**Questionnaire d’examen**

|  |  |
| --- | --- |
| Evaluation de Web : questions spéciales (Vinci) | |
| Titulaire(s) : | Laurent Lelleux, Raphaël Baroni |
| Année(s) d’études : | Bloc 2 |
| Durée : | 3h00 de 09h00 à 12h00  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. Ne vous attardez pas sur l’esthétisme de vos pages, cela ne sera pas évalué.

###### Soumission de votre code

Vous devez **effacer** les répertoires **node\_modules** se trouvant dans vos sous-répertoires **./question1** & **./question2**. Si vous ne le faites pas, vous ne pourrez pas soumettre votre projet sur evalMoodle ! Vous devez aussi **effacer** le répertoire **question1-api** !

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 Web3.**

**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 autorisée.

###### 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 allez développer un frontend et une API en lien avec des personnalités et les blagues qu’elles auraient dites.

# Question 1 : Création d’IHM en React (13 points)

Votre application doit permettre d’afficher des blagues associées à des personnalités.

Une API backend a déjà été créée par une autre équipe. Elle vous est fournie dans le dossier **/question1-api** : entrez dans ce dossier et lancez le web service en utilisant les commandes suivantes :

$ *npm i*

$ *npm start*

Voici la documentation d’une opération proposée par le web service que vous allez devoir utiliser :

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Path** | **Action** | **Format** |
| **GET** | **/famous-jokes** | Lire toutes les blagues de personnalités | Returns : [{name:…, joke:.., imageUrl:…}, {name:…, joke:…, imageUrl:…}] |
| **PATCH** | **/famous-jokes/:id** | Mettre à jour une blague identifiée par son id | Body : { imageUrl: "https://link.be/super.jpg"}  Returns : { name: …, joke: "…", imageUrl:"https://link.be/super.jpg"} |

Vous allez créer 2 pages pour votre frontend au sein du répertoire **/question1**. C’est une application configurée pour utiliser **Vite**.

**Contraintes d’implémentation**

Mettez en place les bonnes pratiques vues dans le cours :

* créez de composants React pour vos éléments de UI dans des dossiers tels que **/src/components/App**…
* utilisez des imports absolus pour vos scripts (tous est déjà configuré pour vous au sein de **vite.config.js** & **jsconfig.json**).
* …

## Créer la page d’accueil

Veuillez créer la **page d’accueil**, disponible avec l’URI « **/ »** et avec un lien nommé « **Home** » dans la barre de navigation. Cette page doit uniquement afficher ce titre : « Page d’accueil des blagues les plus célèbres ! ».

**Contraintes d’implémentation**

* Utilisation du router offert par React.

## Gérer des blagues célèbres

Veuillez créer la **page de gestion des blagues**, disponible avec l’URI **/famous-jokes** et avec un lien nommé « **Gestion de blagues**» dans la barre de navigation.

Cette page doit afficher toutes les blagues en utilisant la requête **GET** du web service.

**Contraintes d’implémentation**

* Utilisation du router offert par React.
* Création et utilisation d’un Context qui doit offrir toutes les opérations sur les blagues.
* À chaque affichage de la page, vous devez faire en sorte que les blagues soient réactualisées en faisant appel au web service.

Pour chaque blague, vous devez afficher :

* le nom complet de la personnalité ;
* le texte de la blague ;
* le lien de l’image associée à la personnalité: par exemple, "<https://e-vinci.github.io/ai-pics/kevin-hart-running2.jpg>"; on doit directement voir que ce lien est modifiable.
* un bouton « Mettre à jour l’image » à côté du lien vers l’image ;
* l’image associée à la personnalité ; l’image doit avoir une hauteur & largeur maximum de 200 pixels. Vous ne devez pas gérer le cas où l’image n’existerait pas.

Lorsque l’utilisateur clique sur le bouton « Mettre à jour l’image» : vous devez :

* Faire appel à la requête **PATCH** du web service pour mettre à jour le lien de l’image associée à la blague.
* Puis faire en sorte que toutes les blagues soient à nouveau « fetch » du web service et que le contenu de la page soit mis à jour.

|  |  |
| --- | --- |
| Figure 1- Affichage de la HomePage | Figure 2 – Affichage des blagues célèbres |
| Figure 1- Affichage lors de la mise à jour du lien, avant de cliquer sur « Mettre à jour l’image » | Figure 1- Affichage après la mise à jour du lien, après avoir cliqué sur « Mettre à jour l’image » |

# Question 2 : Création de services web (7 points)

## Mise en place du projet & de la DB

Vous allez développer un web service permettant de lire et commenter des blagues célèbres.

