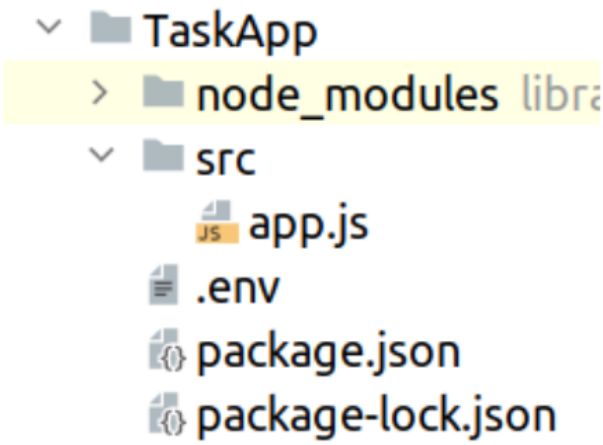
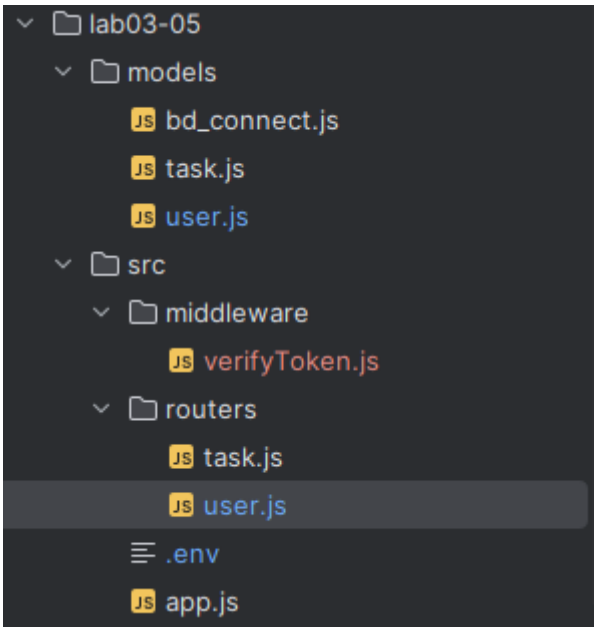
Лабораторна робота №3
Mongoose
Rest API

Завдання 1. Створіть проект TaskApp

- Встановіть модулі *mongoose, validator, express, dotenv*
- Створіть папку *src* та виконавчий файл *src/app.js*





Завдання 2. Підключимось до бази даних та створимо модель User

- Рядок з’єднання винесіть у файл .env в змінну MONGO_URL

```
MONGO_URL=mongodb://localhost:27017/TaskAppNodeaJSDB
PORT=3000
```

```
require("dotenv").config();

const url : string  = process.env.MONGO_URL;

if (!url) {
    console.error("MONGO_URL is not defined in the environment variables");
    process.exit( code: 1);
}

AdamLahovskyi
mongoose.connect(url, options: { useNewUrlParser: true, useUnifiedTopology: true }) Promise<Mongo
    .then(() : any  => console.log("Connected to MongoDB")) Promise<void>
    .catch((error) : void  => {
        console.error("Failed to connect to MongoDB", error);
        process.exit( code: 1);
    });
```

Створимо екземпляр моделі User

```
14  //Створюємо екземпляр моделі
15  const user = new User({name: 'Alex', age: 34});
16
17  //Зберігаємо екземпляр в базу даних та виводимо повідомлення
18  user.save().then(() => {
19      console.log(user);
20  }).catch((error) => {
21      console.log(error);
22  });
```

