TD: Créer une application de microservices de base avec Node.JS

Objectifs

Dans cet exemple, nous allons créer un microservice à l'aide de Node.JS qui se connecte à une API externe. L'exigence pour ce service est d'accepter deux codes postaux et de renvoyer la distance entre eux en Km.

Introduction

La création d'applications du monde réel dans le langage JavaScript nécessite une programmation dynamique, où la taille de l'application JavaScript augmente de manière incontrôlable. De nouvelles fonctionnalités et mises à jour sont publiées et vous devez corriger les bogues pour maintenir le code.

Pour exécuter cela, de nouveaux développeurs doivent être ajoutés au projet, ce qui devient compliqué. La structure des modules et des packages est incapable de réduire et de simplifier l'application. Pour exécuter l'application en douceur, il est essentiel de convertir la grande structure homogène en petits morceaux de programmes indépendants. De telles complexités peuvent être résolues sans effort lorsque les applications JavaScript sont construites sur des microservices, d'autant plus avec l'écosystème Node.js.

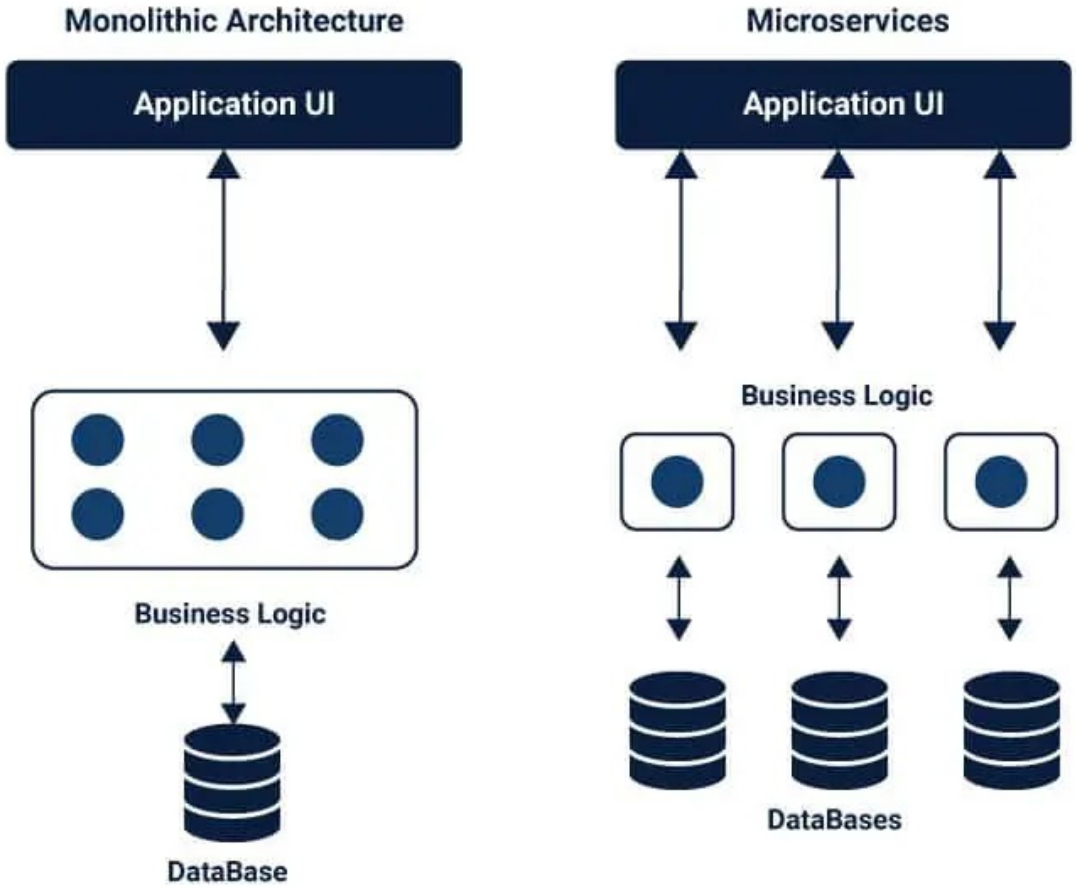
Que sont les microservices?

Dans le développement d'applications logicielles, les microservices structurent l'application sur un assemblage de services interconnectés. Avec les microservices, l'architecture de l'application est construite avec des protocoles légers. Les services sont finement ajoutés dans l'architecture. Les microservices désintègrent l'application en services plus petits et permettent une meilleure modularité.

Par rapport à son prédécesseur, l'architecture monolithique, les microservices sont de loin plus avantageux. Vous n'avez pas besoin de regrouper tous les composants logiciels et services dans un grand conteneur et de les emballer. Avec les microservices, vous pouvez créer une application avec:

- une plus grande flexibilité
- haute scalabilité
- développement continu
- organisation systématique des données
- fiabilité

La création d'applications JavaScript sur des microservices vous aide à vous concentrer sur le développement de modules monofonctionnels avec des opérations clairement définies et des interfaces précises. Le processus de développement d'applications devient plus agile et les défis des tests continus sont atténués.



