Лабораторна робота №3

Mongoose Rest API

mongoose.js

const mongoose = require('mongoose');

const dotenv = require('dotenv');

dotenv.config();

mongoose.connect(process.env.MONGODB\_URL);

module.exports = mongoose;

task.js

const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({

  title: {

    type: String,

    required: true,

    trim: true

  },

  description: {

    type: String,

    required: true,

    trim: true

  },

  completed: {

    type: Boolean,

    default: false

  }

});

const Task = mongoose.model('Task', taskSchema);

module.exports = Task;

models\user.js

const mongoose = require('mongoose');

const validator = require('validator');

const userSchema = new mongoose.Schema({

  name: {

    type: String,

    required: true,

    trim: true

  },

  email: {

    type: String,

    required: true,

    unique: true,

    trim: true,

    lowercase: true,

    validate(value) {

      if (!validator.isEmail(value)) {

        throw new Error('Invalid email');

      }

    }

  },

  password: {

    type: String,

    required: true,

    minlength: 7,

    trim: true,

    validate(value) {

      if (value.toLowerCase().includes('password')) {

        throw new Error('Password cannot contain "password"');

      }

    }

  },

  age: {

    type: Number,

    default: 0,

    validate(value) {

      if (value < 0) {

        throw new Error('Age must be a positive number');

      }

    }

  }

});

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

module.exports = User;

models\task.js

const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({

  title: {

    type: String,

    required: true,

    trim: true

  },

  description: {

    type: String,

    required: true,

    trim: true

  },

  completed: {

    type: Boolean,

    default: false

  }

});

const Task = mongoose.model('Task', taskSchema);

module.exports = Task;

routers\task.js

const express = require('express');

const Task = require('../models/task');

const router = new express.Router();

// Отримання всіх задач

router.get('/', async (req, res) => {

  try {

    const tasks = await Task.find({});

    res.send(tasks);

  } catch (e) {

    res.status(500).send();

  }

});

// Отримання задачі за ID

router.get('/:id', async (req, res) => {

  const \_id = req.params.id;

  try {

    const task = await Task.findById(\_id);

    if (!task) {

      return res.status(404).send();

    }

    res.send(task);

  } catch (e) {

    res.status(500).send();

  }

});

// Додавання нової задачі

router.post('/', async (req, res) => {

  const task = new Task(req.body);

  try {

    await task.save();

    res.status(201).send(task);

  } catch (e) {

    res.status(400).send(e);

  }

});

// Оновлення задачі за ID

router.patch('/:id', async (req, res) => {

  const updates = Object.keys(req.body);

  const allowedUpdates = ['title', 'description', 'completed'];

  const isValidOperation = updates.every((update) => allowedUpdates.includes(update));

  if (!isValidOperation) {

    return res.status(400).send({ error: 'Invalid updates!' });

  }

  try {

    const task = await Task.findById(req.params.id);

    if (!task) {

      return res.status(404).send();

    }

    updates.forEach((update) => (task[update] = req.body[update]));

    await task.save();

    res.send(task);

  } catch (e) {

    res.status(400).send(e);

  }

});

// Видалення задачі за ID

router.delete('/:id', async (req, res) => {

  try {

    const task = await Task.findByIdAndDelete(req.params.id);

    if (!task) {

      return res.status(404).send();

    }

    res.send(task);

  } catch (e) {

    res.status(500).send();

  }

});

// Видалення всіх задач

router.delete('/', async (req, res) => {

  try {

    await Task.deleteMany({});

    res.send({ message: 'All tasks deleted' });

  } catch (e) {

    res.status(500).send();

  }

});

module.exports = router;

// const task = new Task({

//   title: '  My Task  ',

//   description: '  kor tasktask-manager-api.  ',

// });

// task.save()

//   .then(() => {

//       console.log(task); // Успішне збереження

//   })

//   .catch((error) => {

//       console.log('Error:', error.message); // Обробка помилки

//   });

