UFR SCIENCES & TECHNIQUES

UNIVERSITÉ DE FRANCHE-COMTÉ

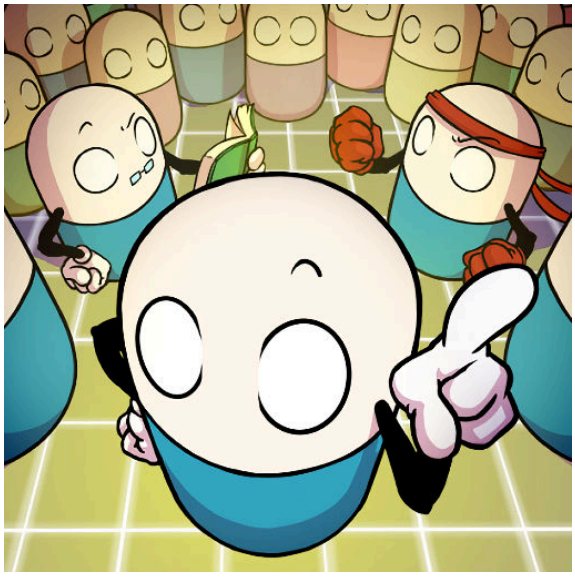
Licence 3 Informatique

UFR Sciences et Techniques de Besançon

Projet tutoré 2020 – 2021

Application d'apprentissage réflexe pour les étudiants pharmaciens

---



Nathanaël Houn

Valentin Perignon

François Poguet

Tuteur : M. Frédéric Dadeau

Tuteurs : M. Marc Pudlo et Mme. Anne-Laure Clairet

Remerciements

Nous voulons d'abord remercier notre tuteur Frédéric Dadeau pour son accompagnement tout au long du projet et pour les conseils prodigués, ainsi que pour sa disponiblité.

Merci à Anne-Laure Clairet et Marc Pudlo pour leur enthousiasme et leurs retours précieux sur notre travail, ainsi que les ressources pharmaceutiques nécessaires à l'application.

Merci et bravo à Yousif Al-Kes Ishaq, Romain Imberti et Victor Villemin pour leur application Travel Journal réalisée l'année dernière ; cette réussite a été un facteur déterminant dans notre motivation pour notre projet.

Nous remercions nos proches et amis qui ont testé l'application et nous ont fait des suggestions pour améliorer l'interface.

Enfin, mais pas des moindres, nous remercions chaleureusement Roxane Herbstmeyer pour toutes les animations et illustrations créées pour ce projet. Sans elle, *Guacamole* n'aurait été qu'une triste application de quiz, fade et sans vie ; c'est grâce à Roxane que nous pouvons être si fiers de montrer ce travail.

Table des matières

1. Introduction

Ce projet s'inscrit dans le module *Projet Tutoré* de la troisième année de licence Informatique à l'Université de Franche-Comté. Le travail réalisé s'est étalé sur plus de 6 mois, de septembre 2020 à mars 2021. L'objectif de ce module est de plonger les étudiants, par groupes de 2 ou 3, dans un cadre de développement d'un projet informatique à long terme en répondant à une demande, un besoin.

En septembre 2020, nous avons rencontré l'enseignant-chercheur Marc Pudlo qui nous a présenté un projet de création d'application de quiz destinée aux étudiants en pharmacie de Besançon, leur permettant d'apprendre tout en s'amusant. Nous avons, suite à ce rendez-vous, commencé à travailler sur une *Progressive Web App* qui sera déployée dans les prochains mois.

Dans un premier temps, nous présenterons plus en détail l'application à laquelle nous avons donné le nom de code *Guacamole*. Par la suite, nous décrirons la phase d'analyse permettant de comprendre nos choix d'architectures et de technologies. Nous aborderons, dans une troisième partie, le travail effectué, notre implémentation et certaines bibliothèques et outils utilisés. Enfin, nous parlerons du résultat obtenu et du travail nécessaire avant le déploiement.

2. Contexte & cahier des charges

La troisième année de Licence Informatique nous donne l'occasion de nous investir pendant l'année sur un projet tutoré. Notre trinôme cherchait un projet de développement axé technologies du web, avec vocation de rendre service aux utilisateurs : ce sujet nous a donc particulièrement séduit depuis le début.

Présentation de nos clients En octobre 2020, nous avons rencontré Marc Pudlo, enseignant-chercheur à l'UFR SMP [1] de l'université de Besançon, qui est à l'initiative de l'application. Il a été accompagné toute l'année d'Anne-Laure Clairet, pharmacienne au sein du pôle Pharmaceutique du CHRU de Besançon et enseignante à l'UFR SMP.

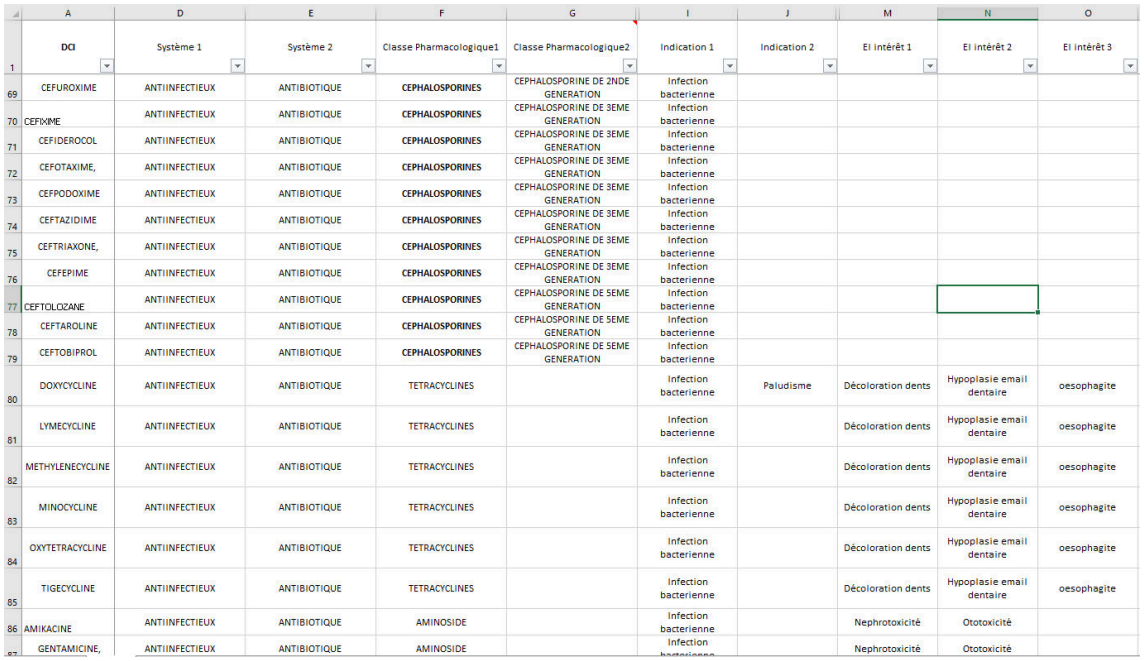
Précédente version de l'application L'an dernier, trois étudiants ont travaillé sur ce projet. Alexis Goy, Jérôme Benetruy et Thomas Flecy ont, tout comme nous, effectué ce travail dans le cadre du *Projet Tutoré*. Pour cette application, les étudiants ont choisi des technologies proches des nôtres : un client développé avec le *framework* React et un serveur réalisé avec Node.js.

En septembre, nous avons eu accès à leur rapport de projet et au dépôt GitHub, ce qui nous a permis d'accéder à leur analyse du projet. Depuis l'année universitaire 2019-2020, les besoins de nos clients ont évolué et les types de questions possibles également. Par exemple, la base de données de l'an dernier est capable d'enregistrer le dosage nécessaire d'une molécule, pour un enfant et pour un adulte, ce qui n'est plus nécessaire aujourd'hui. De plus, l'application était d'abord pensée pour fonctionner sur ordinateur, et devait pouvoir s'adapter à l'écran d'un smartphone. Or, nous souhaitions une application pouvant fonctionner principalement sur mobile, et pouvant s'adapter à un ordinateur. Pour ces raisons, nous avons fait le choix de ne pas repartir du projet existant.

Objectif de l'application Cette application d'apprentissage réflexe propose aux étudiants en faculté de pharmarcie un moyen ludique de réviser leurs cours, depuis leur téléphone portable ou un ordinateur, dans les transports en commun ou depuis leur canapé. Les notions exercées portent sur la reconnaissance des molécules à la base des médicaments, de leurs classes pharmacologiques, de leurs effets indésirables, et des maladies pour lesquelles elles sont indiquées (voir figure 2.1 pour un exemple des données brutes). Les étudiants peuvent se défier en duel pour une série de plusieurs blocs de questions, et celui ou celle qui a le plus de bonnes réponses au total remporte le duel.

Le développement de cette application rentrait dans le cadre du projet « Apprendre en geekant », lauréat d'un appel à projet RITM-BFC [2].

1. Santé, Médecine, Pharmacie
2. Réussir – Innover – Transformer – Mobiliser en Bourgogne-Franche-Comté



	A	D	E	F	G	I	J	M	N	O
1	DCI	Système 1	Système 2	Classe Pharmacologique1	Classe Pharmacologique2	Indication 1	Indication 2	EI intérêt 1	EI intérêt 2	EI intérêt 3
69	CEFUROXIME	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 2NDE GENERATION	Infection bacterienne				
70	CEFIXIME	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 3EME GENERATION	Infection bacterienne				
71	CEFIDEROCOL	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 3EME GENERATION	Infection bacterienne				
72	CEFOTAXIME,	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 3EME GENERATION	Infection bacterienne				
73	CEFPODOXIME	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 3EME GENERATION	Infection bacterienne				
74	CEFTAZIDIME	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 3EME GENERATION	Infection bacterienne				
75	CEFTRIAXONE,	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 3EME GENERATION	Infection bacterienne				
76	CEFEPIME	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 3EME GENERATION	Infection bacterienne				
77	CEFTOLOZANE	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 5EME GENERATION	Infection bacterienne				
78	CEFTAROLINE	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 5EME GENERATION	Infection bacterienne				
79	CEFTOBIPROL	ANTIINFECTIEUX	ANTIBIOTIQUE	CEPHALOSPORINES	CEPHALOSPORINE DE 5EME GENERATION	Infection bacterienne				
80	DOXYCYCLINE	ANTIINFECTIEUX	ANTIBIOTIQUE	TETRACYCLINES		Infection bacterienne	Paludisme	Décoloration dents	Hypoplasie email dentaire	oesophagite
81	LYMECYCLINE	ANTIINFECTIEUX	ANTIBIOTIQUE	TETRACYCLINES		Infection bacterienne		Décoloration dents	Hypoplasie email dentaire	oesophagite
82	METHYLENECYCLINE	ANTIINFECTIEUX	ANTIBIOTIQUE	TETRACYCLINES		Infection bacterienne		Décoloration dents	Hypoplasie email dentaire	oesophagite
83	MINOCYCLINE	ANTIINFECTIEUX	ANTIBIOTIQUE	TETRACYCLINES		Infection bacterienne		Décoloration dents	Hypoplasie email dentaire	oesophagite
84	OXYTETRACYCLINE	ANTIINFECTIEUX	ANTIBIOTIQUE	TETRACYCLINES		Infection bacterienne		Décoloration dents	Hypoplasie email dentaire	oesophagite
85	TIGECYCLINE	ANTIINFECTIEUX	ANTIBIOTIQUE	TETRACYCLINES		Infection bacterienne		Décoloration dents	Hypoplasie email dentaire	oesophagite
86	AMIKACINE	ANTIINFECTIEUX	ANTIBIOTIQUE	AMINOSIDE		Infection bacterienne		Nephrotoxicité	Ototoxicité	
87	GENTAMICINE,	ANTIINFECTIEUX	ANTIBIOTIQUE	AMINOSIDE		Infection bacterienne		Nephrotoxicité	Ototoxicité	

FIGURE 2.1 – Données chimiques manipulées

Fonctionnalités demandées Les premières rencontres avec nos clients ont été consacrées à la définition de notre cahier des charges. Les éléments suivants sont ressortis :

- Lancer un challenge entre deux étudiants, avec un rythme imparti (chacun dispose de 24 h pour jouer sa manche)
- Garder une trace du score de chacun
- Avoir une vue d'entraînement libre, où l'utilisateur pourrait recevoir des questions tant qu'il le souhaite et avoir le corrigé au fur et à mesure,
- Pouvoir choisir, au début du duel, le type de question sur lequel on va être interrogé
- Adapter l'application aux mobiles (iOS, Android)
- Pouvoir importer de nouvelles données
- Proposer une interface conviviale et ludique

Une fois le cahier des charges défini, nous avons pu commencer à réfléchir à l'implémentation.

