Commencé le	vendredi 19 avril 2024, 09:30
État	Terminé
Terminé le	vendredi 19 avril 2024, 12:00
Temps mis	2 heures 30 min
Note	**19,00** sur 40,00 (**47,5**%)

Description

LISEZ CES INSTRUCTIONS AVANT DE COMMENCER L'EXAMEN

L'examen est composé de plusieurs sections. Vous êtes libres 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 multiples impactera la note finale de la question.
- Une bonne réponse sans justification correcte sera pénalisée.
- 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

Non répondue

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.

Question 2

Terminé

Note de 0,00 sur 6,00

Vous travaillez sur un système de journalisation des événements dans votre application React. Ce système permet de sauvegarder différents messages en un endroit centralisé géré par un *Reducer*. Voici l'implémentation de celui-ci :

```
export default function reducer(state, action) {
  switch (action.type) {
    case 'ADD':
      return { messages: [...state.messages, action.payload] };
    case 'RESET':
      return { messages: [] };
    default:
      return state;
  }
}
```

Afin de tester votre système, vous avez la composante suivante qui incrémente un compteur et affiche les messages de votre système de journalisation :

```
function App() {
  const [state, dispatch] = useReducer(reducer, { title: 'Debug Info', messages: [] });
  const [count, setCount] = useState(1);

  return (
    <>
      <h1>{state.title}</h1>
      <button onClick={() => {
        setCount(count + 1);
        dispatch({ type: 'ADD', payload: `Message #${count}` });
      }}> Ajouter un message </button>
      <button onClick={() => dispatch({ type: 'RESET', payload: {} })}> Vider </button>
      <ul>
        {state.messages.map((msg, i) => (
          <li key={i}>{msg}</li>
        ))}
      </ul>
    </>
  )
}
```

a) Lorsque vous cliquez sur les boutons de la composante, vous voyez que l'état est mis à jour, mais certains éléments dans la page ne sont plus affichés. Expliquez la raison de ce comportement et proposez une solution. (3 points)

b) Vous voulez ajouter un message à chaque fois que *count* est modifié. Vous écrivez le code suivant, mais vous obtenez une boucle à l'infini. Pourquoi ? Est-ce que simplement retirer *state* du tableau de dépendance réglerait le problème ? Justifiez. (3 points)

```
useEffect(() => {
    dispatch({ type: 'ADD', payload: `Count a été modifié : ${count}` });
}, [state]);
```

a) L'état est mis à jour car le hook useState incrémente correctement la variable setCount à chaque instance. Cependant, l'information n'est pas communiquée effectivement à toutes les composantes. On peut utiliser le Contexte pour palier à ce problème.Les 2 hooks de React : useState et useContext permettent d'implémenter une gestion d'état local et partagé à travers plusieurs composantes. Le hook useReducer joue le même rôle en rhéorie, mais il est mal utilisé ici, comme il doit être combiné au contexte. il faut un fournisseur pour que les composantes aient accès au nouvel état. Pour cette raison, la logique des actions à effectuer n'est pas dispatched.

b) De retirer simplement state des dépendance présente un danger de boucle infini. L'État est constamment modifié à cause du hook useState. On peut corriger par [state.count] pour réellement détecter le changement du compte.

Commentaire :
a) Il y a 1 seule composante ici. Pas besoin d'un Context. Le titre "Debug Info" sera retiré après la 1re modification de l'état (click sur n'importe quel bouton). La raison est que le reducer retourne un objet qui continent seulement les messages et non tout l'ancien état au complet. -3

b) Retirer state retire le problème de boucle infinie. La boucle est due au changement de state et non à useState(). [state.count] n'est pas une solution valide : count n'est pas un attribut de la variable state -3

Question 3

Terminé

Note de 4,00 sur 5,00

a) Vous travaillez sur un service web pour la sauvegarde et la gestion de fichiers liés à des comptes utilisateurs. Votre collègue implémente le gestionnaire d'authentification suivant qui valide qu'un nom d'utilisateur et un mot de passe fournis sont valides.

