Dev Web

côté serveur

S3 - R3.01 - 2022/2023

Application Notes (Notes App) avec swagger et téléchargement de fichiers

- REST API
- SWAGGER DOC
- EXPRESS-VALIDATOR
- Téléchargement de fichiers

Activité 1 : Importer des modules swagger et initialiser swagger

```
npm install swagger-jsdoc
npm install swagger-ui-express
```

```
// server.js
// ...
const swaggerJsdoc = require("swagger-jsdoc");
const swaggerUi = require("swagger-ui-express");
// ...
```

Activité 1 : Importer des modules swagger et initialiser swagger

```
// server.js
// ...
/** Swagger Initialization - START */
const swaggerOption = {
    swaggerDefinition: (swaggerJsdoc.Options = {
        info: {
            title: "my-notes app",
            description: "API documentation",
            contact: {
                name: "JOSEPH AZAR",
            },
            servers: ["http://localhost:3000/"],
        },
    }),
    apis: ["server.js", "./routes/*.js"],
};

const swaggerDocs = swaggerJsdoc(swaggerOption);
app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerDocs));
/** Swagger Initialization - END */
// ...
```

Activité 2 : Gestion des erreurs en utilisant app.all("*")

```
// server.js
// ...
const AppError = require("./utils/appError");
// ...
app.all('*', (req, res, next) => {
    throw new AppError(`Requested URL ${req.path} not found!`, 404);
});
// ...
```

Activité 3 : Créez la route pour obtenir tous les utilisateurs ou un certain utilisateur et ajoutez-y une documentation

```
// users.router.js
// ...
router.get("/:userId?", usersController.getUsers);
/**
 * @swagger
 * /users/{userId}:
 *   get:
 *      description: Used to get all users
 *      tags:
 *          - users
 *      parameters:
 *          - in: path
 *            name: userId
 *            type: integer
 *            required: false
 *            description: Numeric ID of the user to get (Optional)
 *      responses:
 *          '200':
 *             description: Resource added successfully
 *          '500':
 *             description: Internal server error
 *          '400':
 *             description: Bad request
 */

// ...
```

```
// users.controller.js
// ...
exports.getUsers = (req, res) => {
    console.log(req.params.userId)
    let userId = req.params.userId;
    userId = typeof userId == 'undefined' ? "" : userId;
    if (validator.isEmpty(userId) || userId == "{userId}" || userId == "undefined"){
        usersService.getAllUsers((error, results) => {
            if (error) {
                console.log(error);
                return res.status(400).send({ success: 0, data: "Bad request" });
            }
            return res.status(200).send({
                success: 1,
                data: results,
            });
        });
    }else if (validator.isInt(userId)){
        usersService.getUserById(req.params.userId,(error, results) => {
            if (error) {
                console.log(error);
                return res.status(400).send({ success: 0, data: "Bad request" });
            }
            return res.status(200).send({
                success: 1,
                data: results,
            });
        });
    }else{
        return res.status(400).send({ success: 0, data: "Bad request" });
    }
};
// ...
```

Activité 4 : Créez la route pour mettre à jour un utilisateur avec un identifiant dans les paramètres et le prénom/nom dans le corps et ajoutez une documentation swagger

```
// users.router.js
// ...
router.put("/:id/update",usersController.updateUser)
/**
 * @swagger
 * /users/{userId}/update:
 *   put:
 *      description: Used to update user
 *      tags:
 *          - users
 *      parameters:
 *          - in: path
 *            name: userId
 *            type: integer
 *            description: User id
 *            required: true
 *          - in: body
 *            name: User
 *            description: User data with new values of properties
 *            schema:
 *              type: object
 *              required:
 *                  - firstName
 *                  - lastName
 *              properties:
 *                  firstName:
 *                      type: string
 *                      minLength: 1
 *                      maxLength: 45
 *                      example: James
 *                  lastName:
 *                      type: string
 *                      minLength: 1
 *                      maxLength: 45
 *                      example: Bond
 *      responses:
 *          '200':
 *              description: Resource updated successfully
 *          '500':
 *              description: Internal server error
 *          '400':
 *              description: Bad request
 */
// ...
```

