Министерство образования Республики Беларусь

Учреждение образования

«ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий и робототехники

Кафедра «Информационные системы и технологии»

**КУРСОВОЕ ПРОЕКТИРОВАНИЕ**

по дисциплине «Современные средства разработки серверных приложений»

Разработка REST-сервиса ЖКХ

Выполнил: студент группы 2Ит-12

Станкевич Павел Сергеевич

Проверил: заведующий кафедрой ИСиТ

Казаков Вадим Евгеньевич

Витебск, 2024

# **Введение**

Тема: Разработка REST-сервиса для управления жилищно-коммунальными услугами (ЖКХ).

Цель: Целью данного курсового проекта является разработка высокоэффективного и надёжного REST-сервиса для управления ЖКХ, использую Java с фреймворком Spring Boot.

Задачи:

1. Разработать архитектуру REST-сервиса;
2. Реализовать CRUD операции;
3. Создать методы бизнес-логики для обработки запросов и обеспечения корректного функционирования сервиса;
4. Разработать механизм разграничения доступа к эндпоинтам, обеспечивая безопасность и конфиденциальность данных.

Дальнейшие разделы курсового проекта будут посвящены более детальному рассмотрению предметной области, проектированию и реализации REST-сервиса, анализу методов бизнес-логики и механизмов безопасности, а также оценке эффективности разработанного решения.

# **1. Разработка REST-сервиса**

Стартовой точкой разработки хорошего кода является планирование приложения. В нашем случае REST-сервис «ЖКХ» должен представлять собой комплексную систему управления данными о жильцах, зданиях и квартирах. В приложении будет реализовано добавление новых пользователей, авторизация, и ограничение доступа к эндпоинтам в зависимости от роли пользователя.

## 1.1 Сервис и его структура

Архитектура сервиса будет выглядеть следующим образом:

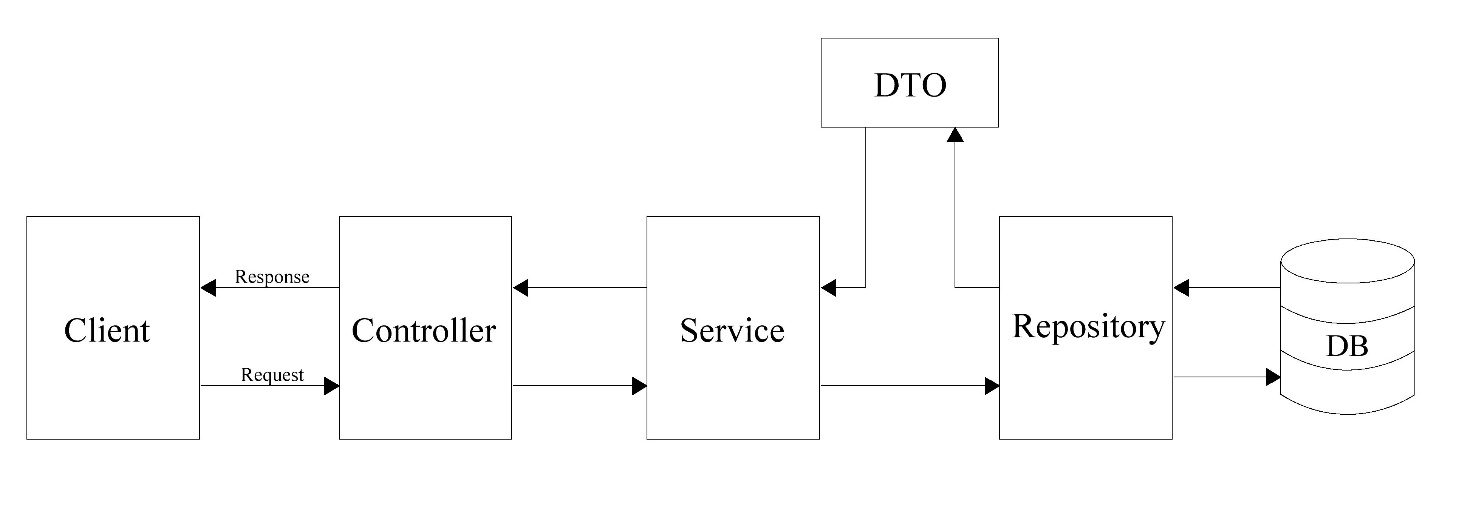


Рисунок 6.1 – Архитектура сервиса

## 1.2 Проектирование базы данных

Спроектируем базу данных, для этого воспользуемся СУБД MySQL Workbench. База данных будет содержать таблицы person, flat, building, person\_link\_flat и user, которым соответствуют сущности Person, Flat, Building, PersonLinkFlat и User.

