FORMATION DEV WEB WEB MOBILE
CCP 2 - DÉVELOPPER LA PARTIE BACK-END D'UNE APPLICATION WEB

NODE.JS

JOUR 3 : GESTION D'ERREURS ET MIDDLEWARE

PRESENTE PAR NATACHA DESSE

AFEC

DEROULE MODULE

- Découverte de Node.JS
- Développer une API REST avec Express
- Gestion d’erreurs et middleware
- Authentification et JWT
- Routing et Accès
- **Documentation dans un projet web**

GESTION D'ERREURS ET MIDDLEWARE

LES OBJECTIFS

Comprendre le rôle des middlewares dans Express et les utiliser correctement.

Savoir capturer et renvoyer des erreurs avec les bons statuts Http.

Centraliser et personnaliser la gestion des erreurs dans une API.

Créer des middlewares pour la sécurité, le logging ou la validation

NUAGE DE MOTS

Qu’avez vous retenu des API ?

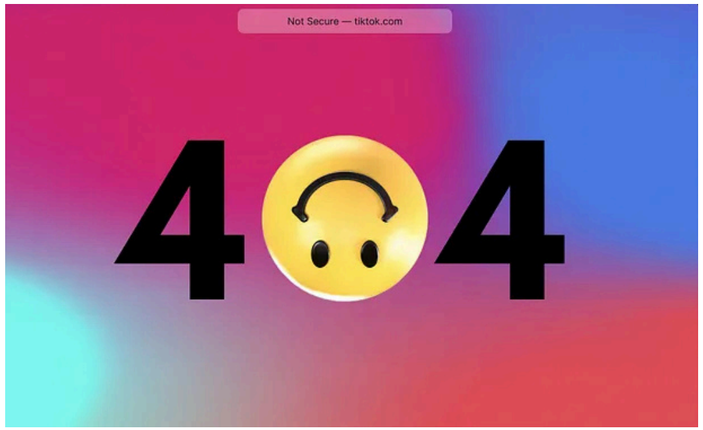
https://www.menti.com/aldpp4773piz

LA GESTION D'ERREUR

POURQUOI GERER LES ERREURS ?

Dans une API, tout peut potentiellement échouer :

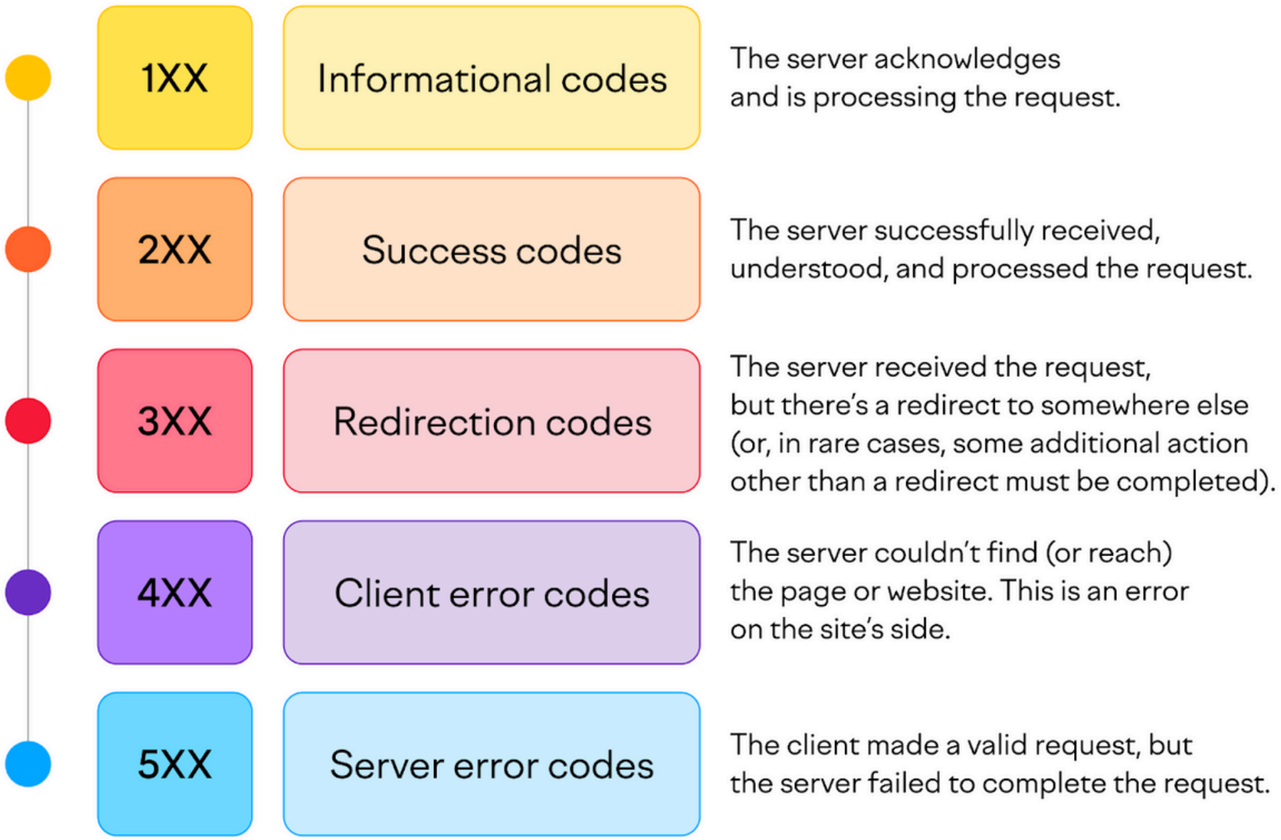
- L'utilisateur envoie une **mauvaise requête**
- Une **donnée** est **manquante**
- Une ressource n'**existe pas**
- Le **serveur** a un **bug**
- Une opération sur un fichier **échoue**



