*Index*

*//requiriendo modulos*

const express = require('express');

const userRoutes = require('./routes/userRoutes');

const adminRoutes = require('./routes/adminRoutes');

*//conexion a base de datos*

require('./database');

const app = express();

*//estas lineas con para recibir datos del body*

app.use(express.json());

app.use(express.urlencoded({

  extended: true

}));

*//routes(rutas)http://localhost:3000/api/users/login*

app.use('/api/users', userRoutes);

app.use('/api/admin', adminRoutes);

*//app.use('/api/articles', articlesRoutes);*

*//app.use('/api/point', pointsRoutes);*

*// iniciar server*

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

    console.log(`servidor andando en: ${PORT}`)

})

User router

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

const usersController = require('../controllers/usersController.js');

*//const auth = require('../middlewares/auth.js');*

router.get('/login', usersController.login);

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

router.get('/register', usersController.register);

router.post('/register', usersController.registerpost);

router.get('/recover', usersController.recover);

router.post('/recover', usersController.recoverpost);

module.exports = router ;

user controllers.js

const mongoose = require('mongoose');

const User = require('../models/users');

const jwt = require('jsonwebtoken');

const bcrypt = require('bcrypt');

const Joi = require('@hapi/joi');

*// area login*

exports.login = async (req, res) => {

    res.json({

        estado: "mostrar vista de formulario login",

    });

};

exports.loginpost = async (req, res) => {

    const schemaSignin = Joi.object({

        username: Joi.string().min(6).max(50).required(),

        password: Joi.string().min(6).max(200).required()

    })

*// validate user verdadero*

    const { error } = schemaSignin.validate(req.body)

*if* (error) {

*return* res.status(400).json(

            {error: error.details[0].message}

        )

    }

*//mongodb enviando a base de datos*

    const user = *await* User.findOne({username: req.body.username});

*if* (!user) {

*return* res.status(400).json({error: 'usuario inexistente'})

    }

*// encript password*

    const validPassword = *await* bcrypt.compare(req.body.password, user.password);

*if* (!validPassword) *return* res.status(400).json({ error: 'contraseña no válida' })

*// create token*

    const token = jwt.sign({first\_name: user.first\_name, \_id: user.\_id}, process.env.TOKEN\_SECRET, {expiresIn : 60

    })

    const user\_login = ({first\_name: user.first\_name, last\_name: user.last\_name, \_id: user.\_id})

    res.header('auth-token', token).json({

        error: null,

        user\_login,

        token

    })

};

exports.register = async (req, res) => {

    res.json({

        estado: "mostrar formulario de registro",

    });

};

*// registro post*

exports.registerpost = async (req, res) => {

    const schemaAddUser = Joi.object({

        first\_name: Joi.string().min(3).max(50).required().label('Nombre'),

        last\_name: Joi.string().min(3).max(50).required(),

        doc\_id: Joi.string().min(5).max(50).required(),

        phone: Joi.number().integer().min(100000).max(99999999).required().label('Teléfono'),

        email: Joi.string().min(6).max(100).required().email(),

        username: Joi.string().min(6).max(20).required(),

        password: Joi.string().min(6).max(1024).required()

    })

*// validate user*

    const { error } = schemaAddUser.validate(req.body)

*if* (error) {

*return* res.status(400).json(

            {error: error.details[0].message}

        )

    }

    const isUserNameExist = *await* User.findOne({ username: req.body.username });

*if* (isUserNameExist) {

*return* res.status(400).json(

            {error: 'Nombre de usuario ya registrado'}

        )

    }

    const isDocIdExist = *await* User.findOne({ doc\_id: req.body.doc\_id });

*if* (isDocIdExist) {

*return* res.status(400).json(

            {error: 'Documento ya registrado'}

        )

    }

    const isEmailExist = *await* User.findOne({ email: req.body.email });

*if* (isEmailExist) {

*return* res.status(400).json(

            {error: 'Email ya registrado'}

        )

    }

    const dataUser = {first\_name, last\_name, doc\_id, phone, email, username, password} = req.body;

    const salt = *await* bcrypt.genSalt(10);

    dataUser.password = *await* bcrypt.hash(req.body.password, salt);

    dataUser.state = 0;

    const user = new User (dataUser);

*try* {

        const saveUser = *await* user.save();

        res.status(200).json({

                user: saveUser,

                message: 'Usuario registrado',

                success: true

            });

    } *catch* (error) {

        res.status(400).json({

                user: saveUser,

                message: 'Error al crear usuario',

                success: false

            });

    }

};

exports.recover = async (req, res) => {

    res.json({

        estado: "mostrar vista de recover",

    });

};

exports.recoverpost = async (req, res) => {

    res.json({

        estado: "enviar datos de recover",

    });

};

Users.js

const mongoose = require('mongoose');

const userSchema = mongoose.Schema({

    first\_name: {

        type: String,

        required: true,

    },

    last\_name: {

        type: String,

        required: true,

    },

    doc\_id: {

        type: String,

    },

    phone: {

        type: Number,

        required: true,

    },

    email: {

        type: String,

        required: true,

    },

    username: {

        type: String,

        required: true,

    },

    password: {

        type: String,

        required: true,

    },

    state: {

        type: Number,

        required: true,

    },

},{

    timestamps: true,

    versionKey: false

});

module.exports = mongoose.model('User', userSchema);