

# API REST avec ExpressJS

**Achref El Mouelhi**

Docteur de l'université d'Aix-Marseille  
Chercheur en programmation par contrainte (IA)  
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`

Express



# Plan

- 1 Introduction
- 2 Postman
- 3 Structure d'une application ExpressJS
  - `*.route.js`
  - **Modèle**
  - `*.controller.js`
- 4 Express et MySQL
  - **Module** `mysql`
  - **Fichier de connexion**
  - `*.dao.js`
- 5 Problème de CORS
- 6 Quelques cas pratiques

## Service web (WS pour Web Service) ?

- Un programme (ensemble de fonctionnalités exposées en temps réel et sans intervention humaine)
- Accessible via internet, ou intranet
- Indépendant de tout système d'exploitation
- Indépendant de tout langage de programmation
- Utilisant un système standard d'échange (**XML** ou **JSON**), ces messages sont généralement transportés par des protocoles internet connus **HTTP** (ou autres comme **FTP**, **SMTP**...)
- Pouvant communiquer avec d'autres WS

# ExpressJS

Les WS peuvent utiliser les technologies web suivantes :

- **HTTP** (Hypertext Transfer Protocol) : le protocole, connu, utilisé par le World Wide Web et inventé par Roy Fielding.
- **REST** (Representational State Transfer) : une architecture de services Web, créée aussi par Roy Fielding en 2000 dans sa thèse de doctorat.
- **SOAP** (Simple object Access Protocol) : un protocole, défini par Microsoft et IBM ensuite standardisé par **W3C**, permettant la transmission de messages entre objets distants (physiquement distribués).
- **WSDL** (Web Services Description Language) : est un langage de description de service web utilisant le format **XML** (standardisé par le **W3C** depuis 2007).
- **UDDI** (Universal Description, Discovery and Integration) : un annuaire de WS.

## HTTP, REST (Representational State Transfer) et RESTful ?

- Les API REST sont basées sur le protocole **HTTP** (architecture client/serveur) et utilisent le concept de ressource.
- Une ressource est identifiée par une URI unique.
- L'API REST utilise donc des méthodes suivantes pour l'échange de données entre client et serveur
  - GET pour la récupération,
  - POST pour l'ajout,
  - DELETE pour la suppression,
  - PUT pour la modification,
  - ...
- Plusieurs formats possibles pour les données échangées : texte, **XML**, **JSON**...
- **RESTful** est l'adjectif qui désigne une API REST.

## Utilisation de **Postman** : simulateur de client **REST**

- Aller sur internet et chercher **Postman**
- Télécharger puis installer l'application **Postman**
- Démarrer l'application

© AC

## Utilisation de **Postman** : simulateur de client **REST**

- Aller sur internet et chercher **Postman**
- Télécharger puis installer l'application **Postman**
- Démarrer l'application

Il existe plusieurs autres tel que ARC (pour Advanced REST Client)...

# ExpressJS

## Comment utiliser **Postman** pour envoyer une requête **HTTP** ?

- Choisir la méthode `GET`
- Saisir l'URL de la ressource : `http://localhost:3000`
- Cliquer sur `Send`

# ExpressJS

Considérons le contenu suivant de `app.js`

```
const express = require('express');
const app = express();

app.get('/personne/', (req, res) => {
  // ...
});
app.post('/personne/', (req, res) => {
  // ...
});
app.put('/personne/:id', (req, res) => {
  // ...
});
// ...
app.get('/adresse/', (req, res) => {
  // ...
});
app.post('/adresse/', (req, res) => {
  // ...
});
app.put('/adresse/:id', (req, res) => {
  // ...
});
// ...
app.all('/*', (req, res) => {
  res
    .status(404)
    .send('Not Found');
});
app.listen(3000, () => console.log('Adresse du serveur : http://localhost:3000'));
```

# ExpressJS

## Constat

Beaucoup de code  $\Rightarrow$  fichier trop volumineux.

© Achref EL MOUELHI ©

# ExpressJS

## Constat

Beaucoup de code  $\Rightarrow$  fichier trop volumineux.

## Solution

Dissocier le routing du reste du code.

# ExpressJS

## Constat

Beaucoup de code  $\Rightarrow$  fichier trop volumineux.

## Solution

Dissocier le routing du reste du code.

## Indice

Utiliser le module `router`.

# ExpressJS

## Commençons par créer les deux fichiers

routes/personne.route.js **et** routes/adresse.route.js

```
const express = require('express');  
const router = express.Router();  
  
module.exports = router;
```

© Actif

# ExpressJS

## Commençons par créer les deux fichiers

routes/personne.route.js **et** routes/adresse.route.js

```
const express = require('express');  
const router = express.Router();  
  
module.exports = router;
```

Pas besoin de `app` car il s'agit de fichier de routage.

