Projet CAPI

Duy LEVAN Quentin LIM | Conception agile de projets | Valentin Lachand-Pascal

Table des matières

INTRODUCTION

Ce projet dans la cadre de la conception agile de projets avait pour but de simuler une application de planning poker qu'on a vu en cours. Ce projet a été réalisé en binôme et en pair programming.

Nous avons choisi d'utiliser Node js pour l'application et de la faire en locale.

Malheureusement nous avons rencontré un problème au niveau de la sauvegarde du fichier json qui nous a empêcher de finir le projet, on développera ce bug plus tard dans la partie Justification des choix techniques.

JUSTIFICATIONS DES PATTERNS UTILISES (3 PATTERNS) (3PTS)

Pour ce projet nous avons utilisé le pattern Singleton pour les Controller(il ne peut y en avoir qu'un). En effet avec Duy, on a décidé de partir sur une architecture MVC, de ce fait, on avait prévu de mettre les patrons de conception suivants :

-singleton pour les Controllers, (Un seul et unique pour chaque, on ajoute le décorateur private sur le constructeur et une méthode publique statique qu'on utilise pour ne le créer qu'une seule fois, s'il existe déjà on avertit l'utilisateur)

Exemple pour l'utilisateur :

```
export default class UserController {

    static instance;

    /**
     * Méthode statique pour obtenir l'instance unique de UserController
     * @returns {UserController} - L'instance unique de UserController
     */
    static getInstance(userModel, userView) {
        if (!UserController.instance) {
            UserController.instance = new UserController(userModel, userView);
        }
        return UserController.instance;
    }

    /**
     * Créer une class UserController.
     * @param {object} model - Le modèle
     * @param {object} view - La vue
     * @param {number} nbUsersMax - Nombre d'utilisateurs maximum, par défaut 4
     * @private
     */
    constructor(model, view, nbUsersMax = 4) {
        if (UserController.instance) {
            throw new Error("Une instance de UserController existe déjà. Utilisez UserController.getInstance() pour l'obtenir.");
        }
```

Figure 1 - Pattern Singleton

Dans notre fichier app.js(fichier principal) on instancie de cette manière notre controller :

```
const userController = UserController.getInstance(userModel, userView);
```

-prototype pour la classe CarteModel,(On veut créer des variantes pour la carte café et la carte interro, qui sont très proche des autres cartes mais différentes, qu'on aurait pu cloner)

```
/**
 * Crée une copie de l'objet CarteModel
 * @returns {CarteModel} - Copie de l'objet CarteModel
 */
clone() {
    return new CarteModel(this.carte);
}
```

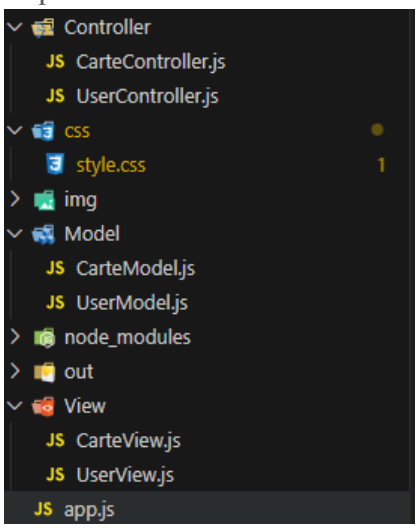
Figure 2 - Pattern Prototype, conception : clonage de carte café et interrogation

-Factory pour la classe CarteVue(créer un objet pour chaque carte où on stock sa valeur)

Au cours de la conception, on s'est rendu compte que ce n'était pas forcément utile de créer une classe CarteVue pour chacune des cartes, car il ne contiendrait qu'un seul champ unique qui est sa valeur. Nous nous sommes adaptez à cela et n'avons pas prioritisez l'implémentation du design factory(car pas nécessaire au fonctionnement de l'application).

JUSTIFICATION DES CHOIX TECHNIQUES (ARCHITECTURE, LANGAGE, CLASSES, ETC.)

Pour commencer, il faut savoir que mon binôme et moi avons des niveaux différents car formation différente. Pour ma part j'avais des connaissances en Framework web (Vue, React, Next) alors que Duy ne connaissait que le javascript natif, on ne savait pas comment faire une interface en python ou en java donc on s'est mis d'accord pour faire en javascript natif. De plus on pensait que cela nous permettrait de facilement implémenter une architecture Client-Serveur. On a par la suite étudié le sujet et nous avons décider que nous avions besoins de 2 classes principales, une pour gérer les utilisateurs et une pour les cartes (système de vote etc...).



Nous avons donc choisi d'utiliser le MVC pour séparer le modèle des vues (interface utilisateurs).

Au cours de l'implémentation, il faut l'avouer que ces choix techniques n'ont pas été les plus optimaux et qu'on aurait dû y réfléchir un plus. En effet on était en pair programming et quand Duy a commencé, on s'est vite rendu compte que ça n'allait pas, j'ai donc pris la main sur le code et Duy réfléchissait à l'infrastructure. C'était la première fois qu'on faisait un projet en javascript natif et la tâche était beaucoup plus ardu que prévu (problème de routage, props, variables globales, et surtout la séparation

Client/Serveur). Avec du recul je pense qu'on aurait été beaucoup plus efficace si j'avais imposé d'utiliser Next 13, car je maîtrise le routage par dossier et la séparation Client/Serveur est très claire (il faut mettre en en-tête d'un fichier « use client » et c'est tout !). Ce problème de Client/Server nous a malheureusement fait très mal car afin de sauvegarder et update le fichier json, qui contient le backlog, on voulait utiliser la librairie « fs » qui est incorporé dans Node js,.Mais, on a échouer à bien segmenter la partie client du serveur et n'avons pas réussi à trouver une solution car notre fichier principal app.js est côté client(création du DOM etc...). De ce fait nous n'avons pas pu implémenter la partie sauvegarde du fichier.

