

x.x | Le projet **SushiAPI**

1] But de l'application



Cette application **NodeJS** aura pour utilité de fournir sous la forme d'une API (limitée à deux types de requêtes) des produits de la société **SushiShop** à une application de type front-end Angular.

<https://www.sushishop.fr/>

• **Installation d'ExpressJS**

ExpressJS est LE framework utilisé pour construire des applications web basées sur NodeJS.

Rappel : sur Windows, il faut utiliser le "command prompt" de NodeJS et être en root sur Linux.

T1.1 | Installez le micro-framework **Express** pour NodeJS avec NPM :

```
#> npm install -g express-generator
```

T1.2 | Installez un utilitaire permettant le rafraîchissement automatique des sources sans relance du serveur NodeJS après une modification :

```
#> npm install -g nodemon
```

• **Création du projet**

Pour cette application, nous dérogeons à la règle MVC car nous n'aurons pas besoin de la partie « VUE ». L'application ne renvoyant que des flux JSon à partir de requêtes sur une BDD MongoDB.

T1.3 | Lancez la création de l'application nommée `sushiapi` dans votre répertoire personnel :

```
$> express --no-view sushiapi
```

Ce qui donne en sortie :

```
create : sushiapi\
create : sushiapi\public\
create : sushiapi\public\javascripts\
create : sushiapi\public\images\
create : sushiapi\public\stylesheets\
create : sushiapi\public\stylesheets\style.css
create : sushiapi\routes\
create : sushiapi\routes\index.js
create : sushiapi\routes\users.js
create : sushiapi\public\index.html
create : sushiapi\app.js
create : sushiapi\package.json
create : sushiapi\bin\
create : sushiapi\bin\www

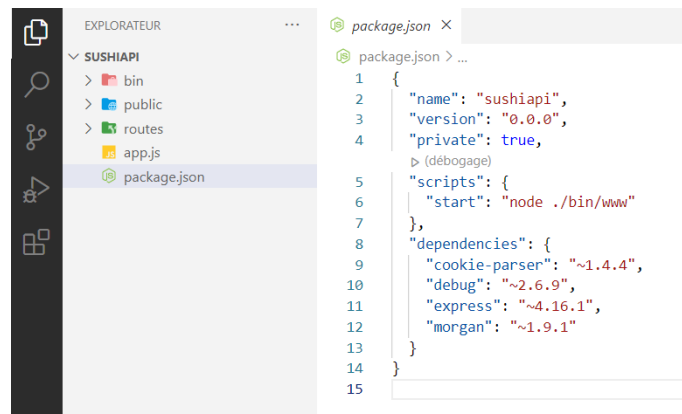
change directory:
> cd sushiapi

install dependencies:
> npm install

run the app:
> SET DEBUG=sushiapi:* & npm start
```

• **Dépendances**

Un projet vient avec un ensemble de modules (dépendances) décrit dans un fichier `package.json` situé à la racine :



T1.4 | Déplacez-vous dans le dossier `sushiapi` et lancez l'installation des dépendances :

```
$> cd sushiapi
$sushiapi> npm install
```

Toutes les dépendances ou modules tiers sont installés dans le répertoire nouvellement créé `node_modules`. Il peut y avoir des messages d'avertissements mais n'en tenez pas compte.

- **Compréhension a minima de la structure**

La création du projet entraîne la création de plusieurs répertoires et fichiers. Retenez pour l'instant que :

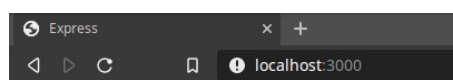
<code>/bin</code>	Fichier de configuration principal de l'application
<code>/public</code>	Éléments accessibles par le navigateur
<code>/routes</code>	Contrôleurs
<code>app.js</code>	Moteur et le point d'entrée de l'application
<code>package-lock.json</code>	Fichier réservé de NPM
<code>package.json</code>	Fichier (déjà vu) descriptif des modules de l'application