Протестуйте додаток. Перевірте валідацію полів при введенні значень невірних типів

```
const mongoose = require("mongoose");
const bcrypt = require("bcrypt");

const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        trim: true
    },
    age: {
        type: Number,
        default: 0,
        validate(value) {
            if (value < 0) {
                throw new Error("Age must be a positive number");
            }
        }
    },
    email: {
        type: String,
        required: true,
        lowercase: true,
        unique: true,
        validate: {
            validator: function(value) {
                const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
                return emailRegex.test(value);
            },
            message: props => `${props.value} is not a valid email!`
        }
    },
    password: {
        type: String,
        required: true,
        minlength: 7,
        trim: true,
        validate: {
            validator: function(value) {
                if (value.toLowerCase().includes("password")) {
                    throw new Error("Password cannot contain the word 'pass-
word'");
                }
            },
            message: props => `Password validation failed`
        }
    }
});

userSchema.pre('save', async function(next) {
    const user = this;
    if (user.isModified('password')) {
        user.password = await bcrypt.hash(user.password, 8);
    }
    next();
});

const User = mongoose.model('User', userSchema);
```

```
module.exports = User;
```

Валідація даних

- Для поля *name* моделі *User* встановіть обов'язкову наявність непорожнього значення:

```
name: {
    type: String,
    required: true
},
```

- Перевірте роботу валідатора даних при створенні екземпляра з порожнім полем *name*

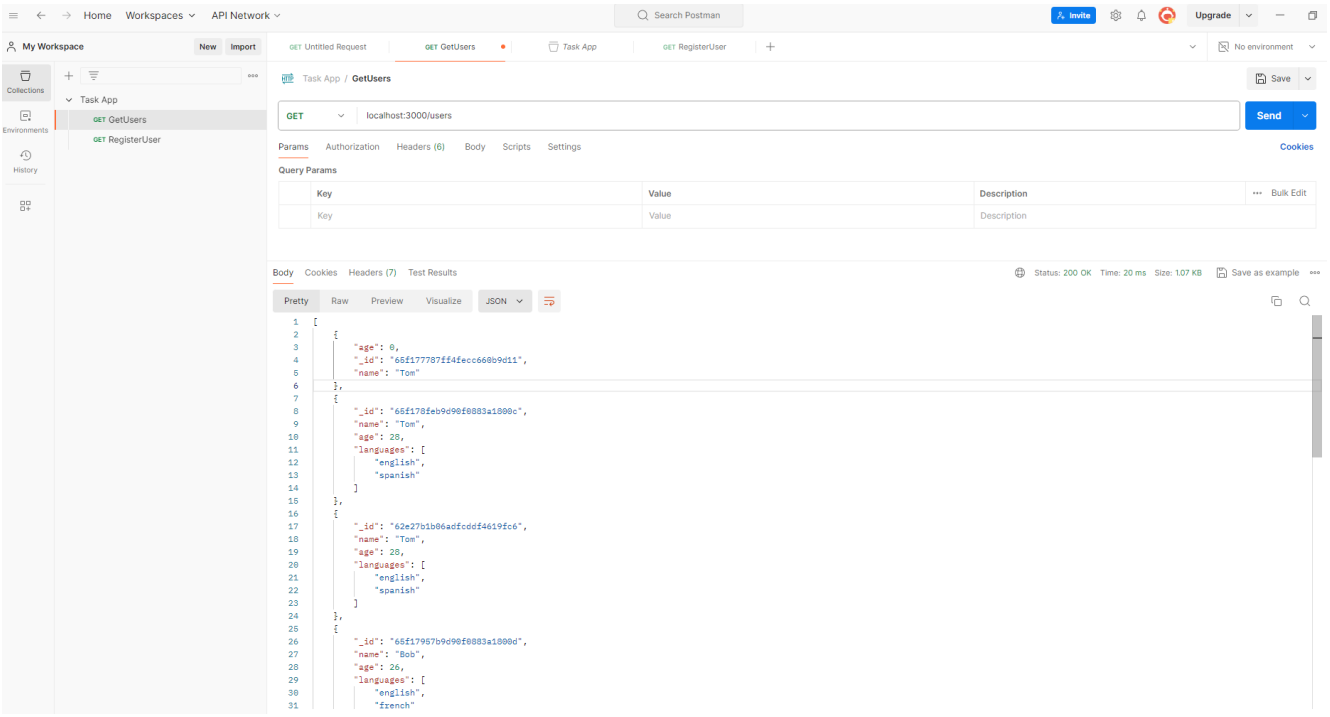
```
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        trim: true
    },
    age: {
        type: Number,
        default: 0,
        validate(value) {
            if (value < 0) {
                throw new Error("Age must be a positive number");
            }
        }
    },
    email: {
        type: String,
        required: true,
        lowercase: true,
        unique: true,
        validate: {
            validator: function(value) {
                const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
                return emailRegex.test(value);
            },
            message: props => `${props.value} is not a valid email!`
        }
    },
    password: {
        type: String,
        required: true,
        minlength: 7,
        trim: true,
        validate: {
            validator: function(value) {
                if (value.toLowerCase().includes("password")) {
                    throw new Error("Password cannot contain the word 'pass-
word'");
                }
```

```
            },
            message: props => `Password validation failed`
        }
    }
});
```

