

Rapport de soutenance


S. Drode - L. Ortolan - A. Pallas - J. Riffard - K. Savaroche

Interface 2037 est un service Web RESTFUL. Cette dernière permet l'usage de soumettre des questions à un serveur. Par la suite, un expert répondra à la question et le client pourra la consulter à tout moment.

Vous trouverez dans ce rapport un détail sur nos méthodes de travail ainsi que des justifications des choix effectués.

I. Utilisation des méthodes agiles

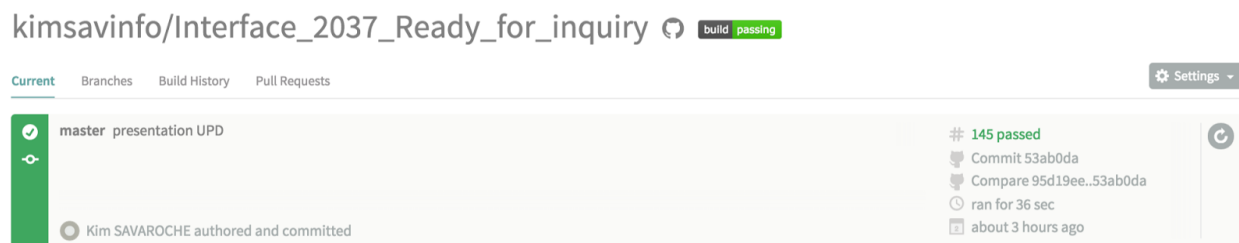
A. Intégration continue

Vous avez peut-être déjà remarqué le badge  dans le README d'un Github. Ce dernier est obtenu grâce au service [Travis-CI](https://travis-ci.com/).

Travis-CI build le projet puis exécute les tâches indiquées dans le fichier [travis.yml](https://travis-ci.com/). Nous l'utilisons pour lancer les tests unitaires et les tests d'acceptation à chaque PUSH.

Ainsi, chaque nouvelle implémentation est contrôlée et nous sommes avertis rapidement si un test ne passe plus. Nous utilisons donc TRAVIS-CI pour l'intégration continue de notre projet.

Interface de TRAVIS-CI



B. Développement logiciel

Nous avons abordé la technique de développement logiciel TDD (Test Driven Development) qui permet d'avoir une certaine qualité dans la production de travail.

Les tests sont découpés en 2 parties :

- tests unitaires : ils font partis de l'API elle-même. Ils sont responsables d'assurer le bon fonctionnement de traitements internes.
- tests d'acceptations : ils testent l'API sans en faire parti. Ils permettent de valider le travail en rapport à la demande cliente. Ces tests reprennent les scénarios de la demande cliente.

== Scenarii ==

Serveur de tests d'acceptation Scenarii lance

Etant donné qu'il n'existe aucune question

Quand le système expert demande la prochaine question au serveur

Alors le serveur indique qu'il n'existe pas de question

Et le système expert se met en veille avant de redemander une question

- Il n'existe aucune question en base

- ✓ Aucune question pour le système expert

Quand l'utilisateur pose une question au serveur

Alors le serveur indique qu'il a enregistré la question

Et il permet à l'utilisateur de localiser la réponse lorsqu'elle sera disponible

- L'utilisateur a posé une question au serveur

- Elle sera disponible à l'adresse : /client/questions/551dc00ead1e4099195e5b03

- ✓ L'utilisateur pose une question au serveur et sait où elle sera disponible

Etant donné qu'il existe une question en attente de réponse

Quand le système expert demande la prochaine question au serveur

Alors il récupère la question en attente

Et la question suivante devient la question en attente

- L'utilisateur a posé une seconde question au serveur

- Le système expert récupère la première question posée et la seconde passe en attente

- ✓ Le système expert propose la question en attente

Etant donné qu'un utilisateur a posé une question

Et aucun système expert n'a pas encore traité la question

Quand l'utilisateur demande à consulter la réponse

Alors le serveur indique que la réponse n'est pas encore disponible

- L'utilisateur visualise la question sans réponse.

- ✓ L'utilisateur consulte la question mais elle n'a pas encore de réponse

Etant donné que le système expert a récupéré une question en attente

Et qu'il a trouvé une réponse

Quand il fournit la réponse au serveur

Alors le serveur indique qu'il a enregistré la réponse à la question

Question répondue avec succès ! id :undefined

- La réponse a bien été enregistrée et est disponible à l'adresse :

- ✓ Le système expert fournit une réponse

Etant donné qu'un utilisateur a posé une question

Et un système expert a traité la question

Quand l'utilisateur demande à consulter la réponse

Alors le serveur affiche la réponse du système expert

- L'utilisateur visualise la question avec la réponse du système expert.