# ExpressJS

Déplaçons les routes de `app.js` vers `routes/personne.route.js` et `routes/adresse.route.js` et remplaçons `app` par `router`

```
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  // ...
});
router.get('/:id', (req, res) => {
  // ...
});
router.post('/', (req, res) => {
  // ...
});
router.put('/:id', (req, res) => {
  // ...
});
router.delete('/:id', (req, res) => {
  // ...
});

module.exports = router;
```

# ExpressJS

## Nouveau code de `app.js`

```
const express = require('express');
const app = express();
const personne = require('./routes/personne.route');
const adresse = require('./routes/adresse.route');

app.use('/personne', personne);
app.use('/adresse', adresse);
app.all('/*', (req, res) => {
  res
    .status(404)
    .send('Not Found');
});

app.listen(3000, () => console.log('Adresse du serveur : http
://localhost:3000'));
```

# ExpressJS

**Pour tester, ajoutons l'instruction suivant dans le POST de `routes/personne.route.js` et vérifions avec Postman la réception de la réponse**

```
router.post('/', (req, res) => {  
    res.send('personne')  
});
```

# ExpressJS

## Constat

Ce n'est pas le rôle du routeur de retourner une réponse.

© Achref EL M...

# ExpressJS

## Constat

Ce n'est pas le rôle du routeur de retourner une réponse.

## Solution

Utiliser les contrôleurs et les modèles.

# ExpressJS

**Commençons par créer la classe** `Personne` **dans**  
`models/personne.js`

```
exports.Personne = class {  
  constructor(num, nom, prenom) {  
    this.num = num;  
    this.nom = nom;  
    this.prenom = prenom;  
  }  
}
```

# ExpressJS

## Remarque

Dans le contrôleur, on va créer un tableau d'objets de type `Personne` pour simuler la base de données.

# ExpressJS

**Code initial du contrôleur** `personne.controller.js` (à créer dans un répertoire `controllers`)

```
const personne = require("../models/personne");

personnes = [
  new personne.Personne(1, "wick", "john"),
  new personne.Personne(2, "maggio", "carol"),
  new personne.Personne(3, "dalton", "jack")
];

exports.getAll = (req, res, next) => { }
exports.getOneById = (req, res, next) => {}
exports.add = (req, res, next) => { }
exports.edit = (req, res, next) => { }
exports.delete = (req, res, next) => { }
```

# ExpressJS

**Nouveau code de** `routes/personne.route.js` **qui référence le contrôleur**

```
const express = require('express');
const router = express.Router();
const personneController = require('../controllers/personne.controller');

router.get('/', personneController.getAll);
router.get('/:id', personneController.getOneById);
router.post('/', personneController.add);
router.put('/:id', personneController.edit);
router.delete('/:id', personneController.delete);

module.exports = router;
```

# ExpressJS

Commençons par implémenter la fonction `getAll`

```
exports.getAll = (req, res, next) => {  
  const query = connection.query("SELECT * FROM   personne",  
    (err, result) => {  
      console.log(query.sql)  
      if (err) {  
        return res.status(500).json({  
          erreur: err  
        });  
      }  
      return res.status(200).json(result);  
    });  
}
```

# ExpressJS

Commençons par implémenter la fonction `getAll`

```
exports.getAll = (req, res, next) => {  
  const query = connection.query("SELECT * FROM  personne",  
    (err, result) => {  
      console.log(query.sql)  
      if (err) {  
        return res.status(500).json({  
          erreur: err  
        });  
      }  
      return res.status(200).json(result);  
    });  
}
```

Pour tester, aller à `http://localhost:3000/personne`.

## Implémentons maintenant la fonction `getOneById`

```
exports.getOneById = (req, res, next) => {
  const id = parseInt(req.params.id);
  const query = connection.query("SELECT * FROM personne WHERE num =
    ?", id, (err, result) => {
    console.log(query.sql)
    if (err) {
      return res.status(500).json({
        erreur: err
      });
    } else if (result.length == 0) {
      return res.status(404).json({
        erreur: `Aucune personne avec l'identifiant ${id}`
      });
    } else {
      return res.status(200).json(result[0]);
    }
  });
}
```

## Implémentons maintenant la fonction `getOneById`

```
exports.getOneById = (req, res, next) => {
  const id = parseInt(req.params.id);
  const query = connection.query("SELECT * FROM personne WHERE num =
    ?", id, (err, result) => {
    console.log(query.sql)
    if (err) {
      return res.status(500).json({
        erreur: err
      });
    } else if (result.length == 0) {
      return res.status(404).json({
        erreur: `Aucune personne avec l'identifiant ${id}`
      });
    } else {
      return res.status(200).json(result[0]);
    }
  });
}
```

Pour tester, aller à <http://localhost:3000/personne/1> et <http://localhost:3000/personne/4>.

