FERNANDES Quentin / HADJARI Idir

Application de Planning Poker

Conception Agile de Projets Informatiques

Table des matières

Présentation globale du projet : 2
Justifications des Patterns utilisés : 2
Justification des choix techniques : 3
Documentation : 4
Schéma de l'architecture du site : 4
Détail du rôle de chaque page : 4
Dictionnaire des variables 6

Présentation globale du projet :

L'application que nous avons réalisée correspond à une application de Planning Poker : le Planning Poker est une application web conçue pour faciliter le processus de planification pour les équipes de développement Agile. L'application que nous avons développée est donc un outil interactif permettant d'estimer le temps nécessaire pour accomplir des tâches dans le contexte du développement Agile. Le projet utilise une architecture web simple, basée sur HTML, CSS et JavaScript. Il propose des fonctionnalités pour l'initialisation du jeu, la gestion des joueurs, la saisie des estimations, etc., que nous allons toutes développer plus bas.

Justifications des Patterns utilisés :

Pour réaliser ce projet, nous avons essayé d'incorporer quelques patterns, sans les appliquer à l'entièreté du code, afin de simplement les tester.

En premier lieu, nous avons essayé d'utiliser un patron architectural de type MVC dans la mesure où nous avons fait le choix de séparer les vues, à savoir les fichiers html, des contrôleurs et modèles, à savoir le fichier Javascript. Nous avons décidé d'utiliser ce patron architectural afin de faciliter la gestion et la maintenance du code par la séparation des vues des modèles et contrôleurs d'une part, de pouvoir mieux changer les vues si besoins sans avoir à toucher à la logique métier sous-jacente d'autre part, mais également de mieux gérer les interactions entre l'application et les utilisateurs grâce au fichier javascript, qui a un rôle de contrôleur lui permettant réagit aux événements générés par l'utilisateur ou le système et de mettre à jour le modèle et la vue en conséquence.

Nous avons également essayé d'utiliser quelques modules patterns dans la mesure où nous avons tenté de regrouper au maximum les gestionnaires d'événements spécifiques à chaque page entre eux afin d'améliorer la visibilité du code, mais également l'ajout de fonctions de navigation entre les pages, tel que la fonction **pageAccueil2**, ainsi que l'utilisation de fonction pour gérer la logique liée au débat et à la validation des avis, tel que les fonctions **debat** ou **validAvis1**, toujours dans un soucis d'améliorer la lisibilité du code. Nous avons ainsi décidé d'utiliser des modules patterns pour la facilité de compréhension et de maintenance du code permise grâce aux modules qui encapsulent des parties spécifiques du code.

Enfin, nous avons également testé un pattern observer en utilisant l'événement de chargement de la page pour appeler la fonction **commencerJeu**. Nous avons tenté d'utiliser ce pattern pour sa capacité à réagir au changement qui est très utile dans le cas de notre application afin de charger les paramètres de la partie en début de celle-ci.

Ainsi, nous ne nous sommes pas limités à un seul pattern puisque nous avons fait le choix d'en tester plusieurs afin de voir à quel point ils permettent d'améliorer la lisibilité du code.

Justification des choix techniques :

Pour réaliser ce projet, nous avons décidé d'utiliser une architecture de type MVC, comme évoqué plus haut.

Concernant le choix du langage, nous avons décidé d'utiliser le langage Javascript car il s'agit d'un langage que nous avons tous les deux étudiés lors de ce semestre dans le cadre du cours « Programmation Web Avancée ».

Nous avons également, dès le début du projet, envisagé de réaliser une application dans laquelle l'utilisateur peut naviguer entre plusieurs pages html car, au moment du début du projet, nous ne savions pas vraiment comment faire une interface graphique avec du Python, alors que nous avons réalisé quelques « jeux » en Javascript qui nécessitaient d'associer HTML et fichier Javascript dans le cours « Programmation Web Avancée ».

Enfin, une dernière raison à ce choix a été notre volonté à renforcer nos connaissances et compétences dans ce langage avec ce projet : ce projet à ainsi été l'occasion d'apprendre de nouveaux mécanismes du langage javascript.

Nous avons fait le choix de ne pas utiliser la programmation orientée objet pour cette application car nous avons jugé que cela allait introduire de la complexité inutile, notamment par la création deux classes « Joueurs » et « Fonctionnalités » : ainsi, au lieu de créer un objet « Joueur » pour chaque joueurs qui possèderait comme attributs son numéro, son pseudo et son vote à une fonctionnalité (qu'il aurait fallu réinitialiser à chaque changement de fonctionnalité », nous avons préféré utiliser deux listes, une contenant le pseudo du joueur et dont le numéro est associé à sa position dans cette liste (par exemple, le joueur à la position '0' est le joueur 1, etc.), et une autre liste pour enregistrer les votes des joueurs à une fonctionnalité et qui est réinitialisé à chaque nouveau vote (là encore, le vote du joueur 1 est à la position « 0 », etc.).