Une bonne gestion des erreurs permet de :

- **Informer** clairement le client
- Renvoyer le bon **code HTTP**
- Garder un serveur **robuste** et **stable**

COMPRENDRE LES CODE ERREURS API



TYPES D’ERREURS COURANTES

Erreur	Statut HTTP	Exemple
Requête invalide	400 Bad Request	Body manquant ou mal formé
Ressource non trouvée	404 Not Found	Film avec un ID inexistant
Conflit ou duplication	409 Conflict	ID déjà utilisé
Erreur serveur	500 Internal Server Error	Problème avec fs, bug inattendu

LES CODES 1XX - INFORMATIONS

Status Code	Function
1XX — Informational	
100	Continue
101	Switching Protocols
102	Processing
103	Early Hints

LES CODES 2XX - SUCCESS

200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

LES CODES 3XX - REDIRECTION

300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
307	Temporary Redirect
308	Permanent Redirect

LES CODES 4XX - CLIENT ERROR

400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Content Too Large
414	URI Too Long

415	Unsupported Media Type
416	Range Not Satisfiable
417	Expectation Failed
421	Misdirected Request
422	Unprocessable Content
423	Locked
424	Failed Dependency
425	Too Early
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
451	Unavailable for Legal Reasons

LES CODES 5XX - SERVER ERROR

500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
511	Network Authentication Required

BONNES PRATIQUES

- Toujours vérifier les entrées

```
if (!req.body.titre) {
  return res.status(400).json({ message: "Le champ 'titre' est requis." });
}
```

- Vérifier l'existence avant d'agir

```
const film = films.find(f => f.id === filmId);
if (!film) {
  return res.status(404).json({ message: "Film non trouvé." });
}
```

