

Di Vrusa Florian
Ducobu Alexandre
Quittet Guillaume

Année 2017–2018

Développement web avancé

Rapport de projet

Table des matières

1	Introduction	4
	Avant-propos	4
2	Technologies utilisées	5
2.1	Angular	5
2.2	Docker	5
2.3	Git	5
2.4	GitHub	5
2.5	Heroku	5
2.6	OAuth	6
2.7	Slack	6
2.8	Spring	6
2.9	Travis	7
2.10	TypeScript	7
3	Nos outils	8
4	Déroulement	9
4.1	Initialisation	9
4.1.1	Configuration	9
4.2	Répartition des tâches	10
4.3	Organisation du code	10
4.4	Compilation et déploiement	10
4.4.1	Travis	11
4.5	Base de données	13
4.5.1	Outil de travail	13
4.6	Back-end	14
4.6.1	Outil de travail	14
4.6.2	Organisation du code	14
4.6.3	Liaison avec la base de données	14
4.7	Front-end	16
4.7.1	Installation	16
4.7.2	Commandes	16
4.8	Intégration avec Spring	16

5	Problèmes rencontrés	18
5.1	Mise en place du projet	18
5.2	Connexion à la base de données	18
5.3	Communication avec la base de données	18
5.4	Aide limitée sur Angular	19
5.5	Problème lié au token	19
5.6	Départ d'un membre de l'équipe	19
5.7	Compilation du projet sur Windows	19
6	Conclusion	20
	Références	21

1 Introduction

Avant-propos

Dans le cadre du cours de *Développement web avancé*, nous avons dû réaliser un projet qui consiste à mettre en place une plateforme web servant à la prise de rendez-vous pour un cabinet vétérinaire.

Le but est d'apprendre à mettre ensemble plusieurs technologies utilisées dans le monde de l'entreprise comme Angular, Docker, Git, Heroku, Spring et Travis.

Le projet ne doit pas forcément être terminé, car l'important c'est de voir comment nous avons travaillé afin de créer un environnement de développement productif et de pouvoir réutiliser nos connaissances dans le monde de l'entreprise à notre sortie dans le monde du travail.

2 Technologies utilisées

2.1 Angular

Angular est un framework, écrit en TypeScript, succédant à AngularJS, son grand frère, qui lui a été écrit en JavaScript[8]. Il est développé par Google qui a décidé de le rendre libre et open source.

Son but est de simplifier le développement d'applications et de pages web[2].

Nous l'avons utiliser afin de faire nos vues. Il est communément appelé « Angular 2+ » ou « Angular 2 ».

2.2 Docker

Docker est une société qui gère le mouvement des conteneurs et est aussi un logiciel libre qui automatise le déploiement d'applications dans des conteneurs logiciels.

Il permet une véritable indépendance entre les applications, l'infrastructure, les développeurs et les opérations informatiques.

Il a été développé en Go.[1][5]

2.3 Git

Git est un logiciel libre de gestion de versions décentralisé.

Il a été conçu et développé par Linus Torvalds en 2005.[6]

2.4 GitHub

GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant Git.

Nous l'avons utilisé afin de synchroniser notre travail, de l'organiser selon la méthode Agile ainsi que d'utiliser Travis CI.

2.5 Heroku

Heroku est un service de « cloud computing » de type plate-forme en tant que service (PaaS). Celui-ci est utilisé comme modèle de déploiement d'applications Web. Heroku Postgres est le service de base de données Cloud (DBaaS) de Heroku basé sur PostgreSQL. Heroku Postgres offre des fonctionnalités telles que la protection continue, le rollback et la haute disponibilité.

2.6 OAuth

OAuth est un protocole libre qui permet d'accéder à des données tout en protégeant le pseudonyme et le mot de passe des utilisateurs.

Nous avons utilisé la version 2 qui est la dernière à ce jour[4][7].

Nous avons choisi cette technologie pour gérer le token, car elle est fortement utilisée en entreprise et qu'elle est parfaitement intégrée avec le framework Spring.

2.7 Slack

Slack est une plateforme de collaboration et un logiciel de gestion de projets.

Celle-ci sera utilisée dans le but de communiquer entre développeurs et avec le professeur.

Elle nous permettra aussi de suivre les différents commits sur le dépôt GitHub grâce à son intégration native.

