

# EVALUATION

---

TP2



# Construire une API REST

## Production

- ◇ Comprimez votre projet au format zip. Ne pas mettre le répertoire node\_modules.
- ◇ Déposez le fichier dans Teams.
- ◇ Nommage du fichier : **Nom\_Prenom\_apiRestNodeJSMySQL.zip**.



node-express-api-mysql

# Construire une API REST















## Cahier des charges

- ◇ Modifiez l'application du TP1 afin de mémoriser les données des parkings dans une base de données MySQL en lieu et place du fichier JSON.
- ◇ Utilisez la programmation asynchrone (async – await).
- ◇ Implémentez les routes suivantes (Cf. TP1) :
  - ◇ GET /parkings Lister l'ensemble des parkings
  - ◇ GET /parkings/:id Récupérer les détails d'un parking à partir de son id
  - ◇ POST /parkings Créer un parking
  - ◇ PUT /parkings/:id Modifier les détails d'un parking à partir de son id
  - ◇ DELETE /parkings/:id Supprimer un parking à partir de son id

# Construire une API REST

## Base de données

- ◇ Installez MySQL via Xampp.
- ◇ Créez la base de données nommée db\_parkings.
- ◇ Créez la table nommée parking (champs : id, name, type et city).

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 <b>id</b> 🔑	int(11)			Non	Aucun(e)		AUTO_INCREMENT	 Modifier  Supprimer  Plus
<input type="checkbox"/>	2 <b>name</b> 🔑	varchar(255)	utf8mb4_general_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/>	3 <b>type</b>	varchar(255)	utf8mb4_general_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus
<input type="checkbox"/>	4 <b>city</b>	varchar(255)	utf8mb4_general_ci		Non	Aucun(e)			 Modifier  Supprimer  Plus

- ◇ Insérez des données dans la table parking.

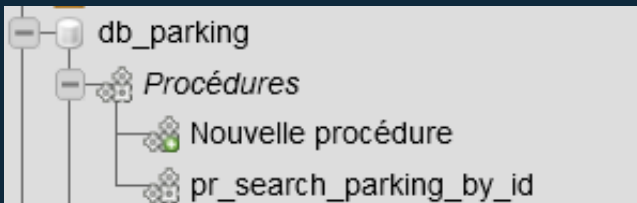
					id	name	type	city
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer		1	Parking 1	AIRPORT	ROISSY EN FRANCE
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer		3	Parking 3	AIRPORT	ORLY
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer		4	Parking 4	AIRPORT	NICE
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer		5	Parking 5	AIRPORT	LILLE
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer		6	Parking 88	AIRPORT	ROISSY EN FRANCE

# Construire une API REST

## Base de données

- ◇ Créez une procédure stockée nommée `pr_search_parking_by_id(id)`. Cette procédure permet de retourner les caractéristiques d'un parking en fonction de son id.

```
DELIMITER $$  
CREATE PROCEDURE `pr_search_parking_by_id`(in idparking int)  
BEGIN  
  SELECT name, type, city  
  FROM parking  
  where id = idparking;  
END $$
```



# Construire une API REST



## Architecture de l'application

```
▼ node-express-api-mysql
  ▼ dao
    JS config.js
    JS db.js
  > node_modules
  ▼ routes
    JS parkings.js
  ▼ service
    JS parking.js
  JS helper.js
  JS index.js
  {} package-lock.json
  {} package.json
  script.sql
```

- ◇ **index.js** : le point d'entrée de l'application.
- ◇ **dao/config.js** : la configuration des informations telles que les informations d'identification de la base de données et les lignes que vous souhaitez afficher par page lorsque vous paginez les résultats.
- ◇ **dao/db.js** : permet de communiquer avec la base de données MySQL.
- ◇ **routes/parkings.js** : le lien entre l'URI et la fonction correspondante dans le service `service/parking.js`. Il contient les routes.
- ◇ **service/parkings.js** : fait office de pont entre la route et la base de données.
- ◇ **helper.js** : contient toutes les fonctions d'assistance, comme le calcul du décalage pour la pagination.

# Construire une API REST

## Fichier routes\parkings.js

```
const express = require("express");
const router = express.Router();
const parking = require("../service/parking");

/* GET parkings */
router.get("/", async function (req, res, next) {
});

/* POST parking */
router.post("/", async function (req, res, next) {
});

/* PUT parking */
router.put("/:id", async function (req, res, next) {
});

/* DELETE parking */
router.delete("/:id", async function (req, res, next){
});

/* GET parking par id */
router.get("/:id", async function (req, res, next) {
});
module.exports = router;
```

## Fichier service\parkings.js

```
const db = require("../dao/db");
const helper = require("../helper");
const config = require("../dao/config");

async function getMultiple(page = 1) {
}

async function create(parking) {
}

async function update(id, parking) {
}

async function remove(id) {
}

async function search(id) {
}

module.exports = {
  getMultiple,
  create,
  update,
  remove,
  search
};
```

# Construire une API REST

## Fichier helper.js

```
function getOffset(currentPage = 1, listPerPage) {  
  return (currentPage - 1) * [listPerPage];  
}  
  
function emptyOrRows(rows) {  
  if (!rows) {  
    return [];  
  }  
  return rows;  
}  
  
module.exports = {  
  getOffset,  
  emptyOrRows  
};
```



# Base de données MySQL



- ◇ Exécutez l'application

```
PS C:\WK_NODE\node-express-api-mysql> nodemon
```

- ◇ Testez votre application via Postman.
- ◇ Vérifiez les modifications apportées à la table parking.