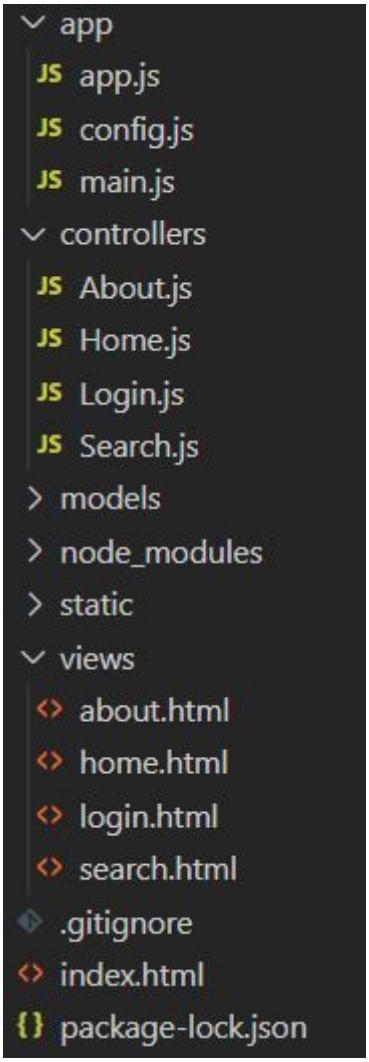
PARIS EVENTS

Enoncé

Nous allons créer une application web permettant de rechercher les événements sur Paris (concerts, spectacles, expositions, etc).
Nous utiliserons pour cela une API (https://opendata.paris.fr/) qui nous retournera les différents événements via un moteur de recherche.
Nous pousserons aussi, afin de créer une interface de connexion en utilisant Firebase.

Nous passerons via des contrôleurs qui seront des classes (exportées par défaut) et importées dans **app.js**

Voici à quoi ressemblera notre structure :



Etape 1) Mise en place du projet de base.

Télécharger le fichier zip sur le moodle, extraire dans le répertoire de travail.

Etape 2) Installation de Vanilla router

A partir de la documentation de vanilla router tentez l'installation du package
https://www.npmjs.com/package/vanilla-router

```
npm install vanilla-router --save
```

Dans certain cas (exemple sous mac) il sera nécessaire de faire un npm init avant l'installation du package :

```
npm init
```

Dans le fichier **index.html** ajoutez le script de vanilla-router :

```
47.<script src="node_modules/vanilla-router/dist/vanilla-router.min.js"></script>
```

Vous pouvez vérifier que le script est correctement chargé via le code source de votre page html (sur votre navigateur).

Etape 3) Test du routeur

Dans un premier temps nous allons tester le router
Créez des routes qui afficheront simplement un console.log contenant le nom de la route courante par exemple.

Etape 4) Organisation du code et mise en place des vraies routes

Dans le fichier app.js ajouter une propriété **router** dans **app.mvc** (qui doit avoir la valeur **null** par défaut) et sera (par la suite) une instance du routeur (vanilla-router en l'occurrence), cette instance sera créée dans **main.js**.
Créer les différentes routes nécessaire à notre application ("/", "/search","/about","/login")
Ces routes sont utilisées dans les liens href du fichier index.html.
Dans un premier temps mettez des console.log pour les différentes routes.

Etape 5) Création du controller Home

Nous allons créer un premier controller (Home) qui sera appelé via la route "/".
Nos controllers seront des classes exportées par défaut en ES6.

Ces classes auront une méthode **constructor** qui aura pour objectif de déclarer un attribut **view** .

Les controllers pourront avoir (mais ce n'est pas obligatoire dans tous les controllers) une méthode executeHttpRequest, cette méthode devra être appelée (si elle existe) après le chargement de la vue (elle pourra ainsi manipuler le DOM).

