Retour sur le Back-end

Dans le monde du développement web, le backend joue un rôle essentiel. Il est responsable du traitement des requêtes, de l'accès à la base de données, de la gestion des utilisateurs et bien plus encore. Dans ce cours, nous explorerons les bases du backend en utilisant Node.js, le framework Express, et l'ORM Sequelize.

Node.js, Express.js et Sequelize sont des technologies populaires pour le développement du backend.

Qu'est-ce que le back-end ?

Le backend, ou côté serveur, est la partie d'une application web qui n'est pas directement accessible aux utilisateurs. Il gère les fonctionnalités "côté serveur", comme le stockage des données, la gestion des utilisateurs, l'authentification, etc.

Node et Express

Node.js est un environnement d'exécution côté serveur construit sur le moteur JavaScript V8 de Google Chrome. Il permet aux développeurs d'utiliser JavaScript pour écrire des scripts côté serveur, facilitant ainsi le développement d'applications web à l'aide d'une seule langue.

Node ajoute également des fonctionnalités que le JavaScript du navigateur standard ne possède pas, comme par exemple l'accès au système de fichiers local.

Express est un framework reposant sur Node, qui facilite la création et la gestion des serveurs Node, comme vous le verrez à mesure que nous progresserons dans ce cours. Il simplifie la création d'applications web en fournissant des fonctionnalités utilitaires pour les routes, les middlewares, la gestion des requêtes et des réponses, etc.

Exemple de code Express.js :

```
const express = require("express");
const app = express();

app.get("/", (req, res) => {
  res.send("Hello, Express.js!");
});

app.listen(3000, () => {
  console.log("Server is running on port 3000");
});
```

C'est quoi npm ?

npm, ou Node Package Manager, est le gestionnaire de paquets pour Node.js. Il permet d'installer, de partager et de gérer les dépendances (bibliothèques, frameworks, etc.) nécessaires à votre projet.

Base de Données

Une base de données est une collection de données structurées. La structure des données, ainsi que l'organisation de la collection dépendent du type de base de données.

Les BD relationnelles

Une base de données SQL est une base de données relationnelle qui emploie le langage **S**tructured **Q**uery **L**anguage pour gérer les données qu'elle contient.

Généralement, une base de données SQL organise ses données dans des tables selon des schémas stricts. Par exemple, si on décide qu'un utilisateur a un prénom, un nom et une adresse mail, on aura une table `Users` avec des colonnes `first_name` , `last_name` et `email` , et chaque rangée correspondra à un utilisateur. Pour être un utilisateur valide, chaque utilisateur doit avoir un prénom, un nom et une adresse mail, donc on peut être certain que chaque utilisateur tiré de la base de données aura ces attributs.

Dans une base de données SQL, les relations entre les différentes tables sont très importantes. Pour cette raison, notre table `Users` aura également une colonne `id` pour l'identifiant individuel unique de chaque utilisateur. Cela signifie que l'on peut faire référence à un utilisateur depuis une autre table.

Les bases de données SQL (comme MySQL ou PostgreSQL) sont très bien adaptées aux données relationnelles et pour des données où des définitions fortes sont nécessaires. Cependant, dans le contexte de projets plus petits, des MVP ou des startups, on ne sait peut-être pas encore à quoi ressemblera notre dernier modèle de données. On souhaitera aussi potentiellement pouvoir grandir rapidement (il est très difficile d'agrandir une base de données SQL au-delà d'une certaine limite) pour pouvoir gérer un plus grand nombre d'utilisateurs. Voilà où le NoSQL montre son utilité, en particulier MongoDB.

Le NoSQL

Le NoSQL signifie que l'on ne peut pas utiliser SQL pour communiquer avec. Les données sont stockées comme des collections de documents individuels décrits en JSON. Il n'y a pas de schéma strict de données (on peut écrire ce que l'on veut où l'on veut), et il n'y a pas de relation concrète entre les différentes données. Cependant, il existe des outils pour nous aider à subvenir à ces besoins.

C'est quoi un ORM (Object-Relational Mapping) ?

Un ORM est un outil qui permet de manipuler des bases de données relationnelles en utilisant des objets JavaScript plutôt que du SQL brut. Cela simplifie le processus d'interaction avec la base de données en permettant aux développeurs d'utiliser des langages de programmation orientés objet.

Sequelize

Sequelize est un ORM pour Node.js. Il simplifie la gestion des bases de données relationnelles en utilisant des modèles d'objets pour représenter les tables et les relations dans la base de données.

