# Fichier : 05 - routes_ai.js

# Tutoriel Node.js : Gestion des requetes avec Express et Dotenv

Dans ce tutoriel, nous allons decomposer un bloc de code Node.js qui utilise les modules `express` et `dotenv`, ainsi que des services et des mocks specifiques pour gerer les requetes.

## Code complet

```
import express from 'express';
import dotenv from 'dotenv';

import chatgptMock from '../mock/llm/chatgpt.mock.js';
import claudeMock from '../mock/llm/claude.mock.js';
import chatgptReal from '../services/llm/chatgpt.service.js';
import claudeReal from '../services/llm/claude.service.js';

dotenv.config();

const router = express.Router();
const useMock = process.env.USE_MOCK === 'true';

function isUnauthorizedError(message) {
  return message.includes('unauthorized') || message.includes('401');
}

function getProvider(llm) {
  const providers = {
    chatgpt: {
      mock: chatgptMock,
      real: chatgptReal,
    },
    claude: {
      mock: claudeMock,
      real: claudeReal,
    },
  };

  return providers[llm] || null;
}

async function handleLLMRequest(type, llm, data) {
  const provider = getProvider(llm);
  if (!provider) { return { error: 'unknown-provider' }; }

  const fn = useMock ? provider.mock : provider.real;

  return { data: await fn(type, data) };
}
```

```
router.post('/:type/:llm', async (req, res) => {
  const { type, llm } = req.params;
  const input = req.body;

  try {
    const { data, error } = await handleLLMRequest(type, llm, input);

    if (error) {
      return res.status(400).json({ success: false, llm: llm, data: error });
    }

    return res.json({ success: true, llm: llm, data: data });

  } catch (err) {
    const msg = err.message?.toLowerCase() || '';
    const isUnauthorized = isUnauthorizedError(msg);

    return res.status(500).json({
      success: false,
      llm: llm,
      data: isUnauthorized ? 'unauthorized API KEY' : 'internal-error',
    });
  }
});

export default router;
```

## Explications

### Importation des modules et configuration

```
import express from 'express';
import dotenv from 'dotenv';

import chatgptMock from '../mock/llm/chatgpt.mock.js';
import claudeMock from '../mock/llm/claude.mock.js';
import chatgptReal from '../services/llm/chatgpt.service.js';
import claudeReal from '../services/llm/claude.service.js';

dotenv.config();

const router = express.Router();
const useMock = process.env.USE_MOCK === 'true';
```

Dans cette section, nous importons les modules necessaires pour notre application. Nous utilisons `express` pour gerer notre serveur HTTP et `dotenv` pour gerer les variables d'environnement. Nous importons egalement des mocks et des services reels pour deux fournisseurs : `chatgpt` et `claude`.

Nous configurons ensuite `dotenv` pour qu'il puisse lire les variables d'environnement de notre fichier `.env`. Ensuite, nous initialisons un routeur `express` et determinons si nous devons utiliser des mocks ou des services reels en fonction de la variable d'environnement `USE_MOCK`.

### Fonctions utilitaires

```
function isUnauthorizedError(message) {
  return message.includes('unauthorized') || message.includes('401');
}

function getProvider(llm) {
  const providers = {
    chatgpt: {
      mock: chatgptMock,
      real: chatgptReal,
    },
    claude: {
      mock: claudeMock,
      real: claudeReal,
    },
  };

  return providers[llm] || null;
}
```

Ici, nous definissons deux fonctions utilitaires. `isUnauthorizedError` verifie si un message d'erreur contient des indications d'une erreur d'autorisation. `getProvider` recupere le bon fournisseur (mock ou reel) en fonction du parametre `llm`.

### Gestion des requetes

```
async function handleLLMRequest(type, llm, data) {
  const provider = getProvider(llm);
  if (!provider) { return { error: 'unknown-provider' }; }

  const fn = useMock ? provider.mock : provider.real;

  return { data: await fn(type, data) };
}

router.post('/:type/:llm', async (req, res) => {
  const { type, llm } = req.params;
  const input = req.body;

  try {
    const { data, error } = await handleLLMRequest(type, llm, input);

    if (error) {
      return res.status(400).json({ success: false, llm: llm, data: error });
    }

    return res.json({ success: true, llm: llm, data: data });

  } catch (err) {
```

```
    const msg = err.message?.toLowerCase() || '';
    const isUnauthorized = isUnauthorizedError(msg);

    return res.status(500).json({
      success: false,
      llm: llm,
      data: isUnauthorized ? 'unauthorized API KEY' : 'internal-error',
    });
  }
});
```

Dans cette section, nous definissons une fonction `handleLLMRequest` pour gerer les requetes a nos fournisseurs. Elle recupere le bon fournisseur, execute la fonction appropriee (mock ou reelle) et renvoie les donnees.

Ensuite, nous definissons une route `POST` pour notre routeur express. Cette route extrait les parametres de la requete, les passe a `handleLLMRequest`, et renvoie une reponse appropriee en fonction du succes ou de l'echec de la requete. En cas d'erreur, elle verifie si l'erreur est due a une autorisation non valide et renvoie un message d'erreur approprie.