

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа № 2
“Знакомство с ORM Sequelize”

Выполнил:

Чан Дык Минь

К33392

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

1. Продумать свою собственную модель пользователя
2. Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize
3. Написать запрос для получения пользователя по id/email

Ход работы

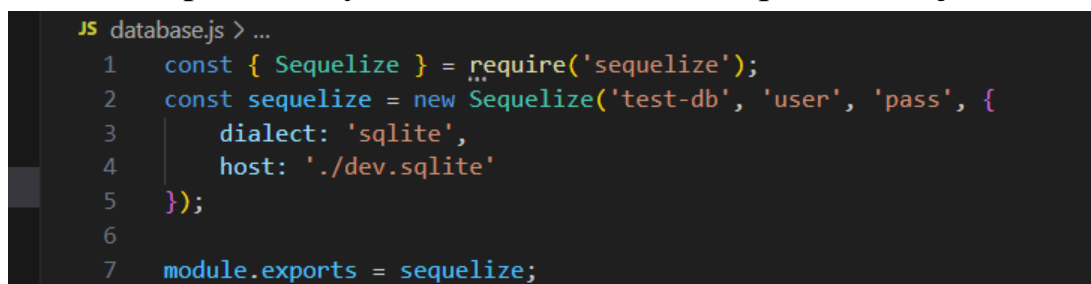
1. Скачать необходимые зависимости:

```
npm install express
npm install --save sequelize
npm install sqlite3
```



```
{
  "name": "hw2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "serve": "nodemon server.js"
  },
  "author": "Minh",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "express": "^4.18.3",
    "path": "^0.12.7",
    "sequelize": "^6.37.1",
    "sqlite3": "^5.1.7"
  },
  "devDependencies": {
    "nodemon": "^3.1.0"
  }
}
```

2. Настроить базу данных с именем dev.sqlite в data.js



```
const { Sequelize } = require('sequelize');
const sequelize = new Sequelize('test-db', 'user', 'pass', {
  dialect: 'sqlite',
  host: './dev.sqlite'
});
module.exports = sequelize;
```

3. Инициализировать базу данных и запустить сервер

```
JS server.js > app
1  const express = require('express');
2  const sequelize = require('./database');
3
4  sequelize.sync().then(() => console.log('db is ready'));
5
6  const app = express()
7
8  app.listen(3000, () => {
9    console.log("Server running")
10  })
```

4. Создать таблицу

```
JS User.js > ...
1  const { Model, DataTypes } = require('sequelize');
2  const sequelize = require('./database');
3
4  class User extends Model {}
5
6  User.init({
7    username: {
8      type: DataTypes.STRING
9    },
10   email: {
11     type: DataTypes.STRING
12   },
13   password: {
14     type: DataTypes.STRING
15   }
16 }, {
17   sequelize,
18   modelName: 'user'
19 })
20
21 module.exports = User;
```

Создать таблицу пользователей с 3 полями: имя пользователя, адрес электронной почты, пароль.

5. Создать методы запроса для таблицы User.

```

exports.findAll = async (req, res) => {
  const users = await User.findAll();
  return res.status(200).send(users.map(u => u.toJSON()));
}

exports.findById = async (req, res) => {
  const user = await User.findByPk(req.params.id);

  if (user) {
    return res.send(user.toJSON());
  }
  return res.status(404).send("User could not be found");
}

exports.findByEmail = async (req, res) => {
  const user = await User.findOne({
    where: {
      email: req.params.email
    }
  });

  if (user) {
    return res.send(user.toJSON());
  }
  return res.status(404).send("User could not be found");
}

```

Я создал 3 запроса для метода GET: `findAll` вернет всех пользователей, `findById` вернет пользователей по соответствующему идентификатору, `findByEmail` вернет пользователей по соответствующему адресу электронной почты.

```

exports.deleteById = async (req, res) => {
  await User.destroy({
    where: {
      id: req.params.id
    }
  });
  res.send("User has been deleted successfully!")
}

```

`deleteById` удалит пользователя по соответствующему Id

```

exports.create = async (req, res) => {
  try {
    const user = await User.create(req.body);
    return res.status(201).send(user.toJSON());
  } catch (e) {
    return res.status(400).send(e.errors.map(err => err.message));
  }
}

```

`create` создает нового пользователя в таблице

```

exports.update = async (req, res) => {
  const userId = req.params.id;
  const payload = req.body;

  const user = await User.findByPk(userId);
  if (!user) {
    return res.status(404).send("User could not be found");
  }

  try {
    const updatedUser = await user.update(payload);
    return res.send(updatedUser.toJSON());
  } catch (e) {
    return res.status(400).send(e.errors.map(err => err.message));
  }
}

```

update создаст нового пользователя в соответствии с идентификатором в таблице.

