# Министерство Образования Республики Беларусь

Учреждение образования

«Гомельский государственный технический университет

имени П.О. Сухого»

Кафедра «Информатика»

по курсу: «Разработка приложений для Интернет»

Лабораторная работа №5

# «Работа с базами данных в Node.js»

Выполнил: студент группы ИП-31

Тарелко Денис Игоревич

Допуск к защите: Проверил: преподаватель

Дата защиты: Самовендюк Николай Владимирович

Гомель 2022

**Цель**: 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.

**Практическая часть:**

**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:**

const express = **require**("express");

const sqlite3 = **require**("sqlite3");

const isTest = process.argv.**slice**(-1)[0] === "--test";

const app = **express**();

const db = new sqlite3.**Database**("./database/educators\_db.db", (err) => {

    if (err) {

        console.**error**(err.message);

        return;

    }

    if (!isTest) {

        console.**log**("Successfully connected to the database...");

    }

});

function **isEducatorValid**(educator) {

    return educator && Number.**isInteger**(educator.id) && educator.id >= 0 && educator.surname !== undefined

    && educator.name !== undefined && !**isNaN**(**parseFloat**(educator.age)) &&

    educator.email !== undefined;

}

function **getObjectFromRow**(row) {

    return {

        id: row.Id,

        surname: row.Surname,

        name: row.Name,

        age: row.Age,

        email: row.Email

    };

}

app.**use**(express.**json**());

app

    .**route**("/educators")

    .**get**((request, response) => {

        const id = request.query.id;

        if (id) {

            db.**serialize**(function() {

                db.**get**("SELECT \* FROM Educators WHERE Id = ?;", id, (err, row) => {

                    if (err) {

                        response.**status**(500).**send**(err.message);

                        return;

                    }

                    if (row === undefined) {

                        response.**status**(404).**send**(`Educator with 'id' = ${id} not found.`);

                        return;

                    }

                    response.**status**(200).**json**({ educator: **getObjectFromRow**(row) });

                });

            });

        }

        else {

            db.**serialize**(function() {

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

                    if (err) {

                        response.**status**(500).**send**(err.message);

                        return;

                    }

                    let array = [];

                    rows.**forEach**((row) => array.**push**(**getObjectFromRow**(row)));

                    response.**status**(200).**json**({ educators: array });

                });

            });

        }

    })

    .**post**((request, response) => {

        const educator = request.body.educator;

        if (**isEducatorValid**(educator)) {

            db.**serialize**(function() {

                db.**get**("SELECT COUNT(\*) AS 'Count' FROM Educators WHERE Id = ?;", educator.id, (err, row) => {

                    if (err) {

                        response.**status**(500).**send**(err.message);

                        return;

                    }

                    if (row.Count != 0) {

                        response.**status**(400).**send**(`Educator with 'id' = ${educator.id} is already exists in the database.`);

                        return;

                    }

                    db.**run**("INSERT INTO Educators VALUES(?, ?, ?, ?, ?);", [ educator.id, educator.surname, educator.name, educator.age, educator.email ], (err) => {

                        if (err) {

                            response.**status**(500).**send**(err.message);

                            return;

                        }

                        response.**status**(200).**send**(`Educator with 'id' = ${educator.id} was successfully added to the database.`);

                    });

                });

            });

        }

        else {

            response.**status**(400).**send**("Bad request body: 'educator' not present correctly.");

        }

    })

    .**put**((request, response) => {

        const educator = request.body.educator;

        if (**isEducatorValid**(educator)) {

            db.**serialize**(function() {

                db.**get**("SELECT COUNT(\*) AS 'Count' FROM Educators WHERE Id = ?;", educator.id, (err, row) => {

                    if (err) {

                        response.**status**(500).**send**(err.message);

                        return;

                    }

                    if (row.Count == 0) {

                        response.**status**(404).**send**(`Educator with 'id' = ${educator.id} not found.`);