3. Analyse & conception

Comme proposé dans le sujet, ce projet est basé sur les technologies du web habituelles (HTML, CSS, JavaScript et MySQL). Afin d'accélérer le développement et de ne pas réinventer la roue, nous avons aussi eu recours à un *framework* pour l'interface (ReactJS), ainsi qu'à quelques bibliothèques bien choisies.

3.1 Architecture logicielle client-serveur

Guacamole s'appuie sur une architecture client-serveur-base de données assez classique (fig. 3.1).

Le client (sur le téléphone de l'utilisateur) effectue des requêtes via une API [1] REST [2], basée sur le protocole HTTP [3]. Ces requêtes sont reçues par le serveur, qui va à son tour interroger la base de données, qui contient les informations. Une fois qu'il a récupéré les informations brutes, le serveur va les traiter et les mettre en forme avant de les renvoyer au client, toujours via l'API REST. Celui-ci va pouvoir se charger de les afficher à l'utilisateur.

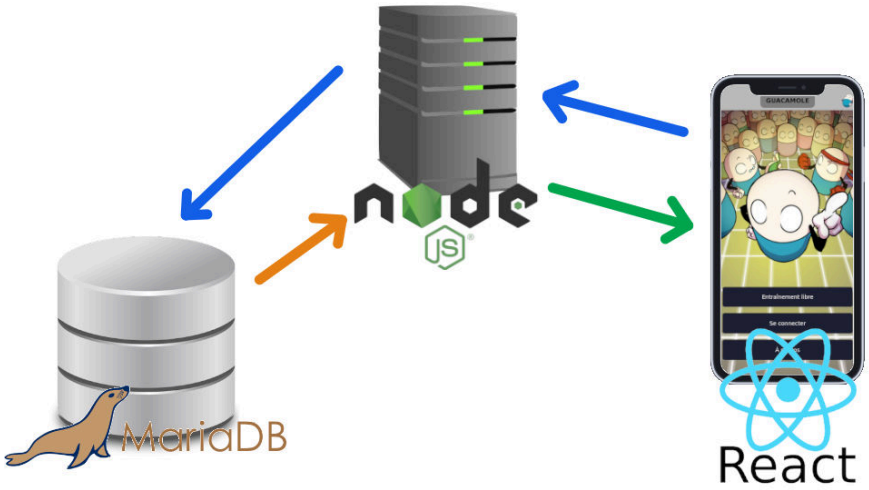


FIGURE 3.1 – Architecture client-serveur-base de données derrière Guacamole

1. *Application Programming Interface*, ou interface de programmation d'application est un ensemble de points de conversation qui sert de façade pour la communication entre deux logiciels (ici, le client et le serveur).
2. REST, pour *Representational State Transfer*, est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web, dont l'architecture client-serveur et l'absence de notion d'état lors du traitement de la requête.
3. Le protocole de communication HTTP est le protocole de communication utilisé sur le Web

3.1.1 Framework React et bibliothèques pour le client

Le développement d'interface web est un travail colossal, c'est pourquoi nous avons choisi le *framework*[4] ReactJS pour nous aider. ReactJS (ou React) permet de manipuler le DOM[5] via du code JSX, qui permet d'intégrer des nœuds HTML directement dans du code JavaScript.

React divise le logiciel en unités appelés composants (fig. 3.2), qui possèdent chacun des propriétés héritées (les `props`) et un état local modifiable (le `state`). Ces composants sont capables de se mettre à jour quand leur état ou leurs propriétés changent, et ainsi de réagir aux interactions de l'utilisateur.

React profite d'une documentation complète, ce qui nous a permis de le prendre en main rapidement, ainsi que d'un large écosystème. Nous sommes partis de la base *Create-React-App*, qui fournit un ensemble d'outils (mise en forme, détection d'erreur, compilation) pour une application React. Nous y avons ajouté des bibliothèques soigneusement choisies telles que `React useQuery` (abordée dans la partie 4.3.3), qui étaient plus rapides à intégrer plutôt que de développer toutes leurs fonctionnalités nous-mêmes.

La mise en forme de l'interface est prise en charge par du CSS, compilé à partir de SCSS[6].

```
01 | import React from "react";
02 |
03 | const Plural = ({ word, count, plural = null }) => {
04 |   const pluralWord = plural ? plural : `${word}s`;
05 |   return <>{count <= 1 ? word : pluralWord}</>;
06 | };
```

FIGURE 3.2 – Composant React simple, permettant la pluralisation

3.1.2 Bibliothèque Workbox pour le service-worker

Pour le développement de notre service-worker, dont nous parlerons plus en détail dans une prochaine partie (voir la section 4.3.2), nous utilisons Workbox, une bibliothèque développée par Google ainsi qu'une communauté de développeurs sur GitHub. Workbox contient de nombreux modules permettant d'écrire un service-worker rapidement tout en utilisant les bonnes pratiques.

Nous utilisons quatre modules de cette bibliothèque. Tout d'abord *worbox-core* permettant de configurer facilement le cache utilisé par l'ensemble des modules. Ensuite, *workbox-precaching*, *workbox-routing* et *workob-strategies* pour la mise en cache des fichiers. Enfin, *workbox-background-sync* afin d'autoriser l'utilisateur à faire des requêtes à l'API même lorsqu'il est hors-ligne.

Chaque module de la bibliothèque est disponible sous forme de paquet NPM.

4. Un *framework*, que l'on pourrait traduire par « infrastructure de développement », ou « cadriciel » comme proposé par l'Office Québécois de la langue française, est un ensemble cohérent de composants logiciels qui sert à créer les fondations d'une architecture logicielle.

5. Le *Document Object Model* est une représentation du document HTML composant une page Web classique.

6. SCSS est une extension du langage CSS qui permet, entre autres, une meilleure hiérarchie du code et l'utilisation de variables et de fonctions.

3.1.3 Framework Express avec Node.js pour le serveur

Notre partie serveur a été développée avec le même langage que le client, JavaScript, qui est exécuté via la plateforme Node.js. Notre serveur ayant pour but unique d'héberger l'API et de la lier avec la base de données, nous avons utilisé le *framework* Express permettant de faire de Node.js un serveur HTTP puissant.

Le serveur est divisé en *routes* (fig. 3.3), qui correspondent chacune à un point d'entrée de l'API. Chacune d'entre elle est reliée à une fonction qui va prendre en charge le traitement de la requête et faire les appels nécessaires à la base de données, avant de renvoyer la réponse au client.

```
01 | // Routes
02 | apiRouter.get("/status", ApiController.status);
03 |
04 | // ApiController.js
05 | export async function status(_, res) {
06 |   // Appel à la base de données
07 |   const version = await getSystemInformation("api_version");
08 |
09 |   res.sendResponse(200, {
10 |     status: "connected",
11 |     apiVersion: version,
12 |   });
13 | }
```

FIGURE 3.3 – Exemple d'une route Express simple

3.2 Conception de la base de données

Une grande partie de l'application repose sur la base de données, il était important dès le début du projet de modéliser les données de manière à pouvoir l'étendre facilement. La base de données contient deux parties principales : d'un côté les données propres à l'application (utilisateurs, duels...), et de l'autre les données chimiques, importées par les administrateurs et utilisées pour la génération de questions.

Un schéma relationnel était nécessaire pour modéliser ces données. De ce fait, nous avons utilisé le language SQL que nous maitrisons et avons fait le choix d'un système MariaDB [7],

3.2.1 Modèle de données des molécules et de leurs propriétés

Une grande partie de la réflexion à propos de la modélisation de la base de données portait sur les données chimiques. Il fallait trouver un moyen de stocker les molécules ainsi que leurs différentes propriétés et classifications, dans un schéma restant simple d'utilisation.

La molécule est l'élément central du modèle, elle appartient à une classe pharmacologique et à un système. Les classes et les systèmes sont hiérarchisés et modélisés par un arbre, chaque élément de la classification a un élément parent, sauf ceux au sommet de la hiérarchie.

Les molécules possèdent également des propriétés, qui peuvent avoir plusieurs valeurs. Par exemple, une molécule peut avoir plusieurs valeurs pour la propriété *Effet indésirable*.

7. MariaDB est un système de gestion de base de données, étant un fork communautaire et libre de MySQL.

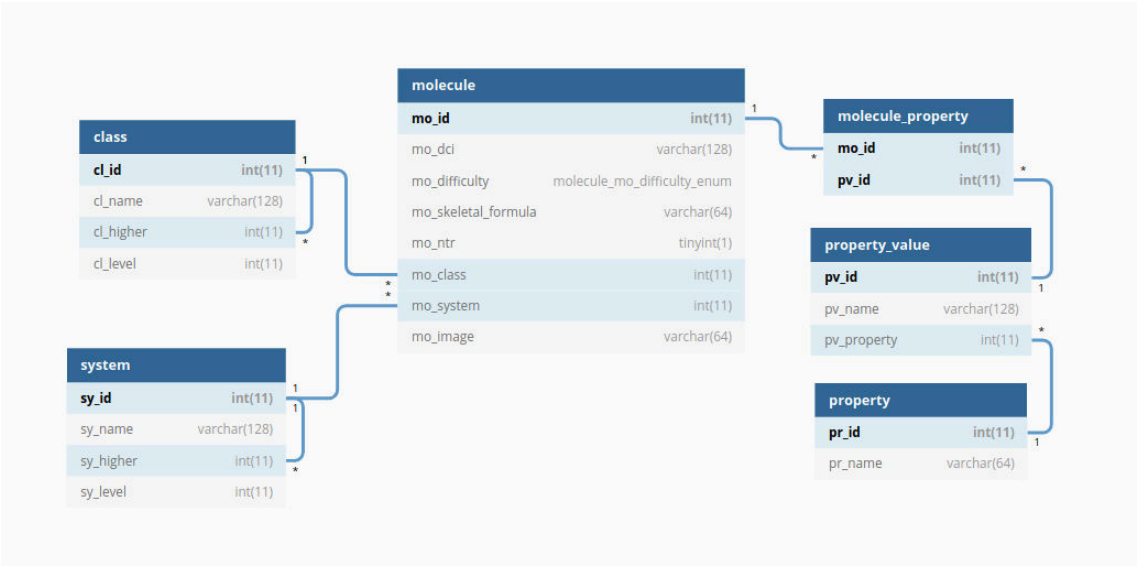


FIGURE 3.4 – Modèle de données chimiques

3.2.2 Modèle de données des utilisateurs et des duels

Le reste des données de l'application suit un schéma beaucoup plus classique, l'élément principal est l'utilisateur. Celui-ci possède notamment des *refresh tokens*[8]. Une table fait la liaison entre l'utilisateur et le duel pour garder entre autres les réponses aux questions du duel.

Une table *server_ information* contient les différentes variables globales de l'application, comme la version actuelle de la base de données, ou encore les variables de configuration des duels.

