# Министерство Образования Республики Беларусь

Учреждение образования

«Гомельский государственный технический университет

имени П.О. Сухого»

Кафедра «Информатика»

по курсу: «Разработка Интернет приложений с использование NodeJS»

Лабораторная работа №5

# «Работа с базами данных в Node.js»

Выполнил: студент группы ИП-32

Суховенко Э. С.

Допуск к защите: Проверил: преподаватель

Дата защиты: Процкая М. А.

Гомель 2022

**Цель работы:** изучить основы работы по написанию скриптов для работы с базами данных в Node.js. Получить навыки работы с orm в Node.js. Научиться создавать и тестировать простые приложения.

**Задание 1:** реализовать Rest Api (добавлении, удаление, обновление, получение) с использование express и базы данных. Предметную область взять по усмотрению студента. Использовать sql запросы к обращению к базе данных.

**Листинг программы:**

// основное приложение

**index.js:**

const express = require("express");

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

const app = express();

const router = require("./routers/index");

app.use(express.json());

app.use("/api", router);

app.listen(port, (err) =>{

if (err) {

console.log(`Error: ${err}`)

} else {

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

}

});

**//** один из роутеров

**employeeRouter.js:**

const express = require("express");

const router = express.Router();

const sqlite3 = require("sqlite3").verbose();

const db = new sqlite3.Database("../database.db");

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

db.all("SELECT \* FROM Employee", (err, rows) => {

if (err) {

res.status(500).send(err);

} else {

res.status(200).json(rows);

}

});

});

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

const id = req.params.id;

db.get("SELECT \* FROM Employee WHERE id = ?", [id], (err, row) => {

if (err) {

res.status(500).send(err);

} else {

res.status(200).json(row);

}

});

});

router.put("/:id", (req, res) => {

const id = req.params.id;

const { Name, Job\_Title, Phone\_no, Sallary} = req.body;

db.run(`UPDATE Employee SET Name = ?, Job\_Title = ?, Phone\_no = ?, Sallary = ? WHERE id = ?`,

[Name, Job\_Title, Phone\_no, Sallary, id], function (err) {

if (err) {

res.status(500).send(err);

} else {

res.status(200).json({ message: "Employee updated successfully" });

}

});

});

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

const { Name, Job\_Title, Phone\_no, Sallary, Dept\_id, Project\_id} = req.body;

db.run(`INSERT INTO Employee (Name, Job\_Title, Phone\_no, Sallary) VALUES (?, ?, ?, ?, ?, ?)`,

[Name, Job\_Title, Phone\_no, Sallary, Dept\_id, Project\_id], function (err) {

if (err) {

res.status(500).send(err);

} else {

res.status(200).json({ message: "Employee added successfully" });

}

});

});

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

const id = req.params.id;

db.run(`DELETE FROM Employee WHERE id = ?`, [id], function (err) {

if (err) {

res.status(500).send(err);

} else {

res.status(200).json({ message: "Employee deleted successfully" });

}

});

});

module.exports = router;

// общий роутер

**index.js:**

const express = require("express");

const router = express.Router();

const employeeRouter = require("./employeeRouter");

const departmentRouter = require("./departmentRouter");

const projectRouter = require("./projectRouter");

router.use("/employee", employeeRouter);

router.use("/department", departmentRouter);

router.use("/project", projectRouter);

module.exports = router;

**Задание 2** Повторить задание 1, но с использование ORM.

**Листинг программы:**

**//** основное приложение является аналогичным заданию 1

// общий роутер является аналогичным заданию 1

// подключение orm к базе данных

const Sequelize = require("sequelize");

const sequelize = new Sequelize('sqlite::memory:', {

host: "localhost",

dialect: "sqlite",

pool: {

max: 5,

min: 0,

idle: 10000

},

storage: "../database.db"

});

module.exports = sequelize;