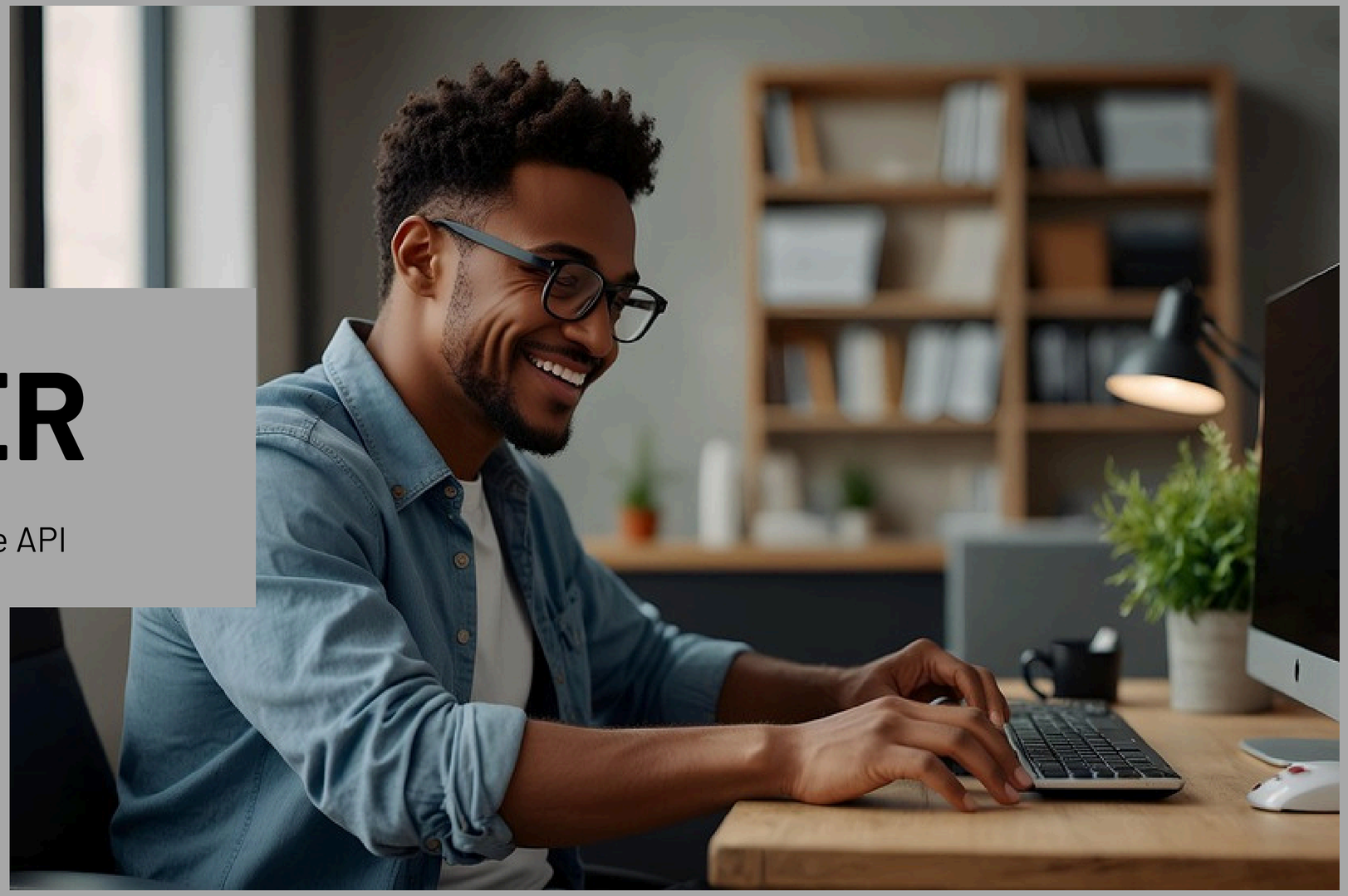
UTILISER TRY/CATCH

- Pour les opérations suceptibles de crasher et rendre des erreurs, utiliser les blocs try / catch

```
try {
  fs.writeFileSync('chemin.json', JSON.stringify(films));
  res.status(201).json({ message: "Film ajouté avec succès." });
} catch (error) {
  res.status(500).json({ message: "Erreur lors de l'enregistrement du fichier." });
}
```