# ExpressJS

Pour pouvoir ajouter une nouvelle personne, implémentons la fonction `add` qui récupère les données à partir de `req.body`

```
exports.add = (req, res, next) => {  
  console.log(req.body);  
  const p = new personne.Personne(  
    req.body.num,  
    req.body.nom,  
    req.body.prenom  
  );  
  personnes.push(p);  
  return res.status(201).json({  
    message: `personne ajoutée avec succès`  
  });  
}
```

# ExpressJS

## Pour ajouter une personne

- Utilisez **Postman** en précisant la méthode `POST`
- Saisissez l'URL de notre ressource `http://localhost:3000/personne`
- Ensuite cliquez sur `Body`, cochez `raw`, choisissez `JSON` et saisissez des données sous format `JSON` correspondant à l'objet `personne` à ajouter

```
{  
  "num": 4,  
  "nom": "mitroglou",  
  "prenom": "kostas"  
}
```

- Cliquez sur `Send`

# ExpressJS

## Constat

```
console.log(req.body) affiche undefined.
```

© Achref EL MOUELHI ©

# ExpressJS

## Constat

```
console.log(req.body) affiche undefined.
```

## Explication

Il faut activer l'utilisation de `body`.

# ExpressJS

## Constat

```
console.log(req.body) affiche undefined.
```

## Explication

Il faut activer l'utilisation de `body`.

## Solution

- Avant la version 4 de **ExpressJS**, il fallait importer `body-parser`.
- Depuis la version 4, il faut ajouter `app.use(express.json())`.

Ajoutons `app.use(express.json())` dans `app.js` et relançons la requête précédente

```
const express = require('express');
const app = express();
const personne = require('./routes/personne.route');
const adresse = require('./routes/adresse.route');

app.use(express.json());
app.use('/personne', personne);
app.use('/adresse', adresse);

app.post('/home', (req, res) => {
  res.send('POST: /home');
});

app.all('/*', (req, res) => {
  res
    .status(404)
    .send('Not Found');
});

app.listen(3000, () => console.log('Adresse du serveur : http://localhost:3000'));
```

# ExpressJS

## Et la fonction `edit`

```
exports.edit = (req, res, next) => {  
  const id = parseInt(req.params.id);  
  let personne = personnes.find(p => p.num === id);  
  if (personne !== undefined) {  
    personne.nom = req.body.nom;  
    personne.prenom = req.body.prenom;  
    return res.status(200).json(personne);  
  }  
  return res.status(404).json({  
    error: `personne avec l'identifiant ${ req.params.id }  
          non trouvée`  
  });  
}
```

# ExpressJS

## Pour modifier une personne

- Utilisez **Postman** en précisant la méthode `PUT`
- Saisissez l'URL de notre ressource `http://localhost:3000/personne/1`
- Ensuite cliquez sur `Body`, cochez `raw`, choisissez `JSON` et saisissez des données sous format `JSON` correspondant à l'objet `personne` à ajouter

```
{  
  "num": 1,  
  "nom": "aouar",  
  "prenom": "houssem"  
}
```

- Cliquez sur `Send`

# ExpressJS

## Et la fonction `delete`

```
exports.delete = (req, res, next) => {
  const id = parseInt(req.params.id);
  let personne = personnes.find(p => p.num === id);
  if (personne !== undefined) {
    personnes.splice(personnes.indexOf(personne), 1)
    return res.status(200).json({
      message: `personne avec l'identifiant ${ req.params
        .id } supprimée avec succès`
    });
  }
  return res.status(404).json({
    error: `personne avec l'identifiant ${ req.params.id }
      non trouvée`
  });
}
```

# ExpressJS

## Pour supprimer une personne

- Utilisez **Postman** en précisant la méthode `PUT`
- Saisissez l'URL de notre ressource  
`http://localhost:3000/personne/1`
- Cliquez sur `Send`

# ExpressJS

## Objectif

Se connecter à une base de données **SQL** et préparer les requêtes pour le CRUD.

© Achref EL MOUELHI

# ExpressJS

## Objectif

Se connecter à une base de données **SQL** et préparer les requêtes pour le CRUD.

## Étapes

- Installer le module **MySQL**
- Importer le module installé : **MySQL**
- Préparer et établir la connexion avec la base de données
- Exécuter des requêtes **SQL**

# ExpressJS

## Script de la création de la base de données

```
CREATE DATABASE cours_expressjs;

USE cours_expressjs;

CREATE TABLE personne (
  num INT PRIMARY KEY AUTO_INCREMENT,
  nom VARCHAR(20),
  prenom VARCHAR(20)
)ENGINE=innodb;

CREATE TABLE adresse (
  id INT PRIMARY KEY AUTO_INCREMENT,
  rue VARCHAR(20),
  ville VARCHAR(20),
  codePostal VARCHAR(20)
)ENGINE=innodb;

CREATE TABLE personne_adresse (
  id INT PRIMARY KEY AUTO_INCREMENT,
  num_personne INT,
  id_adresse INT,
  CONSTRAINT fk_personne FOREIGN KEY (num_personne) REFERENCES personne(num),
  CONSTRAINT fk_adresse FOREIGN KEY (id_adresse) REFERENCES adresse(id)
)ENGINE=innodb;
```