Routers\user.js

// routers/user.js

const express = require('express');

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

const router = new express.Router();

// Отримання всіх користувачів

router.get('/', async (req, res) => {

  try {

    const users = await User.find({});

    res.send(users);

  } catch (e) {

    console.error(e);

    res.status(500).send(e);

  }

});

// Отримання користувача за ID

router.get('/:id', async (req, res) => {

  const \_id = req.params.id;

  try {

    const user = await User.findById(\_id);

    if (!user) {

      return res.status(404).send();

    }

    res.send(user);

  } catch (e) {

    console.error(e);

    res.status(500).send(e);

  }

});

// Додавання нового користувача

router.post('/', async (req, res) => {

  const user = new User(req.body);

  try {

    await user.save();

    res.status(201).send(user);

  } catch (e) {

    console.error(e);

    res.status(400).send(e.message);

  }

});

// Оновлення користувача за ID

router.patch('/:id', async (req, res) => {

  const updates = Object.keys(req.body);

  const allowedUpdates = ['name', 'email', 'password', 'age'];

  const isValidOperation = updates.every((update) => allowedUpdates.includes(update));

  if (!isValidOperation) {

    return res.status(400).send({ error: 'Invalid updates!' });

  }

  try {

    const user = await User.findById(req.params.id);

    if (!user) {

      return res.status(404).send();

    }

    updates.forEach((update) => (user[update] = req.body[update]));

    await user.save();

    res.send(user);

  } catch (e) {

    console.error(e);

    res.status(400).send(e);

  }

});

// Видалення користувача за ID

router.delete('/:id', async (req, res) => {

  try {

    const user = await User.findByIdAndDelete(req.params.id);

    if (!user) {

      return res.status(404).send();

    }

    res.send(user);

  } catch (e) {

    console.error(e);

    res.status(500).send(e);

  }

});

// Видалення всіх користувачів

router.delete('/', async (req, res) => {

  try {

    await User.deleteMany({});

    res.send({ message: 'All users deleted' });

  } catch (e) {

    console.error(e);

    res.status(500).send(e);

  }

});

module.exports = router;

// const user = new User({

//   name: ' gosha mona ',

//   email: 'ynjnj@gamal.com',

//   password: '12345678k',

//   age: 19

// });

// user.save()

//   .then(() => {

//       console.log(user); // Успішне збереження

//   })

//   .catch((error) => {

//       console.log('Error:', error.message); // Обробка помилки

//   });

App.js

const express = require("express");

const mongoose = require("mongoose");

const userRouter = require("./routers/user"); // Підключення роутера для користувачів

const taskRouter = require("./routers/task"); // Підключення роутера для завдань

require("dotenv").config();

const app = express();

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

mongoose.connect('mongodb://127.0.0.1:27017/', { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => console.log('Connected to the database'))

    .catch(error => console.error('Connection error', error));

app.use(express.json());

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

app.use("/tasks", taskRouter);

app.listen(port, () => {

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

});

.env

MONGODB\_URL=mongodb://127.0.0.1:27017/

PORT=3000

package.json

{

  "name": "task-manager-api",

  "version": "1.0.0",

  "description": "",

  "main": "src/app.js",

  "scripts": {

    "start": "node src/app.js",

    "dev": "nodemon src/app.js"

  },

  "dependencies": {

    "dotenv": "^10.0.0",

    "express": "^4.17.1",

    "mongodb": "^6.6.2",

    "mongoose": "^6.0.13",

    "validator": "^13.6.0"