Exemple de code Sequelize :

```
const { Sequelize, DataTypes } = require("sequelize");
const sequelize = new Sequelize("database", "username", "password", {
```

```
  dialect: "mysql",
});

const User = sequelize.define("User", {
  firstName: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  lastName: {
    type: DataTypes.STRING,
  },
});

(async () => {
  await sequelize.sync();
  const user = await User.create({ firstName: "John", lastName: "Doe" });
  console.log(user.toJSON());
})();
```

Notions

CORS

CORS signifie « Cross Origin Resource Sharing ». Il s'agit d'un système de sécurité qui, par défaut, bloque les appels HTTP entre des serveurs différents, ce qui empêche donc les requêtes malveillantes d'accéder à des ressources sensibles.

C'est une politique de sécurité qui détermine si les ressources d'une page web peuvent être demandées à une autre page web en dehors du domaine d'origine.

```
const express = require("express");
const app = express();

// Middleware pour gérer CORS
app.use((req, res, next) => {
  res.header("Access-Control-Allow-Origin", "*");
  res.header(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept"
  );
  next();
});

// ... autres configurations et routes
```

CRUD (Create, Read, Update, Delete)

Opérations de base pour la gestion des données dans une base de données.

Create (création de ressources)

Read (lecture de ressources)

Update (modification de ressources)

Delete (suppression de ressources)

```
const express = require("express");
const app = express();

// Route pour récupérer des données (GET)
app.get("/users", (req, res) => {
  // Logique pour récupérer des utilisateurs depuis la base de données
  res.send("Liste des utilisateurs");
});

// Route pour créer des données (POST)
app.post("/users", (req, res) => {
  // Logique pour créer un nouvel utilisateur dans la base de données
  res.send("Utilisateur créé");
});

// Route pour mettre à jour des données (UPDATE)
app.put("/users/:id", (req, res) => {
  // Logique pour mettre à jour un utilisateur dans la base de données
  res.send("Utilisateur mis à jour");
});

// Route pour supprimer des données (DELETE)
app.delete("/users/:id", (req, res) => {
  // Logique pour supprimer un utilisateur de la base de données
  res.send("Utilisateur supprimé");
});

// ... autres configurations
```

Requêtes GET, POST, UPDATE, DELETE

Actions HTTP correspondant aux opérations CRUD. GET pour récupérer des données, POST pour créer, UPDATE pour mettre à jour, et DELETE pour supprimer.

Routes

Définition des points d'entrée de l'API, associant une URL à une fonction d'exécution.

Rappels

- exécutez la commande de terminal `npm init` pour initialiser votre projet. Ce processus génère un fichier `package.json` vierge, dans lequel seront enregistrés les détails de tous les packages npm que nous utiliserons pour ce projet.

- Node utilise le système de module CommonJS, donc pour importer le contenu d'un module JavaScript, on utilise le mot-clé `require` plutôt que le mot-clé `import` . Ce système est particulièrement utile car il nous permet d'importer les modules de base de Node très facilement (comme le module `http` ici) sans spécifier le chemin exact du fichier. Node sait qu'il doit importer un module de base quand on ne spécifie pas un chemin relatif (qui commence par ./ ou / , par exemple).
- Démarrez le serveur en exécutant `node server` à partir de la ligne de commande
- Pour simplifier le développement Node, vous souhaiterez peut-être installer nodemon . Pour ce faire, exécutez la commande suivante :

  ```
  npm install -g nodemon
  ```

  Désormais, au lieu d'utiliser `node server` pour démarrer votre serveur, vous pouvez utiliser nodemon server . Il surveillera les modifications de vos fichiers et redémarrera le serveur lorsqu'il aura besoin d'être mis à jour. Cela vous garantit d'avoir toujours la dernière version de votre serveur dès que vous sauvegardez, sans devoir relancer manuellement le serveur
- Pour installer un paquet sur votre projet, exécutez la commande suivante à partir de votre dossier de projet : `npm install nom-paquet`

Express

- En utilisant `app.post()` au lieu de `app.use()`, votre backend répondra uniquement aux requêtes de type POST. exemple :

```
app.post("/api/stuff", (req, res, next) => {
  res.status(201).json({
    message: "Objet créé !",
  });
});
```

autre exemple, avec une requête get et delete :

```
app.get('/api/stuff/:id', (req, res, next) => {
  [...] //traitement de la requete GET par le serveur
});
```

```
app.delete('/api/stuff/:id', (req, res, next) => {
  [...] //traitement de la requete DELETE par le serveur
});
```

Sources

- Site officiel de Node.js

- Site officiel de npm
- Site officiel d'Express.js
- Site officiel de Sequelize
- Tutoriel vidéo sur Node.js, Express.js et Sequelize
- Introduction à Node.js et Express.js sur MDN Web Docs
- Documentation Sequelize