WhatEat - Rapport de projet

Version 0.1

26.08.2022

Table des matières

Cours PDG - Rapport de projet

Dans le cadre du cours PDG de la HEIG-VD, nous devons réaliser un travail de groupe qui consiste à développer une application web.

Notre application va vous aider pour tous vos problèmes de repas. Avec **WhatEat** vous obtenez tous les jours des repas personnalisés selon vos préférences.

Groupe de travail

- Rui Filipe Lopes Gouveia (rui.lopesgouveia@heig-vd.ch) - ISCR - Dev frontend
- Thibault Seem (thibault.seem@heig-vd.ch) - ISCL - Dev backend
- Joachim Bailat (joachim.bailat@heig-vd.ch) - ISCS - Dev backend
- Hadrien Louis (hadrien.louis@heig-vd.ch) - ISCL - Chef de projet
- Damiano Mondaini (damiano.mondaini@heig-vd.ch) - ISCL - Dev frontend

Technologies utilisées

Pour la réalisation de ce projet, nous avons utilisé les technologies suivantes :

- Javascript
  - Nous avons décidé de faire une application web, car cela nous permet d'avoir l'application facilement accessible depuis les Smartphones sans devoir se battre avec le Play Store ou l'App Store. Dans l'équipe certains membres étaient plus à l'aise avec JS que PHP ou Java ce qui explique notre choix.
- Node.js
  - Nous avons décidé d'utiliser Node.js comme moteur pour le code JS du backend.
- React
  - Plusieurs membres de l'équipe avaient déjà connaissance de React plutôt que VueJS. C'est donc pour cette raison que nous avons fait ce choix.
- ExpressJS
  - ExpressJS est devenu de facto le framework backend à utiliser en conjonction avec Node.js pour la création d'une API.
- Base de données MongoDB et Mongoose
  - Nous avons fait le choix d'utiliser une base de données MongoDB. Nous avons fait ce choix car plusieurs membres de l'équipe ont déjà travaillé avec cet outil, de plus, Mongo Atlas permet d'héberger gratuitement des DB dans le Cloud. Nous utilisons également Mongoose afin de faciliter la communication entre Node.js et MangoDB.

- Serveur Heroku
  - Après avoir comparé plusieurs services d'hébergement de site web, nous avons fait le choix d'utiliser Heroku. Cet outil est simple à utiliser et gratuit pour les projets non commerciaux. Nous avons utilisé cet outil pour l'hébergement du backend.
  - Nous avions commencé par y héberger le frontend, mais nous nous sommes rendus compte que Heroku avait besoin de « buildpack » pour faire fonctionner React dessus. Après plusieurs tentatives nous avons décidé de l'héberger chez Netlify.
- Serveur Netlify
  - Après avoir essayé de déployer le frontend sur Heroku, nous avons rencontré divers problèmes ce qui nous a poussé à déployer le frontend sur un serveur différent. Nous avons choisi de le mettre sur Netlify ce qui nous permet de faciliter le déploiement.
- Repository Github avec Kanban
  - Nous avons décidé d'utiliser les outils de Github pour la gestion des tâches et ce afin d'appliquer la méthodologie agile.
- Automatisation des tests avec Github Actions
  - Nous avons décidé d'utiliser Github Actions pour executer nos tests d'intégration automatiquement
- Jest pour les tests d'intégration
  - Nous nous sommes aussi renseignés sur Mocha et Chai. Nous avons choisi Jest, car plusieurs membres l'on déjà utilisé auparavant.
- Prettier pour le formattage du code
  - Nous avons préféré Prettier à ESlint car cela est plus facile à utiliser selon nous, car nous ne sommes pas encore très à l'aise avec ce type d'outil.
- Spooncular API
  - Nous avons cherché les différentes API de recettes à disposition et Spooncular est resortie comme étant très complète, mais payante. Nous les avons contactés car nous avons vu qu'il y avait une offre pour les Hackathon ou les projects académique.

Méthodologie

Pour ce projet, nous avons choisi de travailler en appliquant la méthode agile tout en appliquant certains aspects de la programmation eXtreme ainsi que SCRUM. Nous avons fait ce choix car XP se distingue en 4 points qui correspondent parfaitement à nos besoins.

