САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа № 2
“Знакомство с ORM Sequelize”

Выполнил:

Коротин А.М.

K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 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

Продумать свою собственную модель пользователя

Была спроектирована модель пользователя, содержащая следующие атрибуты и ограничения:

— firstName: string, not null;

— lastName: string, not null;

— email: string, not null, email format, unique;

— role: string, default “USER”.

Фрагмент кода сгенерированной модели пользователя будет приведен ниже на рисунке 1.

```
User.init( attributes: {
  firstName: {
    type: DataTypes.STRING,
    allowNull: false
  },
  lastName: {
    type: DataTypes.STRING,
    allowNull: false
  },
  email:{
    type: DataTypes.STRING,
    allowNull: false,
    unique: true,
    validate: {
      isEmail: true
    }
  },
  role: {
    type: DataTypes.STRING,
    default: "USER"
  }
```

Рисунок 1 — Фрагмент сгенерированной модели пользователя

Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize

Для реализации CRUD-методов для работы с пользователями будут созданы следующие конечные точки API:

— GET /users — получение всех пользователей;

— GET /users/:id — получение пользователя по ID;

— POST /users — создание пользователя;

— PATCH /users/:id — частичное изменение пользователя;

— DELETE /users/:id — удаление пользователя.

Для выполнения поставленной задачи был создан отдельный модуль userController, содержащий логику выполнения спроектированных методов. Фрагмент содержимого модуля будет приведен на рисунке 2.

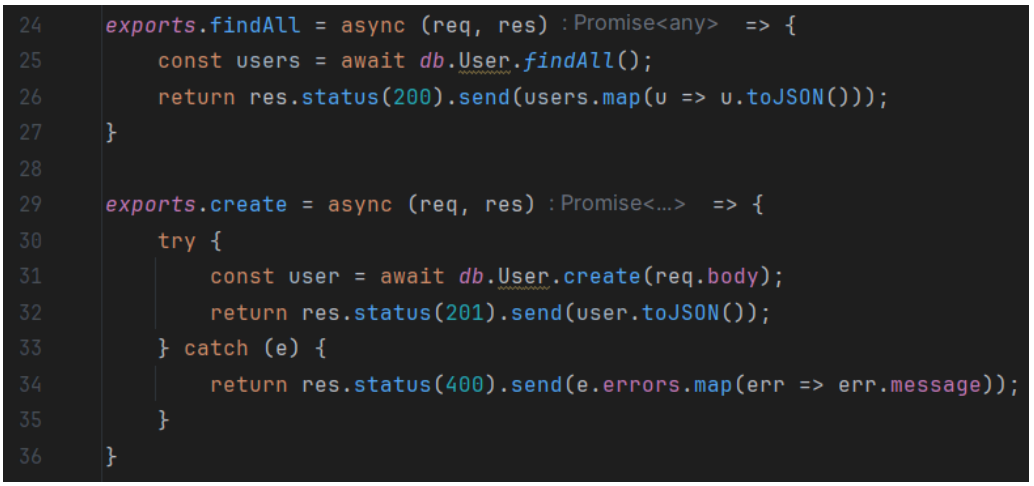

Рисунок 2 — Фрагмент содержимого модуля userController

Далее, для сопоставления входящих HTTP запросов и соответствующей им логике обработки, использовался функционал фреймворка express.js. Было создано приложение, прослушивающее запросы на определенных конечных точках. Для парсинга тела запроса использовался модуль body-parser. Фрагмент кода созданного приложения (из файла index.js) будет приведен ниже на рисунке 3.

```
const app : any | Express  = express();
app.use(bodyParser.json());
const port : number  = 3000;

app.get("/users", userController.findAll);
app.get("/users/:id", userController.findById);
app.delete( path: "/users/:id", userController.deleteById);
app.post( path: "/users", userController.create);
app.patch( path: "/users/:id", userController.update);
```

Рисунок 3 — Фрагмент кода приложения express.js

Для проверки работоспособности реализованного API выполним несколько запросов, используя среду Postman. Произведем создание пользователя, получение по id и удаление.

Результат выполнения запроса на создания пользователя приведен на рисунке 4.

