САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа №2: Создание своего сервера

Выполнил:
Жаров Александр
К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

1) Придумать собственную модель пользователя
2) Реализовать набор из CRUD методов для работы с пользователем
3) Написать запрос для получения пользователя по id или email

Ход работы

1) Устанавливаем необходимые зависимости

```
{
  "name": "hw2",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.3",
    "mysql2": "^3.9.2",
    "sequelize": "^6.37.1",
    "sqlite3": "^5.1.7"
  }
}
```

2) Инициализируем server на express и подключаемся к бд

```
const express = require("express");
const cors = require("cors");  4.5k (gzipped: 1.9k)

const app = express();

const db = require("./app/models");

db.sequelize
  .sync()
  .then(() => {
    console.log("Synced db.");
  })
  .catch((err) => {
    console.log("Failed to sync db: " + err.message);
  });

var corsOptions = {
  origin: "http://localhost:8081",
};

app.use(cors(corsOptions));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.get("/", (req, res) => {
  res.json({ message: "Welcome to application." });
});

require("./app/routes/user.routes")(app);

const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

3) Создаем конфигурационный файл для бд на SQLite

config > db.config.js > ...

```
module.exports = {
  HOST: "localhost",
  USER: "sasha",
  PASSWORD: "1234",
  DB: "mydb",
  dialect: "sqlite",
  storage: "./db/database.sqlite",
};

```

4) Создаем модель пользователя и передаем ее в бд

```
module.exports = (sequelize, Sequelize) => {
  const User = sequelize.define("user", {
    name: {
      type: Sequelize.STRING,
      allowNull: false,
    },
    age: {
      type: Sequelize.INTEGER,
      allowNull: false,
    },
    email: {
      type: Sequelize.STRING,
      allowNull: false,
      unique: true,
    },
    password: {
      type: Sequelize.STRING,
      allowNull: false,
    },
  });

  return User;
};
```

```
const Sequelize = require("sequelize");
const dbConfig = require("../../config/db.config.js");

const sequelize = new Sequelize({
  dialect: dbConfig.dialect,
  storage: dbConfig.storage,
});

const db = {};

db.Sequelize = Sequelize;
db.sequelize = sequelize;

db.users = require("./user.model.js")(sequelize, Sequelize);

module.exports = db;
```

5) Создаем CRUD операции для взаимодействия с бд

```
exports.create = (req, res) => {
  if (
    !req.body.name ||
    !req.body.age ||
    !req.body.email ||
    !req.body.password
  ) {
    res.status(400).send({
      message: "All fields are required!",
    });
    return;
  }

  const user = {
    name: req.body.name,
    age: req.body.age,
    email: req.body.email,
    password: req.body.password,
  };

  User.create(user)
    .then((data) => {
      res.send(data);
    })
    .catch((err) => {
      res.status(500).send({
        message: err.message || "Some error occurred while creating the User.",
      });
    });
};
```

```
exports.findAll = (req, res) => {
  User.findAll()
    .then((data) => {
      res.send(data);
    })
    .catch((err) => {
      res.status(500).send({
        message: err.message || "Some error occurred while retrieving users.",
      });
    });
};
```

```
exports.findOne = (req, res) => {
  const id = req.params.id;

  User.findByPk(id)
    .then((data) => {
      if (data) {
        res.send(data);
      } else {
        res.status(404).send({
          message: `User with id=${id} not found.`,
        });
      }
    })
    .catch((err) => {
      res.status(500).send({
        message: "Error retrieving user with id=" + id,
      });
    });
};
```

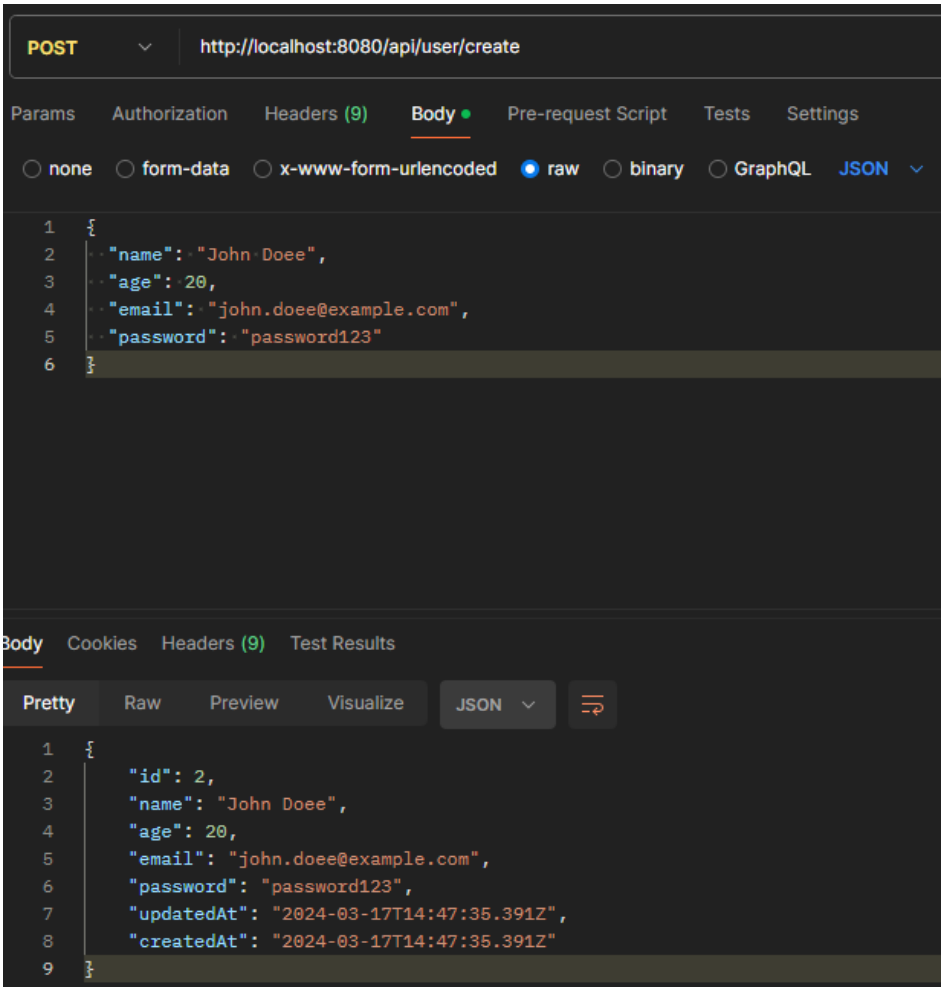
```
exports.update = (req, res) => {
  const id = req.params.id;