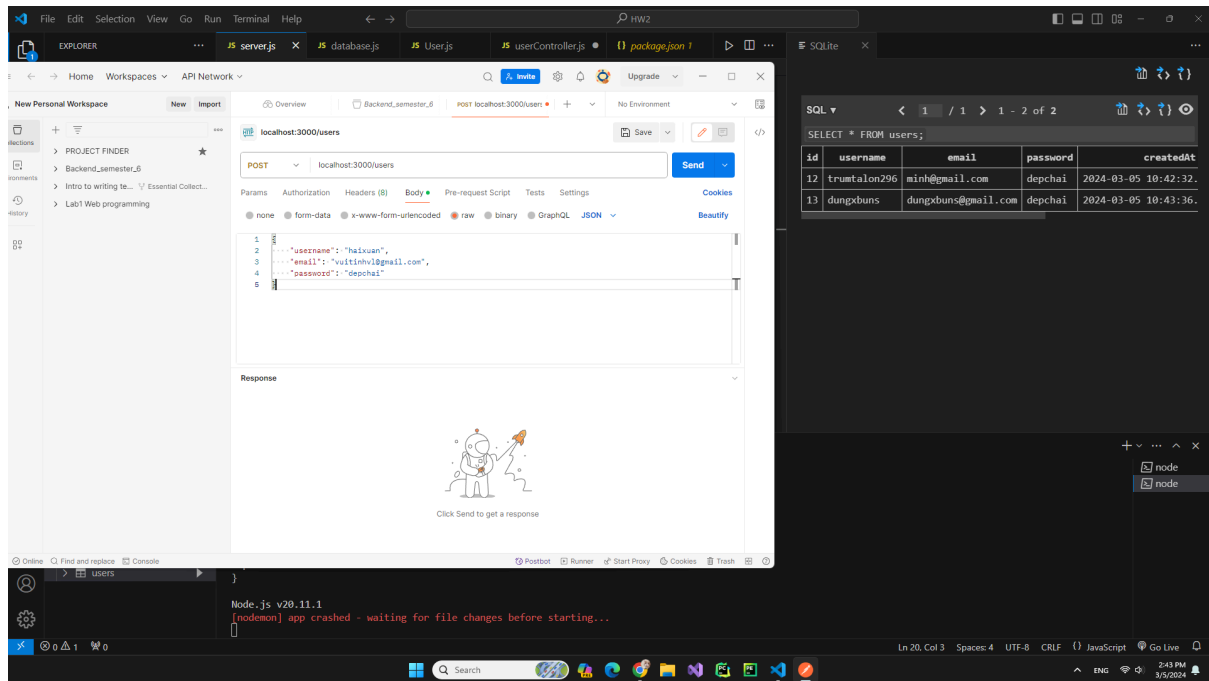
6. Создать конечные точки

```

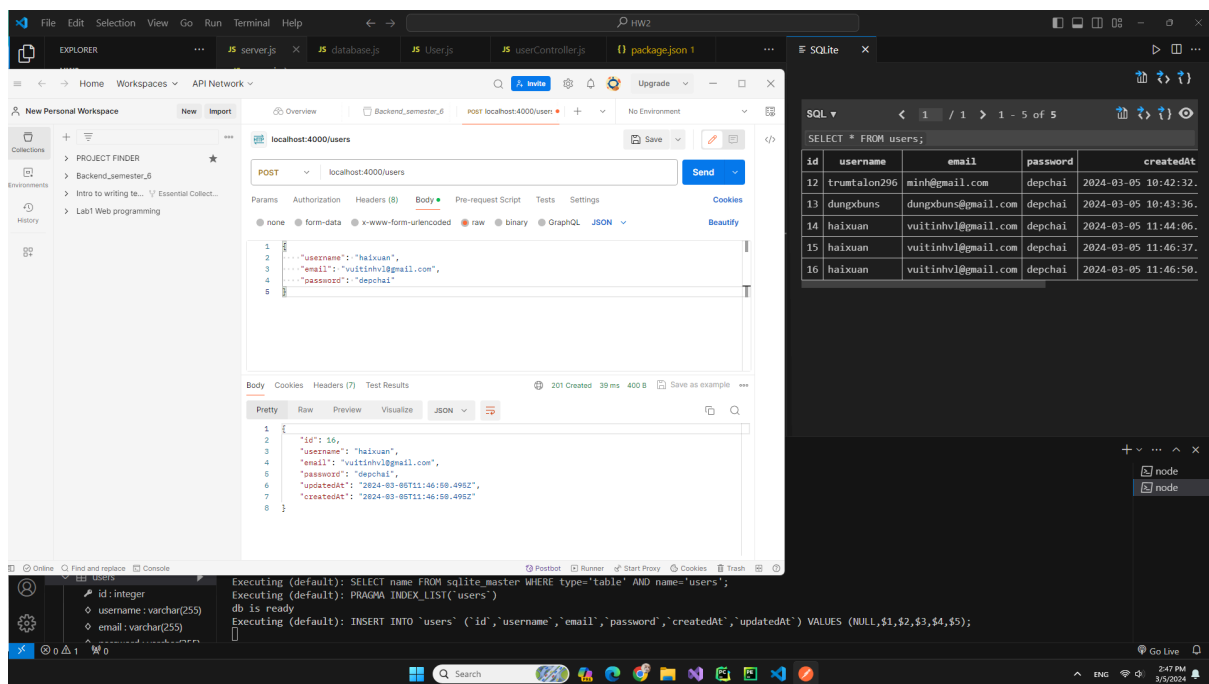
JS server.js > ...
1  const express = require('express');
2  const sequelize = require('./database');
3  const userController = require('./controllers/userController');
4
5  sequelize.sync().then(() => console.log('db is ready'));
6
7  const app = express()
8  app.use(express.json())
9  app.use(express.urlencoded({extended: false}))
10
11 app.get("/users", userController.findAll);
12 app.get("/users/:id", userController.findById);
13 app.get("/users/email/:email", userController.findByEmail);
14 app.delete("/users/:id", userController.deleteById);
15 app.post("/users", userController.create);
16 app.patch("/users/:id", userController.update);
17
18 app.listen(3000, () => {
19   console.log("Server running")
20 })