Activité 4 : Créez la route pour mettre à jour un utilisateur avec un identifiant dans les paramètres et le prénom/nom dans le corps et ajoutez une documentation swagger

```
// users.controller.js
// ...
exports.updateUser = (req, res) =>{
    let id = req.params.id;
    let data = {
        firstname: req.body.firstName,
        lastname: req.body.lastName
    }
    usersService.updateUser(id,data,(error, results) => {
        if (error) {
            console.log(error);
            return res.status(400).send({ success: 0, data: error });
        }
        return res.status(200).send({
            success: 1,
            data: results,
        });
    });
}
// ...
```

Activité 5 : Servir des fichiers statiques dans Express

```
// server.js
// ...
app.use(express.static(path.join(__dirname, 'public')))
app.use('/upload', express.static(path.join(__dirname,'upload/images')));
// ...
```

Activité 6 : Définir la route notes "/" pour envoyer un fichier html "notes.html"

```
// notes.router.js
// ...
router.get("/",notesController.showHTML);
// ...
```

```
// notes.controller.js
// ...
exports.showHTML = (req,res)=>{
    var options = {
        root: "public",
        dotfiles: 'deny',
        headers: {
            'x-timestamp': Date.now(),
            'x-sent': true
        }
    }
    res.sendFile("notes.html", options, function (err) {
        if (err) {
            console.log(err);
        } else {
            console.log('Sent: notes.html')
        }
    })
}
// ...
```

Activité 7 : Créez une route /add pour ajouter une note avec une image. Validez l'image et les entrées du formulaire

```
// imagesProcessor.js
// ...
const resizeImages = async (req, res, next) => {
    if (!req.files) return res.status(400)
      .send(`You must select at least 1 image.`);
  // Ajouter cette ligne
  if (req.files.length <= 0) return res.status(400)
    .send(`You must select at least 1 image.`);

  req.body.images = [];
    await Promise.all(
        req.files.map(async file => {
            const filename = file.originalname.replace(/\..+$/, "");
            const newFilename = `iutbm-s3-${filename}-${Date.now()}.jpeg`;

            await sharp(file.buffer)
                .resize(128, 128)
                .toFormat("jpeg")
                .jpeg({ quality: 90 })
                .toFile(`upload/images/${newFilename}`);

            req.body.images.push(newFilename);
        })
    );

    next();
};
// ...
```

Activité 7 : Créez une route /add pour ajouter une note avec une image. Validez l'image et les entrées du formulaire

```
// users.controller.js
// ...
exports.userExists = (userId) => {
    return usersService.userExists(userId);
}
// ...
```

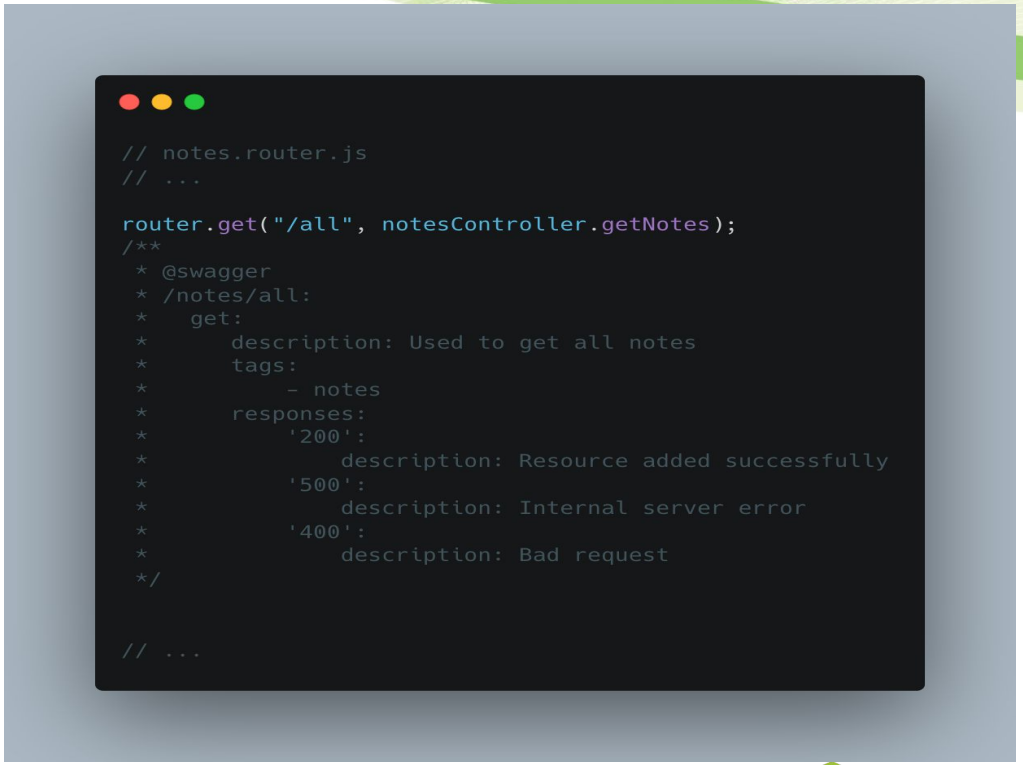
```
// notes.router.js
// ...
const imageProcessor = require("../middlewares/imagesProcessor");
const {check,  validationResult } = require('express-validator');
const {userExists} = require("../controllers/users.controller");
// ...
router.post("/add",
    imageProcessor.uploadImages,
    [
        check('user', 'User is required').not().isEmpty().bail().custom(userExists),
        check('title', 'Title is required').not().isEmpty(),
        check('body', 'Body is required').not().isEmpty(),
    ], (req, res, next)=>{
    let errors = validationResult(req);
    if (!errors.isEmpty()) {
        console.log(errors.mapped());
        return res.status(400).send({ success: 0, data: errors.mapped()});
    }else{
        next();
    }},
    imageProcessor.resizeImages,
    notesController.addNote);
// ...
```

