Commencé le	samedi 23 avril 2022, 09:30
État	Terminé
Terminé le	samedi 23 avril 2022, 11:59
Temps mis	2 heures 28 min
Note	**33,00** sur 40,00 (**83**%)

Description

LISEZ CES INSTRUCTIONS AVANT DE COMMENCER L'EXAMEN

L'examen est composé de 2 sections. Vous êtes libre de changer de section en tout temps et de changer les réponses à vos questions. Votre examen sera évalué seulement après avoir cliqué sur le bouton "Tout envoyer et terminer" et avoir confirmé la soumission.

Voici les règles pour l'évaluation:

- L'examen possède **15** questions notées sur un total de 40 points.
- La note partielle associée à chaque question est marquée à gauche de la question.
- Certaines questions requièrent plusieurs champs à remplir, tandis que d'autres questions sont à choix multiples.
- Un mauvais choix dans une question à réponses multiple impactera la note finale de la question.
- L'espace disponible n'est pas nécessairement représentatif de la taille de la réponse attendue : **ne vous sentez pas obligés de remplir tout l'espace donné**.

Vous avez une seule tentative pour l'examen final! Ne soumettez pas votre tentative à moins d'être 100% sûr(e) d'avoir terminé l'examen !

En cas de doute sur le sens d'une question, faites une supposition raisonnable, énoncez-la clairement dans votre réponse et poursuivez. Une "question" vide est disponible sur la première page d'examen si vous avez besoin de plus de place ou pour des questions spécifiques.

Les téléphones et les ordinateurs portables ne sont pas permis. Sur votre poste de travail, vous pouvez utiliser seulement Moodle.

Les notes de cours peuvent être consultées dans Moodle, dans la section Ressources. Vous avez aussi droit à de la documentation papier (manuscrite ou imprimée).

Bon travail et bonnes vacances!

Question **1**

Non répondue

Non noté

Cette question est un espace dédié pour vos commentaires ou questions sur le contrôle pratique. Aucune réponse ne sera donnée pendant l'examen.

Priorisez de mettre vos commentaires et/ou hypothèses directement dans la question spécifique.

Question **2**

Correct

Note de 1,00 sur 1,00

Plusieurs fonctions et librairies dans le développement web font usage de **callbacks** . C'est quoi un *callback* ?

a. Un *Callback* est un objet qui nous donne accès à des paramètres supplémentaires tels que *request* et *response* pour la librairie Express par exemple.

b. Un *Callback* est simplement une fonction asynchrone.

c. Un *Callback* est une manière d'exécuter du code natif (C++) dans les fonctions natives du navigateur ou NodeJS.

d. Aucune des réponses.

e. Un *Callback* est une fonction appelée après la fin de l'exécution d'une fonction primaire. ✔

Votre réponse est correcte.

La réponse correcte est :
Un *Callback* est une fonction appelée après la fin de l'exécution d'une fonction primaire.

Question **3**

Correct

Note de 2,00 sur 2,00

Parmi les énoncés suivants, lesquels sont **erronés** ?

a. Il n'est pas possible d'annuler une requête HTTP faite avec Fetch. ✔

b. L'API Fetch supporte nativement les Promesses JavaScript.

c. Il n'est pas possible d'annuler une requête HTTP faite avec XMLHttpRequest (XHR).

d. L'API Fetch lancera une exception si le code de retour est un code d'erreur 400-499 ou 500-599.

e. L'utilisation de XMLHttpRequest (XHR) est limitée puisque l'objet supporte seulement des documents XML dans le corps des requêtes et les réponses HTTP. ✔ XMLHttpRequest peut être utilisé pour envoyer des requêtes et recevoir des réponses au format texte (par exemple, JSON, HTML, texte brut) et n'est pas limité spécifiquement à XML.

f. L'API Fetch ne supporte pas nativement les Promesses JavaScript. ✔

g. L'API Fetch est une version plus moderne de faire des requêtes HTTP que XMLHttpRequest (XHR) et qui est supporté seulement par NodeJS. ✔

Votre réponse est correcte.