  },

  "devDependencies": {

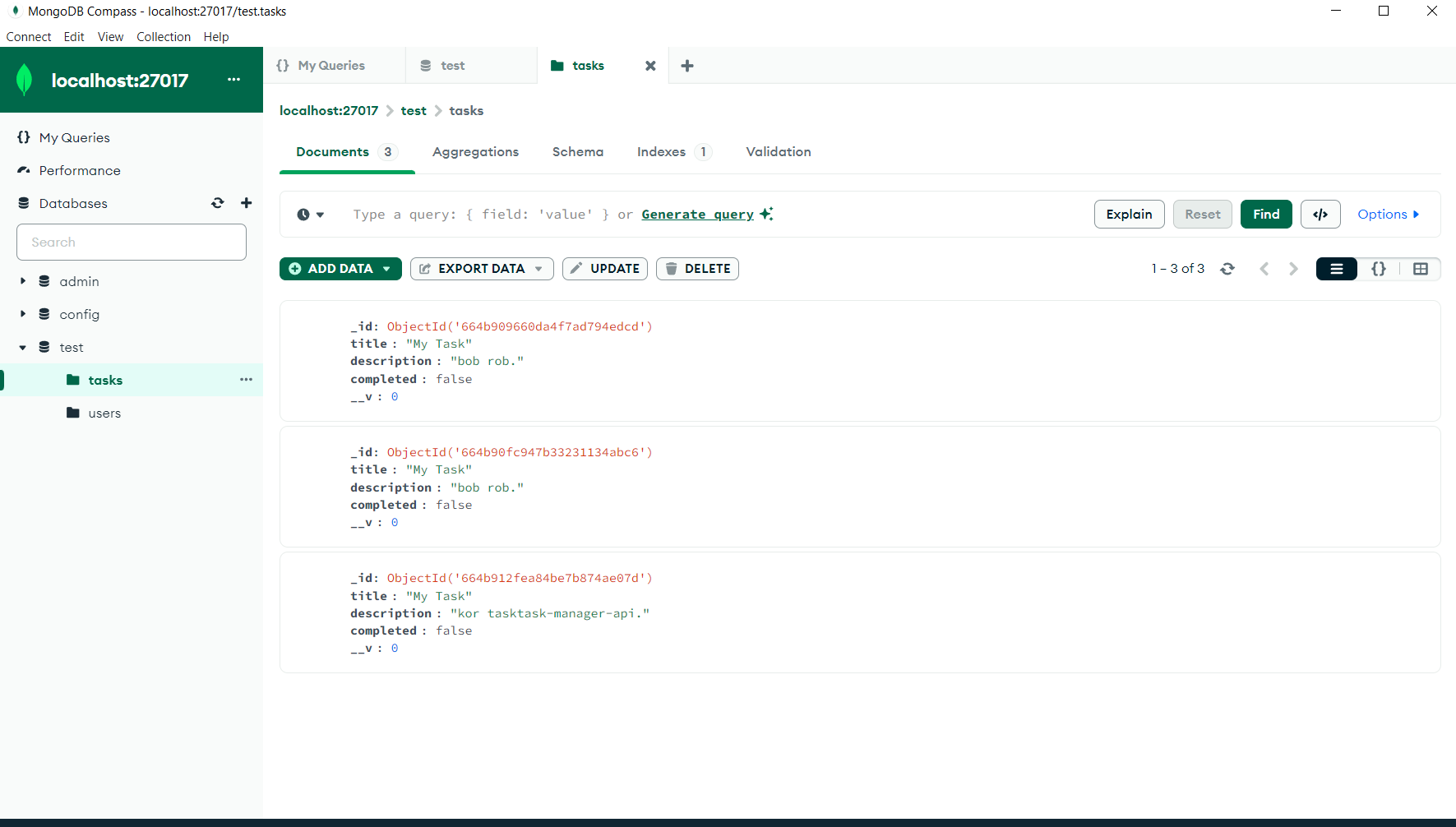
