**קוד שיעור 16 - Node**

**Index.js**

import express from "express";

import { config } from "dotenv";

import { connectToDB } from "./config/dbConfig.js"

import courseRouter from "./routes/course.js";

import userRouter from "./routes/user.js";

import { errorHandling } from "./middlwares/erroHandlingMiddlware.js";

import cors from "cors";

config();//מאפשר לכתוב משתני סביבה בקובץ

connectToDB();

//.env

const app = express();

app.use(express.json());

app.use(cors({origin:"http://127.0.0.1:5400" ,methods:"\*"}))//כך נאפשר שקליינט יוכל

//לגשת לשרת הזה שאנחנו בונים כאן

//אבל רק מקליינט שמורץ על כתובת  הנל

app.use("/api/courses", courseRouter)

app.use("/api/users", userRouter)

app.use(errorHandling)

let port = process.env.PORT || 3500;

app.listen(port, () => {

    console.log(`app is listening on port ${port}`)

})

**Package.json**

{

  "name": "lesson14",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "dependencies": {

    "bcrypt": "^5.1.1",

    "cors": "2.8.5",

    "dotenv": "16.3.1",

    "express": "4.18.2",

    "joi": "17.11.0",

    "jsonwebtoken": "^9.0.2",

    "mongoose": "8.0.3"

  },

  "devDependencies": {

    "nodemon": "3.0.2"

  },

  "type": "module",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "start": "node index.js",

    "dev": "nodemon index.js"

  },

  "author": "",

  "license": "ISC"

}

**Config => dbConfig.js**

import mongoose from "mongoose";

export const connectToDB = async () => {

    // mongoose.connect(process.env.DB\_CONNECTION).then((con) => {

    //     console.log("mongoDB connected successfully!!!", con.connection.host);

    // }).catch((err) => {\\

    //     console.log("cannot connect mongoDB");

    //     console.log(err)

    //     process.exit(1);

    // })

    try {

        let con = await mongoose.connect(process.env.DB\_CONNECTION||"mongodb+srv://learn2024driveboker:dI0yYh3QPtSc8nG3@bokershishi.huhxhgw.mongodb.net/?retryWrites=true&w=majority");

        console.log("mongoDB connected successfully!!!", con.connection.host);

    } catch (err) {

        console.log("cannot connect mongoDB");

        console.log(err)

        process.exit(1);

    }

}

**Config => generateToken.js**

import jwt from "jsonwebtoken";

export const generateToken = (user) => {

    let jwtSecretKey = process.env.JWT\_STRING;

    let data = {

        userName: user.userName,

        \_id: user.\_id,

        role: user.role

    }

    const token = jwt.sign(data, jwtSecretKey, {

        expiresIn: '30s',

    });

    return token;

}

**Controllers => course.js**

import mongoose from "mongoose";

import { Course } from "../models/course.js";

const getAllCourses = async (req, res) => {

    //params -חובה

    //query params

    // http://localhost:4200/cake/1

    // http://localhost:4200/cake?price=50&category=70

    // http://localhost:4200/cake?search=ava

    let { search } = req.query;

    let perPage = req.query.perPage || 40;

    let page = req.query.page || 1;

    //let ex = /ava{1,6}$/

    let ex1 = new RegExp(`${search}`)//המחרוזת תיהיה חייבת להסתיים ב

    try {

        let filter = {};

        if (search);

        filter.name = ex1;//{ name: ex1 }

        let allCourses = await Course.find(filter)

        .skip(page\*(perPage-1))//לדלג על כמות תואצות מסויימת

        .limit(perPage);//שולך כמות מוגבלת של נתונים

        res.json(allCourses);

    }

    catch (err) {

        res.status(400).json({ type: "error", message: err.message })

    }

}

const getCourseById = async (req, res) => {

    try {

        let { id } = req.params;

        if (!mongoose.isValidObjectId(id))

            return res.status(400).json({ type: "id error", message: "id is not valid" })

        const course = await Course.findById(id);

        if (!course)

            return res.status(404).json({ type: "id not found", message: "didnt find " })

        res.json(course);

    }

    catch (err) {

        res.status(400).json({ type: "error", message: err.message })

    }

}

const addCourse = async (req, res) => {

    try {

        let { name, price, tags, numLessons, speaker } = req.body;

        if (!name || !speaker || !price)

            return res.status(404).json({ type: "missing paramters", message: "name or price or speaker" })

        let sameCourse = await Course.findOne({ name, price });

        if (sameCourse)

            return res.status(409).json({ type: "same course", message: "same details" })

        // let newCourse= new Course({name,numLessons,price,tags,speaker});

        // await newCourse.save();

        let newCourse = await Course.create({ name, numLessons, price, tags, speaker });

        res.json(newCourse);

    }

    catch (err) {

        res.status(400).json({ type: "error", message: err.message })

    }

}

export { addCourse, getAllCourses, getCourseById };

**controllers => user.js**

import { generateToken } from "../config/generateToken.js";

import { User, userValidator, userValidatorForLogin } from "../models/user.js";

import { hash, compare } from "bcrypt";

export const addUser = async (req, res) => {

    // let { userName, password, tz, email } = req.body;

    // if (!userName || !password || !tz || !email)

    //     return res.status(404).json({ type: "misiing parameters", message: "missing parameters password /username/tz/email" })

