**Лабораторна робота № 6**

**Тема:** Введення в NodeJS розробку. Data Base. Create RESTFull Api.

**Мета:** Навчитись працювати з NodeJS, реалізувати RESTFull Api.

**Завдання**

19. Видача банківських кредитів

Ви – керівник інформаційно-аналітичного центру коммерційного банку. Одним з існуючих видів діяльності Вашого банку є видача кредитів юридичним особам. Ваше завдання – стеження за динамікою роботи кредитного відділу.

В залежності від умов одержання кредиту, відсоткової ставки і терміну повернення, всі кредитні операції розподіляються на декілька основних видів. Кожний з цих видів має свою назву.

Кредит може одержати юридична особа (клієнт), яка при реєстрації надає наступні відомості: назва, вид власності, адреса, телефон, контактна особа. Кожний факт видачі кредиту реєструється банком, при цьому фіксується сума кредиту, клієнт і дата видачі.

Необхідно враховувати в системі ще й дату фактичного повернення грошей. Необхідно також враховувати, що кредит може повертатися частинами, і за затримку повернення кредиту нараховуються штрафи.

**Хід роботи**

server.js

const ***express*** = require('express');  
  
const app = ***express***();  
app.use(***express***.***json***());  
  
const dotenv = require('dotenv').config();  
  
require('./initDB')();  
  
app.get('/', (req, res) => {  
 res.send('Ви – керівник інформаційно-аналітичного центру коммерційного банку. Одним з існуючих\n' +  
 'видів діяльності Вашого банку є видача кредитів юридичним особам. Ваше завдання –\n' +  
 'стеження за динамікою роботи кредитного відділу.\n' +  
 'В залежності від умов одержання кредиту, відсоткової ставки і терміну повернення, всі\n' +  
 'кредитні операції розподіляються на декілька основних видів. Кожний з цих видів має свою\n' +  
 'назву.\n' +  
 'Кредит може одержати юридична особа (клієнт), яка при реєстрації надає наступні\n' +  
 'відомості: назва, вид власності, адреса, телефон, контактна особа. Кожний факт видачі\n' +  
 'кредиту реєструється банком, при цьому фіксується сума кредиту, клієнт і дата видачі.\n' +  
 'Необхідно враховувати в системі ще й дату фактичного повернення грошей. Необхідно\n' +  
 'також враховувати, що кредит може повертатися частинами, і за затримку повернення\n' +  
 'кредиту нараховуються штрафи.')  
})  
  
const ***clientRouter*** = require("./routes/clientRouter.js");  
const ***creditRouter*** = require("./routes/creditRouter.js");  
  
app.use("/api/clients", ***clientRouter***);  
app.use("/api/credits", ***creditRouter***);  
  
const PORT = ***process***.env.PORT || 8080;  
  
app.listen(PORT, () => {  
 ***console***.log('Server started on port ' + PORT + '...');  
});

.env-example

PORT=example\_port  
MONGODB\_URI=example\_uri  
DB\_NAME=example\_db  
DB\_USER=example\_user  
DB\_PASS=example\_password

initDB.js

const ***mongoose*** = require('mongoose');  
  
module.exports = () => {  
 ***mongoose***.connect(***process***.env.MONGODB\_URI, {  
 dbName: ***process***.env.DB\_NAME,  
 user: ***process***.env.DB\_USER,  
 pass: ***process***.env.DB\_PASS,  
 useNewUrlParser: true,  
 useUnifiedTopology: true  
 })  
 .then(() => {  
 ***console***.log('Mongodb connected....');  
 })  
 .catch(err => ***console***.log(err.message));  
  
 ***mongoose***.***connection***.on('connected', () => {  
 ***console***.log('Mongoose connected to db...');  
 });  
  
 ***mongoose***.***connection***.on('error', err => {  
 ***console***.log(err.message);  
 });  
  
 ***mongoose***.***connection***.on('disconnected', () => {  
 ***console***.log('Mongoose connection is disconnected...');  
 });  
  
