| | |
|---|---|
| **Commencé le** | vendredi 10 décembre 2021 |
| **État** | Terminé |
| **Terminé le** | vendredi 10 décembre 2021 |
| **Temps mis** | |
| **Note** | **27,75** sur 40,00 (**69**%) |

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 **16** 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 obligé 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**

Terminer

Non noté

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

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

ok

Question **2**

Partiellement correct

Note de 0,50 sur 1,00

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

- [ ] a. L'API Fetch supporte nativement les Promesses JavaScript.
- [ ] b. Contrairement à Fetch, XMLHttpRequest (XHR) lancera une exception si le code de retour est un code 400-499 ou 500-599.
- [x] c. Il n'est pas possible d'annuler une requête HTTP faite avec Fetch. ✔
- [x] d. L'API Fetch ne supporte pas nativement les Promesses JavaScript. ✔
- [x] e. Il n'est pas possible d'annuler une requête HTTP faite avec XMLHttpRequest (XHR). ✖
- [x] f. L'utilisation de XMLHttpRequest (XHR) est limitée puisque l'objet supporte seulement des documents XML dans le corps des requêtes et les réponses HTTP. ✔
- [ ] g. L'API Fetch est une version plus moderne de faire des requêtes HTTP que XMLHttpRequest (XHR) et qui est supporté par NodeJS.

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 3.
Les réponses correctes sont :
L'API Fetch est une version plus moderne de faire des requêtes HTTP que XMLHttpRequest (XHR) et qui est supporté 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 **3**

Partiellement correct

Note de 0,50 sur 1,00

Il est actuellement 10h le 11 décembre 2021. 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: no-cache, private, max-age=600
Last-Modified: Sat, 11 Dec 2021 10:00:00 GMT
```

max-age=600; 600 secondes/60 = 10 min; !!MAIS!! no-cache est plus important, cette directive nous dis que le max-age n'est pas important, peut-importe l'heure il faut un get conditionnel

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 ?

La requête est à nouveau acheminée au serveur qui renvoie à nouveau la ressource ✖

La directive no-cache réponse indique que la réponse peut être stockée dans des caches, mais la réponse doit être validée auprès du serveur d'origine avant chaque réutilisation, même lorsque le cache est déconnecté du serveur d'origine.

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 1.
La réponse correcte est :
Il est actuellement 10h le 11 décembre 2021. 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: no-cache, private, max-age=600
Last-Modified: Sat, 11 Dec 2021 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]

Question **4**

Incorrect

Note de 0,00 sur 1,00

La ressource demandée par la requête HTTP suivante sera-t-elle mise en cache ? Pourquoi ?

