Commencé le	mardi 20 décembre 2022, 13:30
État	Terminé
Terminé le	mardi 20 décembre 2022, 15:21
Temps mis	1 heure 50 min
Note	**36,00** sur 40,00 (**90**%)

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.

En cas de ralentissement de votre ordinateur, fermez complètement Chrome et rouvrez l'instance.

Voici les règles pour l'évaluation:

- L'examen est noté 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, bon temps des fêtes et bonnes vacances!

Question **1**

Terminé

Non noté

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

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

C'est quoi le STEP ?

Question **2**

Partiellement correct

Note de 1,00 sur 1,50

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

- [ ] a. La gestion des erreurs est optionnelle pour la syntaxe **async/await**
- [x] b. Le mot clé **await** rend une fonction asynchrone et elle est toujours exécutée sur un fil d'exécution séparé du fil de la boucle d'événements. ✔
- [x] c. La syntaxe **async/await** garantit qu'une fonction asynchrone va toujours retourner une Promesse résolue. ✔
- [x] d. Il est possible de bloquer la boucle d'événements avec le code exécuté dans une fonction **.then()** sur une Promesse. ✖
- [ ] e. La valeur de retour de cette fonction est une Promise : `async function foo(a) { return undefined; }`

Votre réponse est partiellement correcte.

Vous avez sélectionné trop d'options.

Question **3**

Correct

Note de 1,50 sur 1,50

Parmi les méthodes HTTP suivantes, sémantiquement, laquelle ou lesquelles peuvent modifier l'état du serveur ?

- [x] a. DELETE ✔
- [x] b. POST ✔
- [ ] c. HEAD
- [x] d. PUT ✔
- [x] e. PATCH ✔
- [ ] f. GET
- [ ] g. OPTIONS

Votre réponse est correcte.

Question **4**

Correct

Note de 1,50 sur 1,50

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.

Question **5**

Correct

Note de 1,50 sur 1,50

Il est actuellement 13h00 le 20 décembre 2022. Vous envoyez la requête suivante :

```
GET /programmes 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=500
Last-Modified: Tue, 20 Dec 2022 13:00:00 GMT
```

max-age=500; 500 secondes/60 = 8.33 min; donc à 13h10 on doit faire un get mais à 13h05 pas besoin la ressource est encore "fresh"

Vous envoyez à nouveau la même requête à 13h10.

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 13h05.

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

Le navigateur utilise la ressource dans sa cache ✔

Votre réponse est correcte.

Question **6**

Correct

Note de 1,50 sur 1,50

Vous envoyez une requête à un serveur pour récupérer l'information ayant l'id "abc" sur la route /data/:id avec la requête suivante :

```
fetch("https://myWebAPI.com/data/abc")
  .then((response) => response.json())
  .then((json) => console.log(json))
  .catch((e) => console.log("Une erreur est survenue"));
```

Dans quelle(s) situation(s) est-ce qu'on aura le message "Une erreur est survenue" dans la console ?

- [ ] a. Cette requête affiche toujours le message d'erreur puisque l'URI de la requête ne correspond pas à la route du serveur
- [ ] b. Cette requête n'affiche jamais le message d'erreur puisque fetch ne lance pas d'erreurs, contrairement à XHR
- [ ] c. Le serveur ne contient pas de données avec l'id "abc" et retourne le code d'erreur 404
- [x] d. Le client n'a pas de connexion internet ✔
- [ ] e. Le serveur a rencontré une erreur interne et retourne le code 500

Votre réponse est correcte.

Question **7**

Incorrect

Note de 0,00 sur 2,00

Voici l'implémentation de la gestion de la route "/search/:keyword" de recherche par mot clé dans une application Express. La fonction "search" retourne l'objet trouvé en fonction de la valeur reçue en paramètre ou *undefined* si rien n'a été trouvé.

```
app.get("/search/:keyword", (req, res, next) => {
  const result = search(req.params.keyword);
  if (!result) {
    res.status(404).send();
  }
  res.set("Content-Type", "application/json").send(result);
});
```

Vous faites la requête suivante : **GET /search/abc** au serveur. Sachant que "abc" ne fait pas partie des données sur le serveur, qu'est-ce qui va se passer ?

