

# **582-21F-MA**

# **Programmation**

# **d'interface Web 1**

## **TP 2**

# Description

Travail pratique permettant l'intégration des éléments d'apprentissage.

# Pondération

25 %

# Modalité particulière

Travail individuel

# Sujet

Gestionnaire complet d'une liste de tâches (*to-do-list*)

# Date de remise

Cours 20 - vendredi 8 septembre 2023

# Modalités de remise

- Le dossier zippé du projet - à votre nom (nomdefamille-prenom) - doit être remis sur Léa avant le 8 septembre 23h59.
- Le site doit être en ligne avant le 8 septembre 23h59.
- Dans le dossier de remise, vous devez inclure un fichier **.txt** avec l'URL de votre site.

# Retard

Selon les règles du collège, 5 % par jour de retard seront enlevés, jusqu'à 5 jours de retard maximum.

# Énoncé

La commande consiste à créer une page Web (une seule page *HTML*) qui permet à un usager de faire la gestion d'une liste de tâches à accomplir (*to-do-list*).

Comme pour le TP1, ce n'est pas un travail de rendu de maquette. Vous aurez évidemment quelques définitions CSS à écrire pour réaliser les comportements demandés, mais la mise en page est minimale et le *responsive* n'est pas évalué. Toutefois, assurez-vous de soigner votre rigueur, la structure et vos nomenclatures CSS.

Vous devez faire ce projet en programmation orientée objet. Notez que les fonctions fléchées et les librairies externes (par exemple *jQuery*) sont interdites. Vous êtes responsable de l'organisation de vos scripts. Je vous invite à suivre les consignes, celles-ci donnent plusieurs stratégies de développement.

Consultez les références données pour les contenus textuels des titres et intitulés.

## Consignes

### Créer le *DOM* statique

- La première image donnée en référence illustre la structure *HTML* statique du projet. Après l'habituel `<header>`, il y a trois `<section>`, soit :
  - un formulaire d'ajout de tâche ;
  - la liste des tâches ajoutées ;
  - le détail d'une tâche.

# Consignes (suite)

## Ajouter une tâche

- Le formulaire pour ajouter une tâche a :
  - un **input** de type texte (obligatoire) pour le nom de la tâche ;
  - un **input** de type texte pour la description de la tâche ;
  - trois **input** de type radio pour donner une importance (obligatoire) à la tâche (Haute, Moyenne ou Basse).
- Au chargement de la page, instanciez la classe *ES6* nommée (par exemple) **Formulaire** - en prenant soin de passer son élément *DOM* en argument - afin de faire la gestion du formulaire.
- Au clic du bouton, faites d'abord la validation des deux contrôles obligatoires (Nom et Importance).
- Si le formulaire est valide, gardez en mémoire les valeurs saisies. Une bonne stratégie est de les placer à l'intérieur d'un objet littéral, que vous ajoutez ensuite comme dernier index d'un tableau. Exemple :

```
let tache = {  
    tache:           // valeur du champ 'Nouvelle tâche'  
    description:     // valeur du champ 'Description'  
    importance:      // valeur du champ radio coché  
}  
aTaches.push(tache);
```

- Pour avoir un accès global dans tout le projet au tableau **aTaches**, celui-ci est déclaré comme seule instruction du premier fichier *JS* chargé dans le **<head>**. Exemple :

```
const aTaches = [];
```

## Consignes (suite)

- Toujours si le formulaire est valide, assurez-vous de nettoyer tous ses champs afin que l'utilisateur puisse saisir une nouvelle tâche sans avoir à effacer la précédente.
- Ensuite, injectez dynamiquement la nouvelle tâche, son importance et ses deux boutons dans la section 'Liste des tâches'. Aidez-vous en injectant son index dans un attribut **data**.
- La beauté de *ES6* est que vous pouvez faire l'instanciation de nouveaux objets à n'importe quel moment. Ainsi, après avoir injecté la tâche, vous pourrez récupérer son nœud *DOM* puis instancier une classe *ES6* nommée (par exemple) **Tache**, ceci toujours en prenant soin de passer son élément *DOM* en argument. Cette classe prendra en charge les comportements et fonctionnalités associés à chaque tâche.