- Cycles de développement court
- Planification progressive
- Planification flexible
- Tests automatisés

De plus, nous trouvons que les valeurs et principes de cette méthode correspondent à notre philosophie de travail et permettent une bonne cohésion de groupe.

Gestion du projet

Ce paragraphe a pour objectif de décrire la manière dont nous appliquons l'agilité dans ce projet.

Client

Dans le cadre de ce cours PDG, il n'y a pas de client principal. Le projet, ses contraintes et ses limites sont imposées par nous-même. Nous avons donc décidé d'une liste de fonctionnalités à implémenter et à partir de ces dernières, nous créerons des sprints contenant les user stories à réaliser.

User stories

Les users stories sont une description simple d'un besoin ou d'une attente exprimée par l'utilisateur final. Ces stories représentent donc un objectif final et non une fonctionnalité.

La rédaction des stories a été faite au début du projet et sont listées dans le cahier des charges. Au début de chaque sprint, ces stories sont décomposées en tâches ce qui alimente notre product backlog. Toutes les tâches ne doivent pas forcément être faites durant le sprint mais peuvent être faites plus tard. L'équipe sélectionne les tâches qu'elle pense être en mesure de réaliser durant le temps donné.

Tableau Kanban

Pour pouvoir mettre en pratique les user stories, nous avons décidé d'utiliser un tableau Kanban. Ce dernier contient les 4 colonnes suivantes :

- Todo : La liste globale des tâches
- In progress : Les tâches en cours de réalisation
- Waiting merge : Les tâches en attente de merge
- Done : Les tâches terminées

Il est important de noter que les stories correspondent aux besoins fonctionnels du projet. Ces stories sont par la suite décomposées en tâches qui sont insérées dans le backlog produit.

La gestion de ce tableau est semi-automatisée. En effet, l'ajout d'une fonctionnalité dans le backlog se fait automatiquement lors de la création d'une issue. Cependant, la sélection des tâches à réaliser (Colonne Todo) se fait durant le meeting de début de sprint puis lorsqu'une personne commence à

travailler sur une issue elle est en charge de déplacer la tâche dans la colonne Doing. Les colonnes Waiting Merge et Done sont aussi automatisées.

Une fois la tâche terminée et la PR merge avec la branche `develop`, la tâche est automatiquement close et déplacée dans la colonne Done.

Sprints

Chaque cycle de développement se fait en sprint. Au début de chaque sprint un meeting est organisé entre les membres du projet afin de définir les tâches à réaliser. L'objectif étant qu'à la fin du sprint toutes les tâches à faire soient terminées. Pour l'instant, la durée d'un sprint est équivalente à une semaine.

A la fin de chaque sprint, une deuxième réunion est organisée afin de faire un debrief sur le sprint et de re-évaluer certaines tâches si elles n'ont pas été terminées.

Méthodes de développement

Organisation des branches

Nous avons choisi d'utiliser l'organisation de branches suivantes pour notre projet :

- Branche `main` contenant la dernière version stable du logiciel. Cette branche est protégée et il est nécessaire qu'une PR soit faite depuis la branche `develop`. Cette PR doit être validée par un autre membre de l'équipe afin d'être merge.
- Branche `develop` contenant la version en cours de développement du logiciel
- Branche `fb-` pour les nouvelles fonctionnalités à ajouter
- Branche `fix-` pour des bugs à résoudre

Gestion des issues

- Un template d'issue a été créé afin que toutes respectent le même format et ce afin de faciliter la compréhension et la lisibilité.
- Une issue doit contenir une brève description, la liste des tâches à réaliser, la durée ainsi que les éventuelles dépendances à d'autres tâches requises.
- Lorsqu'un membre de l'équipe commence à travailler sur une tâche, il s'associe la tâche et crée une nouvelle `feature branch` pour y écrire le code. Il doit également faire une estimation du temps nécessaire (Optimiste, Pessimiste, Attendu)

- Une fois la feature terminée, il faut créer une PR depuis Github et assigner au moins une personne afin de review le code. Elle doit être validée par au moins un autre membre du groupe avant d'être merge par son créateur ou quelqu'un d'autre.
- Une fois la PR validée, il est alors possible de merge la feature branche avec `develop`, supprimer la branche et fermer l'issue.