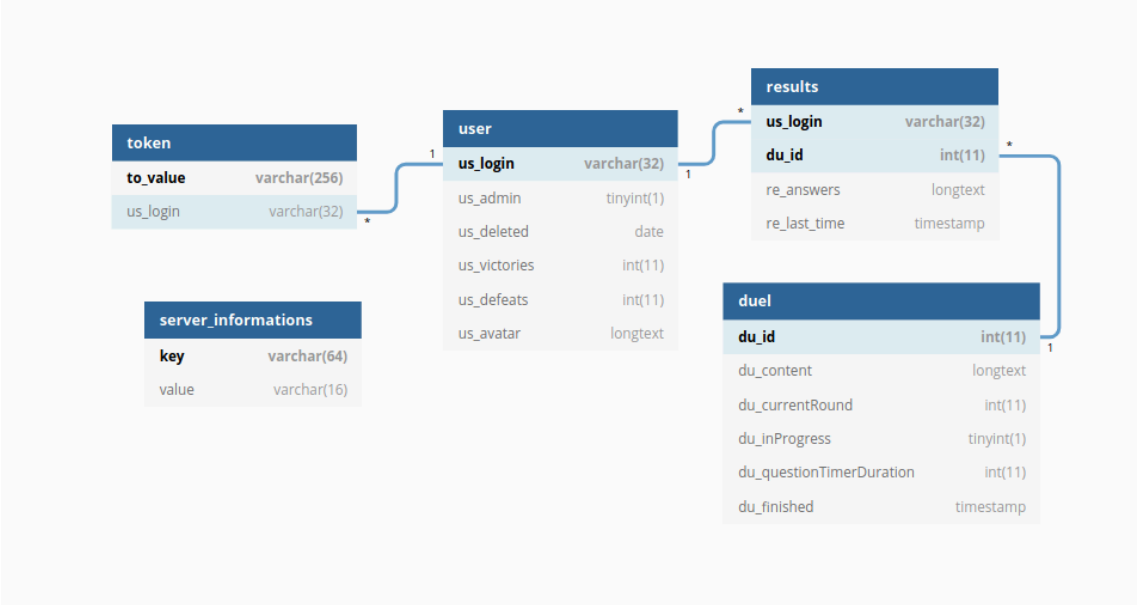


FIGURE 3.5 – Modèle de données utilisateurs

8. Voir la section 4.3.1 sur l'authentification, page 18.

3.2.3 Automatisation des mises à jour

Pour simplifier le travail en équipe, nous avons mis en place un système de mise à jour automatique de la base de données. Cela permet de travailler sur une version locale de l'application et de la base de données, tout en suivant les évolutions que chacun apporte sur le modèle de données. De plus, cela retire les étapes manuelles lors de la mise à jour du serveur en production. [9]

Concrètement, il suffit de fournir au serveur un script SQL de mise à jour, et au prochain lancement, la base de données est créée ou mise à jour si besoin (fig. 3.6).

```
01 | Connected to database!
02 | Creation of database...
03 | -> Database created!
04 | Update database from 2021-01-08 to 2021-01-14...
05 | Update database from 2021-01-14 to 2021-01-16...
06 | Update database from 2021-01-16 to 2021-01-21...
07 | -> Database updated!
08 | Server is running on port 5035.
```

FIGURE 3.6 – Démarrage du serveur

9. Voir la section 4.1.3 sur le déploiement continu, page 12

4. Réalisation

Une fois l'analyse terminée, nous avons pu commencer la réalisation à proprement parler de Guacamole. Nous aborderons dans ce chapitre la mise en place de notre environnement de travail, puis plusieurs points intéressants du développement de la partie serveur et de la partie client de l'application.

4.1 Outils utilisés et méthodes de travail

Un projet de cette envergure nécessite une organisation et des outils adéquats. Nous nous sommes approchés des méthodes agiles, que nous avons adaptées à nos besoins tout au long du projet.

4.1.1 Git et GitHub, nos meilleurs alliés

Un VCS [1] et son écosystème étaient indispensables pour collaborer sur le code : nous avons choisi git et GitHub, car nous les connaissions bien.

Github nous permet de créer des tickets (ou *Issues*) gardant la trace des bugs à résoudre et des fonctionnalités à implémenter. Ces tickets sont rangés dans notre Github Project (fig. 4.1), ce qui permet de voir en un coup d'oeil l'avancée du projet et de n'oublier aucune tâche.

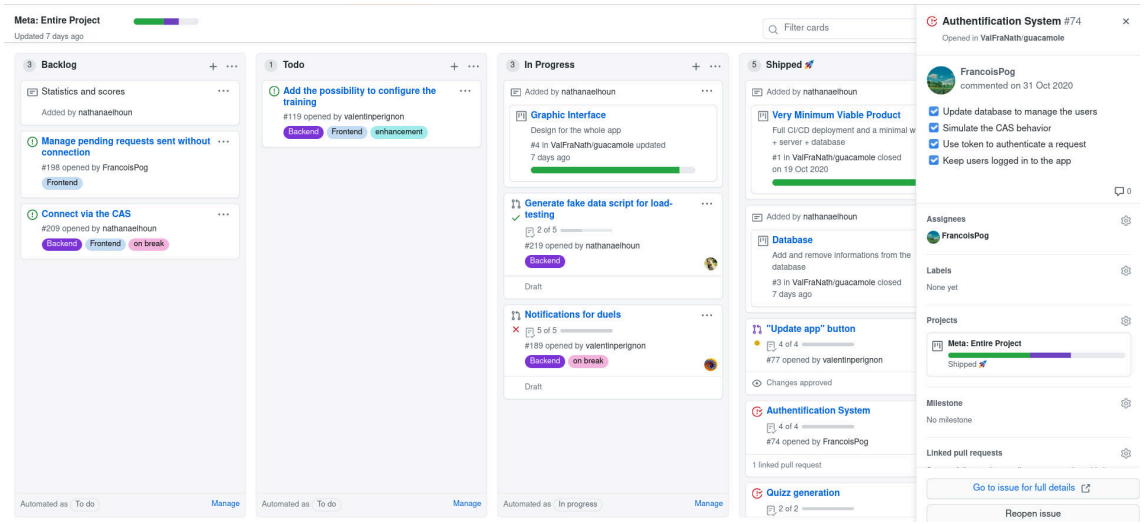


FIGURE 4.1 – GitHub Project, notre tableau Kanban, au milieu du projet

1. *Version Control System*, ou Logiciel de gestion de version

4.1.2 Processus de développement avec les *Pull Requests*

Une fois la fonctionnalité implémentée par un membre de l'équipe, nous mettons en route le mécanisme des *Pull Requests* (ou *PR*). Quand un développeur termine une tâche, plutôt que de pousser le code directement dans la branche principale du dépôt [2], il ouvre une *PR* (fig. 4.2). Github lancera alors les tests logiciels, ainsi qu'un *linter* [3]. Mais surtout, avant d'être validée, la *PR* va être relue par un autre développeur de l'équipe, afin de faire des remarques sur le style ou le fonctionnement.

La *PR* terminée sera fusionnée à la branche principale sous forme d'une seul *commit* [4] (*squash merge*), créant ainsi un historique propre dans cette branche.

Ce fonctionnement, mis en place dès le début du projet, s'est révélé très agréable pour travailler : d'une part, nous sommes plus confiants dans le code produit car il a été relu et les premiers bugs ont été corrigés, et d'autre part chacun a une meilleure vue de l'ensemble de l'application.

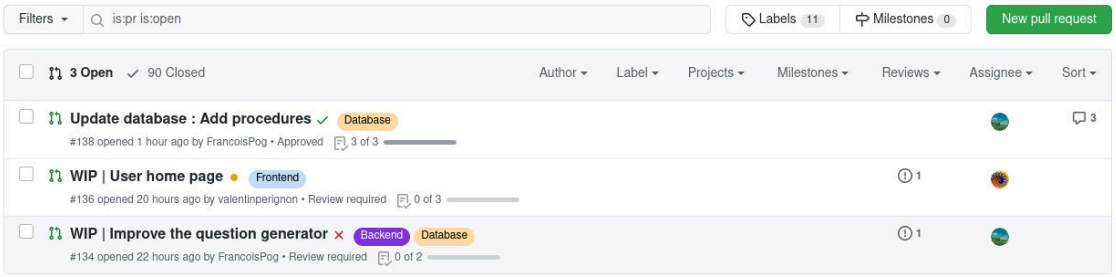


FIGURE 4.2 – Liste des *Pull Requests* en attente dans GitHub

4.1.3 Intégration et déploiement continus (*CI/CD*)

Pour ce projet, nous souhaitions nous installer un environnement de développement confortable depuis le début. L'intégration continue (ou *CI*) via GitHub Actions permet de lancer à chaque version les tests évoqués précédemments sur le code d'une *PR*, ce qui permet de s'assurer qu'aucune partie de l'application ne casse, y compris sur une configuration différente de la nôtre (nos environnements de développement sont sous MacOS, Windows et ArchLinux, et la *CI* sous Ubuntu). En plus, cela permet d'automatiser des tâches comme la compression des nouvelles images.

Le déploiement continu (ou *CD*) nous assure un accès constant à la dernière version de l'application pour pouvoir la présenter lors des réunions, mais aussi la faire tester par des proches. D'un point de vue technique, il est déclenché par un webhook envoyé par GitHub pour chaque *push* sur notre branche principale.

Notre VPS [5] démarre un script qui *pull* [6] le code depuis GitHub, installe les dépendances

2. Branche : rien à voir avec de la botanique, elle correspond à une version du code développée en parallèle avec les autres

3. Logiciel d'analyse statique du code qui permet de relever des erreurs de style, des bugs potentiels et plus encore

4. Dans git, un *commit* est une sauvegarde d'une version du code, accompagnée d'un court message décrivant les changements

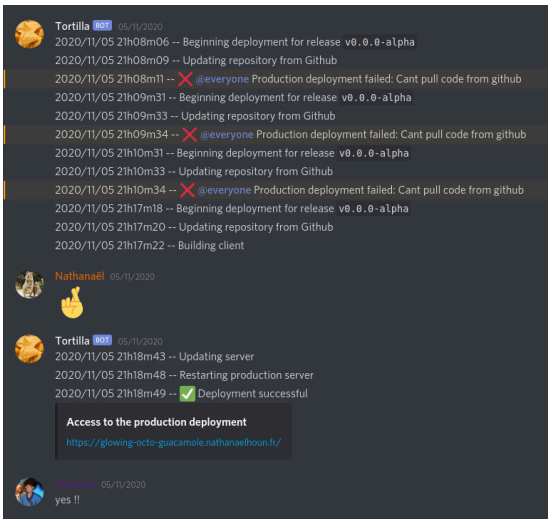
5. *Virtual Private Server*, Serveur Privé Virtuel, hébergeant notre application et la rendant disponible sur Internet

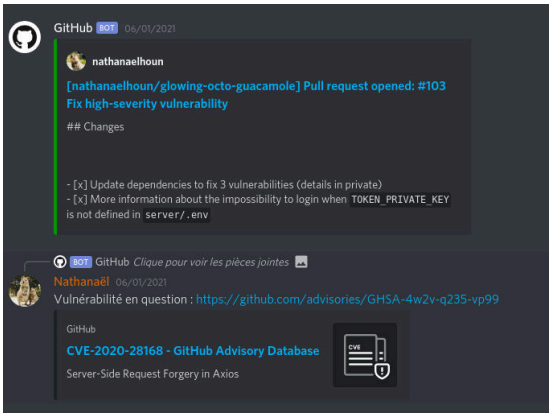
6. Avec git, *pull* est l'opération de récupérer le code depuis le dépôt distant et de le fusionner avec le

nécessaires, compile le client et redémarre avec cette dernière version du serveur.

Cette installation nous a permis de nous assurer que l'application était facilement déployable. Cela s'est montré utile lorsque notre serveur a brûlé avec le reste de son *datacenter* dans les dernières semaines du projet : nous avons pu réinstaller tout ce système sur un nouveau serveur en moins de 2 heures.

Enfin, puisque nous utilisons Discord [7] (fig. 4.3) pour communiquer dans l'équipe, nous y avons rajouté des notifications pour être tenu au courant de l'avancée du déploiement et du statut des *PR*, encore une fois via webhook. Ainsi, toutes les informations y sont centralisées.