Les réponses correctes sont :
L'API Fetch est une version plus moderne de faire des requêtes HTTP que XMLHttpRequest (XHR) et qui est supporté seulement par NodeJS.,

L'API Fetch ne supporte pas nativement les Promesses JavaScript.,

Il n'est pas possible d'annuler une requête HTTP faite avec Fetch.,

L'utilisation de XMLHttpRequest (XHR) est limitée puisque l'objet supporte seulement des documents XML dans le corps des requêtes et les réponses HTTP.

Question **4**

Correct

Note de 1,50 sur 1,50

Voici un URI relatif d'une requête gérée par Express : **/capitals?country=Canada**.

Comment peut-on récupérer la valeur *Canada* à partir de l'URI pour l'utiliser dans le code ?

a. req.country

b. req.values.country

c. req.query.country ✔

d. req.params.country

e. req.body.country

Votre réponse est correcte.

La réponse correcte est :
req.query.country

Question **5**

Correct

Note de 1,50 sur 1,50

Quelle est la différence entre la mise à l'échelle horizontale vs verticale d'une base de données?

a. L'horizontale permet d'avoir différentes colonnes pour chaque ligne d'un document permettant une BD plus flexible que la version verticale

b. L'horizontale est une méthode de réplication où la base de données interagit avec un serveur et une nouvelle instance est choisie en cas d'erreur, alors que la verticale concerne le partitionnement des données sur différents serveurs

c. L'horizontale permet d'avoir un schéma de tables dynamiques, alors que la verticale est liée à un schéma statique.

d. L'horizontale consiste à ajouter des machines dans la grappe de serveur pour diminuer la charge par machine, alors que la verticale consiste à augmenter la capacité d'une base de données en augmentant le nombre de CPUs, de RAM, etc. d'une seule machine. ✔

e. L'horizontale concerne la répartition des données entre plusieurs instances primaires, alors que la verticale concerne la répartition des données entre une instance primaire et des instances secondaires

Votre réponse est correcte.

La réponse correcte est :
L'horizontale consiste à ajouter des machines dans la grappe de serveur pour diminuer la charge par machine, alors que la verticale consiste à augmenter la capacité d'une base de données en augmentant le nombre de CPUs, de RAM, etc. d'une seule machine.

Question **6**

Partiellement correct

Note de 0,75 sur 1,50

Il est actuellement 10h le 23 avril 2022. Vous envoyez la requête suivante :

```
GET / HTTP/1.1

Host: www.polymtl.ca
```

Le serveur vous envoie la réponse suivante (où max-age est en secondes) :

```
HTTP/1.1 200 OK

Cache-Control: private, max-age=600

Last-Modified: Sat, 23 Apr 2022 10:00:00 GMT
```

Vous envoyez à nouveau la même requête à 10h05.

Qu'est-ce qui se produit avec la requête ?

Le navigateur fait un GET conditionnel pour valider la ressource dans sa cache avant de l'utiliser ✖

Si à la place, vous aviez envoyé la requête à 10h15.

Qu'est-ce qui serait produit avec la requête ?

Le navigateur fait un GET conditionnel pour valider la ressource dans sa cache avant de l'utiliser ✔

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 1.
La réponse correcte est :
Il est actuellement 10h le 23 avril 2022. Vous envoyez la requête suivante :

```
GET / HTTP/1.1
Host: www.polymtl.ca
```

Le serveur vous envoie la réponse suivante (où max-age est en secondes) :

```
HTTP/1.1 200 OK
Cache-Control: private, max-age=600
Last-Modified: Sat, 23 Apr 2022 10:00:00 GMT
```

Vous envoyez à nouveau la même requête à 10h05.
Qu'est-ce qui se produit avec la requête ?

[Le navigateur utilise la ressource dans sa cache]

Si à la place, vous aviez envoyé la requête à 10h15.
Qu'est-ce qui serait produit avec la requête ?

[Le navigateur fait un GET conditionnel pour valider la ressource dans sa cache avant de l'utiliser]

Question **7**

Correct

Note de 1,00 sur 1,00

Qu'est-ce qu'un témoin (*cookie*) signé et qui est responsable de sa signature ?

a. Le *cookie* peut être signé par le serveur et le client et permet de transmettre des données de manière sécuritaire entre le client et le serveur.

b. Le *cookie* est signé par le client et permet de détecter s'il a été modifié par le serveur.

c. Le *cookie* est signé par le serveur et permet de détecter s'il a été modifié par le client. ✔

d. Le *cookie* est signé par le client et permet de transmettre des données de manière sécuritaire au serveur.

