ensiie
école nationale supérieure d'informatique
pour l'industrie et l'entreprise

Rapport de projet

TROUVE MON BAR

Equipe

Swann CASTEL
Jérôme O'KEEFFE
Samir LAZZALI
Emilio DE SOUSA

Encadrant

Thomas COMES

Annee 2018

Remerciements

Remerciement à Léa Bouzaid pour sa participation à l’élaboration du design.
Remerciement à M. Comes pour son encadrement et la compréhension dont il a fait preuve.

Glossaire

- SPA: Single Page Application est une page qui ne contient qu'un seul fichier HTML et qui ne communique que par des requêtes asynchrones.
- JSON: JavaScript Notation Object, littéralement la syntaxe d'un objet JavaScript.
- XML: Extensible Markup Language est un langage basé sur des balises.
- JWT: JSON Web Token est un jeton d'authentification basé du du JSON et standardisé par la RFC 7519.
- SOAP: Simple Object Access Protocol est un protocole pour des Web Services bâti sur du XML.
- REST: REpresentational State Transfer est un style d'architecture logicielle pour créer des Web Services basés sur du JSON.
- SSO: Single Sign-On est une méthode d'authentification unique pour accéder à plusieurs applications. Cette abréviation est souvent utilisée pour décrire le serveur qui gère l'authentification unique.
- XHR: XMLHttpRequest est un object JavaScript permettant de récupérer des données via le protocole HTTP.
- DOM: Document Object Model est une interface pour interagir avec un document HTML ou XML par exemple.

Introduction

Dans le cadre de notre enseignement en informatique et plus précisément en cours de Projet Web, nous avons été amenés à réaliser un projet afin de développer notre esprit d'équipe, nos capacités organisationnelles et nos compétences en développement Web. Nous n'avions pas de sujet imposé c'est pourquoi nous avons passé une séance à réunir des idées qui pourraient apporter une valeur ajoutée à notre projet.

Finalement, nous avons opté pour un sujet qui nous correspond à tous et qui, nous pensons, répond à un besoin: trouver un bar où sortir. En effet, qui n'a pas passé des soirées à tourner en rond dans un quartier de paris à la recherche d'un endroit où aller? Ou alors retourner par dépit toujours et encore dans le même bar? C'est grâce à notre application que tous ces problèmes vont être résolus!

Ainsi "**trouve mon bar**" a pour but de proposer un bar pertinent à un utilisateur. Le principe est simple : si vous et une autres personne êtes intéressée par l'actualité et qu'elle aime un certain bar, il y a de grande chance pour que vous l'aimez aussi !
L'objectif est donc d'associer des utilisateurs à un bar grâce à leurs centres d'intérêts.
Ceci va être fait en différentes étapes, lors de l'inscription un utilisateur renseigne des mots clefs qui correspondent à ces centre d'intérêt, il va ensuite pouvoir faire une recherche sur ces mots clefs pour obtenir les bars qui y sont liées. Pour que l'application fonctionne correctement il nous faut donc lier des bars aux mots clefs. Pour cette première version c'est nous qui avons créé les associations. Une idée pour obtenir ces données pourrait être de *forcer* l'utilisateur à entrer des mots clés et des bars qu'il aime déjà dès l'inscription, avant d'avoir assez de données pour pouvoir se passer de ce système.