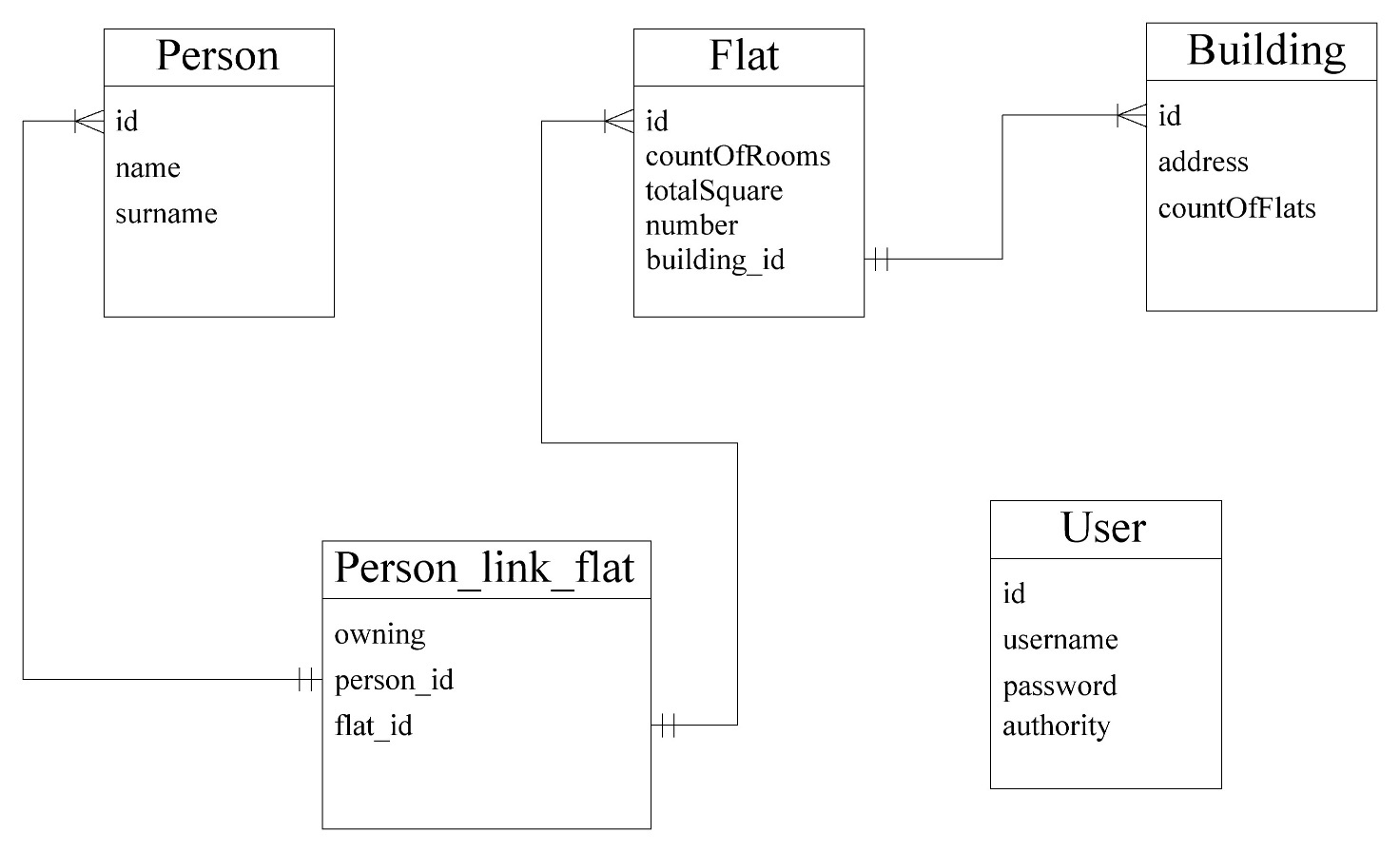


Рисунок 6.2 – Диаграмма базы данных проекта

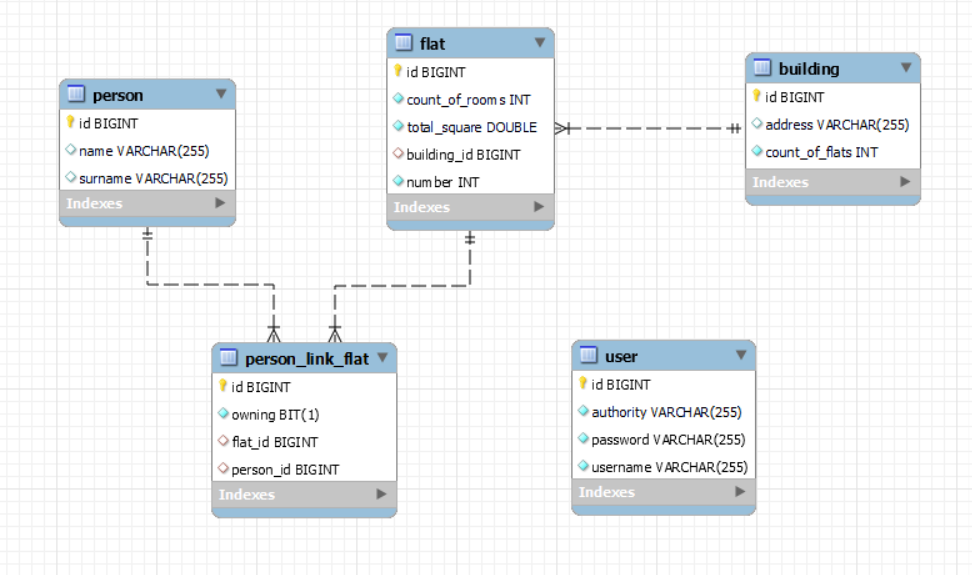


Рисунок 6.3 – Диаграмма базы данных в СУБД MySQL

Person – таблица, для хранения информации о жильцах. Поля:

* id – идентификатор жильца,
* name – имя жильца,
* surname – фамилия жильца.

Flat – таблица, для хранения информации о квартирах. Поля:

* id – идентификатор квартиры,
* count\_of\_rooms – количество комнат,
* total\_square – общая площадь,
* number – номер,
* building\_id – идентификатор здания, в котором расположена квартира.

Building – таблица, для хранения информации о зданиях. Поля:

* id – идентификатор здания,
* address – адрес,
* count\_of\_flats – количество квартир.

Person\_link\_flat – таблица, для хранения связей жильцов и квартир. Поля:

* ­­­id – идентификатор связи,
* owning – владеет ли жилец данной квартирой,
* flat\_id – идентификатор квартиры,
* person\_id – идентификатор жильца.

User – таблица, для хранения информации о пользователях. Поля:

* id – идентификатор пользователя,
* username – имя пользователя,
* password – пароль пользователя,
* authority – роль пользователя для доступа к эндпоинтам.

## 1.3 Создание сущностей

Cоздаём сущности в пакете Entity:

1. Класс Person:

* Представляет жильцов;
* Аннотирован как @Entity, что означает, что он является сущностью, связанной с базой данных;
* Содержит поля для идентификатора (id), имени (name) и фамилии (surname) жильца;
* Устанавливает связь с классом PersonLinkFlat через аннотацию @OneToMany.

1. Класс Flat:

* Представляет квартиры;
* Он аннотирован как @Entity, что означает, что он является сущностью, связанной с базой данных;
* Содержит поля для идентификатора (id), номера (number), количества комнат (countOfRooms) и общей площади (totalSquare) квартиры;
* Устанавливает связь с классом PersonLinkFlat через аннотацию @OneToMany и связь с классом Building через аннотацию @ManyToOne.

1. Класс Building:

* Представляет здания;
* Он аннотирован как @Entity, что означает, что он является сущностью, связанной с базой данных;
* Содержит поля для идентификатора (id), адреса (address) и количества квартир (countOfFlats) здания;
* Устанавливает связь с классом Flat через аннотацию @OneToMany.

1. Класс PersonLinkFlat:

* Представляет собой промежуточную таблицу для связи жильцов и квартир способом many to many;
* Он аннотирован как @Entity, что означает, что он является сущностью, связанной с базой данных;
* Содержит поля для идентификатора (id) и подтверждения того, что жилец является владельцем квартиры (owning);
* Устанавливает связь с классом Person через аннотацию @ManyToOne и связь с классом Flat через аннотацию @ManyToOne.

1. Класс User:

* Представляет пользователя системы;
* Он аннотирован как @Entity, что означает, что он является сущностью, связанной с базой данных;
* Содержит поля для идентификатора (id) и имени (username), пароля (password) и роли (authority) пользователя.

## 1.4 Создание репозиториев

Создаем репозитории для сущностей в пакете Repository:

1. Интерфейс PersonRepository:

* Представляет методы для взаимодействия с базой данных в отношении жильцов;
* Наследует JpaRepository <Person, Long>, что позволяет выполнять некоторые операции с объектами класса Person.

1. Интерфейс FlatRepository:

* Представляет методы для взаимодействия с базой данных в отношении квартир;
* Наследует JpaRepository <Flat, Long>, что позволяет выполнять некоторые операции с объектами класса Flat.

1. Интерфейс BuildingRepository:

* Представляет методы для взаимодействия с базой данных в отношении зданий;
* Наследует JpaRepository <Building, Long>, что позволяет выполнять некоторые операции с объектами класса Building.

1. Интерфейс PersonLinkFlatRepository:

* Представляет методы для взаимодействия с базой данных для связи жильцов и квартир;
* Наследует JpaRepository <PersonLinkFlat, Long>, что позволяет выполнять некоторые операции с объектами класса PersonLinkFlat.

1. Интерфейс UserRepository:

* Представляет методы для взаимодействия с базой данных в отношении пользователей;
* Наследует JpaRepository <User, Long>, что позволяет выполнять некоторые операции с объектами класса User;
* Метод findByUsername позволяет искать пользователя по его логину (username).

## 1.5 Создание сервисов и контроллеров

Создаем сервисы в пакете Service. Эти классы будут реализовывать бизнес-логику.

Создаем контроллеры в пакете Controller. Эти классы будут обрабатывать HTTP-запросы и управлять различными аспектами web-приложения. Они аннотированы как контроллеры, что позволяет Spring обрабатывать HTTP-запросы и возвращать соответствующие ответы.