```
POST www.exemple.com HTTP/1.1
If-Modified-Since: Sat, 11 Dec 2021 12:00:00 GMT
```

- a. Non, puisque l'URI de la requête est mal formaté. ✖
- b. Non, puisqu'il manque l'en-tête *Cache-Control* .
- c. Oui, puisque la requête utilise la version 1.1 du protocole HTTP.
- d. Oui, puisque la requête utilise la version 1.1 du protocole HTTP.
- e. Oui, puisque l'en-tête *If-Modified-Since* est présent.
- f. Non, puisque la requête utilise le verbe POST.

Votre réponse est incorrecte.

La réponse correcte est :
Non, puisque la requête utilise le verbe POST.

Méthode HTTP (Verbe) : La méthode HTTP spécifiée dans la requête est POST. La méthode POST est généralement utilisée pour soumettre des données à traiter à une ressource spécifiée. Contrairement aux méthodes sécurisées (comme GET), POST est considéré comme non idempotent, ce qui signifie qu'il n'est pas garanti que le résultat de la requête soit le même pour des requêtes identiques.

Mise en cache avec POST: généralement, les requêtes POST ne sont pas mises en cache par défaut. La mise en cache est plus courante avec des méthodes sûres comme GET. En effet, les requêtes POST impliquent souvent la soumission de données spécifiques à la requête et la mise en cache de la réponse peut ne pas être appropriée.

Question **5**

Incorrect

Note de 0,00 sur 1,00

Vous devez développer un logiciel qui doit implémenter les fonctionnalités suivantes :

- Il prend en entrée un fichier de données CSV (Comma-separated values), contenant des chiffres séparés par des virgules,
- Le logiciel doit d'abord vérifier que le fichier fourni est valide, selon un certain nombre de règles définies par les développeurs,
- Le logiciel doit ensuite transformer le fichier en commandes SQL (Structured Query Language),
- Le logiciel doit ensuite exécuter les commandes SQL, ce qui ajoute les chiffres fournis initialement dans une base de données relationnelle à distance.

Pour ce logiciel, l'architecture [Trois niveaux] ✖ serait la plus appropriée.

Voici la raison qui explique pourquoi cette architecture serait la plus appropriée :

[Parce que c'est l'architecture la plus sécuritaire] ✖

L'architecture Pipeline serait la plus appropriée pour ce logiciel en raison de sa capacité à diviser le processus en plusieurs étapes distinctes et à traiter chaque étape de manière séquentielle. Les étapes du pipeline peuvent être conçues de manière à garantir la modularité, la réutilisabilité et la facilité de maintenance du code

Question **6**

Correct

Note de 1,00 sur 1,00

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 pipeline

b. Architecture trois niveaux

c. Architecture client-serveur

d. Architecture pair-à-pair

e. Architecture orientée services ✔

f. Architecture orientée événements

Votre réponse est correcte.

La réponse correcte est :
Architecture orientée services

Question **7**

Correct

Note de 1,00 sur 1,00

Quel mécanisme est utilisé pour autoriser un serveur ou une page web d'accéder à des ressources d'un domaine à un autre ?

a. CORS Mécanisme est utilisé pour autoriser un serveur ou une page web d'accéder à des ressources d'un domaine à un autre. ✔

b. MIddleware Logiciel intermédiaire qui agit comme une couche d'assistance entre différentes applications pour faciliter la communication et la gestion des données.

c. Origin L'origine d'une requête HTTP, indiquant le domaine à partir duquel provient la demande.

d. HTTP Protocole de transfert hypertexte utilisé pour la communication sur le World Wide Web, définissant la manière dont les messages sont formatés et transmis.

e. Cookie Petit fichier de données stocké sur l'ordinateur d'un utilisateur, contenant des informations telles que les préférences du site web, utilisé par les navigateurs pour suivre l'activité en ligne.

Votre réponse est correcte.

La réponse correcte est :
CORS

Question **8**

Partiellement correct

Note de 0,50 sur 1,00

Soit la gestion des routes en Node/Express suivante :

```
const app = express()

app.get('/:id', (req, res, next) => {
  if (req.params.id === 'blog') {
    res.send('Blog')
  }
```