Approche mise en place

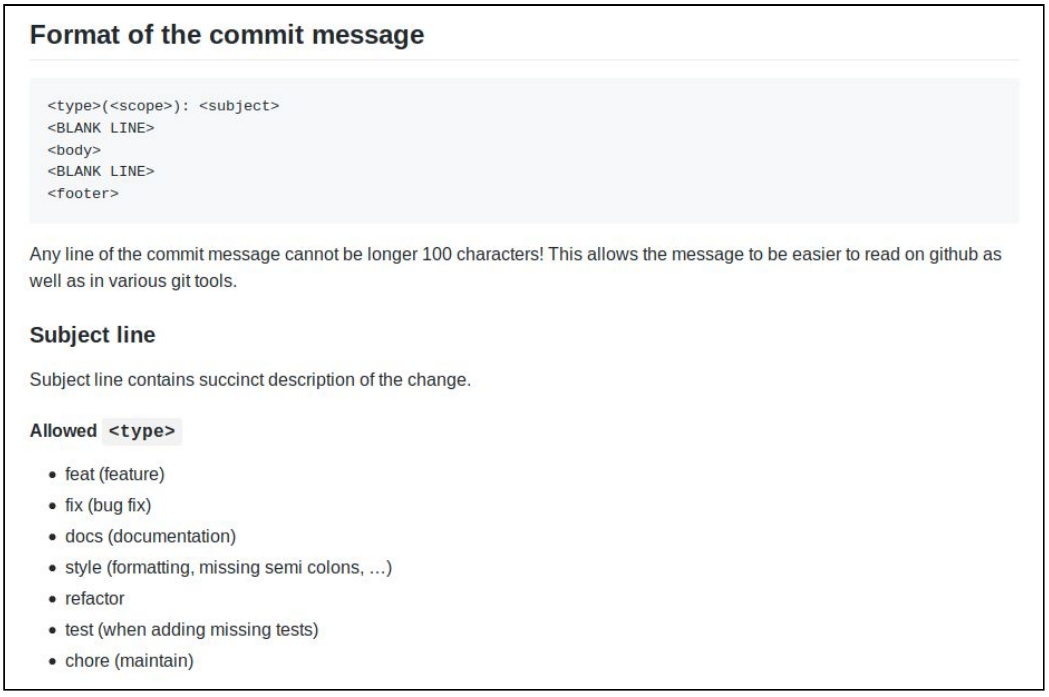
Gestion de projet

L'utilisation de Git est devenu obligatoire dans le monde du développement. Nous avons créé un fork du projet ensiie-project afin de partager notre code. Nous avons choisi Github car l'outillage est très riche. Que ce soit pour le signalement des problèmes dans le code, pour la gestion des Pull Request dans Visual Studio Code ou pour l'utilisation d'un outil d'intégration continue.

Plusieurs boards existent sur internet mais nous avons choisi Trello afin de visualiser et gérer la répartition des tâches. Nous avons connecté le board du projet à notre repository Github pour pouvoir associer des branches ou des issues à des tickets. Nous avons créé un tableau Kanban avec trois colonnes principales: **à faire**, **en cours** et **terminé**. Nous avions aussi des codes couleurs et des étiquettes pour les tickets, que ce soit pour des stories utilisateurs ou pour des bugs.

Lorsqu'un bug était repéré, nous utilisions le système d'issue de Github pour le signaler. Le développeur devait ensuite créer la branche associé à ce bug afin de le résoudre. Nous parlerons de ce type de branche dans la partie suivante.

Nous avons utilisé la convention de nommage de commit d'angular. Cela peut paraître dérisoire mais cela apport beaucoup de clarté dans l'historique des commits. C'est aussi très utilisé sur des projets open source.



Format of the commit message

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

Any line of the commit message cannot be longer 100 characters! This allows the message to be easier to read on github as well as in various git tools.

Subject line

Subject line contains succinct description of the change.

Allowed `<type>`

- feat (feature)
- fix (bug fix)
- docs (documentation)
- style (formatting, missing semi colons, …)
- refactor
- test (when adding missing tests)
- chore (maintain)

Nos méthodologies

Nous avons essayé de tester un maximum l'application que ce soit du côté backend avec Kahlan ou du côté frontend avec vue-test-utils. De plus, nous avons préconisé la méthode du Test Driven Development qui est quasiment incontournable dans le monde du développement logicielle moderne.

Le cours n'était vraiment pas optimisé pour ce genre de pratique mais dès que nous avions un problème, nous faisions du pair programming. Cela nous a permis de partager sur nos compétences en développement Web que ce soit sur Vue.js ou sur de la sécurité mais aussi sur nos différentes idées sur l'implémentation des fonctionnalités.

Pour gérer la politique d'ajout de fonctionnalités et pour avoir une baseline sur l'utilisation des branches sur Git, nous avons utilisé un Gitflow simplifié dans le sens ou nous avions trois types de branches:

- **master**, la branche courante sur laquelle on ajoute des fonctionnalités au cours des développements.
- Des branches de type **feature/<nom_de_la_fonctionnalité>**, tiré à partir de master et qui correspondent à une fonctionnalité que l'on doit implémenter.
- Et pour finir des branches de types **bugfix/<nom_du_bug>** pour corriger un éventuel bug repéré sur master.