(a) Discuter dans Discord nous permet de communiquer des informations non publiques

FIGURE 4.3 – Notifications directement dans notre espace Discord

4.2 Des données aux duels

Guacamole génère automatiquement des questions aléatoires à partir des données qui lui sont fournies. Pour cela, il faut traiter les données, de leur importation jusqu'à la génération de duels complets.

4.2.1 Importation des données

Certaines données de l'application doivent pouvoir évoluer, c'est pourquoi une interface d'importation est nécessaire. Les données importées par les administrateurs sont : les molécules, les images des structures chimiques et les utilisateurs. Pour les données textuelles, nous utilisons le format CSV [8].

Le processus d'importation est le même pour les molécules et les utilisateurs (fig. 4.4) . D'abord le fichier est lu et transformé en un objet JavaScript exploitable. Si des erreurs apparaissent lors de la lecture, elles sont retournées sans continuer l'importation. Le fichier doit respecter certaines contraintes pour être valide, et importé.

Ensuite le fichier est analysé pour détecter de potentielles incohérences dans les don-

dépôt local

7. Logiciel de discussion instantanée, texte et audio
8. *Comma-Separated Values*

nées. L'analyse porte principalement sur la validité et la cohérence des valeurs (détection des doublons, hiérarchies bien structurées, etc.). La liste des potentiels avertissements est retournée.

Si l'utilisateur confirme l'importation, les données importées remplacent les anciennes dans la base de données. Le fichier est sauvegardé sur le serveur, l'utilisateur peut télécharger le dernier fichier importé.

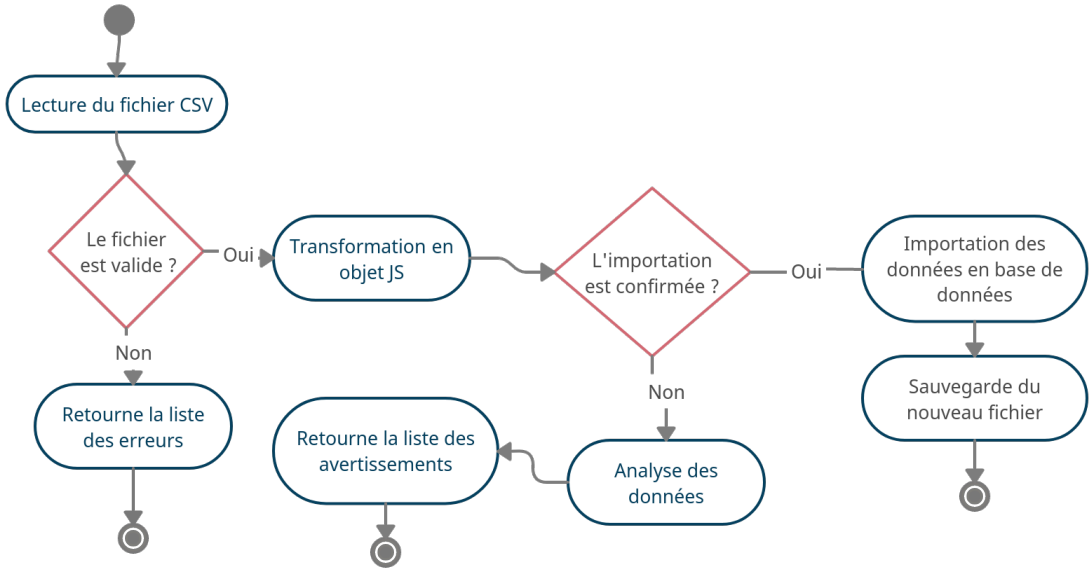


FIGURE 4.4 – Processus d'importation de fichier CSV

Pour les images, le processus d'importation est semblable (fig. 4.5). Dans un premier temps, les fichiers sont analysés, principalement à propos de l'existence de la molécule correspondante. Si l'utilisateur confirme l'importation, les images remplacent les anciennes, et sont liées en base de données aux molécules correspondantes.

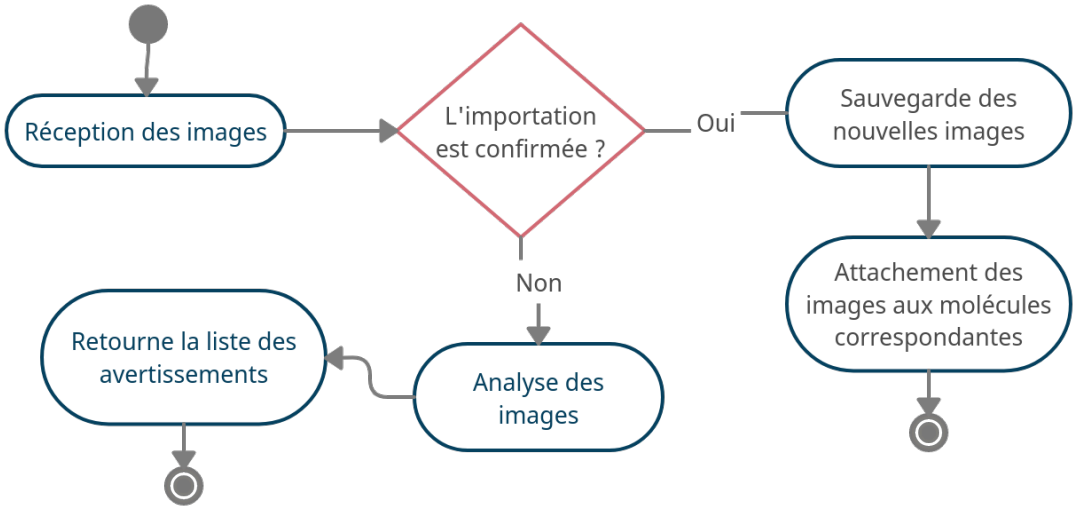


FIGURE 4.5 – Processus d'importation des images

4.2.2 Génération aléatoire des questions

Les questions du quiz sont générées aléatoirement à partir des données importées par les administrateurs. Il existe plusieurs types de questions, chacun portant sur une propriété différente des molécules. Une question est représentée par un sujet, une bonne réponse et trois mauvaises. Le type permet d'afficher la question correctement du côté client.

Pour chaque type, un script SQL génère une question. Le serveur exécute ce script, puis s'occupe de la mettre en forme (création de l'énoncé, etc). Tous les scripts suivent à peu près les mêmes étapes, nous allons détailler le script permettant de générer une question pour laquelle il faut trouver la classe correspondant à une molécule donnée.

Les scripts sont faits de manière à toujours retourner une question, s'il est possible de la créer. Autrement dit, si le script n'arrive pas à générer une question, c'est qu'il n'y a pas assez de données dans la base.

La première étape est la plus importante, on crée une table temporaire dans laquelle on va joindre à chaque molécule toutes les classes auxquelles elle appartient (fig. 4.6a). Pour cela nous avons utilisé les *CTE*[9], un système de requêtes récursives, permettant de traiter des modèles hiérarchiques. Nous en avions besoin ici car le nombre de niveaux dans la hiérarchie des classes peut varier, nous ne pouvons donc pas gérer ce cas avec un nombre défini de jointures.

9. *Common Table Expressions* : Plus d'information sur les CTE.

FIGURE 4.6 – Script de génération aléatoire d'une question

```sql
CREATE TEMPORARY TABLE classes_by_molecule(
    mo_id int(11),
    mo_dci varchar(256),
    cl_id int(11),
    cl_name varchar(256),
    cl_higher int(11),
    cl_level int(11)
);

INSERT INTO classes_by_molecule(
    WITH RECURSIVE classification AS(
        SELECT cl_id,
            cl_name,
            cl_higher,
            cl_level,
            molecule.mo_id,
            molecule.mo_dci
        FROM class JOIN molecule
            ON mo_class = cl_id

        UNION ALL

        SELECT parent.cl_id,
            parent.cl_name,
            parent.cl_higher,
            parent.cl_level,
            c.mo_id,
            c.mo_dci
        FROM class parent INNER JOIN classification c
            ON parent.cl_id = c.cl_higher
    )
    SELECT  mo_id,
        mo_dci,
        cl_id,
        cl_name,
        cl_higher,
        cl_level
    FROM classification
    ORDER BY  mo_id, cl_level
);
```

(a) Jointure des molécules à leurs classes

Après cette étape nous choisissons la classe qui sera la bonne réponse (fig. 4.6b). Pour pouvoir être une bonne réponse, il faut s'assurer qu'il existe au moins 3 autres classes appartenant à la même classe supérieure qu'elle, pour avoir au moins 3 mauvaises réponses possibles.

```sql
SET @class = (  SELECT cl_id
                FROM classes_by_molecule AS C1
                WHERE 3 < ( SELECT COUNT(DISTINCT cl_id)
                            FROM classes_by_molecule AS C2
                            WHERE (C1.cl_higher = C2.cl_higher
                            OR (C1.cl_higher IS NULL
                                AND C2.cl_higher IS NULL))
                            AND C1.cl_id <> C2.cl_id)
                ORDER BY RAND()
                LIMIT 1 );
```

(b) Définition de la bonne réponse

Après ça nous choisissons une molécule appartenant à cette classe (fig. 4.6c).

```
51 | SET @molecule = (   SELECT mo_id
52 |                     FROM classes_by_molecule
53 |                     WHERE cl_id = @class
54 |                     ORDER BY RAND()
55 |                     LIMIT 1);
```

(c) Définition de la molécule

Une fois le sujet et la bonne réponse définis, il ne reste qu'à choisir 3 mauvaises réponses, ici 3 classes appartenant à la même classes supérieure que la bonne réponse (fig. 4.6d). Nous utilisons le niveau de la classe dans la hiérarchie pour gérer le cas où cette classe serait au sommet de la hiérarchie.

```
56 | SET @level = (  SELECT cl_level
57 |                 FROM class
58 |                 WHERE cl_id = @class);
59 |
60 | SELECT DISTINCT (   SELECT mo_dci
61 |                     FROM molecule
62 |                     WHERE mo_id = @molecule
63 |                 ) AS subject,
64 |                 (   SELECT cl_name
65 |                     FROM class
66 |                     WHERE cl_id = @class
67 |                 ) AS good_answer,
68 |                 cl_name AS bad_answer
69 | FROM classes_by_molecule AS C
70 | WHERE cl_id <> @class
71 | AND ((@level > 1 AND C.cl_higher = (SELECT cl_higher
72 |                                     FROM class
73 |                                     WHERE cl_id = @class))
74 | OR (@level = 1 AND C.cl_level = 1))
75 | ORDER BY RAND()
76 | LIMIT 3;
```

(d) Définition des trois mauvaises réponses

Une fois le script exécuté, le serveur reçoit les données nécessaires à la construction de la question, il n'a plus qu'à la mettre en forme.

L'avantage de cette méthode est de séparer chaque type de question dans un script, ce qui donne plus de clarté au niveau du serveur. Un seul appel à la base de données est donc nécessaire pour générer une question.

4.2.3 Création et gestion des duels

Les duels constituent le coeur de l'application, leur implémentation a été faite dans l'optique de faciliter l'ajout de nouvelles fonctionnalités à l'avenir. Leur contenu est constitué d'un ensemble de manches, qui sont des ensembles de questions. Le nombre de manches et le nombre de questions sont configurables dans l'espace administrateur. Lors de leur création, toutes les questions sont générées et la manche courante est initialisée à 1. Le contenu d'un duel n'est plus modifié après la création.

Les joueurs jouent tour à tour les manches du duel, en envoyant leurs réponses au serveur, elles sont stockés dans la table *results*[10]. C'est le serveur qui, lors des appels

10. Voir la figure 3.5 sur le modèle de données des duels, page 9.

à l'API, s'occupe de filtrer les informations, en fonction de l'état du duel[11]. Il renvoie toutes les informations des manches terminées, y compris les réponses de chaque joueur, les questions de la manche courante et uniquement le type des questions pour les manches à venir.