Messages de commit

Afin de garder une certaine uniformité dans nos messages de commits, nous avons décidé d'utiliser la convention de nommage suivante.

Chaque commit doit être préfixé par les labels suivants :

- [WIP] - pour les work in progress
- [Done] - lorsque qu'une tâches est terminée
- [Fix] - lorsqu'un commit fix un bug

Nous utiliserons l'anglais comme langue principale pour les messages de commit. Nous avons également décidé de signer tous les commit avec PGP.

Création d'une nouvelle version du logiciel

Comme cité précédemment, les `feature branch` sont merge avec la branche `develop` dès que la PR est validée. A la fin d'un sprint, une personne est en charge de merge `develop` avec la branche `main` afin qu'une nouvelle version du logiciel soit disponible.

Tests

Pour ce projet, nous avons décidé d'implémenter des tests unitaires pour le backend afin de vérifier le bon fonctionnement de chaque composant ajouté au projet. L'objectif est donc de garantir qu'après chaque ajout de fonctionnalité, aucune autre ne fonctionne plus. Nous essaierons dans la mesure du possible d'aborder une approche TDD au fur et à mesure, cependant, nous ne sommes pas encore familier avec une telle approche.

Ces tests seront rédigés avant ou en même temps que l'ajout d'une fonctionnalité et ce au moyen de l'outil Jest. La validation de tous les tests sera une condition requise afin que cette nouvelle fonctionnalité puisse être ajoutée au projet.

Mise en place des outils de développement

Nous avons décidé d'utiliser VSCode comme IDE avec Node.js et npm pour l'execution du JS et la gestion des dépendances. Nous avons aussi installé git pour pouvoir envoyer le code vers Github.

Lors de la phase de développement, nous faisons tourner le backend et le frontend en local sur des ports différents. Des fichiers `.env` ont été créés pour les deux projets et ce afin de préciser les éventuelles variables globales (URL de la DB, port, mot de passe, etc..)

Avec une telle architecture, il est tout à fait possible de faire des modifications sur le frontend sans endommager le backend et vice-versa. La backend de l'API a également été conçu afin de prendre en charge des versions d'API `/v1/users` et `/v2/users`. Cette solution nous permet aussi de modifier certains endpoints du backend sans corrompre les éventuels appels à des anciens endpoints.

En ce qui concerne la base de données, nous avons fait le choix de créer deux base de données. Une de prod qui sera utilisée par l'application en ligne (Celle utilisée par nos users) puis une base de données de dev que nous utilisons lorsque nous faisons tourner le projet en local sur nos machines.

Pour ce qui est des tests unitaires, ils sont effectués avec Jest. Ils peuvent être lancés en local ce qui va créer une DB locale dans la RAM uniquement pour les tests. Nous avons fait ce choix afin d'éviter des éventuels problèmes si plusieurs dev lancent les tests en même temps.

Mise en place d'un environnement de déploiement

Nous avons 3 cloud provider : * Heroku : Il est utilisé pour host le backend de l'application gratuitement, il watch le repo whateat-backend, dès lors qu'une modification est détéctée sur le repo, il récupère les changements, rebuild de son côté et « serve » le tout.

- Netlify : Il est utilisé pour host le front gratuitement, un bot netlify est installé sur notre repo il lance les tests, et de la même manière qu'Heroku, Netlify watch le repo whateat-front et à chaque changement il récupère les données et les « serve ».
- MongoDB Atlas : Il est utilisé pour host nos bases de données gratuitement (production et développement)