- **Premier lancement**

T1.5 | Toujours dans le dossier `sushiapi`, lancez l'application par `nodemon` :

```
$sushiapi> nodemon
```

Ce qui doit donner simplement dans le navigateur à l'adresse `localhost:3000` :



2] Compléments

- **Décryptage du fichier `app.js`**

Voici, annoté par mes soins, le contenu commenté de l'application actuelle :

```
// Définition des variables générales à l'application
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

// Définition des variables indiquant les routes utilisées
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

// Variable du "moteur" du module Express
var app = express();

// Intégration des chemins et modules à utiliser dans l'application
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

// Association entre l'URL demandée et la route à suivre via le contrôleur
app.use('/', indexRouter);
app.use('/users', usersRouter);

// Exportation des modules et lancement de l'application
module.exports = app;
```

Extrait de la documentation d'Express :

« Express est une infrastructure web middleware et de routage, qui a des fonctions propres minimales : une application Express n'est ni plus ni moins qu'une succession d'appels de fonctions middleware. Les fonctions de middleware sont des fonctions qui peuvent accéder à l'objet Request (req), l'objet response (res) et à la fonction middleware suivant dans le cycle demande-réponse de l'application. La fonction middleware suivant est couramment désignée par une variable nommée next.

...

Si la fonction middleware en cours ne termine pas le cycle de demande-réponse, elle doit appeler la fonction next() pour transmettre le contrôle à la fonction middleware suivant. Sinon, la demande restera bloquée. »

<https://expressjs.com/fr/guide/using-middleware.html>

• **Fonctionnement d'un contrôleur**

Nous avons vu précédemment que l'URL de base (la racine /) se voyait dans app.js associée au contrôleur index.js par la ligne :

```
| app.use('/', indexRouter);
```

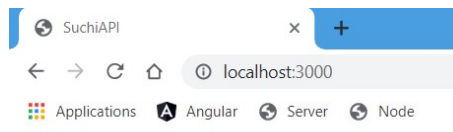
Que fait exactement celui-ci ? Autrement dit, voyons le code de /routes/index.js :

```
| var express = require('express');
| var router = express.Router();
|
| /* GET home page. */
| router.get('/', function(req, res, next) {
|   res.render('index', { title: 'Express' });
| });
|
| module.exports = router;
```

Après les importations (et à la fin l'exportation) des modules nécessaires, on voit que le contrôleur associe une fonction de **callback** à la route (URL) demandée. Cette fonction « rend » (dans ce contexte il faut comprendre : transmettre) à la vue index le paramètre title de valeur Express.

Dans notre projet, **nous n'avons pas utilisé le modèle MVC dans sa totalité...** Le retour actuel donne en fait la page `index.html` **par défaut**, celle qui est dans le dossier `public`, cela fait partie du réglage « à minima » d'Express.

T2.1 | Vérifiez cela en changeant le titre (dans le `head`) dans le fichier `index.html` et ne laissez que le contenu de la balise `<h1>` avec « API OK ! » :



API OK !

3] Routes

- **Écriture de la première route**

L'écriture de la première route correspondra à la première requête, celle qui renverra tous les éléments d'une collection, en l'occurrence ici la **liste des différents plateaux de sushis** de l'entreprise (**boxes** en anglais).

L'URL à utiliser pour correspondre aux règles de construction d'une API REST sera donc :

<http://localhost:3000/boxes>

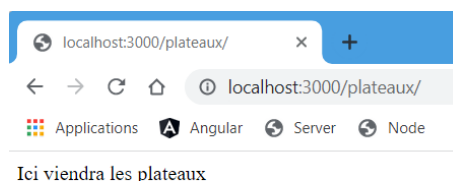
Nous allons utiliser le fichier `users.js` pour cela en changeant son appellation.