## 1.6 Функционал REST-сервиса

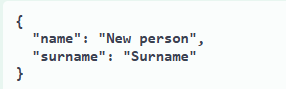
1. Создание нового жильца

URL: /api/person

Person request body is required

method: POST

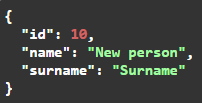
Request body:



Response:

Created:

Response body:



Status: 201

Internal server error:

Response body:



Status: 500

1. Получение списка всех жильцов

URL: /api/person

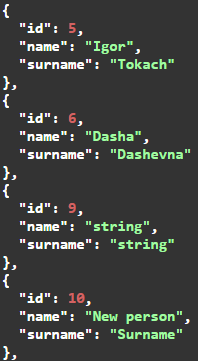
No parameters are required

method: GET

Response:

OK:

Response body:



Status: 200

1. Получение информации о конкретном жильце

URL: /api/person/{id}

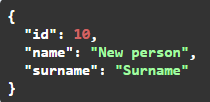
id path variable is required

method: GET

Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

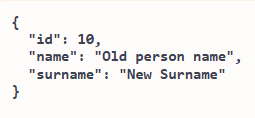
1. Обновление информации о жильце

URL: /api/person

Person request body is required

method: PUT

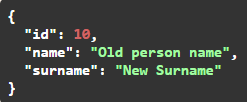
Request body:



Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

1. Удаление жильца

URL: /api/person/{id}

id path variable is required

method: DELETE

Response:

OK

Response body:



Status: 200

Not found:

Response body:



Status: 404

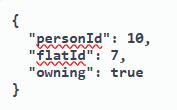
1. Добавление связи между жильцом и квартирой

URL: /api/person/person-link

Map<String, Object> request body is required

method: POST

Request body:



Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

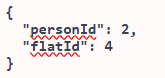
1. Удаление связи между жильцом и квартирой

URL: /api/person/person-link

Map<String, Object> request body is required

method: DELETE

Request body:



Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

1. Получение списка квартир, принадлежащих жильцу

URL: /api/person/person-flats/{id}

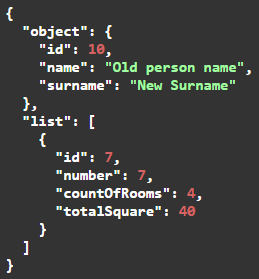
id path variable is required

method: GET

Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

1. Получение количества квартир, принадлежащих жильцу

URL: /api/person/count-flats/{id}

id path variable is required

method: GET

Response:

OK:

Response body:



Not found:

Response body:



Status: 404

1. Получение списка всех квартир

URL: /api/flat

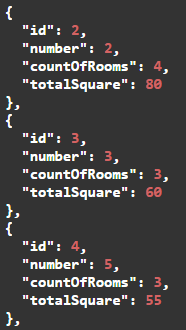
No parameters are required

method: GET

Response:

OK:

Response body:



Status: 200

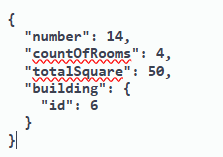
1. Создание новой квартиры

URL: /api/flat

Flat request body is required

method: POST

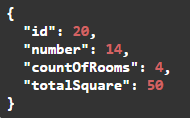
Request body:



Response:

Created:

Response body:



Status: 201

Internal server error:

Response body:



Status: 500

1. Получение информации о конкретной квартире

URL: /api/flat/{id}

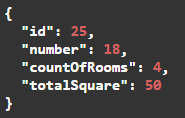
id path variable is required

method: GET

Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

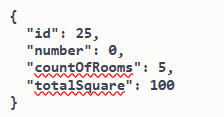
1. Обновление информации о квартире

URL: /api/flat

Flat request body is required

method: PUT

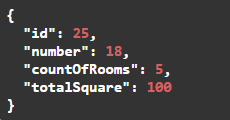
Request body:



Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

1. Удаление квартиры

URL: /api/flat/{id}

id path variable is required

method: DELETE

Response:

OK

Response body:



Status: 200

Not found:

Response body:



Status: 404

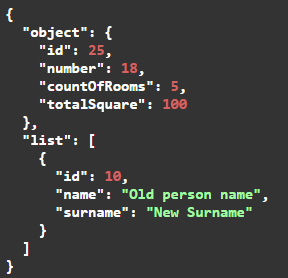
1. Получение списка жильцов, проживающих в квартире

URL: /api/flat/flat-persons/{id}

id path variable is required

method: GET

Response:



Status: 200

Not found:

Response body:



Status: 404

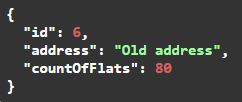
1. Получение информации о здании, в котором находится квартира

URL: /api/flat/building/{id}

id path variable is required

method: GET

Response:



Status: 200

Not found:

Response body:



Status: 404

1. Получение количества жильцов, проживающих в квартире

URL: /api/flat/count-tenants/{id}

id path variable is required

method: GET

Response:



Status: 200

Not found:

Response body:

C:\Users\Admin\AppData\Local\Microsoft\Windows\Clipboard\HistoryData\{78CE9ED1-44DA-47BF-AA2A-739A5757B9A4}\{CB3D4950-0AC1-4873-9B2B-6104E165CCCD}\ResourceMap\{D576E009-8497-4009-A96A-C1D807EFB285}

Status: 404

1. Получение списка всех зданий

URL: api/building

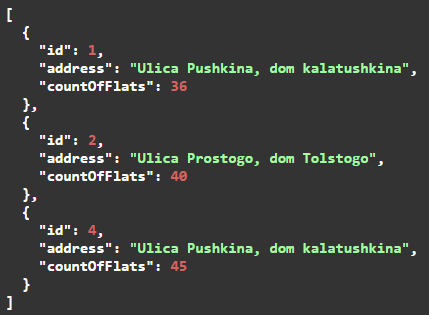
No parameters are required

method: GET

Response:

OK:

Response body:



Status: 200

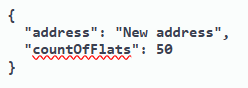
1. Создание нового здания

URL: /api/building

Building request body is required

method: POST

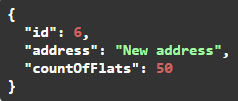
Request body:



Response:

Created:

Response body:



Status: 201

Internal server error:

Response body:



Status: 500

1. Получение информации о конкретном здании

URL: /api/building/{id}

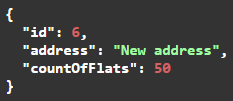
id path variable is required

method: GET

Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

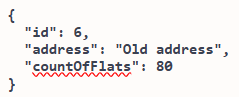
1. Обновление информации о здании

URL: /api/building

Building request body is required

method: PUT

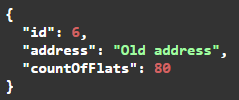
Request body:



Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

1. Удаление здания

URL: /api/building/{id}

id path variable is required

method: DELETE

Response:

OK

Response body:



Status: 200

Not found:

Response body:



Status: 404

1. Получение количества жильцов, проживающих в здании

URL: /api/building/count-tenants/{id}

id path variable is required

method: GET

Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

1. Получение списка владельцев квартир в здании

URL: /api/building/owners/{id}

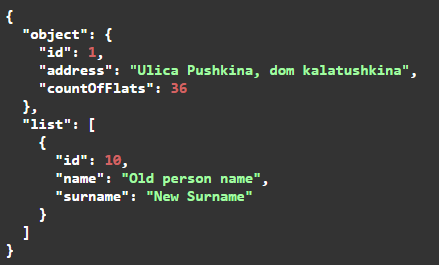
id path variable is required

method: GET

Response:

OK:

Response body:



Status: 200

Not found:

Response body:



Status: 404

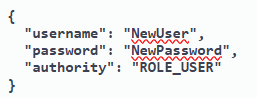
1. Создание нового пользователя

URL: /api/user

User request body is required

method: POST

Request body:



Response:

Created:

Response body:



Status: 201

1. Удаление пользователя

URL: /api/user/{username}

username path variable is required

method: DELETE

Response:

OK

Response body:



Status: 200

Not found:

Response body:



Status: 404

Для настройки Spring Security создаем класс WebSecurityConfig:

* Этот класс аннотирован как @Configuration.
* Включает Spring Security для приложения с помощью @EnableWebSecurity.
* Метод bCryptPasswordEncoder() создает экземпляр BCryptPasswordEncoder, который используется для кодирования паролей.
* Метод authenticationProvider() создает и настраивает AuthenticationProvider, который отвечает за аутентификацию пользователей на основе предоставленных учетных данных.
* Метод securityFilterChain(HttpSecurity http) настраивает правила безопасности для разных HTTP-запросов:
* Эндпоинты HttpMethod.GET, "/api/person/{id}", "/api/flat", "/api/flat/{id}", "/api/flat/building/{id}", "/api/building", "/api/building/{id}" разрешены для пользователей с ролями “USER” или “ADMIN”.
* Все остальные существующие эндпоинты требуют роли “ADMIN”,
* Любые другие запросы будут откланяться методом denyAll().

