Révision pour l'examen final

Plan du cours

- Information sur l'examen
- Exercices
- Questions

Information sur l'examen

- L'examen aura lieu le **10 décembre** de **13h30 à 16h00** (2h30)
  - Salles au Lassonde : L-6611, L-6612, L-6613, L-6614, L-6616 et L-6624
  - Voir l'horaire des examens finaux pour la répartition des salles
- L'examen se fera sur la plateforme Moodle Examen, comme le contrôle pratique :
  - Une **copie des PDF** des notes de cours est disponible sur l'instance Moodle
  - Vous avez aussi droit à de la documentation **papier** (manuscrite ou imprimée)
- L'examen est composé de questions à choix multiple et des questions à développement
  - Il y aura 16 questions au total

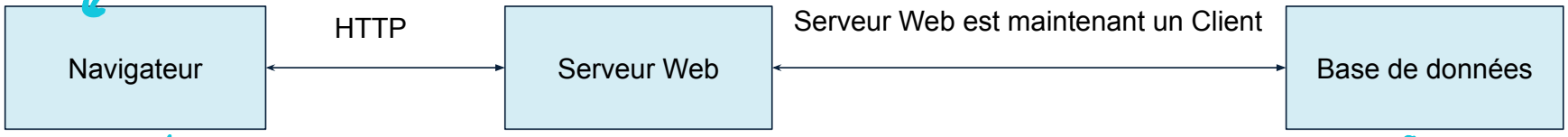
Information sur l'examen

- L'examen porte surtout sur la matière vue après le contrôle pratique :
  - HTTP et AJAX
  - Serveurs Web, NodeJS/Express
  - Architectures logicielles
  - Persistance des données, MongoDB
  - React et programmation réactive
- L'examen se concentre sur votre capacité de faire des liens entre les notions du cours
  - Il pourrait avoir des liens avec la matière du début de la session (HTML et JS)