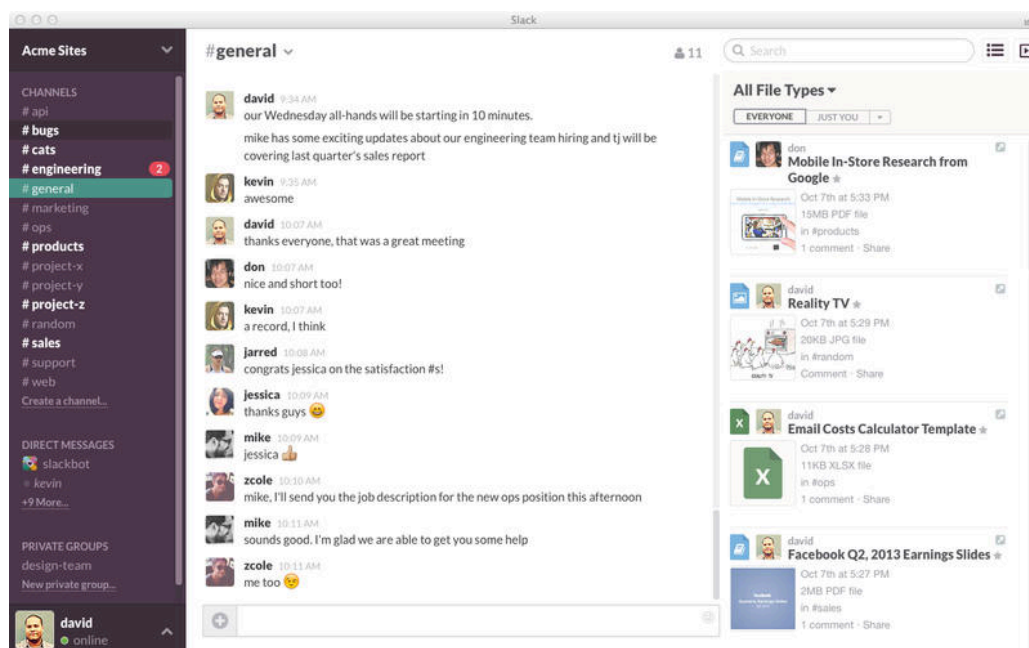


FIGURE 1 – Vue de la plateforme Slack

2.8 Spring

Le framework libre Spring permet de construire et de définir l'infrastructure d'une application Java (back-end), dont il facilite le développement et les tests.

Il s'agit d'une solution modulaire, ce qui en fait sa force et explique, avec sa richesse et son efficacité, l'engouement des développeurs pour ce framework.

Le Java permet d'utiliser la programmation orientée objet. Du côté de la gestion des dépendances et de la compilation du site, nous avons utilisé le moteur de production Gradle.

2.9 Travis

Travis CI fournit un service en ligne utilisé pour compiler, tester et déployer le code source depuis, par exemple, GitHub.

Dès qu'un commit est détecté sur une branche surveillée par Travis, celui-ci s'exécutera d'après la volonté du développeur.

Le projet sera compilé, les tests seront lancés et le projet sera déployé sur Heroku si celui-ci lui est lié.

L'état du projet dans Travis CI est toujours disponible et peut même être envoyé par mail.

2.10 TypeScript

TypeScript est un langage transcompilé libre et open-source développé par Microsoft.

Nous n'avons pas eu le choix d'utiliser ce langage, car les versions d'Angular supérieure à la 2 ne fonctionnent qu'avec lui.

3 Nos outils

Étant sur trois systèmes d'exploitations différents et ayant des affinités avec différents outils, nous avons utilisé des outils différents :

- Atom, un éditeur de texte libre pour macOS, GNU/Linux et Windows développé par GitHub. Il a la particularité de pouvoir se transformer en IDE.
- Eclipse, un IDE principalement utilisé dans la programmation Java. C'est même le plus utilisé chez les développeurs Java.
- Neovim, un éditeur de code accessible depuis le terminal. C'est une version améliorée de Vim qui était, lui, une amélioration de Vi (Vi IMproved).
- Visual Studio Code, un éditeur de code extensible développé par Microsoft pour macOS, GNU/Linux et Windows, et basé sur Atom. Il a la particularité d'être compatible, de base, avec TypeScript.