Votre réponse est correcte.

La réponse correcte est :
Le *cookie* est signé par le serveur et permet de détecter s'il a été modifié par le client.

Question **8**

Correct

Note de 2,00 sur 2,00

Choisissez le(s) déclaration(s) correcte(s) en analysant le **package.json** ci-dessous.

```
{
  "name": "serveur_final",
  "version": "1.0.0",
  "description": "Serveur de l'examen final",
  "main": "main.js",
  "private": true,
  "scripts": {
    "start:watch": "nodemon main.js",
    "test": "jest"
  },
  "license": "ISC",
  "dependencies": {
    "express": "~4.17.1"
  },
  "devDependencies": {
    "jest": "^23.0.2"
  }
}
```

- [ ] a. L'installation d'une nouvelle dépendance en développement (*devDependencies*) dans le *package.json* peut être faite avec la commande **npm install-dev nomDuPackage**
- [x] b. Les commandes *npm test* et *npm run test* sont équivalentes et produiront le même résultat. ✔
- [x] c. La présence de la librairie *jest* n'est pas nécessaire pour le fonctionnement du projet. ✔
- [ ] d. Le nom du fichier dans la commande "start:watch" doit obligatoirement correspondre au nom du fichier dans la variable "main"
- [ ] e. Selon les dépendances, la version *4.18.1* de "express" sera téléchargée par npm étant donné que la dernière version de "express" est *4.18.1*.

Votre réponse est correcte.

Les réponses correctes sont :
La présence de la librairie *jest* n'est pas nécessaire pour le fonctionnement du projet. ,

Les commandes *npm test* et *npm run test* sont équivalentes et produiront le même résultat.

Question **9**

Incorrect

Note de 0,00 sur 1,50

Votre compagnie commence de travailler sur un jeu vidéo multijoueur accessible seulement à travers un site web.

Vous êtes à l'étape de conception de l'architecture de votre système global. Voici vos contraintes :

- Votre jeu doit supporter plusieurs joueurs en même temps sur des machines différentes.

- Afin d'éviter la triche, la logique du jeu doit être exécutée sur des machines autres que ceux des joueurs.

- Votre jeu est un jeu multijoueur seulement : vous n'avez pas à gérer un mode solo.

En fonction de vos besoins et vos contraintes, quelle serait la **meilleure** architecture à utiliser parmi les suivantes ?

a. Architecture Pipeline

b. Architecture orientée Micro-Services ✖

c. Architecture Client-Serveur

d. Architecture Pair-à-Pair

Votre réponse est incorrecte.

La réponse correcte est :
Architecture Client-Serveur

b) Pour un jeu vidéo multijoueur accessible uniquement via un site web, une architecture Client-Serveur serait plus adaptée que l'architecture Pair-à-Pair, en particulier compte tenu des contraintes spécifiées.

Justification :

Logique du Jeu Centralisée : La nécessité d'éviter la triche en exécutant la logique du jeu sur des serveurs distants est mieux prise en charge par une architecture client-serveur. Dans un modèle client-serveur, la logique du jeu est centralisée sur le serveur, ce qui rend plus difficile pour les joueurs de manipuler le jeu et tricher.

Contrôle Centralisé : Pour gérer plusieurs joueurs en même temps sur des machines différentes, une architecture client-serveur offre un contrôle centralisé. Le serveur peut gérer l'état global du jeu, la coordination des actions des joueurs, la gestion des connexions, etc.

Gestion des Ressources : Avec un jeu multijoueur en ligne, la gestion des ressources (comme la synchronisation des données du jeu entre les joueurs) est plus facile à gérer de manière centralisée sur le serveur.

Évolutivité : Une architecture client-serveur permet une évolutivité plus aisée en ajoutant des ressources au serveur pour prendre en charge un nombre croissant de joueurs.

