WOW-COMPARE

Cahier des charges

Francois1.guergadic@epitech.eu

Sommaire

* Description générale du projet
* Technologies utilisées
* Scope/Domaine du projet
* Description des différentes parties du projet
* Decoupage en taches fonctionnelle des différentes parties du projet

Description générale du projet

Wow-Compare est une application web destinée aux joueurs du jeu World of Warcraft.

World of warcraft est un jeu de type MMORPG. Le cote MMO de ce jeu implique que les joueurs doivent monter des groupes pour arriver a bout de certains contenu propose par les développeurs.

La creation de groupe est souvent fastidieuse car les joueurs manque de visibilité sur les compétences des personnes avec qui ils jouent.

Certains site internet proposent déjà de scanner un personnage pour avoir un petit peu plus d’informations sur lui, mais les informations ne sont pas regroupes, et oblige le chef de groupe a chercher les informations pour chacun des membres du groupe.

Le but principal de Wow Compare est donc d’avoir simplement a importer d’un coup tous les membres du groupe dans lequel le joueur est, et de chercher a la place du joueurs les informations relatives a chaque personne du groupe et de lui afficher de manière plus lisible.

Technologies utilisées

- L’application web va être une application en React.js avec ce qui implique un développement en React. L’outil Redux va être utilise pour l’avantage que procure son store et pour permettre une gestion plus facile de nos données.

- Le développement en React utilisera ES6

- Pour la partie DevOps, je vais utiliser l’outils d’integration de GitLab qui permet l’execution de composant docker configurable pour effectuer des taches après des événement sur le depot.

- L’hébergement sera fait sur heroku directement avec le plan gratuit pour le moment.

Scope/Domaine du projet

Le projet va commencer directement des la validation du cahier des charges avec un gros rush pendant les vacances de de fin d’années. Le développement va subir un léger ralentissement fin janvier du a une mise a joue importante du contenu sur World of Warcraft qui pourrait impacter le projet.

L’idée est d’avoir déjà une premiere version utilisable avant le 23 janvier pour pouvoir tester l’outils des l’ouverture de contenu sur Wow.

La partie DevOps sera la partie effectuée en priorité par rapport aux autres.

Le but de ce projet est de developper un projet fonctionnel et utilisable, le tout en essayant de produire le code le plus propre possible afin de pouvoir le montrer plus tard a des entreprises.

Il intègre une bonne fourchette d’outils nécessaire a un développeur et devrait être un bon exemple.

Le fait de rendre ce projet accessible par un nom de domaine devrait permettre a certains joueurs de wow de l’utiliser et de pouvoir donner des retours d’experience d’utilisation.

Description des différentes parties du projet

# Partie 1: Data Fetching

Cette partie est destinée a la recuperation des donnes provenant de site extérieur ou du jeu. Nous récupérons en jeu certaines information concernant les joueurs, ces informations sont importées sur notre webapp avec un simple copie / colle dans une text box. Une fois les informations importées sur la webapp, nous allons a l’aide d’api publique enrichir les données que nous avons pour chaque joueurs.

Cette partie traite donc tout la recuperation de ces donnes sans les traiter. La gestion de notre modele de donnée sera effectue aussi par cette partie.

# Partie 2: Data Analysis

Une fois les informations récupérées, nous allons les traiter ici. Chaque informations arrive sous un certain format que nous allons adapter pour matcher notre modele de donnée défini au dessus.

Certaines informations arrivent et son vraiment trop brutes pour être affichées a l’utilisateur telles que nous les avons. Nous allons donc les retravailler pour en extraire les informations importantes, en regroupant, en supprimant, ou en calculant nous même certaines données.

# Partie 3: DevOps

La partie DevOps regroupe toutes les actions a effectuées sur le projet qui ne touche pas directement a l’application web.

Nous allons faire en sorte que le projet soit disponible sur des outils de versioning (GitHub) et que notre version en ligne de l’application ce mette a jour automatiquement en fonction de certains événements sur notre outil de versioning. Nous voulons une version en ligne stable et une deuxième version nous permettant de tester.

La version en ligne stable embarquera certains elements d’analyses nous permettant de revoir les actions effectuées par un utilisateur, de voir le nombre d’utilisateurs ou de voir les différentes erreurs que nous avons.

# Partie 4: Front Development

Cette partie regroupe tout le développement de la partie visible de l’application. Le développement est découpé sous forme de composants. Chaque composant regroupe un element de l’interface et doit pouvoir être modulable au maximum.

Decoupage en taches fonctionnelle des différentes parties du projet

Data Fetching

Redux:

- Mise en place du store initial

- Mise en place du reducer: Pour chaque service -> 3 actions: REQUEST, SUCCESS et ERROR

- Application du Middleware (Redux Thunk)

- Arborescence des dossiers: Separation du reducer, des services et des actions.

Raider io:

- Creation du service de fetching

- Creation des actions

- Gestion d’erreur

- Remplissage du store Redux dynamiquement grace au reducer

Warcraft Log:

- Creation du service de fetching

- Creation des actions

- Gestion d’erreur

- Remplissage du store Redux dynamiquement grace au reducer

Data Analysis

Importation des informations venant du jeu:

- Recuperation de tous les joueurs du groupe et de tous les joueurs inscrits dans la file d’attente. Pour chaque joueur on veut son nom et le nom du serveur sur lequel il joue.

- Construire la base de notre modele de donnée avec les informations récupérées. Les joueurs seront stockes dans un tableau, chaque joueur étant un object JSON nous permettant plus tard d’aller chercher ses informations sur les sites externes.

Traitement des données venant d’api:

- Regrouper les combats par boss

- ROW\_DATA: Calculer le nombre de dégâts mitigés, de point de vie soignés ou de dégât infligés selon le role du personnage pour chaque combat

- RANK: Pour chaque combat, y associer la performance du joueur en % par rapport aux autres joueurs, 0% étant le plus faible, 100% la meilleurs performance effectuée.

- Pour chaque personnage, et pour chacun des boss, garder la moyenne de ROW\_DATA, le max de ROW\_DATA, la moyenne de RANK, le maximum de RANK

- Algorithm de notation en fonction des elements au dessus donnant une note a un joueur allant de S+ a D-

DevOps

Norme de développement:

- Installation du linter airbnb

- Configuration du linter

- Application des techniques de développement propre au linter airbnb (PropTypes etc..)

Integration continue:

- Bloquer le git pour que avoir des branches protégées (une branche pour production, une pour staging)

- Lancement du Linter a chaque push sur toutes les branches

- Si il a un push sur la branche master ou develop, mettre en ligne le code de la branche sur l’url propre a la version en production ou a la version en staging

- Faire en sorte que les outils d’analytics soit mis que sur la version en production

Analytics:

- Mise en place Fullstory

- Mise en place de Google Analytics

- Mise en place de Sentry

Front Development

Composant React:

- Composant principal (Container)

- Composant barre de texte pour importer les informations venant du jeu

- Graphique de repartition des classes

- Graphique de repartition des roles

- Tooltip sur les graph

- Composant buff manquant

Composant du groupe:

- Nom du joueur (Couleur en fonction de la classe + icône de role + serveur)

- Experience du joueur en Raid sous forme de ProgressBar

- Experience du joueur en donjon a l’aide d’un score

- Affichage du classement du joueur sous forme de % par rapport aux autres joueurs sur chacun des boss(avec un code couleur)

- Affiche du niveau d’objet équipé du joueur avec le code couleur qui va avec