Résumé de la matière

un serveur dynamique peux être statique aussi

peux pas être un serveur

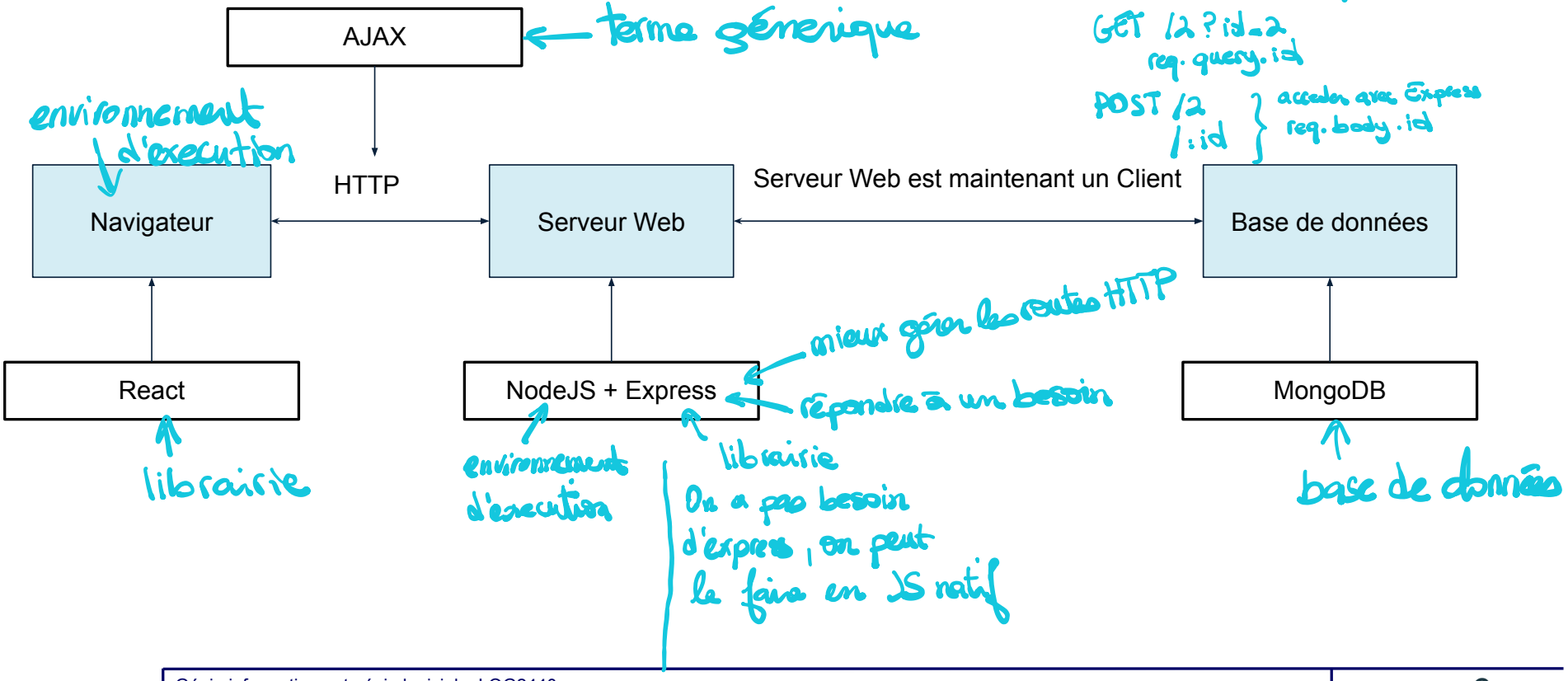


Client

1 serveur

Résumé de la matière : technologies

/:id → GET /?id=2 indefined
req.params.id
/2
/:id :id



Exercices

- Qu'est-ce une requête HTTP idempotente ?
  - ✓ La même requête peut être envoyée plusieurs fois au serveur sans modifier plusieurs fois son état.
  - La requête modifie toujours l'état du serveur.
  - La requête est toujours suivie d'une réponse HTTP. Toute les requête
  - La requête ne modifie pas l'état du serveur.
  - La requête est envoyée avec la méthode GET ou DELETE seulement.

Exercices

- Selon la **sémantique de HTTP**, quelles méthodes peuvent modifier l'état du serveur ?
  - ○ GET
  - ✔ POST
  - ✔ PATCH
  - ✔ PUT
  - ✔ DELETE
  - ○ OPTIONS
  - ○ HEAD même chose que GET mais ne renvoi pas de corps

Exercices

- Vous pouvez supprimer les X ressources les moins utilisées de votre service web à travers une requête DELETE sur la route /deleteLeastUsed?nb=X. Une suppression réussie retourne le code 204 et les identifiants des ressources supprimées.

1. Quel est le problème(s) avec cette approche ? Comment le(s) corriger ?
2. Quel serait le bon code de retour si X est invalide ? Donnez 2 exemples de X invalide

1) problème de sécurité
204 : il y a pas de contenu / à la place on peut envoyer 200
ms on envoi du contenu
- écrire juste /leastUsed
- DELETE n'est pas idempotente donc les données vont changer d'état plusieurs fois
Tant que X est valide on change

2) X != chiffre (pas not found dans ce cas)
X < 0
X > nb de ressources
(code de retour sera 400 Bad Request)

Exercices

- Refaites la requête XHR suivante en utilisant fetch

```
const xhr = new window.XMLHttpRequest();
xhr.open("POST", "https://jsonplaceholder.typicode.com/posts");
xhr.setRequestHeader("Content-Type", "application/json");
const data = { titre: "Message de XHR", contenu: "allo" };
xhr.addEventListener("readystatechange", function () {
  if (xhr.readyState === window.XMLHttpRequest.DONE) {
    const response = JSON.parse(xhr.responseText);
    console.log(response);
  }
});
xhr.send(JSON.stringify(data));
```

Différence entre xhr et Fetch

- Fetch est plus récente
- les deux font des appels async
- Fetch basé sur les promesses
  xhr basé sur les evenements

xhr donne 400 ou 500
Fetch ne rejete pas la promesse donc c'est à nous de verifier si statut != 200, 300

- xhr permet de suivre le progrès de televersement alors que Fetch non.