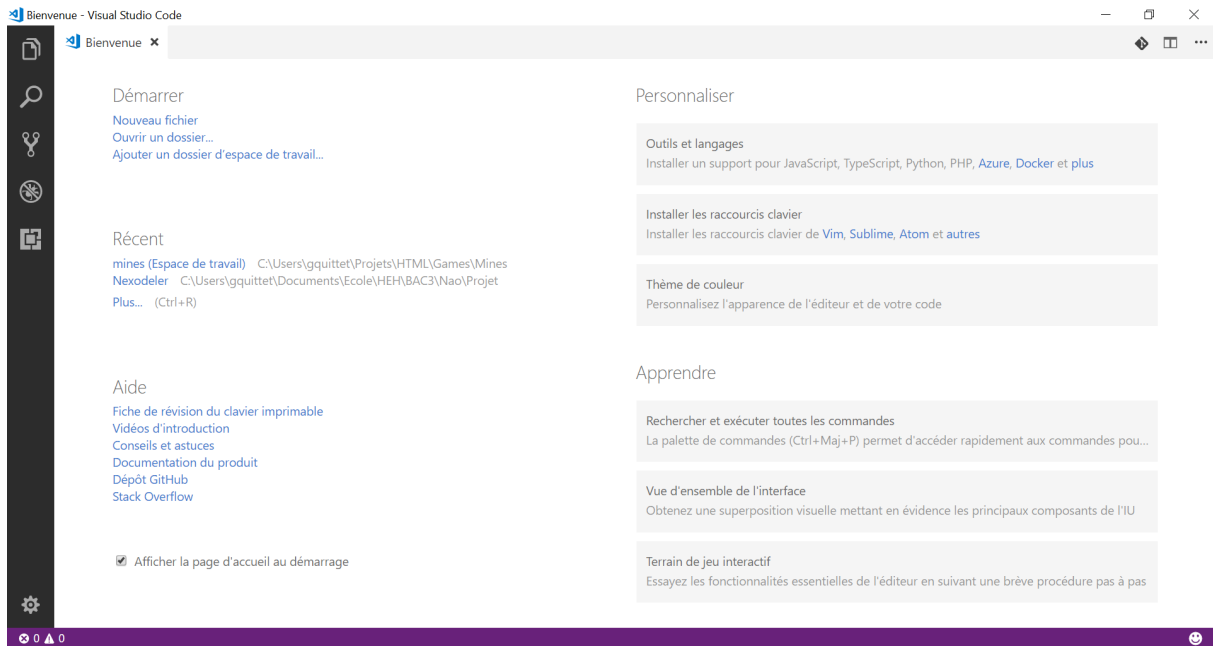


FIGURE 2 – Interface de Visual Studio Code

4 D roulement

4.1 Initialisation

Afin de commencer notre projet, nous avons      sur Spring Initializer. Ce site est la version Web de l'outil int  gr   aux IDE tel que IntelliJ, Eclipse et NetBeans.

Il nous a permis de facilement choisir les d  pendances du projet. Il faut savoir que Spring est un framework qui a pour but d'  viter de se prendre la t  te avec la gestion des d  pendances. Spring choisit les bonnes versions en veillant    la compatibilit   entre elles.

4.1.1 Configuration

Generate

Changeons l'option de base de « Generate », qui est par d  faut sur « Maven Project », par « Gradle Project ». C'est juste le moteur de production qui va servir    t  l  charger et installer les d  pendances, compiler et lancer notre programme. Nous pr  f  rons utiliser Gradle    la place de Maven car il est beaucoup plus simple    comprendre et est plus puissant.

Project Metadata

Dans la section « Project Metadata, nous allons modifier le paquet de notre application ou « Group ». On peut par exemple mettre « be.heh ». Il s'agit g  n  ralement d'un nom de domaine   crit    l'envers. L'« Artifact » correspond au nom de l'application. Il ne contient pas d'espace et pas de caract  res sp  ciaux.

D  pendances

Afin de mener    bien ce projet, nous avons choisit les d  pendances suivantes :

Web Permet d'utiliser notre application Spring comme une application Web.

JPA Sert    faire la liaison entre nos objets Java et les la base de donn  es tout en utilisant Hibernate.

JDBC Est le pilote qui permet de faire la liaison entre notre projet Spring et la base de donn  es.

PostgreSQL Permet de se connecter    la base de donn  es de type PostgreSQL.