En cas de réussite, un jeton unique est renvoyé au client ayant fait la requête, sinon un message d'erreur est retourné. Assumez que la fonction *validateLogin* est bien implémentée.

```
app.post('/login', (req, res) => {
    const token = validateLogin(req.query.username, req.query.password);
    if (token) {
        return res.status(200).send({ authToken: token });
    }
    return res.status(401).send({ error: 'Informations de connexion invalides' });
});
```

Selon votre collègue, l'utilisation de la méthode POST rend la communication sécuritaire. Il y a cependant une faille de sécurité dans son implémentation. Identifiez cette faille et proposez une solution au problème. Justifiez votre réponse (3 points)

b) Votre service reçoit une requête HTTP avec l'en-tête suivant : **Accept: text/html, text/plain;q=0.4, application/*; q=0.1**

La réponse est un fichier JSON. Est-ce que ceci est une réponse acceptable ou il y a eu une erreur de traitement ? Justifiez votre réponse (2 points).

a) La méthode POST n'est ni sécuritaire, ni indempotente. En effet, elle modifie l'état du serveur en transmettant de l'information et crée ainsi une ressource. Un premier probleme est si le username ou le mdp contiennent des caracteres speciaux, alors ils peuvent être mal interprete et req.query.username et req.query.password ne contiendront pas les bonnes valeurs. Ensuite, c'est une faille de sécurité de mettre les username et password en paramètres comme ceci comme ils seront affichés en clair dans l'historique de navigation du client ou même dans les log. Une approche alternative serait de faire une requête post avec un corps de requête qui lui aurait le username et mot de passe encodés en JSON. Aussi, on retourne le code 200 OK ce qui n'est pas bonne pratique car il faut envoyer le code 201 created pour être plus exact.

b) Ici, c'est le concept de négotiation controlée par le serveur. Chaque valeur a un degré de préférence qu'on note avec q.

Accept: text/html, text/plain;q=0.4, application/*; q=0.1 signifie qu'on veut un document de type text/html, mais on accepte text/plain ou n'importe quoi dans cet ordre de préférence. Il n'y a pas erreur de traitement.

Commentaire :
a)

Identification du problème (nom d'utilisateur + mot de passe) envoyé dans l'URI par query parameters: Bonne réponse

Notez que: ça n'a rien à voir avec la "sécurité" ni l'idempotence, ni les caractères spéciaux (cars ils pourraient être encodés décodés correctement)

Bonne solution (mais cela n'a rien à voir avec le HTTP status)

b) Mauvaise justification: on ne s'attends pas à avoir tout, on s'attends à potentiellement, mais non préférablement, recevoir n'importe quel fichier de type application/* (dont application/json fait parti). Il n'y a pas de */* d'indiqué -1pts

Question 4

Terminé

Note de 4,50 sur 5,00

Dans le cadre de votre travail pratique #4, vous avez mis en place un système utilisant la librairie ReactJs pour la construction de votre site web et MongoDB en tant que base de données.

a) Outre la base de données MongoDB, combien de serveurs ont été nécessaires pour le fonctionnement de votre système et quel est le rôle de chaque serveur? Justifiez vos réponses. (3 points)

b) Est-ce que le mécanisme CORS devait être activé pour le bon fonctionnement de votre système? Justifiez votre réponse. (2 points)

a) 2. Serveur dynamique et serveur statique. Pour la construction d'un site web, on nécessite le chargement de balises html, feuilles de style, etc. Ceci est du contenu statique. Par conséquent, c'est le serveur statique qui répond à ces requêtes. Cependant, de la logique de traitement à aussi été mise en place. L'utilisation de hook pour la gestion d'états permise par React nécéssite la mise en place d'un serveur dynamique. MongodB, base de données NoSQL donc = disposition et format de données dynamique, nécessite un serveur dynamique.

b)Oui, vu l'utilisation de 2 serveurs dans un même projet. Pour des requêtes avec comme origine autre que le serveur, il fallait mettre en place le protocole CORS à travers un middleware: app.use(cors());

Commentaire :
a) 3/3

b) 1,5/2 Sans CORS, il n'aurait pas été possible d'accéder les données du serveur dynamique à partir de la page web.