 ***process***.on('SIGINT', () => {  
 ***mongoose***.***connection***.close(() => {  
 ***console***.log(  
 'Mongoose connection is disconnected due to app termination...'  
 );  
 ***process***.exit(0);  
 });  
 });  
};

clientModel.js

const ***mongoose*** = require('mongoose');  
const Schema = ***mongoose***.Schema;  
  
const ClientSchema = new Schema({  
 name: {  
 type: ***String***,  
 required: true  
 },  
 kindOfProperty: {  
 type: ***String***,  
 required: true  
 },  
 address: {  
 type: ***String***,  
 required: true  
 },  
 contactPerson: {  
 type: ***String***,  
 required: true  
 }  
});  
  
const ***Client*** = ***mongoose***.model('clients', ClientSchema);  
module.exports = ***Client***;

creditModel.js

const ***mongoose*** = require('mongoose');  
const Schema = ***mongoose***.Schema;  
const {ObjectId} = require("mongodb");  
  
class CreditType {  
 constructor(name, interestRate, term) {  
 this.name = name;  
 this.interestRate = interestRate;  
 this.term = term;  
 }  
  
 static *INVESTMENT* = new CreditType('INVESTMENT', 5, 15);  
 static *FARMER* = new CreditType('FARMER', 10, 20);  
 static *TREASURY* = new CreditType('TREASURY', 15, 5);  
 static *EASY* = new CreditType('EASY', 3, 1);  
}  
  
const CreditSchema = new Schema({  
 creditType: {  
 type: ***Object***,  
 required: true  
 },  
 amount: {  
 type: ***Number***,  
 required: true  
 },  
 clientId: {  
 type: ObjectId,  
 required: true  
 },  
 dateOfIssue: {  
 type: ***Date***,  
 required: true  
 },  
 dateOfRefund: {  
 type: ***Date***,  
 required: false  
 },  
 fine: {  
 type: ***Number***,  
 required: false  
 }  
});  
  
const Credit = ***mongoose***.model('credits', CreditSchema);  
module.exports = {Credit, CreditType};

baseController.js

module.exports = {  
 sendError(res, status, message) {  
 res.status(status).send({  
 message: message  
 });  
 },  
  
 sendResult(res, message, result) {  
 res.send({  
 message: message,  
 result: ***Array***.isArray(result)?[...result]:result  
 });  
 }  
}

clientController.js

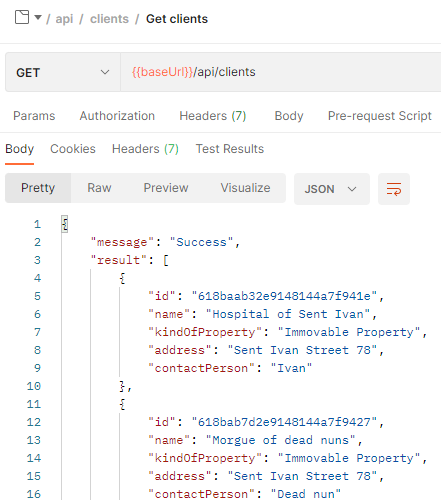
const ***Client*** = require('../models/clientModel');  
const {sendError, sendResult} = require('./baseController');  
const {ObjectId} = require("mongodb");  
  
module.exports = {  
 addClient: async (req, res) => {  
 try {  
 const client = new ***Client***(req.body);   
 const found = await ***Client***.findOne({"name": client.name});  
 if (!found) {  
 await client.save();  
 sendResult(res, 'Success',  
 {  
 "id": client.\_id,  
 "name": client.name,  
 "kindOfProperty": client.kindOfProperty,  
 "address": client.address,  
 "contactPerson": client.contactPerson  
 });  
 } else {  
 sendError(res, 400, 'Client with this name already exists')  
 }  
 } catch (error) {  
 sendError(res, 400, 'Bad request')  
 }  
 },  
 getClients: async (req, res) => {  
 try {  
 const clients = await ***Client***.find({});  
 if (clients.length) {  
 sendResult(res, 'Success', clients.map((item) => {  
 return {  
 "id": item.\_id,  
 "name": item.name,  
 "kindOfProperty": item.kindOfProperty,  
 "address": item.address,  
 "contactPerson": item.contactPerson  
 }  
 }));  
 } else {  
 sendError(res, 400, 'Clients are missing');  
 }  
 } catch (error) {  
 sendError(res, 400, 'Bad request');  
 }  
 },  
 getClient: async (req, res) => {  
 try {  
 const client = await ***Client***.findOne({\_id: new ObjectId(req.body.id)});  
  
 if (client) {  
 sendResult(res, 'Success', {  
 "id": client.\_id,  
 "name": client.name,  
 "kindOfProperty": client.kindOfProperty,  
 "address": client.address,  
 "contactPerson": client.contactPerson  
 });  
 } else {  
 sendError(res, 400, 'Client are missing')  
 }  
 } catch (error) {  
 sendError(res, 400, 'Bad request')  
 }  
 }  
}

