**Cours 3 :**

Utilisation d’un ORM pour gérer sa base de données dans un projet NodeJS

Rappel de la structure du projet

**Resources** : correspond au endpoint pour communiquer avec notre API (déjà fait lors du cours sur la mise en place des middlewares GET / POST / DELETE …)

**Services** : appelés sur les endpoints, contiennent la logique / algo + utilisent la partie repository pour pouvoir manipuler les models (base de données) (déjà fait lors du cours sur la mise en place des middlewares GET / POST / DELETE …)

**Base de données** : models / repository contenu de ce cours

**Views :** (pas encore implémentée, à venir)

Partie client de l’application (GUI / IHM …), visible pas nos utilisateurs (dans un grand projet, il est préférable de supprimer cette partie views du projet contenant le code de l’api, afin d’avoir un projet front (dédié à cette partie client) et une partie API

Dans notre cas, l’application n’a pas spécialement de contrainte de scaling etc… donc un dossier views pour notre partie front suffira largement

**Utilisation de l’ORM : Sequelize (Nodejs)**

* Documentation officielle : <https://sequelize.org/master/>

Installation (pour mysql) :

npm install --save sequelize

npm install --save mysql2

Une fois installé, il y a les étapes suivantes :

* **Initialiser une connexion vers une base de données (local ou distante)**
* **Structure nos tables** (table = modèle ou entity)
* **Mettre en place les différentes méthodes pour manipuler nos modèles (Repository)**
* **Utiliser nos repository pour effectuer une action sur notre base de données depuis nos services**

*Exemple :*

Modèle User => Table User dans mysql

UserRepository => fichier js (ou une class) qui contiendra la définition des différentes functions qui servent à manipuler notre modèle (findAll / findById / create / delete / update …)

UserService => grâce à UserRepository, le service peut manipuler la base de données et en plus appliquer une logique.

Par exemple sur les enregistrements des utilisateurs je souhaite calculer l’âge par rapport à une date, ce calcul sera fait dans le service avant d’appeler la function create du UserRepository qui se chagera de l’enregistrement dans la base

UserResource => Expose une route à l’utilisateur, pour lui permettre d’utiliser notre API (en connectant chaque route avec son service associé)

Avant de coder :

SI aucun docker n’est installé, il suffit de lancer un mysql local avec wamp (en principe juste lancer wamp, lance automatiquement un mysql)

Le code est ici :

<https://github.com/sylvainSUPINTERNET/nodejs-cours>

Initialiser la connection :

A la racine, rajouter un dossier models et y ajouter un fichier index.js + dbConnection.js

https://github.com/sylvainSUPINTERNET/nodejs-cours/blob/master/models/dbConnection.js

