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

# NODE.JS

## JOUR 5 : ROUTING ET ACCES

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

# ROUTING ET ACCES

# LES OBJECTIFS

Comprendre le fonctionnement du routage dans Express.js

Mettre en place des routes protégées à l'aide de JSON Web Tokens (JWT)

Créer un middleware d'authentification pour sécuriser les routes sensibles.

Gérer les rôles utilisateurs pour contrôler l'accès aux différentes parties de l'application.

# NUAGE DE MOTS

Qu’avez vous retenu des JWT ?

https://www.menti.com/aldpp4773piz

# ROUTING

# QUEL EST LE SOUCIS ?

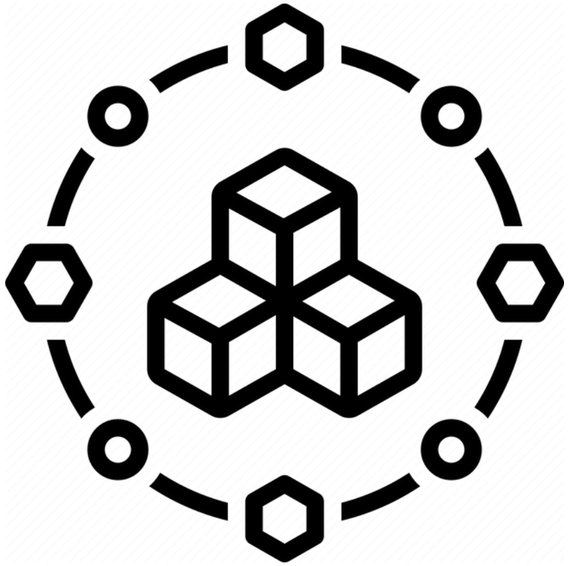
Pour le moment nous avons créé nos routes en mettant **tout** dans **un fichier** index.js

Dès lors que nous avons une application qui grandit il va falloir organiser cet index.js de façon **modulaire** et exporter les routes dans un **module spécifique** car sinon le fichier devient beaucoup trop grand, dur à maintenir et difficilement lisible.

La modularité consiste à organiser son code en **plusieurs petits fichiers**.

Cela nous permet :

- une meilleur **maintenabilité**
- une **réutilisation** des modules
- des **tests** plus faciles à réaliser



# PRINCIPE D'UN ROUTER

Pour les routes nous allons nous aider de la méthode de Router de express pour organiser nos routes.

Express.js utilise un système de routage pour gérer les différentes requêtes HTTP. Chaque route est associée à une méthode HTTP (GET, POST, PUT, DELETE, etc.) et à un chemin spécifique.

Pour le moment on écrit nos routes comme ça :

```
app.get('/home', (req, res) => {
  res.send('Bienvenue sur la page d\'accueil');
});t('/', (req, res) => {
  res.send('Hello, world!');
});
```

# SYNTAXE DU ROUTER DE EXPRESS

index.js

```
app.use([chemin], middlewareFonction)
```

- **chemin** (optionnel) : le chemin de base pour lequel le middleware sera appliqué. Si omis, le middleware sera appliqué à toutes les routes.
- **middlewareFonction** : la fonction middleware ou le routeur à exécuter.

route/admin.js

```
// routes/admin.js
const express = require('express');
const router = express.Router();

router.get('/dashboard', (req, res) => {
  res.send('Tableau de bord admin');
});

module.exports = router;