Question 5

Partiellement correct

Note de 0,50 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();
  }
  return res.set("Content-Type", "application/json").send(result);
});
```

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

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

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 1.
Les réponses correctes sont :
Le client reçoit une réponse HTTP avec le code 404 et aucun corps.,

Le serveur va lancer une erreur.

Question 6

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 envoyées par plusieurs clients, quelle fonction est à prioriser et pourquoi ?

- a. *promises.readFile(fileName)* puisque le code asynchrone est toujours plus performant que le code synchrone.
- b. *promises.readFile(fileName)* puisque la lecture asynchrone ne bloquera pas le reste de l'application. ✓
- c. *readFileSync(fileName)* puisque la lecture synchrone nous garantit d'avoir le contenu complet du fichier.
- d. Il n'y a pas de fonction à prioriser : le résultat est pareil dans les deux cas.
- e. *readFileSync(fileName)* puisque la lecture synchrone ne bloquera pas le reste de l'application.

Votre réponse est correcte.

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

Question 7

Correct

Note de 1,50 sur 1,50

Parmi les énoncés suivants, lequel définit mieux ce qu'est une application monopage (*Single Page Application*) ?

- a. Une application monopage utilise des pages Web générées par un serveur.
- b. Une application monopage ne permet pas de partager les informations entre plusieurs pages du même site.
- c. Une application monopage est restreinte à une seule page HTML et un seul URI.
- d. Une application monopage est une application qui ne recharge pas son contexte JavaScript après son chargement initial. ✓
- e. Une application monopage est une application qui n'a pas besoin d'interagir avec un serveur dynamique.

Votre réponse est correcte.

La réponse correcte est :
Une application monopage est une application qui ne recharge pas son contexte JavaScript après son chargement initial.

Question 8

Correct

Note de 1,50 sur 1,50

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

- a. CORS ✓
- b. TLS
- c. Middleware
- d. HTTPs
- e. Origin
- f. Cookie

Votre réponse est correcte.

La réponse correcte est :
CORS

Question 9

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

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 10

Partiellement correct

Note de 0,50 sur 2,00

Parmi les énoncés suivants concernant une architecture à N-niveaux, lesquels sont faux ?

- a. Le nombre maximal de niveaux est de 3, même dans un système distribué.
- b. Un logiciel qui ne fait pas de communication réseau peut quand même utiliser une architecture de N-niveaux.
- c. Un niveau peut être partagé par plusieurs composantes du même système.
- d. Une séparation physique des différentes couches dans un système n'est pas obligatoire. ✗
- e. Contrairement à une architecture pair-à-pair, une architecture à N-niveaux est toujours plus sécuritaire. ✓

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 1.
Les réponses correctes sont :
Le nombre maximal de niveaux est de 3, même dans un système distribué.,

Contrairement à une architecture pair-à-pair, une architecture à N-niveaux est toujours plus sécuritaire.

Question 11

Partiellement correct

Note de 0,50 sur 2,00

Considérez la composante React suivante utilisant le *hook* useState :

```
import { useState } from "react";

