Questionnaire d'examen

Evaluation de Javascript : avancé (Vinci)	
Titulaire(s) :	Raphaël Baroni, Sébastien Strebelle
Année(s) d'études :	Bloc 2
Durée :	3h de 13h30 à 16h30 Pas de sortie ni soumission durant les 60 premières minutes
Modalités :	Accès à internet & aux supports de cours

Consignes générales

Vous avez trouvé sur EvalMoodle un fichier **examen_js.zip**.
Cette archive contient un boilerplate pour chaque question et les ressources à utiliser pour cet examen.

Développement de vos applications

Votre dossier d'examen doit se trouver localement sur votre machine : il n'est pas autorisé que celui-ci se trouve sur un disque réseau de Vinci ou sur le cloud (OneDrive, Gitlab, Github ou autres).
Renommez le dossier **examen_js** en **NOM_PRENOM**, comme par exemple **UCHIHA_ITACHI**
Pour chaque question, installez d'abord les packages associés au boilerplate.
Vous pouvez modifier ou ajouter autant de fichiers que nécessaires pour chaque question. N'hésitez pas à installer de nouveaux packages si nécessaires ou à utiliser du code offert dans les support du cours de JS. Ne vous attardez pas sur l'esthétisme de vos pages, cela ne sera pas évalué.

Soumission de votre code

Vous devez effacer les 3 répertoires **node_modules** se trouvant dans vos sous-répertoires **./question1**, **./question2** & **./question3**. Si vous ne le faites pas, vous ne pourrez pas soumettre votre projet sur evalMoodle !
Créez **un fichier .zip** nommé **NOM_PRENOM.zip** de votre répertoire **NOM_PRENOM**.
Vérifiez bien votre .zip avant de le poster.
Remettez ce fichier .zip sur Evalmoodle dans le devoir Examen de Javascript.

Remarques

La collaboration entre les étudiants est interdite et sera donc lourdement sanctionnée si elle se produit. Un outil de détection de plagiat sera utilisé.
La génération de code par des outils d'Intelligence Artificielle tels que Github Copilot et ChatGPT est également interdite.
Si une question d'examen ne s'exécute pas ou ne donne aucun résultat fonctionnel, vous aurez d'office moins de 50% des points pour cette question.

Objectif

Vous devez développer deux frontends et une API pour une entreprise d'Intelligence Artificielle.

Question 1 : Page web dynamique (6 points)

Vous devez implémenter une première version pour un chatbot. Votre site web devra permettre d'afficher une réponse textuelle à une question posée par l'utilisateur.

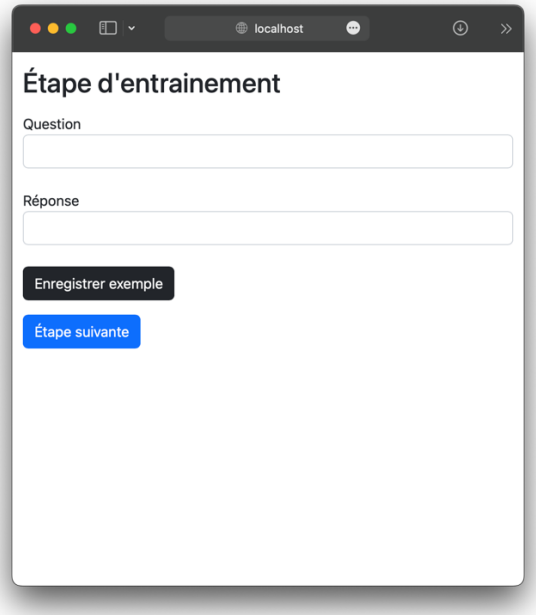
Les algorithmes d'intelligence artificielle fonctionnent en deux phases. La première phase d'apprentissage permet d'entrainer un modèle sur base d'exemples. La deuxième phase génère de nouvelles réponses sur base du modèle.