```

# UTILISATION DU ROUTER

Modularité dans notre code

routes/users.js

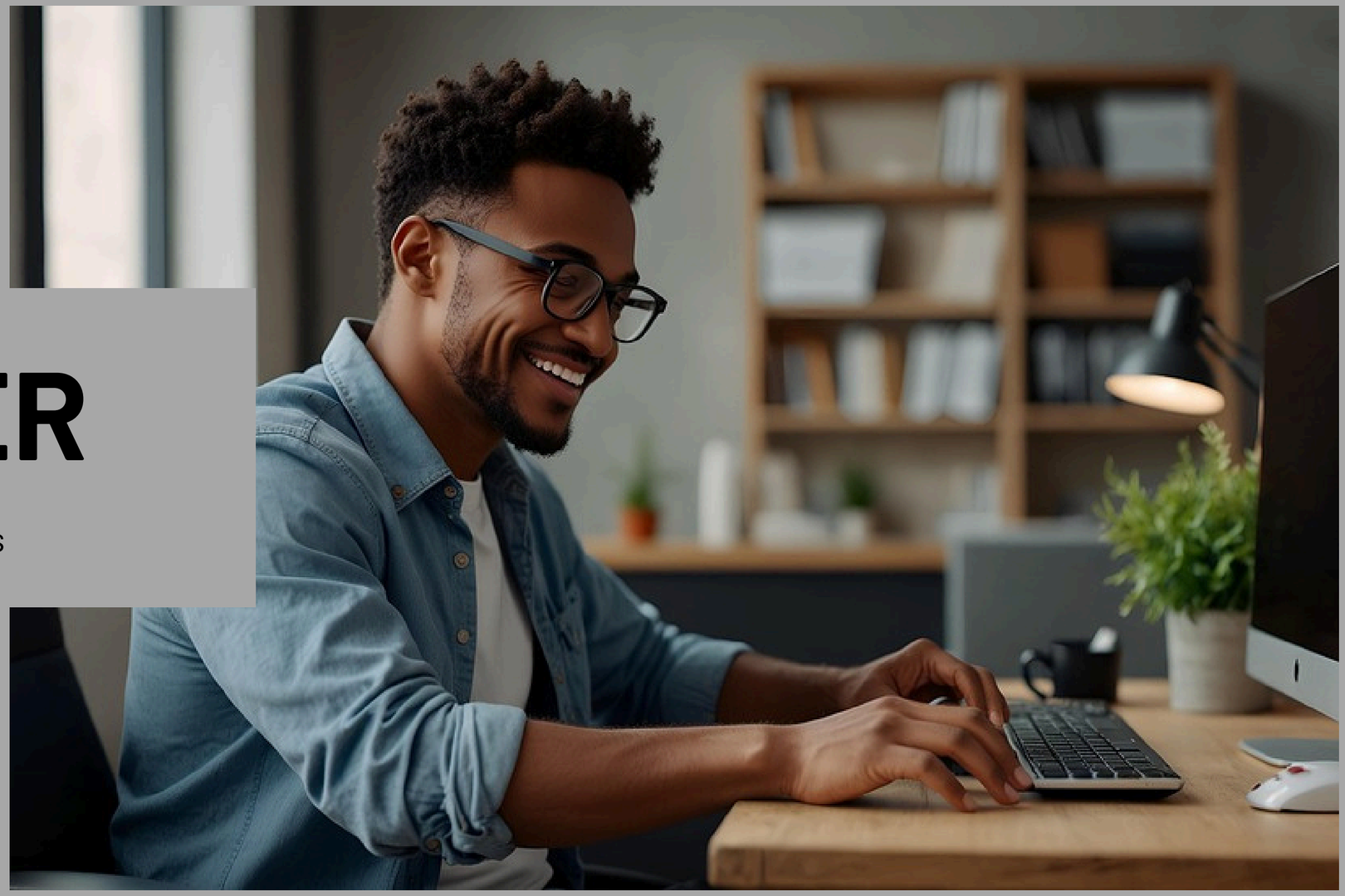
```
1 const express = require('express');
2 const router = express.Router();
3
4 router.get('/profile', (req, res) => {
5   res.send('Page de profil');
6 });
7
8 module.exports = router;
```

index.js

```
1 const profileRoutes = require('./routes/
  profile');
2 app.use('/user', profileRoutes);
```