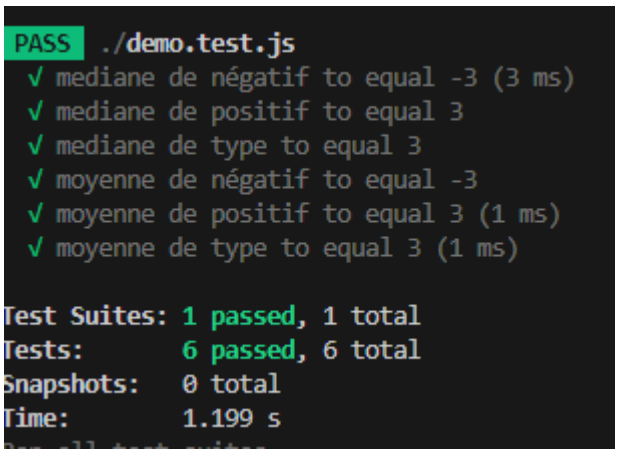
DOCUMENTATION (2PTS)

Concernant la documentation, nous avons utilisé JSDoc avec Doxygen pour la générer. Au début on ne savait pas quoi mettre comme balises, mais on a vite pris l'habitude de mettre la balise @param quand il y en avait, le type de @return, @class et surtout dans la balise description, on décrivait ce que le code correspondait effectue.

On a aussi automatisé la génération de documentation avec GitHub Actions, en configurant le workflow comme montrer dans la ressource déposée sur Moodle.

EXPLICATION DE LA MISE EN PLACE DE L'INTEGRATION CONTINUE (2PTS)

Pour continuer nous avions tout d'abord organiser des séances hebdomadaires avec Duy où on travaillait ensemble dans l'espace coworking de ma résidence, on définissait entre nous les objectifs à atteindre et on se fixait des prochaines missions/fonctionnalités à implémenter pour la prochaine séance. On déposait à chaque fin de séance le code sur GitHub. On avançait plutôt bien jusqu'à la deuxième page, où on a rencontré notre premier problème : le routage. C'est à partir de ce moment là qu'on a commencé à se demander si on ne devait pas recommencer depuis le début en Next 13. Toutefois je ne voulais pas imposer à mon binôme d'apprendre un Framework juste pour un projet, nous avons donc continuer en javascript natif. Un majeur problème qu'on avait pendant l'expérience de peer programming et que Duy est un étudiant étranger et avait du mal à comprendre le français et l'anglais. Notamment pour la compréhension des énoncées, on a dû utiliser des outils de traductions vers le vietnamien. Le peer programming, bien qu'en théorie très efficace a été dans notre cas complexe à mettre en place. Concernant les tests unitaires, nous avons rencontrées plusieurs problèmes, on voulait écrire les tests avant de coder mais on ne savait pas quoi tester exactement. On a donc des tests uniquement pour des fonctions assez évidentes (calcul de la médiane et de la moyenne). Un autre problème est qu'on voulait aussi les intégrer à



GitHub Actions, mais le workflow ne marchait pas à cause de configuration de permission, en revanche les tests marchaient en local. Nous avons comme pour JsDoc suivi les ressources déposées sur Moodle, on a donc utilisé jest et configurer Node Js.

Pour continuer, nous avons aussi mal estimées la difficultés des tâches car on pensait bien maîtriser le javascript mais sans Framework c'est une autre histoire. De plus en termes de temporalité, on aurait dû mettre beaucoup plus de séance et de sprint au début du projet car vers la fin il y avait un cumul avec les examens/autres projets de la fac.

CONCLUSION

Ce projet fut très surprenant car il nous a permis de découvrir de nouvelles choses qui vont s'avérer très utiles pour notre avenir professionnel. Pour notre part c'était la première fois que nous faisions autant de documentation avec JSDoc et nous ne savions même pas que c'était possible de l'automatiser de cette manière.

La découverte des workflows bien que très rapide nous on a montré les possibilités offertes par GitHub et les services d'intégration continus. Le peer programming est aussi une piste qu'on a pu appliquer et l'implémentation des tests aussi (bien que regrettable qu'on en ait si peu car satisfaisant de les voir passer).

En conclusion, avec mon binôme, nous partageons le même avis qu'on a sous-estimé la difficulté du projet. On pensait à n'avoir à effectuer « qu'une petite application web », néanmoins le fait de ne pas s'être plus focalisé sur l'architecture, et de faire des compromis sur le langage (qu'on ne maîtrise pas forcément) pour qu'on soit tout les deux au même niveau nous a été fatale.

En tout cas cela nous a servi de leçon et la prochaine fois nous saurons qu'il vaudrait mieux détailler nous aider plutôt que nous dire par exemple on va faire un MVC et un Client/serveur en js. (Comment implémenter le Client/serveur, est-ce que le MVC est vraiment pertinent ? Les patrons qu'on a choisis ont -ils réellement du sens ? Il vaudrait mieux quelqu'un qui connaît bien une technologie et son binôme non plutôt que deux personnes qui connaissent légèrement une technologie, on implémente des tests mais quels tests faire ?) – Toutes ces questions et remarques que nous tirons de ce projet nous aiderons surement pour nos projets informatiques futur.