T3.1 | Dans le fichier `app.js`, renommez `users.js` en `boxes.js` ; enlevez (ou commentez) les deux lignes (le `require` et le `use`) pour le fichier `index` ; changez à chaque endroit `users` par `boxes`.

T3.2 | Dans le fichier `boxes.js`, changez le code de la méthode par :

```
// Retourne tous les plateaux
router.get('/', function(req, res, next) {
  res.send('Ici viendra les plateaux (ou boxes)');
});
```

L'URL `localhost:3000/boxes` donne :



Explications :

→ le `use` dans `app.js` indique l'implication du fichier `/routes/boxes` contenant les méthodes de réponse.

→ la méthode correspondant au `'/'` indique une URL `boxes/` (ou `boxes` car traitement identique) ; le mot `boxes` concerne le contrôleur, le `/` concerne plus spécifiquement la méthode choisie.

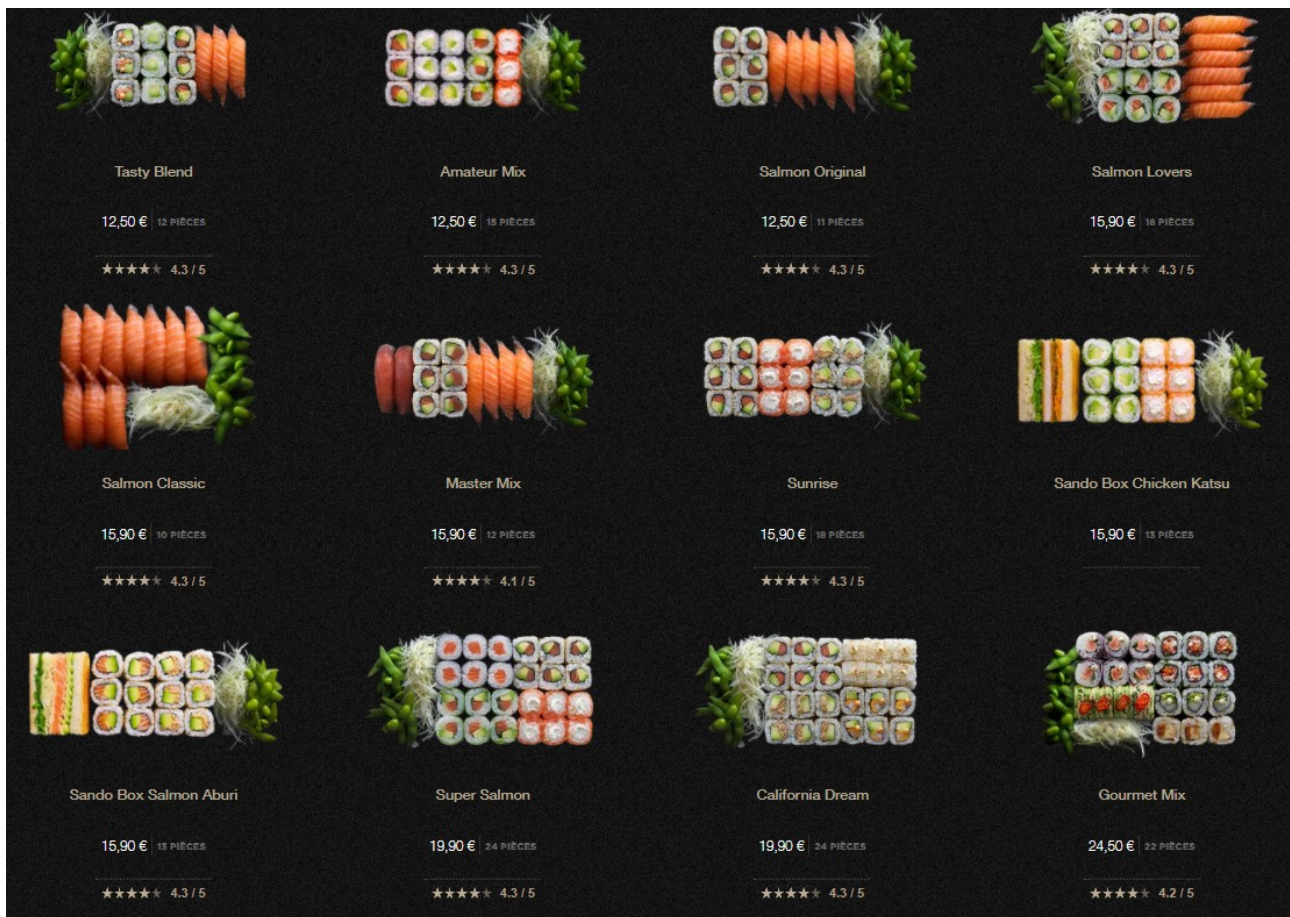
4] MongoDB et NodeJS

- **Pré-requis**

On suppose à ce stade que MongoDB est installé et fonctionnel (Linux ou Windows).

Classiquement maintenant, il faut intervenir sur le fichier `package.json` afin d'intégrer le module nécessaire. Où plutôt les modules car outre MongoDB, nous avons besoin d'un « connecteur » (driver) entre celui-ci et NodeJS.

Voici, à partir du site **SushiShop**, les 13 **Sushi Boxes** servant de base à notre application :



A faire :

→ Trouvez la structure du fichier JSON servant à enregistrer les Sushi Boxes dans MongoDB ; le contenu de celui-ci doit comporter le nom, le prix, le nombre de pièces, la composition, la liste des saveurs et la note.

→ Intégrez par MongoDB Compass ce fichier dans une database nommée `sushishop` et dans une collection nommée `boxes`.

- **MongoDB et Monk**

Pour ce TP, nous n'aurons pas besoin d'un connecteur sophistiqué, c'est-à-dire en fait d'un **ORM** (*Object Relational Mapping*), aussi nous utiliserons donc **Monk** : petit, simple et efficace.

T4.1 | Ajoutez pour l'application par NPM les dernières versions correspondant aux modules `mongodb` et `monk` :

```
$sushiapi> npm install --save mongodb@latest
```

```
$sushiapi> npm install --save monk@latest
```

Les dernières versions à la rédaction de ces lignes sont respectivement 4.2.0 et 7.3.4.

T4.2 | Ajoutez dans le fichier `app.js` le recours aux modules (après les importations du début), avec l'URL de connexion à la base pour Monk :

```
var mongo = require('mongodb');
var monk = require('monk');
var db = monk('localhost:27017/sushishop');
```

T4.3 | Toujours dans `app.js` et **après la ligne** `app.use(express.static(path.join(__dirname, 'public')));` :

```
app.use(function(req, res, next) {
  req.db = db;
  next();
});
```

• Afficher les plateaux

Dans cette méthode pour afficher les plateaux, nous allons envoyer des données sous forme JSON. Il faut donc indiquer (et donner) la possibilité au middleware **Express** de traiter du JSON dans le corps d'une requête.

Ceci doit être déjà possible...

T4.4 | Vérifiez la présence de la ligne ci-dessous dans le fichier `app.js` :

