Fichier : 05 - routes_ai.js

Tutoriel Node.js : Creation d'un routeur Express pour les services LLM

Dans ce tutoriel, nous allons decomposer un bloc de code Node.js qui illustre comment creer un routeur Express pour gerer les requetes vers les services LLM.

Importation des modules necessaires

```
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';
```

Dans cette section, nous importons tous les modules necessaires. `express` pour la creation de notre application web, `dotenv` pour la gestion des variables d'environnement. Nous importons egalement des modules de services mock et reels pour deux fournisseurs LLM, `chatgpt` et `claude`.

Configuration et initialisation

```
dotenv.config();

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

Ici, nous initialisons `dotenv` pour charger les variables d'environnement. Nous creons egalement une nouvelle instance de routeur Express et determinons si nous devons utiliser les services mock ou reels en fonction de la variable d'environnement `USE_MOCK`.

Fonctions auxiliaires

```
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) };
}
```

Dans cette section, nous definissons trois fonctions auxiliaires. `isUnauthorizedError` verifie si un message d'erreur indique une erreur d'autorisation. `getProvider` retourne le fournisseur LLM approprie en fonction du nom fourni. `handleLLMRequest` gere les requetes LLM en appelant la fonction appropriee du fournisseur LLM et retourne le resultat.

Gestion des requetes

```
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;
```

Enfin, nous definissons un gestionnaire pour les requetes POST vers notre routeur. Nous extrayons les parametres de la requete, appelons notre fonction `handleLLMRequest` et renvoyons une reponse appropriee en fonction du resultat. En cas d'erreur, nous verifions si c'est une erreur d'autorisation et renvoyons un message d'erreur approprie.

Et voila ! Vous avez maintenant une meilleure comprehension de la facon dont vous pouvez creer un routeur Express pour gerer les requetes vers les services LLM dans une application Node.js.