Остался последний класс Application, который будет запускать сервис. Он аннотирован как @SpringBootApplication, что указывает, что это Spring Boot-приложение. Метод main является точкой входа в приложения, инициализируя Spring Boot-контекст, при запуске.

# **2. Тестирование приложения**

Выполним запросы ко всем доступным эндпоитам. Для этого авторизуемся как admin, так, как только у него есть доступ ко всему.

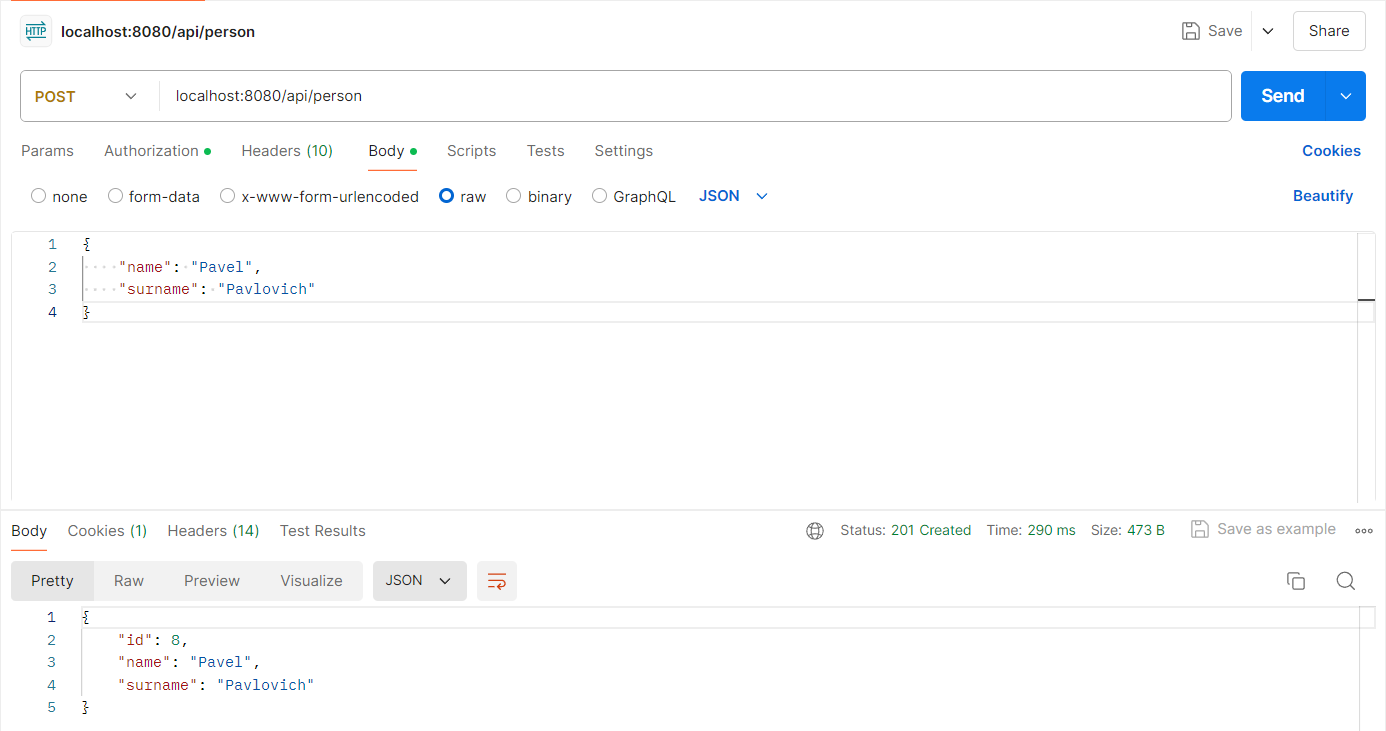


Рисунок 8.1 - Создание нового жильца

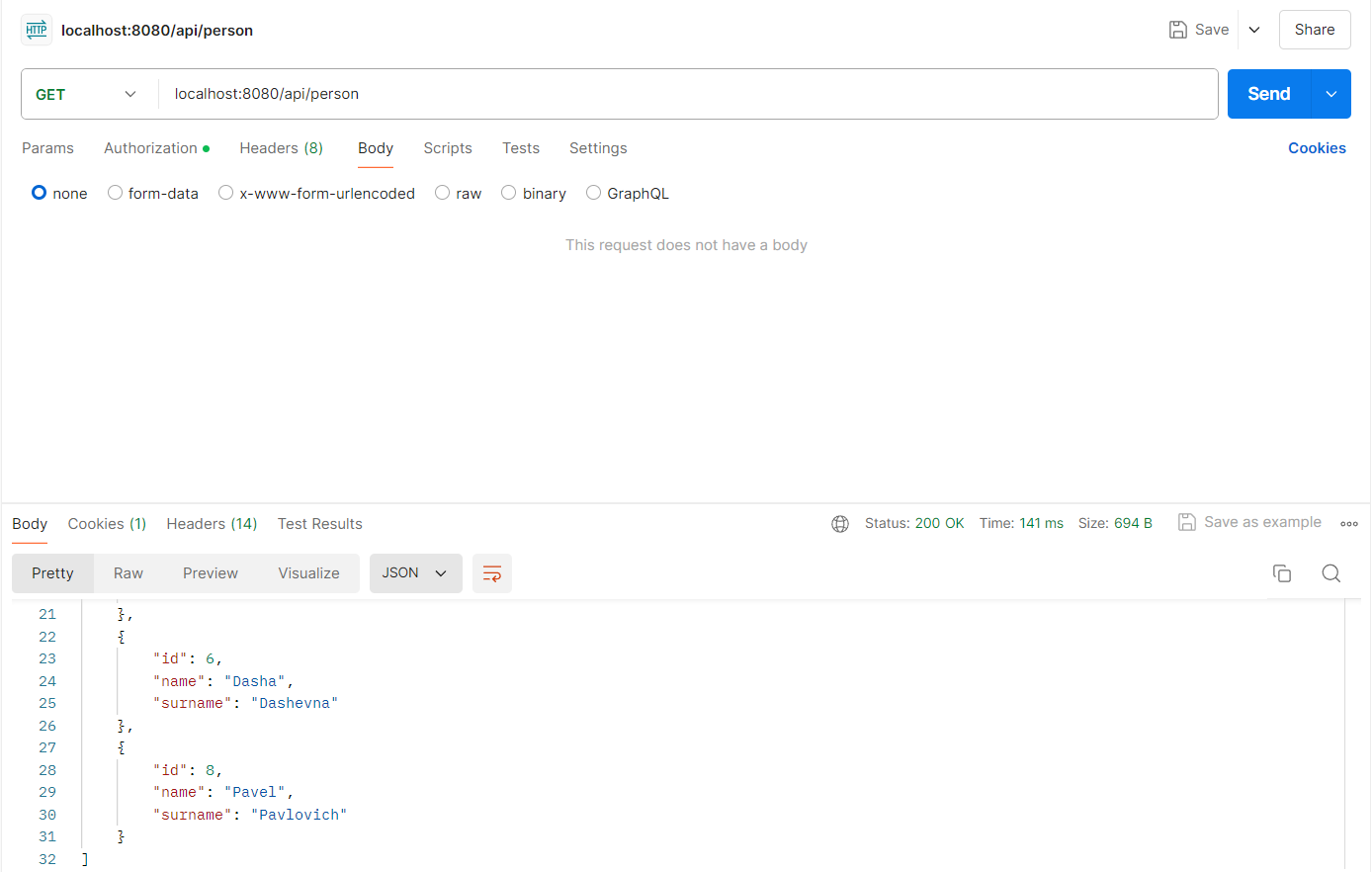