Le boilerplate pour cette question se trouve dans le dossier **/question2**. Il correspond au boilerplate utilisé dans le cours pour une RESTful API bien structurée et dont les données sont traitées via MongoDB & Mongoose.

Pour les utilisateurs de Windows :

* Si vous démarrez l’application se trouvant dans le projet **/question2**, l’API va créer un serveur de développement MongoDB qui sauvegardera les données en mémoire vive (package **mongodb-memory-server**) et va automatiquement ajouter les ressources qui se trouvent dans le fichier **db-data.json**.
* Vous ne devez donc pas ajouter de données manuellement à MongoDB, tout se fait automatiquement pour vous au sein de la DB « exam-web3 » : )

Pour les utilisateurs de Linux : vous devez passer par MongoDB Atlas tel qu’indiqué ci-dessus ! **mongodb-memory-server** ne fonctionne pas sous Ubuntu !

Si vous préférez ou devez (si vous êtes sous Linux) utiliser un serveur distant MongoDB Atlas, vous pouvez le faire en modifiant les variables d’environnement du projet **/question2** pour vous connecter à votre DB. Là aussi, les données de départ seront automatiquement chargées pour vous au démarrage de l’API. Le minuscule avantage d’utiliser Atlas en développement, c’est que vous avez une interface web pour visualiser vos collections, ce qui n’est pas le cas si vous utiliser le serveur local MongoDB de développement. Néanmoins, cette fonctionnalité n’est pas vraiment nécessaire pour développer ce qui vous est demandé...

## Commenter une blague célèbre

Veuillez offrir une opération permettant de commenter une blague présente dans la collection « jokes » sur base de son identifiant. Vous devez choisir un chemin pour atteindre cette opération, une méthode http et des paramètres qui respectent les conventions REST.

Lors d’une requête demandant l’ajout d’un commentaire sur une blague, le client envoie :

* l’identifiant de la blague,
* son username,
* son commentaire (par exemple : « je trouve cette blague très scientifique, tout en étant très légère ») ».

Vous devez renvoyer un « status code » approprié :

* Si le commentaire ne fait pas plus de 5 charactères ;
* Si la blague n’existe pas ;
* Si le username ne fait pas plus de 3 charactères ;
* Si l’utilisateur a déjà fait un commentaire pour cette blague.

Si tout est OK, vous devez enregistrer le commentaire dans la collection existante « jokes » de MongoDB :

* en indiquant le username ;
* en y ajoutant le commentaires.

La réponse de l’opération d’enregistrement de l’évaluation doit renvoyer uniquement ces informations :

* l’id de la blague ayant reçu un commentaire ;
* le username ;
* le commentaire ajoutée.

Vous devez également ajouter dans le fichier **/question2/REST Client/tests.http** (fichier à utiliser avec l’extension Rest Client de VS Code) toutes les requêtes nécessaires pour tester l’API afin :

1. d’ajouter un **commentaire** **correct** pour une blague ;
2. de tenter d’ajouter un **commentaire non valide** de 5 charactères ;
3. de tenter d’ajouter un commentaire pour un **username** **invalide** de 3 charactère ;
4. de tenter d’ajouter un commentaire pour une **blague qui n’existe pas**,
5. de tenter d’ajouter un commentaire pour une **blague déjà commentée par ce username**.

**Contraintes d’implémentation**

* Utilisation des conventions RESTful.
* Utilisation des schémas et modèles **Mongoose**.
* Transformation des propriétés **\_id** en **id** dans votre API.

## Afficher toutes les blagues célèbres

Veuillez offrir une opération permettant d’afficher toutes les blagues célèbres présentes dans la collection « jokes » de votre DB, ainsi que tous les commentaires associés.

Vous devez également ajouter dans le fichier **/question2/REST Client/tests.http** (fichier à utiliser avec l’extension Rest Client de VS Code) une requête pour tester la lecture de toutes les blagues et les commentaires associés.

**Contraintes d’implémentation**

* Utilisation des conventions RESTful.
* Utilisation des schémas et modèles **Mongoose**.
* Transformation des propriétés **\_id** en **id** dans votre API.