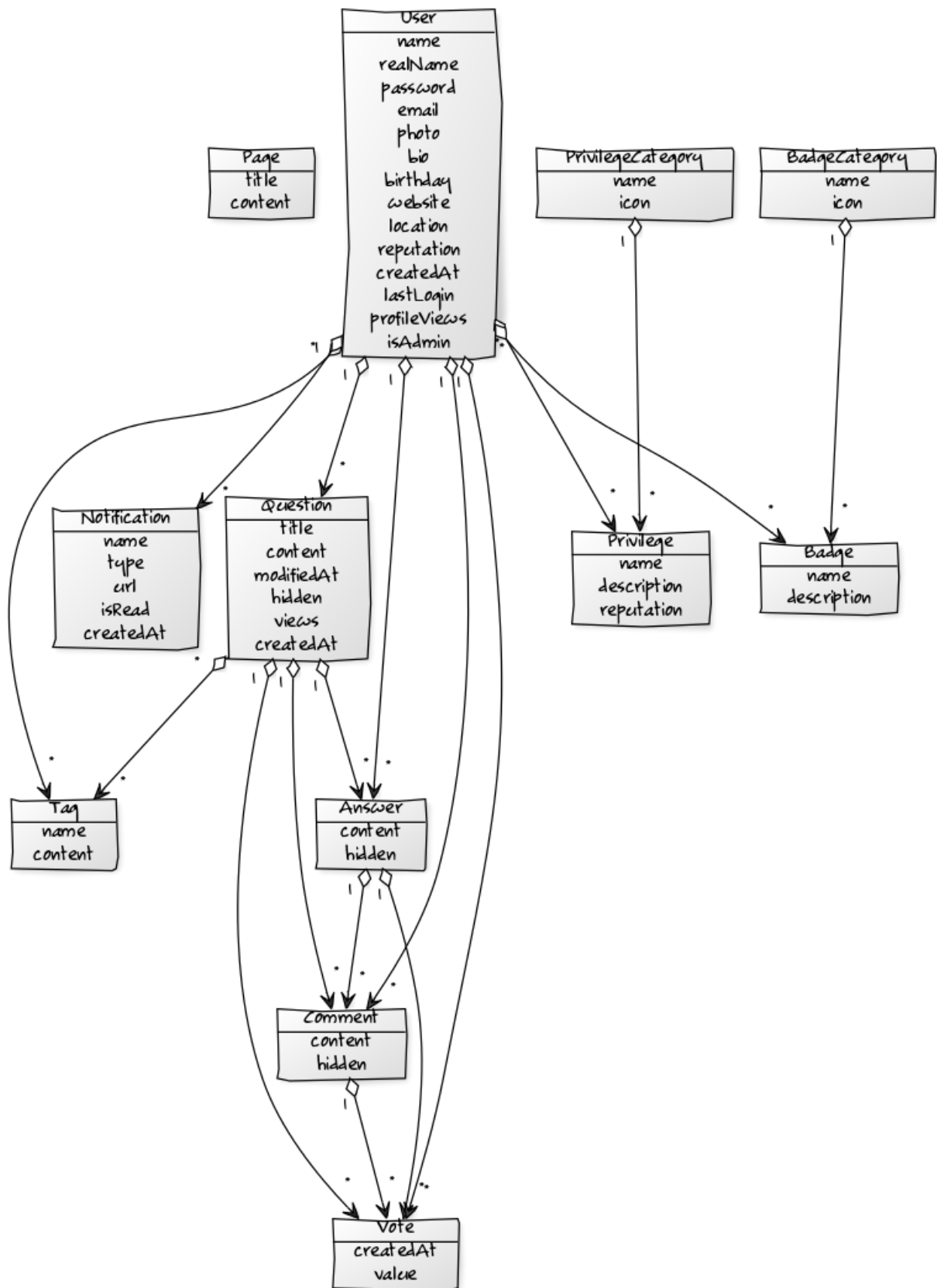


Figure 1: Diagramme de classes de l'application

## 4. Architecture technique

### 4.1. Architecture applicative

Notre application suit le modèle MVC :