Question **10**

Incorrect

Note de 0,00 sur 1,50

Vous faites votre stage d'été dans une *start-up* qui travaille dans le domaine du *machine learning* et l'entrainement de modèles de réseaux de neurones avec des données géo-spatialles.

Voici les besoins de votre projet :

- Les données sont des coordonnées 2D ou 3D numériques simples : { x, y, z} (l'attribut "z" peut ne pas exister sur certaines données) et ne dépendent pas les unes des autres.

- Vous avez besoin d'un accès rapide en lecture et écriture pour vos données.

- Il est possible que plusieurs clients différents aient besoin d'être notifiés lorsque des données sont ajoutées ou modifiées sur la base de données.

En fonction de ces besoins, quel serait le meilleur type de base de données pour votre projet parmi les suivants ?

a. Une base de données relationnelle

b. Une base de données orientée documents

c. Une base de données orientée colonnes ✖

d. Une base de données orientée graphe

e. Une base de données clé-valeur

Votre réponse est incorrecte.

La réponse correcte est :
Une base de données clé-valeur

Question **11**

Correct

Note de 1,50 sur 1,50

Le module fs (*File System*) de NodeJs offre les 2 fonctions suivantes pour lire un fichier présent sur disque :

promises.readFile(fileName) qui lit un fichier de manière asynchrone et retourne une promesse qui contient le contenu du fichier dans un objet *Buffer*.

readFileSync(fileName) qui lit un fichier de manière synchrone et, à la fin de la lecture au complet, retourne le contenu du fichier dans un objet *Buffer*.

En sachant que vous aurez à potentiellement lire des fichiers d'un très grand volume et de traiter des requêtes HTTP qui sont envoyés par plusieurs clients, quelle fonction est à prioriser et pourquoi ?

a. Il n'y a pas de fonction à prioriser : le résultat est pareil dans les deux cas.

b. *readFileSync(fileName)* puisque la lecture synchrone nous garantit d'avoir le contenu complet du fichier.

c. *readFileSync(fileName)* puisque la lecture synchrone ne bloquera pas le reste de l'application.

d. *promises.readFile(fileName)* puisque la lecture asynchrone ne bloquera pas le reste de l'application. ✔

e. *promises.readFile(fileName)* puisque le code asynchrone est toujours plus performant que le code synchrone.

Votre réponse est correcte.

La réponse correcte est :
promises.readFile(fileName) puisque la lecture asynchrone ne bloquera pas le reste de l'application.

L'utilisation de lectures asynchrones est essentielle lorsque l'application doit gérer de nombreuses opérations simultanées. Avec la lecture synchrone, chaque lecture de fichier bloquerait l'exécution, ce qui pourrait rapidement entraîner des retards et une mauvaise réactivité, en particulier dans une application serveur traitant des requêtes concurrentes. Aussi évite le "Callback Hell", l'utilisation de promesses (comme avec promises.readFile) peut rendre le code plus lisible et plus facile à gérer, surtout lorsque vous avez plusieurs opérations asynchrones à enchaîner.

Question **12**

Terminer

Note de 5,75 sur 7,00

Vous travaillez sur un service web de réseaux sociaux pour gérer des utilisateurs et leurs amis (d'autres utilisateurs). Vous commencez le projet par un simple prototype qui vous permet de créer la base de votre service.

Un utilisateur est représenté par un **identifiant unique**, un **nom** et un **prénom**.

Vous retrouverez ci-bas une ébauche d'une partie de ce service web.
Vous devez le modifier pour qu'il respecte au minimum le niveau 2 de Richardson d'un service web REST et qu'il suive les bonnes pratiques de conception de service web REST.

Vous pouvez modifier les routes, la structure des URIs et/ou corps des requêtes et réponses si vous le jugez nécessaire.

Spécifiez clairement les codes de réponse et les valeurs de retour du serveur. Dans le cas de plusieurs codes de réponse, spécifiez dans quel cas chaque code est retourné.