Рисунок 8.2 – Получение списка всех жильцов

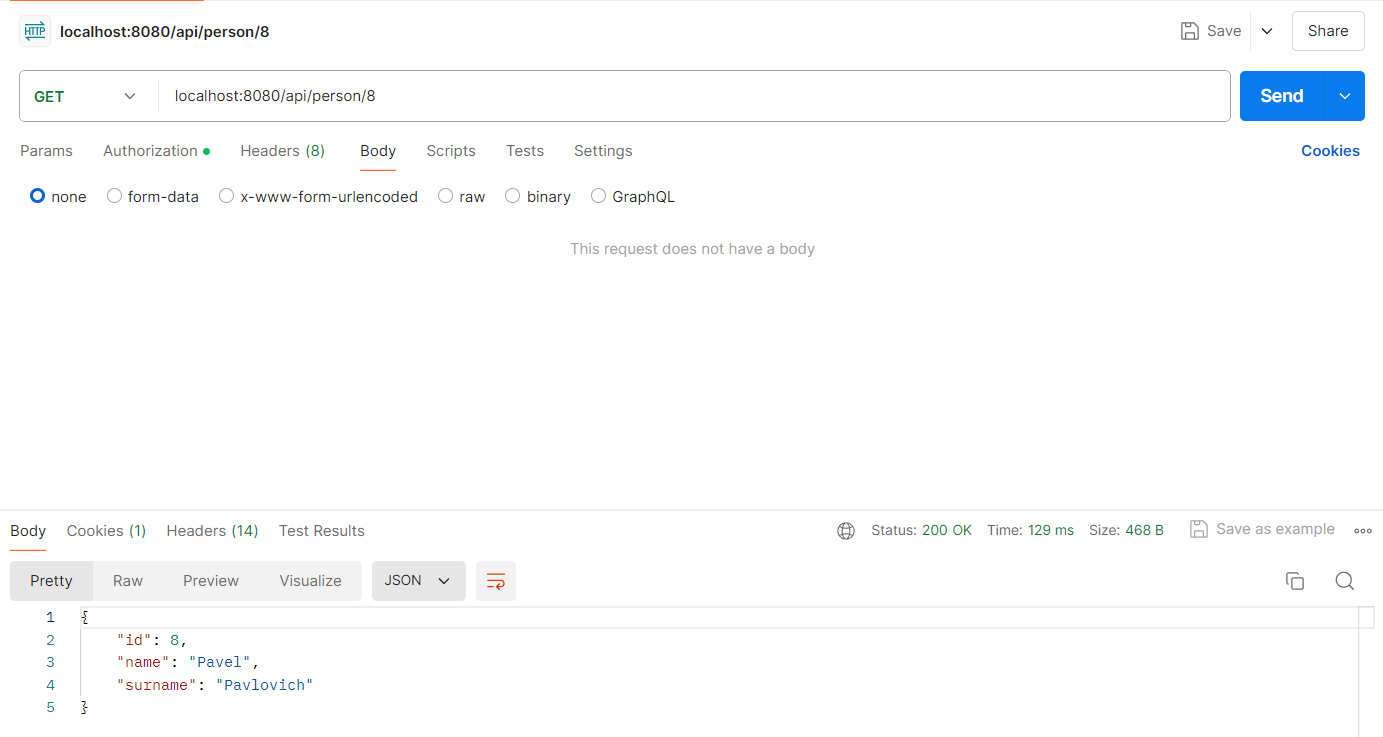


Рисунок 8.3 – Получение информации о конкретном жильце

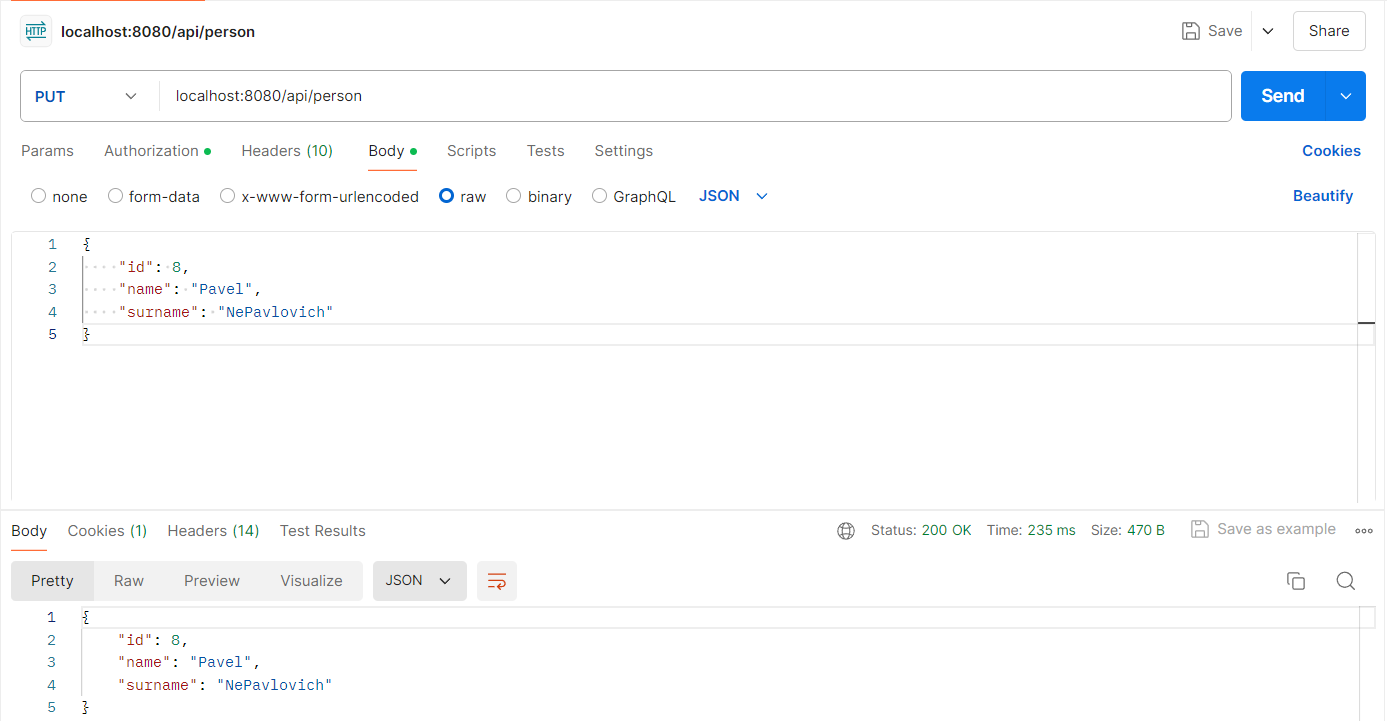


Рисунок 8.4 – Обновление информации о жильце

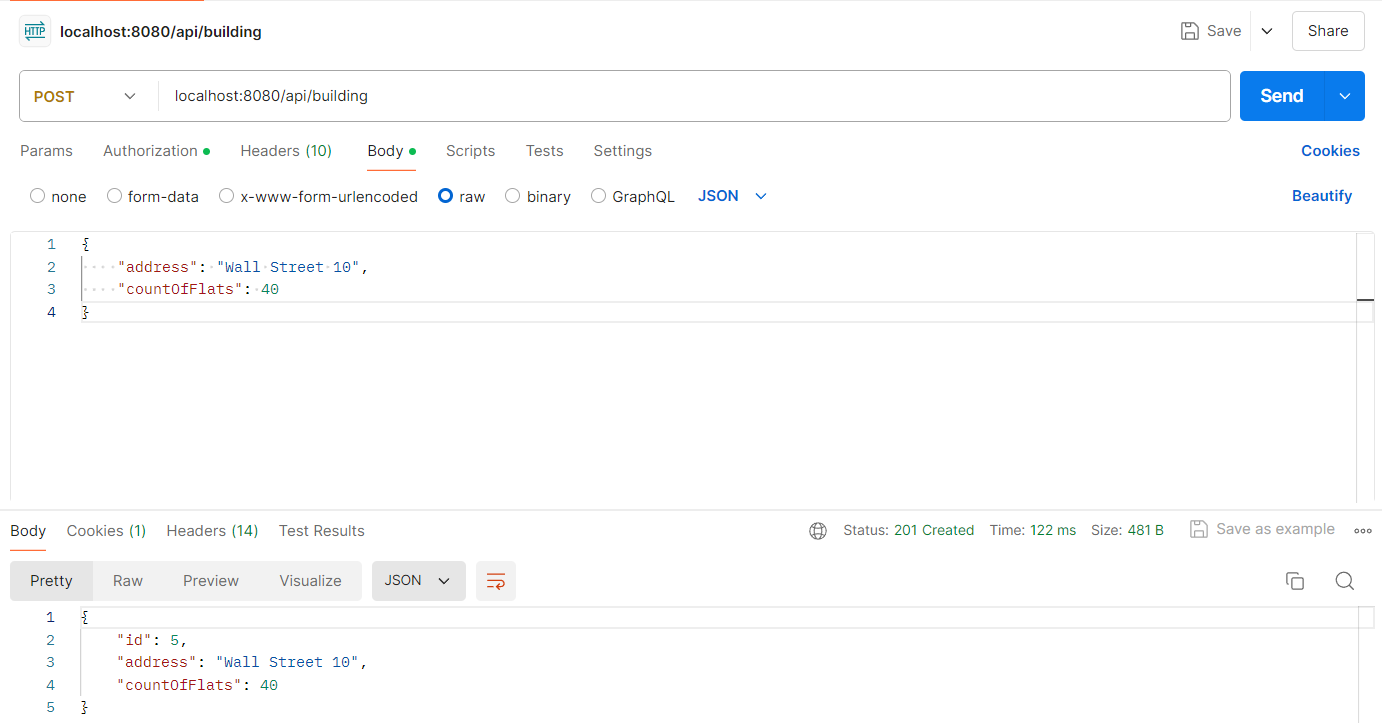


Рисунок 8.5 – Создание нового здания

