Questionnaire d'examen

Evaluation de Javascript : avancé (Vinci)	
Titulaire(s) :	Raphaël Baroni, Sébastien Strebelle
Année(s) d'études :	Bloc 2
Durée :	3h de 14h00 à 17h00 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 ! Vous devez aussi **effacer** le répertoire **backend-q3** !
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 qui créée des jeux éducatifs pour enfants.

1 Question 1 : Page web dynamique (8 points)

Votre application doit proposer des jeux de devinettes éducatifs pour aider les enfants à améliorer leur logique, leur raisonnement et leurs compétences de résolution de problèmes.

Au chargement de votre application, vous devez aléatoirement afficher trois devinettes **sur base d'un import** des questions se trouvant dans **/question1/src/utils/questions.js**. Vous devez aussi ajouter un bouton à la fin du questionnaire permettant de calculer le score.

Voici comment vous devez afficher chaque devinette :

- La question doit être facilement reconnaissable par rapport aux réponses.
- Chaque réponse doit être associée à un « radio button ».
- Il doit être impossible de « checker » plus qu'un « radio button » par devinette (car il n'y a jamais qu'une seul réponse possible !).

Lorsque l'utilisateur clique sur le bouton pour calculer le score, vous devez **afficher uniquement** le **score** de l'utilisateur et un **bouton permettant de recommencer un jeu de devinettes**. Les questions ne sont donc plus visibles. Pour calculer le score :

- Une réponse juste amène 1 point.
  NB : la réponse juste à une question doit être déterminée grâce à **isCorrect** dans **/question1/src/utils/questions.js** ; si **isCorrect** est vrai, c'est que c'est la bonne réponse.
- Une réponse fausse amène 0 points. Si aucune réponse est sélectionnée pour une question, cela reviendra à une réponse fausse.
- Le score sera donné sur 3 points.

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

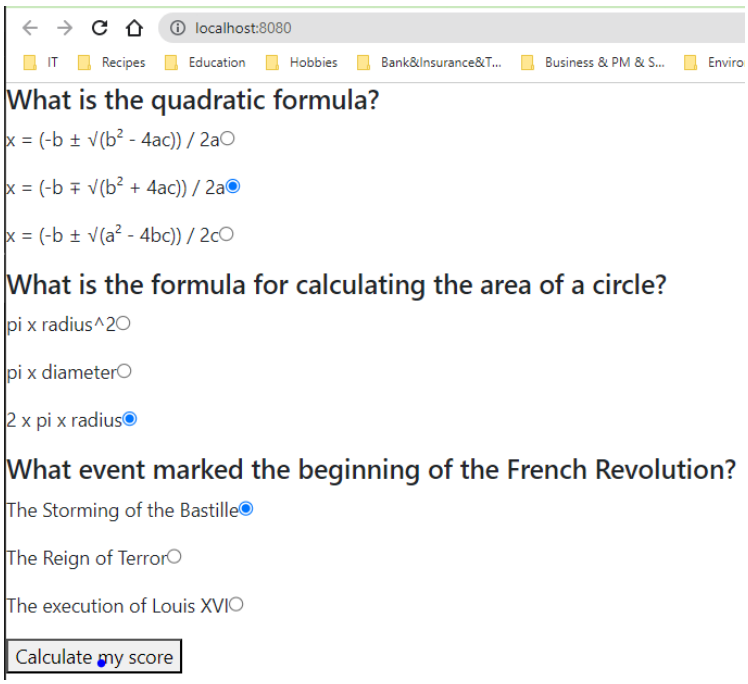

Figure 1- 3 devinettes avec les réponses de l'utilisateur

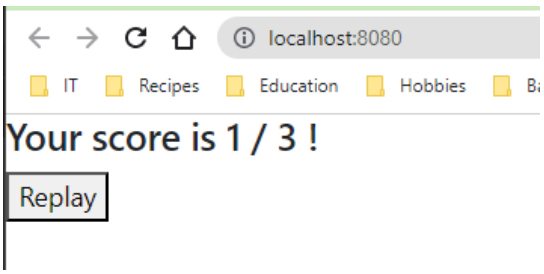

Figure 2 - Affichage du score de l'utilisateur après avoir demandé de calculer le score

Vérifiez bien que quand vous cliquez sur le bouton pour recommencer un jeu, 3 devinettes sont sélectionnées aléatoirement et affichées !

2 Question 2 : RESTful API (7 points)

Vous devez créer une API backend permettant de gérer des parties de devinettes éducatives.

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

Vous allez créer les opérations qui suivent.

2.1 Récupérer un jeu de devinettes éducatives

Cette opération doit avoir le chemin **/games/start ou /games/start?level=value**. Cette opération doit renvoyer un jeu de 3 devinettes contenant une question ainsi que les réponses associées, pour le niveau sélectionné.

Un paramètre de requête **level** doit permettre de forcer d'avoir un jeu de 3 devinettes pour un niveau. Si le paramètre de requête n'est pas indiqué, on sélectionne des devinettes parmi tous les niveaux. Si le **level** donné ne correspond pas à un **level** existant, vous devez renvoyer le code d'erreur approprié.

Vous devez utiliser les devinettes existantes qui se trouvent dans **/question2/data/questions.json** pour sélectionner 3 devinettes aléatoirement.

NB : Vous devez renvoyer toutes les informations associées à une question (id, level, category, question, answers, text, isCorrect).

Vous devez également créer dans le fichier **/question2/REST Client/tests.http** avec au minimum deux requêtes pour tester l'API :

- une sans paramètre de requête **level**.
- une avec le paramètre de requête **level**.