// пример одной из модели базы данных

const { Sequelize } = require("sequelize");

const sequlize = require('../database');

const Employee = sequlize.define('Employee', {

id: {

type: Sequelize.INTEGER,

primaryKey: true,

autoIncrement: true

},

Name: {

type: Sequelize.STRING,

allowNull: false

},

Job\_Title: {

type: Sequelize.STRING,

allowNull: false

},

Sallary: {

type: Sequelize.INTEGER,

allowNull: false

},

Dept\_id: {

type: Sequelize.INTEGER,

allowNull: false,

references: {

model: 'Department',

key: 'Dept\_id'

}

},

Project\_id: {

type: Sequelize.INTEGER,

allowNull: false,

references: {

model: 'Project',

key: 'Project\_id'

}

}

}, {

tableName: 'Employee',

timestamps: false

});

module.exports = Employee;

// пример одного из роутеров

const express = require("express");

const router = express.Router();

const Employee = require("../models/employeeModel");

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

Employee.findAll({

}).then(employees => {

res.json(employees);

});

});

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

Employee.findOne({

where: {

id: req.params.id

}

}).then(employee => {

res.json(employee);

});

});

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

Employee.create({

Name: req.body.Name,

Job\_Title: req.body.Job\_Title,

Sallary: req.body.Sallary,

Dept\_id: req.body.Dept\_id,

Project\_id: req.body.Project\_id

}).then(employee => {

console.log(employee);

res.json(employee);

});

});

router.put("/:id", (req, res) => {

Employee.update({

Name: req.body.Name,

Job\_Title: req.body.Job\_Title,

Sallary: req.body.Sallary,

Dept\_id: req.body.Dept\_id,

Project\_id: req.body.Project\_id

}, {

where: {

id: req.params.id

}

}).then(employee => {

res.json(employee);

});

});

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

Employee.destroy({

where: {

id: req.params.id

}

}).then(employee => {

res.json(employee);

});

});

module.exports = router;

**Задание 3**. Протестировать ранее созданное Api используя библиотеки для тестов – Jest, Mocha, supertest.

**Листинг программы:**

**//** Используя Jest и supertest

**employee.test.js:**

const request = require("supertest");

const app = require("../index");

describe("GET api/employee", () => {

it("should return all employees", (done) => {

request(app)

.get("/api/employee")

.expect(200)

.expect("Content-Type", /json/)

.end((err, res) => {

if (err) {

return done(err);

}

done();

});

});

});

describe("GET api/employee/:id", () => {

it("should return a single employee", (done) => {

request(app)

.get("/api/employee/1")

.expect(200)

.expect("Content-Type", /json/)

.end((err, res) => {

if (err) {

return done(err);

}

done();

});

});

});

describe("POST api/employee", () => {

it("should create a new employee", (done) => {

request(app)

.post("/api/employee/")

.send({

Name: "test",

Job\_Title: "test",

Sallary: 60000,

Dept\_id: 1,

Project\_id: 1

})

.end((err, res) => {

console.log(res.body);

if (err) {

return done(err);

}

done();

});

});

}

);

describe("PUT api/employee/:id", () => {

it("should update a single employee", (done) => {

request(app)

.put("/api/employee/3")

.send({

Name: "test",

Job\_Title: "test",

Sallary: 60000,

Dept\_id: 1,

Project\_id: 1

})

.expect(200)

.end((err, res) => {

console.log(res.body);

if (err) {

return done(err);

}

done();

});

});

}

);

**Задание 4**. Реализовать паттерн MVC, создать три папки в которых будут хранится классы моделей, контроллеров и представлений. Для представлений можно использовать handlebars. Классы сервисов и маршрутов находятся в отдельных файлах.