- [ ] a. Le serveur va lancer une erreur.
- [ ] b. Le client reçoit une réponse HTTP avec le code 404 et une réponse HTTP avec le code 200.
- [x] c. Le client reçoit une réponse HTTP avec le code 404 et aucun corps.
- [ ] d. Le client va lancer une erreur.
- [x] e. Le client reçoit une réponse HTTP avec le code 404 et la valeur "undefined" dans le corps. ✖
- [ ] f. Le client reçoit une réponse HTTP avec le code 200 et la valeur "undefined" dans le corps.
- [ ] g. Le client reçoit une réponse HTTP avec le code 200 et un objet vide dans le corps.

La fonction va retourner undefined mais rien n'implique que ça serait dans le corps

Votre réponse est incorrecte.

Question **8**

Correct

Note de 1,50 sur 1,50

Il est souvent recommandé d'utiliser la commande **npm ci** au lieu de **npm install** si un fichier **package-lock.json** est disponible pour un projet web. Pourquoi ?

- a. **npm ci** s'assure d'installer les versions exactes des dépendances du projet et toujours reproduire le même environnement, ce qui n'est pas garanti avec npm install. ✔
- b. **npm ci** veut dire **C**onsole **I**nstall et est simplement un alias de **npm install**.
- c. C'est faux, **npm install** doit toujours être priorisé par rapport à **npm ci**.
- d. Il n'y a pas de différence entre les commandes et les 2 peuvent être utilisées de manière interchangeable.

Votre réponse est correcte.

Question **9**

Partiellement correct

Note de 1,00 sur 1,50

Soit une collection MongoDB qui contient des documents JSON avec ces champs (tous de type *string*):

```
{
    "LOG2440": "",
    "polymtl": "",
    "D_pour_diplome": "",
    "C_pour_cool": ""
}
```

Et soit le code suivant:

```
1 const mongo_response = await collection.find({LOG2440: "cool"}).toArray();
2 const important_properties = mongo_response.map((item) => return {item.polymtl, item.D_pour_diplome};);
3 console.log(important_properties);
```

Sélectionner la ou les réponses qui affichent la même chose que le console.log de la ligne 3.

- ☑ a. ✖
```
1 const mongo_response = await collection.find({LOG2440: "cool"}, {projection: { polymtl: 1, D_pour_diplome:
2 1}}).toArray();
  console.log(mongo_response);
```
- ☐ b.
```
1 const mongo_response = await collection.find({LOG2440: "cool"}).limit(2);
2 console.log(mongo_response);
```
- ☐ c.
```
1 const mongo_response = await collection.find({LOG2440: "cool"}, {projection: { LOG2440: 0, C_pour_cool:
2 0}}).toArray();
  console.log(mongo_response);
```
- ☑ d. ✔
```
1 const mongo_response = await collection.find({LOG2440: "cool"}, {projection: { _id:0, polymtl: 1,
2 D_pour_diplome: 1}}).toArray();
  console.log(mongo_response);
```

Votre réponse est partiellement correcte.

Vous avez sélectionné trop d'options.

Question **10**

Correct

Note de 1,50 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 géospatiales.

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 orientée graphe

b. Une base de données relationnelle

c. Une base de données orientée documents

d. Une base de données orientée colonnes

e. Une base de données clé-valeur ✔

Votre réponse est correcte.

Question **11**

Correct

Note de 1,50 sur 1,50

Un Reducer basé le patron SAM, applique le principe *Single Source of Truth*. Quelle(s) réponse(s) en lien avec ce principe est(sont) vraie(s) ?

a. Un *Reducer* applique le principe *Single Source of Truth* ce qui signifie que l'état d'une application est sauvegardé dans une place commune, accessible par tous ceux qui ont accès à cette place. ✔

b. Un *Reducer* applique le principe *Single Source of Truth* ce qui signifie que chaque composante sauvegarde l'état global d'une application dans une place commune.

c. Un *Reducer* applique le principe *Single Source of Truth* ce qui signifie que la composante au plus haut niveau contienne tous les états utilisés à travers l'application. Ils sont alors passés en propriété (props) aux enfants.

d. Le patron *SAM* et le *Reducer* n'appliquent pas le principe *Single Source of Truth*.

e. Un *Reducer* applique le principe *Single Source of Truth* ce qui signifie que seulement une seule composante à la fois peut modifier l'état de l'application.

Votre réponse est correcte.

Question **12**

Correct

Note de 1,50 sur 1,50

Vous devez développer un site web de commerce électronique. Voici vos requis :

- Votre équipe se situe un peu partout dans le monde, donc vous souhaitez que chaque division s'occupe d'une partie du site web.
- Si une division ne fait pas son travail, le site web n'est pas en péril.
- On souhaite avoir une séparation claire entre la composante qui gère l'interface graphique et les composantes qui gèrent les données.

a. Architecture client-serveur

b. Architecture pair-à-pair

c. Architecture pipeline

d. Architecture orientée événements

e. Architecture trois niveaux

f. Architecture orientée services ✔

Votre réponse est correcte.

