Examen de Langages de Scripts Dynamiques orientés Web

Août 2023

Consignes

Fichiers et répertoires

- Récupérez l'archive dans le cours **examenJsAout2023.zip** qui contient :

  `/dictio-q213456/css/styles.css` :
  la feuille de style (ne pas modifier),

  `/dictio-q213456/index.html` :
  l'unique page HTML de l'application, intégrez l'appel à votre script JS,

  `/dictio-q213456/libs/dictio.mjs :`
  les classes et fonctions que vous devez programmer

  `/dictio-q213456/libs/app.mjs :`
  l'application que vous devez programmer

  `/dictio-q213456/tests/student.spec.mjs` :
  fichier à compléter avec 2 tests unitaires de la fonction **cleanString()**

  `/dictio-q213456/tests/dictio-marvi.spec.min.mjs` :
  code source obscurci des tests unitaires de validation de votre solution (ne pas modifier)

  `/dictio-q213456/dictio-test.html` :
  page d'exécution des tests unitaires de validation de votre solution (ne pas modifier)

Veuillez renommez le répertoire **/dictio-q213456** par **/dictio-votre matricule** !!

Communications sur Teams

- Rejoignez l'équipe Teams « IN-B2-UE20-JS » avec le code : **1jcij9w**
- Le canal *Examen 31 août* pourra servir si des informations importantes devaient vous être communiquées durant l'examen,
- Si vous avez des questions de compréhension de l'énoncé ou rencontrez un problème particulier, contactez Monsieur Schumacker ou moi-même par message privé !

Dépôts sur GitLab

- Créez un dépôt git intitulé « js-examen-aout-2023-matricule » (Exemple : js-examen-aout-2023-q123456).
- N'intégrez par les répertoires **/.idea** et **/node_modules** dans le dépôt (fichier **.gitignore**)
- Après 1h, toutes les **30min**, il vous sera demandé d'effectuer **un commit** et **un push sur le serveur**.
- Le non-respect des consignes ci-dessus entrainera une pénalité de 2 points par manquement constaté.
- L'examen se termine à 16h30, le dernier commit sur le serveur devra être effectué à **16h35** au plus tard.
- Si vous bénéficiez d'1/3 temps supplémentaire grâce à un aménagement inclusion, votre dernier commit devra être effectué à **17h35** au plus tard.
- Durant l'examen, vous êtes libres de consulter toutes les **ressources** à votre disposition (tutoriels, résolutions d'exercices, …). C'est une épreuve **individuelle** et il est strictement défendu d'aider ou de bénéficier de l'aide d'un tiers. Tout manquement constaté sera **sanctionné** d'une cote nulle pour l'épreuve et la constitution d'un **dossier disciplinaire**.