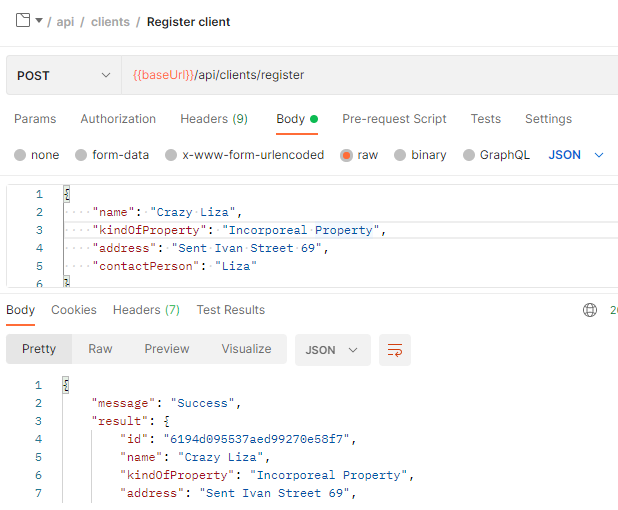
creditController.js

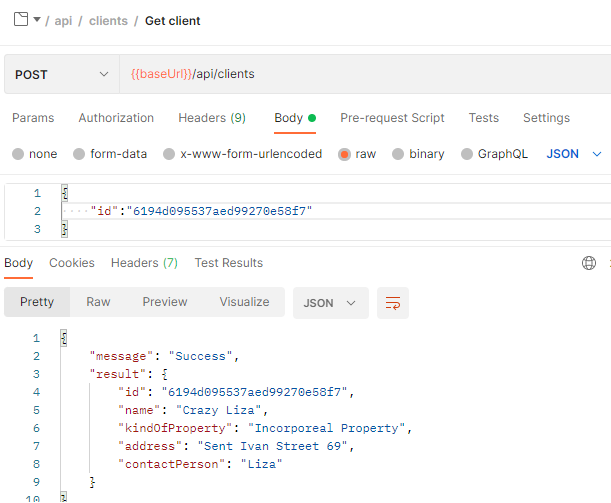
const {Credit, CreditType} = require('../models/creditModel');  
const ***Client*** = require('../models/clientModel');  
const {ObjectId} = require("mongodb");  
const {sendError, sendResult} = require('./baseController');  
  
async function getOpenCreditByClientId(clientId) {  
 const found = await Credit.find({"clientId": clientId});  
 return found.find((item) => !item.dateOfRefund);  
}  
  
function round(number) {  
 return parseFloat(number.toFixed(2))  
}  
  
module.exports = {  
 addCredit: async (req, res) => {  
 try {  
 const credit = new Credit({...req.body, creditType: CreditType[req.body.creditType]});  
 if (!await getOpenCreditByClientId(credit.clientId)) {  
 const client = await ***Client***.findOne({\_id: new ObjectId(credit.clientId)});  
 if (client) {  
 await credit.save();  
 sendResult(res, 'Success', {  
 "id": credit.\_id,  
 "creditType": credit.creditType.name,  
 "amount": credit.amount,  
 "clientId": credit.clientId,  
 "dateOfIssue": credit.dateOfIssue,  
 "dateOfRefund": credit.dateOfRefund || 'The credit is not closed'  
 });  
 } else {  
 sendError(res, 400, 'This client is missing');  
 }  
 } else {  
 sendError(res, 400, 'This client already has an open credit');  
 }  
 } catch (error) {  
 sendError(res, 400, 'Bad request')  
 }  
 },  
 getCredits: async (req, res) => {  
 try {  
 const credits = await Credit.find({});  
 if (credits.length) {  
 sendResult(res, 'Success', credits.map((item) => {  
 return {  
 "id": item.\_id,  
 "creditType": item.creditType.name,  
 "amount": item.amount,  
 "clientId": item.clientId,  
 "dateOfIssue": item.dateOfIssue,  
 "dateOfRefund": item.dateOfRefund || 'The credit is not closed'  
 }  
 }));  
 } else {  
 sendError(res, 400, 'Credits are missing')  
 }  
 } catch (error) {  
 sendError(res, 400, 'Bad request')  
 }  
 },  
 payCredit: async (req, res) => {  
 try {  
 const credit = await Credit.findOne({\_id: new ObjectId(req.body.id)});  
 let change = 0;  
 let fine = 0;  
 if (await getOpenCreditByClientId(credit.clientId)) {  
 const realAmount = round(req.body.amount \* (1 - credit.creditType.interestRate / 100));  
 if (credit.amount > realAmount) {  
 credit.amount -= parseFloat(realAmount.toFixed(2));  
 } else {  
 change = realAmount - credit.amount;  
 credit.dateOfRefund = new ***Date***();  
 const diff = credit.dateOfRefund.getFullYear() - credit.dateOfIssue.getFullYear();  
 if (diff > credit.creditType.term) {  
 fine = credit.amount \* (1 + credit.creditType.interestRate / 100);  
 }  
 credit.amount = 0;  
 }  
 await credit.save();  
 sendResult(res, 'Success', {  
 "id": credit.\_id,  
 "creditType": credit.creditType.name,  
 "amountPaid": realAmount,  
 "amount": credit.amount,  
 "clientId": credit.clientId,  
 "dateOfIssue": credit.dateOfIssue,  
 "dateOfRefund": credit.dateOfRefund || 'The credit is not closed',  
 "fine": round(fine),  
 "change": round(change)  
 });  
 } else {  
 sendError(res, 400, 'Credit is missing or closed')  
 }  
 } catch (error) {  
 sendError(res, 400, 'Bad request')  
 }  
 }  
}