- ✓ L'utilisateur consulte la question et le système a bien répondu

Etant donné que le système expert a récupéré une question en attente

Et qu'il a trouvé une réponse

Quand il notifie le serveur de son échec

Alors le serveur enregistre que cette question n'a pas de réponse connue.

Question répondue avec succès ! id :undefined

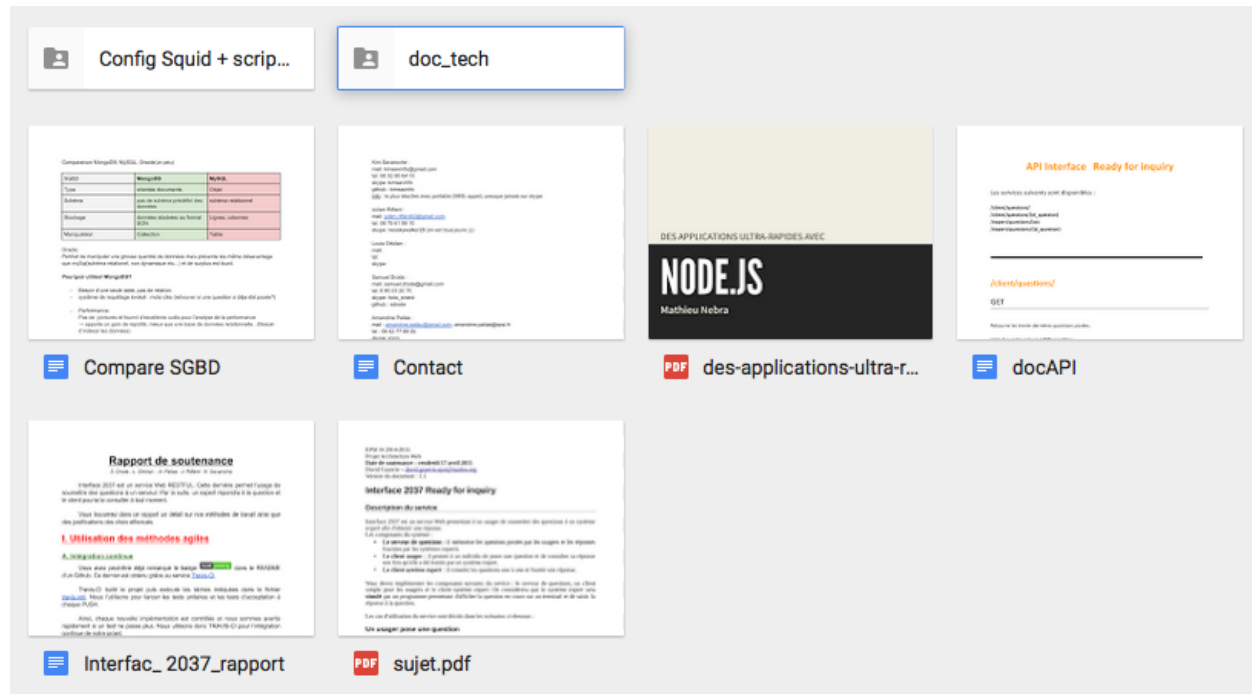
- La réponse a bien été enregistrée et est disponible à l'adresse :

- L'utilisateur visualise la question qui n'a pu être répondue par l'expert

- ✓ Le système expert indique qu'il ne connaît pas la réponse

C. Communication

Pendant la phase de développement du projet, plusieurs méthodes de communication ont été utilisées. Premièrement, nous avons utilisé google drive et Github pour la mise en commun des sources et documentations.



Pour ce qui est de la communication directe, le groupe s'est appuyé sur les moyens de télécommunication actuels (téléphone, mail, skype etc...) pour travailler à distance ainsi que des séances organisées à l'EPSI pour la mise en commun des travaux individuels.

A chaque début de séance, nous faisons un point sur l'avancée de chacun : l'objectif est-il atteint, a-t-on besoin d'aide pour un point, quel est programme de la séance, etc. A la fin, nous nous fixons des objectifs pour la semaine d'alternance en entreprise.