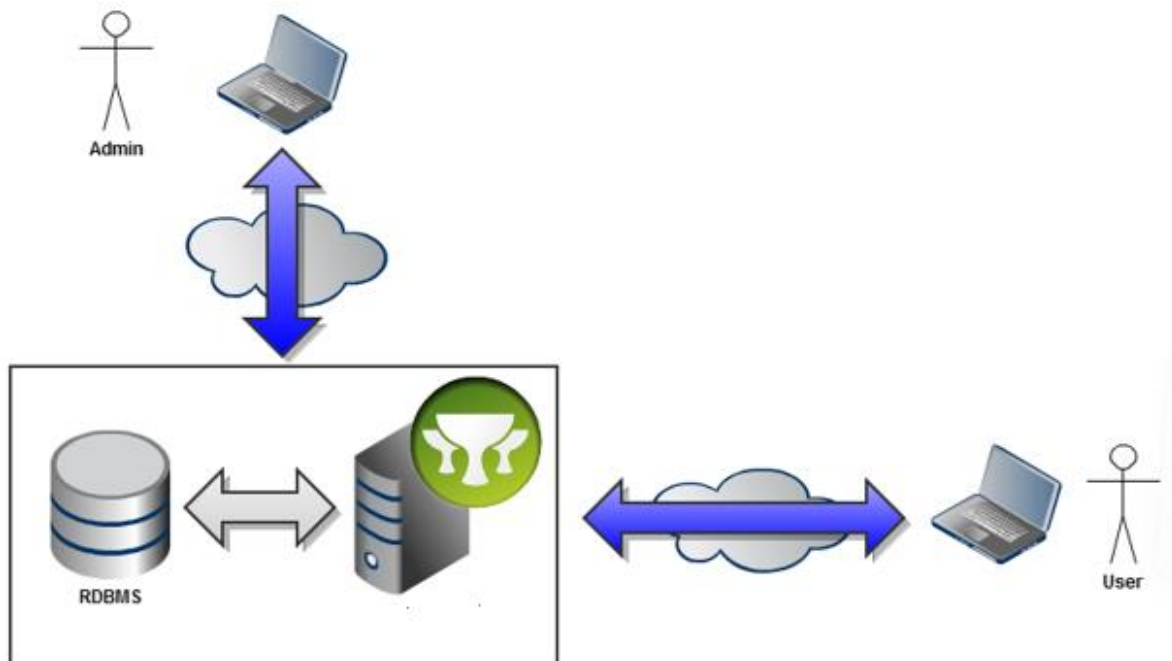


Figure 2: Architecture de l'application

#### 4.1.1 Les domaines

La partie couche d'accès aux données est préconfigurée côté serveur dans le fichier **BuildConfig.groovy**. Les données envoyées/récupérées par le client vers/de la base de données passent par les contrôleurs qui font des traitements et stockent ou récupèrent des données depuis/vers la base de données de type **Mysql**. Le fameux GORM qui est le mapping objet relationnel du framework Grails s'appuie sur Hibernate. Celui-ci est responsable de mapper les domaines définis dans la côté serveur et les tables relationnelles côté base de données.

#### Remarque

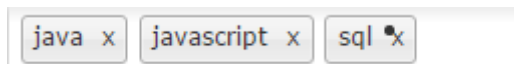
Avant de lancer l'application il faut d'abord changer les paramètres définis dans le fichier **BuildConfig.groovy** notamment l'utilisateur de la base de données mysql (root par défaut), le mot de passe (« » par défaut) et le nom de la base de données (data par défaut).

#### 4.1.2 Couche contrôleur

On a décidé de développer un contrôleur par domaine. Par exemple le contrôleur de l'utilisateur est responsable de traiter toutes requêtes liées à l'utilisateur notamment l'ajout d'un utilisateur lors de l'inscription, la vérification lors de la connexion...

### 4.1.3 La couche vue

C'est un ensemble de pages **gsp** pour la représentation des différentes interfaces à l'utilisateur. On a également utilisé le **Bootstrap** qui est un framework pour faire le design des pages web. On a utilisé les « **RemoteLink** » et « **RemoteForm** » pour faire des requêtes ajax pour la mise à jour des div sans avoir à rafraichir toute la page web. (par exemple le upvoting et le downvoting). On a également utilisé le plugin **OpenJS-Autofill** [1] qui est un plugin **Jquery** pour permettre l'ajout des différents tags dans un champ.



## 5- Tests unitaires

On a fait quelques tests unitaires pour tester les contrôleurs de l'application comme le test de « UserController »

```
@TestFor(UserController)
@Mock([User, Question])
class UserControllerSpec extends Specification {

    void 'test save'() {
        when:
            params.email="julien@gmail.com"
            params.userName="Julien"
            params.password="xx"
            controller.save()

        then:
            view == '/index1'// If we succeed to save a user in database the rendered page is index1.gsp
    }

    void 'test getUsers'() {
        when:
            controller.getUsers()

        then:
            view == '/users'// If we succeed to get all users from database the rendered page is users.gsp
    }

    void 'test logout'() {
        when:
            controller.logout()

        then:
            session["logged"]==false;
    }
}
```

Runs: 3/3   Errors: 0   Failures: 0

controllers.UserControllerSpec [Runner: JUnit 4] (11,570 s)

- test save (10,471 s)
- test getUsers (1,010 s)
- test logout (0,089 s)

Failure Trace

Figure 3: Test unitaire de UserController

## 6- Règles du codage

Les règles de codage qu'on a mis sont un ensemble de règles à suivre pour uniformiser les pratiques de développement de notre application, diffuser les bonnes pratiques de développement et éviter les erreurs de développement "classiques".

Les règles de codage s'articulent autour de plusieurs thèmes, les plus courants étant:

Le nommage et l'organisation des fichiers du code source : notre code est indenté, et les conventions de nommage des variables et les méthodes et très importantes afin d'éviter l'incohérence lors de l'appel d'une fonctionnalité depuis le client. Ainsi que, notre code source gère bien les différentes erreurs.