SOA permet une distribution du travail entre différentes équipes, chacune gérant un service spécifique. Chaque service expose une interface claire, facilitant la séparation entre la composante qui gère l'interface graphique et les composantes qui gèrent les données. De plus, si une division ne remplit pas son rôle, les autres services peuvent continuer à fonctionner, minimisant ainsi l'impact sur l'ensemble du site web.

Question **13**

Terminé

Note de 6,00 sur 6,00

a) Voici l'implémentation d'un component qui gère un état quelconque sous la forme d'un tableau (variable state). Quel est le danger potentiel d'assigner setState à la propriété elementModifier du component ChildComponent à la ligne 17 ? Pourquoi est-ce que l'utilisation de addElement à la ligne 18 est-elle à préférer ? Assumez que le but de ChildComponent est de modifier l'état de l'application. (2 points)

```
1  const App = () => {
2    const [state, setState] = useState([]);
3
4    function deleteElement(selectedElement) {
5      setState(state.filter((x) => x.id !== selectedElement.id));
6    }
7
8    function addElement(newElement) {
9      setState([...state].push(newElement));
10   }
11
12   function replaceElement(oldElement, newElement) {
13     setState(state.map((x) => (x.id !== oldElement.id ? x : newElement)));
14   }
15   return (
16     <>
17       <ChildComponent elementModifier={setState} />
18       <ChildComponent elementModifier={addElement} />
19     </>
20   );
21 };
```

b) Une fois le code JSX transpilé en HTML et JS, quelle est la différence entre l'élément HTML parent du component App et l'élément parent des components ChildComponent des lignes 17, 18 ? (1 point)

c) Sachant que ChildComponent possède plusieurs autres descendants (components React) directs et indirects qui ont besoin d'accéder au tableau *state*, comment est-ce que vous changeriez la manière de gérer l'état ? Donnez tous les éléments que vous considérez nécessaires pour vos changements. (3 points)

a) Le danger d'assigner setState à l'enfant est que celui pourra alors faire n'importe quelle modification à la variable state, sans aucun contrôle possible de la part du parent. Au contraire, le fait d'utiliser addElement permet de limiter les changements, en n'autorisant ici que l'ajout de nouveaux éléments et pas la suppression par exemple. Cela permet de s'assurer au niveau du parent qu'on sait quelles sont les modifications possibles de notre variable state (ici l'ajout et le remplacement).

b) Lors de la transpilation, Le component <> sera supprimé car son rôle est uniquement de servir de parent aux deux composants enfants car on ne peut pas retourner deux components, il n'a pas d'équivalent réel dans le DOM HTML. L'élément parent des deux ChildComponent sera donc le même que le parent du component App.

c) Si beaucoup de components doivent utiliser et modifier un state qui est défini plus haut, il devient avantageux d'utiliser un Reducer comme Redux pour centraliser la gestion du state. On pourra alors dans le Reducer définir le state initial et toutes les actions possibles ainsi que la logique de modification du state en fonction de chacune des actions.

Ensuite, on pourra faire `const dispatch = useDispatch();` dans les components qui doivent modifier le state pour que ceux-ci puissent envoyer des actions au reducer. Celui-ci appliquera les modifications selon la logique établie, puis les components qui ont besoin d'accéder au state pourront utiliser `useSlice()` pour accéder au subset de l'état dont ils ont besoin.

Commentaire :
a) ok

b) ok

c) Ok. Tenez vous à la matière vu dans le cours. L'utilisation de Redux serait une bonne réponse, mais pas nécessaire vu que le même fonctionnement peut être atteint juste avec React. La réponse manque certains éléments pour bien expliquer comment utiliser useDispatch même avec Redux. Vous avez quand même besoin d'avoir un store comme parent du component qui utilise le hook

Question **14**

Terminé

Note de 5,00 sur 6,00

Les questions suivantes portent sur votre Travail Pratique #5 et le système que vous avez mis en place.

a) Vous utilisez au moins 3 serveurs différents dans votre système. Expliquez quels sont ces serveurs et expliquez brièvement le rôle de chacun dans votre système. (4 points)

b) Les components de page Playlist et CreatePlaylist récupèrent les informations de la playlist du serveur à travers "getPlaylistById" à chaque fois qu'ils sont chargés . Sachant que ces components partagent le même environnement JS, est-ce que ces appels différents au serveur sont nécessaires ? **Justifiez**. (2 points)

a) Les trois serveurs que nous utilisons sont le serveur backend, le serveur frontend et le serveur de base de données.