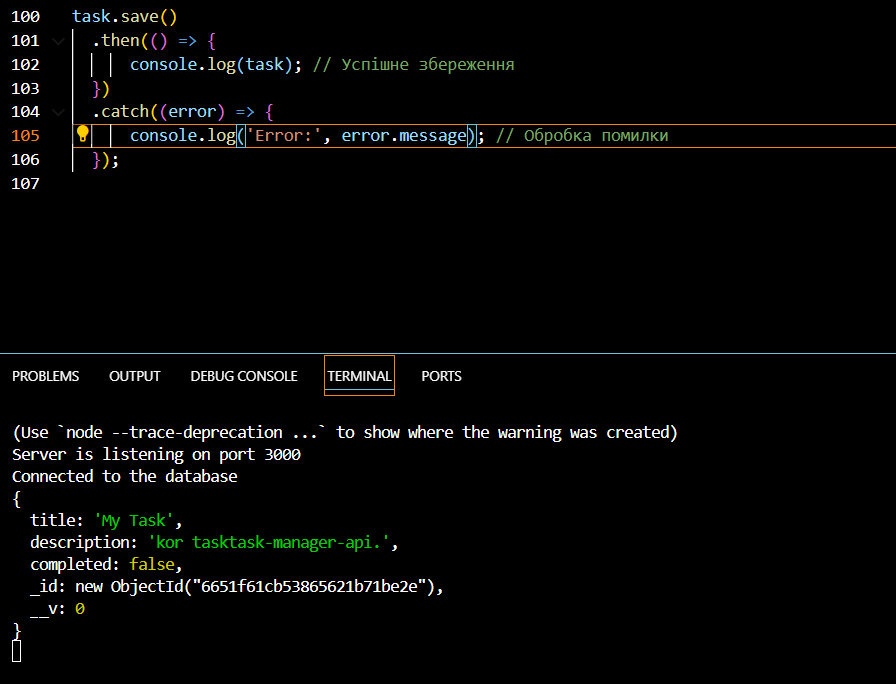
    "nodemon": "^2.0.15"