Étape 1: Générez un fichier package.json

Créez un répertoire pour votre application (TD10). Dans ce répertoire, exécutez ceci depuis votre terminal:

```
$ npm init --yes
```

Cela initialise le gestionnaire de dépendances npm (npm Dependency Manager).

Étape 2: installer les dépendances

Nous allons utiliser le package **Express** pour créer notre service. **Express** est un framework établi pour les applications Node et continue de bénéficier du soutien de la Fondation Node.js. Nous allons également utiliser le package **Axios** pour nous permettre de nous connecter à une API tierce sur le Web.

Nous allons ajouter ces packages à notre projet avec la commande suivante:

```
$ npm install express axios –save
```

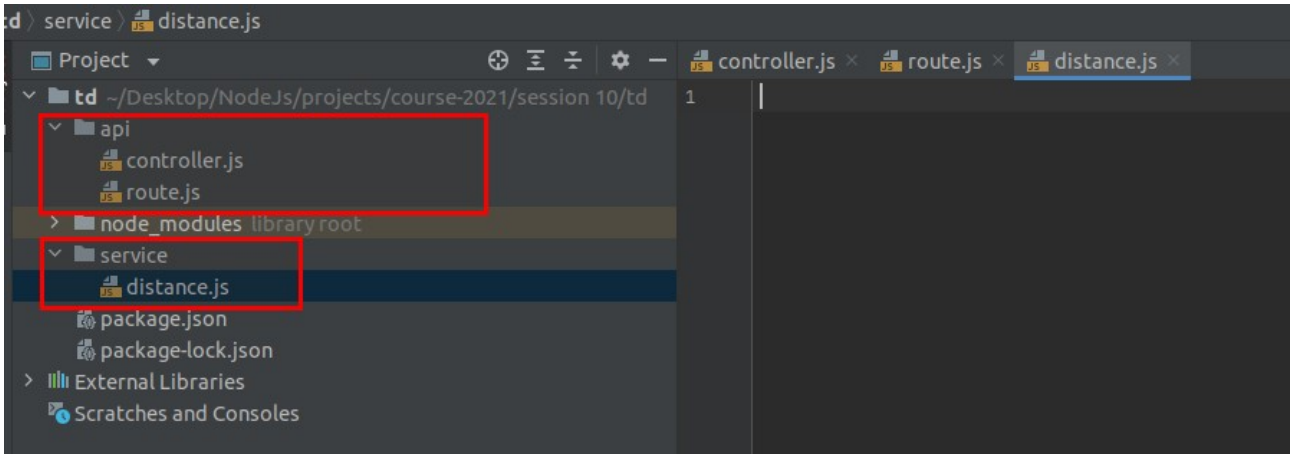
Étape 3: Création des répertoires "api" et "service"

Créez le répertoire "**api**" avec ces deux fichiers à l'intérieur:

- **controller.js**
- **route.js**

Créez le répertoire "**service**" avec ce fichier à l'intérieur:

- **distance.js**



Étape 4: Création d'un serveur pour accepter les requêtes

Dans votre répertoire racine, créez le fichier **server.js:**

```
$ touch server.js
```

Ce fichier contient le code ci-dessous:

```
const express = require('express')
const app = express();
const port = 3000;
const routes = require('./api/route');
app.use(routes);
app.listen(port, function() {
      console.log('Serveur ecoute sur le port: ' + port);
});
```

La première chose que nous allons faire est d'importer le package **express** dans le fichier. Nous allons l'utiliser pour créer un nouvel objet **app**. Nous préciserons également le port.

La ligne suivante apporte un objet "**routes**" du fichier **route.js** dans le dossier api. Nous utiliserons cet objet comme middleware pour exécuter la fonction de rappel qui définit les routes pour notre application. Enfin, nous dirons à l'application de commencer à écouter sur le port que nous avons défini et d'afficher un message sur la console lorsque ce processus est terminé.

Étape 5: Définir la route

L'étape suivante consiste à définir les routes pour le serveur et à attribuer chacune à une cible dans notre objet contrôleur. Nous allons construire le contrôleur à l'étape suivante. Nous aurons un point de terminaison. Un point de terminaison de distance qui inclut deux paramètres de chemin (**path**), les deux codes postaux. Ce point de terminaison renvoie la distance, en km, entre ces deux codes postaux aux États-Unis.