# ExpressJS

## Script d'insertion de données

```
INSERT INTO personne (nom, prenom) VALUES  
  ('Cohen', 'Sophie'),  
  ('Dalton', 'Jack'),  
  ('Wick', 'John');
```

```
INSERT INTO adresse (rue, ville, codePostal) VALUES  
  ('Paradis', 'Marseille', '13006'),  
  ('Victoire', 'Paris', '75014'),  
  ('Villeurbanne', 'Lyon', '69000');
```

```
INSERT INTO personne_adresse (num_personne, id_adresse) VALUES  
  (1, 1),  
  (1, 2),  
  (2, 2);
```

# ExpressJS

## Installation du module MySQL

```
npm install mysql --save
```

© Achref EL MOU

# ExpressJS

## Installation du module MySQL

```
npm install mysql --save
```

## Importation du module installé

```
const mysql = require('mysql');
```

# ExpressJS

## Préparation de la connexion

```
const connection = mysql.createConnection({  
  host: 'localhost',  
  user: 'root',  
  password: 'root',  
  database: 'cours_expressjs'  
});
```

© Achref EL

# ExpressJS

## Préparation de la connexion

```
const connection = mysql.createConnection({  
  host: 'localhost',  
  user: 'root',  
  password: 'root',  
  database: 'cours_expressjs'  
});
```

## Établissement de la connexion

```
connection.connect((err) => {  
  if (err) throw err;  
  console.log("Connexion établie avec la base de  
    données");  
});;
```

# ExpressJS

## Contenu de database.js

```
const mysql = require('mysql');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'cours_expressjs'
});

connection.connect((err) => {
  if (err) throw err;
  console.log("Connexion établie avec la base de données");
});

module.exports = connection;
```

# ExpressJS

**Dans** `controllers/personne.controllers.js` **commençons**  
**par importer le fichier de connexion** `database.js`

```
const connection = require('../database.js');
```

© Achref EL MOUËLTA

# ExpressJS

**Dans** `controllers/personne.controllers.js` **commençons**  
**par importer le fichier de connexion** `database.js`

```
const connection = require('../database.js');
```

**Supprimons ensuite le tableau** `personnes`

```
personnes = [  
  new personne.Personne(1, "wick", "john"),  
  new personne.Personne(2, "maggio", "carol"),  
  new personne.Personne(3, "dalton", "jack")  
];
```

# ExpressJS

## Modifions la méthode `getAll`

```
exports.getAll = (req, res, next) => {  
  const query = connection.query("SELECT * FROM  
    personne", (err, result) => {  
    console.log(query.sql)  
    return res.status(200).json(result);  
  });  
}
```

# ExpressJS

En allant à `localhost:3000/personne`, le résultat est

```
[
  {
    "num": 1,
    "nom": "Cohen",
    "prenom": "Sophie"
  },
  {
    "num": 2,
    "nom": "Dalton",
    "prenom": "Jack"
  },
  {
    "num": 3,
    "nom": "Wick",
    "prenom": "John"
  }
]
```

# ExpressJS

## La méthode `getOneById`

```
exports.getOneById = (req, res, next) => {
  const query = connection.query("SELECT * FROM personne
    WHERE num = ?", req.params.id, (err, result) => {
    console.log(query.sql);
    if (err) {

      return res.status(404).json({
        error: `personne avec l'identifiant ${ req.
          params.id } non trouvée`
      });
    } else {
      return res.status(200).json(result[0]);
    }
  });
}
```

# ExpressJS

En allant à `localhost:3000/personne/2`, le résultat est

```
{  
  "num": 2,  
  "nom": "Dalton",  
  "prenom": "Jack"  
}
```

# ExpressJS

## La méthode `add`

```
exports.add = (req, res, next) => {
  const p = new personne.Personne(
    req.body.num,
    req.body.nom,
    req.body.prenom
  );
  const query = connection.query("INSERT INTO personne SET ?", p, (
    err, result) => {
    console.log(query.sql);
    if (err) {
      console.log("error: ", err);
      return res.status(500).json({
        error: `problème d'insertion`
      });
    }
    else {
      p.num = result.insertId;
      return res.status(201).json(p);
    }
  });
}
```

# ExpressJS

## Pour ajouter une personne

- Utilisez **Postman** en précisant la méthode `POST`
- Saisissez l'URL de notre ressource `http://localhost:3000/personne`
- Ensuite cliquez sur `Body`, cochez `raw`, choisissez `JSON` et saisissez des données sous format `JSON` correspondant à l'objet `personne` à ajouter

```
{  
  "nom": "mitroglou",  
  "prenom": "kostas"  
}
```

- Cliquez sur `Send`

# ExpressJS

**Le résultat est**

```
{  
  "num": 4,  
  "nom": "mitroglou",  
  "prenom": "kostas"  
}
```

## La méthode `edit`

```
exports.edit = (req, res, next) => {
  const id = parseInt(req.params.id);
  const p = new personne.Personne(
    req.body.num,
    req.body.nom,
    req.body.prenom
  );
  const query = connection.query("UPDATE personne SET ? WHERE num =
    ?", [p, id], (err, result) => {
    console.log(query.sql)
    if (err) {
      console.log("error: ", err);
      return res.status(500).json({
        error: `problème de mise à jour`
      });
    }
    else {
      return res.status(200).json({
        message: `personne avec l'identifiant ${ req.params.id
          } modifiée avec succès`
      });
    }
  });
};
```