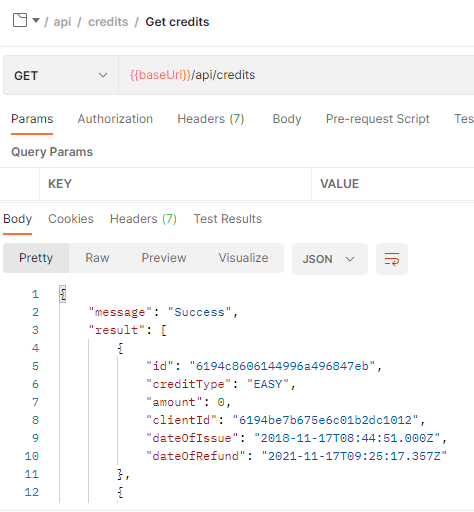
Результат

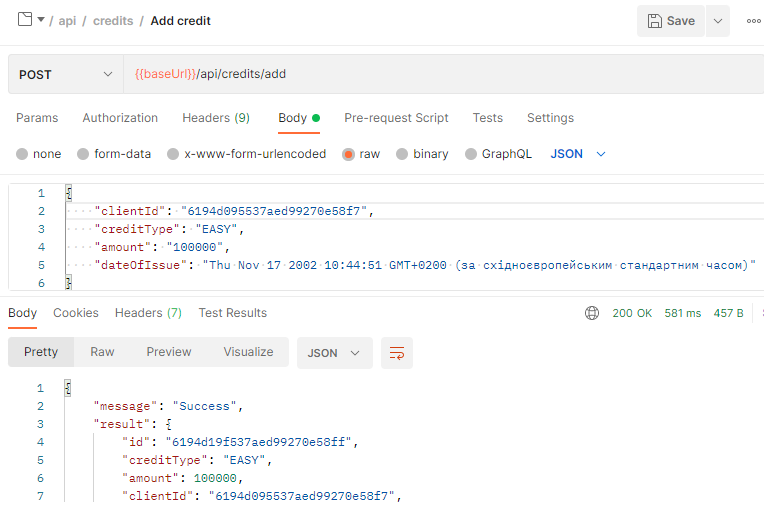
Успіх:

