

Chapitre 3

Express JS

Le framework Express

Le framework Express JS

- Framework minimaliste (2010) pour la création d'API robustes
- npm install express
- npm init -y
- création d'un fichier app.js / index.js

```
...          Uploaded using RayThis Extension

const express = require('express');
const app = express();
const port = 3000;

app.use(express.json());
app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

Une première route

- <http://localhost:3000/helloworld>

... Uploaded using RayThis Extension

```
app.get('/hello', (req, res) => {
  res.send('Hello World!');
})
```

Récupérer les url-params

- <http://localhost:3000/url-params/test>

● ● ● Uploaded using RayThis Extension

```
app.get('/url-params/:name', (req, res) => {
  const name = req.params.name;
  res.send(`Hello ${name}!`);
})
```

Récupérer les url-params

- `http://localhost:3000/path?name=test`



Uploaded using RayThis Extension

```
app.get('/path', (req, res) => {
  const name = req.query.name;
  res.send(`Hello ${name}!`);
}
```

Code de retour

- <http://localhost:3000/error>



Uploaded using RayThis Extension

```
app.get('/error', (req, res) => {
  res.status(400).send('None shall pass!');
}
```

Les types de réponses

```
● ● ●          Uploaded using RayThis Extension

app.get('/html', (req, res) => {
    res.send('<h1>Hello World!</h1>');
}

app.get('/json', (req, res) => {
    res.json({ message: 'Hello World!' });
}

app.get('/file', (req, res) => {
    res.send('path/to/your/file.txt');
})
```

Structurer en packages

app.js

```
const express = require("express")
const app = express()
const productRouter = require("./router/productRouter")

app.use("/product", productRouter)
app.listen(3000, () => {
    console.log("Server running on port 3000")
})
```

productRouter.js

```
const express = require("express")
const router = express.Router()
const controller = require("../controller/productController")

router.get("", controller.getAll)
router.get("/:id", controller.findById)
router.post("", controller.store)
router.put("/:id", controller.update)
router.delete("/:id", controller.destroy)

module.exports = router
```

productController.js

```
const controller = {
    getAll: (req, res) => {
        res.send("Get all resources")
    },
    findById: (req, res) => {
        res.send("Get resource by id")
    },
    store: (req, res) => {
        res.send("Create new resource")
    },
    update: (req, res) => {
        res.send("Update resource")
    },
    destroy: (req, res) => {
        res.send("Delete resource")
    }
}

module.exports = controller
```

Uploaded using RayThis Extension

```
// app.js
const express = require("express")
const app = express()
app.listen(3000, () => {
    console.log("Server running on port 3000")
})
```

```
// router.js
const express = require("express")
const router = express.Router()
```

```
router.get("", controller.getAll)
router.get("/:id", controller.findById)
router.post("", controller.store)
router.put("/:id", controller.update)
router.delete("/:id", controller.destroy)
```

```
// controller.js
const controller = {
    getAll: (req, res) => {
        res.send("Get all resources")
    },
    findById: (req, res) => {
        res.send("Get resource by id")
    },
    store: (req, res) => {
        res.send("Create new resource")
    },
    update: (req, res) => {
        res.send("Update resource")
    },
    destroy: (req, res) => {
        res.send("Delete resource")
    }
}
```

Notion de middleware

- Gérer les requêtes de manière modulaire.
- Ajout de fonctionnalités (gestion d'erreurs par exemple)

```
● ● ●          Uploaded using RayThis Extension

const productRequest = (req, res, next) => {
    if(req.body.name != undefined && req.body.price != undefined) {
        return next();
    } else {
        res.status(403).send("Price or name is missing");
    }
}
router.post("/store", productRequest, controller.store);
```

Exercices

Chapitre 4

Gestion des données

Partie 1

CRUD en SQL

Configuration avec Sequelize

- Object-Relational Mapping (ORM)
- Facilite les interactions avec les BDD relationnelles
- On travaille avec des objets JS au lieu d'effectuer des requêtes SQL

```
...          Uploaded using RayThis Extension

const Sequelize = require('sequelize');

const db = new Sequelize(
  "database_name",
  "username",
  "password",
  {
    dialect: "mysql",
    host: "localhost"
  }
);

module.exports = db;
```

Définir un modèle

```
...  
Uploaded using RayThis Extension  
  
const Sequelize = require('sequelize');  
const db = require('../config/db');  
  
const productSchema = db.define('product', {  
    id: {  
        type: Sequelize.INTEGER,  
        primaryKey: true,  
        autoIncrement: true  
    },  
    name: {  
        type: Sequelize.STRING,  
        allowNull: false  
    },  
    price: {  
        type: Sequelize.FLOAT,  
        allowNull: false  
    }  
});  
  
module.exports = productSchema;
```

Faire des requêtes

Uploaded using RayThis Extension

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

router.get('/:id', productController.getProductById);
router.get('/', productController.getAllProducts);
router.post('/', productController.createProduct);
router.put('/:id', productController.updateProduct);
router.delete('/:id', productController.deleteProduct);

module.exports = router;
```

Uploaded using RayThis Extension

```
const { Product } = require('../models'); // Assure-toi d'avoir importé ton modèle Sequelize correctement

const getProductById = async (req, res) => {
    try {
        const product = await Product.findOne({ where: { id: req.params.id } });
        if (!product) { return res.status(404).json({ message: 'No product found with that id' }); }
        res.json(product);
    } catch (err) { res.status(500).json({ message: 'Error retrieving product', error: err }); }
};

const getAllProducts = async (req, res) => {
    try {
        const products = await Product.findAll();
        res.json(products);
    } catch (err) { res.status(500).json({ message: 'Error retrieving products', error: err }); }
};

const createProduct = async (req, res) => {
    try {
        const product = await Product.create(req.body);
        res.json(product);
    } catch (err) { res.status(500).json({ message: 'Error creating product', error: err }); }
};

const updateProduct = async (req, res) => {
    try {
        const [updatedRows] = await Product.update(req.body, { where: { id: req.params.id } });
        if (updatedRows === 0) { return res.status(404).json({ message: 'No product found with that id to update' }); }
        const updatedProduct = await Product.findOne({ where: { id: req.params.id } });
        res.json(updatedProduct);
    } catch (err) { res.status(500).json({ message: 'Error updating product', error: err }); }
};

const deleteProduct = async (req, res) => {
    try {
        const deletedRows = await Product.destroy({ where: { id: req.params.id } });
        if (deletedRows === 0) { return res.status(404).json({ message: 'No product found with that id to delete' }); }
        res.json({ message: 'Product deleted successfully' });
    } catch (err) { res.status(500).json({ message: 'Error deleting product', error: err }); }
};

module.exports = { getProductById, getAllProducts, createProduct, updateProduct, deleteProduct };
```

Exercices



TP 2 : Validation des acquis