```

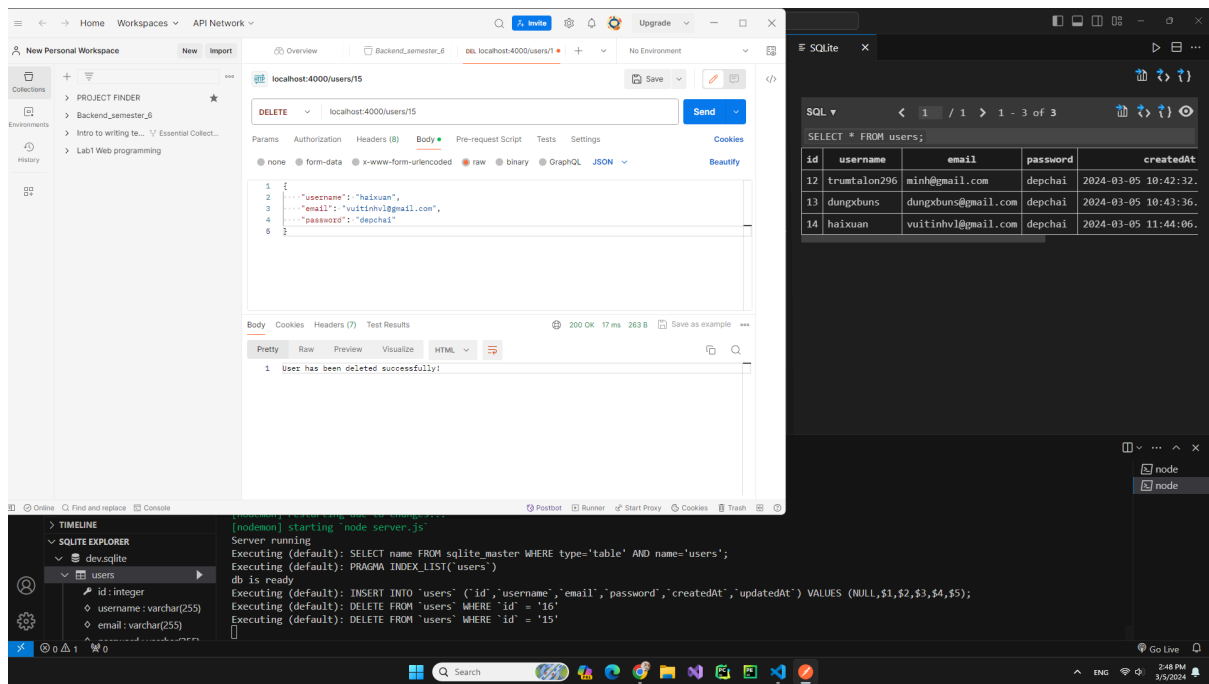
7. Тестовые запросы с помощью приложения postman