Recevabilité

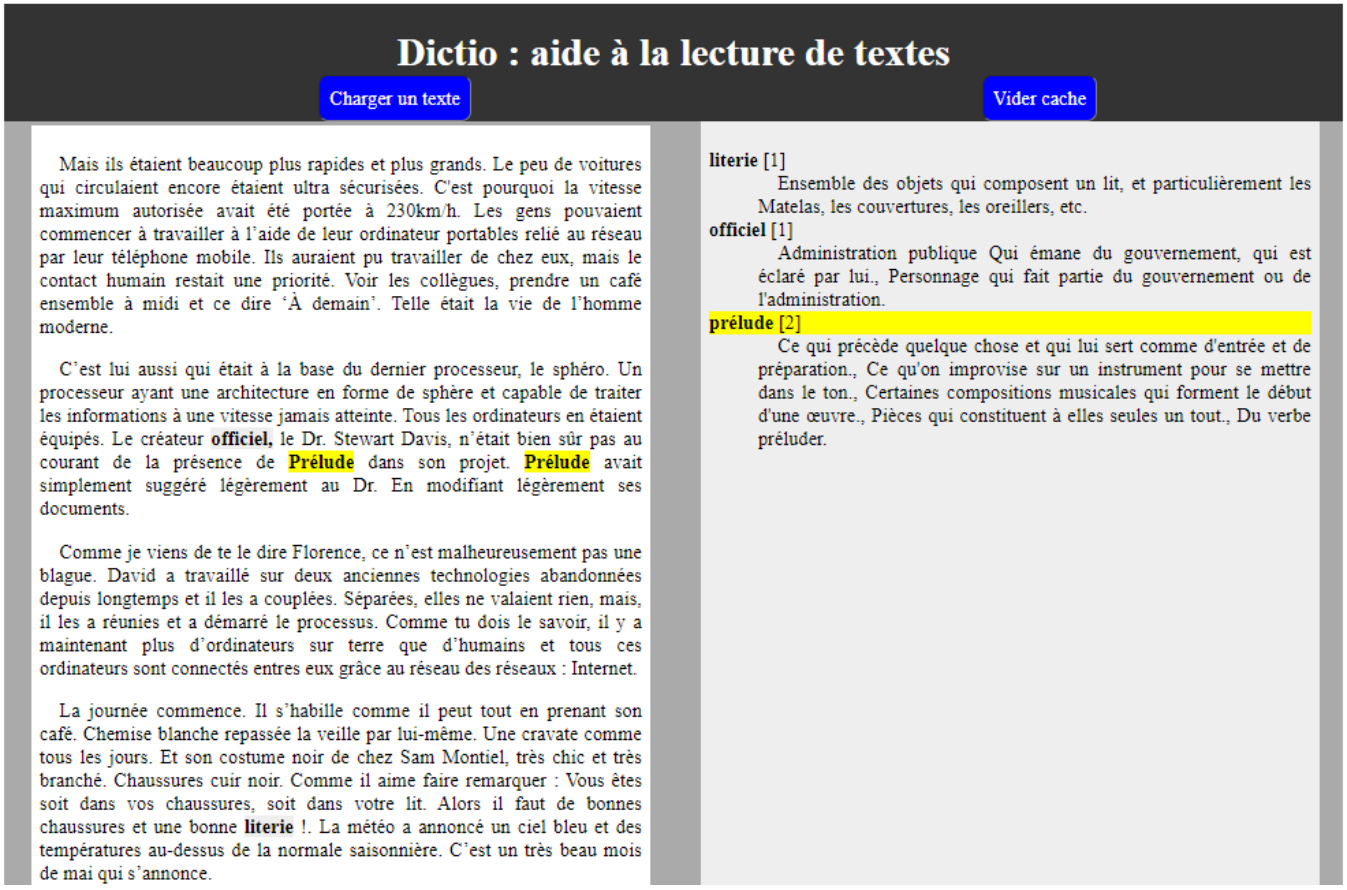
- Durant cette épreuve, vous devez **obligatoirement** démontrer votre capacité à effectuer des appels AJAX, à manipuler des fonctions asynchrones et déclarer une classe en Javascript. Certains éléments de l'énoncé sont marqués par **(*)** ; ils doivent être implémentés correctement pour que votre copie soit jugée recevable. Ces éléments sont :

  - la déclaration des classes (Entry, Dictionnary), de leurs propriétés et de leur constructeur (les autres méthodes n'impactent pas la recevabilité)

  - la déclaration de la fonction (DictioConnector.getText(lang)) qui effectue des appels AJAX et retourne une promesse

Vue générale de l'application

Vous allez développer un petit prototype d'application d'aide à la lecture de documents. L'utilisateur peut charger un texte à lire. S'il clique sur un mot du texte, toutes les occurrences de ce mot sont mises en surbrillance et la définition est affichée sur l'écran de droite.



L'application repose sur plusieurs dictionnaires (couples mot-définition) :

- Un dictionnaire consultable par appels AJAX – [classe **DictioConnector**]
- Un dictionnaire conservé localement en permanence avec tous les mots déjà recherchés par l'utilisateur. Ce dictionnaire sert uniquement à accélérer la consultation ultérieure de mots déjà recherchés. Son contenu n'est pas affiché – [classe **DictioRepository**]
- Un dictionnaire conservé localement temporairement avec les mots recherchés dans le texte courant. Le contenu de ce dictionnaire est affiché dans la partie droite de l'écran de l'application. Lorsqu'un nouveau texte est chargé, ce dictionnaire est réinitialisé – [classe **DictioTemporary**]

Exercice 1 : Déclaration de classes du domaine (45 min)

Les classes du domaine peuvent être testées via /dictio-q213456/dictio-test.html.

Dans **/libs/dictio.mjs,** complètez et exposez la classe **Entry** :

(*) Propriétés :

- **word** (chaine, mot du dictionnaire, chaine vide par défaut),
- **definition** (chaine, définition du mot, chaine vide par défaut)

Méthode :

- **(*)** le constructeur utilise **l'affectation par décomposition**
  Exemple :
  ```
  let entry = new Entry({word : "clavier", definition : " Ensemble des touches d'une machine à écrire sur lesquelles on appuie avec les doigts pour écrire." });
  ```

Dans **/libs/dictio.mjs,** complétez et exposez la classe **DictioTemporary** :

(*) Propriété :

- **entries** (objet dont les propriétés sont les mots et les valeurs les définitions),

Méthodes :

- **(*)** le **constructeur**
  Exemple : `let dictionary = new DictioTemporary({clavier : " Ensemble des touches d'une machine à écrire sur lesquelles on appuie avec les doigts pour écrire.", crayon: "Morceau de bois cylindrique fin contenant une mine en son centre, taillé afin d'obtenir une pointe à l'extrémité et qui permet de dessiner, colorier ou écrire avec la main"});`

Lors de l'appel au constructeur, les entrées du dictionnaire sont enregistrées dans la mémoire locale temporaire. Un seul dictionnaire temporaire est conservé à la fois.

