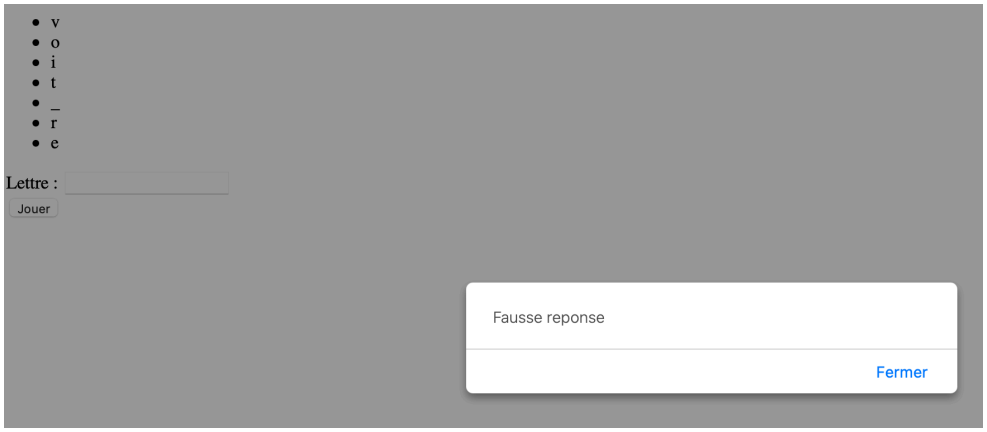
TP3 Architecture multi-couches

Dans ce TP, nous allons faire une version simplifiée du jeu de pendu sous la forme d'une application Web. Nous aurons, d'un côté un moteur de jeu qui proposera un mot à faire deviner et de l'autre côté depuis son navigateur web, un utilisateur pourra proposer des lettres. Les mots à deviner seront obtenus à partir d'un dictionnaire. Nous imposerons une contrainte : les communications entre navigateur et moteur de jeux se feront via des Websockets.



Question 1 : Quelle type d'architecture devons nos mettre en place ?

Question 2 : combien de couches aurez-vous ? Lesquelles ?

Question 3 : Dessinez cette architecture. Vous pouvez la modifier au fur et à mesure que vous réaliserez le TP.

Préparation

Téléchargez l'archive de ressources que vous trouverez sur Moodle. Cette archive contient le squelette de votre application :

- Un fichier **db.json**. Ce fichier contient la liste des mots pour le jeu de pendu. Il fera office de base de données.
- **package.json** : Un fichier **package.json**. Ce fichier est le script de construction nous permettant de lancer un petit serveur web et d'exposer db.json comme s'il s'agissait d'une base de données.
- **serveur.js** : Notre serveur web socket qui écoute les messages des clients, et gère le mot à trouver.
- **index.html** : La page HTML pour le client.
- **client.js** : Le code JavaScript côté client pour recevoir un mot du serveur et envoyer des propositions de lettre pendant le jeu.

Lancer votre dictionnaire de mots !

- Installer npm et nodejs si ce n'est pas déjà fait.
- Lancez l'installation de votre projet avec la commande : `npm install (depuis le répertoire contenant le fichier package.json)`
- Démarrez la base de données en lançant la commande suivante depuis la racine de votre projet (le répertoire contenant le fichier package.json) : `npm run jsonserver`
  Vous devez obtenir l'affichage suivant :

```
> @ jsonserver /Users/ferrynico/Google Drive/IUT/Cours/ArchiLogicielles/TPs/TP3/pendu
> json-server -p 3001 --watch db.json


  \{^_^}/ hi!

  Loading db.json
  Done

  Resources
  http://localhost:3001/mots

  Home
  http://localhost:3001

  Type s + enter at any time to create a snapshot of the database
  Watching...
```

- Vous pouvez consulter la liste des mots disponible à l'adresse suivante : http://127.0.0.1:3001/mots

Lancer votre moteur de jeux

- Dans un autre terminal, démarrez votre serveur avec la commande suivante, lancée depuis la racine de votre projet (le répertoire contenant le fichier package.json): `npm start`
- Côté serveur, nous utiliserons le module '`ws`' pour gérer les Websockets. Regardez le code du fichier `serveur.js`.

