Institut Paul Lambin Session d'août 2023

Examen de l'UE : Développement Web – Questions spéciales

Titulaire : Laurent Leleux
Bloc : 3BIN

Date et heure : 16/08/2023 à 9h30
Locaux : Ch43 B22
Durée de l'examen : 3h

Consignes : sur machine, à cours ouvert, internet ouvert, communication entre étudiants ou avec l'extérieur interdits.

/!\ Lisez attentivement les consignes et l'introduction AVANT de démarrer l'examen /!\

Consignes et informations générales

Vous avez à votre disposition :

- Internet : vous pourrez donc **consulter** Moodle, les solutions des séances, des tutoriels en ligne, des forums... Cependant, **toute communication entre vous ou avec l'extérieur est formellement interdite**, sera considérée comme tricherie, et est donc passible de sanctions lourdes conformément au règlement des études.
- Vos notes au format papier.
- Vos notes au format électronique (USB), mais l'usage de clefs USB n'est techniquement pas garantie sur les machines d'examen.
- Le boilerplate de l'examen est disponible sur EvalMoodle. Il contient les sources d'un projet de base qu'il vous faudra améliorer (dossiers **api** et **webapp**) ainsi qu'un fichier db.json qui contient les données qui seront automatiquement chargées dans votre DB MongoDB, et un fichier Readme.

Comment démarrer :

- Veuillez dézipper le boilerplate sur votre machine, sur le bureau. Ne travaillez pas sur le lecteur réseau, ce sera beaucoup trop lent.
- Pour les utilisateurs de Windows :
    - Si vous démarrez l'application se trouvant dans le projet « api », 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.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 » :)
- Si vous préférez utiliser un serveur distant MongoDB Atlas, vous pouvez le faire en modifiant les variables d'environnement du projet « api » 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...

- Pour les utilisateurs de Linux : vous devez passer par MongoDB Atlas tel qu'indiqué au bullet ci-dessus ! mongodb-memory-server ne fonctionne pas sous Ubuntu !
- Vous pouvez à présent travailler dans les deux dossiers du projet (api et webapp).
- Une fois l'examen terminé, faites une archive .zip de votre dossier « projet » **sans les dossiers « node_modules »**, et appelez-là « **web3_NOM_PRENOM.zip** ». Ensuite veuillez soumettre cette archive sur evalMoodle avant la fin de l'examen.
- À moins d'avoir ajouté des images, votre projet ne doit pas faire plus qu'un MB. La soumission n'autorise **pas d'envoi de plus de 10 MB**.

Nous allons exécuter votre code sur un logiciel de **détection de plagiat**. **Toute tentative de fraude, tout partage de code... sera sévèrement punie.**

Introduction

Dans ce projet, nous allons développer une application qui permettra de gérer des blagues et de permettre aux utilisateurs de leur donner un score sur 10.

Pour cela, nous allons avoir une liste de blagues, et pour chaque blague, une liste de scores donnés par les utilisateurs.

Pour cette version du projet, nous ne gérons pas l'authentification des utilisateurs ni la sécurisation des opérations des API. Nous considérons que les utilisateurs sont déjà inscrits.

1. Fonctionnalités back-end à développer

1.1 API – ressources et routes de base (4 points)

Un squelette de l'API est fourni, cependant celui-ci ne contient aucune ressource et aucune route pour l'instant.

Ajoutez une ressource `jokes`, et une ressource `scores`. Ces ressources doivent être récupérées dans la DB à l'aide des schémas et modèles Mongoose. Pour connaitre les schémas, utilisez le fichier `db.json` qui vous est fourni, et que vous avez déjà injecté dans votre DB sur MongoDB Atlas.

La ressource de type scores contient un id de `jokes`. Il s'agit en quelque-sorte d'une FK. Pour la modéliser, vous pouvez utiliser le type « ObjectId » de Mongoose dans votre Schéma.

Pour les identifiants de vos ressources, même si nous avons mis `_id` dans `db.json`, car c'est la clé que MongoDB utilise par défaut pour les identifiants, nous souhaitons qu'au niveau de votre API, les ressources soient identifiées par `id` ; n'oubliez donc pas de transformer les propriétés `_id` en `id` dans votre API !

Pour les ressources de type `jokes`, il faut les routes de CRUD suivantes :

- Read all
- Read one (by id)
- Delete one (by id)
- Create one

Pour les ressources de type score, il faut les routes suivante :

- Read all
- Create one

N'hésitez pas à utiliser le dossier `requests` avec vos requêtes (fichiers *.http) pour faire vos tests. Pour rappel, il faut l'extension « humao.rest-client » pour utiliser ces fichiers avec VSCode. Vous pouvez également utiliser postman, insomnia...

1.2 API – validation (2 points)

Lors de l'insertion d'une blague (ressource de type `jokes`) ou d'un score (ressource de type `scores`), assurez-vous que toutes les informations utiles sont contenues (cfr. `db.json`).