ATELIER

Gestion d’erreur sur notre API



GESTION D'ERREUR SUR CINECLUB

https://github.com/dessna12/02-Server-Express

🎯 Enoncé

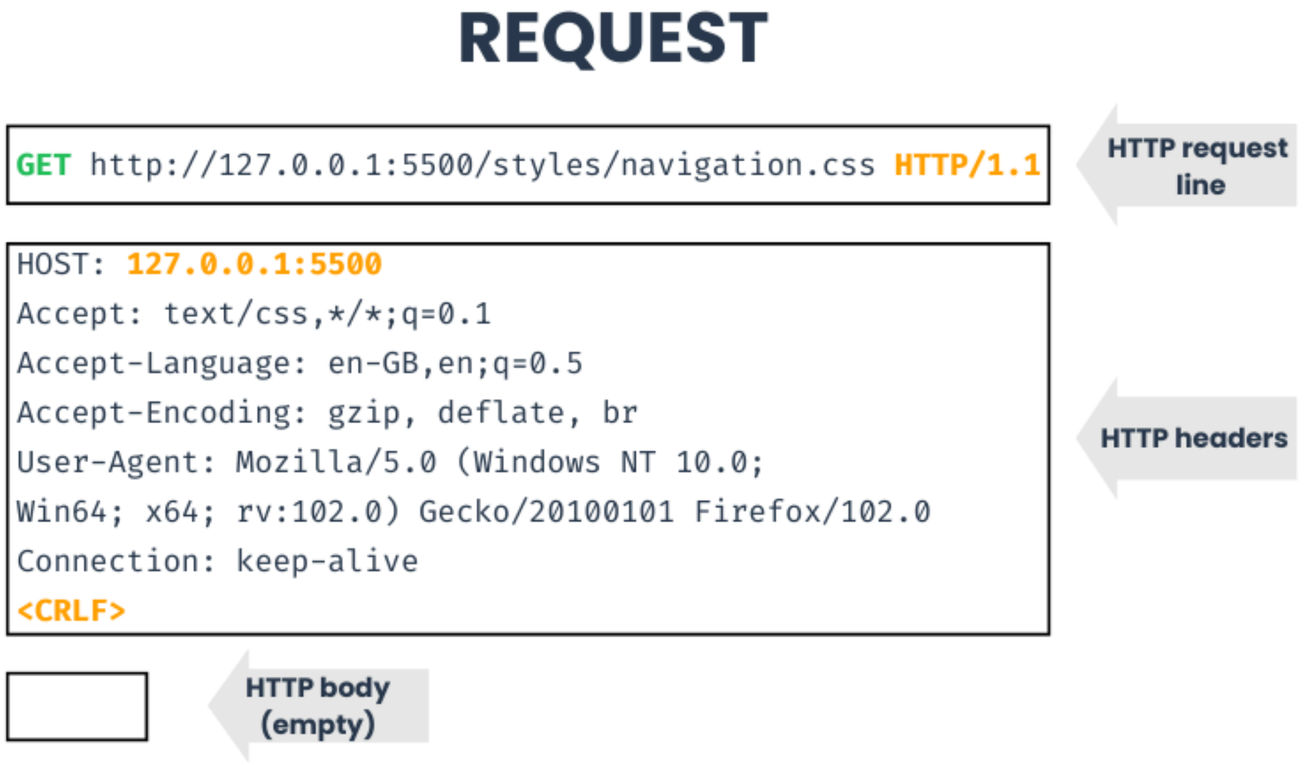
Dans notre API, il y a plusieurs cas d'erreurs qui peuvent arriver qui n'ont pas été gérés.

Ta mission si tu l'acceptes :

- Liste les erreurs qui peuvent survenir lors de nos appels
- Renvoie les bons code d'erreurs HTTP lorsque les erreurs sont levées
- Entoure de blocs try / catch pour les opérations synchrones suceptibles de planter.

LES REQUETES HTTP

A QUOI RESSEMBLE UNE REQUETE HTTP ?