1. Pour obtenir des utilisateurs

```
POST /users
```

Corps de la requête

```
{ limit: <Number> } (limit : X = récupérer les X premiers utilisateurs (optionnel) )
```

Réponse

```
Statut: 200 OK

<Liste d'identifiants d'utilisateurs>
```

2. Pour obtenir un utilisateur particulier

```
POST /users
```

Corps de la requête

```
{ id: <String> }
```

Réponse

```
Statut: 200 OK

{ id: <String>, nom: <String>, prenom: <String> }
```

```
OU

Statut: 500 INTERNAL SERVER ERROR si l'identifiant n'existe pas
```

3. Pour ajouter un utilisateur comme ami

```
POST /friends
```

Corps de la requête

```
{ id1: <String>, id2: <String> }
```

Réponse

```
Statut: 200 OK

(Le corps est vide)
```

4. Pour supprimer un utilisateur comme ami

```
POST /friendsDelete
```

Corps de la requête

```
{ id: <String> }
```

Réponse

```
Statut: 200 OK
(Le corps est vide)
```

*Tout d'abord on peut appliquer deux routage, /friends et /users pour eviter la duplication

1- nous voulons obtenir des utilisateurs, donc la bonne methode à utiliser est GET (non POST). Par la suite, un get ne contient pas de body, donc on doit inserer les parametres dans l'uri pour qu'on puisse faire la requete. ex: Get /users?limit=<Number>. pour le code de retour on retourne 200 OK en cas de succes et on retourne `<Liste d'identifiants d'utilisateurs>`

2- nous voulons obtenir un utilisateur particulier, donc la bonne methode à utiliser est GET(non POST). Par la suite, un get ne contient pas de body, donc on doit changer l'uri pour qu'on puisse faire la requete. ex: GET /users?id=<string>. pour le code de retour on retourne 200 OK en cas de succes, sinon on retourne 404 NOT FOUND `si l'identifiant n'existe pas`

3- nous voulons ajouter un utilsateur comme ami, donc nous devons faire une methode POST et dans notre corps de requête on aura nos deux id `{ id1: <String>, id2: <String> }` qui sont les informations à transmettre, il n'est pas nécéssaire de l'avoir dans l'URI. Pour la reponse le statut doit etre 201 CREATED, et si on voudrait respecter le niveau 3 de richardson, on pourrait retourner la ressource ajoutée

4- nous voulons supprimer un utilisateur comme ami donc nous devons proceder à une methode DELETE (non POST). Cependant, nous aurons donc DELETE /friendsDelete et nous avons une redondance de delete, donc nous pouvons la reecrire de cette façon DELETE /friends. Pour le corps de la requete, il peut etre vide ou nous pouvons mettre l'identifiant de la ressource dans l'URI. Pour le statut de retour, le code de retour est 200 OK ou 202 ACCEPTED (requête acceptée, mais action pas encore effectuée) ,400 BAD REQUEST ou 404 NOT FOUND si la ressource n'existe pas.

Commentaire :

2. Attention : /users?id impliquerait que "id" est optionnel, ce qui n'est pas le cas. Il faudrait plutôt que ça soit /users/:id ou tout autre notation qui rend "id" obligatoire.

3. Manque un code en cas de problème avec la requête : 400 ou 404 pour des ids invalides par exemple

4. Il manque un 2e id pour idéntifier les 2 utilisateurs à qui faut retirer la relation d'amitié

Obtenir des utilisateurs :
Requête :
GET /users?limit=<Number>
L'utilisation de GET est plus appropriée pour récupérer des données.
Réponse :
Statut: 200 OK

```
{
  "users": [
    {"id": "<String>", "nom": "<String>", "prenom": "<String>"},
    {"id": "<String>", "nom": "<String>", "prenom": "<String>"},
    ...
  ]
}
```

Le corps de la réponse contient une liste d'utilisateurs.

Obtenir un utilisateur particulier :
Requête :
GET /users/<id>
Utilisation de GET pour récupérer un utilisateur spécifique.
Réponse :
Statut: 200 OK
{"id": "<String>", "nom": "<String>", "prenom": "<String>"}
Le corps de la réponse contient les détails de l'utilisateur demandé.
Statut: 404 NOT FOUND
Retourné si l'identifiant n'existe pas.

Ajouter un utilisateur comme ami :
Requête :
POST /friendships
Utilisation de POST pour créer une relation d'amitié.
Corps de la requête :
{ "id1": "<String>", "id2": "<String>" }
Les identifiants des deux utilisateurs à lier en amitié.
Réponse :
Statut: 201 CREATED
Le corps de la réponse est vide.