OAuth2 Permet de faire le code qui va permettre d'envoyer le token et de g  rer sa validit  .

REST Permet de faire le code de l'API Rest.

DevTools Permet de compiler le code automatiquement de façon asynchrone à chaque sauvegarde afin de voir les modifications en direct dans le navigateur.

Nous sommes passé par ce service pour la simple et bonne raison que nous utilisons tous un éditeur différent. Le fait de passer par le site nous a permis de gagner un peu de temps.

SPRING INITIALIZR bootstrap your application now

Generate a Gradle Project with Java and Spring Boot 1.5.8

Project Metadata

Artifact coordinates

Group

be.heh

Artifact

testspring

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Web JPA Thymeleaf PostgreSQL

Generate Project alt + ⌘

Don't know what to look for? Want more options? [Switch to the full version.](#)

start.spring.io is powered by [Spring Initializr](#) and [Pivotal Web Services](#)

FIGURE 3 – Interface de Spring Initializer

4.2 Répartition des tâches

Afin de travailler plus efficacement, nous nous sommes réparti les différentes tâches. La partie front-end, plus conséquente, a été attribuée à Florian et à Alexandre. Quant à la partie back-end, elle a été attribuée à Guillaume. La partie base de données a été attribuée à Florian, car Alexandre avait déjà bien avancé sur le front-end.

4.3 Organisation du code

Afin de séparer correctement nos parties, nous avons voulu utiliser une méthode qui consiste à avoir un dossier qui contient toute la partie back-end et un dossier qui contient toute la partie front-end. Pour que tout compile correctement depuis la racine, nous avons créé des scripts et des fichiers de configurations qui permettent de spécifier les différents dossiers à compiler ainsi que les sorties de compilation.

4.4 Compilation et déploiement

Afin de compiler et déployer notre application, nous avons utilisé Travis et Heroku comme demandé.

4.4.1 Travis

Travis est un outil super pratique en ligne qui permet de facilement compiler et déployer son code.

Pour le configurer, il suffit de créer le fichier « *.travis.yml* » à la racine du projet.

On a mis plusieurs options comme :

- Désactiver les notifications par email (spam).
- Mettre la clé d'API générée sur le compte qui a créé la base de données.
- Spécifier le nom de l'application.

Voici le contenu mis :

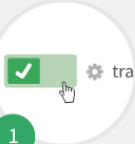
```
1 language: java
2
3 notifications:
4   email: false
5
6 branches:
7   only:
8     - develop
9     - master
10
11 deploy:
12   provider: heroku
13   api_key: "la clé API"
14   app: lenomdelapp
```

HeH-Projects

Sync account


We're only showing your public repositories. You can find your private projects on travis-ci.com.

1




Click the repository switch on

2




Add .travis.yml file to your repository


3




Trigger your first build with a git push




HeH-Projects/team-agozzino




HeH-Projects/team-cailleaux



HeH-Projects/team-diduqui



HeH-Projects/team-simond



HeH-Projects/team-urbain

Pour crypter notre clé, nous sommes allés sur le site <https://travis-encrypt.github.io/> et nous avons changé dans la configuration de Travis « `api_key :` » par « `api_key : secure :` ».

4.5 Base de données

4.5.1 Outil de travail

Afin de travailler sur la base de données, Florian a installé avec l'aide de Docker le programme PostgreSQL.

Il a récupéré une image afin de créer un container.

Il a par la suite créé des scripts qui lui ont permis de créer la base de données avec les tables.

Docker

Vu qu'Heroku utilise le PostgreSQL, nous avons dû télécharger cette image :

```
1 docker search postgresql
2 docker pull postgres
```

Installons l'image dans un container en partageant la base de données qui se trouve dans un dossier de notre ordinateur (*ex* : C:/Users/test/database).

```
1 docker run --name postgresDB -d -p 1000:5432 -v C:
  /Users/test/monapplication/database:/var/lib/postgresql
  postgres
```

On donne un nom au container afin de le manipuler plus facilement dans le futur.

Il est toujours conseillé de prendre les images officielles.

Pour lancer une invite de commande dans le container, il suffit de taper :

```
1 docker exec -it postgresDB /bin/bash
```

Une fois dans le container, on peut par exemple se connecter à la base de données d'Heroku avec la commande suivante :

```
1 psql -h hostname -U username -d database
```

On entre le mot de passe et nous sommes connectés à notre base de données.