Nous avons fait le choix d'envoyer les bonnes réponses avec les questions courantes à jouer, dans le but de limiter les appels au serveur pendant que le joueur joue le duel. Nous avons priorisé la fluidité, après réflexion avec nos tuteurs, plutôt que la sécurisation complète des réponses.

Un script s'exécutant régulièrement sur le serveur supprime les duels terminés depuis plusieurs jours (la durée est configurable). Le délai de réponse à un duel étant limité, un autre script s'occupe de faire perdre la manche aux joueurs qui n'ont pas joué dans les 24h après le début de la manche.

4.3 Communication client-serveur

Pour fonctionner, l'application fait de nombreuses requêtes réseaux, que ce soit pour obtenir des fichiers JavaScript, des images ou faire des appels à l'API. En premier lieu, ces requêtes ont besoin d'être authentifiées pour les utilisateurs connectées. Ensuite, proposer une application fluide à nos utilisateurs nécessite de la mise en cache accélérer ces appels coûteux. Voici comment nous avons procédé pour Guacamole.

4.3.1 Authentification des requêtes

Le système d'authentification de l'application est basé sur des jetons[12], en particulier des jetons d'accès (*access tokens*) et des jetons de rafraîchissement (*refresh tokens*). Pour les générer, nous avons utilisé le package *JSON Web Token*.

Les jetons d'accès permettent d'authentifier les requêtes API envoyées par le client, ils sont stockés dans le *localStorage* du navigateur. Dans un premier temps nous n'utilisions que des jetons d'accès avec une longue durée de vie. Ces derniers pouvant être déchiffrés sans interroger la base de données, l'authentification était rapide. Le problème de ce système est l'impossibilité de révoquer des jetons, ce qui pose des problèmes de sécurité.

C'est pour cette raison que nous avons introduit les jetons de rafraîchissement. Ces derniers sont stockés en base de données et permettent de générer de nouveaux jetons d'accès, qui ont maintenant une très courte durée de vie (de l'ordre de 10 minutes).

Combiner ces deux types permet d'avoir un compromis entre les avantages et les inconvénients de chacun. Les jetons d'accès permettent une authentification rapide et les jetons de rafraîchissement assurent la possibilité de les invalider.

Lorsque le client se connecte au serveur, il reçoit un jeton d'accès et un jeton de rafraîchissement (fig. 4.7).

11. Voir la représentation JSON d'un duel (fig. 7.1) en annexe, page 33
12. Plus d'informations sur le système de *refresh tokens* : What are refresh tokens ?

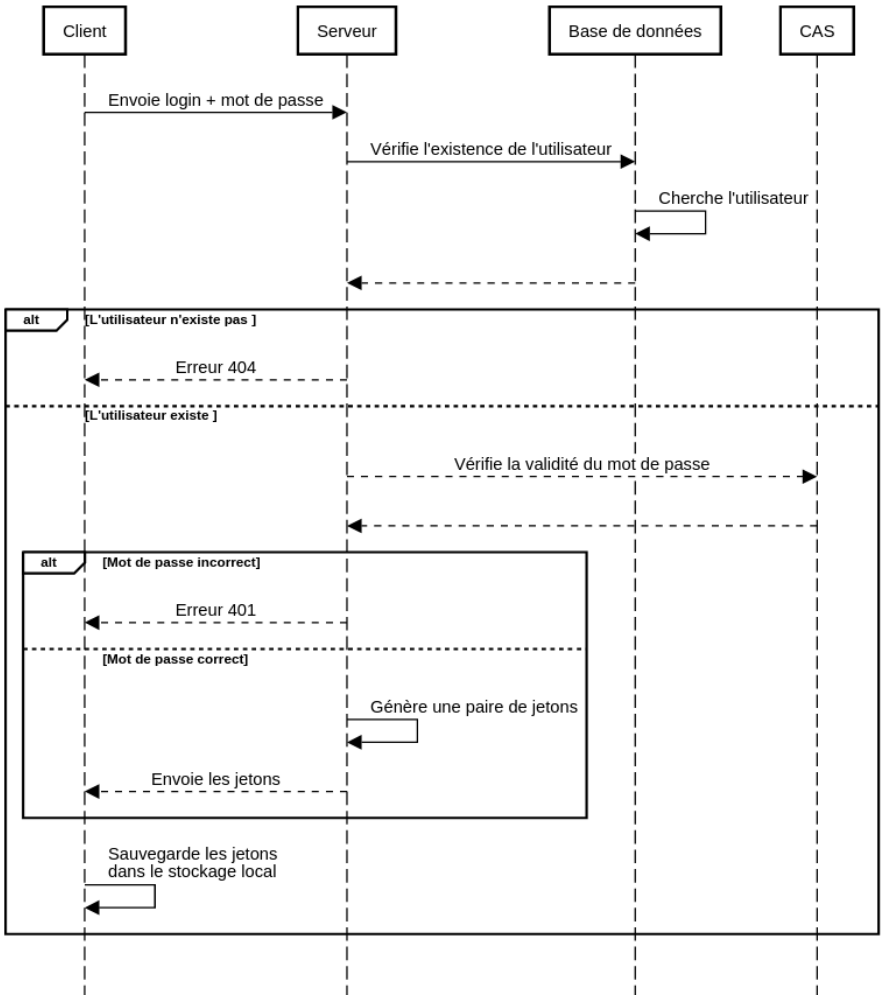


FIGURE 4.7 – Connexion d'un utilisateur

Ensuite, il ajoutera le jeton d'accès dans l'en-tête de chaque requête pour permettre leur authentification. Si ce jeton est expiré, le serveur renverra une erreur 401 : dans ce cas le client effectue automatiquement une requête de rafraîchissement [13], dans laquelle il enverra le jeton de rafraîchissement, et le serveur lui renverra un nouveau jeton d'accès (fig. 4.8).

13. Voir l'implémentation sur la figure 7.3 en annexes, page 34.

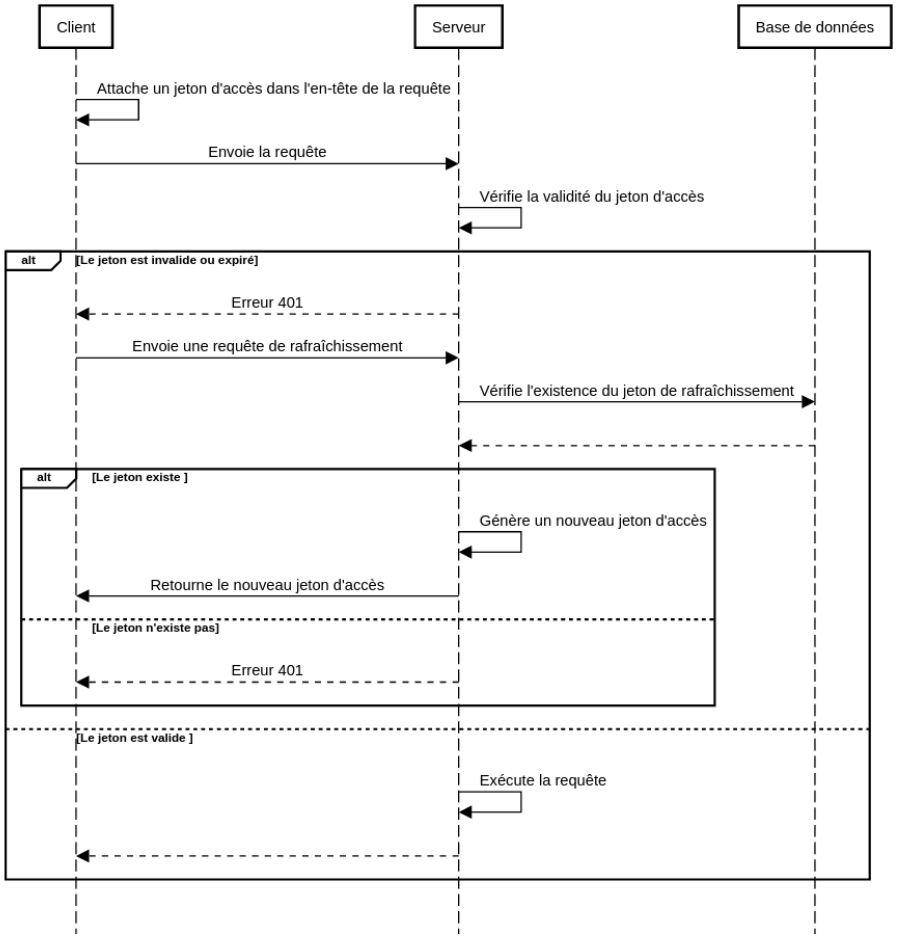


FIGURE 4.8 – Authentification d'une requête

Lorsque le client se déconnecte, le serveur supprime le jeton de rafraîchissement de la base de données, ainsi il ne sera plus valide, et le client devra se reconnecter (fig. 4.9) .

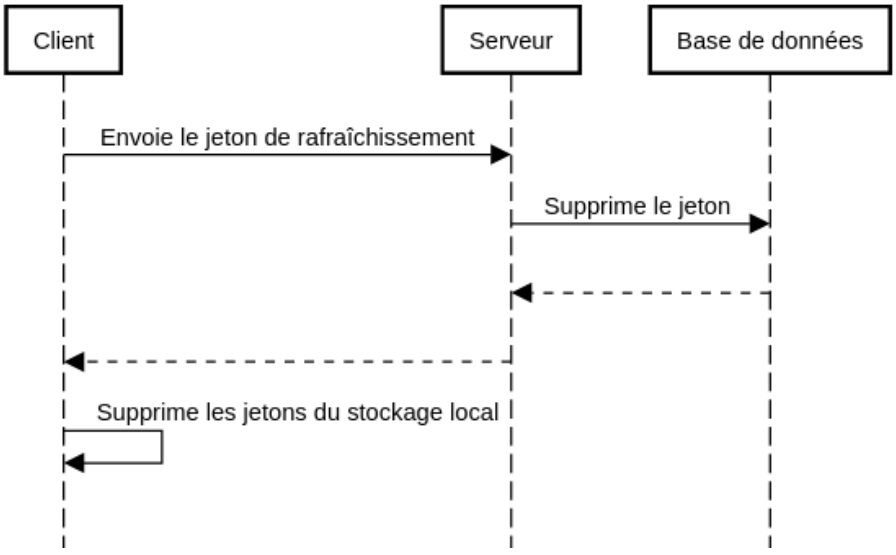


FIGURE 4.9 – Déconnexion de l'utilisateur

4.3.2 Mise en cache des fichiers avec le service-worker

Les Progressive Web Apps peuvent utiliser un service-worker. Il s'agit d'un script JavaScript que le navigateur exécute en arrière-plan, indépendamment des pages web de l'application.

Lorsque le client lance pour la première fois l'application, le service-worker s'installe et s'active (fig. 4.10). Au cours de la première étape, le script met en cache l'ensemble des fichiers HTML, JavaScript, CSS et des images nécessaires au client qui se trouvent dans le dossier `/static` (fig. 4.12). La bibliothèque Workbox fournit une variable nommée `__WB_MANIFEST` qui est un tableau contenant l'ensemble des fichiers à mettre en cache. Cette étape est réalisée grâce au module *worbox-precaching*. Par la suite, le service-worker agit comme un proxy entre l'utilisateur et le réseau. Chaque fois que le client fait une requête sur l'un de ces fichiers, le service-worker répondra avec la donnée mise dans le cache nommée `précache`.

```
01 | /*
02 | * Precaching
03 | * When the service-worker is installed, Workbox precaches all /static files
04 | */
05 | precacheAndRoute(self.__WB_MANIFEST);
```

FIGURE 4.10 – Mise en cache des fichiers lors de l'installation du service-worker

Pour les requêtes (hors celles faites à l'API) qui ne concernent pas les fichiers mis en cache dès le début, nous appliquons une autre stratégie. Cette stratégie, appelée *Stale While Revalidate* en anglais (fig. 4.13, dans ce cas il n'y a pas de durée de péremption), consiste à fournir dans un premier temps une réponse depuis le cache. Pendant ce temps, le service-worker fait une requête sur le réseau pour obtenir la ressource et met à jour le cache (fig. 4.11). Cependant si la ressource n'était pas présente, le script la retourne depuis le réseau et l'enregistre.

```
01 | /**
02 |  * Fetch resources (GET requests)
03 |  * Workbox responds with the files in the cache, and then updates them through
       the network
04 |  * API requests are not cached
05 |  */
06 | const matchRouteToCache = ({ url }) => !url.pathname.startsWith("/api");
07 | registerRoute(matchRouteToCache, new StaleWhileRevalidate());
```

FIGURE 4.11 – Stratégie Stale While Revalidate pour les requêtes hors API

Cette technique permet à l'utilisateur de toujours garder une application à jour, sans manipulation de sa part, tout en accédent aux ressources le plus rapidement possible.