                        return;

                    }

                    db.**run**("UPDATE Educators SET Surname = ?, Name = ?, Age = ?, Email = ? WHERE Id = ?;",

                           [ educator.surname, educator.name, educator.age, educator.email, educator.id ], (err) => {

                            if (err) {

                                response.**status**(500).**send**(err.message);

                                return;

                            }

                            response.**status**(200).**send**(`Educator with 'id' = ${educator.id} was successfully updated.`);

                    });

                });

            });

        }

        else {

            response.**status**(400).**send**("Bad request body: 'educator' not present correctly.");

        }

    })

    .**delete**((request, response) => {

        const id = request.query.id;

        if (id) {

            db.**serialize**(function() {

                db.**get**("SELECT COUNT(\*) AS 'Count' FROM Educators WHERE Id = ?;", id, (err, row) => {

                    if (err) {

                        response.**status**(500).**send**(err.message);

                        return;

                    }

                    if (row.Count == 0) {

                        response.**status**(404).**send**(`Educator with 'id' = ${id} not found.`);

                        return;

                    }

                    db.**run**("DELETE FROM Educators WHERE Id = ?;", id, (err) => {

                        if (err) {

                            response.**status**(500).**send**(err.message);

                            return;

                        }

                        response.**status**(200).**send**(`Educator with 'id' = ${id} was successfully deleted.`);

                    });

                });

            });

        }

        else {

            response.**status**(400).**send**("Bad delete request: request must contains 'id' parameter.");

        }

    });

if (!isTest) {

    app.**listen**(3000, () => console.**log**("Server is listening on port 3000..."));

}

else {

    module.exports.app = app;

}

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

**Листинг программы:**

const express = **require**("express");

const { Sequelize, DataTypes } = **require**('sequelize');

const isTest = process.argv.**slice**(-1)[0] === "--test";

const app = **express**();

const sequelize = new **Sequelize**('sqlite:./database/educators\_db.db', { logging: !isTest });

const Educator = sequelize.**define**('Educator', {

    Id: {

        type: DataTypes.INTEGER,

        autoIncrement: true,

        primaryKey: true,

        allowNull: false

    },

    Surname: {

        type: DataTypes.STRING,

        allowNull: false

    },

    Name: {

        type: DataTypes.STRING,

        allowNull: false

    },

    Age: {

        type: DataTypes.INTEGER,

        allowNull: false

    },

    Email: {

        type: DataTypes.STRING,

        allowNull: false

    }

}, {

    timestamps: false

});

function **isEducatorValid**(educator) {

    return educator && Number.**isInteger**(educator.id) && educator.id >= 0 && educator.surname !== undefined

    && educator.name !== undefined && !**isNaN**(**parseFloat**(educator.age)) &&

    educator.email !== undefined;

}

function **getObjectFromModel**(model) {

    return {

        id: model.Id,

        surname: model.Surname,

        name: model.Name,

        age: model.Age,

        email: model.Email

    };

}

app.**use**(express.**json**());

app

    .**route**("/educators")

    .**get**((request, response) => {

        const id = request.query.id;

        if (id) {

            Educator.**findOne**({ where: { Id: id }})

            .**then**((model) => {

                if (model === null) {

                    response.**status**(404).**send**(`Educator with 'id' = ${id} not found.`);

                    return;

                }

                response.**status**(200).**json**({ educator: **getObjectFromModel**(model) });

            })

            .**catch**((err) => response.**status**(500).**send**(err.message));

        }

        else {

            Educator.**findAll**()

            .**then**((models) => {

                let array = [];

                models.**forEach**((model) => array.**push**(**getObjectFromModel**(model)));

                response.**status**(200).**json**({ educators: array });

            })

            .**catch**((err) => response.**status**(500).**send**(err.message));

        }

    })