2.2 Enregistrer le résultat d'un jeu d'un utilisateur

Cette opération doit avoir le chemin */games.*

Pour cette version de l'API :

- il est acceptable que ça soit le client qui calcule le score et pas l'API.
- il est acceptable que l'opération ne soit pas protégée par un token JWT.

Le client vous envoie un score & son username lors de la requête demandant l'enregistrement du résultat d'un jeu d'un utilisateur. Si le score n'est pas compris entre 0 et 3, vous devez renvoyer un « status code » approprié.

Vous devez enregistrer de manière persistante au sein de **/question2/data/games.json** le résultat du jeu :

- en indiquant le username ;
- en reprenant le score ;
- en y ajoutant le temps système (une date) au moment de l'enregistrement du résultat.

La réponse de l'opération d'enregistrement du résultat d'un jeu doit renvoyer la ressource ajoutée au fichier **games.json**.

Vous devez également ajouter dans le fichier **/question2/REST Client/tests.http** 2 requêtes pour tester l'API :

- une requête pour créer un résultat d'un jeu en envoyant un **score correct** pour un utilisateur ;
- une requête pour créer un résultat d'un jeu en envoyant un **mauvais score** pour un utilisateur.

3 Question 3 : Single Page Application (5 points)

Vous devez créer une application web permettant aux enfants de demander des devinettes sur des sujets qui leur tiennent à cœur.

Notons que pour cette version de l'application, il est acceptable que les utilisateurs ne doivent pas se loguer et que la fonctionnalité de mise à jour du statut ne soit pas encore disponible.

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	/queries	Créer une demande de devinettes sur un sujet précis.	Body : { subject: "chemical reaction", status:"requested"} Returns : { id: …, subject: "chemical reaction", status:"requested"}
GET	/queries	Lire toutes les demandes de devinettes	Returns : [{id:…, subject:.., status:…}, {id:…, subject:…, status:…}]

Vous devez créer les deux pages qui suivent pour votre application web (la page d'accueil ne doit rien afficher, et aucune autre page ne doit être disponible).

3.1 Création d'une demande

Veuillez créer la **page de création d'une demande**, disponible avec l'URI **/queries/create** et avec un lien nommé « **Create a query** » dans la barre de navigation.

Cette page doit afficher un formulaire avec un seul champs, pour le sujet de la demande de devinette. Lorsque ce formulaire est soumis, vous devez utiliser la requête POST du backend pour

enregistrer une nouvelle demande au statut "requested". Après avoir soumis le formulaire avec succès, l'utilisateur doit être **redirigé** vers la **page de gestion des demandes**.

3.2 Gestion des demandes

Veuillez créer la **page de gestion des demandes**, disponible avec l'URI ***/queries*** et avec un lien nommé « **Manage queries** » dans la barre de navigation.

Cette page doit afficher toutes les demandes en utilisant la requête GET du backend. Le statut pour chaque requête doit être affiché dans une liste déroulante qui contiendra toujours ces 4 statuts :

- "requested"
- "accepted"
- "refused"
- "done"

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

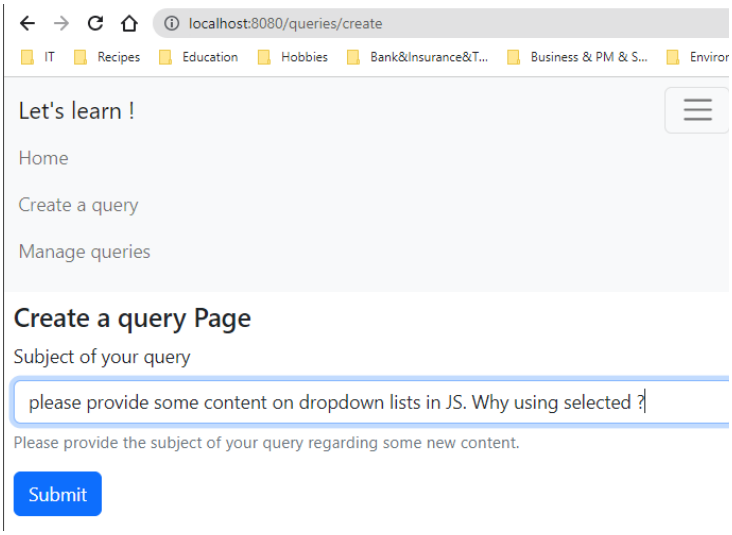

Figure 3- Page pour la création d'une demande

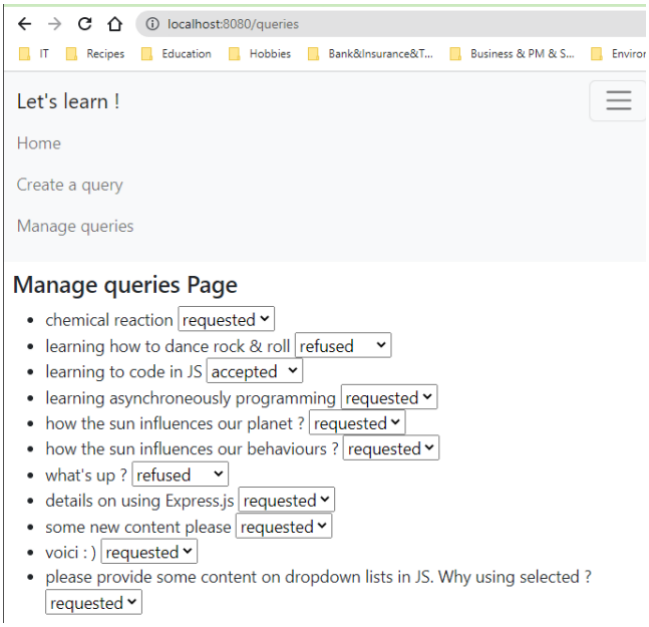

Figure 4 – Page permettant de gérer les demandes