|  |  |
| --- | --- |
|  |  |
|  |  |
| New Sequelize prend en paramètre   * course : nom de la base de donnée qui va être générer sur le wamp * root : nom d’utilisateur de mysql (par défaut sur window pour wamp) * root : mot de passe mysql (par défaut sur window la valeur est est une chaine de caractère vide (pas de de mot de passe) ) * host : localhost (instance mysql local de wamp) * dialect : ‘mysql’ (nous utilisons mysql pour notre exemple, d’où l’instanllation de mysql2, mais il tout à fait possible d’utiliser postgresql, mongodb ….) |  |
|  |  |
|  |  |
| Mise à jour du point d’entrée de notre application (index.js) pour indiquer à notre application qu’au démarrage, celle-ci devra se connecter à notre base de données (ici local)  https://github.com/sylvainSUPINTERNET/nodejs-*cours*/blob/master/index.js |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Ici, l’application ne lancera pas … car comme l’indique l’exemple, au démarrage, la base de données est initialisée, MAIS les modèles (tables) eux n’existent toujours pas (// init models …)   En effet, il faut aussi initialiser nos modèles au démarrage de l’application pour générer automatiquement nos tables dans notre base de données  Info : A chaque démarrage ici, on synchronise nos modèles avec notre base de données, l’idée est que lorsqu’on effectue une modification sur nos modèles niveau code, celle-ci est automatiquement synchronisé avec nos tables dans notre db local  Pour nos modèles, nous ajoutons un dossier pour chaque notion de notre application  Exemple :  Besoin d’une table product ?  Ajouter ces deux fichiers  => ./models/product/product.model.js  => ./models/repository/product.repository.js  product.model.js : contiendra la définition de notre table produit  product.repository.js : contiendra la définition des fonctions utilisées pour manipuler notre model (table)  https://github.com/sylvainSUPINTERNET/nodejs-cours/blob/master/models/product/product.model.js |  |
|  |  |
| Ici on définit notre model, comme on peut le voir, on définit les champs comme un object json, ainsi que les différentes contraintes (tel que le type de la donnée)  A partir de ce simple object json et de notre script d’initialisation placer dans le démarrage de l’application, sequelize va être capable de générer notre base de données (course) + nos tables (product)  <https://github.com/sylvainSUPINTERNET/nodejs-cours/blob/master/models/repository/product.repository.js> |  |
|  |  |
|  |  |
|  |  |
|  |  |

Dans notre repository, on import juste notre modèle et définit des méthodes (ici une méthode create et une méthode pour récupérer un item par son id getById)

Ces méthodes vont simplement utiliser des fonctions sequelize déjà faites tels que findByPk ou bien create (méthodes qui sont disponibles sur les model Sequelize, tel que product dans notre cas)

https://github.com/sylvainSUPINTERNET/nodejs-cours/blob/master/models/index.js



Pour finir on rend accessible notre modèle grâce à l’index.js (c’est ici que lorsqu’un nouveau modèle est créé, que cet export doit être mis à jour, puisqu’au démarrage de l’application, c’est object exports qui va être lut et permettre une synchronisation automatique de Sequelize avec notre base de données local

Rendu final du dossier models :

<https://github.com/sylvainSUPINTERNET/nodejs-cours/tree/master/models>

2) Mise en place du service product pour manipuler notre base de données proprement

* Ajouter un dossier./services à la racine du projet
* Ajouter un index.js
* Ajouter un dossier product tel que :

./services/product et y mettre un fichier product.service.js tel que : ./services/product/product.service.js

<https://github.com/sylvainSUPINTERNET/nodejs-cours/blob/master/services/product/product.service.js>



Ici comme déjà fait sur les cours d’avant, on définit notre propre middleware express, qui utilisent notre repository précédemment créé.

Exemple :

Middleware express : getProduct

Récupère l’id de la requête présente dans les paramètres de l’Url tel que http://localhost:4000/products/5

Ce middleware utilise simplement la méthode de notre repository product, getById, qui elle-même va appeler une méthode du modèle sequelize (getByPk), qui retournera alors le product pour l’id reçu dans le middleware

Mettre à jour l’index.js pour nos services

https://github.com/sylvainSUPINTERNET/nodejs-cours/blob/master/services/index.js



Rendu final :

<https://github.com/sylvainSUPINTERNET/nodejs-cours/tree/master/services>

Dernière étape, permettre à l’utilisateur d’utiliser notre API : création de la resource