Supprimer un utilisateur comme ami :
Requête :
DELETE /friendships/<id>
Utilisation de DELETE pour supprimer une relation d'amitié.
Réponse :
Statut: 200 OK
Le corps de la réponse est vide.

Question **13**

Terminer

Note de 8,00 sur 8,00

Voici une partie du code d'une application qui permet d'afficher un playlist de chansons en utilisant React. Vous pouvez modifier l'ordre des chansons dans la liste à travers des boutons à côté de chaque chanson ou générer un ordre aléatoire à travers le bouton "Shuffle". Finalement, vous pouvez ajouter une chanson aléatoire du Top 50 du mois à votre liste à travers le bouton "Ajouter Aléatoire". Voici une image de l'interface finale :

Le projet contient 2 Components principaux : **PlayList** et **Song** ainsi que l'utilisation d'un **reducer** et **Context** pour gérer l'état de votre application (un titre et un tableau de chansons).

Vous pouvez assumer que le code fourni est fonctionnel et que les fonctions "*swap*()" et "*getRandomSong*()" ne contiennent pas de défauts.

reducer.js

```
const getRandomSong = () => {...};

const swap = (arr, a, b) => {...};

export default function reducer(state, action) {
    switch (action.type) {
        case ACTIONS.ADD:
            return { songs: [...state.songs, getRandomSong()], title: state.title };
        case ACTIONS.UP:
            if (action.payload > 0) {
                return { songs: swap(state.songs, action.payload, action.payload - 1), title: state.title };
            }
            return state;
        case ACTIONS.DOWN:
            if (action.payload < state.songs.length - 1) {
                return { songs: swap(state.songs, action.payload, action.payload + 1), title: state.title };
            }
            return state;
        case ACTIONS.RENAME:
            return { songs: state.songs, title: action.payload };
        default:
            return state;
    }
}
```

PlayList.jsx

```
1  const PlayList = () => {
2    const { state, dispatch } = useContext(Context);
3
4    const shuffleSongs = () => {
5      const shuffled = state.songs
6        .map((value) => ({ value, sort: Math.random() }))
7        .sort((a, b) => a.sort - b.sort)
8        .map(({ value }) => value);
9      state.songs = shuffled;
10     // ça force le changement, mais on ne sait pas pourquoi ??
11     dispatch({ type: ACTIONS.RENAME, payload: state.title });
12   };
13
14   return (
15     <div id="play-list">
16       <p>Playlist : {state.title}</p>
17       <p>Total : {state.songs.length} chansons</p>
18       {state.songs.map((song, i) => { return <Song song={song} index={i}></Song>;})}
19       <div id="play-list-buttons">
20         <button onClick={shuffleSongs}>Shuffle</button>
21         <button onClick={() => dispatch({ type: ACTIONS.ADD })}>Ajouter Aléatoire</button>
22       </div>
23     </div>
24   );
25 };
```

Song.jsx

```
1  const Song = ({ index, song }) => {
2    const { dispatch } = useContext(Context);
3
4    return (
5      <div className="song">
6        <div id="song-info">
7          <span> #{index + 1} : {song.title} - {song.artist} </span>
8        </div>
9        <div id="song-buttons">
10         <button onClick={() => dispatch({ type: ACTIONS.UP, payload: index })}>▲</button>
11         <button onClick={() => dispatch({ type: ACTIONS.DOWN, payload: index })}>▼</button>
12       </div>
13     </div>
14   );
15 };
```

a) Il y a un problème avec l'ajout des components **Song** dans **PlayList.jsx** . Quel est ce problème et quel est son impact ? (2 points)

b) La fonction *shuffleSong()* dans **PlayList.jsx** est problématique au niveau architectural. Pourquoi ? Comment peut-on régler ce problème ? (3 points)

c) Un de vos amis propose de retirer l'utilisation du **reducer** et du **Context** et simplement implémenter la logique de gestion dans le component parent de **PlayList** (App.jsx) et passer des fonctions et des objets à **PlayList** et **Song** à travers leurs propriétés. Ceci évitera de créer des fichiers supplémentaires autres que les components. Est-ce que vous êtes d'accord avec sa proposition ? Si oui, pourquoi ? Sinon, quel est le problème potentiel avec cette approche ? (4 points)

