W1 - PHP Avancé

W-WEB-250

my_micro_services

Introduction aux micro-services

1.3

my_micro_services

delivery method: microservices on Github
language: PHP, JSON

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

ETAPE 1 - API MICRO-FRAMEWORK PHP

Bonjour !
Pour cette introduction, nous allons construire ce que nous appelons un microservice, ici, un microservice de messagerie.

Un micro-service est une petite API qui fait peu de choses. Un site Web entier peut être divisé en plusieurs microservices.

Voici quelques liens utiles:

- Sur les micro-service

Il existe plusieurs micro-frameworks PHP, nous vous laissons le soin de choisir celui qui sera le plus adapté à vos besoins.

Pour la suite, nous utiliserons Slim.

Installation de Slim

Introduction

Installez Slim et un squelette d'application avec `slim-skeleton`.

La BDD

Slim peut utiliser un ORM pour gérer ses models et interfacer la BDD.
Choisissez Eloquent pour la suite.
Installez puis configurez Eloquent pour Slim.
Enfin, ajoutez votre ORM à votre Service Factory.
Assurez-vous que votre installation est stable et fonctionne en créant une query simple sur une table de test.

MY_FIRST_CRUD

Maintenant vous allez devoir créer 2 modèles (Message et User), ainsi que leurs CRUD respectif (et donc leurs routes).

Etape 2 - Connexion / Inscription / Messages

Une fois la configuration de Slim réussie, vous allez devoir implémenter un modèle User avec la possibilité de se connecter et de s'inscrire.
Vous devrez implémenter un système de session. Lors de la connexion un JWT (JSON Web Token) devra être créé qui permettra à identifier l'user pour les autres requêtes.
Liens utiles :

- Json Web Token.
- Sécuriser une API REST.

Une fois le système d'user fonctionnel, vous devrez gérer un système de messagerie.
Le principe de cette API sera de pouvoir ajouter des messages, les récupérer, les modifier et les supprimer.
À vous de voir comment les requêtes seront formées et ce qu'elles retourneront (Token, JSON…).
Liens utiles :

- Postman pourra être votre ami pour tester vos routes.

ETAPE 3 - API EXPRESSJS NOSQL

Pour cette étape vous aurez besoin de :

- Node.JS
- MongoDB
- Express

Après avoir créé un dossier api_express, lancez la commande suivante en complétant les champs :

Terminal

```
~/W-WEB-250> npm init
```

Les commandes ci-dessus entraînent la création d'un fichier package.json. Le fichier package.json est utilisé pour gérer les packages npm installés localement.
Il inclut également les métadonnées sur le projet, telles que le nom et le numéro de version.

Ensuite, nous devons installer les packages que nous utiliserons pour notre API.
Les packages sont:

- ExpressJS est une application Web flexible Node.JS qui offre de nombreuses fonctionnalités pour les applications Web et mobiles.
- Mongoose est l'ODM mongoDB pour Node.JS.
- body-parser est un package pouvant être utilisé pour gérer les requêtes JSON.

Terminal

```
~/W-WEB-250> npm install -save express body-parser mongoose
```

Créez et complétez le fichier app.js :

Terminal

```
~/W-WEB-250> cat app.js
//app.js
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

// init app
const app = express();

// connect MongoDB with mongoose
let dev_db_url = 'url_de_votre_mongo';
let mongoDB = process.env.MONGODB_URI || dev_db_url;
mongoose.connect(mongoDB);
mongoose.Promise = global.Promise;
let db = mongoose.connection;
db.on('error', console.error.bind(console, 'Connexion error on MongoDB: '));

// Utilisation de body parser
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: false}));

let port = 5555;

app.listen(
    port, () => {
        console.log('Server running on : ' + port);
    }
);
```

Lancez votre serveur à l'aide de la commande :

Terminal

```
~/W-WEB-250> node app.js
```

ETAPE 4 - LES CRUDS

À présent créez les dossiers controllers, models, routes.

Le principe est de créer une API NoSQL qui s'occupera de gérer uniquement les discussions.

Créez un fichier discussion.model.js dans le dossier model. Ce fichier instanciera et exportera un nouveau schema : DiscussionSchema.
(A vous de voir les champs nécessaires). Vous trouverez de l'aide ici.
Créez un fichier discussion.route.js. Dans ce fichier on y retrouvera toutes les routes nécessaires (CRUD).
Vous trouverez de l'aide ici (n'oubliez pas d'use vos routes dans app.js).
Allez dans le répertoire de controllers et créez un nouveau fichier js nommé discussion.controller.js.
Implémentez les fonctions nécessaires (n'oubliez pas les validators).

ETAPE 5 - LE CONNECTEUR : RELIER LES 2 API

Dans cette étape vous devez créer un nouveau projet avec votre mini-framework qui relie les deux API précédemment créées.

A vous de penser à toutes les différentes routes à implémenter ainsi que de faire les liaisons entres les deux API.

Représentation du schéma de fonctionnement final

Bonus

- Messagerie graphique
- Client en React (le client PHP du reste du sujet doit rester présent malgré tout dans le rendu.)
- Surprenez-nous