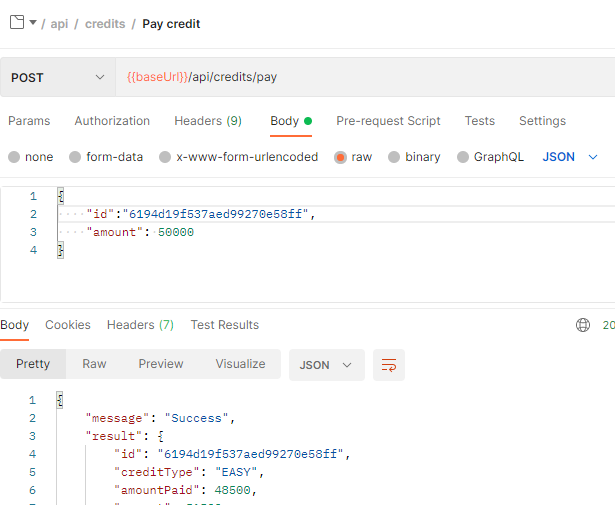


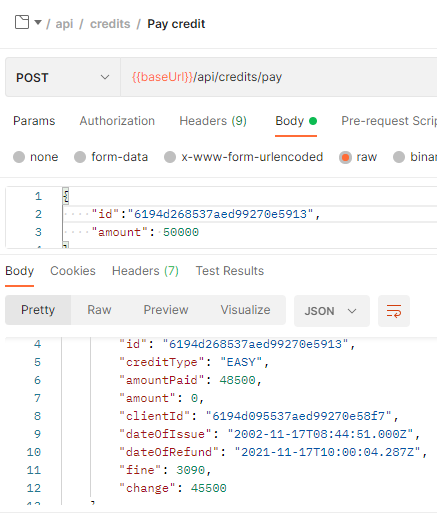




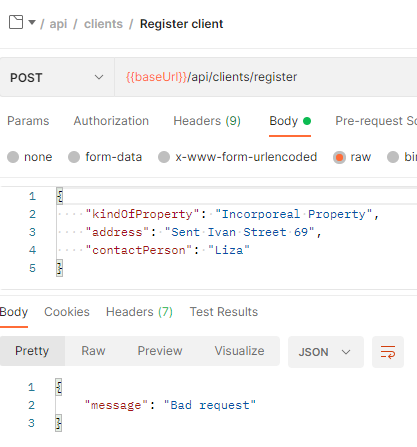


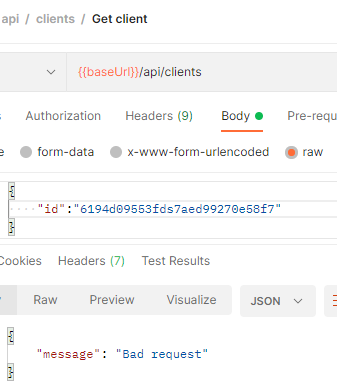


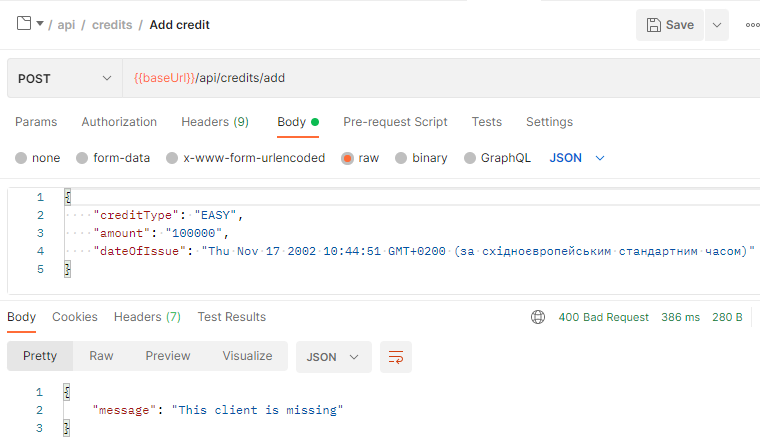


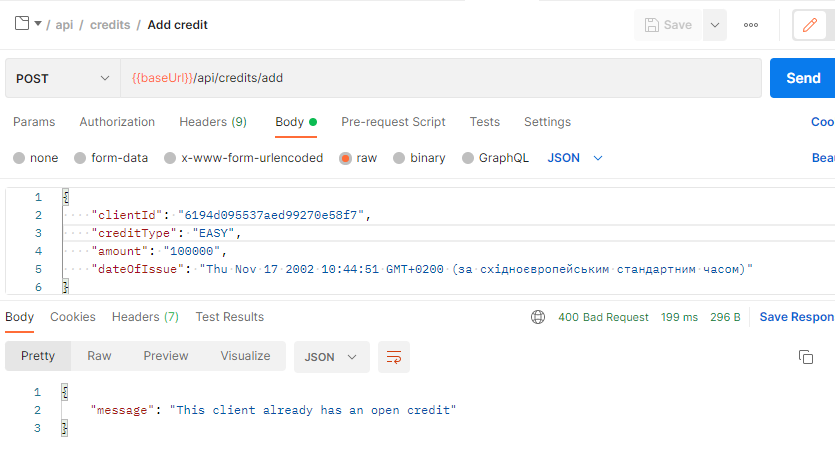


Невдача:









**Висновок:** Навчилась працювати з NodeJS, реалізувала RESTFull Api.