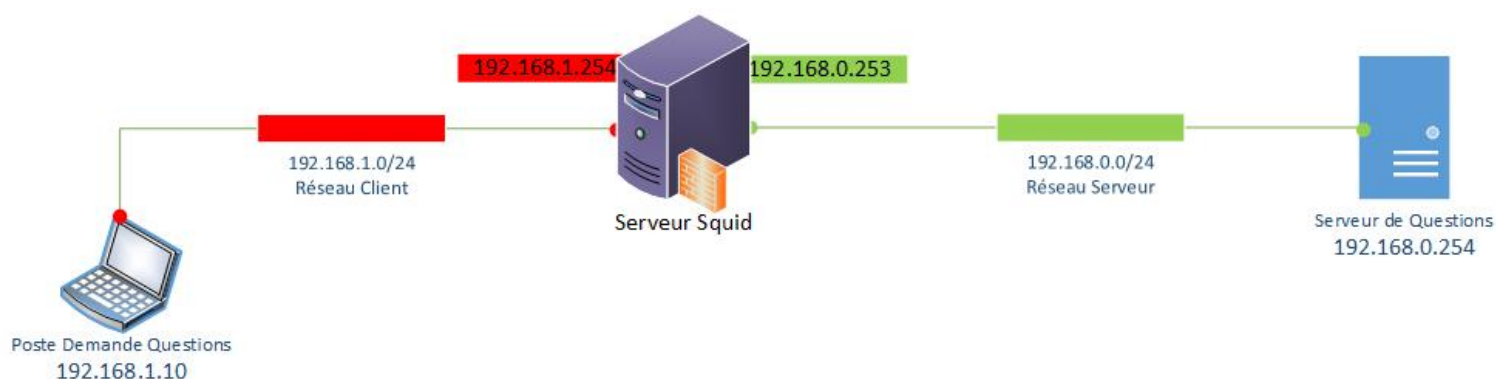
II. Architecture

A. Application

L'application se divise en deux parties : une partie API RESTFUL et une partie Interface graphique cliente (au sens large).

L'API est disponible sur Github a la racine du projet. L'interface graphique se trouve sur Github dans le dossier "*public*".

B. Schéma Réseau



Légende		
Symbole	Total	Description
	1	Serveur
	1	Ordinateur portable
	2	Réseau Ethernet
	1	Serveur Proxy

Choix de notre infrastructure :

Nous avons décidé de séparer l'infrastructure par deux réseaux différents **192.168.0.0/24** et **192.168.1.0/24**. Le serveur de questions étant configuré en **192.168.0.254** et le client de question en **192.168.1.10**.

Le serveur proxy Squid joue le rôle de routeur entre les deux réseaux. Il a donc été configuré avec deux interfaces réseaux : **192.168.1.254** pour la partie client et **192.168.0.253** pour la partie serveur.

Cette configuration permet ainsi de cloisonner chaque partie de l'infrastructure, client d'une part et le serveur de d'autre part.

En tant que serveur mandataire ainsi qu'une mise en place de règles de routage, toutes les requêtes du client de question passent par le serveur Squid. C'est ce dernier qui transmettra les requêtes au serveur de réponse.

Le Squid transmettra alors la réponse du serveur de réponse avec une information sur la propension de cette réponse à être mise en cache, comme la fraîcheur, sa date de création, si elle doit être conservée dans le futur.

Ainsi, notre infrastructure est en adéquation avec l'architecture RESTful :

- Système hiérarchisé par **couche** (Client de questions, Serveur Squid, Serveur de réponses)
- Mise en cache via le Serveur Squid permettant au client de ne pas faire de requêtes inutiles. De plus, ceci permet de répondre à la problématique de l'extension à N usagers. En effet, la mise en cache permet de ne pas solliciter continuellement le serveur et donc supporter un grand nombre de clients usagers.

Fichier de configuration réseau du serveur Squid (/etc/network/interfaces)

```

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# Interface coté serveur de question
auto eth0
iface eth0 inet static
address 192.168.0.253
network 192.168.0.0
netmask 255.255.255.0

#Interface coté client
auto eth1
iface eth1 inet static
address 192.168.1.254
network 192.168.1.0
netmask 255.255.255.0

#Regles de routage
#On route le trafic à destination du réseau 192.168.0.0 via l'interface eth1:
up route add -net 192.168.0.0/24 gw 192.168.1.254/24 dev eth1
#On route le trafic à destination du réseau 192.168.1.0 via l'interace eth0:
up route add -net 192.168.1.0/24 gw 192.168.0.253/24 dev eth0
~
~
~
"/etc/network/interfaces" 26L, 767C                               1,1          Tout