Activité 7 : Créez une route /add pour ajouter une note avec une image. Validez l'image et les entrées du formulaire

```
// notes.controller.js
// ...

exports.addNote = (req,res) => {
    const images = req.body.images
        .map(image => "" + image + "")
        .join(";");
    let data = {
        user: req.body.user,
        title: req.body.title,
        body: req.body.body,
        images: images.split(";")
    }

    notesService.saveNote(data,(error,results) => {
        if (error) {
            console.log(error);
            return res.status(400).send({ success: 0, data: error });
        }
        res.redirect("/notes");
    });
}

// ...
```

```
// notes.service.js
// ...
const getNextId = (notes) => {
    return (Math.max.apply(Math, notes.map( note => {
        return note.id;
    })) + 1);
}       console.log(error);
            return res.status(400).send({ success: 0, data: error });
        }
        res.redirect("/notes");
    });
}

// ...
```

Activité 8 : Ajoutez la route "/notes/all" pour obtenir toutes les notes et ajouter la documentation swagger



```
// notes.router.js
// ...

router.get("/all", notesController.getNotes);
/**
 * @swagger
 * /notes/all:
 *   get:
 *       description: Used to get all notes
 *       tags:
 *           - notes
 *       responses:
 *           '200':
 *               description: Resource added successfully
 *           '500':
 *               description: Internal server error
 *           '400':
 *               description: Bad request
 */


// ...
```

Activité 9 : Implémenter la route "/notes/delete" pour supprimer une note en utilisant son identifiant de la requête (query)

```
// notes.router.js
// ...

router.delete("/delete",notesController.deleteNote);
/**
 * @swagger
 * /notes/delete:
 *   delete:
 *      description: Used to delete a note
 *      tags:
 *          - notes
 *      parameters:
 *          - in: query
 *            name: noteId
 *            description: Note ID
 *            required: true
 *            type: integer
 *      responses:
 *          '200':
 *              description: Resource updated successfully
 *          '500':
 *              description: Internal server error
 *          '400':
 *              description: Bad request
 */


// ...
```

Activité 9 : Implémenter la route "/notes/delete" pour supprimer une note en utilisant son identifiant de la requête (query)

```
// notes.service.js
// ...

const removeNote = (id, callback) => {
    const notes = loadNotes();
    const notesToKeep = notes.filter((note) => note.id != id)

    if (notes.length > notesToKeep.length) {
        notes.forEach(note=>{
            if(note.id == id){
                note.images.forEach(image=>{
                    try{fs.unlinkSync(`upload/images/${image}`);}catch (e){};
                })
            }
        })
        const dataJSON = JSON.stringify(notesToKeep)
        fs.writeFileSync('notes.json', dataJSON)
        return callback(null, "Note deleted successfully!")
    } else {
        return callback("Operation failed!")
    }
}

// ...
```