LES METHODES DE REQUETES ACCESSIBLES SUR EXPRESS

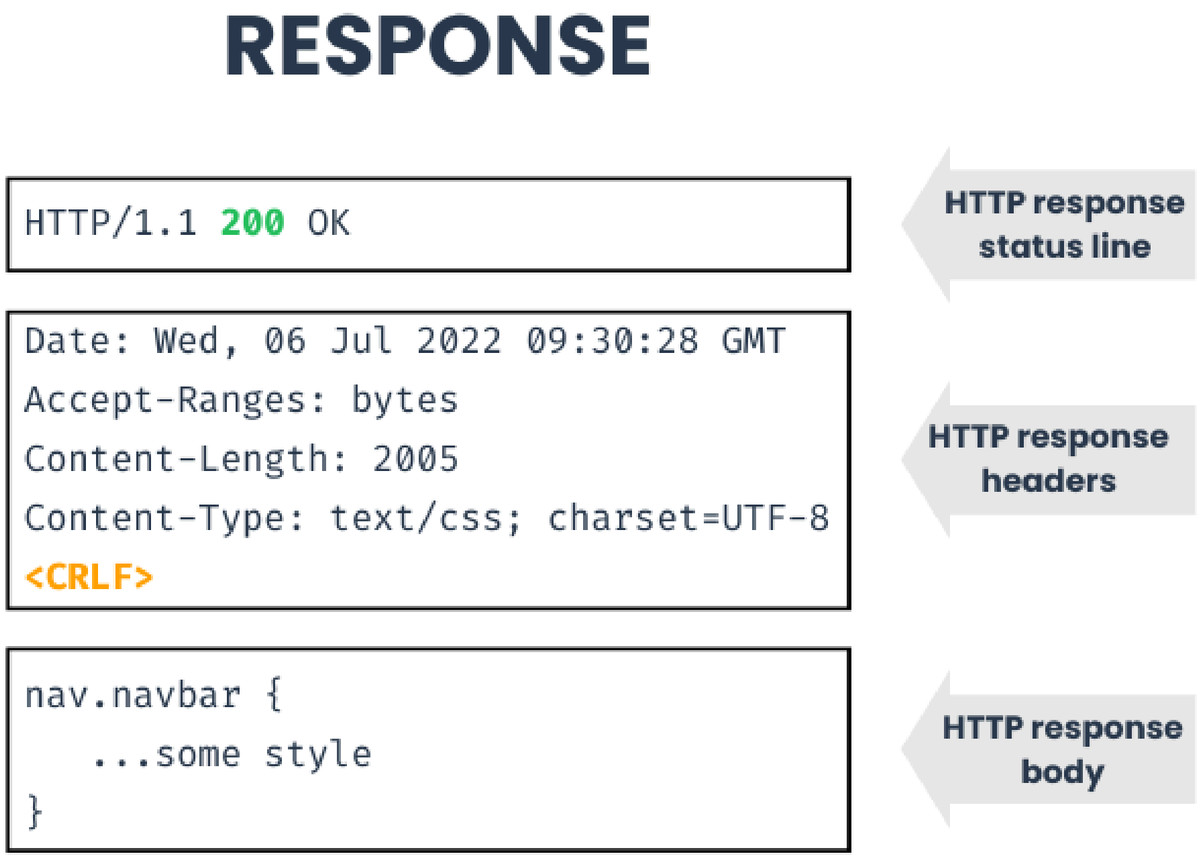
req....	**Contenu**
method	Méthode HTTP (GET, POST, etc.)
url	URL brute
params	Paramètres dynamiques dans l'URL (:id)
query	Paramètres après ? (query string)
headers	Métadonnées envoyées par le client
body	Données envoyées (POST, PUT...)

LES ELEMENTS D'UNE REQUETE

Une requête contient plusieurs éléments accessibles via l'objet req dans Express

```
app.post('/utilisateurs/:id', (req, res) => {
  console.log(req.method);        // 'POST'
  console.log(req.url);           // '/utilisateurs/42'
  console.log(req.params);        // { id: '42' }
  console.log(req.query);         // { tri: 'nom' } (si ?tri=nom)
  console.log(req.headers);       // objet avec les headers HTTP
  console.log(req.body);          // { nom: 'Alice' } (si JSON envoyé)
}
```

LES REPONSES HTTP

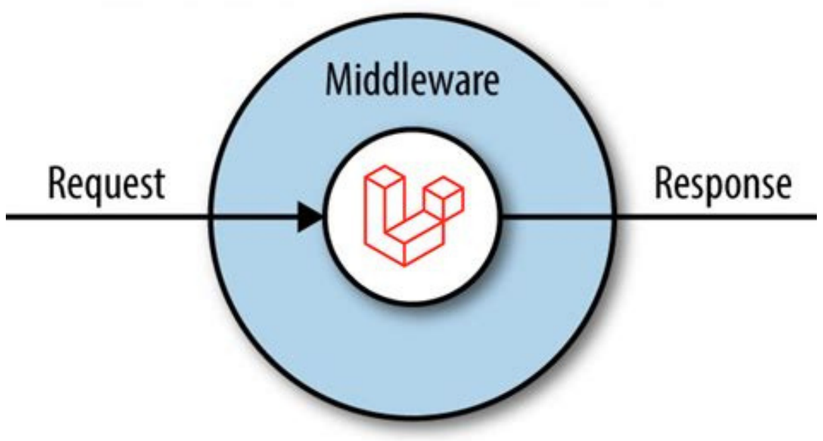


LES MIDDLEWARE

QU'EST CE QU'UN MIDDLEWARE ?

Définition

Un **middleware** est une fonction qui **intercepte** une requête HTTP (request) avant qu'elle n'arrive à sa destination finale.