```
  res.set('Content-Type','text/plain')
  res.send('Autre')
})
```

Que se passe-t-il si l'utilisateur navigue sur l'URI **/blog** ?

- [ ] a. La valeur "Autre" est affichée sur le navigateur
- [x] b. La valeur "Blog" est affichée sur le navigateur ✔
- [ ] c. Erreur d'exécution du côté client
- [ ] d. Erreur d'exécution du côté serveur
- [ ] e. La page est vide

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 1.
Les réponses correctes sont :
Erreur d'exécution du côté serveur,

La valeur "Blog" est affichée sur le navigateur

Lorsque l'utilisateur navigue sur l'URI /blog, la valeur "Blog" est affichée sur le navigateur. C'est parce que dans le gestionnaire de route, il y a une condition qui vérifie si l'id (qui est une variable de paramètre de route) est égal à 'blog'. Si c'est le cas, il envoie 'Blog' comme réponse.

Cependant, il y a un problème avec ce code. Après avoir envoyé la réponse 'Blog', le code continue à s'exécuter et essaie d'envoyer une autre réponse 'Autre'. En Express.js, une fois qu'une réponse a été envoyée, vous ne pouvez pas en envoyer une autre. Cela provoquera une erreur : Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client.

Pour corriger ce problème, vous pouvez ajouter un return dans le bloc if pour arrêter l'exécution du code une fois que 'Blog' a été envoyé, ou vous pouvez ajouter un else pour envoyer 'Autre' seulement si id n'est pas 'blog'.

Question **9**

Correct

Note de 1,00 sur 1,00

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.body.country Ne s'applique pas à la méthode GET, mais pour illustration, si vous aviez une requête POST avec un corps JSON comme { "country": "Canada" }

b. req.values.country Pas valide dans express

c. req.params.country Ne s'applique pas à cet exemple d'URI, car les paramètres sont passés via la requête, pas dans la route.

d. req.query.country /capitals?country=Canada ✔

e. req.country Pas valide dans express

Votre réponse est correcte.

La réponse correcte est :
req.query.country

Question **10**

Incorrect

Note de 0,00 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. 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*. ✖
- b. L'installation d'une nouvelle dépendance en développement (*devDependencies*) dans le *package.json* peut être faite avec la commande **npm install-dev nomDuPackage** ✖
- c. Le nom du fichier dans la commande "start:watch" doit obligatoirement correspondre au nom du fichier dans la variable "main" ✖
- d. La présence de la librairie *jest* n'est pas nécessaire pour le fonctionnement du projet. ✔
- e. Les commandes *npm test* et *npm run test* sont équivalentes et produiront le même résultat.

Votre réponse est incorrecte.

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.

a. Fausse. La déclaration indique que la version 4.18.1 de "express" sera téléchargée, mais le fichier package.json spécifie que la version compatible est "4.17.1". La notation "" signifie que les patchs mineurs peuvent être mis à jour automatiquement, mais pas les versions mineures. Ainsi, la version téléchargée serait la dernière version compatible avec la spécification, mais pas nécessairement 4.18.1.

b. Fausse. La commande correcte pour installer une nouvelle dépendance en développement est npm install --save-dev nomDuPackage. L'option correcte est --save-dev, pas -dev.

c. Fausse. Dans la commande "start:watch", le nom du fichier n'est pas nécessairement lié au nom du fichier dans la variable "main". La commande spécifie simplement le fichier à exécuter avec nodemon lors du lancement avec npm start: watch.

d. Vraie. La présence de Jest dans les devDependencies indique qu'il est utilisé pour les tests, mais n'est pas nécessaire pour le fonctionnement normal du projet. Les devDependencies sont des dépendances nécessaires uniquement pour le développement.

e. Vraie. Les commandes npm test et npm run test sont effectivement équivalentes et exécuteront la commande spécifiée dans la section "scripts" du package.json, qui est "jest" dans ce cas.

Question **11**

Correct

Note de 1,00 sur 1,00

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

- [x] a. 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.
- [ ] 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 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
- [ ] d. L'horizontale permet d'avoir un schéma de tables dynamiques, alors que la verticale est liée à un schéma statique.
- [ ] e. L'horizontale permet d'avoir différentes colonnes pour chaque ligne d'un document permettant une BD plus flexible que la version verticale

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 **12**

Terminer

Note de 4,00 sur 4,00

Soit le middleware suivant sur un serveur NodeJs qui offre un service de traduction et dictionnaires en ligne:

```
1 app.post("/sendDeleteDictionary", (req, res) => {
2     deleteDictionary(req.body.index);
3     res.status(201).send();
4 });
```

a) Selon l'architecture REST, quel est le niveau de maturité de Richardson de ce middleware? Justifiez votre réponse.

b) Selon l'architecture REST, proposez une nouvelle version du middleware afin d'atteindre le niveau 2 du modèle de maturité de Richardson. Vous pouvez assumer que la fonction deleteDictionary() retourne un booléen qui indique la réussite (ou non) de la suppression.

a) On est clairement au niveau 0. On utilise une requête POST alors que ce n'est pas le type de requête adapté (on devrait utiliser DELETE). On retourne le mauvais code de retour HTTP. On transmet des information dans le body alors qu'on devrait le faire dans l'URI.

b)