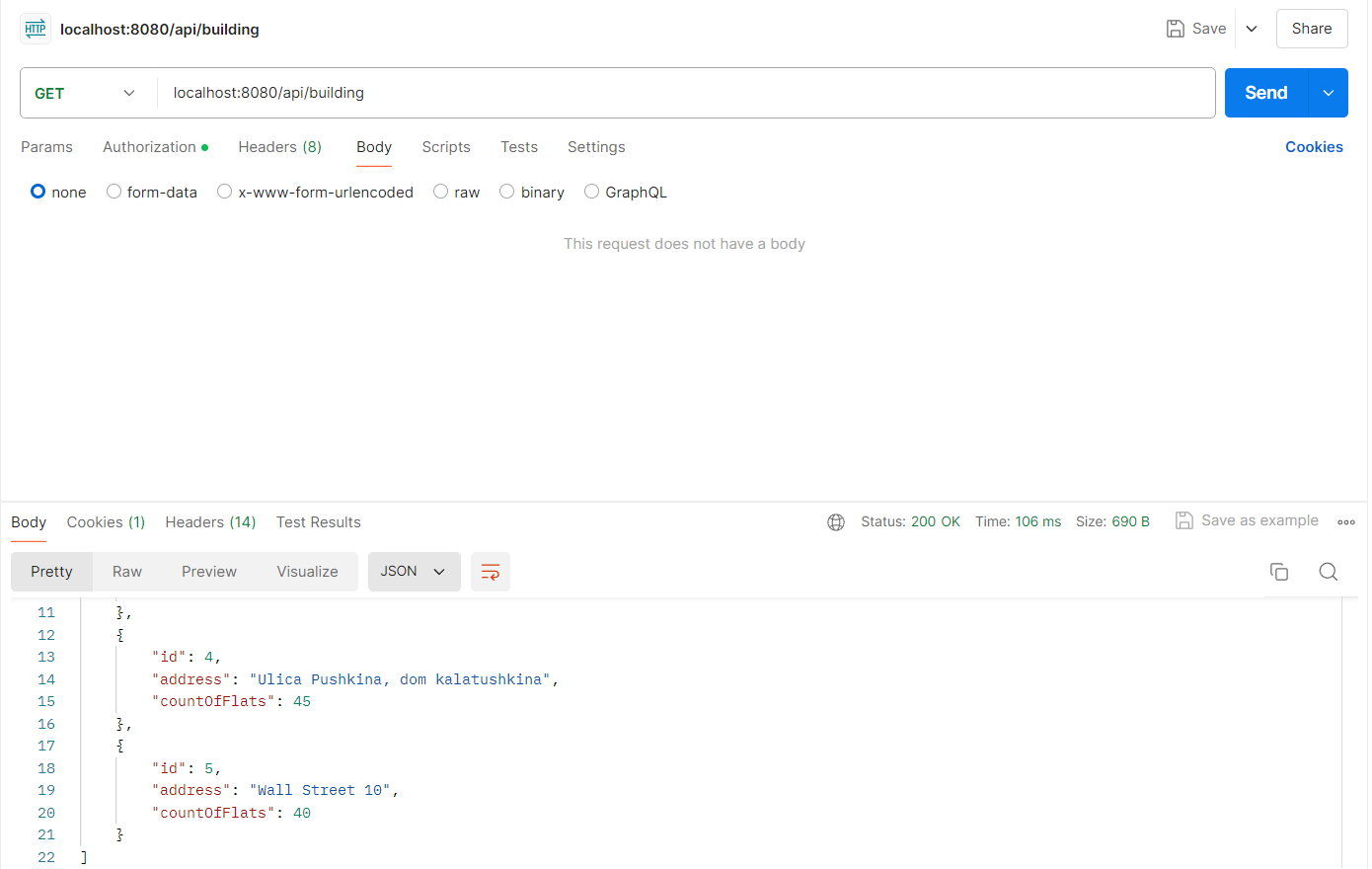


Рисунок 8.6 – Получение списка всех зданий

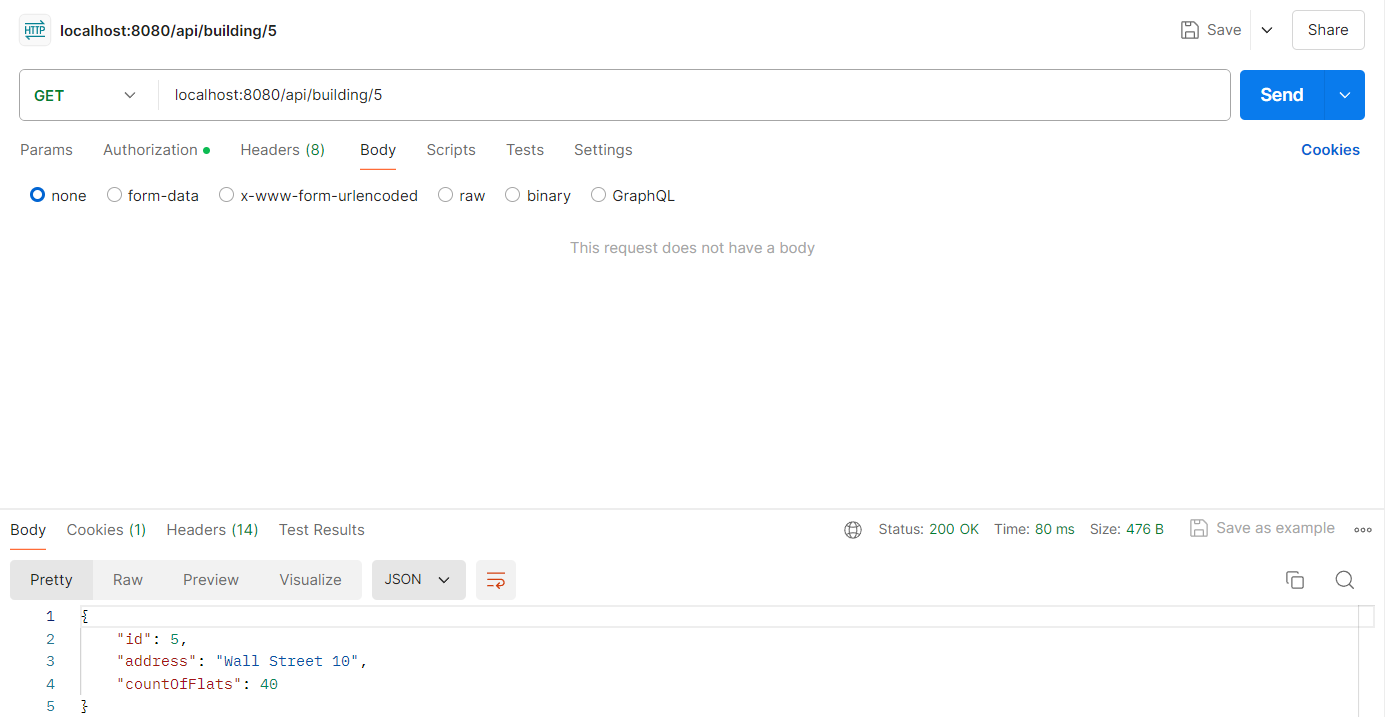


Рисунок 8.7 – Получение информации о конкретном здании

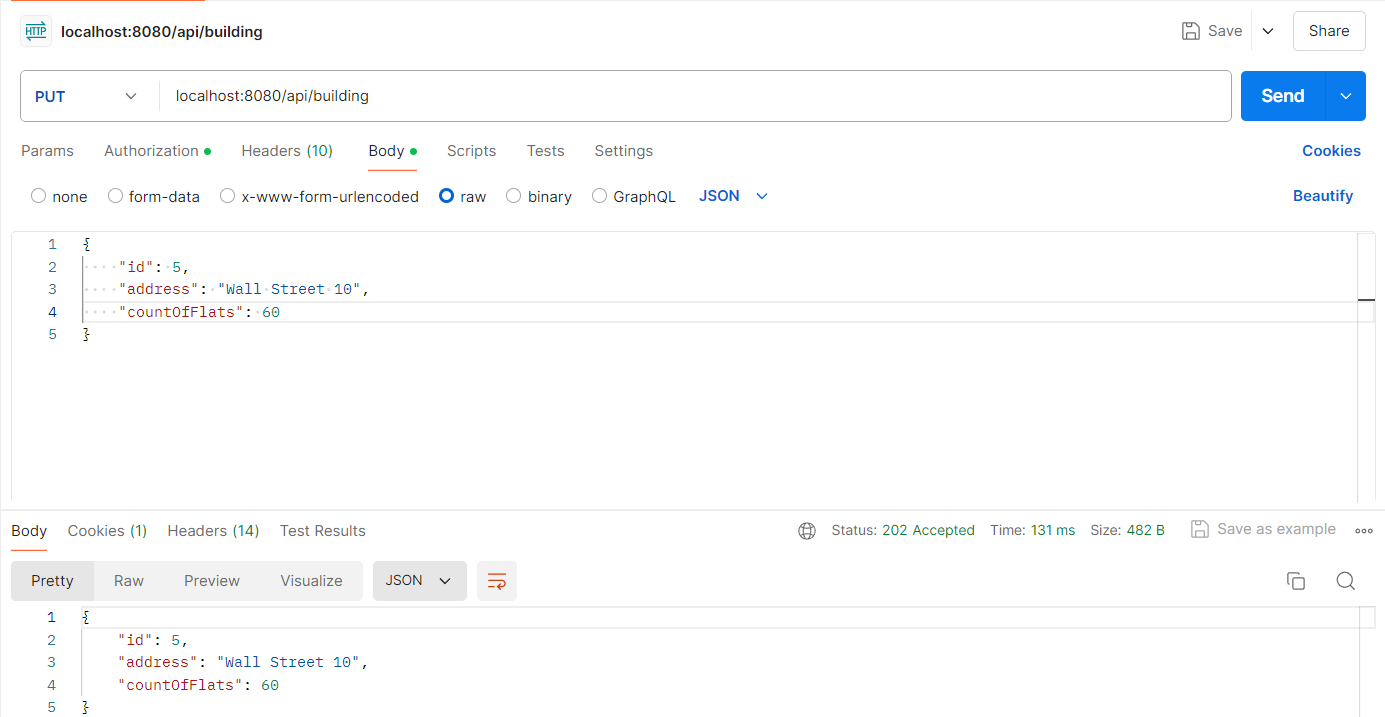


Рисунок 8.8 – Обновление информации о здании

