

Feuille de travaux pratiques

Mini-projet : échecs en ligne

Objectifs

La fin du semestre approche, il est temps de montrer tout ce que vous avez appris et de proposer au monde entier votre première application web (webapp) ! L'objectif de ce mini-projet est d'intégrer et adapter tout le travail réalisé lors des TP et TD précédents, afin de réaliser une application de jeu d'échecs jouable (et observable) en multijoueur réseau.

Rendu

L'ensemble de votre projet doit être archivé au format zip et déposé sur Madoc au format

`NOM1_NOM2_NOM3_NOM4.zip`

avant la date indiquée sur la zone de dépôt. Chaque fichier supplémentaire ajouté dans le projet doit contenir en entête un **commentaire** comportant une description du fichier et l'ensemble de vos noms et prénoms. Inspirez-vous des sources fournies dans le projet. Vous pouvez également modifier le fichier `README` pour y décrire l'organisation de vos modules, les fonctionnalités et les changements apportés dans le projet ainsi que les limites rencontrées.

Exercice 1 (Mise en place)

Ouvrez une console et récupérez le projet git :

```
git clone https://gitlab.univ-nantes.fr/tonneau-q/onlineChess.git
```

Le dossier obtenu est composé des fichiers suivants :

- `README.md` : fichier de présentation du projet (page principale gitlab)
- `.gitignore` : liste des fichiers à ignorer par git
- `package.json` : informations du projet en cours de développement et liste des modules dépendants
- `tsconfig.json` : configuration de la compilation TypeScript
- `client` : fichiers destinés au navigateur (page web, feuille de styles, scripts)
- `serveur` : modules et algorithme principal du serveur
- `build` : dossier (automatique) où seront stockés les fichiers compilés (.js) pour ne pas polluer le reste de l'arborescence

Afin de lancer un serveur web, ce projet nécessite d'importer des modules externes depuis internet, en l'occurrence les bibliothèques `express` et `body-parser` précisées dans le fichier `package.json`.

Afin d'importer automatiquement ces dépendances, placez-vous à la racine du projet et lancez la commande

```
npm install
```

Fonctionnement de l'application

Le fichier principal du serveur (`main.json`) est chargé de démarrer un mini-serveur web capable de recevoir les différentes requêtes provenant des navigateurs connectés à l'application :

- `"/` : distribue le fichier `index.html`
- `"/ (post)"` : reçoit et traite un coup à jouer
- `"/status.js"` : génère et distribue l'échiquier en cours

Ces trois traitements correspondent aux différents `app.get` et `app.post` du fichier principal.

Chronologie d'une partie :

1. Lorsqu'un utilisateur se connecte à l'application (adresse "/"), le serveur distribue alors la page html principale composée d'un échiquier vierge et d'une zone de saisie permettant à l'utilisateur de remplir le coup à jouer.
2. Le navigateur internet récupère immédiatement les informations de la partie en cours présentes à l'adresse `/status.js` et remplit l'échiquier à l'aide d'un script situé dans le fichier `script.js`.
3. Un clic sur le bouton "Envoyer" effectue une requête de type *POST* au à l'adresse / du serveur, contenant les informations du champs de texte associé. Le serveur traite alors la requête afin de jouer le coup demandé.
4. La page internet du joueur est alors rechargée automatiquement, affichant ainsi le nouvel état de la partie.
5. etc...

Travail à réaliser

Exercice 2 (Initialiser l'échiquier côté serveur)

Le fichier **main.ts** proposé est également incomplet et ne possède pas de gestion de la (les) parties.

1. Importez les modules développés dans les TP précédant dans le répertoire du projet
2. Adaptez à vos algorithmes la première PARTIE ELEVE 1 du fichier serveur **main.ts**. N'oubliez pas d'importer les différentes interfaces et fonctions avant de les utiliser. Vous devriez obtenir les informations d'une partie vierge en vous rendant manuellement sur la page <http://localhost:8080/status.js>

Exercice 3 (Remplir l'échiquier côté navigateur)

Adaptez maintenant le code PARTIE ELEVE 3 situé dans le fichier **script.js** pour remplir l'échiquier (table html) à l'aide des informations présentes dans la variable `echiquier` (correspondant à la page précédente).

Nous arrivons enfin à générer un échiquier côté serveur et l'afficher dans le navigateur !

Exercice 4 (On la joue cette partie ?)

Il ne nous (vous !) reste plus qu'à récupérer les commandes que les joueurs envoient à l'adresse / (**post**), analyser la commande et modifier l'échiquier en conséquence. Quelle chance, vous avez déjà développé la grande majorité des fonctions nécessaires (analyse de coup, recherche de pièce, déplacement).

1. Importez les modules et fonctions nécessaires à l'analyse et l'exécution d'un coup sur un échiquier
2. Adaptez la PARTIE ELEVE 2 du fichier serveur **main.ts** pour vos algorithmes
3. Testez le fonctionnement de votre application d'échecs

Exercice 5 (Et après ?)

Voici quelques pistes d'amélioration de votre projet avant (ou après) rendu :

- En tapant dans un terminal la commande `ifconfig`, vous devriez obtenir l'adresse IP de votre ordinateur (serveur). Ouvrez un navigateur sur un autre PC de la salle et rendez-vous à l'adresse <http://IPDUSERVEUR:8080>
Vous pourrez ainsi jouer à plusieurs en réseau ou tout simplement regarder une partie se dérouler à l'autre bout de la salle. **Pensez à rafraîchir la page de temps en temps...**
- Comme vous le savez, un utilisateur peut parfois être malicieux lorsqu'il s'agit de remplir un champ de texte. Testez en profondeur le fonctionnement de votre webapp et assurez-vous qu'une erreur de saisie n'entraîne pas de catastrophe algorithmique.
- Vous pouvez remplacer les symboles de vos pièces par des liens d'images placées à la racine du projet. Vous devrez alors adapter les `<td></td>` de votre échiquier et votre script client (exemple donné en commentaire). Qui a parlé de Pokémon ?