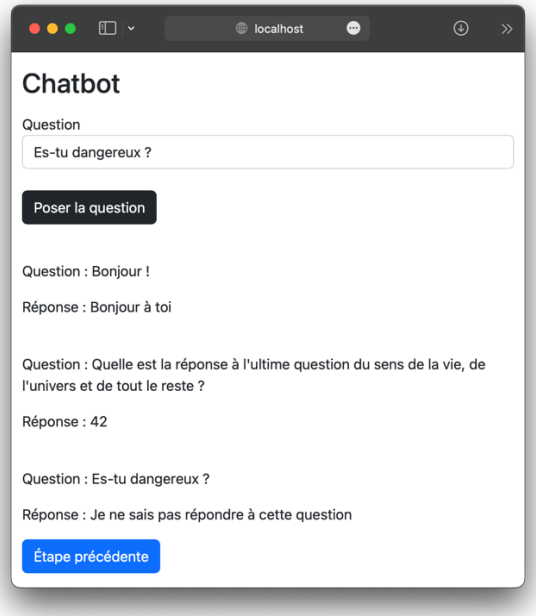
Vous ne devrez pas développer d'algorithme d'intelligence artificielle pour cet examen. Votre objectif est de créer la maquette de ce site web afin de le tester avant de rajouter la couche d'IA. Afin de simuler l'algorithme, les réponses seront directement tirées des exemples fournis lors de la phase d'apprentissage.

Le boilerplate pour cette question se trouve dans le dossier **/question1**. Il correspond au boilerplate *js-basic-boilerplate* vu au cours. Complétez le pour affichez une page web permettant de :

- Dans un premier temps, le site web affiche un formulaire avec deux champs : un champ pour une question et un champ pour sa réponse.
  Un bouton de soumission de ce formulaire enregistre cette paire de question/réponse et vide les champs du formulaire.
  Un autre bouton permet de passer à l'étape suivante.
- Dans un deuxième temps, le site web n'affiche plus le formulaire précédent et affiche à la place un formulaire permettant de rentrer une question.
  En soumettant la question avec un bouton, le site web cherche une réponse à cette question. Pour cette version, vous pouvez vous contenter d'une recherche basique, en vérifiant si la question posée correspond exactement à une des questions enregistrées.
  Le site web rajoute à la suite de la page la question posée et la réponse qui a été trouvée. Si aucune réponse n'a été trouvée, la page affiche à la place « Je ne sais pas répondre à cette question ». L'ensemble des questions posées et leurs réponses doivent être affichées, afin de pouvoir visualiser toute la conversation avec le chatbot.
  Un autre bouton permet de revenir à l'étape précédente pour rajouter de nouvelles paires de questions/réponses. Les paires précédentes doivent être conservées.

Voici un exemple de ce à quoi pourrait ressembler votre site web :





Question 2 : RESTful API (8 points)

Vous devez créer une API backend permettant d'enregistrer les achats d'utilisateurs afin de créer un système de recommandations utilisant l'intelligence artificielle. Encore une fois, vous ne devrez pas développer le système d'intelligence artificielle pour cet examen. Votre objectif est de créer la maquette de ce backend afin de le tester avant de rajouter la couche d'IA.

Le boilerplate pour cette question se trouve dans le dossier **/question2**. Il correspond au boilerplate *basic-api-boilerplate* vu au cours.

Au sein de votre boilerplate se trouve un fichier **constants.js** contenant une liste de pseudo d'utilisateurs existants, et une liste de produits existants avec leur ID, nom et prix d'achat unitaire.

Vous devez créer les opérations suivantes :

- **Enregistrer l'achat d'un utilisateur.**

  Cette opération doit avoir le chemin */purchases.*
  L'opération doit prendre en entrée le pseudo de l'utilisateur, l'ID du produit acheté et la quantité achetée. L'opération doit échouer si le pseudo n'est pas parmi les pseudos existants, ainsi que si l'ID du produit n'est pas parmi les produits existants. Enregistrez l'achat dans une liste sauvegardée dans un fichier *historique.json*.