Pour réaliser cette application, nous avons choisi d'utiliser un seul fichier javascript plutôt que plusieurs fichiers spécifiques à chaque pages html car nous avons jugé, en raison de la faible taille du code, qu'avoir un seul fichier permettait d'une part de simplifier la gestion du code, d'autre part de charger ce fichier plus rapidement sur le navigateur car il y a moins de requêtes réseau à effectuer, mais aussi afin d'avoir moins de complexité pour gérer les dépendances et l'ordre d'exécution du code.

Nous avons utilisé plusieurs variables globales pour cette application, ainsi que créée plusieurs fonctions, présentées plus bas, afin de gérer les différents événements de la partie.

Nous avons veillé à annoter le code et à séparer les fonctions spécifiques à chaque page html dans le fichier javascript afin de mieux distinguer les fonctions qui relèvent de chaque page. Ainsi, les fonctions spécifiques à la page *initialisation.html* sont parmi les premières fonctions disponibles dans le fichier, tandis que les fonctions spécifiques à la page *partie.html* se trouvent vers le bas du fichier.

Enfin, il est important de noter que par manque de temps, nous ne sommes pas parvenus à implémenter les mécanismes d'intégration continue.

Documentation :

Schéma de l'architecture du site :

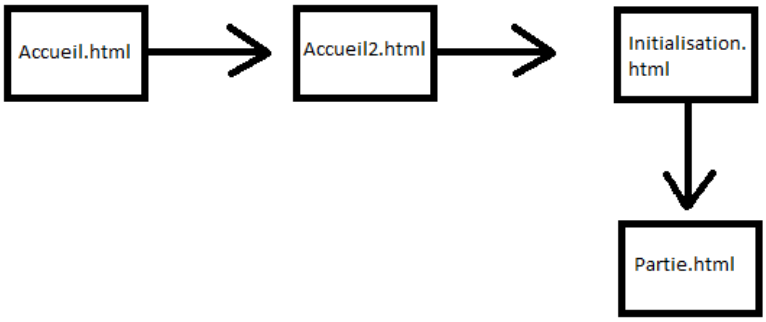

Figure 1: Schéma des liens entre les différentes pages de l'application

L'application est constituée de 4 pages html dans lesquelles les utilisateurs peuvent naviguer en respectant le « chemin » représenté par le schéma : lorsque l'application se lance, les utilisateurs sont dans la page *« accueil.html »,* puis ils redirigés vers la page *« accueil2.html »*, avant d'être redirigés vers la page *« initilisation.html »,* avant de finalement atteindre la page *« partie.html* ».

Détail du rôle de chaque page :

Page « accueil.html » :

Cette page ne contient qu'un seul bouton, un bouton « commencer » qui redirige l'utilisateur vers la page « accueil2.html ». Aucune fonction ne se déclenche sur cette page.

Page « accueil2.html » :

Dans cette page, l'utilisateur dispose de 3 boutons qui lui permettent de soi commencer une nouvelle partie, auquel cas il sera redirigé vers la page « initilisation.html », de charger une partie déjà commencer, auquel cas il pourra choisir le fichier JSON de la partie déjà entamé, ou bien revenir au menu d'accueil, auquel cas il sera redirigé vers la page « accueil.html ».

Page « initialisation.html » :

Avec cette page, l'utilisateur règle les paramètres de la partie : il doit d'abord saisir le nombre de joueur, puis leurs pseudos respectifs dans des zones de textes dédiées, puis le nombre de fonctionnalités que compte le projet, ainsi que les intitulés de ces fonctions, et enfin le mode de jeu, qui peut être soit l'unanimité, soit la moyenne. Le paramétrage de la partie s'effectue dans cet ordre puisque l'affichage de chaque champs s'effectue selon les boutons de validations qui leurs sont associés et qui sont liés à des fonctions du fichier javascript : 5 fonctions sont ainsi associées à cette page :

- ***afficherPseudos***, qui affiche les zones de texte de sélection des pseudos ;
- ***afficherNbrFonction*** qui affiche la barre de sélection du nombre de fonctionnalités ;
- ***afficherFonction*** qui affiche les zones de texte de sélection des intitulés des fonctionnalités ;
- ***afficherModeJeu*** qui affiche la liste déroulante de sélection des règles de la partie ;
- ***pagePartie*** qui stock les variables relatives au mode de jeu choisi (*modeJeu*), aux pseudos et aux fonctionnalités qui seront votées dans le localStorage, et redirige le joueur vers la page « partie.html » (voir le dictionnaire des variables pour plus de précision sur les variables).