**Листинг программы:**

**//** добавим к существующему проекту класс контроллеров, которые будут возвращать представления.

**employeeController.js:**

const Employee = require("../models/employeeModel");

// create class controller which return hbs with data

class EmployeeController {

// get all employees

static getAllEmployees(req, res) {

Employee.findAll({raw: true}).then(employees => {

res.render("index.hbs", {

employees: employees

});

});

}

// get one employee

static getOneEmployee(req, res) {

Employee.findOne({

where: {

id: req.params.id

},

raw: true

}).then(employee => {

res.render("index.hbs", {

employee: employee

});

});

}

static addEmployee(req, res) {

res.render("addEmployee.hbs",{});

}

// create employee

static createEmployee(req, res) {

Employee.create({

Name: req.body.Name,

Job\_Title: req.body.Job\_Title,

Sallary: req.body.Sallary,

Dept\_id: req.body.Dept\_id,

Project\_id: req.body.Project\_id

}).then(() => {

res.redirect("/");

});

}

// update employee

static updateEmployee(req, res) {

Employee.update({

Name: req.body.Name,

Job\_Title: req.body.Job\_Title,

Sallary: req.body.Sallary,

Dept\_id: req.body.Dept\_id,

Project\_id: req.body.Project\_id

}, {

where: {

id: req.params.id

}

}).then(() => {

res.redirect("/");

});

}

// delete employee

static deleteEmployee(req, res) {

Employee.destroy({

where: {

id: req.params.id

}

}).then(employee => {

res.redirect("/");

});

}

}

module.exports = EmployeeController;

**//** советующий роутер для представлений

**emplViewRouter.js:**

const express = require("express");

const router = express.Router();

const EmployeeController = require("../controllers/employeeController");

router.get("/create-employee", EmployeeController.addEmployee)

router.get("/", EmployeeController.getAllEmployees);

router.get("/:id", EmployeeController.getOneEmployee);

router.post("/", EmployeeController.createEmployee);

module.exports = router;

**//** Вид представлений

**layaout.hbs:**

<!DOCTYPE html>

<html>

<head>

<title>{{title}}</title>

<meta charset="utf-8" />

</head>

<style>

.box {

display: flex;

align-items: center;

justify-content: center;

}

input[type=text], select {

width: 100%;

padding: 12px 20px;

margin: 8px 0;

display: inline-block;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

}

input[type=submit] {

width: 100%;

background-color: #4CAF50;

color: white;

padding: 14px 20px;

margin: 8px 0;

border: none;

border-radius: 4px;

cursor: pointer;

}

input[type=submit]:hover {

background-color: #45a049;

}

.table {

width: 800px;

margin-bottom: 20px;

border: 1px solid #dddddd;

font-size: 35px;

font-family: Arial, Helvetica, sans-serif;

border-collapse: collapse;

}

.table th {

font-weight: bold;

padding: 5px;

background: #efefef;

border: 1px solid #dddddd;

}

.table td {

border: 1px solid #dddddd;

padding: 5px;

}

</style>

<body>

{{{body}}}

</body>

<html>

**index.hbs:**

<div class="box">

<div id="content">

<h1>Список Работников</h1>

<table class="table">

<tr><th>Имя</th><th>Зарплата</th><th>Работа</th></tr>

{{#if employees}}

{{#each employees}}

<tr><td>{{this.Name}}</td><td>{{this.Sallary}}</td><td>{{this.Job\_Title}}</td></tr>

{{/each}}

{{/if}}

{{#if employee}}

<tr><td>{{employee.Name}}</td><td>{{employee.Sallary}}</td><td>{{employee.Job\_Title}}</td></tr>

{{/if}}

</table>

</div>

</div>

**addEmployee.hbs:**

<div class="box">

<div class="content">

<h1>Добавление пользователя</h1>

<form action="/" method="post">

<label for="name">Имя</label>

<input type="text" name="name" id="name" /><br/>

<label for="sallary">Зарплата</label>

<input type="text" name="sallary" id="sallary" /><br/>

<label for="job\_title">Работа</label>

<input type="text" name="job\_title" id="job\_title" /><br/>

<input type="submit" value="Добавить" />

</form>

<a href="/">Назад</a>

</div>

</div>

**Результат работы программы:**

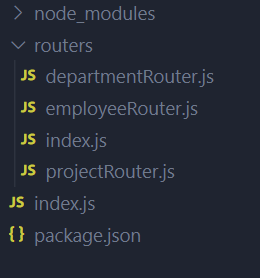
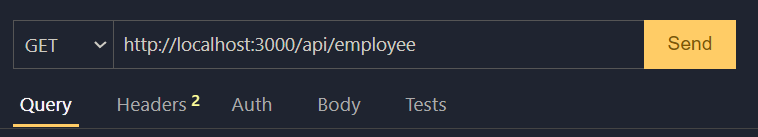