Client-serveur

1. Parcourez la documentation du module `'ws'`. Comment fait-on pour envoyer un message ? (https://www.npmjs.com/package/ws)
2. Dans le fichier `serveur.js` :
   a. Définissez une constante qui contiendra le mot à trouver. Faites une requête sur http://127.0.0.1:3001/mots (notre base de données) pour initialiser votre mot à trouver. Vous pouvez vous inspirer du code suivant pour réaliser votre requête HTTP.

Avec le module node-fetch

Il faut ajouter la dépendance vers le module (dans le fichier package.json) :

```
{
```

```
  "scripts": {
    "start": "node serveur.js",
    "jsonserver": "json-server -p 3001 --watch db.json"
  },
  "dependencies": {
    "ws": "^7.0.1",
    "json-server": "^0.14.2",
    "node-fetch": "2.6.1"
  }
}
```

Le code :

```
fetch('http://localhost:3001/mots) // une requete GET
    .then((response) => { // lorsque la requête est terminée on traite la réponse
        response.json() // que l'on passe en JSON
        .then((mots) => { // On fois fait on peut travailler sur la liste des mots
            for(mot of mots){…}
            }
        });
    }).catch(error => console.error);
```

Avec le module http

```
http.get('http://nodejs.org/dist/index.json', (res) => {
  const { statusCode } = res;
  const contentType = res.headers['content-type'];

  let error;
  // Any 2xx status code signals a successful response but
  // here we're only checking for 200.
  if (statusCode !== 200) {
    error = new Error('Request Failed.\n' +
                      `Status Code: ${statusCode}`);
  } else if (!/^application\/json/.test(contentType)) {
    error = new Error('Invalid content-type.\n' +
                      `Expected application/json but received
${contentType}`);
  }
  if (error) {
    console.error(error.message);
```

```
    // Consume response data to free up memory
    res.resume();
    return;
  }

  res.setEncoding('utf8');
  let rawData = '';
  res.on('data', (chunk) => { rawData += chunk; });
  res.on('end', () => {
    try {
      const parsedData = JSON.parse(rawData);
      console.log(parsedData);
    } catch (e) {
      console.error(e.message);
    }
  });
}).on('error', (e) => {
  console.error(`Got error: ${e.message}`);
});
```

   b. Faites-en sorte que lors de la connexion d'un client, le serveur lui retourne un message avec la taille du mot à trouver.

3. Dans le fichier `client.js` :
   a. Ajoutez le code pour faire la connexion avec le serveur (inspirez-vous de l'exemple donné dans le cours).
   b. Essayez votre code en ouvrant le fichier `index.html` dans votre navigateur. Vous devez voir un message de log côté server.
   c. Écoutez les messages du serveur. Sur la réception du premier message du serveur contenant la taille du mot à trouver, ajoutez à votre élément `ul` autant de fils (`li`) qu'il y a de lettres dans le mot. Lorsqu'une lettre n'est pas encore trouvée elle est matérialisée par le caractère `'_'`. Si vous n'êtes pas familier avec le DOM, vous pouvez ajouter un li avec le code suivant :

```
let li = document.createElement('li');

li.innerText = "_";

document.querySelector('#mot').append(li);
```

4. Dans le fichier `serveur.js` : Sur la réception d'un message il faut : vérifier si la lettre est dans le mot (on pourra utiliser simplement la fonction `indexOf`) et, si c'est le cas, retourner au client la lettre et sa position dans le mot. Sinon, on retourne au client un message lui indiquant qu'il a fait un mauvais choix.
5. Dans le fichier `client.js`, lorsque nous recevons un message indiquant que la lettre est dans le mot, nous la positionnons dans notre liste (`ul`). Sinon, nous affichons une alerte indiquant que la lettre n'est pas dans le mot.

Pour finir

Mettez à jour le schéma de votre architecture si ce n'est pas encore fait.