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

```json
{
  "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"
  }
}
```

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

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

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

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) Le niveau de maturité de ce middleware est 1, il utilisent une méthode post afin de delete un élément et retourne un code 201, ce qui ne devrait pas se passer apres un delete. Pourtant, le point d'entrée, l'information entrée par le client est utilisé dans la requête ce qui en fait un niveau 1 et non un niveau 0.

b) La partie le plus importante de ce middleware est l'utilisation de post a la place de delete. Nous pourrions utiliser ce middleware a la place :

```
app.delete("/sendDeleteDictionary", (req, res) => {
    const index = req.body.index
    deleteDictionary(index);
    res.status(200).send();
});
```

```
app.delete("/dictionaries/:index", (req, res) => {
  const success = deleteDictionary(req.params.index);
  if (success) {
    res.status(204).send(); // 204 No Content for successful deletion
  } else {
    res.status(404).send(); // 404 Not Found if the resource does not exist
  }
});
```

Commentaire :
b) mauvais path -0.5
pas de error checking -0.5

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.

La redondance est tout simplement une forme de backup de nos données, les mêmes informations sont écrites plusieurs fois a des endroits différents, tandis que la répartition consiste a séparer l'information en chunks sans la répliquer, ce n'est qu'un moyen de séparer l'information.

La redondance est très utile, par exemple, elle permet de changer d'instance si l'instance primaire tombe en panne, ce qui empêcherai un effondrement complet des fonctionnalités.

Le partitionnement est également utile, il permet de regrouper les clés proches (qui sont souvent liés aux autres ou utilisés en même temps) dans les mêmes partitions, ce qui nous permet de cibler des partitions lorsque nous voulons accéder a des données en particulier, sans a avoir a utiliser toute l'information présente.

Commentaire :
On peut cibler des données repliqués aussi. Ce n'est pas un "avantage" de la répartition

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 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
- [ ] c. 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
- [x] 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 permet d'avoir un schéma de tables dynamiques, alors que la verticale est liée à un schéma statique.

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.

Notre serveur peut être représenté par une architecture orientée évènements, il s'agit d'un système interactif avec le client comme producteurs d'évènements et le serveur serait le consommateur d'évènements. Les avantages et désavantages sont visible dans nos TPs, notre site web est très interactif avec des réaction rapides aux évènements de l'utilisateur , mais la gestions des évènements n'est pas triviale.

Nous pourrions également le comparer avec une architecture a trois niveaux, notre couche de présentation est ce que nous montrons au client, notre couche de traitement est la logique par rapport aux entrées de l'utilisateur et notre couche d'acces aux données est notre communication avec notre serveur MongoDB.

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) Il est important de savoir qu'il existe deux DOM virtuels, un de notre état courant et un de notre état précédent. Lorsque un changement est apporté a un de nœuds du DOM, le DOM Virtuel de notre état courant est modifié et est par la suite comparé au DOM virtuel de notre état précèdent. Les différences sont calculées et les modifications nécessaires sont appliqués au DOM réel, en ne modifiant que les composantes concernées, ce qui nous évite a avoir a recharger toute la page ce qui serait une opération couteuse. Par la suite, les deux DOMs virtuelles se synchroinise.

b) Il y a plusieurs similarités entre les deux, comme pour nos deux DOMs virtuels, les modifications ne sont pas appliqués aux deux de manière instantanées, les modifications sont d'abord appliqués a l'instance primaire de notre redondance et sont ensuite appliqués aux instances secondaires de maniere asynchrone, la meme chose se passent pour nos deux DOMs virtuels. Une fois les opérations aux DOMs ou les changement a la databse terminées, les deux instances (ou plus) de la database ont les memes données tout commne les deux instances des DOMS virtuel.

Commentaire :
a) OK

b) On applique les changements du DOM virtuel au DOM réel et non l'ancien état -0.5

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 elle n'est pas nécessaire, nous pourrions simplement créer une liste modifiées et l'implémenter dans le HTML avec setCompteur

b) Il existe quelques avantages a passer les méthodes, dans notre méthode présente toute les informations doivent être donner de la composante parent aux composantes enfants ce qui alourdit le code et l'arborescence est rigide. Pourtant, notre façon de fonctionner précédemment est fonctionnel et le couplage entre les composantes enfants, de plus la gestion des compteurs est centralisé dans app.

c) Il n'y a pas de clé individuelle pour chaque compteur, ce qui pourrait compliquer l'application des fonctions sur les compteurs. il suffirait d'attribuer a une clé une variable qui s'incrémente automatiquement comme i.

d) Il serait plus intéressant d'avoir uniquement une liste des compteurs sans id introduit, avec des key définit pour chaque compteur. Il serait également intéressant d'insérer une variablke par défaut. Une implémentation possible serait celle ci :

const [counters, setCounters] = useState(default_classes);

Commentaire :

a) Comment créer une liste modifiée sans variable locale ? -2

b) Réponse incomplète -1

c) On veut l'attribut "key" -0.5

d) Il y a déjà un ID ET des états par défaut -3

1) POST ne devrait pas être utilisé ici, nous voulons récupérer des dossiers alors le verbe correcte est GET. De plus, la méthode get n'a pas de corps, il faut donc inserer les parametres direction et sort dans l'URI

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

Corps de la requête vide

Réponse :

```
Status: 200 OK

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

2) La requete POST devrait avoir un corps contenant l'information a transmettre, il n'est pas nécessaire de l'avoir dans L'URI. De plus, lors de la création d'un élément le code 201 devrait etre retourné.Puisque nous avons un 201, pas besoin de l'en tête location.

POST /users/:username/repos

Corps de la requête

```
{ name: <string> , description=<string>}
```

Réponse :

```
Statut: 201 CREATED

(Le corps est vide)
```

3) Ici, la requete DELETE devrait être utilisé, le nom de la ressource n'est pas nécessaire, seule son identifiant est utile, il peut être dans le corps ou dans l'URI. »Le code de retour 200 est acceptable.

DELETE /users/:username/repos/delete

Corps de la requête

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

Réponse :

```
Statut: 200 OK
```

(Le corps est vide)

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: 200 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)
```

4) PATCH devrait être utiliser ici, nous cherchons uniquement a mettre a jour une ressource existante de maniere partielle. De plus, l'information a modifier devrait etre dans le corps de le requete. De plus, le corps de la reponse devrait avoir la ressource mis a jour.

PATCH /users/:username

Corps de la requête

```
{ id: <string>, owner: <string>, star: <boolean> }
```

Réponse :

```
Statut: 200 OK
<entrepot etoilé>
```

5) Meme chose que au dessus.

PATCH /users/:username

Corps de la requête

```
{ id: <string>, owner: <string>, star: <boolean> }
```

Réponse :

```
Statut: 200 OK
<entrepot non etoilé>
```

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

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

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 ✔

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 Pipeline ✔ serait la plus appropriée.

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

Aucune de ces raisons n'explique adéquatement le choix de cette architecture. ✔

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 trois niveaux
- [ ] b. Architecture client-serveur
- [ ] c. Architecture pipeline
- [ ] d. Architecture orientée événements
- [x] e. Architecture orientée services ✔
- [ ] f. Architecture pair-à-pair

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.values.country
- [x] b. req.query.country
- [ ] c. req.params.country
- [ ] d. req.country
- [ ] e. req.body.country

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
```

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

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

- [ ] a. HTTP
- [ ] b. MIddleware
- [ ] c. Cookie
- [ ] d. Origin
- [x] e. CORS ✔

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** ?

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