Рисунок 1 – структура приложения (Задание 1)



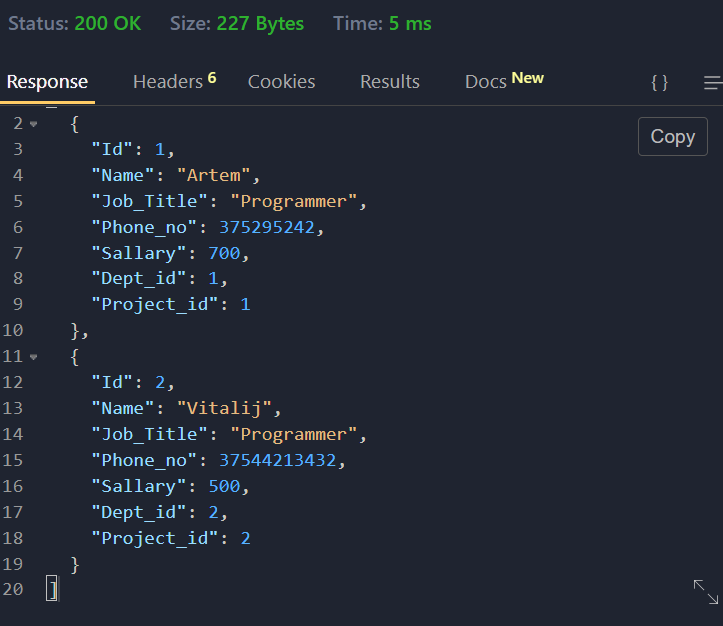


Рисунок 2 – Проверка метода GET сотрудников

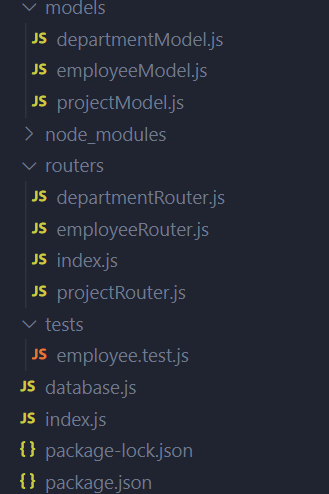


Рисунок 3 – структура приложения (Задание 2)

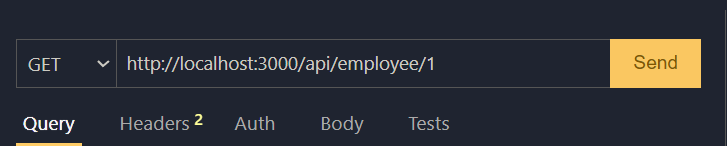
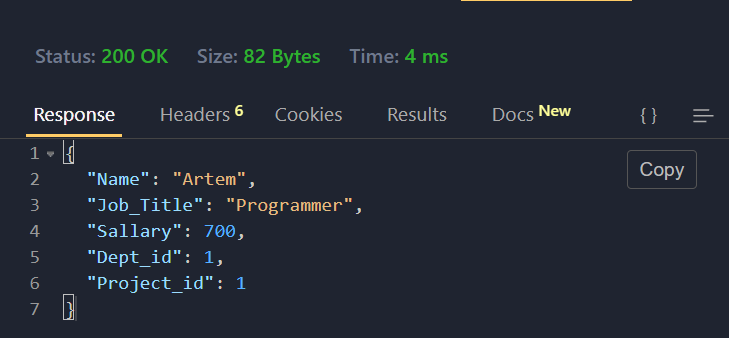
 