Lors de l'insertion d'une blague, assurez-vous que `question, answer & category` fasse au minimum 3 caractères.

Lors de l'insertion d'un score, assurez-vous que la blague associée existe bien dans la DB et que `username` fasse au minimum 3 caractères. On ne voudrait pas insérer un score qui ne soit associé à aucune blague…
Assurez-vous aussi que le username n'aie pas déjà donné un score à cette blague… En effet, on ne voudrait pas permettre au même utilisateur de donner plusieurs scores à la même blague !

Veillez à employer un statut d'erreur http pertinent en cas d'erreur. Le message n'est pas important.

2. Fonctionnalités front-end à développer

Attention, depuis la fin du cours, une nouvelle version de Ant.design est sortie. Dans cet examen, nous ne l'avons pas utilisée pour ne pas changer par rapport à ce qui a été travaillé durant l'année.

2.1 Context (4 points)

Créez un `Context` qui va contenir :

- La liste de toutes les `jokes`
- La liste de tous les `scores`

Les blagues et les scores sont chargés depuis l'API automatiquement au chargement du `Context`.

Pour la liste de blagues, vous devez ajouter à chaque blague ces propriétés :

- Le nombre de scores donnés par des utilisateurs, à nommer `scoreCount` ;
- Le score moyen, à appeler `averageScore`. Le score moyen (somme de tous les scores associés à une blague divisé par le nombre de scores) est à arrondir à un chiffre après la virgule.

Pour faire les requêtes à l'API, utilisez et complétez les services fournis : `jokeApi` et `scoreApi`.

2.2 Données d'une blague (1 point)

Pour afficher une blague avec la liste de tous les scores associés, nous avons besoin d'une méthode supplémentaire dans le `Context` : `getJokeWithScores(id)` qui renvoie la blague, avec dedans ces propriétés :

- `scores` : contient une liste avec toutes les scores associés.
- scoreCount : contient le nombre de scores donnés par les utilisateurs à cette blague.

- averageScore : contient le score moyen.

Voici ce qu'on pourrait recevoir :


```
{
    "question": "Why are modern programming languages so materialistic?",
    "answer": "Because they are object-oriented.",
    "category": "Programming",
    "id": "6461f476d9a9da9dbeade34e",
    "scoreCount": 2,
    "averageScore": 9,
    "scores": [
        {
            "username": "camille",
            "date": "2021-01-02T10:17:35.457Z",
            "score": 10,
            "joke": "6461f476d9a9da9dbeade34e",
            "id": "646b7150425fdc1628b206eb"
        },
        {
            "username": "laurent",
            "date": "2023-01-02T15:49:27.457Z",
            "score": 8,
            "joke": "6461f476d9a9da9dbeade34e",
            "id": "646b7150425fdc1628b206ed"
        }
    ]
}
```


2.3 Liste des blagues (2 points)

- Faites en sorte d'afficher une liste de toutes les blagues, en utilisant les données du `Context`. Cette liste peut être rudimentaire, inutile d'y ajouter du style. Il n'y a que `question` & `answer` qui doivent y être affichés.

2.4 Routage (3 points)

Dans cet exercice, vous ne devez pas encore rendre fonctionnel la page d'une blague (avec la liste des scores associés), ni faire une page « about » exhaustive (vous donnerez simplement un titre à la page « About »)… Vous allez vous concentrer sur le routage.

Modifiez l'application pour qu'elle soit composée de trois routes distinctes :

- `/about` : la page qui décrira le site.
- `/jokes` : contient la liste de toutes les blagues.
- `/jokes/id` : contiendra une page d'une blague (id en paramètre), avec la liste de ses scores, ainsi que le formulaire d'ajout d'un score.

Utilisez pour cela le module `react-router`. Vous pouvez choisir la version utilisée.

Veillez à garder une structure cohérente de l'application (découpe en composants).

Faites en sorte que la navbar apparaisse tout le temps, sur chaque page, et que les liens redirigent correctement sur chaque page. Pour voir la page d'une blague, il faut cliquer sur la blague dans la liste. Attention, l'application doit rester une Single Page Application.

2.5 Gestion des scores (4 points)

Rendez fonctionnel la page d'une blague, avec la liste des scores associés, ainsi que l'ajout de scores... On doit pouvoir :

- Voir les informations d'une blague (category, question & answer)
- Visualiser le score moyen associé à une blague ainsi que le nombre de scores donnés
- Visualiser la liste des scores de la blague, triés par score décroissants (les scores les plus grands au-dessus) et par date plus récentes en cas de scores identiques
- Ajouter un score à l'aide d'un input pour le username et d'un input pour le score ne permettant que des chiffres entre 0 et 10 et d'un bouton « ajouter ».
  NB : L'ajout d'un score devrait se faire en appelant une nouvelle fonction de votre Context qui devrait faire appel au service scoreApi pour consommer l'API.

Après l'ajout d'un score, la liste doit automatiquement être mise à jour avec le nouveau score, et le formulaire doit être reset.