4.6 Back-end

4.6.1 Outil de travail

Afin de faire correctement le back-end, Guillaume a voulu utiliser Eclipse, car bien qu'IntelliJ est mieux, quand nous allons sortir de l'école, nous n'aurons plus la licence afin de l'utiliser. Vu le prix de cette dernière, il faut mieux s'habituer à utiliser des outils un peu moins performants mais qui restent gratuits.

4.6.2 Organisation du code

Afin de ne pas se faire surprendre par le nombre de fichiers, Guillaume a préféré utiliser des paquets.

En voici la liste :

domain Contient la partie API REST et objet de la base de données. On y retrouve les paquets suivants :

entities Les objets de la base de données.

repositories Les fonctions qui permettent de retourner ou modifier des données au format JSON, puis Java suivant l'URL utilisé.

security La partie sécurité du site avec le token.

web Contient le contrôleur principal. Il sert juste à afficher l'application.

4.6.3 Liaison avec la base de données

Comme dit précédemment, nous avons utilisé JPA afin de se connecter à la base de données. Pour que celui-ci fonctionne correctement, nous avons dû créer une URL correcte.

En effet, l'URL données sur le site d'Heroku n'est pas valide quand on travaille avec le pilote JDBC. Il a donc fallu l'adapter.

Voici la configuration utilisée, à la page suivante :

```

1  spring.datasource.url = jdbc:postgresql://Host:Port/Database?
    sslmode=require&user=User&password=Password
2  spring.datasource.username = User
3  spring.datasource.password = Password
4  spring.datasource.maxActive = 1
5  spring.datasource.maxIdle = 1
6  spring.datasource.minIdle = 0
7  spring.datasource.initialSize = 1
8  spring.datasource.removeAbandoned = true
9  spring.datasource.driver-class-name = org.postgresql.Driver
10 spring.jpa.database-platform =
    org.hibernate.dialect.PostgreSQLDialect
11 spring.jpa.show-sql = false
12 spring.jpa.hibernate.ddl-auto = none
13 spring.output.ansi.enabled = ALWAYS
14 server.port = 8080

```

La variable de configuration « `spring.jpa.hibernate.ddl-auto` » est obligatoire, car on ne veut pas que Spring génère automatiquement les tables.

Pour les variables utilisées, nous avons été sur le site d’Heroku qui nous les fournis :

ADMINISTRATION

Database Credentials
Get credentials for manual connections to this database. Cancel

Please note that **these credentials are not permanent**.
Heroku rotates credentials periodically and updates applications where this database is attached.

Host	ec2-46-137-97-169.eu-west-1.compute.amazonaws.com
Database	d9lbr7bgkufe46
User	qnhiwzkwdmslx
Port	5432
Password	d95b6478ecfe4faa0c601f747e52d818fedc4fc6d4b8eb5418f106174502344a
URI	postgres://qnhiwzkwdmslx:d95b6478ecfe4faa0c601f747e52d818fedc4fc6d4b8eb5418f106174502344a@ec2-46-137-97-169.eu-west-1.compute.amazonaws.com:5432/d9lbr7bgkufe46
Heroku CLI	heroku pg:psql postgresql-rigid-81700 --app testspring-tri

FIGURE 5 – Partie base de données du site Heroku

4.7 Front-end

4.7.1 Installation

Pour la partie frontend, nous avons utilisé Angular. Pour commencer, on a installé Node.JS avec la commande suivante :

```
1 npm install -g @angular/cli
```

Si jamais vous avez comme message d'erreur que « user root does not have permission to access the dev dir », il suffit de taper la commande suivante :

```
1 npm install -g @angular/cli --unsafe-perm
```

4.7.2 Commandes

Voici la liste des commandes Angular utilisées :

ng new nomduprojet Créer un projet Angular

ng generate component nomducomposant Créer un composant

ng serve Démarrer le serveur

Si jamais la commande « ng serve » demande d'installer les node_modules, il suffit d'effectuer la commande suivante dans le dossier de l'application Angular : npm install

4.8 Intégration avec Spring

Nous avons créé un fichier à la racine de l'application Angular et nous l'avons nommé « proxy.conf.json ». Voici son contenu :

```
1 {
2   "/api": {
3     "target": "http://localhost:8080",
4     "secure": false
5   }
6 }
```

target, nous permet de spécifier le lien de notre application Spring avec le bon port.