Le serveur backend expose une API HTTP REST qui permet d'accéder, de créer, de modifier et de supprimer des playlists et des chansons, en exposant différentes routes comme `/playlists` ou `/songs`. Il y a ensuite une logique qui va valider les données envoyées, faire la modification ou la recherche en se connectant à la base de données, puis retourner le résultat au client.

Le serveur frontend, qui pourrait être un serveur HTTP statique, donne au navigateur web le corps HTML de base, les feuilles de style, ainsi que toute la logique d'application côté client, en utilisant notamment React. La logique côté client va ensuite se connecter au serveur backend à l'aide de différentes requêtes AJAX pour récupérer les informations sur les chansons et les playlists et faire les actions demandées par l'utilisateur (création, modification, suppression...).

Le serveur de base de données, qui est un serveur MongoDB (malheureusement, parce que le NoSQL c'est moche), permet de persister les playlists et les chansons sur le long terme. Le serveur backend s'y connecte lorsqu'il a besoin d'obtenir de l'information ou de faire des modifications, et le serveur de base de données enregistre ces informations dans des fichiers sur son disque pour ne pas les perdre si le serveur est redémarré. On peut donc avoir des données qui continuent d'exister à travers les redémarrages des deux autres serveurs.

En terme de MVC, la base de données gère le modèle, le serveur backend gère la logique serveur, et le serveur frontend gère la logique client et la vue.

b) Dans l'état actuel du TP5, il est nécessaire de faire deux appels à getPlaylistById car les deux pages ne peuvent pas facilement se partager des informations, aucune n'est parent de l'autre et on ne peut donc pas utiliser les props pour que l'une partage les données avec l'autre. Cependant, en utilisant des Reducers, on pourrait avoir une liste globale des playlists qui serait gérée par Redux, et les pages Playlist et CreatePlaylist pourraient juste récupérer la playlist dont ils ont besoin dans le state global. On aurait alors un seul appel à l'API qui récupèrerait toutes les playlists puis on réutiliserait le résultat dès que nécessaire (avec idéalement une vérification régulière des changements).

Commentaire :
b) Les 2 partagent le même espace JS donc peuvent utiliser un Context pour se partager l'information. "(avec idéalement une vérification régulière des changements)" : il faut le faire puisqu'un autre utilisateur peut modifier une playlist sur le serveur et utiliser la copie locale lorsqu'on essaie de la modifier (de Playlist à CreatePlaylist), on peut se retrouver avec un état désynchronisé
-1

Question **15**

Terminé

Note de 6,00 sur 6,00

Voici un extrait du code de votre serveur qui offre une API de gestions de réservations. Vous pouvez créer une réservation ainsi qu'obtenir des informations ou annuler une réservation existante. Considérez que les fonctions "validateLogin" et "validateHeader" sont correctement implémentées.

```
1  const app = express();
2  app.use(express.json());
3
4  app.get("/login/:username/:password?", (req, res) => {
5    const user = req.params.username;
6    const password = req.params.password;
7    res.send(validateLogin(user, password));
8  });
9
10 app.post("/reservation/:id", (req, res) => {
11   const authHeader = req.get("X-AUTH");
12   if (validateHeader(authHeader)) {
13     res.send(reservationService.getReservation(req.params.id));
14   } else {
15     res.sendStatus(401);
16   }
17 });
18
19 app.post("/reservation/create", (req, res) => {
20   const authHeader = req.get("X-AUTH");
21   if (validateHeader(authHeader)) {
22     res.send(reservationService.createReservation(req.body));
23   } else {
24     res.sendStatus(401);
25   }
26 });
27
28 app.post("/reservation/cancel/:id", (req, res) => {
29   const authHeader = req.get("X-AUTH");
30   if (validateHeader(authHeader)) {
31     res.send(reservationService.cancelReservation(req.params.id));
32   } else {
33     res.sendStatus(401);
34   }
35 });
```

Afin de pouvoir utiliser l'API, un utilisateur doit se connecter avec un nom d'utilisateur et mot de passe. Si la connexion est réussie, le client reçoit un identifiant unique qu'il doit passer dans l'en-tête "X-AUTH" dans ses requêtes pour être correctement identifié.