export default function UseStateQ() {
  const [data, setData] = useState([]);
  return (
    <div>
      <h1>UseStateQ</h1>
      {data.map((d, i) => (<p key={i}>{d}</p>))}
      <button onClick={() => {/* TODO */ }}>Ajout</button>
    </div>
  );
}
```

Parmi les choix suivants, lesquels permettent de rajouter une valeur aléatoire entre 0 et 1 correctement à l'attribut "data" à travers un bouton ?

- a. <button onClick={ () => setData(Math.random().toFixed(2)) }>Ajout 3</button>
- b. <button onClick={ () => setData(data + Math.random().toFixed(2)) }>Ajout 5</button> ✗
- c. <button onClick={ () => setData([...data, Math.random().toFixed(2)]) }>Ajout 1</button>
- d. <button onClick={ () => setData([data + Math.random().toFixed(2)]) }>Ajout 2</button>
- e. <button onClick={ () => setData((x) => x.concat(Math.random().toFixed(2))) }>Ajout 4</button> ✓

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 1.
Les réponses correctes sont :
<button onClick={ () => setData([...data, Math.random().toFixed(2)]) }>Ajout 1</button>,

<button onClick={ () => setData((x) => x.concat(Math.random().toFixed(2))) }>Ajout 4</button>

Question 12

Correct

Note de 1,50 sur 1,50

Pourquoi est-ce que les méthodes du pilote MongoDB de NodeJS et l'API Fetch du navigateur sont asynchrones ?

- a. NodeJS impose à ce que toutes ces méthodes soient asynchrones.
- b. La communication avec un serveur ou une base de données doit être asynchrone puisqu'elle se fait par HTTP.
- c. C'est faux : seulement les méthodes de l'API Fetch sont asynchrones.
- d. La communication avec un serveur externe doit être asynchrone pour ne pas bloquer le fil d'exécution du navigateur ou de NodeJS. ✓
- e. Ceci est un choix fait par l'équipe de développement du pilote et des navigateurs.

Votre réponse est correcte.

La réponse correcte est :
La communication avec un serveur externe doit être asynchrone pour ne pas bloquer le fil d'exécution du navigateur ou de NodeJS.

Question 13

Incorrect

Note de 0,00 sur 1,50

Quel est le rôle du mot clé **await** dans une fonction asynchrone ?

- a. Il attend de manière synchrone le retour de la promesse attendue avant de continuer l'exécution du reste du code. ✗
- b. Il force la fonction à renvoyer un objet Promise.
- c. Aucune de ses réponses.
- d. Il met en pause l'exécution de la fonction asynchrone jusqu'à ce que la promesse soit résolue ou rejetée.
- e. Il saute l'exécution de la fonction et attend l'événement asynchrone suivant.

Votre réponse est incorrecte.

La réponse correcte est :
Il met en pause l'exécution de la fonction asynchrone jusqu'à ce que la promesse soit résolue ou rejetée.

Question 14

Correct

Note de 1,50 sur 1,50

Pour quelle méthode HTTP parmi les suivantes, un serveur doit-il obligatoirement retourner un code de retour ?

- a. OPTIONS
- b. Aucune de ces méthodes
- c. POST
- d. PATCH
- e. GET
- f. Toutes ces méthodes ✓
- g. HEAD
- h. DELETE

Votre réponse est correcte.

La réponse correcte est :
Toutes ces méthodes

Question 15

Incorrect

Note de 0,00 sur 1,50

Vous travaillez sur un service de gestion d'images et albums d'images en ligne.

Afin de pouvoir faire une mise à jour du contenu d'un album, un client doit envoyer la requête suivante à votre serveur (vous êtes à la version 2 de votre service):

PUT https://api.polyimage.com/2/album/update/:albumID

Le corps de la requête contient seulement les informations de l'album (titre, description, images, etc.) à mettre à jour.

Considérant que vous voulez avoir un service REST de maturité 2, quelle serait une meilleure **route** si on veut conserver le même contenu pour le corps de la requête ?

- a. PUT https://api.polyimage.com/2/album/:albumID
- b. GET https://api.polyimage.com/2/album/:albumID
- c. POST https://api.polyimage.com/2/album/update/:albumID ✗
- d. GET https://api.polyimage.com/2/album/update/:albumID
- e. PATCH https://api.polyimage.com/2/album/:albumID

Votre réponse est incorrecte.

La réponse correcte est :
PATCH https://api.polyimage.com/2/album/:albumID

Question 16

Incorrect

Note de 0,00 sur 2,00

Voici le code suivant qui fait appel à la fonction **promiseGenerator(input)**. Cette fonction prend en paramètre une chaîne de caractère **input** et retourne une Promesse avec 2 possibilités :

1. La promesse est résolue avec la valeur **input** (type : *string*) passée en paramètre

2. La promesse est rejetée avec le code **-1** (type : *number*)

```
async function fetchDataAsync() {
  const res = await promiseGenerator('Allo')
    .then((x) => { x.toUpperCase() })
    .catch(() => 'erreur dans le code');
  console.log(`résultat final: ${res}`);
};

fetchDataAsync();
```

Basé sur le code plus haut, parmi les énoncés suivants, lesquels sont vrais ?

- [ ] a. L'appel de la fonction est invalide puisque les mots clé **async** et **await** sont mal utilisés.
- [x] b. Une promesse rejetée afficherait "résultat final: erreur dans le code" dans la console. ✓
- [ ] c. Une promesse résolue n'afficherait rien dans la console.
- [x] d. Une promesse résolue afficherait "résultat final: ALLO" dans la console. ✗
- [ ] e. Une promesse résolue afficherait "résultat final: undefined" dans la console.
- [x] f. Une promesse rejetée afficherait "erreur dans le code" dans la console. ✗
- [ ] g. Cette fonction est invalide puisqu'on ne peut pas retourner deux types différents de la même fonction.

Votre réponse est incorrecte.

Les réponses correctes sont :
Une promesse résolue afficherait "résultat final: undefined" dans la console.,

Une promesse rejetée afficherait "résultat final: erreur dans le code" dans la console.

Question 17

Incorrect

Note de 0,00 sur 2,00

Vous avez une base de données MongoDB avec des documents qui représentent des jeux vidéo dans un agrégateur de revues en ligne :
En voici un exemple :