Vous devez créer la fonction qui va gérer ce "dispatching" que nous placerons dans **app.mvc** et que nous appellerons **dispatchRoute**, cette méthode attend un controller en paramètre et a pour rôle de vérifier si l'attribut view existe (si il n'existe pas il faudra lever une erreur, si il existe il faudra charger la vue).
https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

https://developer.mozilla.org/fr/docs/Web/API/Body/text
Exemple pour manipuler le résultat d'une requête HTTP sur un contenu html :

```
80.fetch(PATH_DE_LA_VUE)
81.     .then(response => response.text())
82.     .then(html => {
83.         // ***
84.      });
```

Il suffira une fois le contenu de la vue chargée d'écrire son contenu dans la balise "main.container" (qui existe déjà dans le fichier index.html).
https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector
https://developer.mozilla.org/fr/docs/Web/API/Element/innertHTML

Une fois la vue chargée on vérifiera si la méthode executeHttpRequest existe dans le controller et l'exécuterons.

Voici une exemple d'appel du dispatcher dans le initializeRouter()

```
85. app.mvc.router.add('', () => app.mvc.dispatchRoute(new Home()));
```

Dans la vue il doit y avoir uniquement la partie concernant la page, exemple pour la page d'accueil le fichier views/home.html pourrait être :

```
1. <h2>Bienvenue sur Paris Events</h2>
2.
3. <p>L'application qui permet de rechercher des événements Parisiens et de
   les ajouter en favoris.</p>
4.
5. <p>Pour commencer une recherche, utilisez le menu ci-dessus ou <a
   href="/#/search">cliquez ici</a>.</p>
```

Etape 6) Création des controllers et des vues About, Search et Login

Dans un premier temps nous n'allons faire aucun code dans executeHttpRequest, nous allons uniquement créer les controllers et les vues pour les différents liens.

views/about.html

```
1. <h2>À propos</h2>
2.
3. <p>Cette application à pour vocation de pratiquer le code JavaScript
   Vanilla (sans framework) et de faire une application multi-pages en
   intoduisant les concepts suivants :</p>
4.
5. <ul>
6.     <li>MVC</li>
7.     <li>Routeur</li>
8.     <li>Promesses et asynchronicité</li>
9.     <li>Authentification avec Firebase</li>
10. </ul>
```

views/search.html

```
<h2>Évènements à Paris</h2>
<form id="formSearch" class="row">
    <div class="form-group col-md-12 col-sm-12 col-lg-3">
        <label for="q">Terme de recherche :</label>
        <input type="search" class="form-control" name="q" id="q"
placeholder="Titre d'un événement ...">
    </div>
    <div class="form-group col-md-6 col-sm-6 col-lg-3">
        <label for="dateStart">Année des événements :</label>
        <select name="dateStart" id="dateStart" class="form-control">
            <option value="2022">2022</option>
            <option value="2021">2021</option>
            <option value="2020">2020</option>
            <option value="2019">2019</option>
            <option value="2018">2018</option>
        </select>
    </div>
    <div class="form-group col-md-6 col-sm-6 col-lg-3">
        <label for="sortBy">Trier par :</label>
        <select name="sortBy" id="sortBy" class="form-control">
            <option value="date_start">Date (croissant)</option>
            <option value="-date_start">Date (décroissant)</option>
            <option value="title">Titre</option>
        </select>
    </div>
    <div class="col-md-12">
        <button type="submit" class="btn btn-primary">Rechercher
!</button>
    </div>
</form>
<hr>
<template id="event-template">
    <div class="col">
        <h6 class="event-title"></h6>
        <img class="event-image" src="">
    </div>
</template>
<div class="event-list row">
    <!-- Liste de résultats -->
</div>
```

Commencez par about, puis login (ou vous devez uniquement faire un bouton) puis finissez par la vue search ou vous devrez mettre en place un formulaire comme celui ci.
Attention nous ne souhaitons pas avoir un formulaire avec une soumission des données, donc pas de bouton submit , ou ajoutez une preventDefault sur l'événement submit. Les données seront récupérées et manipuler uniquement en JS (et non en HTTP), voir étape suivante :

https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector

Etape 7) Prise en compte des données du formulaire

Ici nous allons devoir récupérer les informations du formulaire via la méthode executeHttpRequest dans le controller Search.
Vous devez faire évoluer le dispatchRoute pour permettre l'executeHttpRequest si il est dans le controller

La méthode executeHttpRequest de notre controller Search devra contenir un écouteur d'événement sur le clique du bouton rechercher, quand le bouton est cliqué récupérer les différentess valeur des champs.