  User.update(req.body, {
    where: { id: id },
  })
    .then((num) => {
      if (num == 1) {
        res.send({
          message: "User was updated successfully.",
        });
      } else {
        res.send({
          message: `Cannot update user with id=${id}. Maybe user was not found or req.body is empty!`,
        });
      }
    })
    .catch((err) => {
      res.status(500).send({
        message: "Error updating user with id=" + id,
      });
    });
};
```

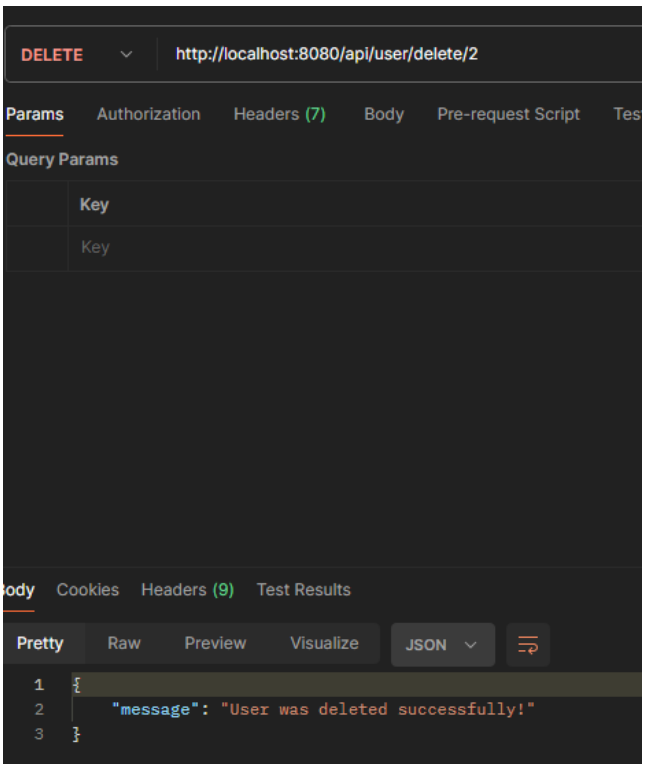
```
exports.delete = (req, res) => {
  const id = req.params.id;

  User.destroy({
    where: { id: id },
  })
    .then((num) => {
      if (num == 1) {
        res.send({
          message: "User was deleted successfully!",
        });
      } else {
        res.send({
          message: `Cannot delete user with id=${id}. Maybe user was not found!`,
        });
      }
    })
    .catch((err) => {
      res.status(500).send({
        message: "Could not delete user with id=" + id,
      });
    });
};
```

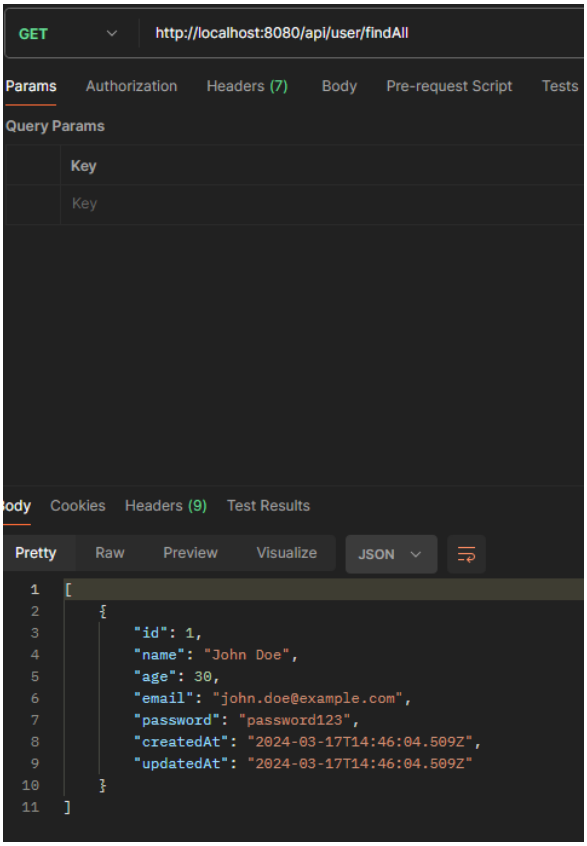
6) Создаем модуль с роутами

```
module.exports = (app) => {
  const users = require("../controllers/user.controller.js");

  var router = require("express").Router();

  router.post("/create", users.create);
  router.get("/findAll", users.findAll);
  router.get("/find/:id", users.findOne);
  router.put("/update/:id", users.update);
  router.delete("/delete/:id", users.delete);

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

```

7) Проверяем работу в postman







Вывод

В ходе работы я написал собственный сервер, создал базу данных, где хранятся пользователи и написал апи для этой бд.