    .**post**((request, response) => {

        const educator = request.body.educator;

        if (**isEducatorValid**(educator)) {

            Educator.**findOrCreate**({

                where: { Id: educator.id },

                defaults: {

                    Id: educator.id,

                    Surname: educator.surname,

                    Name: educator.name,

                    Age: educator.age,

                    Email: educator.email

                }

            })

            .**then**((result) => {

                if (result[1] === false) {

                    response.**status**(400).**send**(`Educator with 'id' = ${educator.id} already exists in the database.`);

                    return;

                }

                response.**status**(200).**send**(`Educator with 'id' = ${educator.id} was successfully added to the database.`);

            })

            .**catch**((err) => response.**status**(500).**send**(err.message));

        }

        else {

            response.**status**(400).**send**("Bad request body: 'educator' not present correctly.");

        }

    })

    .**put**((request, response) => {

        const educator = request.body.educator;

        if (**isEducatorValid**(educator)) {

            Educator.**update**({

                Surname: educator.surname,

                Name: educator.name,

                Age: educator.age,

                Email: educator.email

            }, {

                where: {

                    Id: educator.id

                }

            })

            .**then**(count => {

                if (count == 0) {

                    response.**status**(404).**send**(`Educator with 'id' = ${educator.id} not found.`);

                    return;

                }

                response.**status**(200).**send**(`Educator with 'id' = ${educator.id} was successfully updated.`);

            })

            .**catch**((err) => response.**status**(500).**send**(err.message));

        }

        else {

            response.**status**(400).**send**("Bad request body: 'educator' not present correctly.");

        }

    })

    .**delete**((request, response) => {

        const id = request.query.id;

        if (id) {

            Educator.**destroy**({ where: { Id: id } })

            .**then**(count => {

                if (count == 0) {

                    response.**status**(404).**send**(`Educator with 'id' = ${id} not found.`);

                    return;

                }

                response.**status**(200).**send**(`Educator with 'id' = ${id} was successfully deleted.`);

            })

            .**catch**((err) => response.**status**(500).**send**(err.message));

        }

        else {

            response.**status**(400).**send**("Bad delete request: request must contains 'id' parameter.");

        }

    });

sequelize.**sync**()

.**then**(() => {

    if (!isTest) {

        console.**log**("Successfully connected to the database...");

        app.**listen**(3000, () => console.**log**("Server is listening on port 3000..."));

    }

})

.**catch**((err) => console.**error**(err));

if (isTest) {

    module.exports.app = app;

}

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

const request = **require**("supertest");

const assert = **require**("assert");

const app = **require**("../lab5\_1/app").app;

const educator3 = {

    educator: {

        id: 3,

        surname: "Tarelko",

        name: "Igor",

        age: 51,

        email: "igorek\_the\_best@mail.ru"

    }

};

const educator3\_upd = {

    educator: {

        id: 3,

        surname: "Tarelko",

        name: "Lina",

        age: 53,

        email: "tarelko\_lina@mail.ru"

    }

};

**describe**("Testing API with SQL queries", () => {

**describe**("GET /educators", () => {

**it**("Should return 200 \"OK\" with JSON", (done) => {

**request**(app)

            .**get**("/educators")

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

            .**expect**(200)

            .**end**(done);

        });

    });

**describe**("GET /educators?id=3 NON-EXISTENT", () => {

**it**("Should return 404 \"Not Found\" with Text", (done) => {

**request**(app)

              .**get**("/educators")

              .**query**({id: 0})

              .**expect**("Content-Type", /text/)

              .**expect**(404)

              .**end**(done);

          });

    });

**describe**("POST /educators with id=3 NON-EXISTENT", () => {

**it**("Should return 200 \"OK\" with Text", (done) => {

**request**(app)

            .**post**("/educators")

            .**send**(educator3)

            .**expect**("Content-Type", /text/)

            .**expect**(200)

            .**end**(done);

        });

    });