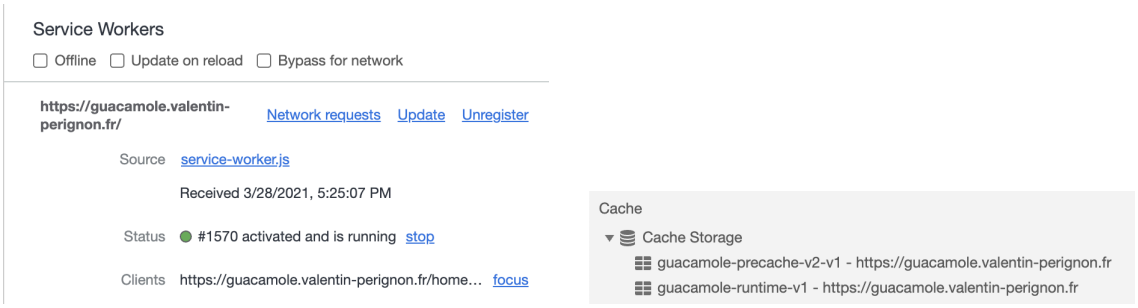


FIGURE 4.12 – Un service-worker activé dans Google Chrome et les caches pour les fichiers précachés et pour les autres ressources

4.3.3 Mise en cache des requêtes API avec React useQuery

Besoins différents du service-worker Les requêtes pour l'API ne peuvent pas être gérées de la même manière : les réponses changent régulièrement, on ne peut donc pas les mettre en cache indéfiniment comme on le fait avec les images.

Cependant, refaire les requêtes à chaque chargement de la page ralentit considérablement la navigation dans l'application, notamment les retours en arrière. D'autres complications existent : il faudrait éviter de lancer deux fois la même requête en parallèle, automatiquement rafraîchir certaines requêtes au bout d'un temps donné, automatiquement effacer une donnée mise en cache au bout d'un certain temps.

Deux bibliothèques réalisent toute cette gestion et s'intègrent parfaitement avec React : `React useQuery` et `SWR`[14]. Nous nous sommes tournés vers la première car son API est plus complète, et elle propose en expérimental un mode hors-ligne qui permettra, plus tard, de garder du cache d'API d'un lancement de l'application au suivant.

Utilisation concrète de useQuery Intégrer une nouvelle bibliothèque dans notre application, déjà bien avancée, a demandé un travail important de refactorisation du code. Voilà ce que cette réécriture donne sur le composant React `DuelCreate`, qui propose une liste d'utilisateurs à défier.

Le composant tel qu'écrit au départ se trouve dans la figure 7.2a (en annexe page 35). On y retrouve la fonction React `componentDidMount()`, qui s'exécute une fois le composant affiché à l'écran. Dans cette fonction, on exécute deux requêtes chaînées : une pour obtenir la liste des utilisateurs, la suivante pour obtenir la liste des duels en cours. On filtre ensuite la première liste avec la deuxième, pour obtenir les utilisateurs que l'on peut défier.

Pour commencer à utiliser `useQuery`, il a d'abord fallu réécrire le composant sous forme fonctionnelle (et non sous forme de classe). En effet, seuls les composants fonctionnels de React supportent les *hooks* personnalisés[15] qui viennent avec `useQuery`.

La première réécriture du composant sous forme fonctionnelle (dans la figure 7.2a) ne fait pas apparaître d'avantage clair, ce sont purement des changement de style pour l'instant. Mais maintenant, nous allons intégrer `UseQuery` dans notre composant (fig. 7.2c).

14. Pour *Stale While Revalidate*, concept introduit par la RFC 5861 qui consiste à servir les données du cache (« périmées ») tout en cherchant les données à jour, pour finalement afficher ces données à jour. Voir la figure 4.13 qui détaille la procédure.

15. Les *hooks* personnalisés (*Custom Hooks* en anglais) sont des fonctions particulières de React qui permettent d'isoler et de réutiliser la logique du code dans plusieurs composants. En apprendre plus dans la documentation officielle.

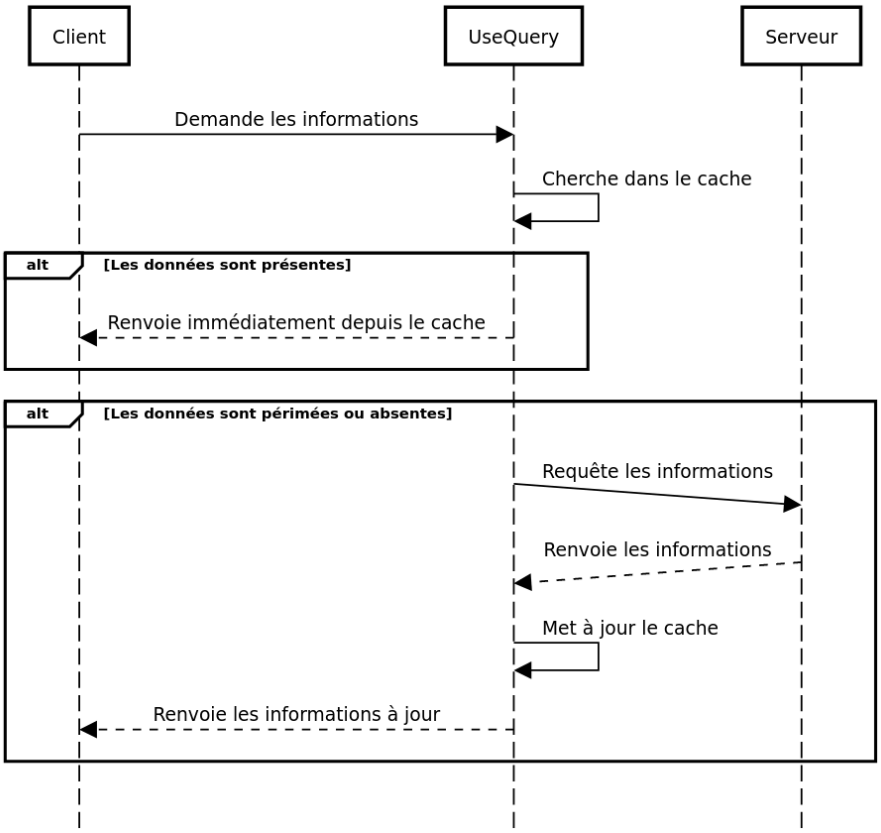


FIGURE 4.13 – Principe du *Stale While Revalidate*, implémenté par `useQuery` et `SWR`

`useQuery` identifie les requêtes par un identifiant (ici, `["users", "challengeable"]`), et stocke la date de dernière mise à jour avec le résultat de la requête, afin de savoir quand la rafraichîr automatiquement en arrière-plan.

Le composant final est beaucoup plus concis, et ne contient plus que du code relié à l'affichage. La requête est gérée dans une fonction `getChallengeableUsers` partagée entre les composants. Comme la même fonction servira toujours pour la requête `["users", "challengeable"]`, on peut même la configurer au niveau global (fig. 7.2d).

Le temps de mise en cache de cette requête est assez long (1 heure). Il faut cependant pouvoir la rafraîchir immédiatement à certaines occasions, par exemple quand on lance un duel. `useQuery` propose aussi cela : lors de la création du duel, il suffit de faire un appel à `queryClient.invalidateQueries(["users", "challengeable"])`, et la requête sera marquée comme *stale* immédiatement.

Le dernier avantage de `useQuery` est qu'il renvoie aussi l'état de la requête (en chargement, erreur, réussie) dans le composant. Cela permet d'intégrer très simplement une animation de chargement (fig. 7.2c) dans tous nos composants : la touche finale pour une navigation fluide !

4.3.4 Autoriser les requêtes hors-ligne

En l'état notre application n'est pas 100% fonctionnelle hors-ligne. Nous avons tenu, cependant, à apporter le plus de fonctionnalités possible pour que Guacamole dispose de tous les outils nécessaires pour cette adaptation. Pour cette raison, nous avons intégré

l'API Background Sync à notre service-worker. Cette API permet, lorsque l'utilisateur ne dispose d'une connexion stable ou que le serveur n'est pas joignable, de différer les requêtes en attendant de retrouver le réseau (fig. 4.14).

Cette API est encore récente, apparue en 2019 elle n'est pas encore implémentée dans tous les navigateurs. C'est pourquoi nous utilisons le module `workbox-background-sync` de Workbox. Si l'API n'est pas disponible sur le navigateur, le module tentera lui-même d'envoyer les requêtes en attente jusqu'à ce que le réseau soit à nouveau disponible.

Nous utilisons Background Sync pour toutes les requêtes `PATCH`, `POST` et `PUT` faites à l'API.

```
01 | /*
02 | * Background Sync
03 | * If the user has no network or the server is not working, the API request (
       PATCH, POST, PUT) is saved until it works again
04 | */
05 | const bgSyncPlugin = new BackgroundSyncPlugin("apiQueue", {
06 |   maxRetentionTime: 24 * 60, // retry for max 24 hours
07 | });
08 | const matchRouteBackgroundSync = ({ url }) => url.pathname.startsWith("/api");
09 | registerRoute(matchRouteBackgroundSync, new NetworkOnly({ plugins: [
       bgSyncPlugin] }), "PATCH");
10 | registerRoute(matchRouteBackgroundSync, new NetworkOnly({ plugins: [
       bgSyncPlugin] }), "POST");
11 | registerRoute(matchRouteBackgroundSync, new NetworkOnly({ plugins: [
       bgSyncPlugin] }), "PUT");
```

FIGURE 4.14 – Background Sync pour les requêtes PATCH, POST et PUT

5. Présentation du résultat

À ce jour, Guacamole est une PWA fonctionnelle, accessible via Internet depuis un ordinateur ou un téléphone portable. Les fonctionnalités demandées ont été implémentées, et sont visibles sur les figures 5.2a :

— Une interface conviviale et ludique, comportant des graphismes et animations réalisés par Roxane Herbstmeyer
— Un mode d'entraînement libre (fig. 5.2f)
— Un mode de duel, avec un temps de 24 h pour jouer chaque manche (fig. 5.2d et 5.2e)
— Une vue administrateur sommaire, permettant de fournir les données pour l'application (fig. 5.2h)

Nous avons aussi ajouté des fonctionnalités qui nous paraissaient utiles, en concertation avec nos clients :

— Un nettoyage automatique de la base de données pour supprimer les anciens duels, configurable depuis la vue administrateur
— Un système d'avatar personnalisable à partir d'une liste d'objets (yeux, chapeaux, couleurs...) (fig. 5.2g)
— Un thème sombre pour une meilleure intégration sur les téléphones utilisant ce thème (fig. 5.2f)
— Un mode hors-ligne basique, permettant de pallier les coupures courtes de réseau

Le code source de notre projet est disponible sur GitHub, sous licence MIT, à l'adresse https://github.com/ValFraNath/guacamole. La documentation de notre projet (en anglais) est accessible sur https://github.com/ValFraNath/guacamole/wiki pour le code, et https://valfranath.github.io/guacamole-api-docs/ pour l'API.