Nous avons mis en place un système de revue de code pour ajouter des fonctionnalités sur la branche master. Grâce à Gitflow, un développeur estimait avoir implémenté une fonctionnalité sur sa branche de feature. Il faisait donc une Pull Request sur Github, pour ajouter le code sur master. Cet ajout était bloqué jusqu'à l'approbation d'au moins un autre développeur pour valider la fonctionnalité. Cela nous a permis de partager un maximum sur l'implémentation des différentes features et réduire considérablement l'ajout de bug.

Nos Rôles

Nous étions tous des développeurs full stack. Tout le monde a touché aux moins une fois au front et au back.

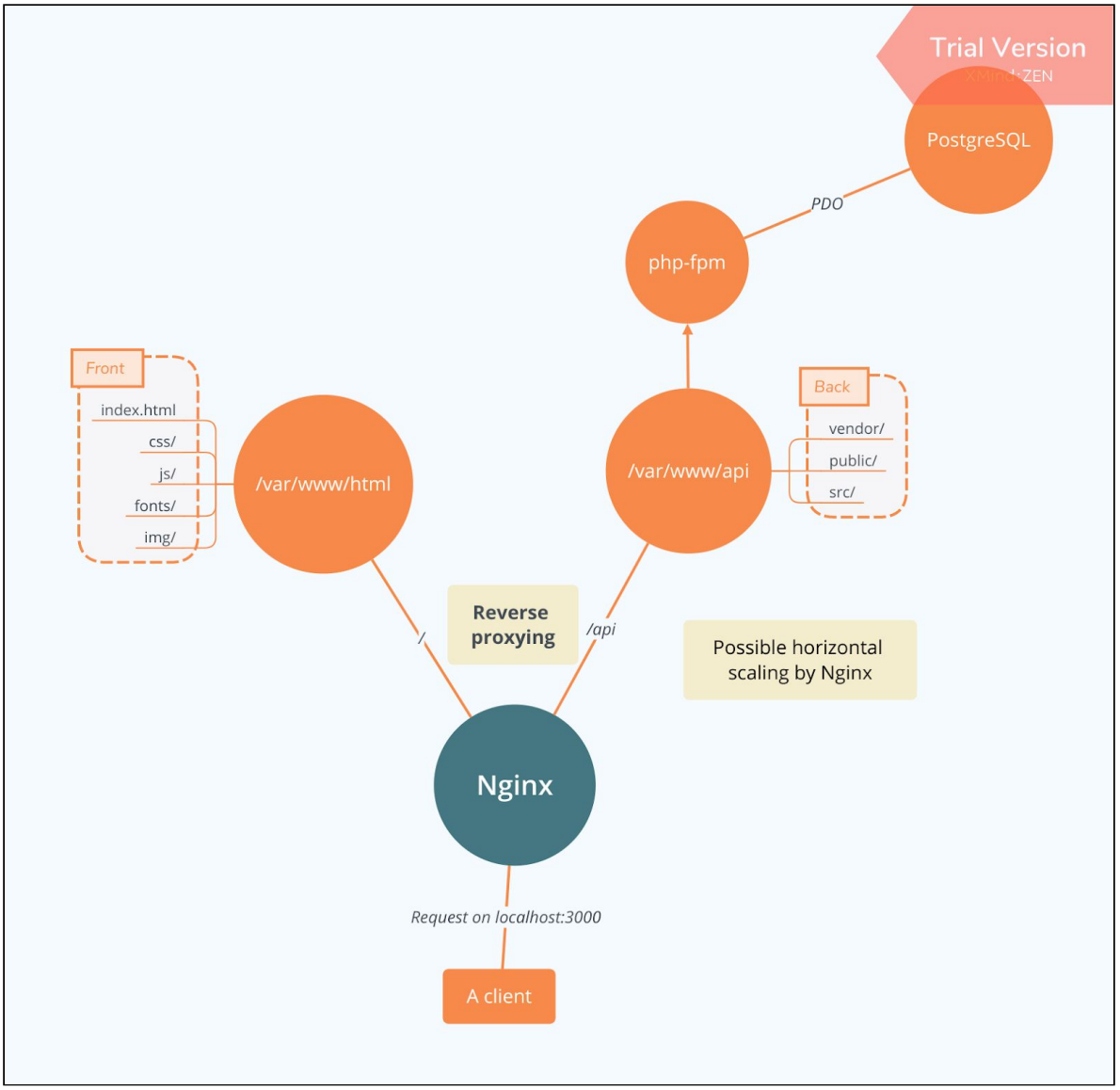
Swann => Architecte logiciel, installation de la tuyauterie, mise en place des bonnes pratiques de test, code review et CI/CD. Mise en place de Vuejs, routeur, store et gestion de l'authentification.