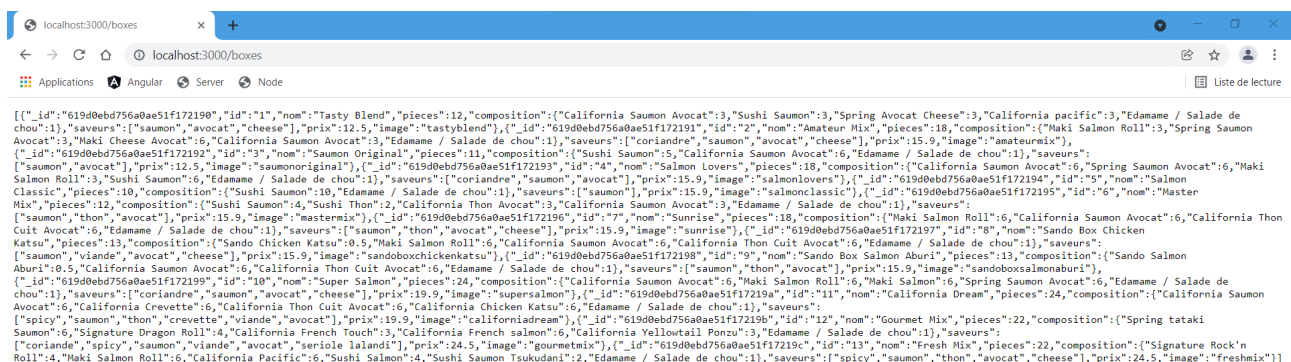
```
app.use(express.json());
```

T4.5 | Remplacez le code de la méthode du fichier `boxes.js` pour obtenir ceci :

```
// Retourne tous les plateaux
router.get('/', function(req, res, next) {
  var db = req.db;
  var collection = db.get('boxes');

  collection.find({}, function(e, listeBoxes) {
    res.send(listeBoxes);
  });
});
```

Une fois MongoDB démarré, le résultat de l'URL ci-dessous donne l'ensemble de la collection :



```
[{"id":"619d0ebd756a0ae51f172198","id":"1","nom":"Tasty Blend","pieces":12,"composition":{"California Saumon Avocat":3,"Sushi Saumon":3,"Spring Avocat Cheese":3,"California pacific":3,"Edamame / Salade de chou":1},"saveurs":{"saumon","avocat","cheese"},"prix":12.5,"image":"tastyblend"},{"id":"619d0ebd756a0ae51f172191","id":"2","nom":"Amateur Mix","pieces":18,"composition":{"Maki Salmon Roll":3,"Spring Saumon Avocat":3,"Maki Cheese Avocat":6,"California Saumon Avocat":3,"Edamame / Salade de chou":1},"saveurs":{"coriandre","saumon","avocat","cheese"},"prix":15.9,"image":"amateurmix"},{"id":"619d0ebd756a0ae51f172192","id":"3","nom":"Saumon Original","pieces":11,"composition":{"Sushi Saumon":5,"California Saumon Avocat":6,"Edamame / Salade de chou":1},"saveurs":{"saumon","avocat"},"prix":12.5,"image":"saumonoriginal"},{"id":"619d0ebd756a0ae51f172193","id":"4","nom":"Salmon Lovers","pieces":18,"composition":{"California Saumon Avocat":6,"Spring Saumon Avocat":6,"Maki Salmon Roll":3,"Sushi Saumon":6,"Edamame / Salade de chou":1},"saveurs":{"coriandre","saumon","avocat"},"prix":15.9,"image":"salmonlovers"},{"id":"619d0ebd756a0ae51f172194","id":"5","nom":"Salmon Classic","pieces":10,"composition":{"Sushi Saumon":10,"Edamame / Salade de chou":1},"saveurs":{"saumon"},"prix":15.9,"image":"salmonclassic"},{"id":"619d0ebd756a0ae51f172195","id":"6","nom":"Master Mix","pieces":12,"composition":{"Sushi Saumon":4,"Sushi Thon":2,"California Thon Avocat":3,"California Saumon Avocat":3,"Edamame / Salade de chou":1},"saveurs":{"saumon","thon","avocat"},"prix":15.9,"image":"mastermix"},{"id":"619d0ebd756a0ae51f172196","id":"7","nom":"Sunrise","pieces":18,"composition":{"Maki