**describe**("GET /educators?id=3 ADDED", () => {

**it**("Should return 200 \"OK\" with expected JSON", (done) => {

**request**(app)

            .**get**("/educators")

            .**query**({id: 3})

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

            .**expect**(200)

            .**expect**((response) => {

                assert.**deepStrictEqual**(response.body, educator3);

            })

            .**end**(done);

        });

    });

**describe**("POST /educators with id=3 EXISTENT", () => {

**it**("Should return 400 \"Bad Request\" with Text", (done) => {

**request**(app)

            .**post**("/educators")

            .**send**(educator3)

            .**expect**("Content-Type", /text/)

            .**expect**(400)

            .**end**(done);

        });

    });

**describe**("PUT /educators with id=3 EXISTENT", () => {

**it**("Should return 200 \"OK\" with Text", (done) => {

**request**(app)

            .**put**("/educators")

            .**send**(educator3\_upd)

            .**expect**("Content-Type", /text/)

            .**expect**(200)

            .**end**(done);

        });

    });

**describe**("GET /educators?id=3 UPDATED", () => {

**it**("Should return 200 \"OK\" with expected JSON", (done) => {

**request**(app)

            .**get**("/educators")

            .**query**({id: 3})

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

            .**expect**(200)

            .**expect**((response) => {

                assert.**deepStrictEqual**(response.body, educator3\_upd);

            })

            .**end**(done);

        });

    });

**describe**("DELETE /educators with id=3 EXISTENT", () => {

**it**("Should return 200 \"OK\" with Text", (done) => {

**request**(app)

            .**delete**("/educators")

            .**query**({id: 3})

            .**expect**("Content-Type", /text/)

            .**expect**(200)

            .**end**(done);

        });

    });

**describe**("GET /educators?id=3 DELETED", () => {

**it**("Should return 404 \"Not Found\" with Text", (done) => {

**request**(app)

              .**get**("/educators")

              .**query**({id: 0})

              .**expect**("Content-Type", /text/)

              .**expect**(404)

              .**end**(done);

          });

    });

**describe**("PUT /educators with id=3 DELETED", () => {

**it**("Should return 404 \"Not Found\" with Text", (done) => {

**request**(app)

            .**put**("/educators")

            .**send**(educator3\_upd)

            .**expect**("Content-Type", /text/)

            .**expect**(404)

            .**end**(done);

        });

    });

**describe**("DELETE /educators with id=3 DELETED", () => {

**it**("Should return 404 \"Not Found\" with Text", (done) => {

**request**(app)

            .**delete**("/educators")

            .**query**({id: 3})

            .**expect**("Content-Type", /text/)

            .**expect**(404)

            .**end**(done);

        });

    });

});

const request = **require**("supertest");

const assert = **require**("assert");

const app = **require**("../lab5\_2/app").app;

const educator3 = {

    educator: {

        id: 3,

        surname: "Tarelko",

        name: "Igor",

        age: 51,

        email: "igorek\_the\_best@mail.ru"

    }

};

const educator3\_upd = {

    educator: {

        id: 3,

        surname: "Tarelko",

        name: "Lina",

        age: 53,

        email: "tarelko\_lina@mail.ru"

    }

};

**describe**("Testing API with ORM Sequelize", () => {

**describe**("GET /educators", () => {

**it**("Should return 200 \"OK\" with JSON", (done) => {

**request**(app)

            .**get**("/educators")

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

            .**expect**(200)

            .**end**(done);

        });

    });

**describe**("GET /educators?id=3 NON-EXISTENT", () => {

**it**("Should return 404 \"Not Found\" with Text", (done) => {

**request**(app)

              .**get**("/educators")

              .**query**({id: 0})

              .**expect**("Content-Type", /text/)

              .**expect**(404)

              .**end**(done);

          });

    });

**describe**("POST /educators with id=3 NON-EXISTENT", () => {

**it**("Should return 200 \"OK\" with Text", (done) => {

**request**(app)

            .**post**("/educators")