  },

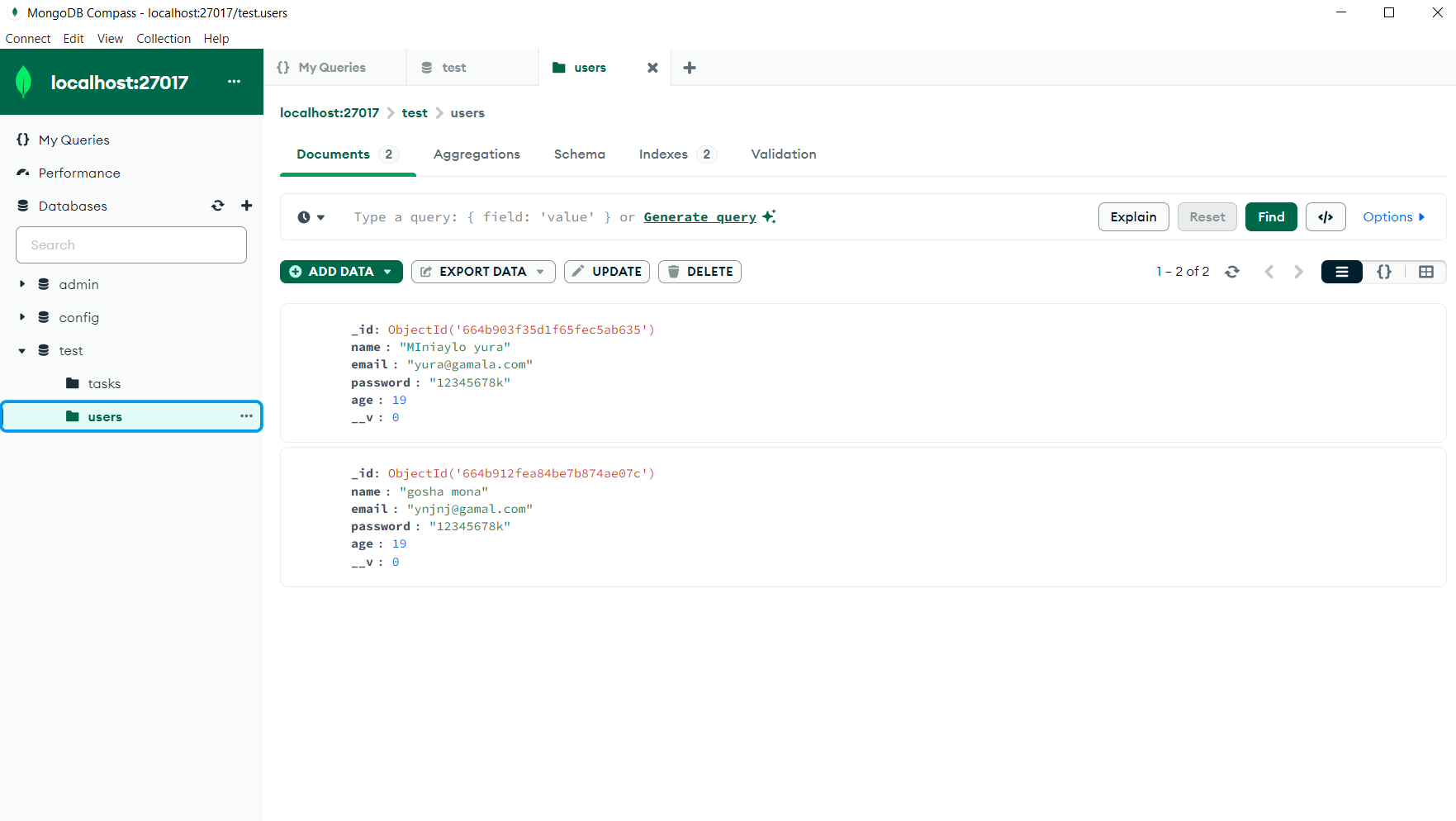
  "author": "",

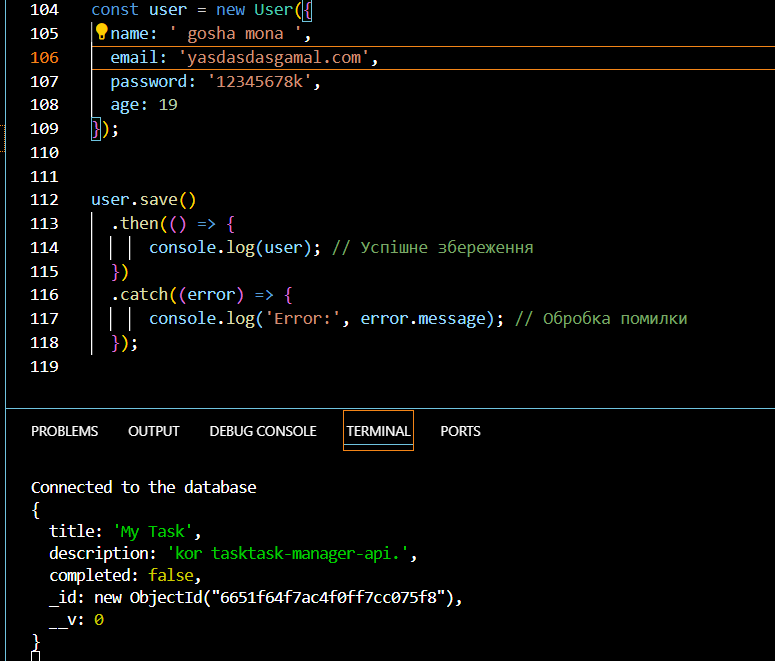
  "license": "ISC"