Salmon Roll":6,"California Saumon Avocat":6,"California Thon Cuit Avocat":6,"Edamame / Salade de chou":1},"saveurs":{"saumon","thon","avocat","cheese"},"prix":15.9,"image":"sunrise"},{"id":"619d0ebd756a0ae51f172197","id":"8","nom":"Sando Box Chicken Katsu","pieces":13,"composition":{"Sando Chicken Katsu":0.5,"Maki Salmon Roll":6,"California Saumon Avocat":6,"California Thon Cuit Avocat":6,"Edamame / Salade de chou":1},"saveurs":{"saumon","viande","avocat","cheese"},"prix":15.9,"image":"sandocheckboxchickenkatsu"},{"id":"619d0ebd756a0ae51f172198","id":"9","nom":"Sando Box Salmon Aburi","pieces":13,"composition":{"Sando Salmon Aburi":0.5,"California Saumon Avocat":6,"California Thon Cuit Avocat":6,"Edamame / Salade de chou":1},"saveurs":{"saumon","thon","avocat"},"prix":15.9,"image":"sandocheckboxsalmonaburi"},{"id":"619d0ebd756a0ae51f172199","id":"10","nom":"Super Salmon","pieces":24,"composition":{"California Saumon Avocat":6,"Maki Salmon Roll":6,"Maki Salmon":6,"Spring Saumon Avocat":6,"Edamame / Salade de chou":1},"saveurs":{"coriandre","saumon","avocat","cheese"},"prix":19.9,"image":"supersalmon"},{"id":"619d0ebd756a0ae51f17219a","id":"11","nom":"California Dream","pieces":24,"composition":{"California Saumon Avocat":6,"California Crevette":6,"California Thon Cuit Avocat":6,"California Chicken Katsu":6,"Edamame / Salade de chou":1},"saveurs":{"spicy","saumon","thon","crevette","viande","avocat"},"prix":19.9,"image":"californiadream"},{"id":"619d0ebd756a0ae51f17219b","id":"12","nom":"Gourmet Mix","pieces":22,"composition":{"Spring tataki Saumon":6,"Signature Dragon Roll":4,"California French Touch":3,"California French salmon":6,"California Yellowtail Ponzu":3,"Edamame / Salade de chou":1},"saveurs":{"coriandre","spicy","saumon","viande","avocat","seriole lalandi"},"prix":24.5,"image":"gourmetmix"},{"id":"619d0ebd756a0ae51f17219c","id":"13","nom":"Fresh Mix","pieces":22,"composition":{"Signature Rock'n Roll":4,"Maki Salmon Roll":6,"California Pacific":6,"Sushi Salmon":4,"Sushi Saumon Tsukudani":2,"Edamame / Salade de chou":1},"saveurs":{"spicy","saumon","thon","avocat","cheese"},"prix":24.5,"image":"freshmix"}]
```