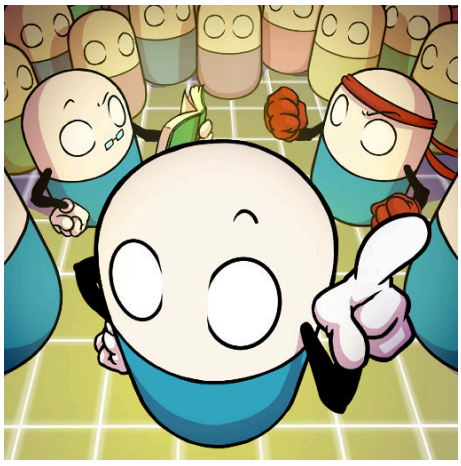
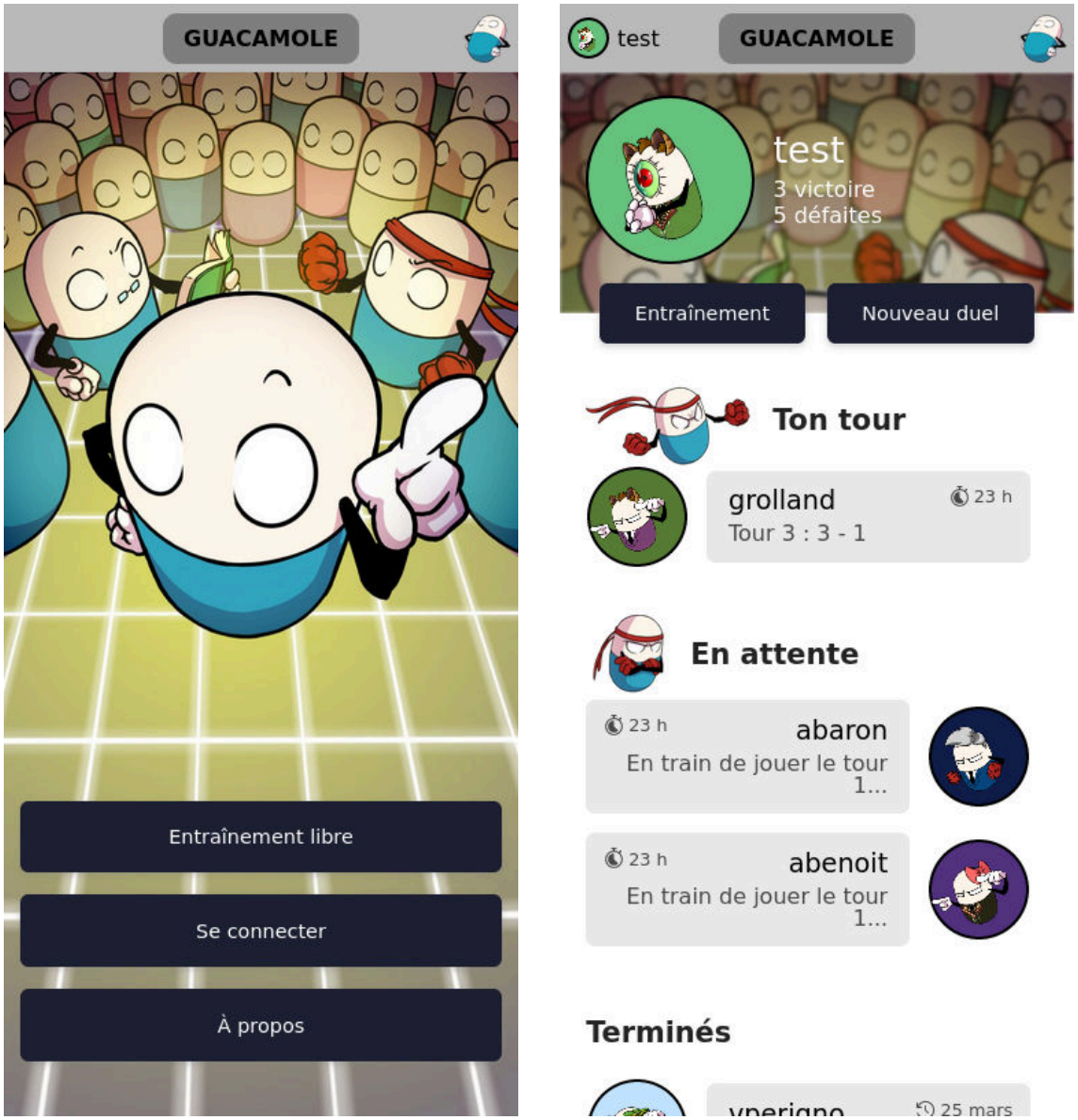


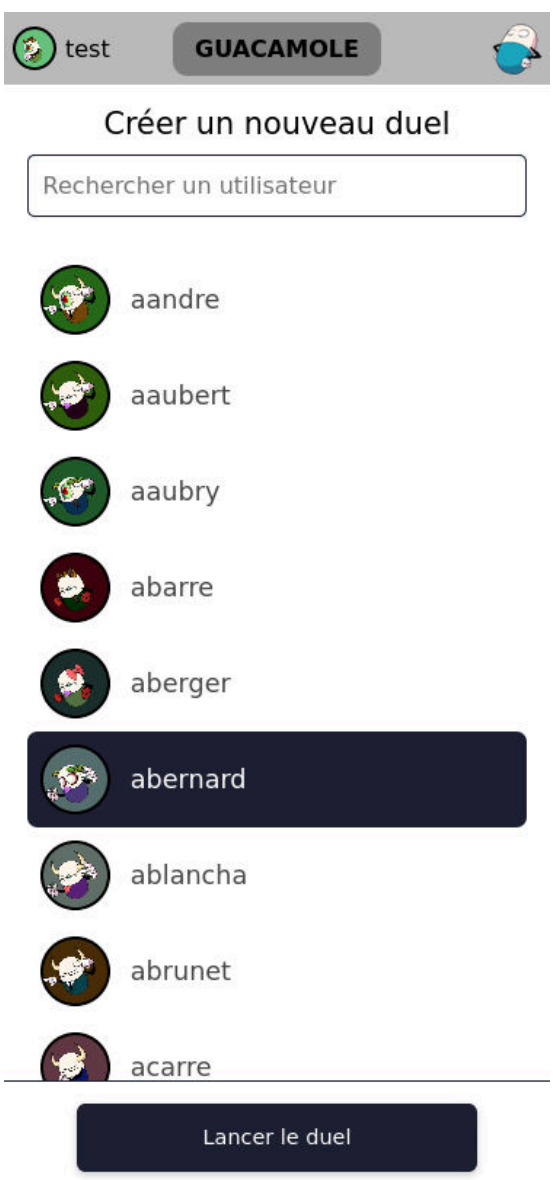
Figure 5.1 – Logo de l'application Guacamole



(a) Écran d'accueil d'un visiteur

(b) Écran d'accueil d'un utilisateur connecté

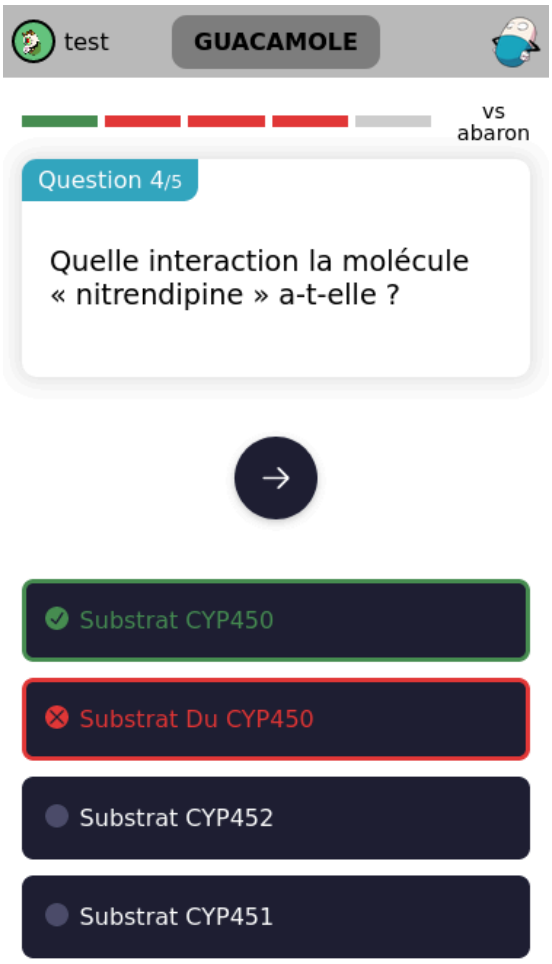
FIGURE 5.2 – Captures d'écran de l'application (1/4)



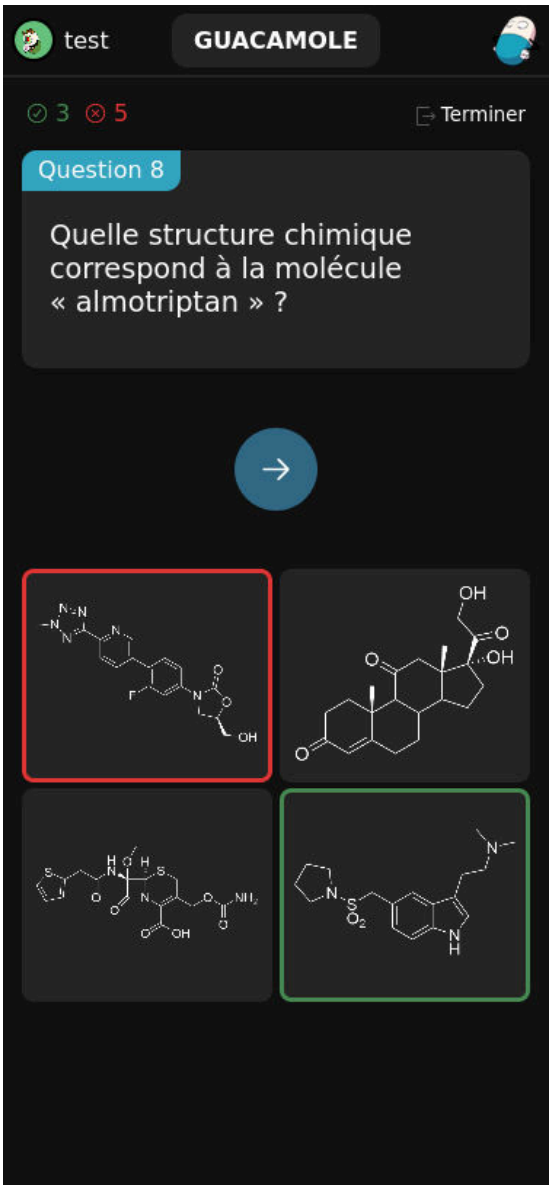
(c) Choix d'un adversaire

(d) Vue d'ensemble d'un duel

FIGURE 5.2 – Captures d'écran de l'application (2/4)



(e) Jeu d'un duel en thème clair



(f) Entraînement libre en thème sombre

FIGURE 5.2 – Captures d'écran de l'application (3/4)

(g) Page de profil et choix de l'avatar

(h) Espace administrateur (vue ordinateur)

FIGURE 5.2 – Captures d'écran de l'application (4/4)

6. Bilan

Ce projet est le plus important que nous ayons réalisé jusqu'ici, et nous aura demandé beaucoup d'implication. Voici quelques statistiques :

— 1354 commits individuels réalisés sur 6 mois
— 178 *Pull Requests*[1] (fig. 6.1)
— 7200 lignes de JavaScript pur
— 2400 lignes de JSX (pour les composants React)
— 700 lignes de scripts SQL
— 1500 lignes de SCSS
— 100 lignes de script Bash
— 20 points d'entrée API pour le serveur
— Environ 200 $\sim$ 250 h de travail chacun

Une fois le projet lancé, les plus longues tâches ont été le système d'importation des molécules (voir section 4.2.1), l'intégration de `React useQuery` dans le projet (voir section 4.3.3) et les essais pour l'implémentation des notifications.

