Documentation

pROJET 4

Pierre Abi Chedid – Lama Keyrouz | NFP121 | 8-16-2021

# Comment présenté le projet ?

Après avoir choisi notre projet, nous avons décidé de repartir les taches d’une façon équitable et juste, dans le but de pouvoir travailler ensemble de manière cohérente. Pour cela, il nous semblait évident que notre projet prenne la forme d’une application web. Grace à ce choix, nous aurons l’opportunité de vous présenter tout au long de cette documentation et dans notre présentation finale :

* Plusieurs patrons de conception
* Une interface interactive
* La manipulation des données à travers une base de données et d’une logique implémentée dans le code coté serveur.

## DISTRIBUTION DES TACHES

* Lama Keyrouz : Front-End. Cela implique le travail sur l’interface et les diverses options offerte à l’utilisateur à travers l’application pour que notre client ait une expérience idéale lors de sa visite sur notre application web.
* Pierre Abi Chedid : Back-End. Cela implique le travail sur la manipulation des données, la logique derrière les diverses fonctionnalités offertes par le site, et les restrictions de certaines possibilités données à l’utilisateur.

## Languages and frameworks

Que ce soit pour leur compatibilité, leur popularité, ou leur structure moderne qui permet l’implémentation de plusieurs patrons de conception, il nous semblait évident d’utiliser :

* JavaScript pour notre front-end.
* NodeJs pour notre back-end (utilisant le modèle REST pour les APIs)

# Analyse

Les 5 Ws :

What : un site Web interactif utilisé pour aider à planifier et organiser les cours des utilisateurs tout en suggérant le plan optimal en forme de calendrier pour les données saisies avec les salles et les heures disponibles

Where : Un site web utilisant pour le coté front-end le langage JavaScript et pour le côté back-end le langage de NodeJs

When : Le site Web est disponible à chaque fois que l'utilisateur se connecte.

Who : Les étudiants qui suivent des cours.

HoW : L'utilisateur entre dans l'écran d'accueil où il peut se connecter ou s'inscrire, puis il entre dans les principales fonctionnalités du site où il peut ajouter, visualiser, éditer et associer des informations sur ses cours. Un calendrier sera automatiquement créé et mis à jour à chaque fois qu'il ajoute où modifier de nouvelles informations.

**Coté Front-End :**

Le patron « MVVM » a était utiliser pour une meilleur structure et architecture de code

# Problèmes

De nombreux problèmes se sont produits lors du développement du côté frontal :

* La création des objets en JavaScript et l’ajout des propriété utilise beaucoup de ligne de code si l’objet a plusieurs propriétés
* De nombreux widgets, pages et objets ont beaucoup de propriété et fonction en commun ce qui conduit à écrire plusieurs fois le même code qui sera difficile à maintenir et gérer.
* Certains objets qui contiennent des configurations n’ont pas besoin de créer un nouvel objet à chaque fois que l'objet de configuration est requis quelque part dans le système.
* La répétition des parties d’un algorithme avec de très petites modifications.
* JavaScript dans sa forme simple ne prend en charge ni les classes ni les modificateurs d'accès, et un développeur ne peut pas déterminer les droits d'accès pour ses membres
* Faible communication entre des parties disparates
* Le parcours de donne et la manipulation d’une collection d'objets complexe est inefficace

# Solutions

L’utilisation des patrons de conceptions offre une solution approprier à ces problèmes

* Le patron ‘Constructor’ aide a simplifié la création des objets en utilisant la fonction constructrice et l’initialisation de ses propriétés de la même manière que nous le ferions avec un constructeur de langage classique
* Le patron ‘Factory’ aide à gérer, maintenir et manipuler des collections d’objets qui sont diffèrent mais en même temps on de nombreuse caractéristique en commun.
* Le patron ‘Singleton’ aide à avoir des classes utilisant une seule instance commune
* Le patron ‘Template’ aide à définir le squelette d'un algorithme dans une opération, en reportant certaines étapes aux sous-classes. Elle permet aussi aux sous-classes de redéfinir certaines étapes d'un algorithme sans changer la structure de l'algorithme.
* Le patron ‘Module’ aide a imité le comportement nécessaire en utilisant les fermetures, nous pouvons créer des objets avec des parties privées et publiques.
* Le patron ‘Observer’ est un outil très utile lorsque nous avons un scénario où nous devons améliorer la communication entre des parties disparates de notre système de manière optimisée. Il favorise un couplage lâche entre les objets.
* Le patron ‘Iterator’ résout ce problème en séparant la collection d'objets du parcours de ces objets en implémentant un itérateur spécialisé.

**Coté Back-End**

# Problèmes

Plusieurs problèmes se présente lors de la construction du code coté serveur.

* Le premier étant l’implémentation d’une structure claire et cohérente qui permettrait un changement dynamique du code si nécessaire tout en prenant en compte les demandes de la base de données et la logique qui doit être implémenté.
* Un autre problème est les différents services demandés par notre application, et comment les gérés de façons à ne pas répéter le même code plusieurs fois et à donner une claire représentation de mon travail et des différentes fonctions utiliser.
* Comment être sûre que l’utilisateur est authentifié lors de la visite du site web ?
* Comment diriger les diverses requêtes demandées par le site web ?

# Solutions

L’utilisation des patrons de conceptions offre une solution approprier à ces problèmes

* Le concept **MVC** (modele-view-controlleur) permet de clairement distribuer les diverses fonctions que le code coté serveur nécessite. Pour cela, avec chaque entité demandée pour l’application, un folder est attribué possédant le modèle de nos entité (schema) et un contrôleur qui gère nos différentes fonctions. La partie view est géré par le front-end et cela à partir de l’interface offerte.
* Le concept ‘**Command’**. Ce concept aide à visualiser les différents services que l’application nécessite. Un fichier ‘services.js’ contient les commandes des différentes fonctionnalités de l’application. Ces commandes sont ensuite appelées par le contrôleur d’une façon claire et méthodique. Ainsi, un même service peut être appeler plusieurs fois sans répéter le code et peut être dynamiquement modifier à partir des différentes demandes du site.
* Grace au **Middleware** concept ! Pour vérifier le JSON Web Token, et être sûre que tous mes APIs utilise soit sous effet de l’authentification, j’ai créé un middleware ‘is-auth.js’ qui pourra être appeler à chaque appel API et vérifier que l’utilisateur a bien été authentifié.
* Grace au concept **Proxy.** En implémentent un ‘router.js’ fichier dans chaque folder, chaque requête est traitée différemment, (par un contrôleur diffèrent) et cela dépend du la requête demander par le front-end.