            .**send**(educator3)

            .**expect**("Content-Type", /text/)

            .**expect**(200)

            .**end**(done);

        });

    });

**describe**("GET /educators?id=3 ADDED", () => {

**it**("Should return 200 \"OK\" with expected JSON", (done) => {

**request**(app)

            .**get**("/educators")

            .**query**({id: 3})

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

            .**expect**(200)

            .**expect**((response) => {

                assert.**deepStrictEqual**(response.body, educator3);

            })

            .**end**(done);

        });

    });

**describe**("POST /educators with id=3 EXISTENT", () => {

**it**("Should return 400 \"Bad Request\" with Text", (done) => {

**request**(app)

            .**post**("/educators")

            .**send**(educator3)

            .**expect**("Content-Type", /text/)

            .**expect**(400)

            .**end**(done);

        });

    });

**describe**("PUT /educators with id=3 EXISTENT", () => {

**it**("Should return 200 \"OK\" with Text", (done) => {

**request**(app)

            .**put**("/educators")

            .**send**(educator3\_upd)

            .**expect**("Content-Type", /text/)

            .**expect**(200)

            .**end**(done);

        });

    });

**describe**("GET /educators?id=3 UPDATED", () => {

**it**("Should return 200 \"OK\" with expected JSON", (done) => {

**request**(app)

            .**get**("/educators")

            .**query**({id: 3})

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

            .**expect**(200)

            .**expect**((response) => {

                assert.**deepStrictEqual**(response.body, educator3\_upd);

            })

            .**end**(done);

        });

    });

**describe**("DELETE /educators with id=3 EXISTENT", () => {

**it**("Should return 200 \"OK\" with Text", (done) => {

**request**(app)

            .**delete**("/educators")

            .**query**({id: 3})

            .**expect**("Content-Type", /text/)

            .**expect**(200)

            .**end**(done);

        });

    });

**describe**("GET /educators?id=3 DELETED", () => {

**it**("Should return 404 \"Not Found\" with Text", (done) => {

**request**(app)

              .**get**("/educators")

              .**query**({id: 0})

              .**expect**("Content-Type", /text/)

              .**expect**(404)

              .**end**(done);

          });

    });

**describe**("PUT /educators with id=3 DELETED", () => {

**it**("Should return 404 \"Not Found\" with Text", (done) => {

**request**(app)

            .**put**("/educators")

            .**send**(educator3\_upd)

            .**expect**("Content-Type", /text/)

            .**expect**(404)

            .**end**(done);

        });

    });

**describe**("DELETE /educators with id=3 DELETED", () => {

**it**("Should return 404 \"Not Found\" with Text", (done) => {

**request**(app)

            .**delete**("/educators")

            .**query**({id: 3})

            .**expect**("Content-Type", /text/)

            .**expect**(404)

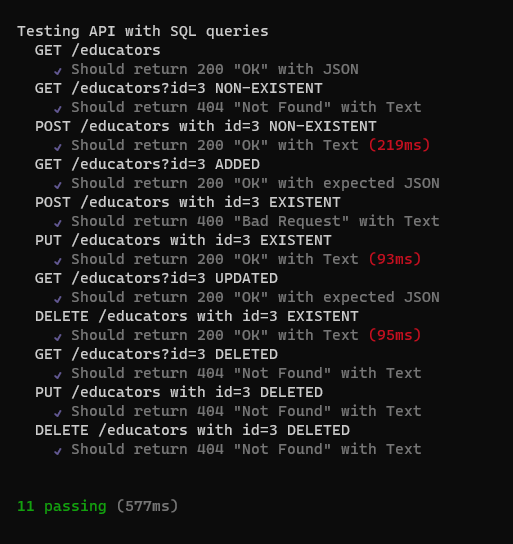
            .**end**(done);