Exercices

- Refaites la requête XHR suivante en utilisant fetch avec then

```
const data: {titre: "Message de Fetch", contenu: "allo avec Fetch"};
const options = {
  method: 'POST'
  headers: {'Content-Type' : "application/json", },
  body: Json.stringify(data), };
const url = "https://jsonplaceholder.typicode.com/posts";
fetch (url, options)
    .then ((response) => response.json())
    .then ((json) => console.log (json));
```

Exercices

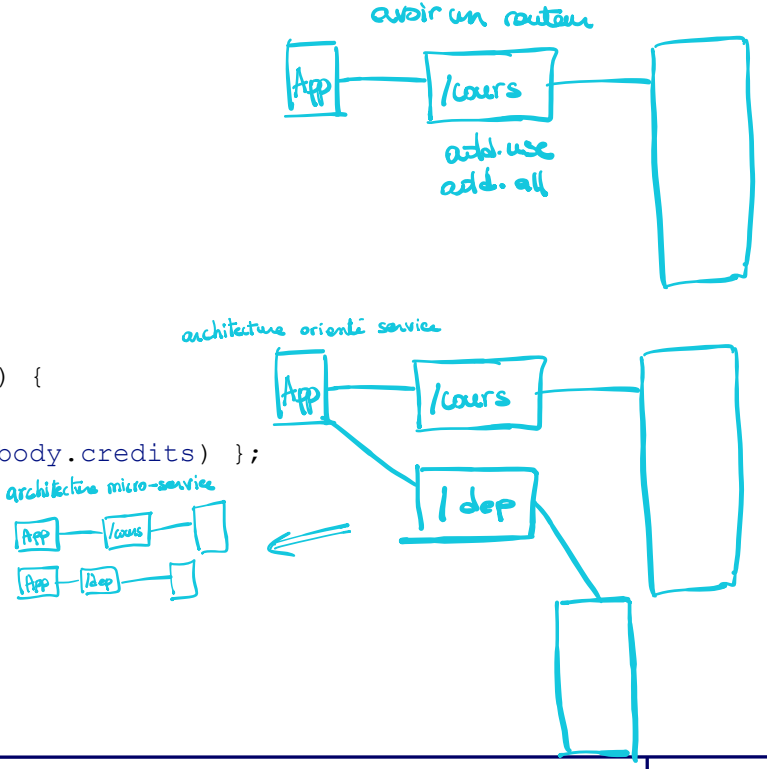
- Refaites la requête XHR suivante en utilisant fetch avec async/await

```
const fetch_request = async () => {
  const data: {titre: "Message de Fetch", contenu: "allo avec Fetch"};
  const options = {
    method: 'POST'
    headers: {'Content-Type' : "application/json", },
    body: Json.stringify(data), };
  const url = "https://jsonplaceholder.typicode.com/posts";
   const response = await fetch (url, options)
   const content = await response.json()
     console.log(content)
  }
```

Exercices

- Quel est le problème avec le service web REST suivant (Exemple AJAX du cours)

```
app.get("/obtenirCours", function (req, res) {
 res.send(listeDeCours);
});
app.get("/obtenirCours/:sigle", function (req, res) {
 const cours = listeDeCours.find((c) => {
   return c.sigle === req.params.sigle;
 });
 res.send(cours);
});
app.post("/ajouterCours", urlencodedParser, function (req, res) {
 if (!req.body) return res.sendStatus(400);
 const cours = { sigle: req.body.sigle, credits: parseInt(req.body.credits) };
 listeDeCours.push(cours);
 res.status(201).send(cours.sigle + " a été ajouté");
});
```



Exercices

- Quel est le problème avec le service web REST suivant (Exemple AJAX du cours)

```
app.delete("/supprimerCours/:sigle", function (req, res) {
 const taille = listeDeCours.length;
 listeDeCours = listeDeCours.filter((c) => c.sigle !== req.params.sigle);
 if (taille > listeDeCours.length) res.send("Cours supprimé."); ← envoi 200 par défaut
 else res.status(400).send("Echec de suppression : cours introuvable dans la liste");
});

app.patch("/modifierCours/", urlencodedParser, function (req, res) {
 const cours = listeDeCours.find((c) => {
   return c.sigle === req.body.sigle;
 });                                    ← 404 au lieu de 400
 if (!cours) return res.status(400).send("Ce cours n'existe pas");
 cours.credits = req.body.credits;
 res.send("Cours modifié");
});
```