Etape 8) Test de l'api opendata paris

API OpenData paris
https://opendata.paris.fr/explore/dataset/que-faire-a-paris-/information/?disjunctive.category&disjunctive.tags&disjunctive.address_zipcode&disjunctive.address_city&disjunctive.access_type&disjunctive.price_type

Vous pouvez tester l'api ici :
https://opendata.paris.fr/explore/dataset/que-faire-a-paris-/api/?disjunctive.category&disjunctive.tags&disjunctive.address_zipcode&disjunctive.address_city&disjunctive.access_type&disjunctive.price_type

En testant l'api il est simple de récupérer l'url de base:

Nous allons devoir ici créer un model
Créez le fichier **models/ParisEvents.js**

```
export default class ParisEvents {

    constructor() {
        this.urlBase = 'https://opendata.paris.fr/api/records/1.0/search/?dataset=que-faire-a-paris-';
    }
}
```

Nous devons maintenant étudier l'API pour comprendre comment communiquer avec elle, pour rappel dans notre cas les éléments suivant nous intéresse : "Terme de recherche", "Année des événements", "Trier par"

L'api va nous retourner du json :
https://developer.mozilla.org/en-US/docs/Web/API/Body/json

Nous pourrons récupérer uniquement les valeurs nécessaires via :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/map

Etape 9) Affichage des résultats de recherche

Dans le but de construire l'affichage des événements récupéré via l'api, nous allons utiliser la balise html template que nous allons pouvoir cloner pour créer nos différents événements.
https://developer.mozilla.org/fr/docs/Web/HTML/Element/template

N'oubliez pas que nous avons dans **app.js** un attribut **app.dom** permettant d'ajouter des fonctions pour manipuler le dom, il serait judicieux de construire une fonction permettant de manipuler les templates.

Etape Bonus 1) Uniquement si vous êtes en avance

BONUS PARIS EVENTS

Etape 10) Ajout du système de connexion via Firebase

Dans l'interface de connexion nous allons créer des boutons de connexion. Par exemple :

Quand on clique sur un des boutons firebase utilise le provider en question (google, github ou facebook)

Firebase :
Firebase + authentification avec Google

Connexion GitHub avec Firebase :
Firebase authentification avec GitHub

Connexion Facebook avec Firebase :
Firebase authentification avec Facebook

Au besoin voici la documentation officielle :
https://firebase.google.com/docs/auth/web/start

Etape Bonus 2) Uniquement si vous êtes en avance

Ajouter un système de pagination pour l'affichage des events.
L'api open-data Paris permet de connaître le nombre d'enregistrement concernant une recherche "nhits", elle permet également de récupérer un nombre d'éléments précis "rows" et elle permet de commencer à récupérer à partir d'un certain nombre d'élément "start".

Il est donc possible par exemple de récupérer :

12 éléments pour la page 1 en utilisant : **&rows=12&start=0**
12 éléments pour la page 2 en utilisant : **&rows=12&start=12**
12 éléments pour la page 3 en utilisant : **&rows=12&start=24**
etc.

Il est également possible de connaître le nombre de page (en supposant que l'on souhaite afficher 12 éléments par page) :
Math.ceil(nb_hits/12);

nhits étant un attribut récupérable dans le JSON récupéré.
Math.ceil est une méthode javascript permettant de récupérer l'entier supérieur le plus proche.