- **addEntry**(entry) qui reçoit une instance d'**Entry**, ajoute ou met à jour le mot et sa définition
- **getWords**() qui retourne la liste des **mots** du dictionnaire temporaire, **triés alphabétiquement**, sous forme d'un tableau de chaines.
  Exemple : `dictionary.getWords() retourne ["clavier", "crayon"]`

- **clear**() supprime toutes les entrées du dictionnaire

- **static load**() : méthode de fabrique qui instancie un dictionnaire temporaire sur base du dernier dictionnaire temporaire local enregistré.

Dans **/libs/dictio.mjs,** complétez et exposez la fonction **cleanString**(text) :

Cette fonction reçoit une chaine de caractère dont elle **supprime** toutes les **occurrences des caractères** point (.), point-virgule (;), virgule (,), point d'exclamation (!), point d'interrogation (?), guillemet double ("), deux-points (:). Les guillemets simples sont conservés. La chaine retournée est également écrite en **minuscules**.

Exercice 2 : Premier appel AJAX (15 min)

Dans **/libs/dictio.mjs,** complétez et exposez la classe **DictioConnector**() qui ne contient que des méthodes **statiques**.

- **(*)** **static** **getText**(lang) qui retourne une **promesse** ; cette fonction effectue une requête **GET** vers http://192.168.128.13/~p150107/js2/dictio/dictio.php avec le paramètre **lang** qui contient la langue désirée ("fr" par défaut). La réponse HTTP contient un objet **JSON** avec le contenu suivant :
  `{"data" : "texte à lire", "message" : "message d'erreur éventuel"}`
  En cas de succès, la valeur de la promesse est le texte contenu dans la propriété "data".
  En cas d'échec, la valeur de la promesse est une exception dont le message est le message d'erreur.

Exercice 3 : Tests unitaires (15 min)

- Dans **/tests/student.spec.mjs,** validez la fonction **cleanString(text)** avec **3 tests unitaires :** une chaine vide, une chaine non vide sans caractère spécial, une chaine avec caractères spéciaux et majuscules.

Exercice 4 : Stockage local permanent (20 min)

Dans **/libs/dictio.mjs,** complétez et exposez la classe **DictioRepository** qui ne contient que des méthodes **statiques**.

- **static** **loadEntries**() : retourne les entrées du dictionnaire local stocké en permanence sous forme d'un objet dont les propriétés sont les mots et les valeurs les définitions. (*Attention : cette fonction ne retourne pas une collection d'instances Entry !!!*)
- **static** **saveEntry**(entry) : reçoit une instance d'**Entry** à ajouter ou mettre à jour dans le dictionnaire permanent.
- **static** **clearEntries**() : supprime toutes les entrées du dictionnaire permanent.
- **static** **getDefinition**(word) : reçoit un mot à chercher dans le dictionnaire et retourne une **promesse**.
  En cas de succès, la valeur de la promesse est une chaine de caractère : la définition correspondante au mot cherché.
  En cas d'échec, par exemple si le mot n'est pas trouvé, la valeur de la promesse est **undefined**.

Exercice 5 : Deuxième appel AJAX (30 min)

Dans **/libs/dictio.mjs,** complétez la classe **DictioConnector**()

- **(*)** **static** **getDefinition**(word) qui retourne une **promesse** ;
  cette fonction reçoit un mot et exploite la version nettoyée de celui-ci (appel à **cleanString**). La fonction recherche la définition liée au mot en parallèle dans la mémoire locale permanente et via une requête **POST** vers http://192.168.128.13/~p150107/js2/dictio/dictio.php avec le paramètre **word** qui contient le mot nettoyé recherché. La réponse HTTP contient un objet **JSON** avec le contenu suivant :
  `{"data" : "définition si trouvée", "message" : "message d'erreur éventuel"}`
  En cas de succès, la valeur de la promesse est la définition contenue dans la propriété "data".
  En cas d'échec, ou si le mot n'a pas été trouvé (data est null), la valeur de la promesse est une exception dont le message est le message d'erreur ou « Mot non trouvé ! ».
  La promesse se termine en succès dès qu'elle reçoit une définition pour le mot. Elle échoue si la définition n'a pu être trouvée ni en local ni via le webservice.

Si vous avez assez de temps, cette fonction sera à améliorer pour trouver la définition du mot au singulier si celui-ci est au pluriel (voir Exercice 7).

Exercice 6 : Application (45 min)

Dans **/libs/app.mjs,** créez les fonctions suivantes :

La fonction **renderText**(text) qui reçoit un texte en argument :

- place chaque élément du texte dans une balise . Un élément du texte est une séquence de caractères délimitée par une espace.