Ajoutez le code suivant à **route.js**:

```
const Controller = require("./controller")
const express = require("express");
let router = express.Router();
router.get("/distance/:zipcode1/:zipcode2",Controller.getDistance);
module.exports = router;
```

Étape 6: codez le contrôleur

Dans le fichier du contrôleur, nous allons créer une méthode **getDistance** pour amener la distance entre deux codes postaux.

```
const distance = require('../service/distance');
exports.getDistance = (req,res)=>{
	distance.find(req,res,(error,data)=>{
		if (error){
			res.send({distance: -1});
		}else{
			res.send(data);
		}
	});
};
```

Nous passerons les objets de requête et de réponse à la fonction find dans ce module. Cette fonction comprend également une fonction de rappel (callback). Cette fonction accepte un objet d'erreur (error) et un objet de données (data). S'il y a une erreur dans la réponse, nous retournons -1 dans notre réponse ; sinon, nous renvoyons les résultats de la fonction.

Étape 7: codez le service “distance”

Ajoutez le code suivant à **service/distance.js:**

```
const axios = require("axios");
const APP_KEY = "YOUR_APP_KEY_HERE";
const zipCodeURL = 'https://www.zipcodeapi.com/rest/';
exports.find = (req,res,callback) =>{
	let URL = zipCodeURL + APP_KEY
		+ '/distance.json/' + req.params.zipcode1 + '/'
		+ req.params.zipcode2 + '/km';
	axios.get(URL)
		.then((response) => {
			console.log(response.data);
			console.log(response.status);
			console.log(response.statusText);
			console.log(response.headers);
			console.log(response.config);
			return callback(null, response.data);
```

```
        }).catch(function (error) {
            if (error.response) {
                // Request made and server responded
                console.log(error.response.data);
                console.log(error.response.status);
                console.log(error.response.headers);
            } else if (error.request) {
                // The request was made but no response was received
                console.log(error.request);
            } else {
                // Something happened in setting up the request that triggered an Error
                console.log('Error', error.message);
            }
            return callback(error);
    });
};
```

Afin d'envoyer une requête http get, nous allons utiliser le module **axios**. Reportez-vous à cette documentation pour savoir comment utiliser **axios**:

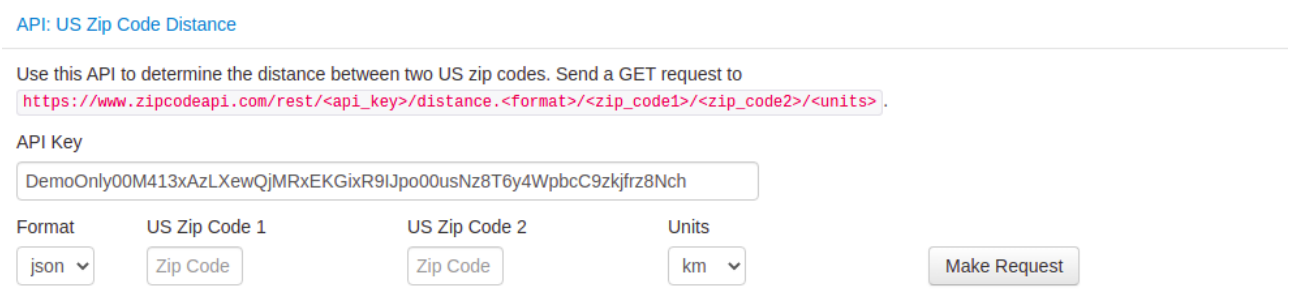
https://axios-http.com/docs/api_intro

https://github.com/axios/axios

Nous utiliserons l'API de distance fournie par **ZipCodeAPI.com**. Une étape importante à faire est d'obtenir une clé API pour utiliser cette API, et c'est gratuit si vous vous inscrivez. Envisagez de vous inscrire et d'obtenir une clé API en utilisant ce lien: https://www.zipcodeapi.com/register

Lorsque vous obtenez votre clé, remplacez la variable **APP_KEY** par votre clé.

Vous pouvez vous référer à cette documentation API pour tester l'API de distance: https://www.zipcodeapi.com/API



L'URL que nous allons utiliser est la suivante:
https://www.zipcodeapi.com/rest/<api_key>/distance.<format>/<zip_code1>/<zip_code2>/<units>

- api_key → votre clé API
- format → json
- zip_code1 → premier paramètre passé dans le chemin de la requête
- zip_code2 → deuxième paramètre passé dans le chemin de la requête
- units → km

Étape 8: Exécutez votre application

$ node server.js

Assurez-vous de tester votre application avec de vrais codes postaux qui existent aux États-Unis. Vous pouvez tester cette URL pour obtenir la distance entre l'Alaska et l'Arizona:

http://localhost:3000/distance/99501/85001 (modifier le port si vous en avez choisi un autre que 3000)

Quelque chose à quoi penser:

Réfléchissez à la manière d'optimiser votre application à l'aide de stratégies de mise en cache. Pouvez-vous trouver un moyen de traiter les requêtes répétées qui demandent la distance entre les deux mêmes codes postaux afin de ne pas demander les données de l'API externe à chaque fois?

Quelques liens que vous pouvez vérifier:

- https://www.digitalocean.com/community/tutorials/how-to-optimize-node-requests-with-simple-caching-strategies
- https://medium.com/the-node-js-collection/simple-server-side-cache-for-express-js-with-node-js-45ff296ca0f0