Exercices

- Quel est le problème avec le service web REST suivant (Exemple AJAX du cours)

avoir un router pour les différents cours grâce à app.use et d'enlever la description de la requête dans l'URI et de juste mettre /cours

Exercices

- Quel est le rôle des lignes suivantes ?

```
const app = express();
app.use(cors());
const urlencodedParser = express.urlencoded({ extended: false });
app.use(express.json());
```

← crée une instance de express
← avoir accès aux ressources du serveurs
← Middleware natif de Express
Middleware pour la gestion du corps de requête JSON actif

- Quelle est la différence entre *urlencoderParser* et *express.json()* dans ces 2 routes ?

```
app.post("/ajouterCours", urlencodedParser, function (req, res) {
 if (!req.body) return res.sendStatus(400);
 const cours = { sigle: req.body.sigle, credits: parseInt(req.body.credits) };
 listeDeCours.push(cours);
 res.status(201).send(cours.sigle + " a été ajouté");
});
app.get("/obtenirCours", function (req, res) {
 res.send(listeDeCours);
});
```

express.json() pour les deux
urlencoded Parser va être appelé que dans la requête post

Exercices

- Pourquoi, dans le cas d'un système distribué, parle-t-on d'une architecture **N-niveaux** plutôt que 3-niveaux (*three tiers*) ?

certains des 3 niveaux sont partagés par plusieurs sous-systèmes

- Donnez 2 exemples où nous avons **N-niveaux** .

Moodle : niveau de présentation, traitement du côté client, plusieurs traitements dans la base de données.

Client-Serveur
traitements divisé sur le client et le serveur
présentation divisé sur le client et le serveur avec du rendu côté serveur (SSR)
système dont le traitement est fait par plusieurs microservice spécialisés

Exercices

Votre base de données MongoDB contient des informations sur des sentiers de randonnée. Chaque document possède un attribut "**distance**" qui représente la distance du sentier en kilomètres. Voici 2 manières d'obtenir tous les sentiers dont la distance est de plus de 5km.

Y-a-t-il une manière qui est à privilégier ? Si oui, laquelle et pourquoi? Si non, pourquoi ?

✓ #1:
```
const data = db.myCollection.find( { distance: { $gt: 5 } } ).toArray();
```

#2:
```
const data = db.myCollection.find( { } ).toArray().filter( ( x ) => x.distance > 5 );
```

puisque le traitement est fait directement sur la BD et évite de copier la collection au complet du côté serveur.

Exercices

- Les fonctions du *Driver* de MongoDB pour NodeJS retournent des Promesses. **Pourquoi** ?

pour pas bloquer la boucle d'evenement de NodeJS

- MongoDB est accessible qu'à partir d'un serveur non à partir d'un site web. **Pourquoi** ?

question de sécurité

offre pas une communication HTTP : il faut utiliser des adaptateurs

Exercices

- Quelle fonctionnalités supplémentaires peut-on avoir si on développe notre application React en utilisant la syntaxe JSX vs la syntaxe de JS/HTML natifs ?

aucune, on peut faire la même chose avec les 2.

- Quelle est la différence entre une composante fonctionnelle et une composante de classe ? Y-a-t-il une manière de faire qui offre plus de fonctionnalités que l'autre ?

pas d'état

il est possible d'avoir une gestion de l'état avec hook, notamment useState ce qui rend les composantes fonctionnelles au même niveau que les composantes de classes en terme de fonctionnalités

Exercices

- Dans quels cas utilise-t-on *useEffect* dans les composantes fonctionnelles de React ?
  - ✓ Modification du DOM directement à travers les méthodes natives
  - ✓ Déclenchement d'un code suite à la modification de l'état d'une composante
  - ○ Modification de l'état d'une composante
  - ✓ Communication avec un serveur à l'initialisation d'une composante
  - ○ Communication avec un serveur suite à un événement du DOM (onClick)