Au cours de ces mois de travail, nous avons rencontré quelques difficultés. Le meilleur exemple de cette catégorie concerne l'implémentation des notifications. Nous souhaitions envoyer des notifications push à l'utilisateur chaque fois qu'un nouveau défi était disponible. La première étape consistait à lire la documentation sur l'*API Notification* permettant d'afficher des notifications avec l'interface du système d'exploitation de l'utilisateur, et de l'*API Push* qui permet à une application web de recevoir des messages depuis un serveur, qu'elle soit ou non active au premier plan. Nous nous sommes ensuite renseigné sur Firebase qui est un ensemble de services propulsé par Google. Il propose par exemple des bases de données NoSQL, un système d'authentification sociale ou, plus important, un service pour notifications push. Firebase permet d'envoyer des notifications push depuis notre serveur à n'importe quel appareil supportant les APIs citées précédemment. Par la suite, lors de l'intégration de Firebase à Guacamole, nous avons rencontré de nombreux problèmes. Le temps écoulé depuis le début de cette tâche et le fait que cette fonctionnalité

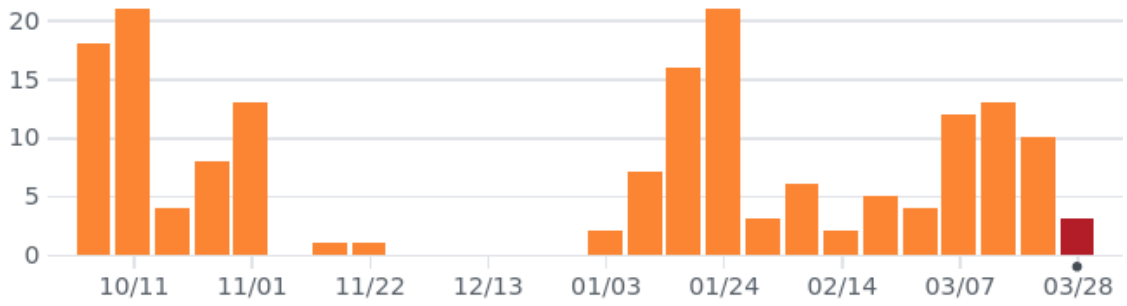


FIGURE 6.1 – Nombre de *Pull Requests* sur la durée du projet.
(la fin du semestre 5 nous a forcé à faire une pause)

1. Nous expliquons notre rythme de travail avec les *Pull Requests* dans la partie 4.1.2

ne sera pas disponible sur les téléphones iOS (à cause d'une limitation d'Apple) nous ont incités à arrêter son développement. Cependant, la *Pull Request* correspondante est toujours disponible sur notre dépôt GitHub et a été documentée pour que son développement puisse être repris plus tard.

Nous avions largement sous-estimé le temps nécessaire pour la lecture de la documentation. De septembre à octobre, il a d'abord fallu nous renseigner sur le bon usage de React. Mais ensuite, et pendant toute la durée du projet, nous avons continué à lire de la documentation : celle des bibliothèques tierces importées, celle des services externes utilisés (comme par exemple pour les notifications). Certaines après-midi consacrées au projet se déroulaient sans éditeur de code ouvert, uniquement à lire de la documentation et à prendre des notes. Si ces moments peuvent ressembler à des pertes de temps ou de productivité, nous avons réalisé qu'il n'en était rien : se renseigner sur le code qu'on introduit dans notre projet est essentiel, en premier temps pour l'utiliser correctement, et en deuxième pour connaître et prévoir les bugs potentiels. Lire la documentation est une compétence essentielle et nous sommes contents de l'avoir travaillée.

7. Conclusion & perspectives

État actuel du projet Les six mois de travaux sur ce projet nous ont permis d'aboutir à une version satisfaisante de l'application, correspondant globalement aux spécifications attendues. L'application n'exploite pas complètement les possibilités offertes par une *PWA* [1], en particulier, l'absence de cache persistant [2] empêche l'utilisation de l'application en mode hors connexion. En raison de la durée du projet, nous avons privilégié le bon fonctionnement de l'application pour une utilisation classique, tout en préparant au mieux les futures extensions. Le bilan reste tout de même très positif, l'application est fonctionnelle, testée et visuellement au goût de toute l'équipe.

Finalisation du projet La finalisation du projet fait l'objet d'un stage de 3e année de Licence Informatique, occupé par un camarade de promotion. Il portera principalement sur l'extension de l'espace administrateur, la mise en place de l'authentification via le *CAS* [3] ainsi que la mise en production de l'application. Un premier déploiement est prévu à mi-parcours du stage, afin d'exploiter les premiers retours des utilisateurs pour préparer au mieux le déploiement final.

Potentielles améliorations Comme évoqué plus haut, la gestion du mode hors connexion permettrait à l'application d'exploiter le plein potentiel d'une *PWA*, et ainsi d'en améliorer l'expérience utilisateur. D'autres améliorations sont envisageables, notamment la possibilité de configurer un duel, en choisissant le niveau de difficulté, ou de choisir un thème spécifique pour les questions. Certaines optimisations seront peut-être aussi à prévoir, en fonction des performances de l'application lors d'une utilisation réelle.

Conclusions personnelles Ce projet était pour nous l'occasion de mettre en oeuvre nos compétences, dans le domaine que nous recherchions. C'était la première fois que nous collaborions sur un projet de cette envergure, dans l'objectif d'une réelle utilisation. Cette expérience fut très riche dans de nombreux domaines, nous avons appris à utiliser de nouvelles technologies, découvert les bienfaits et limites d'un *framework*, amélioré nos méthodes de travail et collaboré avec des professionnels d'autres domaines. Nous mesurons la chance que nous avons eue de pouvoir réaliser ce projet, avec une équipe aussi agréable qu'efficace. Nous prendrons plaisir à suivre l'évolution et l'envol de Guacamole, qui restera pour nous un souvenir riche et précieux de notre troisième année de Licence.

1. *Progressive Web Application*
2. Cache stockant des données au niveau du client, même lorsque l'application est fermée.
3. *Central Authentication Service* : Service d'authentification utilisé par l'université.

Annexes

```
{
  "id": 42,
  "currentRound": 2,
  "rounds": [
    [ // Tableau de questions de la manche 1
      {
        "type": 1,
        "subject": "anticonvulsivants",
        "answers": ["aciclovir", "oseltamivir", "camion", "efavirenz"],
        "goodAnswer": 2,
        "myAnswer": 3,
        "opponentAnswer": 2
      },
      {
        "type": "1",
        "subject": "anticonvulsivants",
        "answers": ["aciclovir", "oseltamivir", "camion", "efavirenz"],
        "goodAnswer": 3,
        "myAnswer": 3,
        "opponentAnswer": 0
      }
    ],
    [ // Tableau de questions de la manche 2
      {
        "type": 2,
        "subject": "anticonvulsivants",
        "answers": ["aciclovir", "oseltamivir", "camion", "efavirenz"],
        "goodAnswer": 3,
      },
      {
        "type": 2,
        "subject": "anticonvulsivants",
        "answers": ["aciclovir", "oseltamivir", "camion", "efavirenz"],
        "goodAnswer": 3,
      }
    ],
    [ // Tableau de question de la manche 3
      {
        "type": 6
      },
      {
        "type": 6
      }
    ]
    // ...
  ]
}
```

FIGURE 7.1 – Représentation JSON d'un duel en cours

```
axios.interceptors.response.use(
  (response) => response, // Laisse passer les réponses sans erreur
  async (error) => {
    // Garde la configuration de la requête originale
    const originalRequest = error.config;
    // Récupère le jeton de rafraîchissement de l'utilisateur
    let { refreshToken } = AuthService.getCurrentUser();

    // Si l'erreur est due à une mauvaise authentification
    if (  refreshToken &&
          error.response.status === 401 &&
          !originalRequest._retry ) {

      originalRequest._retry = true;
      // Demande la génération d'un nouveau jeton d'accès
      const res = await axios.post("/api/v1/users/token", { refreshToken });

      if (res.status === 200) {
        // Met à jour le jeton d'accès
        AuthService.updateAccesToken(res.data.accessToken);
        console.info("Access token refreshed!");
        // Execute la requête originale nouveau
        return axios(originalRequest);
      }
    }
    return Promise.reject(error);
  }
);
```

FIGURE 7.3 – Automatisation du rafraîchissement des jetons

```
class DuelCreate extends Component {
  constructor(props) {
    super(props);
    this.state = {
      listOfUsers: {},
      opponent: null, // Utilisateur sélectionné
      search: "",     // Recherche actuelle
    };
  }

  componentDidMount() {
    axios
      .get("/api/v1/users")
      .then((resUsers) => {
        axios
          .get("/api/v1/duels")
          .then((resDuels) => {
            // [...] Traitement de la requête
            this.setState({
              listOfUsers: usersToDisplay,
            });
          })
          .catch((err) => console.error(err));
      })
    .catch((err) => console.error(err));
  }

  // [...] Gestion du filtre de recherche et de la requête de création

  render() {
    // [...] Affichage en JSX
  }
}
```

(a) Implémentation de départ, sous forme de classe

```
const DuelCreate = () => {
  const [searchRegex, setSearchRegex] = useState(null);
  const [selected, setSelected] = useState(null);
  const [listOfUsers, setListOfUsers] = useState({});

  useEffect(() => {
    // [...] Même requête axios
  }, []);

  // [...] Gestion du filtre de recherche et de la requête de création

  return (
    // [...] Affichage en JSX
  )
}
```

(b) Composant `DuelCreate` sous forme fonctionnelle

FIGURE 7.2 – Réécriture de `DuelCreate` en 4 étapes (partie 1)

```jsx
const DuelCreate = () => {
  const [searchRegex, setSearchRegex] = useState(null);
  const [selected, setSelected] = useState(null);

  const { data: listOfUsers, isSuccess } = useQuery(
    ["users", "challengeable"],
    {
      queryFn: getChallengeableUsers,
      staleTime: 60 * 60 * 1000,
    }
  );

  if (!isSuccess) {
    return <Loading />;
  }

  // [...] Gestion du filtre de recherche et de la requête de création

  return (
    // [...] Affichage en JSX
  )
}
```

(c) Composant `DuelCreate` avec `useQuery`

```jsx
// Fonction de récupération des données
export async function getChallengeableUsers() {
  const res = await axios.get("/api/v1/users?challengeable=true");
  return res.data;
}

// La requête ["users", "challengeable"] utilise toujours ces paramètres
queryClient.setQueryDefaults(["users", "challengeable"], {
  queryFn: getChallengeableUsers,
  staleTime: 60 * 60 * 1000,
});

// Utilisation finale dans les composants
const queryRes = useQuery(["users", "challengeable"]);
```

(d) Configuration finale de `useQuery`

FIGURE 7.2 – Réécriture de `DuelCreate` en 4 étapes (partie 2)

Résumé

Guacamole est une application d'apprentissage réflexe, destinée aux étudiants en faculté de pharmacie. Ce rapport présente son développement dans le cadre de notre projet tutoré, réalisé en troisième année de Licence Informatique. Utilisable sur ordinateur, tablette et smartphone sous la forme d'une *Progressive Web Application*, elle permet aux utilisateurs de s'entrainer tout seuls ou à plusieurs au travers de duels. L'application présente également un espace administrateur, permettant notamment l'importation de nouvelles données, utilisées pour la génération aléatoire de questions. Une graphiste a participé à la conception et la réalisation des différents visuels de l'application, afin de la rendre agréable et ludique.

Mots-clés : ReactJS, Node.js, MariaDB, CSS, Jeu sérieux

Abstract

Guacamole is a reflex learning application for pharmacy students. This report presents its development as part of our tutored project, carried out during our third year of computer sciences bachelor's degree. Available on computer, tablet and smartphone as a *Progressive Web Application*, it allows users to train alone, or against others through duels. The application also provides an administrator area, allowing to import new data, used in the random generation of questions. A graphic designer participated in the design and production of many visuals for the application, to make it pleasant and fun.

Keywords : ReactJS, Node.js, MariaDB, CSS, serious game