**BUILDING A REST API FROM SCRATCH**

const express = require('express');

// Setting up an instance in express

const app = express();

//Adding middleware

app.use(express.json());

const PORT = 9090;

// Performing GET, POST, PUT & DELETE

const books = [];

app.listen(PORT, () => {

    console.log(`Application running on port ${PORT}`)

})

// GET request

app.get('/recruitment/books', (req, res, next) => {

    return res.status(200).json({

        status: 'success',

        code: 200,

        data: books

    })

})

// POST request

app.post('/recruitment/books', (req, res, next) => {

    const { title } = req.body

     // Check if book exists

     const bookExist = books.find((element) => element.title === title)

     if(bookExist) {

         return res.status(409).json({

             status: 'error',

             code: 409,

             message: 'Book already exists',

             data: null

         })

     }

     books.push(req.body)

     return res.status(201).json({

         status: 'success',

         code: 201,

         message: 'Book inserted successfully',

         data: books

     })

})

//UPDATE request

app.put('/recruitment/books/:id', (req, res, next) => {

    const { id } = req.params

    const { title, author } = req.body

    const bookIndex = books.findIndex((element) => element.id === id)

    if(bookIndex >= 0) {

        books[bookIndex] = {

            ...books[bookIndex],

            title,

            author

        }

        return res.status(200).json({

            status: 'success',

            message: 'Book updated successfully',

            code: 200,

            data: books

        })

    }

    return res.status(400).json({

        status: 'error',

        message: 'Book not found',

        code: 400,

        data: null

    })

})

// DELETE request

app.delete('/recruitment/books/:id', (req, res, next) => {

    const { id } = req.params

    const bookIndex = books.findIndex((element) => element.id === id)

    if (bookIndex >= 0) {

        books.splice(bookIndex, 1);

        return res.status(200).json({

            status: 'success',

            message: 'book deleted successfully',

            code: 200,

            data: books

        })

    }

    return res.status(400).json({

        status: 'error',

        message: 'book not found',

        code: 400,

        data: null

    })

})

Notes:

1. Jsonwebtoken basically encodes the information. It requires an authentication of an user, thus, you don’t have access. Whereas, authorization asks the question that do you have rights to perform a task or access an info.
2. Dotenv is a package manager for managing your environments, thus, your dotenv files
3. Putting your authorization or verifyAccount into a ‘Try-catch’ means it needs to have a default error handler.
4. The error handler should always be the last route, so that it executes after everything completing everything else.
5. Research into the use of ‘logger’ within our backend operations.
6. Ideally, your controllers should not be where you’re doing you’re business logic.
7. Whenever there’s an error we ‘throw’ instead of return
8. Migrations basically allows you to manage your tables.
9. “db-migrate create add-user-id-column-to-books –sql-file”: this command is to add a new column to an already created table. UP: ALTER TABLE books ADD COLUMN user\_id INT; DOWN: ALTER TABLE books DROP COLUMN user\_id; Use “db-migrate up” to add up the updates.