  Exemple : *"Aujourd'hui, il fait beau." donnerait "Aujourd'hui, il fait beau."*
  Notez bien que deux balises **successives** sont séparées par un **caractère espace**. *("il fait" et non "ilfait")*

- place ensuite chaque ligne délimitée par un caractère saut de ligne (\n) dans une balise , et enfin

  Exemple : *"Coucou\nCharles !" devient " Coucou Charles ! "*

- place le tout dans <section id="reader"> de la page.

La fonction **renderConsole**(text) qui reçoit un texte en argument et le place dans `<label id="console">`

La fonction **getFrequency**(word) qui reçoit un mot et retourne le nombre de balises `<span>` qui contiennent la version nettoyée de ce mot dans la balise `<section id="reader">`.

> Rappel: les mots du dictionnaire sont nettoyés et en mininuscules; ce n'est pas le cas du contenu d'une balise (exemple: "charles" versus "Charles !").

La fonction **renderDictionary**(dictio) qui reçoit une instance de **DictioTemporary** et pour chaque mot par ordre alphabétique, place celui-ci dans une balise `<dt>mot <span class="frequency">n</span></dt>` et sa définition dans une balise `<dd>definition…</dd>`. Le tout est placé dans la balise `<dl id="words">` de la page.

```
Exemple:
<dl id="words">
    <dt>literie<span class="frequency">1</span></dt>
    <dd>Ensemble des objets qui composent un lit, et particulièrement les Matelas, les couvertures, les oreillers, etc.</dd>
    <dt>officiel<span class="frequency">1</span></dt>
    <dd>Administration publique Qui émane du gouvernement, qui est éclaré par lui., Personnage qui fait partie du gouvernement ou de l'administration.</dd>
    <dt>prélude<span class="frequency">2</span></dt>
    <dd>Ce qui précède quelque chose et qui lui sert comme d'entrée et de préparation., Ce qu'on improvise sur un instrument pour se mettre dans le ton., Certaines compositions musicales qui forment le début d'une œuvre., Pièces qui constituent à elles seules un tout., Du verbe préluder.</dd>
</dl>
```

Par ailleurs, dans le texte affiché, chaque `<span>` correspondant à un mot du dictionnaire est marqué en lui ajoutant la classe CSS « **tag** » qui entraînera l'affichage du mot en gras.

L'application proprement dite :

1) Au démarrage de l'application, le dictionnaire temporaire est chargé (DictioTemporary.**load**()) et affiché (**renderDictionary**()).

2) Un clic sur Vider cache supprime les entrées du dictionnaire permanent (DictioRepository.clearEntries()) et affiche le message « Dictionnaire permanent supprimé. » dans <label id="console">. (**renderConsole**())
3) Un clic sur `<a id="loadText" href="#">Charger un texte</a>`, affiche le message *« Lecture d'un texte… »* dans `<label id="console">`, lit un texte via le webservice (DictioConnector.**getText**(lang)), effectue son rendu (**renderText**(text)) et affiche le message « Texte chargé. » dans `<label id="console">` (**renderConsole**()). Le dictionnaire temporaire est vidé (*myDictioTemporary*.**clear()**) et l'affichage est mis à jour (**renderDictionary**(dictio)).

4) Complétez la fonction renderText() de telle sorte qu'un clic sur un élément du texte (un `<span>`) provoque la recherche de sa définition (DictioConnector.**getDefinition**(word)), sa sauvegarde dans les dictionnaires temporaire (*DictioTemporary***addEntry**(entry)) et permanent (DictioRepository.**saveEntry**(entry)), et la mise à jour du rendu du dictionnaire temporaire (**renderDictionary**(dictio)) en ce compris la mise à jour des classes CSS « **tag** ».

5) Complétez le traitement d'un clic sur un élément du texte de telle sorte que la balise `<dt>` dans le rendu du dictionnaire et les `<span>` correspondant au `<span>` cliqué soient marqués comme actifs en ajoutant la classe CSS « **active** » qui entraînera la mise en surbrillance jaune des éléments correspondants. Cette classe CSS est retirée des autres balises si nécessaire.

Exercice 7 : Gestion des pluriels (10 min)

Dans **/libs/dictio.mjs,** améliorez la méthode **getDefinition()** de la classe **DictioConnector** :

Si le mot nettoyé cherché termine par la lettre « s », la définition du mot potentiel au singulier est recherchée. Si aucune définition n'est trouvée, la recherche s'effectue seulement ensuite sur le mot nettoyé avec « s » final.

Les autres formes de pluriel sont ignorées dans notre prototype d'application.

Dans **/libs/app.mjs,** améliorez la fonction **getFrequency**(word) pour compter la somme des occurrences du mot nettoyé au pluriel et au singulier.

Bon travail 😊