Ces différentes fonctions se déclenchent en fonction d'écouteurs placés sur les boutons de validation de chaque champ des paramètres, et qui sont gérés par des conditions (des « if ») afin de vérifier si l'utilisateur est bien sur la page qui nécessite d'activer ces fonctions.

Page « partie.html » :

Cette page correspond à la page de jeu sur laquelle les participants attribuent différents niveaux de difficulté à chaque fonctionnalité. Pour cela, les utilisateurs disposent de plusieurs cartes issues du planning poker, et ils n'ont qu'à cliquer sur l'une des cartes afin d'attribuer une difficulté estimée à chaque tâche que comprend la partie. Sur cette page, les utilisateurs peuvent voir le numéro et l'intitulé de la fonctionnalité débattue, ainsi que le numéro et le pseudo du joueur qui doit voter.

Lorsque tous les joueurs ont voté et qu'ils ne sont pas tous d'accord, deux zones de texte prennent la place des cartes pour permettre aux deux joueurs extrêmes d'expliquer les raisons de leurs votes respectifs. Enfin, lorsque la partie est terminée, l'application enregistre un fichier JSON contenant le nom des fonctionnalités, leurs difficultés attribuées par les utilisateurs, les règles du jeu qui ont été utilisées, ainsi que la liste des pseudos des joueurs qui ont participé à ces votes.

Plusieurs fonctions définies dans le fichier javascript sont associés à cette page :

- ***commencerJeu***, qui récupère les paramètres définies lors de l'initialisation de la partie, tel que les pseudos et les fonctionnalités, les stocks dans des listes adaptées, et gère l'affichage du pseudo du premier joueur qui doit voter, ainsi que la première fonctionnalité qui sera voté ;

- ***handleMouseUp,*** qui gère toutes les actions arrivent après la sélection d'une des cartes par un utilisateur, tel que la gestion du plus petit vote (grâce aux variables ***voteMin*** et ***joueurMin***), la gestion du plus grand vote (grâce aux variables ***voteMax*** et ***joueurMax***), l'affichage du pseudo du joueur qui doit à présent voter, la gestion du cas où tous les joueurs ont voté la carte « café », la gestion du cas où tous les joueurs ont voté « ??? », du cas où tous les votes sont égaux (donc qu'il y a unanimité), du cas où tous les joueurs ne sont pas d'accord (et auquel cas la fonction ***debat*** est appelée), ainsi que les cas où les règles sont « unanimité » ou « moyenne ».
- ***debat,*** qui masque les cartes, réinitialise la liste contenant les votes des utilisateurs pour une fonctionnalité donné, et affiche la première zone de débat ;
- ***validAvis1,*** qui récupère la justification du joueur ayant réalisé le vote le moins élevé pour une fonctionnalité donné, la stock dans le local storage, masque la première zone de débat et affiche la seconde ;
- ***validAvis2,*** qui récupère la justification du joueur ayant réalisé le vote le plus élevé pour une fonctionnalité donné, la stock dans le local storage, masque la seconde zone de débat et réaffiche les cartes ;
- ***tourFonction,*** qui gère l'affichage de la fonctionnalité qui doit être débattu lorsque la fonctionnalité précédente à été validé, ou bien appel la fonction ***finPartie*** s'il ne reste plus de fonctionnalités ;
- ***finPartie,*** qui récupère la liste des joueurs, la liste des fonctions, la liste des notes attribuées à chaque fonction, et les règles du jeu et les enregistre dans un fichier JSON directement dans l'ordinateur depuis lequel est utilisée l'application.
-

Dictionnaire des variables

Voici un rapide présentation des principales variables de l'application, à savoir els variables globales :