Emilio => Développeur / Graphiste. Choix du design et des couleurs pour le projet. Création des templates de pages. Choix du framework graphique Vuetify

Jerôme => Développeur / Responsable Sécurité. Respect des règles de sécurité pour une application web.

Samir => Développeur R&D, Administrateur Base de données Exploration des possibilités de recherche avec Google API + création de la base de données et intégration de l'API

Architecture de l'application



Problématiques rencontrées

Utilisation d'une Single Page Application

Les avantages d'une Single Page Application sont conséquents et permettent une expérience utilisateur accrue. Ne pas avoir à recharger complètement la page permet de créer de belles transitions mais cela apporte beaucoup de problématiques propres aux SPA. L'utilisation massive de JavaScript peut ralentir le navigateur et peut possiblement ralentir le réseau, bien que les navigateurs modernes soient ultra performants sur l'interprétation du JavaScript. Nous avons dû résoudre plusieurs problèmes concernant l'utilisation d'une SPA.

Premièrement, la communication entre notre backend et notre frontend. Elle est donc forcément asynchrone et passe par des requêtes XHR depuis le navigateur. Pour pouvoir répondre à ce type de communication, il existe plusieurs types de Web Services comme du SOAP qui est assez ancien, verbeux et plus trop utilisé actuellement ou du REST qui est la solution retenue. Nous avons configuré Nginx pour transmettre toutes les requêtes sur l'URL "/api" au fichier index.php, puis créer un routeur maison pour gérer nos endpoints REST.

Ensuite, l'utilisation massive de JavaScript. Nous avons utilisé un outil incontournable pour créer un projet JavaScript front et gérer la minification, le lazy-load, le TreeShaking et la modularisation de notre Javascript, j'ai nommé Webpack. Ce "bundler" JavaScript, nous a permis d'optimiser nos fichiers Javascript et ainsi réduire la taille de ceux-ci. Nginx permet aussi de gérer facilement les headers qui seront envoyés aux différents navigateurs et nous a permis de leur dire de mettre en cache tous nos assets (html, js, img et fonts). la combinaison lazy-load et le système de cache des navigateurs nous permet d'alléger considérablement les appels réseaux.

Un autre problème avec une SPA est la gestion des sessions. Comme les fichiers HTML ne proviennent plus d'un script PHP, nous ne pouvions plus utiliser les fonctions natives du PHP pour gérer les sessions. Après avoir pensé différents tokens très basiques pour authentifier un utilisateur, nous avons opté pour une implémentation du JWT pour plusieurs raisons. C'est utilisé par beaucoup de protocoles (de délégation) d'authentification comme OpenID Connect ou OAuth2 et donc pour pas mal de SSO. Et c'est aussi très utilisé pour des architectures microservices qui nécessitent des authentifications décentralisées. Cela a eu un réel intérêt pédagogique de l'implémenter sur ce projet. Il a fallu aussi sécuriser nos endpoints lorsque ceux-ci nécessitent des droits, et ainsi vérifier que le token donne suffisamment de droits pour la ressource demandée.

Gestion des bars

La gestion des bars s'est faite par itération. La première à été réalisé naïvement en allant récupérer une liste des meilleurs bar parisien par quartier. Il nous fallait une base de bar pour pouvoir développer les autres fonctionnalités. Cette solution à l'avantage de proposer des bar existants, c'était un choix de l'équipe que d'être en phase avec la réalité. En revanche, cette solution n'en propose qu'un échantillon.
La problématique de la prochaine étape était : comment avoir plus de vrai bar.
Pour se faire deux solutions, se distingue :

- OpenData : gratuite mais ne fournit pas les noms des établissement et reste très limité.
- Google Places : très (très) complète mais plus compliqué à prendre en main. En revanche la facturation est assez floue, nous avons utilisé le quota gratuit mais sans en connaître vraiment les limites.