Рисунок 4 – Проверка метода GET сотрудников

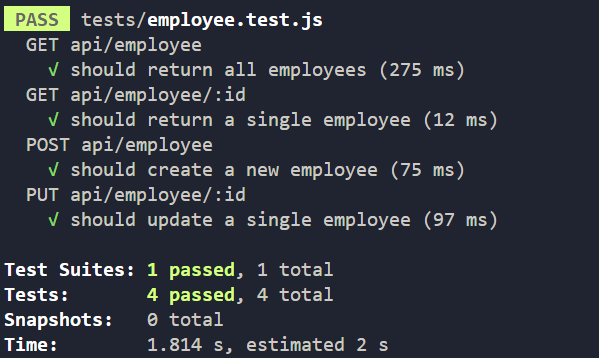


Рисунок 4 – Проверка Grud – методов используя Jest и supertest

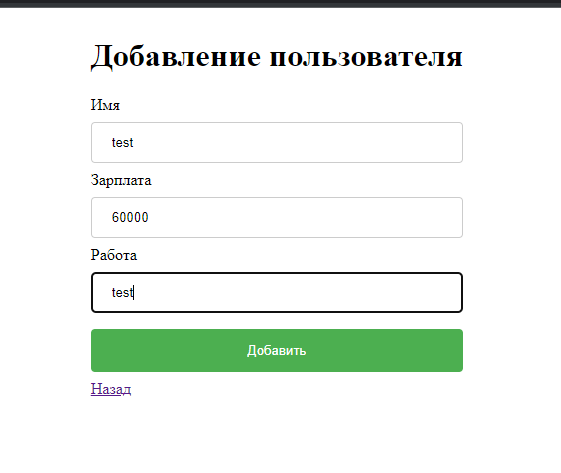


Рисунок 5 – Представление добавления работника

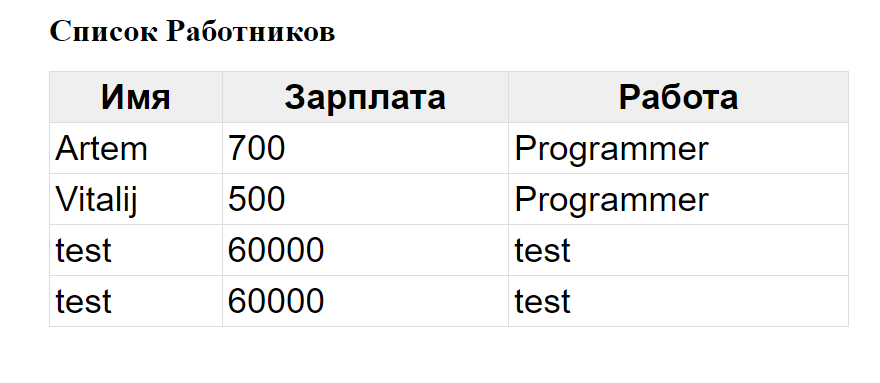


Рисунок 6 – Представление всех работников

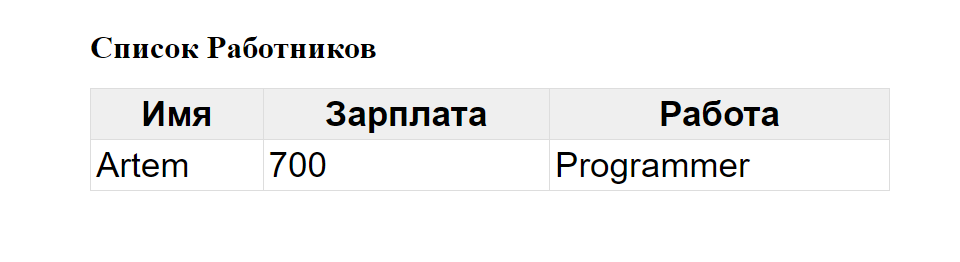


Рисунок 7 – Представление одного работника

**Вывод:** Были изучены основное способы работы с базами данных на платформе NodeJS, создано приложение на RestApi, а также протестировано с помощью советующих инструментов.