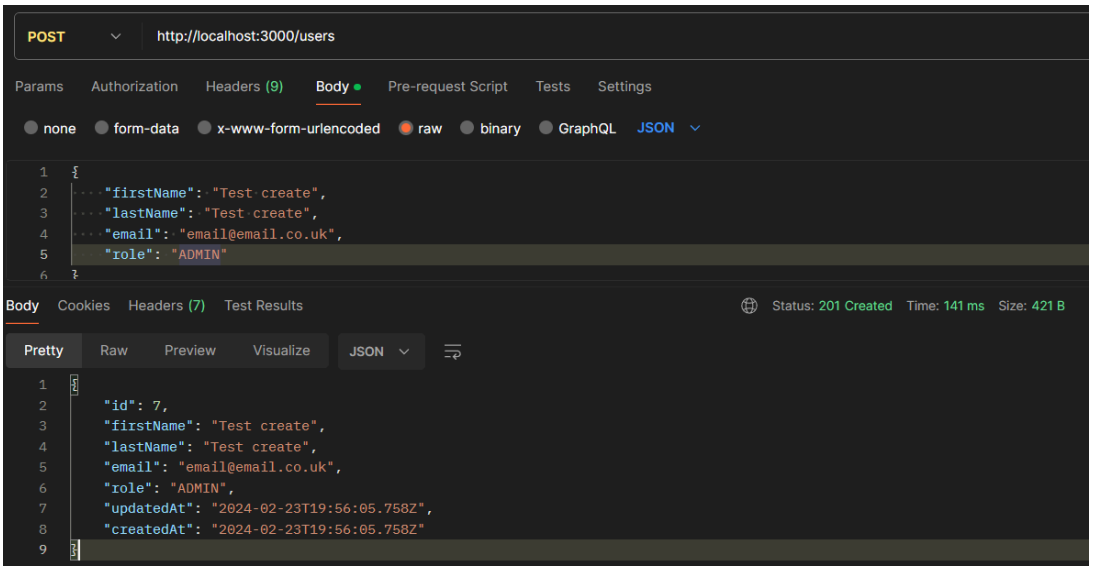

Рисунок 4 — Создание пользователя

Можно заметить, что созданному пользователю был присвоен идентификатор “7”. Получим созданного пользователя по данному идентификатору (рисунок 5).

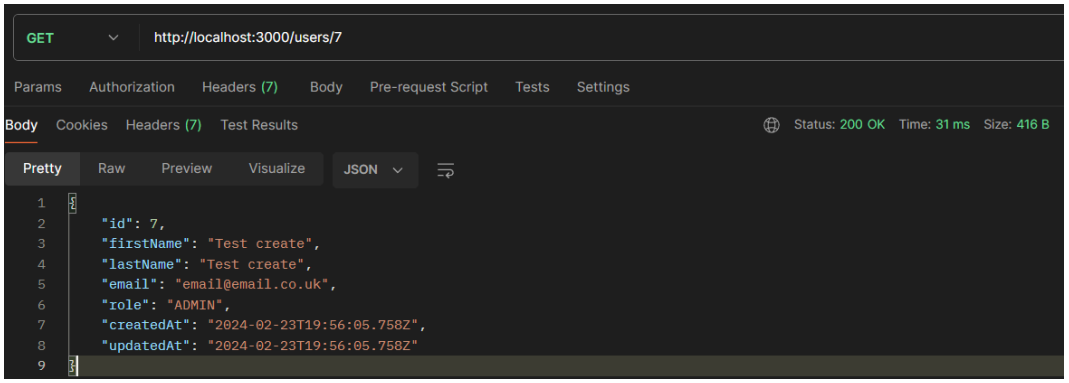

Рисунок 5 — Получение пользователя по идентификатору

Далее, все по тому же идентификатору произведем операцию удаления пользователя. Результат приведен на рисунке 6.

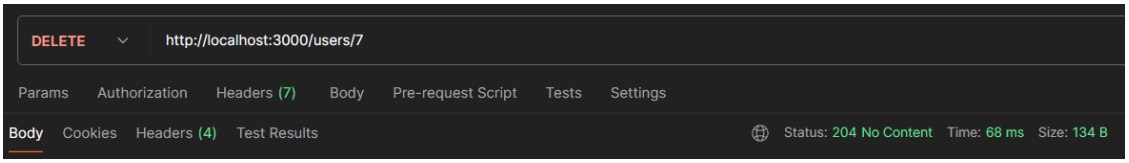

Рисунок 6 — Удаление пользователя по идентификатору

Для проверки правильности работы произведенной операции удаления, попробуем получить удаленного пользователя по идентификатору. Ожидается, что API вернет HTTP-статус 404 NOT FOUND. Как можем видеть на рисунке 7, операция удаления была выполнена успешно.

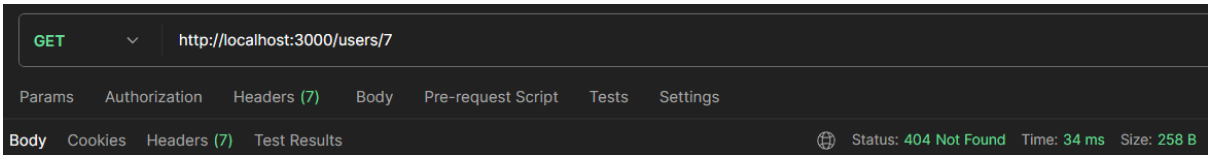

Рисунок 7 — Получение удаленного пользователя по идентификатору

Написать запрос для получения пользователя по id/email

Для выполнения запроса получения пользователя по id/email в ORM Sequelize используется следующий синтаксис (рисунок 8).

```
await db.User.findOne({
    where: {
        // condition
    }
})
```

Рисунок 8 — Синтаксис запроса получения пользователя в Sequelize

Для получения пользователя по заданным параметрам id или email, эти параметры следует поместить на место блока условия. Пример приведен на рисунке 9.

```
await db.User.findOne({
    where: {
        id: 1,
        email: "email@email.co.uk"
    }
})
```

Рисунок 9 — Запрос получения пользователя с заданными параметрами

Вывод

В ходе выполнения работы были изучены основы использования ORM Sequelize в связке с микро фреймворком express.js. В результате было создано небольшое приложение, предоставляющее CRUD API для сущности пользователя.