```
app.delete("dic/:id", (req, res) => {

if(deleteDictionary(req.params.id)) {

    res.status(204).send();

} else {

    res.status(404).send();

}});
```

Commentaire :

Question **13**

Terminer

Note de 3,00 sur 3,00

Cette session, dans les séances de travaux pratiques, vous avez travaillé sur la conception d'un site web de recettes. À partir du TP4, vous avez ajouté à votre projet un serveur dynamique utilisant NodeJs pour traiter les requêtes client. Pour ce faire, votre système suit une architecture Client-Serveur. Mis à part ce patron d'architecture, quel autre patron d'architecture pourrait représenter l'architecture de votre projet, notamment votre serveur?

**Justifiez** votre réponse en quelques lignes.

Architecture: Architecture orienté services

Dans le TP4, la gestion des recettes et la gestion de contact étaient traité indépendamment l'une de l'autre. Ce qui veut dire que, par exemple, un dysfonctionnement d'un des services ne provoquera pas d'erreur du côté de l'autre service et vice versa. Nous avons d'ailleurs implémenté ces deux fonctionnalité totalement en parallèle sans que ça ne nous cause le moindre problème.

Par contre, je ne considère pas que l'architecture était orienté micro-services, étant donné que les deux services dépendait du même routeur. Sans cette dépendance commune, ma réponse aurait été micro-services.

Commentaire :

Question **14**

Terminer

Note de 3,25 sur 4,00

Une des particularités de la librairie React est l'utilisation d'un DOM virtuel.

a) Expliquez ce que représente le DOM virtuel en React. Autrement dit, expliquez à quoi sert le DOM virtuel et comment les modifications au DOM virtuel affectent le DOM réel.

b) Expliquez en quoi le DOM virtuel de React et la gestion de la redondance dans la persistance des données sur MongoDB sont similaires.

a)

Le DOM virtuel permet de se tenir à jour des modifications qui devraient être apportées au DOM réel sans nécessairement modifier avoir à modifier le DOM réel à chaque fois. J'espère que c'est clair.

Donc, par exemple, il pourrait y avoir plusieurs modifications sur la page qui, sans la présence du DOM virtuel aurait modifié le DOM réel à chaque fois. En utilisant le DOM virtuel, on peut modifier le DOM virtuel après chaque petite modification, puis quand c'est nécessaire, mettre à jour le DOM réel grâce au DOM virtuel. La modification du DOM virtuel est moins demandante, c'est pour cette raison qu'on utilise cette méthode.

b)

Extrait des notes de cours à ce sujet :

" Représentation virtuelle de la structure du DOM (sous la forme d'un arbre) conservée en mémoire et éventuellement synchronisée (Réconciliation) avec le DOM réel. "

" MongoDB permet également d'avoir de la redondance où les mêmes données sont présentes sur plusieurs instances et sont synchronisées après une opération. "

Je vais expliquer ces deux phrases dans mes mots pour avoir mes points, même si je considère qu'elles sont déjà assez exlicite.

Point similaire : Dans les deux cas, l'information est dupliqué à au moins deux endroits différents (DOM réel et DOM virtuel, une instance et une autre différente) et est synchronisé. La synchronisation de ces informations est une opération plutôt couteuse, donc on l'effectue seulement quand on la juge réellement nécessaire, pour ce dernier point ça semble moins être le cas avec MongoDB.

Commentaire :
a) Le DOM virtuel se compare avec l'état précédant pour trouver les différences -0.25

b) Quelle est l'instance primaire (DOM Virtuel) et secondaire (DOM réel) ? -0.5

Question **15**

Terminer

Note de 4,00 sur 4,00

Expliquez la différence entre la redondance et la répartition (sharding) des données. Ensuite, donnez un avantage de la redondance et un avantage de la répartition.

Justifiez votre réponse.

Différence entre le redondance et la répartition :

Avec la redondance, la **même** information est dupliqué. Mais elle n'est pas nécessairement distribuée. Ex : Un usager a ses fichiers en local sur sa machine et a les même fichiers sur un serveur Microsoft grâce à OneDrive.

Avec la répartition, l'**ensemble de l'information** est **distribué**. L'information n'est pas nécessairement dupliquée. Ex : Un usager a ses fichiers personnels sur sa machine et ses fichiers scolaires sur OneDrive.

Avantage de la redondance : En cas d'incident (ex : un datacenter de OVH qui brûle), on peut toujours accéder à l'information en allant sur une autre instance ou était notre information.

Avantage de la répartition : On peut optimiser le coût d'accès à l'information selon nos besoins. Ex : Je conserve mes informations scolaires sur un serveur de l'école étant donné que je ne les utilise qu'à l'école (accès facilité) et je conserve mes informations personnelles à la maison étant donné que je ne les utilise que chez moi (accès facilité). Toute mon information n'est pas dupliquée, elle est répartie sur plusieurs instance de manière optimale.

Mes exemples sont du point de vue d'un usager, mais ils auraient tout aussi bien pu être du point de vu du côté serveur.

Commentaire :
L'exemple de la redondance est un exemple de distribution sur plusieurs places.

Les exemples sont un peu loins du cadre du cours, mais ok....

Question **16**

Terminer

Note de 2,00 sur 8,00

Voici le code source d'une simple application de compteurs développée en React qui contient 3 composantes : App, CounterList et Counter. Chaque compteur possède 2 boutons qui permettent d'incrémenter ou décrémenter sa valeur. Le bouton "Remise à 0" remet les valeurs de tous les compteurs à 0. Vous pouvez assumer que le code est fonctionnel.

## App :