A la racine du projet ajouter un dossier ./resources

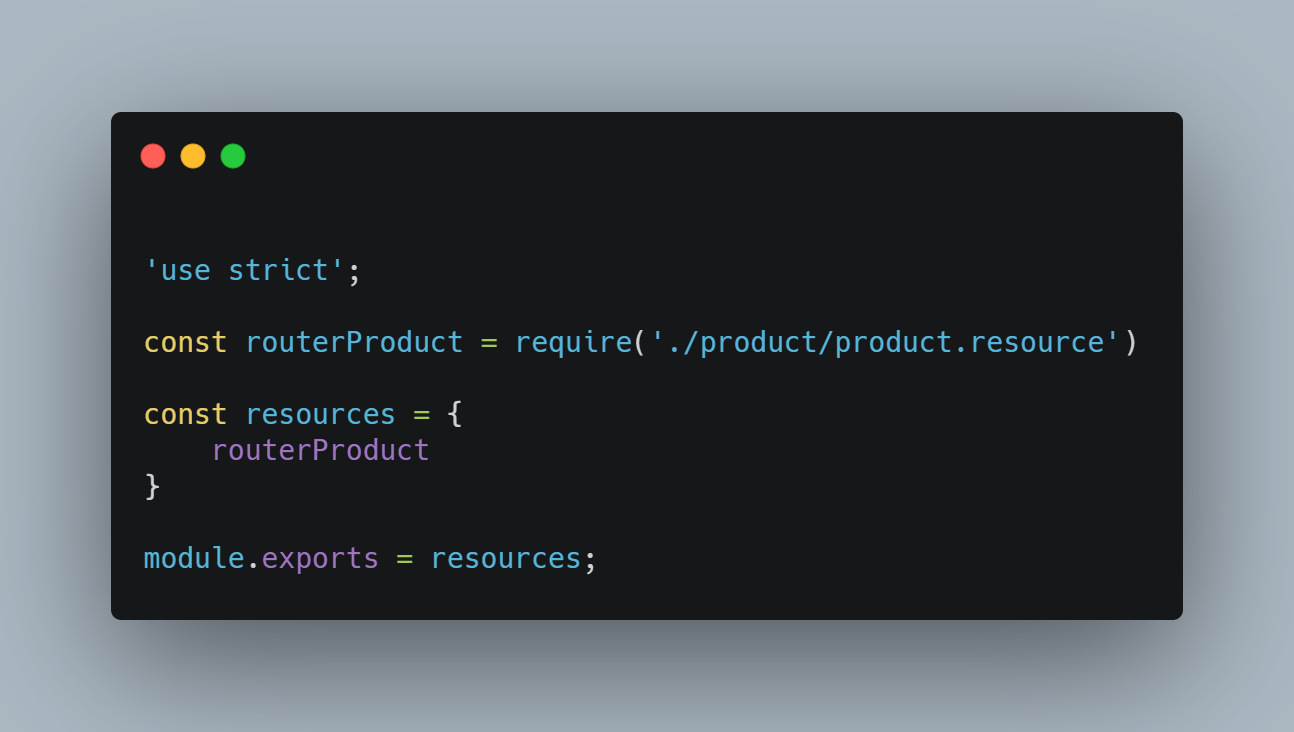
* Ajouter un fichier index.js tel que ./resources/index.js
* Ajouter un dossier product tel que ./resources/product
* Ajouter un fichier product.resource tel que product.resource.js

https://github.com/sylvainSUPINTERNET/nodejs-cours/blob/master/resources/product/product.resource.js

C’est dans ce fichier resource, qu’on utilise notre middleware (définit dans le service)

On met ensuite à jour notre index pour nos resources :