### Gérer chaque tâche

- La classe **Tache** instanciée pour chaque tâche a essentiellement deux fonctionnalités :
  - afficher le détail de la tâche ;
  - supprimer la tâche.
- Au clic du bouton 'Afficher le détail' d'une tâche, vous devez injecter dynamiquement ses informations (tâche, description et importance) dans la section 'Détail d'une tâche'.
- À cette étape, rappelez-vous que le champ description n'était pas obligatoire, si cette valeur est une chaîne vide, injectez le texte : 'Aucune description disponible.'

## Consignes (suite)

- Au clic du bouton 'Supprimer' d'une tâche, vous devrez la supprimer de la liste. Pour assurer la séquence d'affichage des tâches avec le tableau d'objets littéraux **aTaches**, il faudra supprimer la tâche du tableau **aTaches** puis réinjecter dans le *DOM* toutes les tâches toujours présentes. N'oubliez pas que c'est du nouveau *DOM*, il faudra alors ré-instancier la classe **Tache** pour chaque tâche réinjectée.

### Filtrer les tâches

- La section 'Liste des tâches' a deux boutons pour les filtrer :
  - par ordre alphabétique ;
  - par ordre d'importance.
- Pour cette fonctionnalité, je vous suggère de créer une nouvelle classe nommée (par exemple) **TrierTaches**, celle-ci instanciée au chargement de la page. Ici encore, prenez soin de passer son élément *DOM* en argument.
- Nous avons vu ensemble la méthode **sort()** sur les tableaux, mais voici une référence qui vous aidera :  
[Sort array by firstname \(alphabetically\) in Javascript \[duplicate\]](#)
- Comme pour la suppression d'une tâche, il vous faudra réinjecter dans le *DOM* toutes les tâches, mais suivant le nouvel ordre du tableau **aTaches**. Encore une fois, n'oubliez pas que c'est du nouveau *DOM*, il faudra alors ré-instancier la classe **Tache** pour chaque tâche réinjectée.

## Plus-value (nice- to-haves)

- Animation de type accordéon (sans *jQuery*) de la section affichant le détail d'une tâche.
- Dessiner le chevron en CSS afin d'éviter l'ajout d'un SVG.
- Persévérance des tâches (côté client) au rechargement de la page. Pour ce faire, il faut utiliser l'API Web Storage que nous n'avons pas vu en classe, je pourrai vous aider au besoin.
- Factoriser la mécanique d'injection de tâche en utilisant la syntaxe *ES6* d'héritage. Ici aussi, comme c'est une matière à voir la session prochaine, je pourrai vous aider au besoin.

## Exigences techniques et critères d'évaluation

- Fonctionnement conforme aux exigences
- Réussite des différentes fonctionnalités
- Réussite de la fonctionnalité d'ajout d'une tâche (validée) une semaine après la remise de l'énoncé
- *HTML* sémantique
- Utilisation de la programmation orientée objet
- Absence de fonctions fléchées (*arrow functions*)
- Qualité des algorithmes
- Qualité du code source (clarté et organisation)
- Commentaires fréquents et pertinents
- Le site est en ligne

# Plagiat

Nous travaillons dans le Web, alors les réponses s'y trouvent et il va de soi que vous allez devoir chercher sur *Google*. Dans l'esprit *open-source* du Web, vous avez tout à fait le droit de copier-coller un bout de code que vous avez trouvé. Cela-dit, vous devez citer les extraits de code qui dépassent une ligne ou deux. Évidemment, générer du code via un outil d'intelligence artificiel tel que *ChatGPT* et/ou suivre un tutoriel (en tout ou en partie) sera considéré comme du plagiat. Pour prévenir le copier-coller direct d'Internet, aucune syntaxe autre que celles vues en classe ne sera acceptée.

Il s'agit d'un travail strictement individuel. Vous pouvez vous aider quelque peu entre vous, mais faites attention à ne pas partager ou récupérer des fonctionnalités complètes. Je rappelle encore une fois qu'un des objectifs principaux de la formation est de développer votre autonomie, c'est important. En cas de plagiat, je devrai appliquer les sanctions de la *PIEA* pour les étudiant.e.s ayant reçu et partagé le code.

Si vous êtes vraiment bloqué, contactez-moi.