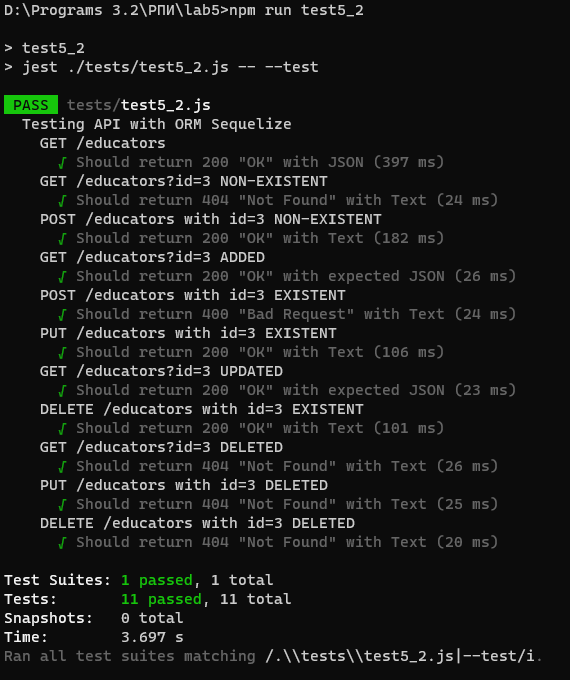
        });

    });

});

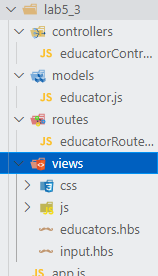
**Результаты тестов:**

****

****

**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.

**Структура проекта:**

****

**app.js**

const express = **require**("express");

const { Sequelize } = **require**('sequelize');

const app = **express**();

const sequelize = new **Sequelize**('sqlite:../database/educators\_db.db');

module.exports.sequelize = sequelize;

const { educatorRouter } = **require**("./routes/educatorRouter");

app.**set**("view engine", "hbs");

app.**set**('views', "./views");

app.**use**("/css", express.**static**(\_\_dirname + "/views/css"));

app.**use**("/js", express.**static**(\_\_dirname + "/views/js"));

app.**use**(express.**urlencoded**({ extended: false }));

app.**use**("/educators", educatorRouter);

app.**use**((\_, response) => {

    response.**status**(404).**send**("Page Not Found");

});

sequelize.**sync**()

.**then**(() => {

    console.**log**("Successfully connected to the database...");

    app.**listen**(3000, () => console.**log**("Server is listening on port 3000..."));

})

.**catch**((err) => console.**error**(err));

**models/educator.js**

const { DataTypes } = **require**('sequelize');

const { sequelize } = **require**("../app");

const Educator = sequelize.**define**('Educator', {

    Id: {

        type: DataTypes.INTEGER,

        autoIncrement: true,

        primaryKey: true,

        allowNull: false

    },

    Surname: {

        type: DataTypes.STRING,

        allowNull: false

    },

    Name: {

        type: DataTypes.STRING,

        allowNull: false

    },

    Age: {

        type: DataTypes.INTEGER,

        allowNull: false

    },

    Email: {

        type: DataTypes.STRING,

        allowNull: false

    }

}, {

    timestamps: false

});

module.exports.Educator = Educator;

**routes/educatorRouter**

const express = **require**("express");

const educatorController = **require**("../controllers/educatorController");

const educatorRouter = express.**Router**();

educatorRouter.**get**("/", educatorController.getAllEducators);

educatorRouter.**get**("/input", educatorController.getEducatorInput);

educatorRouter.**post**("/create", educatorController.createEducator);

educatorRouter.**post**("/update", educatorController.updateEducator);

educatorRouter.**post**("/delete", educatorController.deleteEducator);

module.exports.educatorRouter = educatorRouter;

