Projet Inventaire REST

INVENTAIRE REST

CREATEUR DU DOCUMENT

NOM	Date	Version
SEFFER LOIC	27/11/2020	V1
DUPUY REMI	27/11/2020	V1
LAIOLO LOUIS	27/11/2020	V1

VALIDATION

NOM	Function	Date

REVISIONS

NOM	Date de Revision	Raison

Documents adjoins

Titres

Projet Inventaire REST

Contents

1. INTRODUCTION

1.1. But du projet

Ce projet vise à créer une application en architecture REST. Nous avons donc choisi de créer un logiciel de gestion d'inventaire en ligne. Pourquoi ce choix ?

Il se trouve que Louis Laiolo, l'un des membres de l'équipe, travaillait durant la même période sur le même type de logiciel, mais programmé en C++, et sous forme d'application non Web.
C'est de cette idée que nous est venue l'envie de créer le même type de programme, mais en version web.

Comme demandé, l'architecture de ce projet est en REST, c'est-à-dire qu'il s'appuie sur l'utilisation du protocole HTTP et l'utilisation de format JSON, qui sont des technologies issues du WEB.

2. LES TECHNOLOGIES UTILISEES

Comme indiqué plus haut, il a été demandé de programmer le projet en architecture REST. Pour se faire, nous avions besoins de technologies inhérentes au web.

Nous avons donc utilisé Angular pour produire le front end, et nous avons choisis d'utiliser la base NoSQL MangoDB, pour faire écho aux cours de base de données de M1. Nous nous épargnons ainsi l'utilisations de requêtes, et nous ne faisons que de simples « get » dans la base de donnée.

Nous avons aussi utilisé Node.js afin de faire tourner Angular, mais aussi de setup un serveur coté machine pour tester l'application et épargner la mise en place d'un serveur Apache/Nginx lors d'un éventuel déploiement à grande échelle de notre application.

Enfin, pour le plus important, nous avons choisis de programmer le back-end (donc l'API) en TypeScript, qui a l'avantage d'être du Javascript typé, ce qui casse certes la liberté de programmation qu'apporte JS, mais qui permet aussi de supprimer « l'aspect abstrait » de ce dernier.

2.1. Angular

Angular est un framework javascript qui permet de créer des pages internet capable de s'auto-actualiser sans se recharger (système de single Page Applications). Il s'agit, pour simplifier, d'une méthode pour faire de l'AJAX en beaucoup plus rapide et trivial.

Facile à utiliser et à modifier, il est architecturé en MVC, ce qui permet de facilement structurer la base de donnée, le front et le bac, sans devoir tout changer en cas de problème. Au cours de ce projet, on essayera au mieux d'utiliser les syntaxes et la mise en forme officielle normé par les créateurs d'Angular : Google.

Il est important de ne pas le confondre avec AngularJS, qui n'est pas la même technologie, et présente des différences. Contrairement à AngularJS, ce Framework fonctionne comme une sorte de d'arbre de DOM, ce qui va permettre la suppression de la notion de Controller.

2.2. Node.js

Node.js est un moteur logiciel en Javascript. Il est très utile et efficace pour gérer des situations de réseau évènementiel, ce qui exactement ce que nous recherchons pour la page d'accueil.

Autre chose intéressante, notamment pour la facilité de déploiement, il est possible de se passer de serveur Apache/Nginx lors de la mise en place d'applications développées avec node.js.

Dans tous les cas, il était obligatoire de l'utiliser pour pouvoir programmer en Angular.

2.3. MongoDB

MongoDB est l'une des bases noSQL les plus connues. Orienté sur la gestion de données typées document, elle est la base de choix pour ce projet de feuille d'appel dématérialisé.

Son absence de schéma prédéterminé permettra la création de clefs à la volée, ce qui nous aidera pour récupérer les noms des professeurs et des salles pour les intégrer à notre squelette de feuille, qui sera elle-même stockée, ainsi que tous les PDF des feuilles passées, dans la base de donnée.

2.4. TypeScript

TypeScript est un langage de programmation créé par Microsoft, au système de Typage dynamique, ce qui lui permet d'être, entre autre, plus digeste à programmer et à comprendre que le JavaScript classique. Il n'y a quasiment aucun apport par rapport JS. Pour exemple, il est possible de transcompiler le TS en JS.

2.5. Explications des choix

Le choix du combo MongoDB/Node/TS/Angular devient peu à peu un classique dans l'industrie du développement web, ce qui nous a motivé, entre autre, à utiliser ces outils.

En outre, la facilité de modification d'Angular et de MangoDB faciliteront les tests, et le déploiement dans un éventuel futur de l'application.

3. LES ROUTES

L'api est structurée autour d'un système de route.

Chacune d'elles appellent des méthodes provenant des contrôleurs qui appellent eux même des services en utilisant des Model pour structurer les données.

Par exemple, articles_routes appelle le controller articleController qui appelle le service d'article et utilise la structure de données d'article.