# ATELIER

Factoriser les routes



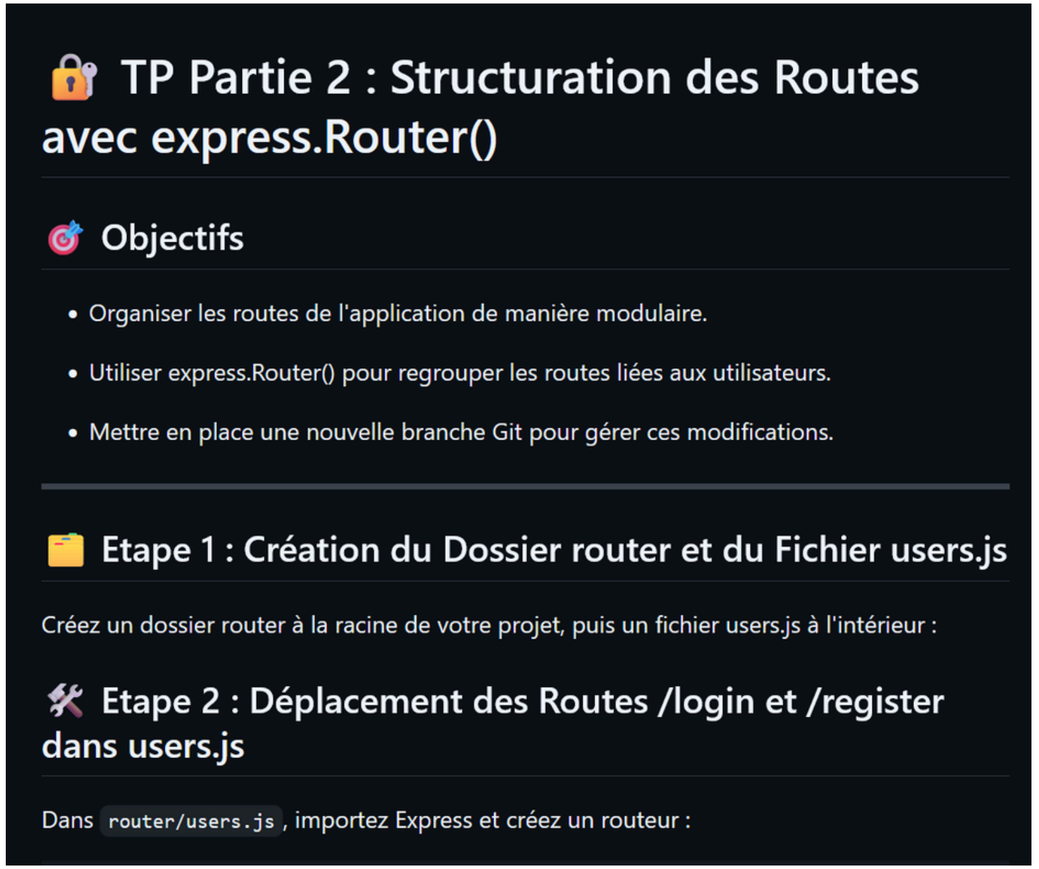
# CONSIGNES

https://github.com/dessna12/04-JWT

Nous allons reprendre notre TP d'authentification et créer un dossier / router dans lequel nous allons créer un fichier users.js qui va regrouper l'ensemble de nos routes

- POST / login
- POST / register

Créer une nouvelle branche feat_router, réaliser les modifications et poussez sur vos branches



## 🔐 TP Partie 2 : Structuration des Routes avec express.Router()

### 🎯 Objectifs

- Organiser les routes de l'application de manière modulaire.
- Utiliser express.Router() pour regrouper les routes liées aux utilisateurs.
- Mettre en place une nouvelle branche Git pour gérer ces modifications.

---

### 🗂️ Etape 1 : Création du Dossier router et du Fichier users.js

Créez un dossier router à la racine de votre projet, puis un fichier users.js à l'intérieur :

### 🛠️ Etape 2 : Déplacement des Routes /login et /register dans users.js

Dans `router/users.js`, importez Express et créez un routeur :

# ACCES ROUTES PAR TOKEN

# MIDDLEWARE

Nous avons pour le moment vu comment chiffrer un mot de passe et le vérifier avec son mais nous n'avons pas mis en place un système d'accès aux routes avec vérification du token.

Nous allons pour cela créer un middleware authenticateToken()

Ce middleware peut ensuite être appelé dans les routes pour protéger leur accès :

```
app.get('/dashboard', authenticateToken, (req, res) => {
  res.send(`Bienvenue ${req.user.email} sur votre tableau de bord`);
});
```

```
const jwt = require('jsonwebtoken');
const SECRET_KEY = 'votre_clé_secrète';

function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1]; // Format: "Bearer TOKEN"

  if (!token) return res.status(401).send('Token manquant');

  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return res.status(403).send('Token invalide');
    req.user = user;
    next();
  });
}
```

# GESTION DES ROLES ET UTILISATEURS

Pour contrôler l'accès à certaines routes en fonction du rôle de l'utilisateur (par exemple, admin, utilisateur standard), on peut inclure le rôle dans le payload du JWT lors de la connexion :

```
const token = jwt.sign({ email: user.email, role: user.role }, SECRET_KEY, { expiresIn: '1h' })
```