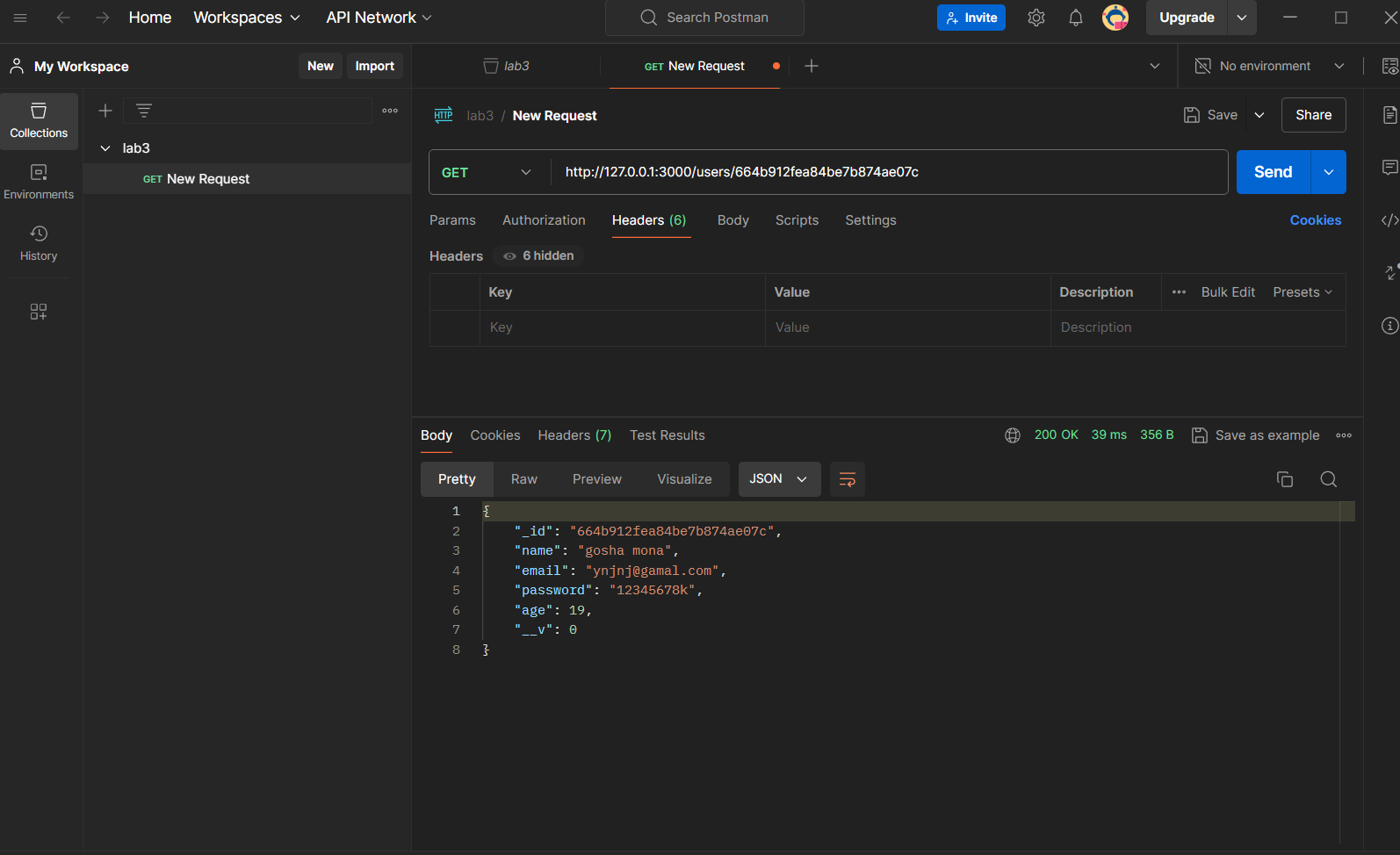
}

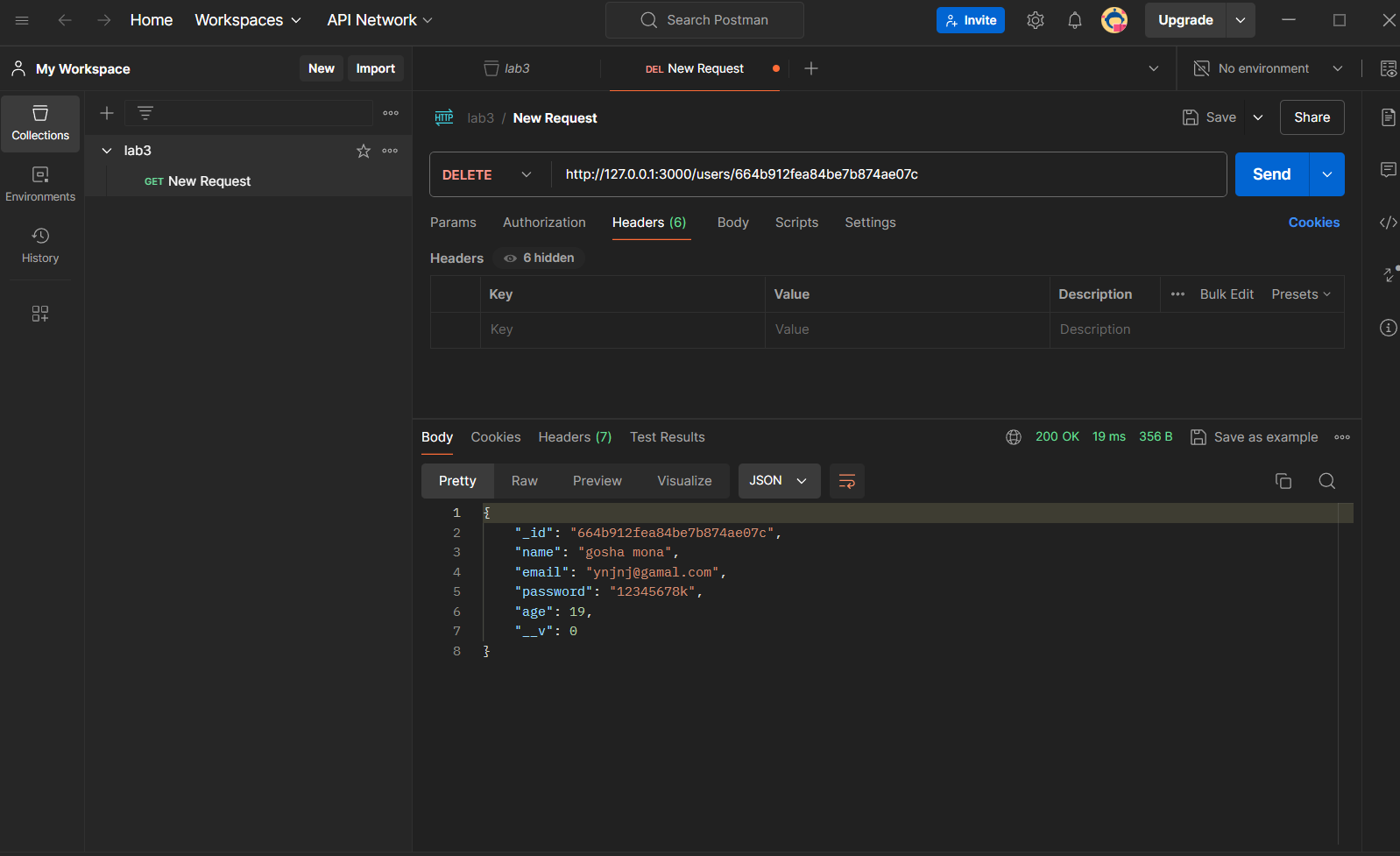
  