Ainsi, Nos routes sont contrôlées par les Controller qui appellent le service souhaité. Toutes nos données sont structurées par les model et les schéma grâce à mongoose

Projet Inventaire REST

ACTIONS DES ROUTES		
Articles		
	Route	Description
GET	/api/article	récupère et retourne les articles en base de données
	/api/article/ :id	prends en paramètre un id d'article et retourne cet article s'il est en base.
POST	/api/article	prends en paramètre les informations d'un article, crée un objet article en fonction des paramètres, l'ajoute dans la base de données.
PUT	/api/article/ :id	prends en paramètres des attributs saisies et l'id d'un article existant et modifie l'article concerné dans la bdd
DELETE	/api/article/ :id	prends en paramètre l'id d'un article et le supprime de la base de données
Fournisseur		
	Route	Description
GET	/api/fournisseur	récupère et retourne les fournisseurs en base de données
	/api/fournisseur/:id	prends en paramètre un id d'un fournisseur et retourne ce fournisseur s'il est en base
POST	/api/fournisseur	prends en paramètre les informations d'un fournisseur, crée un objet fournisseur en fonction des paramètres et l'ajoute dans la base de données.
PUT	/api/fournisseur/:id	prends en paramètres des attributs saisies et l'id d'un fournisseur existant et modifie le fournisseur concerné dans la bdd
DELETE	/api/fournisseur/:id	prends en paramètre l'id d'un fournisseur et le supprime de la base de données

Projet Inventaire REST

Historique		
	Route	Description
GET	/api/historique	récupère et retourne les historiques en base de données
	/api/historique/:id	prends en paramètre un id d'un historique et retourne cet historique s'il est en base
POST	/api/historique	prends en paramètre les informations d'un historique, crée un objet historique en fonction des paramètres et l'ajoute dans la base de données
PUT	/api/historique/:id	prends en paramètres des attributs saisies et l'id d'un historique existant et modifie l'historique concerné dans la bdd
DELETE	/api/historique/:id	prends en paramètre l'id d'un historique et le supprime de la base de données
Magasin		
	Route	Description
GET	/api/magasin	récupère et retourne les magasins en base de données
	/api/magasin/:id	prends en paramètre un id d'un magasin et retourne cet article s'il est en base
POST	/api/magasin	prends en paramètre les informations d'un magasin, crée un objet magasin en fonction des paramètres et l'ajoute dans la base de données
PUT	/api/magasin/:id	prends en paramètres des attributs saisies et l'id d'un magasin existant et modifie le magasin concerné dans la bdd
DELETE	/api/magasin/:id	prends en paramètre l'id d'un magasin et le supprime de la base de données
Membre		
	Route	Description
GET	/api/membre	récupère et retourne les membres en base de données
	/api/membre/:id	prends en paramètre un id d'un membre et retourne cet article s'il est en base
POST	/api/membre	prends en paramètre les informations d'un membre, crée un objet membre en fonction des paramètres et l'ajoute dans la base de données
PUT	/api/membre/:id	prends en paramètres des attributs saisies et l'id d'un membre existant et modifie le membre concerné dans la bdd
DELETE	/api/membre/:id	prends en paramètre l'id d'un membre et le supprime de la base de données

Projet Inventaire REST

Roles		
	Route	Description
GET	/api/role	récupère et retourne les roles en base de données
	/api/role/:id	prends en paramètre un id d'un role et retourne ce role s'il est en base
POST	/api/role	prends en paramètre les informations d'un role, crée un objet role en fonction des paramètres et l'ajoute dans la base de données
PUT	/api/role/:id	prends en paramètres des attributs saisies et l'id d'un role existant et modifie le role concerné dans la bdd
DELETE	/api/role/:id	prends en paramètre l'id d'un role et le supprime de la base de données
Stocks		
	Route	Description
GET	/api/stock	récupère et retourne les stocks en base de données
	/api/stock/:id	prends en paramètre un id d'un stock et retourne cet article s'il est en base
POST	/api/stock	prends en paramètre les informations d'un stock, crée un objet stock en fonction des paramètres et l'ajoute dans la base de données
PUT	/api/stock/:id	prends en paramètres des attributs saisies et l'id d'un stock existant et modifie l'article concerné dans la bdd
DELETE	/api/stock/:id	prends en paramètre l'id d'un stock et le supprime de la base de données
Transfert		
	Route	Description
GET	/api/transfert	récupère et retourne les transferts en base de données
	/api/transfert/:id	prends en paramètre un id d'un transfert et retourne cet article s'il est en base
POST	/api/transfert	prends en paramètre les informations d'un transfert, crée un objet transfert en fonction des paramètres et l'ajoute dans la base de données
PUT	/api/transfert/:id	prends en paramètres des attributs saisies et l'id d'un transfert existant et modifie l'article concerné dans la bdd
DELETE	/api/transfert/:id	prends en paramètre l'id d'un transfert et le supprime de la base de données