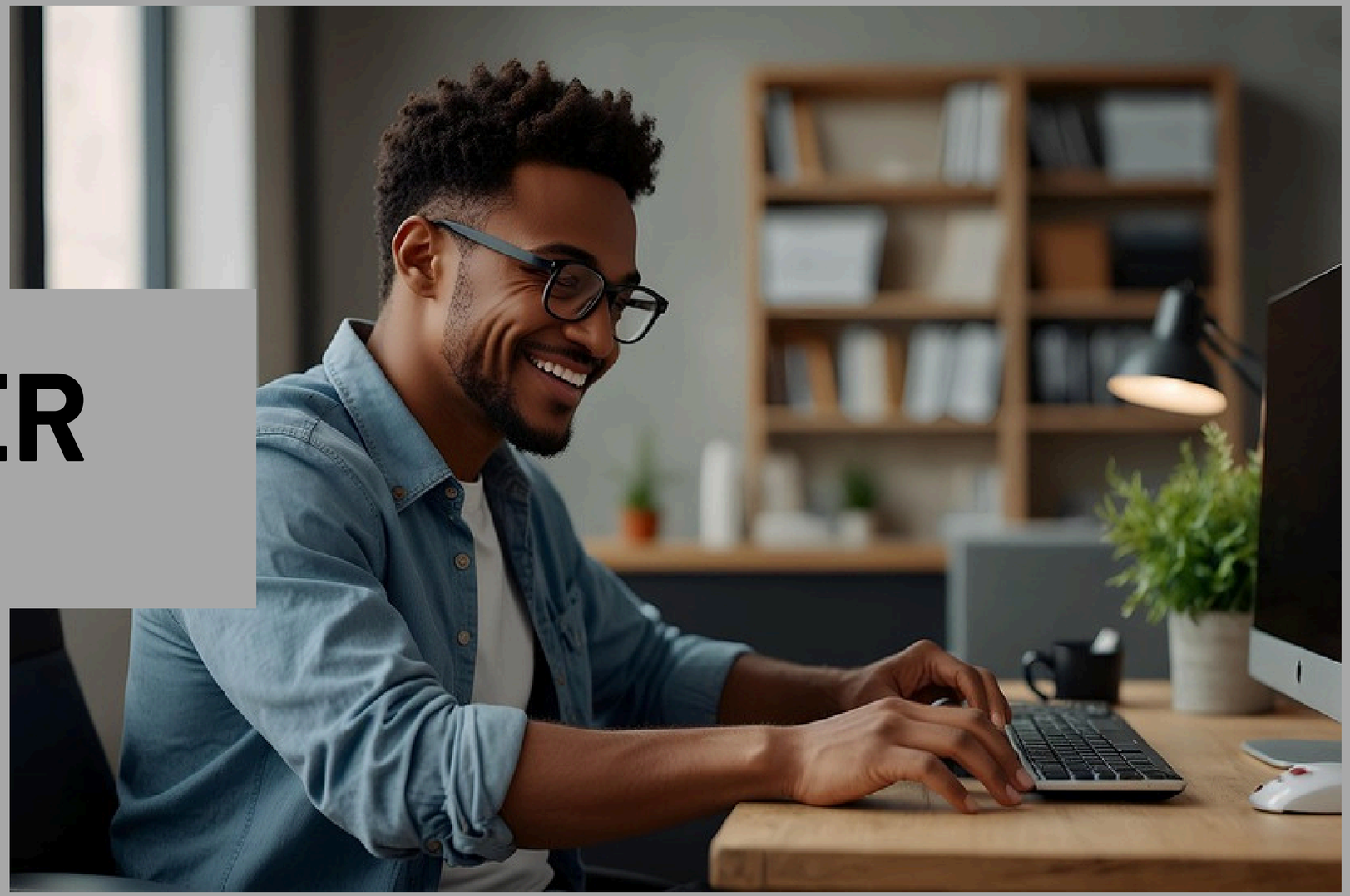
Ensuite, dans notre middleware ou directement dans les routes, on peut peut vérifier le rôle :

```
function authorizeRoles(...allowedRoles) {
  return (req, res, next) => {
    if (!allowedRoles.includes(req.user.role)) {
      return res.status(403).send('Accès interdit');
    }
    next();
  };
}

// Utilisation dans une route
app.get('/admin', authenticateToken, authorizeRoles('admin'), (req, res) => {
  res.send('Bienvenue dans la zone admin');
});
```

# ATELIER

Cineclub



# CONSIGNES

https://github.com/dessna12/05-Routes

**On commence à organiser mieux notre code !**

Nous allons implémenter un router et une vérification de token à l'aide d'un middleware sur notre projet cineclub

## 🎬 CineClub API – Partie 3 : Structuration & sécurisation des routes

### 🎯 Objectifs

- Organiser les routes avec `express.Router()`
- Déplacer la logique films dans un fichier dédié
- Appliquer un middleware de validation d'ID
- Intégrer ce routeur dans l'application principale

### 🛠️ Étape 1 : Créer le routeur

1. Crée un dossier `router/`
2. Crée un fichier `router/films.js`

### ✍️ Étape 2 : Créer le routeur Express

À l'intérieur de `films.js` :

- Importe `express`, `fs`, et `path`
- Initialise un `Router()` avec `express.Router()`
- Charge le fichier JSON des films (avec `require` ou `fs.readFileSync`)
- Prépare un middleware **local** pour valider `req.params.id`

*(Pas besoin d'écrire tout de suite le code complet — commence par la structure.)*

# QUIZZ