    // if (!/[0-9]{1,2}/.test(password))

    //     return res.status(400).json({ type: "invalid password", message: "try again" })

    let validate = userValidator(req.body);

    if (validate.error)

        return res.status(400).json({ type: "not valid body ", message: validate.error.details[0].message });

    let { userName, password, tz, email } = req.body;

    try {

        let sameUser = await User.findOne({ $or: [{ userName: userName }, { tz: tz }] })

        if (sameUser)

            return res.status(409).json({ type: "same user", message: "user with same credentials already exists" })

        let hashedPassword = await hash(password, 15);

        let newUser = new User({ userName, password: hashedPassword, email, tz });

        await newUser.save();

        let token = generateToken(newUser);

        return res.json({ token })

    } catch (err) {

        res.status(400).json({ type: "error", message: err.message })

    }

}

export const login = async (req, res) => {

    // let { password, userName } = req.body;

    // if (!password || !userName)

    //     return res.status(404).json({ type: "misiing parameters", message: "missing parameters password /username" })

    let validate = userValidatorForLogin(req.body);

    if (validate.error)

        return res.status(400).json({ type: "not valid body ", message: validate.error.details[0].message });

    try {

        let user = await User.findOne({ userName: req.body.userName })

        if (!user || !await compare(req.body.password, user.password))

            res.status(404).json({ type: "no such user", message: "please sign up" })

        let token = generateToken(user);

        return res.json({ token })

        // user.password = "\*\*\*\*";

        // return user;

    }

    catch (err) {

        res.status(400).json({ type: "error", message: err.message })

    }

}

export const getAllUsers = async (req, res) => {

    try {

        const allUsers = await User.find({}, "-password");//אפשר להגביל את השדות שברצוננו לשלוף

        res.json(allUsers)

    }

    catch (err) {

        res.status(400).json({ type: "error", message: err.message })

    }

}

//authorization --הרשאות

//authentication--אימות

**Middlewares => auth.js**

import jwt from "jsonwebtoken";

export const auth = async (req, res, next) => {

    let token = req.headers["xxx-token"];

    if (!token)

        return res.status(401).json({ type: "not authorized", message: "user not authorized" })

    try {

        const decoded = jwt.verify(token, process.env.JWT\_STRING);

        console.log(decoded)

        // if (!decoded)

        //     return res.status(401).json({ type: "not authorized", message: "user not authorized" })

        next();

    } catch (err) {

        return res.status(401).json({ type: "not authorized", message: "user not authorized" })

    }

}

**Moddlewarea => erroHandlingMiddlware.js**

export const errorHandling=(err, req, res, next) => {

    let statusCode = res.statusCode || 500;

    let message = err.message || "מצטערים התרחשה שגיאה בשרת"

    res.statu(statusCode).send(message);

}

**Models => course.js**

import mongoose from "mongoose";

const SpeakerSchema = mongoose.Schema({

    name: String,

    phone: String,

    tz: String,

    startDate: { type: Date, default: Date.now() }

})

const courseSchema = mongoose.Schema({

    name: String,

    price: Number,

    numLessons: Number,

    speaker: SpeakerSchema,

    tags: [String],

    startDate: { type: Date, default: Date.now() }

})

export const Course = mongoose.model("courses", courseSchema);

**models => uses.js**

import Joi from "joi";

import mongoose from "mongoose";

const minimumCourse = mongoose.Schema({

    name: String,

    numLessons: Number,

    speakerFullName: String,

    tags: [String]

})

let userSchema = mongoose.Schema({

    tz: String,

    userName: String,

    password: String,

    email: String,

    courses: [minimumCourse],

    role: { type: String, default: "user" } // admin/user

})

export const User = mongoose.model("users", userSchema)

export const userValidatorForLogin = (\_user) => {

    const schema = Joi.object({

        userName: Joi.string().min(3).max(30).required(),

        password: Joi.string().pattern(new RegExp('^[a-zA-Z0-9]{3,15}$')).required(),

    });

    return schema.validate(\_user);

}

export const userValidator = (\_user) => {

    const schema = Joi.object({

        userName: Joi.string().min(3).max(30).required(),

        password: Joi.string().pattern(new RegExp('^[a-zA-Z0-9]{3,15}$')).required(),

        tz: Joi.string().min(9).max(9).pattern(/^[0-9]{9}$/).required(),

        email: Joi.string().email().required()

    });

    return schema.validate(\_user);

}

**Routes => course.js**

import { addCourse, getAllCourses, getCourseById } from '../controllers/course.js';

import express from "express";

import { auth } from '../middlwares/auth.js';

const router = express.Router();

router.get('/', getAllCourses);

router.get('/:id', getCourseById);

router.post('/', auth, addCourse);

export default router;

**routes => user.js**

import { addUser, login, getAllUsers } from '../controllers/user.js';

import express from "express";

const router = express.Router();

router.post('/', addUser);

router.post('/login', login);

router.get('/', getAllUsers);

export default router;

**.env**

PORT=5000

DB\_CONNECTION=mongodb+srv://learn2024driveboker:dI0yYh3QPtSc8nG3@bokershishi.huhxhgw.mongodb.net/miclol?retryWrites=true&w=majority

JWT\_STRING=sjdklfjsdklfjklsdjfkl