# ExpressJS

## Pour modifier une personne

- Utilisez **Postman** en précisant la méthode `PUT`
- Saisissez l'URL de notre ressource `http://localhost:3000/personne/4`
- Ensuite cliquez sur `Body`, cochez `raw`, choisissez `JSON` et saisissez des données sous format `JSON` correspondant à l'objet `personne` à ajouter

```
{  
  "num": 44,  
  "nom": "thauvin",  
  "prenom": "florian"  
}
```

- Cliquez sur `Send`

## La méthode `delete`

```
exports.delete = (req, res, next) => {
  const id = parseInt(req.params.id);
  const query = connection.query("DELETE FROM personne WHERE
    num = ?", id, (err, result) => {
    console.log(query.sql)
    if (err) {
      console.log("error: ", err);
      return res.status(500).json({
        error: `problème de suppression`
      });
    }
    else {
      return res.status(200).json({
        message: `personne avec l'identifiant ${ req.
          params.id } supprimée avec succès`
      });
    }
  });
};
```

# ExpressJS

## Pour supprimer une personne

- Utilisez **Postman** en précisant la méthode `DELETE`
- Saisissez l'URL de notre ressource  
`http://localhost:3000/personne/4`
- Cliquez sur `Send`

# ExpressJS

## Bonne pratique

Placer le code d'accès aux données (DAO : Data Access Object) dans des nouveaux fichiers : `*.dao.js`.

# ExpressJS

## Contenu de dao/personne.dao.js (première partie)

```
const connection = require('../database.js');

exports.getAll = () => {
  return new Promise((resolve, reject) => {
    const req = connection.query("SELECT * FROM personne", (err,
      result) => {
      console.log(req.sql)
      err ? reject(err) : resolve(result);
    });
  });
};

exports.getOneById = (id) => {
  return new Promise((resolve, reject) => {
    const req = connection.query("SELECT * FROM personne WHERE num
      = ?", id, (err, result) => {
      console.log(req.sql)
      err || result.length == 0 ? reject(err) : resolve(result);
    });
  });
};
```

# ExpressJS

## Contenu de dao/personne.dao.js (deuxième partie)

```
exports.add = (p) => {
  return new Promise((resolve, reject) => {
    const req = connection.query("INSERT INTO personne SET nom = ?, prenom = ?", [p.nom, p.prenom], (err, result) => {
      console.log(req.sql)
      err ? reject(err) : resolve(result);
    });
  });
};

exports.edit = (id, p) => {
  return new Promise((resolve, reject) => {
    const req = connection.query("UPDATE personne SET ? WHERE num = ?", [p, id], (err, result) => {
      console.log(req.sql)
      err || result.affectedRows == 0 ? reject(err) : resolve(result);
    });
  });
};

exports.delete = (id) => {
  return new Promise((resolve, reject) => {
    const req = connection.query("DELETE FROM personne WHERE num = ?", id, (err, result) => {
      console.log(req.sql)
      err || result.affectedRows == 0 ? reject(err) : resolve(result);
    });
  });
};
```

# ExpressJS

## Contenu de controllers/personne.controller.js (première partie)

```
const personne = require("../models/personne");
const personneDao = require('../dao/personne.dao');

exports.getAll = (req, res, next) => {
  personneDao.getAll()
    .then(result => res.status(200).json(result))
    .catch(err => {
      return res.status(500).json({
        error: `problème de récupération de données : ${err}`
      })
    });
}

exports.getOneById = (req, res, next) => {
  const id = parseInt(req.params.id);
  personneDao.getOneById(id)
    .then(result => res.status(200).json(result[0]))
    .catch(err => {
      if (!err) {
        return res.status(404).json({
          error: `Aucune personne avec l'identifiant ${id}`
        });
      }
      return res.status(500).json({
        error: `problème de récupération de données : ${err}`
      });
    });
}
```

# ExpressJS

## Contenu de controllers/personne.controller.js (deuxième partie)

```
exports.add = (req, res, next) => {  
  const p = new personne.Personne(  
    req.body.num,  
    req.body.nom,  
    req.body.prenom  
  );  
  personneDao.add(p)  
    .then(result => {  
      p.num = result.insertId;  
      return res.status(201).json(p);  
    })  
    .catch(err => {  
      return res.status(500).json({  
        error: `problème d'insertion : ${err}`  
      });  
    });  
}
```