a) Présentez au moins 2 problèmes avec la gestion de la connexion des lignes 4 à 8 et donnez une solution possible. Considérez que la fonction "validateLogin" est correctement implémentée et retourne un identifiant unique ou la chaine vide ("") si les attributs sont invalides. (2 points)

b) Quelles sont les modifications à faire au code présenté pour avoir une API REST de maturité Niveau 2 ? Considérez que les méthodes de l'objet *reservationService* sont bien implémentées et les méthodes *createReservation* et *cancelReservation* retournent un booléen qui indique si l'opération a réussi ou non. (3 points)

c) Comment pouvez-vous réduire la duplication de code présente aux lignes 10 à 35 ? (1 point)

a) Un premier problème est que si le nom d'utilisateur ou le mot de passe contiennent des caractères spéciaux, comme '/', '&' ou '?', ils pourraient être interprétés comme de la syntaxe de requête HTTP et req.params ne contiendra alors pas les bonnes valeurs. Un autre problème est le fait de passer le mot de passe en paramètre d'URL, car cela signifie que le mot de passe sera affiché en clair dans l'historique du navigateur du client ainsi que dans les logs du serveur HTTP dans beaucoup de cas.

Une solution serait d'utiliser une requête POST avec un request body qui contiendrait le nom d'utilisateur et le mot de passe encodés en JSON. Il est en plus avantageux de faire une requête POST car le fait de se connecter modifie l'état du serveur, en ajoutant l'identifiant unique à la liste des identifiants autorisés, on ne devrait donc pas utiliser de requête GET dans ce cas.

b) Pour commencer, il faut utiliser les bonnes méthodes HTTP avec des noms qui ne sont pas redondants. On passerait donc de

- GET /login/username/password
- POST /reservation/1234
- POST /reservation/create

- POST /reservation/cancel/1234

à

- POST /login (request body contains username and password)
- GET /reservation/1234
- POST /reservation
- DELETE /reservation/1234

Il faudrait aussi envoyer des codes d'erreurs quand les opérations demandées n'ont pas réussi. Il faudrait que /login renvoie 400 quand le username ou le mot de passe est incorrect, que le GET renvoie 404 si la reservation n'existe pas (si getReservation renvoie null par exemple), que le POST renvoie une erreur en fonction du problème qui a pu avoir lieu, par exemple 400 s'il manque un champ dans l'objet envoyé, et enfin que le DELETE renvoie 404 si l'utilisateur n'existe pas, ou par exemple 403 si l'utilisateur n'a pas l'autorité pour annuler une réservation qui n'est pas à lui. Le POST pourrait aussi renvoyer 201 quand une réservation a été crée avec succès plutôt que juste 200.

c) Pour réduire la duplication de code liée à l'authentification de l'utilisateur avec le header X-AUTH, on pourrait utiliser un middleware qui s'appliquerait à toutes les requêtes qui commencent par /reservation et qui vérifierait que l'utilisateur est authentifié, et répondrait prématurément à la requête s'il ne l'est pas.

Commentaire :

Question **16**

Terminé

Note de 5,00 sur 5,00

a) Les méthodes de manipulation d'une collection de MongoDB dans le *driver* de NodeJS retournent toutes une *Promise*. Pourquoi ? (3 points)

b) Vous voulez utiliser le système du TP5 avec un très grand nombre de playlists et chansons. Les chansons d'une playlist sont toujours référées à travers leur attribut "id".
La collection "song" qui contient les informations de vos chansons est maintenant partitionnée sur plusieurs instances de MongoDB avec l'attribut "id" comme clé de partitionnement. Quelle stratégie de partitionnement allez-vous utiliser et pourquoi ? (2 points)

a) Toutes ces méthodes renvoient une Promise car les requêtes à MongoDB consistent en l'échange d'informations sur le réseau entre le serveur web et le serveur MongoDB, et peuvent mettre un certain délai avant que le résultat ne soit disponible. On préfère donc que la fonction soit asynchrone pour que l'appel ne soit pas bloquant et qu'il soit possible de faire d'autres actions en attendant d'avoir le résultat de l'appel à la base de données.

b) On va préférer un partitionnement par hachage car celui-ci permet en moyenne d'avoir une répartition équitable et uniforme des chansons entre les différentes instances contrairement au partitionnement par intervalle qui pourrait être inégal. De plus, rien n'indique que deux chansons qui ont des id "proches" devraient être sur la même instance, deux chansons de la même playlist pourraient très bien avoir des id très éloignés. Il n'y a donc pas de raison d'utiliser un partitionnement par intervalle.

Commentaire :
a) 3

b) 2

Aller à...

Introduction ►