- **Récupérer l'utilisateur achetant le plus fréquemment un produit.**

  Cette opération doit avoir le chemin */purchases/:productId.*
  L'opération doit prendre en entrée l'ID d'un produit. Elle doit échouer si l'ID du produit n'est pas parmi les produits existants. Cette opération renvoie le pseudo de l'utilisateur ayant acheté le plus d'unités de ce produit, selon la liste des achats enregistré dans le fichier *historique.json*.

- **Récupérer le produit recommandé pour un utilisateur.**

  Cette opération doit avoir le chemin */recommendations/:username.*
  L'opération doit prendre en entrée le pseudo de l'utilisateur. Elle doit échouer si le pseudo n'est pas parmi les pseudos existants. Cette opération renvoie le produit recommandé pour cet utilisateur. Pour cette maquette, choisissez un produit aléatoirement parmi la liste des produits existants.
- Vous devez également créer dans le fichier **/question2/RESTClient/tests.http** au minimum une requête par opération que vous devez implémenter. Vous pouvez également créer plus de tests si vous le souhaitez.

Contraintes d'implémentation :

- L'API que vous devez créer doit correspondre aux principes d'une API RESTful.
- Votre API ne peut pas proposer d'autres opérations que celles décrites ci-dessus.
- Les demandes qui sont créées doivent persister et donc survivre au redémarrage de votre application. Vous pouvez faire persister les demandes côté serveur de la manière que vous voulez, du moment que cela soit en JSON. N'hésitez pas à utiliser **/question2/utils/json.js**.
- Aucune autorisation JWT est nécessaire.

Question 3 : Single Page Application (6 points)

Vous devez créer un site web permettant d'entrainer un modèle d'IA de traduction français-anglais. Pour cette première version, l'API ne sait traduire que des mots uniques et pas de phrases.

Le boilerplate pour cette question se trouve dans le dossier **/question3**. Il correspond au boilerplate *js-router-boilerplate* vu au cours.

Une API backend a déjà été créée par une autre équipe. Elle vous est fournie dans le dossier **/backend-q3**. Vous pouvez lancer cette API en utilisant les commandes suivantes dans ce dossier :

$ *npm i*

$ *npm start*

Voici la documentation des opérations proposées par ce backend :

Method	Path	Action	Format
POST	/traduction	Ajoute la traduction d'un mot existant pour entrainer le modèle d'IA	Body : { fr, en }
GET	/traduction/fr?query=value	Traduit un mot du français vers l'anglais	Returns : { fr, en }
GET	/traduction/en?query=value	Traduit un mot de l'anglais vers le français	Returns : { fr, en }

Vous devez créer deux pages sur le site web (la page d'accueil ne doit rien afficher, et aucune autre page ne doit être disponible) :

- **La page d'entrainement**, disponible avec l'URI **/train** et avec un lien nommé « **Entrainement** » dans la barre de navigation.

  Cette page doit afficher un formulaire avec deux champs, un pour le mot en français et un autre pour sa traduction en anglais, et un bouton de soumission. Lorsque ce formulaire est soumis, vous devez utiliser la requête POST du backend pour enregistrer une nouvelle traduction au modèle d'IA. Après avoir soumis le formulaire, ses champs doivent être vidés pour que de nouvelles traductions puissent être soumises.

- **La page de traduction**, disponible avec l'URI **/trad** et avec un lien nommé « **Traduction** » dans la barre de navigation.

  Cette page doit afficher deux formulaires. Le premier permet d'entrer un mot en français et d'obtenir sa traduction anglaise, et le deuxième permet d'entrer un mot anglais et d'obtenir sa traduction française. Dans les deux cas, vous devez utiliser les requêtes GET du backend pour effectuer la traduction. Celle-ci sera écrite en dessous du formulaire correspondant. Si la requête renvoie une erreur signifiant que le mot n'a pas su être traduit, vous devez afficher en rouge le message « Impossible d'obtenir la traduction » à la place de la traduction.

Voici un exemple de ce à quoi pourrait ressembler votre site web :