```
function App() {
  const [counters, setCounters] = useState([
    { id: 1, value: 0 },
    { id: 2, value: 1 },
    { id: 3, value: 0 },
  ]);
  const handleIncrement = (counter) => {
    const localCounters = [...counters];
    const index = localCounters.indexOf(counter);
    counters[index].value++;
    setCounters(localCounters);
  };

  const handleDecrement = (counter) => {
    const localCounters = [...counters];
    const index = counters.indexOf(counter);
    localCounters[index].value--;
    setCounters(localCounters);
  };

  const handleReset = () => {
    setCounters(
      counters.map((x) => {
        return { id: x.id, value: 0 };
      })
    );
  };

  return (
    <div className="App">
      <CounterList counters={counters} onIncrement={handleIncrement} onDecrement={handleDecrement} onReset={handleReset} />
    </div>
  );
}
```

## CounterList :

```
const CounterList = ({ counters, onReset, onIncrement, onDecrement }) => {
  return (
    <div>
      {counters.map((counter) => {
        return <Counter counter={counter} inc={onIncrement} dec={onDecrement} />;
      })}
      <button onClick={onReset}>Remise à 0</button>
    </div>
  );
};
```

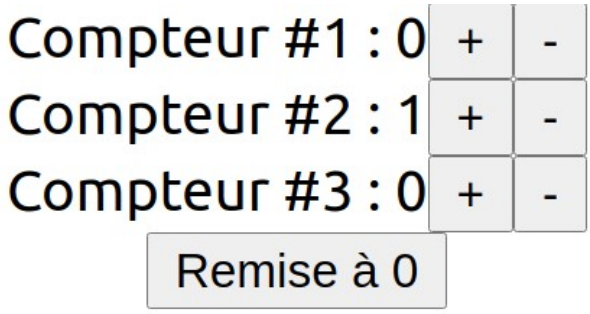
c)

```
const CounterList = ({ counters, onReset, onIncrement, onDecrement }) => {
  return (
    <div>
      {counters.map((counter) => {
        return (
          <Counter
            key={counter.id}
            counter={counter}
            onIncrement={() => onIncrement(counter.id)}
            onDecrement={() => onDecrement(counter.id)}
          />
        );
      })}
      <button onClick={onReset}>Remise à 0</button>
    </div>
  );
};
```

## Counter :