- ***listJoueur*** : il s'agit d'une liste qui stocke les pseudos des joueurs. Le joueur à l'indice 0 est le joueur 1, le joueur à l'indice 1 le joueur 2 etc. Elle est notamment utilisée afin de connaitre le pseudo du joueur qui doit voter pour la difficulté d'une fonctionnalité. Elle est censée contenir entre 3 et 12 joueurs (en fonction du nombre de joueurs définis dans la page *« initialisation.html »).*
- ***listFonction*** : Il s'agit là encore d'une liste qui est très similaire à la ***listJoueur***, mais pour les fonctionnalités : cette liste sert ainsi à stocker les fonctionnalités qui seront débattues au cours de la partie. Elle peut contenir entre 1 et 15 fonctionnalités (en fonction du nombre de fonctionnalités définies dans la page *« initialisation.html »).*
- ***noteJoueur*** : Il s'agit d'une liste qui va stocker le vote des joueurs pour une fonctionnalité donnée. Les valeurs contenues dans cette liste peuvent être n'importe quelles valeurs présentes sur les cartes de planning poker, y compris ' ???' et 'cafe', en fonction des votes des joueurs. Les indices des votes et les indices des joueurs sont ainsi les mêmes afin de retrouver rapidement le vote de chaque joueur pour une fonctionnalité donnée : ainsi, la valeur située à l'indice 0 de la liste correspond au vote du joueur 1, à l'indice 1, c'est le vote du joueur 2, etc. Cette liste est réinitialisée, ou plutôt toutes ses valeurs sont effacées, chaque fois que les joueurs doivent procéder à un vote (après qu'ils aient tous voté pour une fonctionnalité donnée, et que cette dernière ait été validée ou non). La longueur de cette liste est censée être la même que la liste ***listJoueur***.

- ***noteFonction*** : Cette liste fonctionne comme la liste ***noteJoueur*** puisqu'elle stock les valeurs attribuées par les joueurs à chaque fonctionnalités débattues dans la partie. Là aussi, la valeur située à l'indice 0 correspond à la difficulté estimée par les joueurs pour la première fonctionnalité (en fonction des règles de jeu choisies), etc. La longueur de cette liste est censée être la même que la liste ***listFonction***.
- ***tourVote*** : cette variable stock le nombre de tour de vote pour une fonctionnalité : par exemple, lors du premier vote des joueurs sur la fonctionnalité 2, sa valeur est de 1, puis si les joueurs votent une seconde fois pour cette même fonctionnalité, sa valeur sera de 2 etc.
- ***showPseudos*** = C'est une variable qui contient la balise html dans laquelle se trouve les zones de texte de saisie des pseudos des joueurs, ainsi qu'un bouton de validation.
- ***showMode*** : C'est une variable qui contient la balise html dans laquelle se trouve la liste déroulante de sélection des modes de jeu, ainsi que le bouton de validation du mode de jeu.
- ***showFonction*** : C'est une variable qui contient la balise html dans laquelle se trouve les zones de texte de saisie des fonctionnalités de la partie, ainsi qu'un bouton de validation.
- ***showNbrFonction*** : C'est une variable qui contient la balise html dans laquelle se trouve la barre de sélection du nombre de fonctionnalités que comptera la partie, ainsi que le bouton de validation.

- ***voteMin*** : Cette variable contient la valeur du plus petit vote d'un joueur pour une fonctionnalité. Elle est notamment comparée avec la variable ***voteMax*** afin de savoir si tous les joueurs ont voté la même chose pour une fonctionnalité, ou non. Elle est réinitialisée à chaque nouveau vote.

- ***voteMax*** : Cette variable contient la valeur du plus grand vote d'un joueur pour une fonctionnalité. Elle est notamment comparée avec la variable ***voteMin*** afin de savoir si tous les joueurs ont voté la même chose pour une fonctionnalité, ou non. Elle est réinitialisée à chaque nouveau vote

- ***joueurMin*** : Cette variable contient le pseudo du joueur ayant réalisé le plus petit vote pour une fonctionnalité. Cette variable permet ainsi de savoir quel joueur devra participer au débat si tous les joueurs ne se sont pas mis d'accord pour évaluer la difficulté d'une fonctionnalité.

- ***joueurMax*** : Cette variable contient le pseudo du joueur ayant réalisé le plus grand vote pour une fonctionnalité. Cette variable permet ainsi de savoir quel joueur devra participer au débat si tous les joueurs ne se sont pas mis d'accord pour évaluer la difficulté d'une fonctionnalité.

- ***regle*** : Cette variable contient le nom de la règle de jeu qui a été choisie dans la page « initialisation.html ». Elle permet ainsi d'appeler certaines fonctions selon les règles choisies (comme la fonction **debat** que nous verrons plus bas), ainsi que des traitements spécifiques à certaines variables.

- ***cartes*** : variable qui contient les balises html de toutes les cartes de la page *partie.html.* Cette variable permet ainsi de gérer les cartes lors de la partie, en permettant par exemple de leurs appliquer des animations ou bien de récupérer leurs valeurs en cas de clique de la part des joueurs sur celles-ci.