Utilité

Les middlewares permettent de réutiliser du code pour :

- Sécuriser les routes (ex. : vérifier un token d'authentification)
- Nettoyer ou enrichir la requête (ex. : parser un JSON, ajouter un utilisateur à req)
- Journaliser les accès (ex. : logger la date, l'URL, etc.)
- Gérer les erreurs de manière centralisée
- Valider les données d'entrée (body, params...)

COMMENT CA MARCHE ?

Définition

Un middleware est une fonction qui prend 3 paramètres :

- **req** : l'objet représentant la requête
- **res** : l'objet représentant la réponse
- **next** : une fonction à appeler pour passer au middleware suivant

(req, res, next) => { ... }

```
1  const express = require('express');
2  const app = express();
3
4  // Un middleware maison
5  app.use((req, res, next) => {
6    console.log(`${req.method} ${req.url}`);
7    next(); // Passe au middleware ou à la route suivante
8  });
9
10 app.get('/', (req, res) => {
11   res.send('Hello, world!');
12 });
```

Exemple de middleware fait maison

CYCLE DE VIE D'UN MIDDLEWARE

Une requête HTTP passe par tous les middleware dans
l'ordre où ils sont déclarés, jusqu'à arriver au route handler.

```
1 app.use(middleware1)
2 app.use(middleware2)
3 app.get('/', routeHandler)
```

TYPES DE MIDDLEWARES

Globaux

appliqués à toutes les routes (app.use(...))

Spécifique à une route

(app.get('/route', middleware, handler))

Middleware d’erreur

avec 4 paramètres (err, req, res, next)

MIDDLEWARE GLOBAUX

Utilisé avec **app.use( )** ou une méthode comme app.get( ). Ils sont **disponible pour toute l'application.**

Le middleware app.use(express.json( )) dans Express **sert à analyser automatiquement le corps des requêtes HTTP contenant du JSON** (c'est-à-dire les requêtes avec Content-Type: application/json) et à rendre ce contenu accessible via req.body.

```
app.use(express.json())  // corps JSON
app.use(middlewarePerso)

```

Sans express.json( ), req.body sera undefined car Express ne sait pas comment interpréter le corps de la requête.

```
app.post('/api/users', (req, res) => {
  console.log(req.body); // undefined !
});
```

MIDDLEWARE DE ROUTAGE

Un middleware de routage est un middleware **appliqué à une ou plusieurs routes spécifiques**, plutôt qu'à toutes les requêtes de l'application (contrairement à app.use() global).

Il est placé entre l'URL et la fonction de réponse, et ne s'exécute que si la route correspond.

Syntaxe de base :

```
app.get('/chemin', middleware, (req, res) => {
  res.send('Réponse');
});
```

On peut chainer plusieurs middleware

```
app.post('/users', middleware1, middleware2,
(req, res) => {
  res.send('Nouvel utilisateur');
});
```

Un middleware de routage peut :

- filtrer l'accès à une route (authentification, rôle, etc.)
- valider les données envoyées (body, params...)
- logger ou traquer certaines actions
- injecter des données dans req à utiliser dans le handler final

MIDDLEWARE D'ERREUR

Un middleware d'erreur dans Express est une fonction spéciale conçue pour **intercepter les erreurs** survenues dans une route ou dans un autre middleware. **A placer à la fin des autres middlewares et routes.**

Syntaxe de base :

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ message: 'Erreur serveur' });
});
```

Comment déclencher une erreur ?

Manuellement avec next (error)

```
app.get('/boom', (req, res, next) => {
  const err = new Error('Quelque chose a explosé');
  next(err); // saute les autres middleware, va direct au middleware d'erreur
});

```

Avec un throw Error et try/catch