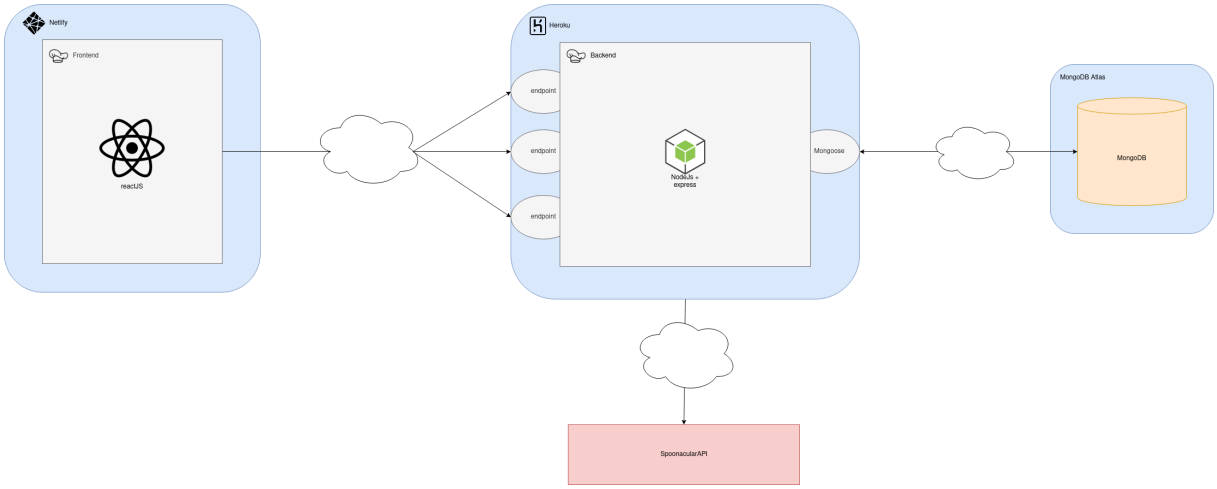

FIGURE 1 – Infrastructure

Mise en place d'un pipeline de livraison et de déploiement (CI/CD)

Nous avons mis en place 2 pipelines, un pour le frontend, l'autre pour le backend.

Les deux pipelines ont la même logique de base à savoir : 1. Le développeur merge sa feature branch sur la branche de dev 2. Toutes les fin de sprints la branche dev est merge sur la branch main : * Les CI/CD Github sont executées à ce moment * Un développeur doit review et valider le merge 3. Lorsque la branch main a été changé les clouds providers (Netlify, Heroku) viennent prendre les derniers changements et s'occupe de serve ce contenu.

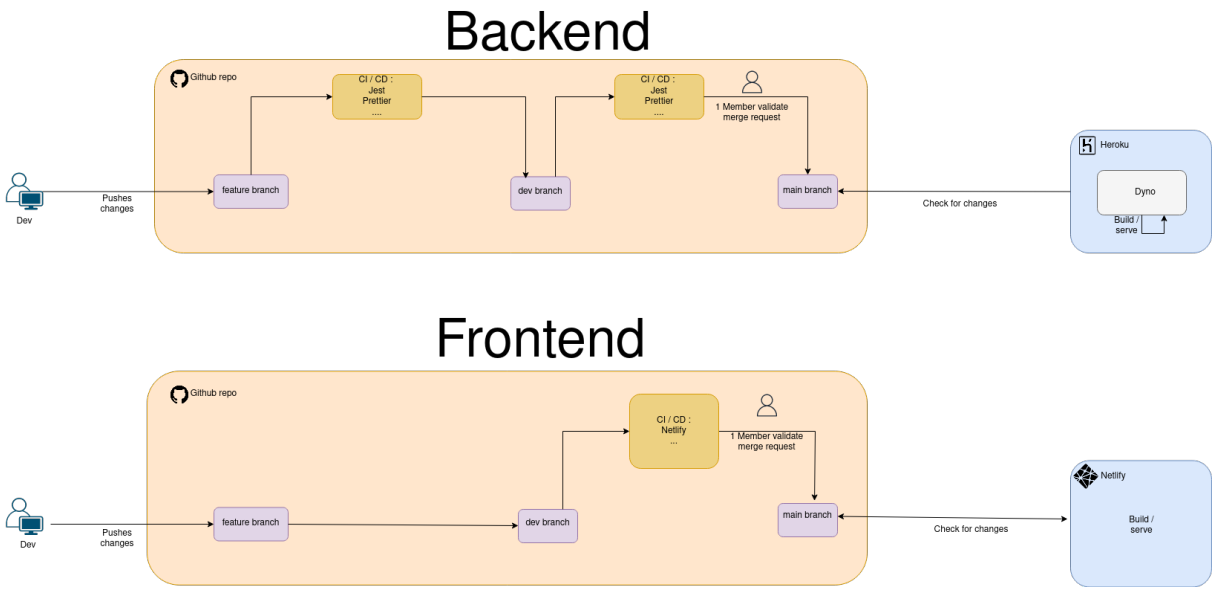

FIGURE 2 – Pipeline drawio