A) il y a un problème avec l'ajout des components Song dans playList.jsx car les différentes song n'ont pas de key différents. Le DOM Virtuel a besoin d'un identifiant unique dans le cas d'une itération sinon on aura une erreur lors de l'exécution. On utilise la propriété key sur l'élément HTML généré

b) La fonction *shuffleSong()* dans PlayList.jsx est problématique au niveau architectural, car il s'agit d'une action et ne respecte pas l'architecture de notre reducer La solution on pourra ajouter un case dans notre reducer.js qui s'appellerait ACTION.SHUFFLESONG et dans la ligne 20 du fichier playlist.jsx on mettrait ceci: `<button onClick={() => dispatch({ type: ACTION.SHUFFLESONG })>`

c) je ne suis pas d'accord, car sans reducer et du context, l'approche top-down nécessite à ce que la composante parent donne toutes les informations nécessaire à ses enfants qui ne peuvent pas remonter dans l'arbre. De plus, L'arborescence est rigide et difficile à modifier. C'est pour cela que l'on utilise Context, car elle permet une approche bottom-up
où les composantes peuvent remonter dans l'arborescence pour les données. Ceci induit un couplage dans les composantes et peut nuire à la réutilisation et Garde le contexte bas et accessible que par les composantes qui en ont besoin. Pour le Reducer, il prend un State et une Action et retourne un nouveau State. Donc certes, il y a plus de fichiers, mais cela garantira une meilleur gestion des etats

Commentaire :

a) ok

b) OK

c) OK

Question **14**

Terminer

Note de 6,00 sur 6,00

Vous travaillez sur un service web de gestion de chansons, albums et playlists en utilisant la librairie Express.

Voici un extrait de votre service web qui traite la gestion des playlists de votre système. Votre code valide si une requête est autorisée d'y accéder ou non et affiche l'heure de fin de traitement des requêtes.

Vous pouvez assumer que les fonctions *validateHeader()* et celles de l'objet *PlayListService* sont bien implémentées :

```
1  app.use("/playlist", (req, res, next) => {
2    const authHeader = req.get("Authorization");
3    if (validateHeader(authHeader)) { next(); }
4    else { res.sendStatus(401).end(); }
5  });
6
7  app.get("/playlist/:id", async (req, res, next) => {
8    const playlist = await PlayListService.getPlaylist(req.params.id);
9    if (playlist) res.send(playlist);
10   else {
11     res.status(404);
12     next();
13   }
14 });
15
16 app.post("/playlist", async (req, res, next) => {
17   const playlist = req.body;
18   if (playlist) { await PlayListService.createPlayList(playlist); }
19   res.sendStatus(playlist ? 201 : 400);
20   next();
21 });
22
23 app.use("/playlist", (req, res, next) => {
24   const time = `Terminé à ${new Date().toLocaleTimeString()}`;
25   console.log(time);
26   res.send(time);
27 });
```

Questions à répondre :

a) Le middleware des lignes 1 à 5 vérifie si la requête HTTP est acceptée ou non par le reste du service. Qu'est-ce que la requête HTTP doit contenir pour pouvoir être acceptée par le middleware ? (1 point)

b) Vous envoyez la requête **GET /playlist/123** qui vous retourne la playlist avec id "123", mais aucun message n'est affiché dans la console du serveur. Vous envoyez la requête **GET /playlist/12** qui n'est pas un id valide, mais le serveur affiche l'heure de fin de traitement dans sa console. Pourquoi ? Quel est le code de retour et le corps de la réponse HTTP pour un id invalide ? (2 points)

c) Vous envoyez la requête **POST /playlist** valide avec une nouvelle playlist dans le corps de la requête. Le serveur affiche l'heure de traitement dans la console, mais lance également une erreur dans la console. Pourquoi ? (2 points)

d) Que pouvez-vous faire pour éviter de dupliquer "**/playlist**" dans la définition des URIs gérés par vous routes ? (1 point)