```
const Counter = ({ counter, inc, dec }) => {
  return (
    <div>
      <span>
        Compteur #{counter.id} : {counter.value}
      </span>
      <button onClick={() => inc(counter)}> + </button>
      <button onClick={() => dec(counter)}> - </button>
    </div>
  );
};
```

Voici également une capture d'écran de l'état initial de l'application (assumez qu'aucun CSS n'a été utilisé) :




En fonction du code source et la capture d'écran, répondez aux questions suivantes :

a) Est-ce que la variable *localCounters* des lignes **8** et **15** de App.js est nécessaire ? Si oui, **pourquoi** ? Si non, **comment simplifier la méthode** sans utiliser une variable locale ?

b) Est-ce qu'il y a un avantage de passer les méthodes *handleIncrement, handleDecrement* et *handleReset* aux composantes *CounterList* et *Counter* au lieu de passer seulement la méthode *setState()* et implémenter la logique de manipulation dans les composantes ? **Justifiez** votre réponse.

c) Il a un problème potentiel avec la manière de créer les composantes *Counter* dans *CounterList*. Quel est ce problème et comment doit-on le régler ?

d) Êtes-vous d'accord avec la gestion de l'état de la variable *counters* ? Si oui, **pourquoi** ? Si non, **quel(s) changement(s)** apportiez-vous au code ? **Justifiez.**

a) Non, on aurait pu écrire :

1. const index = [...counters].indexOf(counter);
2. localCounters[index].value--;
3. setCounters(localCounters);

```
const handleDecrement = (counter) => {
  setCounters((prevCounters) => {
    const updatedCounters = [...prevCounters];
    const index = updatedCounters.indexOf(counter);
    updatedCounters[index].value--;
    return updatedCounters;
  });
};
```

b) Oui, il y a un avantage. Si le développeur décide de modifier l'application en ajoutant, par exemple, une composante qui ne fait qu'augmenter, il pourra facilement le faire étant donné qu'il est déjà possible de passer la méthode handleIncrement au composant.

c) Il pourrait y avoir un conflit entre les évènements passés en paramètres.

d) Oui, UseState semble bien adapté pour gérer l'état d'un composant. Le seul petit problème que je pourrais voir avec ça c'est qu'on ne change pas la valeur de la variable, on la recrée à chaque fois. Ça peut être coûteux. Je ne vois pas de meilleur option.

b) Réutilisation du code : En passant les méthodes spécifiques aux composants, vous permettez une plus grande réutilisation du code. Ces méthodes sont spécifiquement définies pour effectuer des opérations sur les compteurs, et si vous avez d'autres composants qui nécessitent une logique similaire, vous pouvez les réutiliser sans duplicer le code.

Maintenabilité : En cas de modification de la logique métier liée aux compteurs, il est plus facile d'apporter des changements dans une seule partie du code (dans App), plutôt que de disperser la logique dans plusieurs composants.

Commentaire :

a) L'exemple contredit la réponse ; vous avez une variable locale -2

b) OK

c) Pas de justification -1

d) CounterList reçoit des props qu'il ne fait que passer à Counter. Il faudrait un Context ou mettre toute la logique dans Counter -3

c) Le problème potentiel dans la création des composantes Counter dans CounterList réside dans le passage des méthodes d'incrémentation et de décrémentation en tant que propriétés aux composantes Counter. Pour résoudre cela, il est recommandé d'utiliser la propriété key unique pour chaque composante Counter afin d'éviter tout conflit potentiel entre les événements passés en paramètres. Cela garantit que chaque composante Counter est identifiée de manière unique, évitant ainsi des comportements inattendus lors des mises à jour d'état.

Question **17**

Terminer

Note de 6,00 sur 6,00

Vous développez un service web comme GitHub pour manipuler des utilisateurs et des entrepôts (*repositories*), et étoiler (*star*) nos dépôts favoris.

Un entrepôt est représenté par un **nom d'utilisateur** et un **nom d'entrepôt unique par utilisateur** ( Ex : polymtl_log2440/TP6 où le nom de l'utilisateur est "polymtl_log2440" et le nom de l'entrepôt est "TP6" ) ainsi qu'une description textuelle.

Voici une ébauche d'une partie de ce service web pour différentes fonctionnalités. Assumez que ":username" représente le nom de l'utilisateur à insérer dans l'URI relatif.

Vous devez le modifier pour qu'il respecte au minimum le niveau 2 de Richardson d'un service web REST.

Spécifiez **clairement** tous les changements à apporter (verbes HTTP, format des requêtes, codes de réponses, valeurs de retour du serveur, etc.).

1. Pour obtenir les dépôts créés par un certain utilisateur et triés selon certains critères

POST /users/:username/repos

**Corps de la requête**