# ExpressJS

## Contenu de controllers/personne.controller.js (troisième partie)

```
exports.edit = (req, res, next) => {
  const id = parseInt(req.params.id);
  const p = new personne.Personne(
    req.body.num,
    req.body.nom,
    req.body.prenom
  );
  personneDao.edit(id, p)
    .then(result => {
      return res.status(200).json({
        message: `personne avec l'identifiant ${id} modifiée avec succès`
      });
    })
    .catch(err => {
      if (!err) {
        return res.status(404).json({
          error: `Aucune personne avec l'identifiant ${id}`
        });
      }
      return res.status(500).json({
        error: `problème de mise à jour : ${err}`
      });
    });
}
```

# ExpressJS

## Contenu de controllers/personne.controller.js (quatrième partie)

```
exports.delete = (req, res, next) => {  
  const id = parseInt(req.params.id);  
  personneDao.delete(id)  
    .then(result => {  
    return res.status(200).json({  
      message: `personne avec l'identifiant ${id} supprimée  
avec succès`  
    });  
  })  
  .catch(err => {  
    if (!err) {  
      return res.status(404).json({  
        error: `Aucune personne avec l'identifiant ${id}`  
      });  
    }  
    return res.status(500).json({  
      error: `problème de suppression : ${err}`  
    });  
  });  
}
```

# ExpressJS

## Exercice 1

Implémentez toutes les fonctions de **CRUD** dans `adresse.dao.js` et `adresse.controller.js`, mettez à jour `adresse.route.js` et tester avec **Postman**.

**Vous pouvez utiliser la classe Adresse (à placer dans `models/adresse.js`)**

```
exports.Adresse = class {
  constructor(id, rue, codePostal, ville) {
    this.id = id;
    this.rue = rue;
    this.codePostal = codePostal;
    this.ville = ville;
  }
}
```

# ExpressJS

## Exercice 2

Appeler ces méthodes depuis **Angular** en utilisant les méthodes **HTTP** dans les services.

## Problème de **CORS** : Cross-Origin Resource Sharing

- Système de sécurité
- Permettant de bloquer les échanges **HTTP** entre deux ou plusieurs domaines différents : `localhost:3000` et `localhost:4200`

# ExpressJS

Dans app.js

```
const express = require('express');
const app = express();
const personne = require('./routes/personne.route');
const adresse = require('./routes/adresse.route');

app.use(express.json());

app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content, Accept, Content-Type, Authorization');
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE');
  next();
});

app.post('/home', (req, res) => {
  res.send('POST: /home');
});

app.use('/personne', personne);
app.use('/adresse', adresse);
app.all('/*', (req, res) => {
  res
    .status(404)
    .send('Not Found');
});

app.listen(3000, () => console.log('Adresse du serveur : http://localhost:3000'))
);
```

# ExpressJS

## Question 1

Dans `personne-adresse.dao.js`, créez une fonction `getAllAdressesOfPersonne (idPersonne)` qui retourne la liste d'adresses d'une personne dont l'identifiant est passé en paramètre.

© Achref EL ME

# ExpressJS

## Question 1

Dans `personne-adresse.dao.js`, créez une fonction `getAllAdressesOfPersonne (idPersonne)` qui retourne la liste d'adresses d'une personne dont l'identifiant est passé en paramètre.

## Question 2

Dans `personne.controller.js`, modifiez les fonctions `getAll` et `getOneById` pour qu'elles retournent la liste d'adresses de chaque personne.

# ExpressJS

## Contenu de `personne-adresse.dao.js`

```
const connection = require('../database.js');

exports.getAllAdressesOfPersonne = (idPersonne) => {
  return new Promise((resolve, reject) => {
    const req = connection.query("SELECT a.id, rue,
      codePostal, ville FROM adresse a JOIN
      personne_adresse pa ON pa.id_adresse = a.id WHERE
      num_personne = ? ", idPersonne, (err, result) => {
      console.log(req.sql)
      err ? reject(err) : resolve(result);
    });
  });
};
```

Fonction `getAll` de `personne.controller.js`

```

const personne = require("../models/personne");
const personneDao = require('../dao/personne.dao');
const personneAdresseDao = require('../dao/personne-adresse.dao');

exports.getAll = async (req, res, next) => {
  let personnes = await personneDao.getAll().catch(err => {
    return res.status(500).json({
      error: `problème de récupération de données : ${err}`
    })
  });
  for (let p of personnes) {
    await personneAdresseDao.getAllAdressesOfPersonne(p.num)
      .then(a => p.adresses = a)
      .catch(err => {
        return res.status(500).json({
          error: `problème de récupération d'adresses : ${err}`
        })
      });
  }
  res.status(200).json(personnes)
}

```

# ExpressJS

Fonction `getOneById` de `personne.controller.js`

```
exports.getOneById = (req, res, next) => {
  const id = parseInt(req.params.id);
  personneDao.getOneById(id)
    .then(async (personnes) => {
      let p = personnes[0];
      p.adresses = await personneAdresseDao.getAllAdressesOfPersonne(p.num)
        .catch(err => {
          res.status(500).json({
            error: `problème de récupération d'adresses : ${err}`
          });
        });
      return res.status(200).json(p);
    })
    .catch(err => {
      if (!err) {
        return res.status(404).json({
          error: `Aucune personne avec l'identifiant ${id}`
        });
      }
      return res.status(500).json({
        error: `problème de récupération de personne : ${err}`
      });
    });
}
```