**controllers/educatorController**

const { is } = **require**("express/lib/request");

const { Educator } = **require**("../models/educator");

function **isEducatorBodyValid**(body) {

    let id = **Number**(body.id);

    let age = **Number**(body.age);

    console.**error**(id)

    console.**error**(Number.**isInteger**(id))

    console.**error**(**isNaN**(age))

    body.id = id;

    body.age = age;

    return body && Number.**isInteger**(body.id) && body.id >= 0 && body.surname !== undefined

        && body.name !== undefined && !**isNaN**(**parseFloat**(body.age)) &&

        body.email !== undefined;

}

function **getObjectFromModel**(model) {

    return {

        id: model.Id,

        surname: model.Surname,

        name: model.Name,

        age: model.Age,

        email: model.Email

    };

}

module.exports.**getAllEducators** = (\_, response) => {

    Educator.**findAll**()

    .**then**((models) => {

        let array = [];

        models.**forEach**((model) => array.**push**(**getObjectFromModel**(model)));

        response.**render**("../views/educators.hbs", {

            educators: array

        });

    })

    .**catch**((err) => response.**status**(500).**send**(err));

};

module.exports.**getEducatorInput** = (request, response) => {

    const action = request.query.action;

    if (action && (action === "create" || action === "update")) {

        if (action === "create") {

            response.**render**("input.hbs", {

                path: "create"

            });

        }

        else if (request.query.id) {

            Educator.**findOne**({ where: { Id: request.query.id }})

            .**then**((model) => {

                if (model === null) {

                    response.**status**(404).**send**(`Educator with 'id' = ${id} not found.`);

                    return;

                }

                response.**render**("input.hbs", {

                    path: "update",

                    educator: **getObjectFromModel**(model)

                });

            })

            .**catch**((err) => response.**status**(500).**send**(err.message));

        }

        else {

            response.**status**(400).**send**("Bad request: must contains 'id' parameter.");

        }

    }

    else {

        response.**status**(400).**send**("Bad request: must contains 'action' parameter.");

    }

};

module.exports.**createEducator** = (request, response) => {

    const body = request.body;

    if (**isEducatorBodyValid**(body)) {

        Educator.**findOrCreate**({

            where: { Id: body.id },

            defaults: {

                Id: body.id,

                Surname: body.surname,

                Name: body.name,

                Age: body.age,

                Email: body.email

            }

        })

        .**then**((result) => {

            if (result[1] === false) {

                response.**status**(400).**send**(`Educator with 'id' = ${body.id} already exists in the database.`);

                return;

            }

            response.**redirect**("/educators");

        })

        .**catch**((err) => response.**status**(500).**send**(err.message));

    }

    else {

        response.**status**(400).**send**("Bad request body: parameters not present correctly.");

    }

};

module.exports.**updateEducator** = (request, response) => {

    const body = request.body;

    if (**isEducatorBodyValid**(body)) {

        Educator.**update**({

            Surname: body.surname,

            Name: body.name,

            Age: body.age,

            Email: body.email

        }, {

            where: {

                Id: body.id

            }

        })

        .**then**(count => {

            if (count == 0) {

                response.**status**(404).**send**(`Educator with 'id' = ${body.id} not found.`);

                return;

            }

            response.**redirect**("/educators");

        })

        .**catch**((err) => response.**status**(500).**send**(err.message));

    }

    else {

        response.**status**(400).**send**("Bad request body: parameters not present correctly.");

    }

};

module.exports.**deleteEducator** = (request, response) => {

    const id = request.body.id;

    if (id) {

        Educator.**destroy**({ where: { Id: id } })

        .**then**(count => {

            if (count == 0) {

                response.**status**(404).**send**(`Educator with 'id' = ${id} not found.`);

                return;

            }

            response.**redirect**("/educators");

        })

        .**catch**((err) => response.**status**(500).**send**(err.message));

    }

    else {

        response.**status**(400).**send**("Bad request body: must contains 'id' parameter.");

    }

};