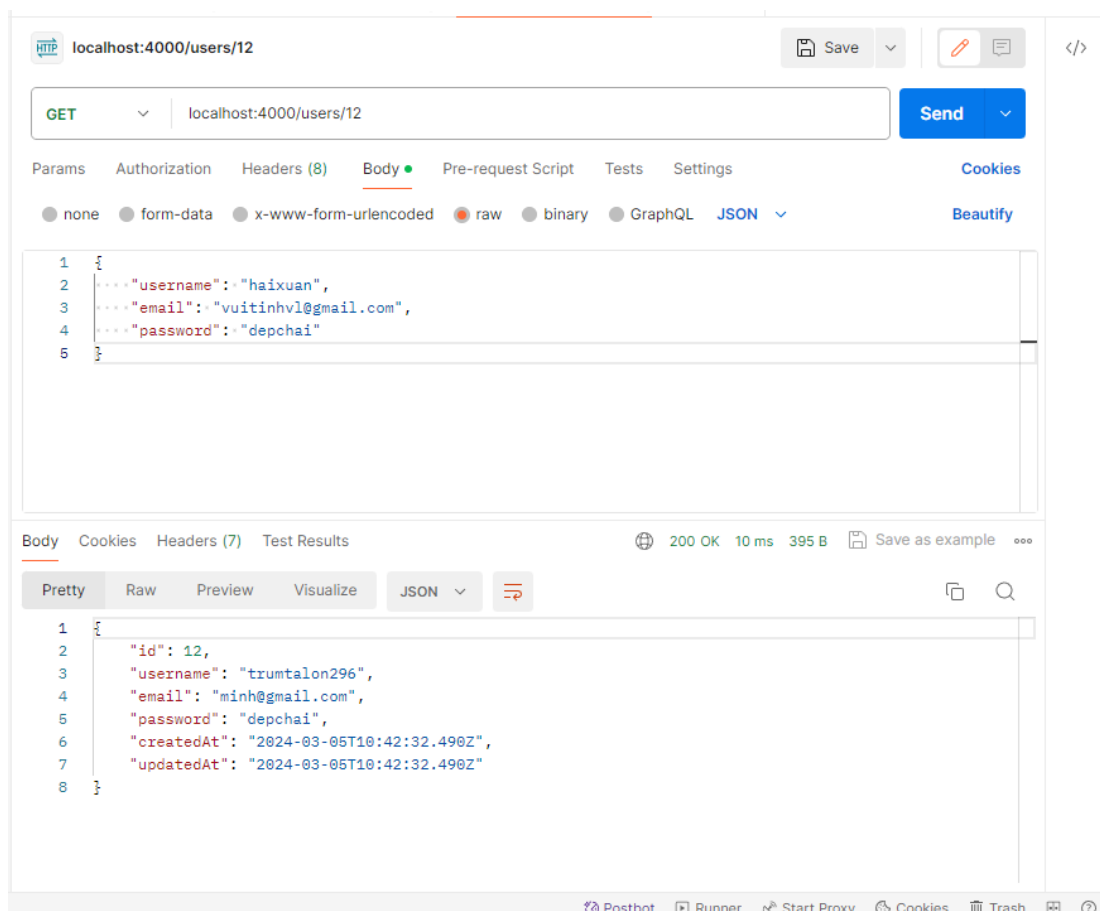
перед созданием имени пользователя хuanhai



После создания имени пользователя хuan два-три раза



После удаления имени пользователя xuanhai с идентификатором 15, 16



Поиск пользователя с идентификатором 12

The screenshot displays a REST client interface. At the top, a GET request is configured to `localhost:4000/users/email/dungxbuns@gmail.com`. The 'Body' tab is selected, showing a JSON payload: `{ "username": "haixuan", "email": "vuitinhvl@gmail.com", "password": "depchai" }`. Below the request, the response is shown with a status of 200 OK, 7 ms latency, and 397 B size. The response body is formatted as JSON: `{ "id": 13, "username": "dungxbuns", "email": "dungxbuns@gmail.com", "password": "depchai", "createdAt": "2024-03-05T10:43:36.597Z", "updatedAt": "2024-03-05T10:45:00.103Z" }`.

Поиск пользователей по электронной почте `dungxbuns@gmail.com`

Вывод

Во время домашней работы я научился использовать Express с Sequelize вместе с sqlite. Я уже знаю, как писать API, создавать простую базу данных и создавать конечные точки.