Обробка GET-запиту

- В додатку створіть обробник запиту на отримання всі користувачів з БД:

```
router.get("/", async (req, res) => {
    try {
        const users = await User.find();
        res.json(users);
    } catch (error) {
        res.status(500).send(error.message);
    }
});
```



Здійсніть рефакторинг коду в додатку Task Application з використанням *async/await*

```
const express = require("express");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const User = require("../../models/user");

const router = express.Router();

router.get("/", async (req, res) => {
    try {
        const users = await User.find();
        res.json(users);
    } catch (error) {
        res.status(500).send(error.message);
    }
});

router.get("/:id", async (req, res) => {
    try {
        const user = await User.findById(req.params.id);
        if (!user) {
            return res.status(404).send("User not found");
        }
        res.json(user);
    } catch (error) {
        res.status(500).send(error.message);
    }
});

router.post("/", async (req, res) => {
    try {
        const user = new User(req.body);
        await user.save();
        res.status(201).send(user);
    } catch (error) {
        res.status(400).send(error.message);
    }
});

router.patch("/:id", async (req, res) => {
    try {
        const user = await User.findById(req.params.id);
        if (!user) {
            return res.status(404).send("User not found");
        }
        const fieldsToUpdate = ["firstName", "lastName", "age", "password"];
        fieldsToUpdate.forEach(field => {
            if (req.body[field]) {
                user[field] = req.body[field];
            }
        });
        await user.save();
        res.json(user);
    } catch (error) {
        res.status(500).send(error.message);
    }
});

router.post("/login", async (req, res) => {
```

```
    try {
        const { email, password } = req.body;

        const user = await User.findOne({ email });
        if (!user) {
            return res.status(404).send("User not found");
        }

        const isPasswordMatch = await bcrypt.compare(password, user.password);
        if (!isPasswordMatch) {
            return res.status(401).send("Invalid email or password");
        }

        const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, { ex-
piresIn: "1h" });

        user.tokens.push({ token });
        await user.save();

        res.json({ authToken: token });
    } catch (error) {
        res.status(500).send(error.message);
    }
});

router.post('/logout', auth, async (req, res) => {
    try {
        req.user.tokens = req.user.tokens.filter(token => token.token !==
req.token);
        await req.user.save();
        res.send('Logged out successfully');
    } catch (error) {
        res.status(500).send(error.message);
    }
});

router.post('/logoutAll', auth, async (req, res) => {
    try {
        req.user.tokens = [];
        await req.user.save();
        res.send('Logged out from all devices successfully');
    } catch (error) {
        res.status(500).send(error.message);
    }
});

module.exports = router;
```

Висновок: під час виконання даної лабораторної роботи, я вдосконалив свої знання з Node.

GitHub: https://github.com/AdamLahovskyi/NodeJS