**views/input.hbs**

<html lang="en">

<head>

    <title>Ввод воспитателя</title>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="/css/input\_styles.css">

</head>

<body>

    <h2>Ввод воспитателя</h2>

    <form action="/educators/{{path}}" method="POST">

        <label>ID:</label><br>

        <input type="text" name="id" value="{{educator.id}}"><br><br>

        <label>Фамилия:</label><br>

        <input type="text" name="surname" value="{{educator.surname}}"><br><br>

        <label>Имя:</label><br>

        <input type="text" name="name" value="{{educator.name}}"><br><br>

        <label>Возраст:</label><br>

        <input type="text" name="age" value="{{educator.age}}"><br><br>

        <label>Email:</label><br>

        <input type="text" name="email" value="{{educator.email}}"><br><br>

        <input type="submit" value="OK">

    </form>

</body>

</html>

**views/educators.hbs**

<!DOCTYPE html>

<html>

<head>

    <title>Воспитатели</title>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" href="/css/educators\_styles.css">

    <script type="text/javascript" src="/js/jquery-3.6.0.js"></script>

</head>

<body>

    <h2>Таблица воспитателей</h2>

    <table>

        <tr>

            <th>ID</th>

            <th>Фамилия</th>

            <th>Имя</th>

            <th>Возраст</th>

            <th>Email</th>

        </tr>

        {{#each educators}}

            <tr>

                <td>{{this.id}}</td>

                <td>{{this.surname}}</td>

                <td>{{this.name}}</td>

                <td>{{this.age}}</td>

                <td>{{this.email}}</td>

            </tr>

        {{/each}}

    </table>

    <div style="margin-top: 10px;">

        <form id="inputForm" action="/educators/input" mathod="GET">

            <input type="hidden" id="inputFormAction" name="action">

            <input type="hidden" id="inputIdField" name="id">

            <input type="submit" id="createButton" value="Добавить воспитателя">

            <input type="submit" id="updateButton" value="Изменить воспитателя">

        </form>

        <form id="deleteForm" action="/educators/delete" method="POST">

            <input type="hidden" id="deleteIdField" name="id">

            <input type="submit" id="delete" value="Удалить воспитателя">

        </form>

    </div>

    <script>

        let lastSelected = -1;

**$**("tr").**click**((e) => {

            let nextSelected = **$**("tr").**index**(e.currentTarget);

            if (lastSelected != -1 && nextSelected != 0) {

**$**(**$**("tr")[lastSelected]).**css**("background-color", "");

            }

            if (nextSelected != 0) {

**$**(e.currentTarget).**css**("background-color", "limegreen");

                lastSelected = nextSelected;

            }

        });

**$**("#inputForm").**submit**((e) => {

            if (**$**(e.originalEvent.submitter).**is**(**$**("#createButton"))) {

**$**("#inputFormAction").**val**("create");

            }

            else {

                if (lastSelected == -1) {

                    e.**preventDefault**();

                }

                else {

**$**("#inputFormAction").**val**("update");

**$**("#inputIdField").**val**(**$**(**$**("tr")[lastSelected]).**children**().**first**().**text**());

                }

            }

        });

**$**("#deleteForm").**submit**((e) => {

            if (lastSelected == -1) {

                e.**preventDefault**();

            }

            else {

**$**("#deleteIdField").**val**(**$**(**$**("tr")[lastSelected]).**children**().**first**().**text**());

            }

        });

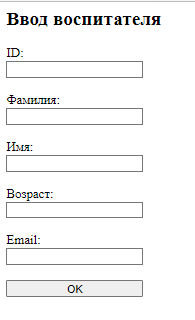
    </script>

</body>

<html>

**Результат выполнения:**

****

****

**+**

**Вывод:** в процессе выполнения лабораторной работы изучены основы работы по написанию скриптов для работы с базами данных в Node.js, получены навыки работы с orm в Node.js. Научился создавать и тестировать простые приложения.