Ensuite, nous avons édité le fichier « package.json ». Nous avons modifié la partie « scripts » comme ceci :

```
1 "scripts": {  
2   "ng": "ng",  
3   "start": "ng serve --proxy-config proxy.conf.json",  
4   "build": "ng build -prod",  
5   "postbuild": "npm run deploy",  
6   "predeploy": "rimraf cheminVersLeDossierStatic && mkdirp  
   cheminVersLeDossierStatic",  
7   "deploy": "cp -rf dist/** cheminVersLeDossierStatic",  
8   "test": "ng test",  
9   "lint": "ng lint",  
10  "e2e": "ng e2e"  
11 },
```

Pour nous, le chemin vers le dossier static est « ../resources/static/ ».

En modifiant la partie « start », on spécifie à Angular de passer par le projet Spring pour utiliser la bonne API.

Ensuite, on spécifie à la partie « build » que l'on veut une compilation de production.

On ajoute 3 options en dessous de cette ligne :

postbuild Lance le déploiement juste après la compilation.

predeploy Supprime le dossier « static » se trouvant dans « resources » et le récrée. Cela permet de nettoyer tous les fichiers qui étaient dedans.

deploy Permet de copier tous les fichiers empaquetés par Angular dans le dossier « resources/static ». C'est là où Spring va rechercher les documents HTML.

Pour compiler l'application Angular afin de l'intégrer à Spring, il ne reste plus qu'à taper la commande suivante : « ng run build ». Ainsi, on compile et empaquète notre projet Angular.

Dans Spring, il ne reste plus qu'à créer un « controller » qui va permettre de lancer la page web générée précédemment en fonction de l'URL.

5 Problèmes rencontrés

5.1 Mise en place du projet

La mise en place du projet ne fut pas une chose facile. Au départ, nous utilisions IntelliJ afin de créer celui-ci. Cet éditeur devait nous générer une base complète pour réaliser notre projet dans les meilleures conditions, or nous avons eu pas mal de problèmes. Des erreurs diverses apparaissaient dans le code généré, certaines dépendances n'étaient pas correctement ajoutées au projet et l'arborescence n'était pas très optimisée.

Afin de pallier ce problème, nous avons donc utilisé un outil en ligne permettant de générer la base de notre projet (Spring Initializr). Celui-ci nous permet de sélectionner plusieurs options comme le moteur de production (Gradle ou Maven), le langage, la version du framework Spring ainsi que les dépendances. Une fois les options sélectionnées, le site génère un projet à télécharger sous forme d'archive.

5.2 Connexion à la base de données

Pour stocker les données, nous utilisons une base de données PostgreSQL hébergée sur Heroku. Pour effectuer certains tests (création de tables, ajouts de colonnes, de données), nous avons utilisé l'outil de gestion intégré à IntelliJ. Cependant, l'URL de connexion fournie par Heroku était incorrecte. Il nous était alors impossible de se connecter à la base de données aussi bien avec l'outil qu'avec notre application.

Cependant, Heroku fournit également les informations de connexion nécessaires (hôte, nom d'utilisateur, mot de passe, numéro de port, ...). Grâce à ces informations et après une lecture plus approfondie de la documentation d'Heroku, nous sommes parvenus à écrire une URL correcte pour la connexion à la base de données.

5.3 Communication avec la base de données

Alors que nous avançons dans le projet, nous nous sommes rendu compte que le nombre de connexions était limité à 20 sur Heroku. Cela n'aurait pas dû poser de problèmes car nous n'étions que 3 à nous connecter en simultané sur la base de données. Or, Heroku comptabilisait, la plupart du temps, 10 ou 11 connexions. De temps en temps, il nous était même devenu impossible de nous connecter à la base de données car le nombre de connexions limite était atteint. Heureusement, il suffisait de tuer manuellement les connexions à la base de données pour pouvoir se reconnecter à nouveau.

5.4 Aide limitée sur Angular

Les tutos et forums traitants d'Angular sont nombreux, cependant, ceux-ci traitent la plupart du temps AngularJS. Étant donné que nous travaillions avec TypeScript, nous avons eu plus de mal à avancer rapidement dans notre projet, la documentation étant moins fournie et moins précise.