```
{
    "title": "Helldivers 2",
    "score": 82,
    "platform": [
        "PC",
        "Xbox",
        "PS5"
    ],
    "date": "2024-02-08"
}
```

Voici le code suivant qui récupère les jeux disponibles sur la plateforme **PS5** ayant un score supérieur à **75** et affiche certaines informations dans la console :

```
const res = await collection.find(
  { platform: "PS5", score: { $gt: 75 } }).toArray();
const mongo_response = res.map((item) => ({ title: item.title }));
console.log(mongo_response);
```

Assumez que l'objet **collection** représente une collection existante sur votre base de données.
Sélectionnez les réponses qui affichent la même chose que le console.log de la ligne **4**.

a.
```
const mongo_response = ((await collection.find({}).toArray())
  .reduce((acc, item) => {
    if (item.platform.includes('PS5') && item.score > 75)
      acc.push({ title: item.title });
      return acc;
  }, []));
console.log(mongo_response);
```

b.
```
const mongo_response = await collection.find(
  { platform: "PS5", score: { $gt: 75 } },
  { projection: { _id: 0, date: 0, score: 0, platform: 0 } }).toArray();
console.log(mongo_response);
```

c. ✕
```
const mongo_response = (await collection.find({},
  { projection: { _id: 0, title: 1 } }).toArray())
  .filter((item) => item.platform.includes("PS5") && item.score > 75);
console.log(mongo_response);
```

d. ✕
```
const mongo_response = await collection.find(
  { platform: "PS5", { score > 75 } },
  { projection: { _id: 0, title: 1 } }).toArray();
console.log(mongo_response);
```

Votre réponse est incorrecte.

Les réponses correctes sont :

```
const mongo_response = ((await collection.find({}).toArray())
  .reduce((acc, item) => {
    if (item.platform.includes('PS5') && item.score > 75)
      acc.push({ title: item.title });
      return acc;
  }, []));
console.log(mongo_response);
```

,

```
const mongo_response = await collection.find(
  { platform: "PS5", score: { $gt: 75 } },
  { projection: { _id: 0, date: 0, score: 0, platform: 0 } }).toArray();
console.log(mongo_response);
```

Question 18

Incorrect

Note de 0,00 sur 2,00

Voici le code d'un serveur utilisant la librairie ExpressJS.

```
const express = require('express');
const http = require('http');
const app = express();

function handleRequest(req, res, nextFn, handler) {
    const message = `Requête ${req.method} reçue sur ${req.originalUrl} et capté par ${handler}`;
    console.log(message);
    res.setHeader(`X-${handler}`, handler);
    nextFn();
}
app.get("/", (req, res, next) => { handleRequest(req, res, next, 'GET') });
app.post("/cours/*", (req, res, next) => { handleRequest(req, res, next, 'POST') });
app.put("/cours", (req, res, next) => { handleRequest(req, res, next, 'PUT') });
app.delete("/cours", (req, res, next) => { handleRequest(req, res, next, 'DELETE') });
app.use("/cours", (req, res, next) => { handleRequest(req, res, next, 'USE') });
app.all("/cours/gigl", (req, res, next) => { handleRequest(req, res, next, 'ALL') });

// Renvoyer la réponse
app.use((req, res, next) => { res.status(200).send('Bonjour du serveur!') });

http.createServer(app).listen(3000);
console.log('Serveur avec Express démarré sur localhost:3000');
```

Vous envoyez la requête suivante : **POST /cours/gigl/log2440**

Quels en-têtes seront présents dans la réponse du serveur ?

- a. X-Get : GET ✗
- b. X-Use : USE ✓
- c. X-Delete : DELETE ✗
- d. X-Put : PUT ✗
- e. X-Post : POST ✓
- f. X-Patch : PATCH
- g. Aucun : le serveur va générer une erreur puisqu'on essaie de modifier l'objet **res** à plusieurs reprises
- h. X-All : ALL ✗

Votre réponse est incorrecte.

Les réponses correctes sont :
X-Post : POST,

X-Use : USE