## 7- Quelques interfaces de l'application

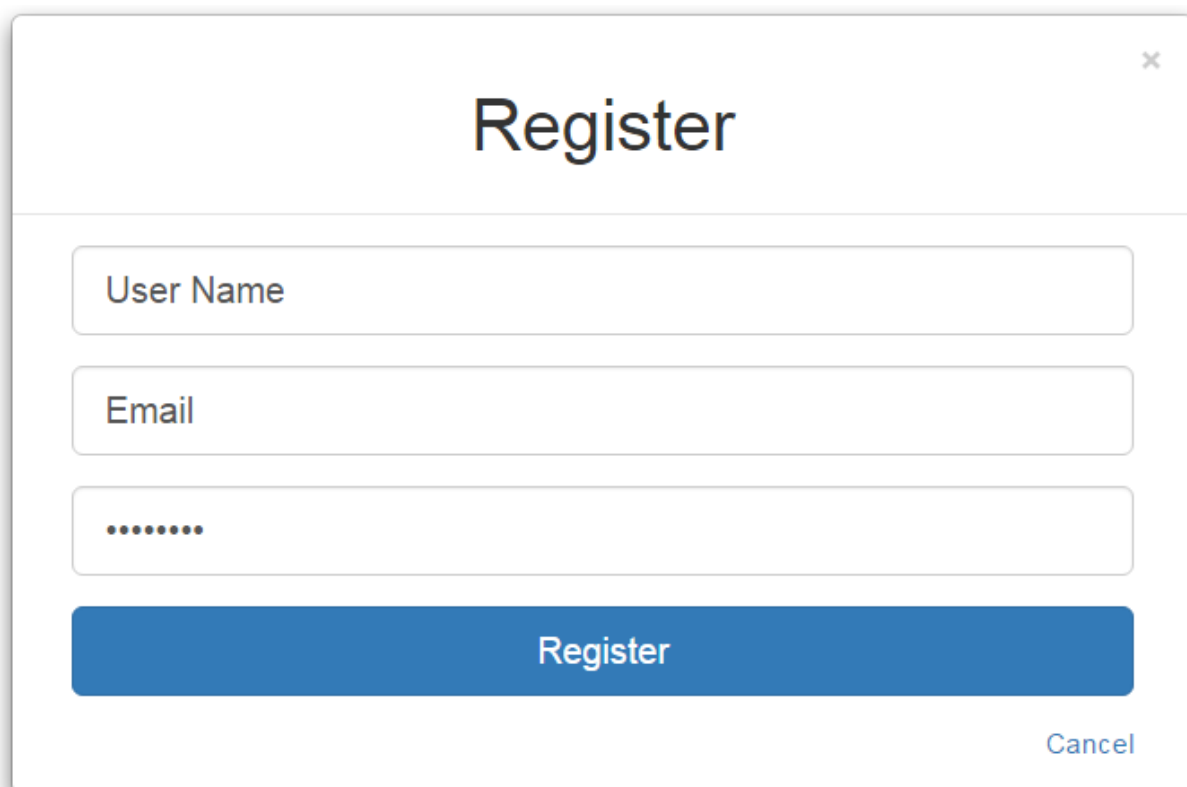
A screenshot of a web application's registration interface. It features a white rectangular box with a light gray border and a close button (an 'x' icon) in the top right corner. The title 'Register' is centered at the top in a large, bold, black font. Below the title, there are three input fields stacked vertically: the first is labeled 'User Name', the second is labeled 'Email', and the third contains a series of dots, indicating a password field. At the bottom of the form, there is a prominent blue button with the text 'Register' in white. To the right of this button, the word 'Cancel' is written in a smaller, blue font.

Figure 4: Interface d'inscription



×

# Login

Email

.....

Login

[Need help?](#)
[Register](#)
[Cancel](#)

Figure 6: Interface d'authentification

Gask	Home	Questions	Ask a question	Tags	Users	Profile	Logout	James
grails								
1 asks								
c++11								
1 asks								
c++								
1 asks								
c#								
1 asks								
compiler								
1 asks								

Figure 5: Liste des tags

Gask	Home	Questions	Ask a question	Tags	Users	Profile	Logout	James
Questions								
Answers	Votes	Title	Tag					
1	1	how to do save or update in Grails?	grails	gorm				
0	0	C# List of objects, how do I get the sum of a property	c#					
0	0	Proper way to prevent deallocation using std::allocator and std::move	c++	compiler	c++11			

Figure 7: Liste des questions

Ask a question

title

content

Ask

Figure 8: Poser une question

▲

how to do save or update in Grails?

if the record doesnt exist, it should insert. And if the record exists, Grails should do an update

gormgrails

1

James answered at 2016-02-19 11:23:41.0

▲

Dynamic finders are precisely available for this purpose.  
Using findOrCreateBy\* your first sample code would be written as such:  
def entity = MyEntity.findOrCreateByAttr1AndAttr2( val1, val2 )

0

Answer

Figure 9: Question & Réponses

9