Intégration continue pour le web

TP n°4 Le developpement web

Lors de ce TP, nous allons commencer le développement du jeu **Les Bâtisseurs - Moyen-Âge**. Dans un premier temps, je vous conseille soit de lire les règles soit de regarder la vidéo explicative.

Partie 1: Récupération du projet

Pour récupérer le projet, nous allons nous rendre sur le dépôt Gitlab hébergeant le starter. Une fois sur la page connectée à Gitlab, il suffit de forker le projet avec le bouton *fork* en haut à droite [aide]. Cela va créer une copie du dépôt vous appartenant afin que chacun puisse travailler sur son propre dépôt.

Je vous conseille de passer le dépôt en privé dans les paramètres afin d'éviter toute tentations [aide].

Maintenant que vous avez votre copie, il suffit de la télécharger sur votre ordinateur via la commande `git clone [URL]` où l'url est celle indiquée en haut à droite lorsque l'on clique sur le bouton *clone*. Il existe deux façon de cloner un projet, soit en utilisant SSH soit en utilisant HTTPS. L'avantage de SSH est qu'il n'est pas nécessaire de taper son login/mot de passe à chaque `git push` mais il faut générer et configurer une clé privée [aide]. Je vous conseille d'utiliser HTTPS pour l'instant.

Partie 2 : Démarrage du projet

Je vous conseille d'utiliser Visual Studio Code lors de vos développement. C'est un IDE développé par Microsoft et Open Source très utilisé dans le monde Javascript. Il intègre un système de plugins, une interface graphique pour Git, un terminal intégré et pleins d'autres fonctionnalités. Une liste de plugins recommandés est disponible dans le Readme du projet.

Une fois le projet ouvert, ouvrez un terminal depuis *code* (Menu Terminal -> New Terminal) puis installer les dépendances *Node* via la commande `npm install`. Un répertoire *node_modules* a été créé contenant l'ensemble des dépendances de l'application.

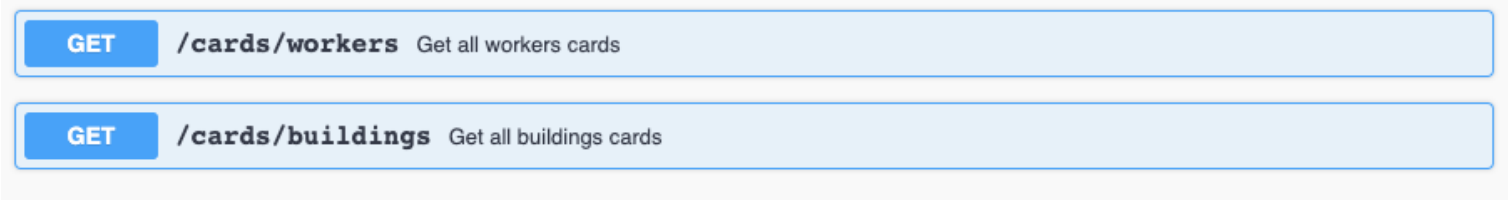
Il suffit maintenant d'utiliser la commande `npm run start` pour démarrer l'application. Rendez-vous sur `http://localhost:3000/health` et le message `{ health: "ok" }` devrait s'afficher confirmant le bon fonctionnement du site 🎉.

Avant de commencer à développer, il est important de **toujours** travailler dans une branche. Pour cela nous allons créer une branche `feature/cards-list` avec la commande `git branch feature/cards-list` puis nous déplacer dessus avec la commande `git checkout feature/cards-list`. N'hésitez pas à découper votre travail en plusieurs commits. Pour rappel, les commits doivent être **atomique** avec une description **claire**.

Partie 3 : Création de nouvelles routes

Nous allons développer deux nouvelles routes permettant d'afficher l'ensemble des cartes du jeu. Une documentation détaillée de l'ensemble des routes attendues est disponible sur `http://localhost:3000/api-docs/`. Il est **important** de suivre cette documentation lors du développement du projet sinon le Front-end ne marchera pas.

Lors de ce TP, nous allons développer `GET /cards/workers` et `GET /cards/buildings` des routes permettant de récupérer respectivement les cartes des ouvriers et les cartes des bâtiments.



On va créer un nouveau router `src/routes/cardRouter.js` et y coller le code suivant :

```
import express from "express"
const router = express.Router()

// routes

export default router
```

et l'inscrire dans le fichier `src/routes/index.js`.

```
router.use("/cards", cardRouter)
```

On va maintenant initialiser les deux routes répondant à notre besoin.

```
router.get("/workers", function(req, res) {
  const workers = [] // C'est ici que l'on récupérera nos ouvriers
  res.json(workers)
})

router.get("/buildings", function(req, res) {
  const buildings = [] // C'est ici que l'on récupérera nos bâtiments
  res.json(buildings)
})
```

Rendez-vous sur `http://localhost:3000/cards/workers` et `http://localhost:3000/cards/buildings` et vous devriez obtenir un tableau vide 🎉.

Partie 4 : Création du service

⚠️ Ne jamais écrire du code métier dans des routeurs. Le code métier s'écrit dans des services. Il faut garder les routeurs les plus simples possibles.

Nous allons créer un fichier service `src/services/cardService.js` afin d'écrire notre code métier en dehors de notre routeur. Nous pourrons importer et utiliser notre nouveau service dans notre routeur. Un service est simplement un module exportant des fonctions réutilisables effectuant du code métier. Par exemple, lire et retourner une liste de cartes est du code métier. On peut déjà imaginer que notre service comportera deux fonctions `importWorkers()` et `importBuildings()`.

```
import fs from "fs"

export async function importBuildings() {
  // Import buildings code
}

export async function importWorkers() {
  // Import workers code
}
```

Vous n'avez plus qu'à coder ces deux fonctions sachant que la liste des cartes se trouve au format CSV dans le répertoire `src/ressources` et qu'il existe la fonction fs.readFile() pour lire un fichier.

```
// __dirname correspond au répertoire du fichier courant `src/services/`
fs.readFile(__dirname + "/../ressources/buildings.csv", "utf8", (err, file) => {
  if (err) {
    console.error(err)
    return
  };
  console.log(file) // Contenu du fichier
})
```

ℹ️ La fonction readFile utilise un callback, il peut être intéressant (😉) de créer un wrapper la transformant en promesse.

Il nous suffit maintenant de remplacer le code dans le routeur par l'utilisation de notre service et tout devrait fonctionner 🎉 .

```
// ...
import * as cardService from "../services/cardService"
// ...
router.get("/workers", async function(req, res) {
  const workers = await cardService.importWorkers()
  res.json(workers)
})

router.get("/buildings", async function(req, res) {
  const buildings = await cardService.importBuildings()
  res.json(buildings)
})
// ...
```

ℹ️ Pour tester vos nouvelles routes, vous pouvez soit utiliser votre navigateur soit le logiciel Postman [aide].