```
app.get('/users/:id', async (req, res, next) => {
  try {
    const user = await getUserFromDB(req.params.id);
    if (!user) throw new Error('Utilisateur introuvable');
    res.json(user);
  } catch (err) {
    next(err); // on transmet l'erreur
  }
});

```

EXEMPLE DE MIDDLEWARE D'ERREUR

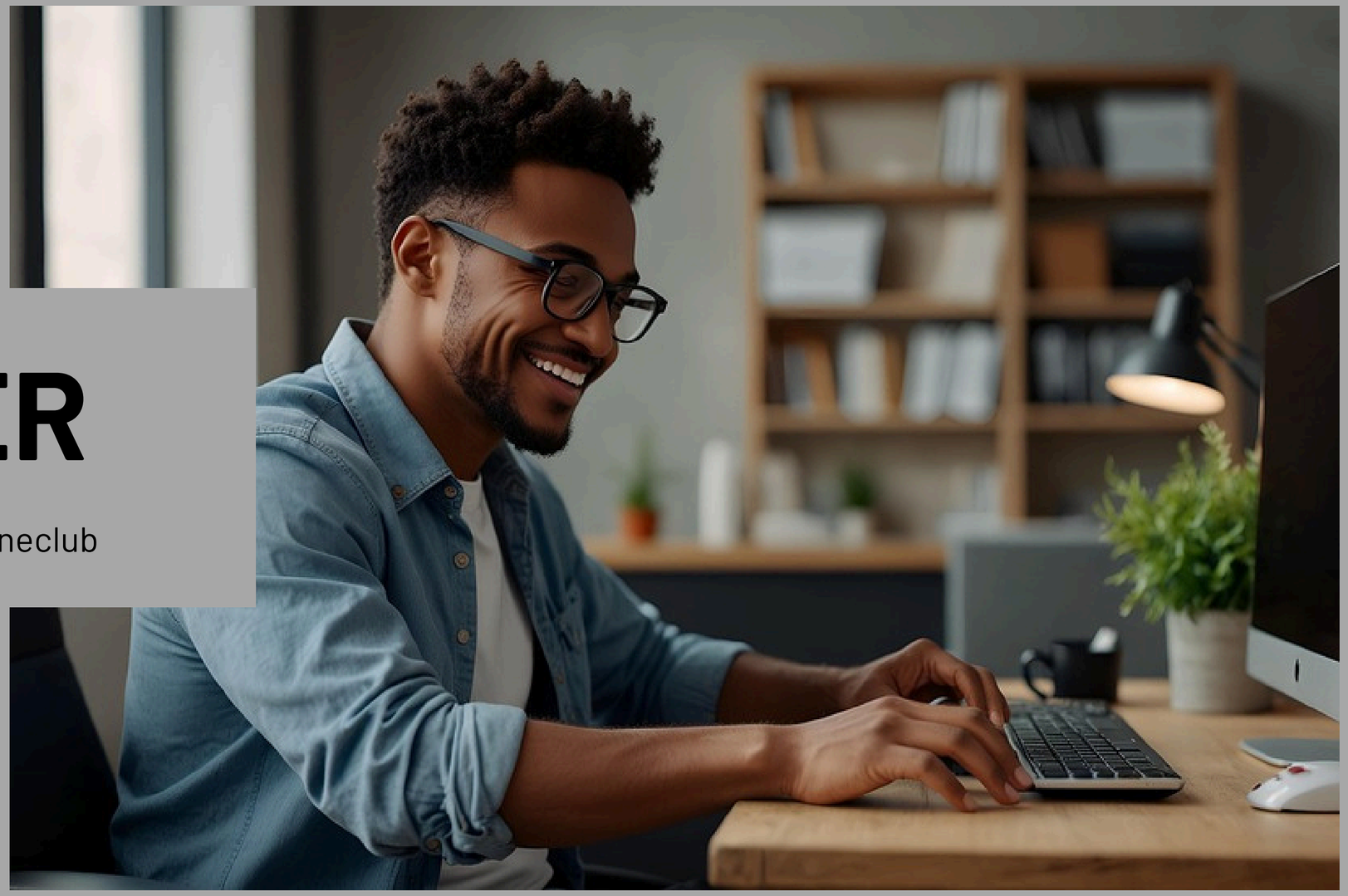
```
app.use((err, req, res, next) => {
  if (err.status === 404) {
    return res.status(404).json({ message: err.message || 'Ressource non trouvée' });
  }

  console.error(err.stack);
  res.status(500).json({ message: 'Erreur interne du serveur' });
});

```

ATELIER

Ajout de middleware sur cineclub



MIDDLEWARE SUR CINECLUB

https://github.com/dessna12/02-Server-Express

🎯 Enoncé

Dans cette deuxième partie nous allons maintenant implémenter des middlewares pour mieux gérer les cas d'erreurs qui peuvent arriver

1. Implémenter un middleware au début pour vérifier que les id sont bien des Int
2. Implémenter un middleware d'erreur au début pour centraliser les erreurs
3. Implémenter un middleware à la fin pour attraper toutes les requêtes qui n'ont pas de routes.

QUIZZ