# ExpressJS

Dans `personne.route.js`, définissons les deux nouvelles routes `/personne/:idPersonne/adresses` et `/personne/:idPersonne/adresses/:idAdresse`

```
const express = require('express');
const router = express.Router();
const personneController = require('../controllers/personne.controller')

router.get('/', personneController.getAll);
router.get('/:id', personneController.getOneById);
router.get('/:idPersonne/adresses', personneController.getAdressesByIdPersonne);
router.get('/:idPersonne/adresses/:idAdresse', personneController.getAdresseByIdPersonne);
router.post('/', personneController.add);
router.put('/:id', personneController.edit);
router.delete('/:id', personneController.delete);

module.exports = router;
```

# ExpressJS

## Question 1

Dans `personne-adresse.dao.js`, créez une fonction `getOneAdresseOfPersonne(idPersonne, idAdresse)` qui retourne l'adresse d'une personne dont les identifiants sont passés en paramètre.

© Achref EL

# ExpressJS

## Question 1

Dans `personne-adresse.dao.js`, créez une fonction `getOneAdresseOfPersonne(idPersonne, idAdresse)` qui retourne l'adresse d'une personne dont les identifiants sont passés en paramètre.

## Question 2

Dans `personne.controller.js`, définissez les fonctions `getAdressesByIdPersonne` et `getAdresseByIdPersonne` qui retournent l'adresse ou la liste d'adresses d'une personne.

# ExpressJS

**Fonction** `getOneAdresseOfPersonne` **de** `personne-adresse.dao.js`

```
exports.getOneAdresseOfPersonne = (idPersonne, idAdresse) => {  
  return new Promise((resolve, reject) => {  
    const req = connection.query("SELECT a.id, rue, codePostal,  
      ville FROM adresse a JOIN personne_adresse pa ON pa.  
      id_adresse = a.id WHERE num_personne = ? AND id_adresse = ?  
      ", [idPersonne, idAdresse], (err, result) => {  
        console.log(req.sql)  
        err ? reject(err) : resolve(result);  
      });  
  });  
}
```

# ExpressJS

Fonctions `getAdressesByIdPersonne` et `getAdresseByIdPersonne` de `personne.controller.js`

```
exports.getAdressesByIdPersonne = (req, res, next) => {
  const id = parseInt(req.params.idPersonne);
  personneAdresseDao.getAllAdressesOfPersonne(id)
    .then(adresses => res.status(200).json(adresses))
    .catch(err => {
      return res.status(404).json({
        error: `Aucune correspondance pour la personne ${idPersonne}`
      });
    });
}

exports.getAdresseByIdPersonne = (req, res, next) => {
  const idPersonne = parseInt(req.params.idPersonne);
  const idAdresse = parseInt(req.params.idAdresse);
  personneAdresseDao.getOneAdresseOfPersonne(idPersonne, idAdresse)
    .then(a => res.status(200).json(a[0]))
    .catch(err => {
      return res.status(404).json({
        error: `Aucune correspondance entre personne ${idPersonne} et adresse ${idAdresse}`
      });
    });
}
```

# ExpressJS

Pour la suite, intégrons `adresses` dans `models/personne.js`

```
exports.Personne = class {  
  constructor(num, nom, prenom, adresses) {  
    this.num = num;  
    this.nom = nom;  
    this.prenom = prenom;  
    this.adresses = adresses;  
  }  
}
```

## Question 1

Faites le nécessaire pour que la fonction `add` permet d'ajouter une personne avec ses adresses (si une adresse a un identifiant, deux tuples seront ajoutés : un dans la table `adresse` et un deuxième dans `personne_adresse`, sinon un seul tuple dans `personne_adresse`).

## Corps de la requête

```
{
  "nom": "Benamar",
  "prenom": "Karim",
  "adresses": [
    {
      "id": 1,
      "rue": "Paradis",
      "ville": "Marseille",
      "codePostal": "13006"
    },
    {
      "rue": "Gervaise",
      "ville": "Valenciennes",
      "codePostal": "59000"
    }
  ]
}
```

## Réponse attendue

```
{
  "num": 5,
  "nom": "Benamar",
  "prenom": "Karim",
  "adresses": [
    {
      "id": 1,
      "rue": "Paradis",
      "ville": "Marseille",
      "codePostal": "13006"
    },
    {
      "rue": "Gervaise",
      "ville": "Valenciennes",
      "codePostal": "59000",
      "id": 5
    }
  ]
}
```

Une personne a été ajoutée avec une adresse existante (à Marseille) et une nouvelle (à Valenciennes).