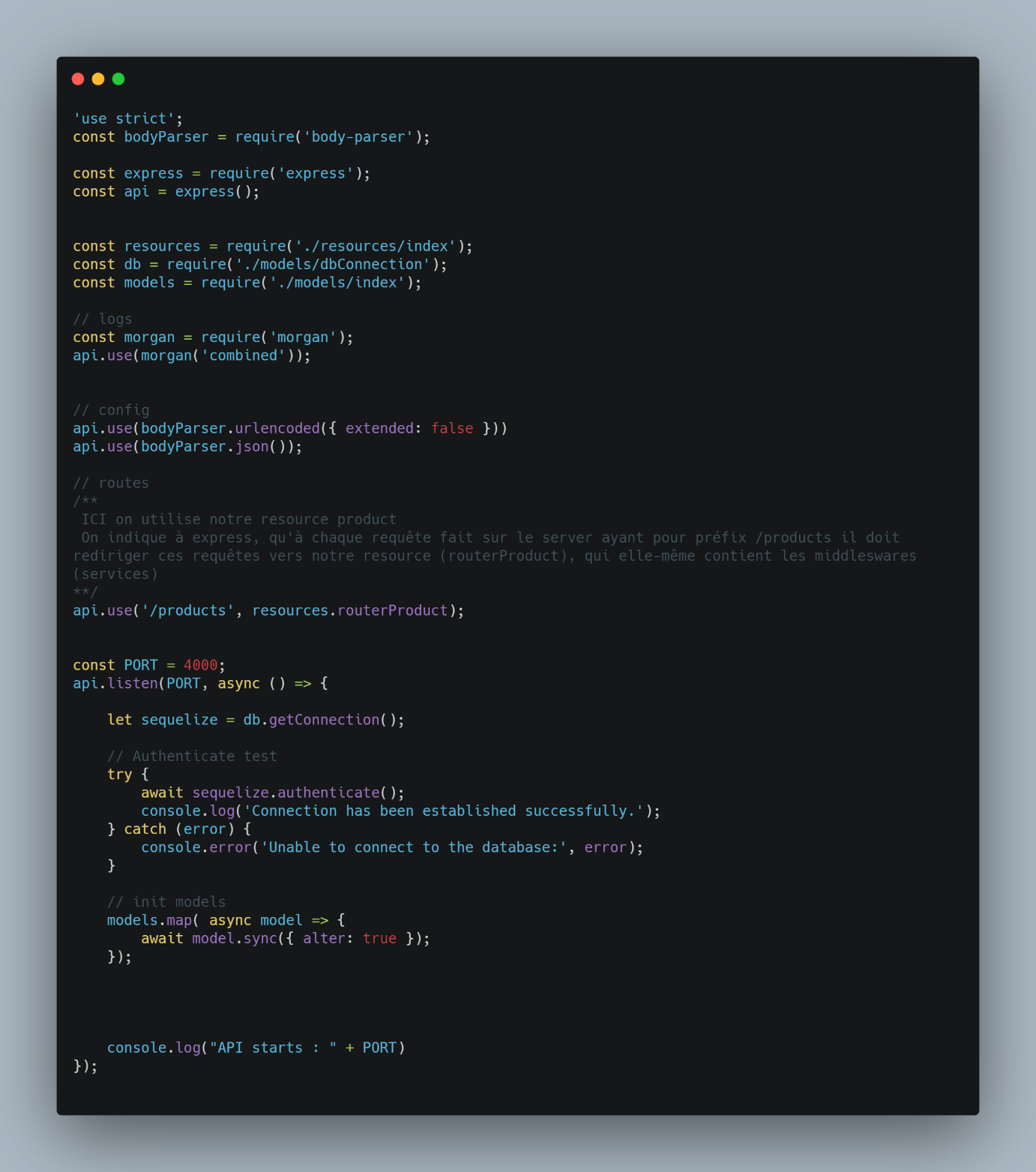
https://github.com/sylvainSUPINTERNET/nodejs-cours/blob/master/resources/index.js



La resource est prête, il faut maintenant l’indiquer à notre server express :

Dans le point d’entrée (index.js à la racine), il faut alors importer notre resource, et dire à express de l’utiliser

https://github.com/sylvainSUPINTERNET/nodejs-cours/blob/master/index.js



On peut ici voir que notre resource est utilisé par express

On dit alors à express que chaque requête dont l’url commencent par /products (<http://localhost/products/XXXX>)

Devra être traitées par la resource product (routerProduct)

Cette resource ensuite, par le jeu des middlewares va réagir à cette requête, et déclenché le service correspondant.

Ce service, va lui-même appeler notre repository, qui lui-même va effectuer le traitement au niveau base de données (create / getProduct …)