a) la requete HTTP doit contenir l'"Autorization" pour pouvoir être acceptée par le middleware

b) lorsqu'on a envoyé la premiere requete GET/playlist/123, nous allons dans le if(ligne 8) car on a bel et bien une playlist. Aucun message n'est affiché dans la console du serveur (on a pas fait console.log(playlist)), mais on retourne bel et bien la playlist en question en faisant res.send(playlist) et on retourne le statut 200 OK. Pour la deuxieme requete GET/playlist/12, nous allons dans le else(ligne 10) car la variable playlist n'existe pas donc nous avons un code de retour 404 NOT FOUND.

Pourquoi on affiche l'heure dans la duexieme requete et pas la premiere? car dans le if(requete 1) nous n'avons pas de next() qui nous permettrait d'aller au prochain middleware, tandis que pour la requete 2, dans le else il y a un next() donc on passe au prochain middleware qui est `app.use("/playlist", (req, res, next)` et c'est à ce moment que l'on retourne en console l'heure de fin de traitement. Le code de retour pour un id invalide est 404 NOT FOUND et son corps est `Terminé à ${new Date().toLocaleTimeString()}`;

c) Lorsqu'on envoie la requête POST /playlist valide, on va dans le if de la ligne 18, on l'ajoute puis, on retourne le statut 201 CREATED, et on execute next() qui nous envoie à notre prochain middleware `app.use("/playlist", (req, res, next). A ce moment la on affiche l'heure dans la console, mais on affiche une erreur car nous essayons de retourner deux codes de retour. dans la methode post on retourne 201 CREATED, et dans la methode use, on retourne 200 OK. C'est pas possible de retourner plusieurs code de retour à la fois.`

d) Pour eviter de dupliquer "/playlist", on peut utiliser le routage donc on regroupe plusieurs routes qui traitent du même domaine(/playlist) et séparer les différentes routes dans des fichiers différents allège le code et permet de mieux diviser la logique interne du serveur

Commentaire :
a) "Authorization" est un en-tête spécifiquement -0.5

b) OK

c) send ajoute 200 seulement si aucun autre code n'a déjà été configuré. OK pour l'erreur étant le renvoi de la réponse 2 fois

d) OK

Question **15**

Terminer

Note de 2,00 sur 4,00

a) Dans le cadre de votre TP5, vous n'aviez besoin que du nom, image et temps d'une recette pour l'afficher dans la liste des recettes dans la composante *RecipeCard* de votre site web. Votre partenaire de TP propose alors d'utiliser la projection suivante pour obtenir que les informations nécessaires :

const projection = { projection: { _id: 1, name: 1, time: 1, img: 1, category: 0, ingredients: 0, tools: 0, steps: 0 } };

Malheureusement, cette projection ne fonctionne pas. Expliquez **pourquoi** et donnez une solution possible pour une projection correcte. (2 points)

b) L'architecture de votre TP5 est une **architecture orientée services** avec votre serveur dynamique donnant accès aux services *ContactsService* et *RecipesService*. Qu'est-ce que ça prendrait pour transformer cette architecture en une **architecture de microservices** ? Vous pouvez proposer tous les changements que vous jugez nécessaires au projet. (2 points)

a) Cela ne marche pas, car notre RecipeCard ne reconnait pas l'id de la recette, on doit vraiment lui retourner que le nom, image et temps d'une recette. solution: const projection = { projection: { name: 1, time: 1, img: 1} };

b) une architecture de microservice est un systeme dont les différentes fonctionnalités sont séparées en service qui ont des taches encore plus spécifique que ceux de l'architecture orientée service, et sont donc fortement découplés des autres composantes du système. donc il s'agirait de faire des divisions très précise des responsabilités aux différents services disponibles comme ContactsService et RecipesService.

Commentaire :
a) _id peut être ignoré sans problèmes. Le problème est qu'on ne peut pas mélanger inclusion et exclusion (1 et 0) -1

b) Cette division est excessive : les services sont déjà assez précis. Il faudrait mettre chaque service sur son serveur HTTP séparé -1

◄ LOG2440 - Automne 2022 - Contrôle Pratique

Aller à...

Introduction ►