# ExpressJS

Commençons par définir la fonction `add` dans `dao/personne-controller.dao.js`

```
exports.add = (idPersonne, idAdresse) => {  
  return new Promise((resolve, reject) => {  
    const req = connection.query("INSERT INTO personne_adresse SET  
      num_personne = ?, id_adresse = ?", [idPersonne, idAdresse],  
      (err, result) => {  
        console.log(req.sql)  
        err ? reject(err) : resolve(result);  
      });  
  });  
};
```

Définissons maintenant la fonction `add` dans `controller/personne-controller.controller.js`

```
exports.add = (req, res, next) => {
  const p = new personne.Personne(
    req.body.num,
    req.body.nom,
    req.body.prenom,
    req.body.adresses
  );
  personneDao.add(p).catch(err => {
    return res.status(500).json({ error: `problème d'insertion de personne: ${err}` });
  })
  .then(async (result) => {
    p.num = result.insertId;
    for (let adresse of p.adresses) {
      if (!adresse.id) {
        let res = await adresseDao.add(adresse).catch(err => {
          return res.status(500).json({
            error: `problème d'insertion dans adresse: ${err}`
          });
        });
        adresse.id = res.insertId;
      }
      if (adresse.id) {
        await personneAdresseDao.add(p.num, adresse.id).catch(err => {
          return res.status(500).json({
            error: `problème d'insertion dans personne_adresse : ${err}`
          });
        });
      }
    }
    return res.status(201).json(p);
  });
}
```

# ExpressJS

## Question 2

Faites la même chose pour la fonction `delete`. En supprimant une personne, toutes ses occurrences dans `personne_adresse` seront également supprimées.

# ExpressJS

Commençons par définir la fonction `delete` dans `dao/personne-controller.dao.js`

```
exports.deleteByIdPersonne = (idPersonne) => {  
  return new Promise((resolve, reject) => {  
    const req = connection.query("DELETE FROM personne_adresse  
      WHERE num_personne = ?", idPersonne, (err, result) => {  
      console.log(req.sql)  
      err ? reject(err) : resolve(result);  
    });  
  });  
};
```

# ExpressJS

Définissons maintenant la fonction `delete` dans `controller/personne-controller.controller.js`

```
exports.delete = async (req, res, next) => {
  const id = parseInt(req.params.id);
  personneAdresseDao.deleteByIdPersonne(id)
    .catch(err => {
      return res.status(500).json({
        error: `problème de suppression d'adresse : ${err}`
      });
    })
    .then((result) => {
      personneDao.delete(id)
        .then(result => {
          return res.status(200).json({
            message: `personne avec l'identifiant ${id} supprimée avec succès`
          });
        })
        .catch(err => {
          if (!err) {
            return res.status(404).json({
              error: `Aucune personne avec l'identifiant ${id}`
            });
          }
          return res.status(500).json({
            error: `problème de suppression de personne : ${err}`
          });
        })
    });
});
}
```

## Question 3

Faites la même chose pour la fonction `edit`.

## Corps de la requête

```
{
  "num": 1,
  "nom": "Dupont",
  "prenom": "Sophie",
  "adresses": [
    {
      "id": 1,
      "rue": "La viste",
      "ville": "Marseille",
      "codePostal": "13015"
    },
    {
      "rue": "Défense",
      "ville": "Paris",
      "codePostal": "75000"
    }
  ]
}
```

## Réponse attendue

```
{
  "num": 1,
  "nom": "Dupont",
  "prenom": "Sophie",
  "adresses": [
    {
      "id": 1,
      "rue": "La viste",
      "ville": "Marseille",
      "codePostal": "13015"
    },
    {
      "rue": "Défense",
      "ville": "Paris",
      "codePostal": "75000",
      "id": 5
    }
  ]
}
```

Le nom a été modifié, et l'adresse de Marseille aussi. L'adresse Paris Victoire n'apparaît plus et une nouvelle vient d'être créée (Paris Défense).

Définissons la fonction `edit` dans `controller/personne-controller.controller.js`

```
exports.edit = async (req, res, next) => {
  const id = parseInt(req.params.id);
  const p = new personne.Personne(req.body.num, req.body.nom, req.body.prenom, req.body.adresses);
  await personneDao.edit(id, p).then(async (result) => {
    await personneAdresseDao.deleteByIdPersonne(id).catch(err => {
      return res.status(500).json({
        error: `problème de suppression d'adresse : ${err}`
      });
    });
    for (let adresse of p.adresses) {
      if (!adresse.id) {
        let res = await adresseDao.add(adresse).catch(err => {
          return res.status(500).json({
            error: `problème d'insertion dans adresse: ${err}`
          });
        });
        adresse.id = res.insertId;
      }
      await personneAdresseDao.add(p.num, adresse.id).catch(err => {
        return res.status(500).json({
          error: `problème d'insertion dans personne_adresse : ${err}`
        });
      });
    }
    return res.status(202).json(p);
  })
  .catch(err => {
    if (!err)
      return res.status(404).json({ error: `Aucune personne avec l'id ${id}` });
    return res.status(500).json({ error: `problème de modification : ${err}` });
  });
}
```