• Afficher un plateau

Il faut ajouter une méthode prenant en paramètre l'ID (ici ce sera plus simple), effectuant la requête dans la base de donnée et retournant le résultat.

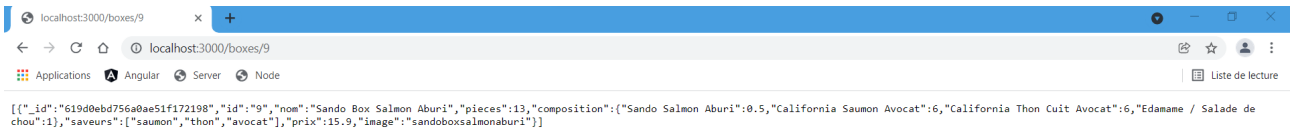
T4.6 | Ajoutez le code de la méthode dans le fichier `boxes.js` (AVANT la ligne d'exportation du module) :

```
// Affichage d'un plateau
router.get('/:id', (req, res) => {
  var db = req.db;
  var collection = db.get('boxes');
  let { id } = req.params;

  collection.find({"id":id},function(e,unBoxe) {
    res.send(unBoxe);
  });
});
```

Nous avons déjà vu cette façon de procéder dans un TD précédent...

Cette fois pour le plateau numéro 9, cela doit donner ceci :



Votre mini API est opérationnelle et peut répondre à ce genre de requêtes venant d'une application de type Front-End.