Nous avons donc décidé d'utiliser l'API de Google dans le but de profiter de l'incroyable quantité de bar référencé, de plus l'offre gratuite nous a permis de l'utiliser sans payer (normalement). A ce stade notre base de données reste initialement statique met elle est

indirectement alimenté par l'API de google. La structure d'un bar a donc dû évoluer pour inclure *placeId, lat, lng*, *rating* et *photoReference* qui sont des champs propres à l'API.
A ce stade notre base est remplie de plus de bar avec plus d'information venant de google, notre base est donc prête à accueillir des bars directement de l'API.
L'étape suivante consiste à permettre à un utilisateur d'interroger l'API et d'ajouter des bars dans notre base de données depuis l'interface. Ceci est possible grâce à la page /addbar. Elle met à disposition une barre de recherche permettant de soumettre des mots clefs au back qui lui effectue la requête auprès de google, s'il y a des résultats ceux ci sont formaté puis renvoyer au front. L'utilisateur a ensuite l'opportunité "*d'aimer*" un bar ce qui l'ajoute à notre base de données ainsi qu'à la liste 'liked' de l'utilisateur (inexploité pour le moment). A ce stade lorsqu'un user 'like' un bar celui ci est ajouté à la base et ces mots clés sont aussi associé au bar. L'application est donc 'auto-suffisante', elle peut fonctionner sans insertion manuelle de bar. Pour le moment les bars liké ou blacklisté ne sont pas pris en compte dans le feed.

Utilisation de Vue.js

On pourrait croire que faire du Vue est une solution de facilité, sa réactivité permet de réévaluer le DOM HTML automatiquement mais cela apporte son lot de complexité supplémentaire. La syntaxe des composants Vue est vraiment simple mais nous devions quand même comprendre le fonctionnement sur la liaison des données, la communication des composants parent / enfant et la gestion des événements.

Parfois, nous avions besoin de gérer un état global sur le front pour la gestion de l'authentification ou pour les informations sur l'utilisateur courant. Cette problématique est bien connue des développeurs frontend et une des solutions pour le résoudre est l'utilisation d'un store global avec le pattern Flux. L'implémentation du pattern Flux pattern, en Vue, est Vuex. Son utilisation nous à permis de gérer très simplement l'authentification ainsi que les informations de l'utilisateur. Il fallait cependant bien distinguer les différentes parties de Flux afin de respecter leurs différentes responsabilités.

Afin d'améliorer l'expérience utilisateur, sur notre site, nous avons utilisé le système de routage du Vue. Son utilisation est très simpliste et nous a permis d'organiser notre code en fonction des pages. Cela nous a permis également de créer des transitions très plaisantes pour les yeux. Il fallait cependant configurer Nginx pour qu'il laisse la main au routeur front lorsqu'une requête était à sa destination: par exemple l'url "http://localhost:3000/feed". Si mal configuré, Nginx aurait essayé de résoudre cette url sur le file system et le front n'aurait jamais analysé cette url.

Le Salage en PHP

Nous avons été confronté à une problématique répandue dans le hachage des mots de passe qui est le salage ou "salt". Le salage est une technique visant à renforcer la sécurité des données hachées en ajoutant une donnée supplémentaire au mot de passe. Nous nous sommes posé la question sur notre solution pour ajouter du salage mais elle n'était pas du tout sécurisé, soit la donnée du salage se retrouvait dans notre repository Git

soit elle différait de celle en production. En regardant de plus près les bonnes pratiques sur PhpTheRightWay, nous avons trouvé une solution native à Php. Cette solution nous a permis d'être totalement sécurisé et d'utiliser les meilleurs algorithmes de hachage en vigueur.

Finalité du projet

Ce que ça nous a apporté :

- Beaucoup de plaisir à réaliser ce projet et nous avons globalement répondu au problème rencontré
- Mise en place de bonnes pratiques du développement logicielle
- Vision actuelle d'une application Web, utilisation de concepts modernes peut importe la technologie (REST API, JWT, conteneurisation…)
- Utilisation de nouvelles technologies JavaScript (JS moderne ES6+, Vue.js et son écosystème, Axios)
- Utilisation d'une librairie de composant graphique front-end basée sur Material Design, Vuetify

Bonus

Nous avons configuré une pipeline d'intégration et de déploiement continu sur un serveur privé. Nous avons utilisé l'outil Travis CI avec 2 responsabilités: lancer les tests pour s'assurer que l'on n'introduit pas de régression dans le code. Puis déployer l'application sur notre serveur si les tests passent. Cela demandait aussi une certaine rigeure pour configurer des variables d'environnements spécifiques à la production ainsi que l'installation de la base de données sur le serveur cible.