```
{ sort: <"created" ou "name">, direction: <"asc" ou "desc"> }
```

**Réponse :**

```
Status: 200 OK

<Liste des entrepôts de l'utilisateur>
```

2. Pour créer un nouveau dépôt pour un certain utilisateur (name et description sont des <string> quelconques)

POST /users/:username/repos?name=<string>&description=<string>

**Corps de la requête vide**

**Réponse :**

```
Statut: 20O OK
Location: /users/:username/repos

(Le corps est vide)
```

3. Pour supprimer le dépôt d'un utilisateur (name est une <string> quelconque)

POST /users/:username/repos/delete?name=<string>

**Corps de la requête**

```
{ id: <string> }
```

**Réponse :**
(Le corps est vide)

```
Statut: 200 OK
```

4. Pour étoiler un dépôt d'un autre utilisateur

POST /users/:username/starred/:owner/:repo

**Corps de la requête vide**

**Réponse :**

Statut: 200 OK
(Le corps est vide)

5. Pour retirer l'étoile d'un dépôt d'un autre utilisateur

PATCH /users/:username/starred/:owner/:repo

**Corps de la requête vide**

**Réponse :**

Statut: 200 OK
(Le corps est vide)

1. Pour obtenir les dépôts créés par un certain utilisateur et triés selon certains critères <type> vaut created ou name et <dir> vaut asc ou desc, ce sera à l'utilisateur de choisir

GET /users/:username/repos?sort=<type>&direction<dir>

**Corps de la requête**

**Corps de la requête vide**

**Réponse :**

Status: 200 OK

<Liste des entrepôts de l'utilisateur>

2. Pour créer un nouveau dépôt pour un certain utilisateur (name et description sont des <string> quelconques)

POST /users/:username/repos

**{ name : <string>, description: <string>}**

**Réponse :**

Statut: 201 CREATED
Location: /users/:username/repos

(Le corps est vide)

3. Pour supprimer le dépôt d'un utilisateur

DELETE /users/:username/repos/:id

**Corps de la requête**

**Corps de la requête vide**

**Réponse :**
(Le corps est vide)

Statut: 204 NO CONTENT

(Le corps est vide)

4. Pour étoiler un dépôt d'un autre utilisateur

PATCH /users/:username/:owner/:repo?star=true

**Corps de la requête vide**

**Réponse :**

```
Statut: 204 NO CONTENT
(Le corps est vide)
```

5. Pour retirer l'étoile d'un dépôt d'un autre utilisateur

PATCH /users/:username/:owner/:repo?star=false

Corps de la requête vide
Réponse :

Statut: 204 NO CONTENT

```
(Le corps est vide)
```

D'après ce que j'ai compris, on doit modifier les requêtes et les réponses HTTP afin de respecter le niveau 2 de Richardson.

Mes modifications sont de couleur rouge.

Commentaire :
Excellente façon de répondre à la question. Super clair et facile pour la correction

◄ LOG2440 - Automne 2021 - Contrôle Pratique

Aller à...

Introduction ►