Зображення 1 

Зображення 2

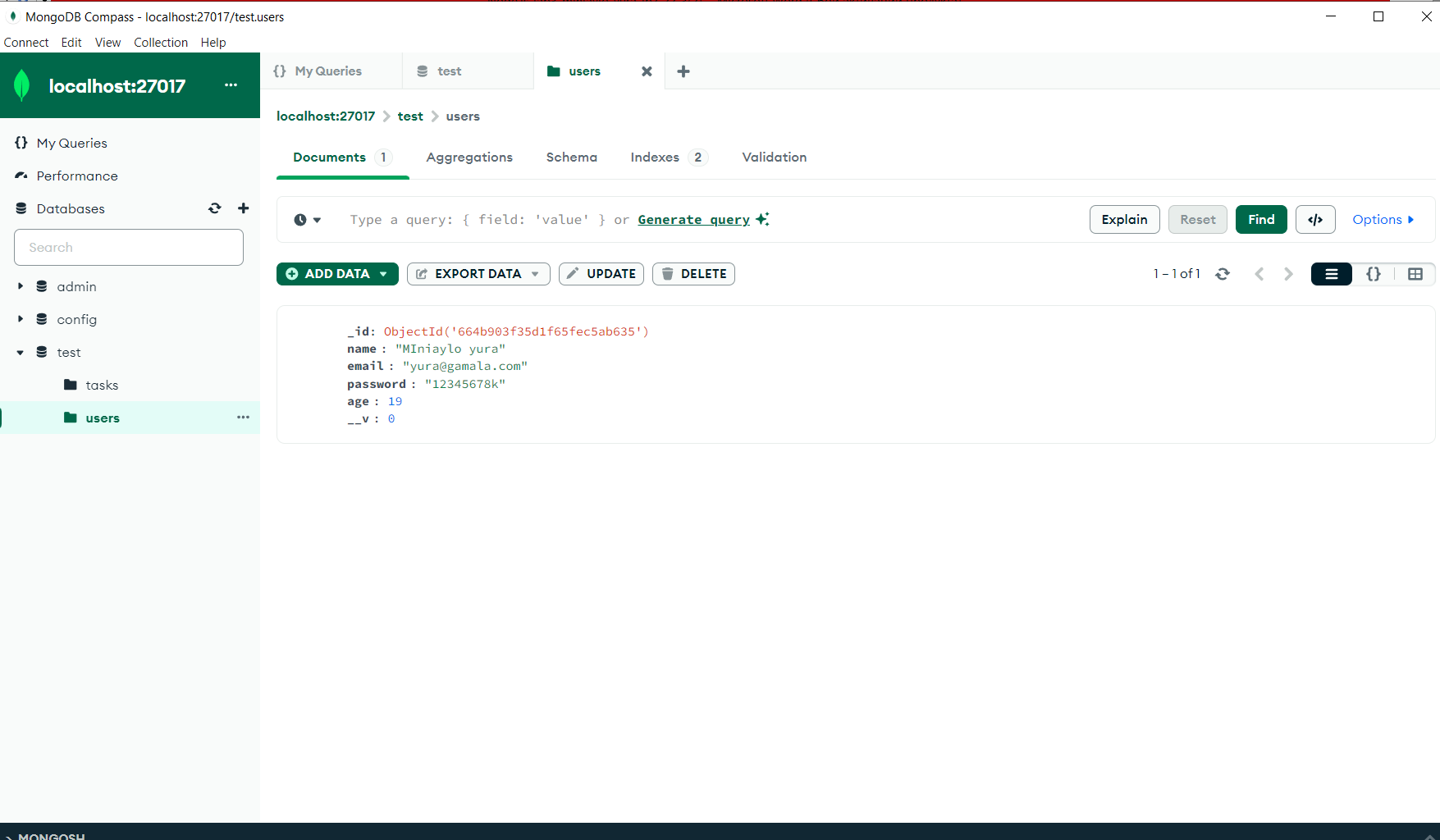


Зображення 3 Зображення 4



Зображення 5

Зображення 6



Зображення 7