```

Activation du routage sur le serveur Squid (/etc/sysctl.conf)

```

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

```

Fichier de configuration réseau du serveur de question

[illegible]

Fichier de configuration du Squid (/etc/squid3/squid.conf)

```

1  # Nom du serveur SQUID.
2  visible_hostname EPSI-SQUID
3
4  # Port d'écoute du SQUID.
5  http_port 3000
6
7  # Créations des règles acl des réseaux connexes
8  acl localnet src 192.168.1.0/24
9  acl localhost src 192.168.0.0/24
10
11 # Autorisation des règles acl
12 cache_peer 192.168.0.254 parent 3000 0 no-query originserver name=myAccel
13 http_access allow localnet
14 http_access allow localhost
15 cache_peer_access allow myAccel
16
17 # Mémoire cache utilisée
18 cache_mem 128 MB
19
20 # Répertoire de stockage du cache. 1024 pour la taille de cache max. 16 et 256 correspondent aux
21 # arborescences des répertoires qui accueillent le hashage du cache.
22 # On aura donc ici, 16 répertoires qui contiendront chacun 256 répertoires.
23 cache_dir ufs /var/spool/squid3 1024 16 256
24
25 # Limitations en dehors desquels les objets ne seront
26 minimum_object_size 0 KB
27 maximum_object_size 10240 KB
28
29 # Logs erreurs écrites en français
30 error_directory error_directory /usr/share/squid3/errors/French
31
32
33 # Nombre maximum de versions archivées. En jours.
34 logfile_rotate 30
35
36 # Délais au delà duquel la transaction échoue.
37 connect_timeout 30 seconds
38 request_timeout 30 seconds
39 read_timeout 2 minutes

```

Voici les scripts utilisés pour automatiser les tâches de démarrage :


```
#####  
#Script de lancement du serveur de l'interface client  
#####
```

```
#!/bin/bash
```

```
# deplacement dans le dossier de l'interface client
```

```
cd /home/julien/Documents/Interface_2037_Ready_for_inquiry/public
```

```
#on lance l'interface client
```

```
npm start
```

```
#####  
#Script du lancement de la base donnees mongodb  
#####
```

```
#!/bin/bash
```

```
#deplacement dans le dossier de la base de donnees
```

```
cd /home/julien/Téléchargements/mongodb-linux-x86_64-2.6.7/bin
```

```
#lancement de la base de données
```

```
./mongod --dbpath db/ --smallfiles
```

```
#####  
#Script de lancement du serveur de question  
#####
```

```
#!/bin/bash
```

```
#Deplacement dans e dossier de l'application insterface
```

```
cd /home/julien/Documents/Interface_2037_Ready_for_inquiry
```

```
#On lance le serveur de question
```

```
npm start
```

III. Contraintes REST

Vous trouverez ci-dessous notre approche pour répondre aux contraintes REST d'une API :

A. Hypermédia

Lorsqu'un client se connecte à l'API, celle-ci lui renvoie une représentation de la ou des ressources. Chacune de ces ressources possède des informations hypermédia comme par exemple sa localisation :

Exemple :

```
[
  {
    "_id": "552a5e18f7e024ef02ed38a5",
    "publicationDate": 1428839957751,
    "answer": "reponse test",
    "label": "question-test",
    "links": [
      {
        "rel": "self",
        "href": "/client/questions/552a5e18f7e024ef02ed38a5"
      }
    ]
  }
]
```

B. Sans état

Le serveur a été conçu sans l'implémentation de sessions utilisateurs. Chaque requête Http se suffit à elle-même pour que le serveur puisse la traiter.

C. Client / Serveur

Ce principe consiste à limiter les rôles de chaque élément du réseau. Ainsi, notre partie client gère l'interface utilisateur, elle affiche les questions. Le SQUID, administre le cache. La partie serveur reçoit les requêtes et attaque mongoDB. Chaque entité n'assure donc qu'un seul et unique rôle

D. Interface uniforme

L'API utilise l'interface HTTP pour dialoguer avec les clients. Cette interface est composée de méthodes (GET, OPTIONS, POST, PUT etc....). De plus un soin particulier a été apporté pour respecter les codes statuts.

E. Layered System

Comme le montre notre schéma réseau, le serveur SQUID est au coeur de l'application. Chaque information passe par SQUID. Ainsi lorsqu'un client requête le serveur, cette requête transite d'abord par le SQUID qui l'examine et la transmet au serveur si besoin est. Le serveur répond ensuite au SQUID qui lui répond au client.