5.5 Problème lié au token

Il a fallu passer plusieurs nuits blanches pour réussir à mettre en place le système de token. On ne savait pas par où commencer et la plupart des cours sur ce sujet étaient incomplets. C'est finalement en regardant une vidéo d'un des développeurs de Spring que notre groupe a réussi à faire fonctionner le oauth2 du côté du back-end.

Par la suite, nous avons eu énormément de mal pour récupérer le token du côté du front-end à cause, une nouvelle fois, du manque de documentation.

Le dernier bug que nous avons c'est que quand on rafraichissait la page, notre utilisateur perdait son token. Nous avons résolu ce problème en stockant ce dernier dans la mémoire du navigateur.

5.6 Départ d'un membre de l'équipe

Après avoir commencé le projet, nous nous sommes retrouvés confronté à une scission de l'équipe en raison de différends sur la masse de travail à accomplir pour ce projet. La scission du groupe ne pose pas de problème en soi, le problème vient du fait qu'il a fallu réorganiser tous nos outils (Github, ZenHub, Travis, Heroku, Slack, ...) et cela nous a fait perdre pas mal de temps pour avancer dans notre projet.

5.7 Compilation du projet sur Windows

Afin de pouvoir compiler le projet proprement à la main et sans problèmes, un script a été écrit en bash. Cependant, celui-ci refusait de s'exécuter sur la machine Windows en raison d'une commande non reconnue. Et cela peu importe le terminal utilisé (GitBash, PowerShell, cmd, ...). Nous avons ensuite eu l'idée d'installer le bash Ubuntu disponible dans les programmes et fonctionnalités de Windows. Celui-ci s'installe directement sur la machine (pas de virtualisation ou de conteneur) et permet d'avoir accès à tous les fichiers de la machine Windows. De cette manière, il est possible de compiler le projet également sous Windows.

6 Conclusion

Pour conclure, ce projet nous a permis de découvrir le monde du développement en entreprise tout en restant à l'école. Le fait de travailler en équipe tout en utilisant un gestionnaire de versions et des technologies utilisées nous prépare efficacement à travailler dès la fin de cette année.

Toucher à toutes les étapes de ce projet (front-end, back-end, organisation, ...), nous montre ce qui nous attend afin de mieux nous former.

La plateforme web de prise de rendez-vous est accessible depuis une page de connexion des secrétaires. Une fois la connexion effectuée, l'accès au site est sécurisé par un token.

Les secrétaires peuvent naviguer entre différents onglets afin d'afficher les données provenant de la base de données : les docteurs, les clients, les salles, ...

De plus, il a été construit sur un design personnalisé, adaptatif et soigné.

Bien qu'ayant rencontré différents problèmes et imprévus, nous nous sommes dépassés afin de les régler. Ainsi, nous avons pu continuer à avancer dans le projet.

En entreprise, ces problèmes sont tout autant susceptibles de faire leur apparition.

En effet, tout le monde ne travaille pas toujours avec le même environnement, il peut arriver qu'un collègue soit absent plusieurs mois, l'entreprise peut utiliser une technologie peut documentée, ...

Grâce à ce projet de groupe d'un genre inédit sur nos trois années, nous avons appris à travailler en équipe sur un travail tel qu'il pourrait nous être demandé lors de notre stage en entreprise.

Cela en fait une corde de plus à ajouter à notre arc.

Références

- [1] Docker. What is docker ? [Consulté le 18 janvier 2018 à 15h43].
- [2] Google. About angular. [Consulté le 18 janvier 2018 à 13h10].
- [3] HEH. Logo du campus technique. [Consulté le 9 janvier 2018 à 23h48].
- [4] OAuth. Oauth 2.0. [Consulté le 19 janvier 2018 à 13h13].
- [5] Wikipedia. Docker (software), 12 janvier 2018 (17h32). [Consulté le 18 janvier 2018 à 15h46].
- [6] Wikipedia. Git, 17 janvier 2018 (10h17). [Consulté le 18 janvier 2018 à 19h05].
- [7] Wikipedia. Oauth, 17 janvier 2018 (21h29). [Consulté le 19 janvier 2018 à 13h13].
- [8] Wikipedia. Angularjs, 30 septembre 2017 (16h39). [Consulté le 18 janvier 2018 à 13h00].