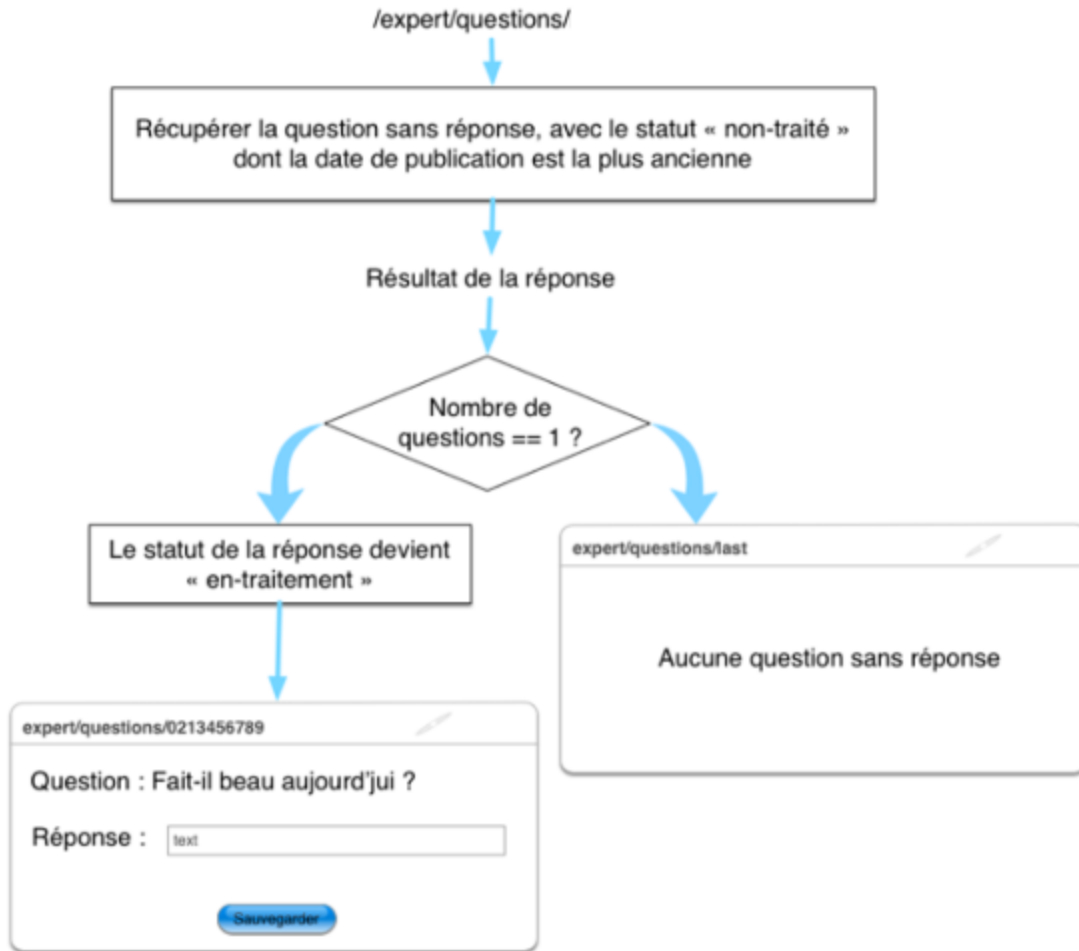
F. Mise en cache

La gestion du cache est gérée par le SQUID. Cependant l'API doit l'informer que certaines données sont cachables. Pour ce faire, le header de la réponse HTTP doit posséder l'information *Cache-control*.

Header d'une réponse

```
res.setHeader('Cache-Control', 'public, max-age=31557600000');
```

IV. Extension N experts



Ici le service doit garantir qu'une réponse ne sera fournie que par un seul client système expert:

- Le champ "statut" de la base de données permet de savoir si la réponse a bien été prise en compte.
- Dans le cas où le champ "statut" de la base de données est différent de "non traité", aucun autre client expert n'a accès à la question, donc aucune question ne sera disponible dans l'interface.
- Dans le cas où aucune question dont le champ "statut" de la base de données est "non traité", le statut devient "en traitement" par l'accès à l'écriture de la réponse par le client expert.

V. Sources du projet

Github : https://github.com/kimsavinfo/Interface_2037_Ready_for_inquiry

Vous avez à disposition dans le dossier **/docs** le support de la présentation orale ainsi que la documentation technique du service Web.