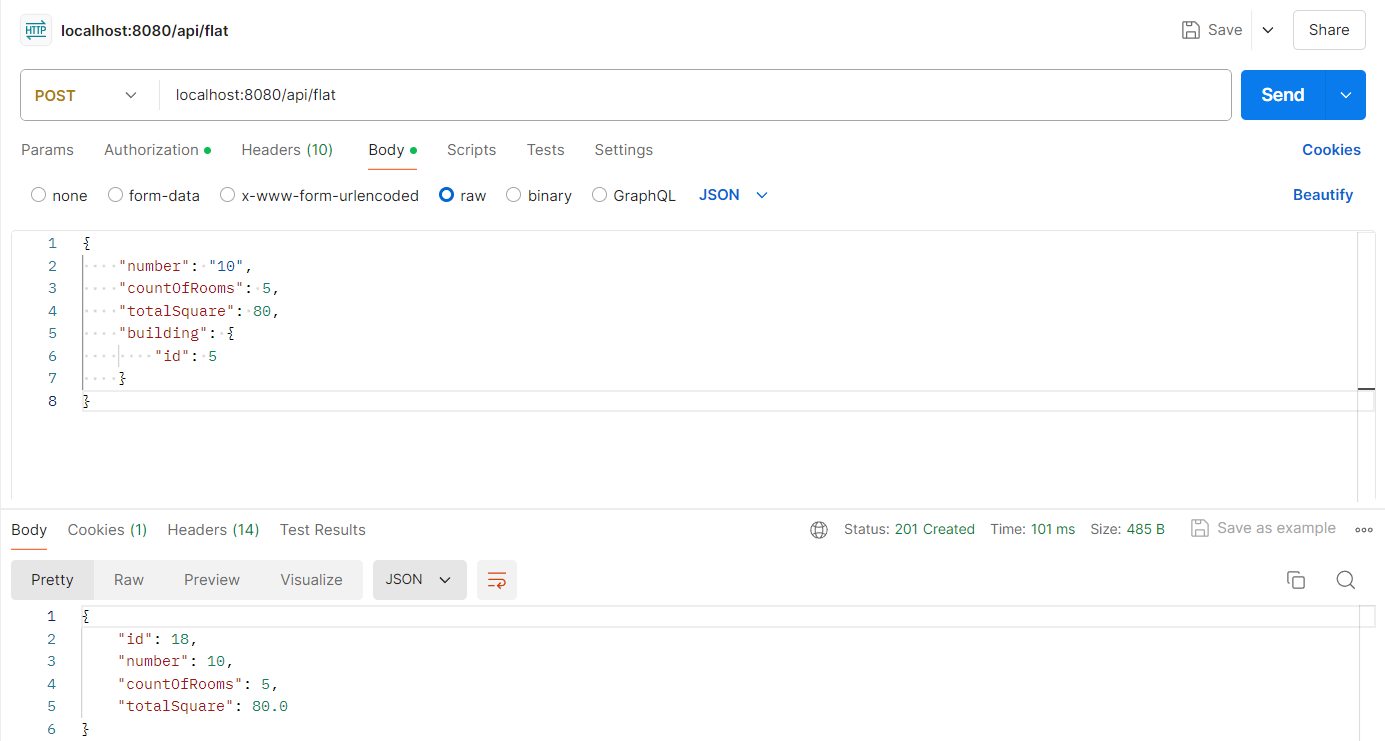


Рисунок 8.9 – Создание новой квартиры

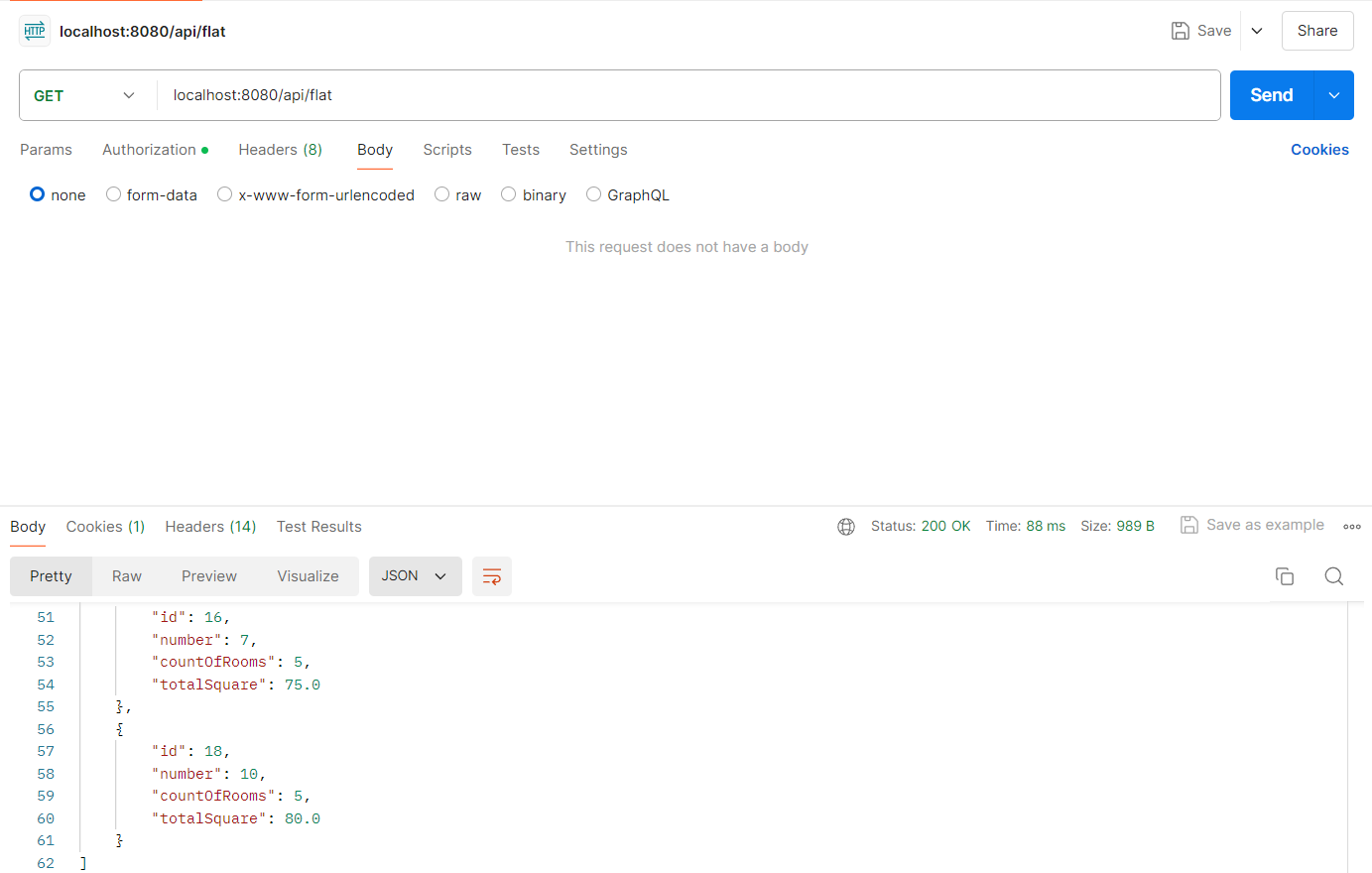


Рисунок 8.10 – Получение списка всех квартир

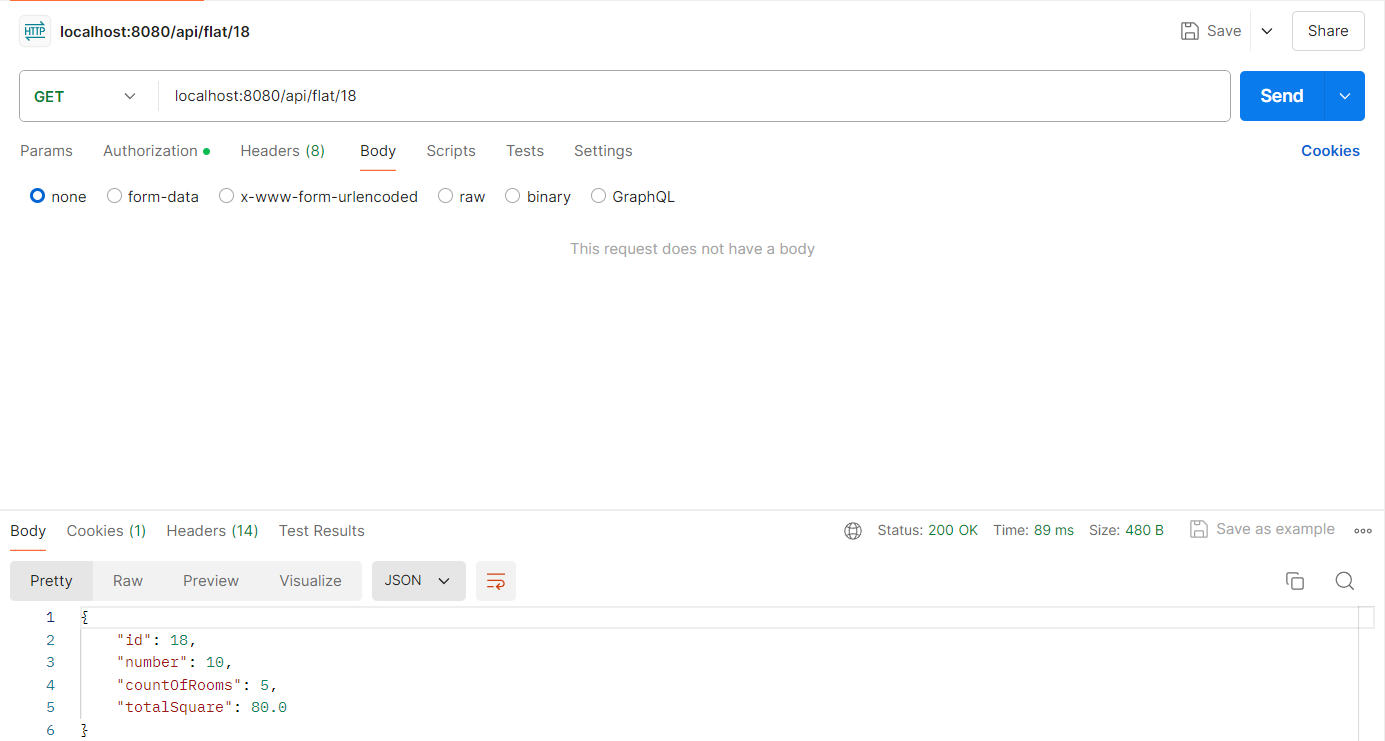


Рисунок 8.11 – Получение информации о конкретной квартире

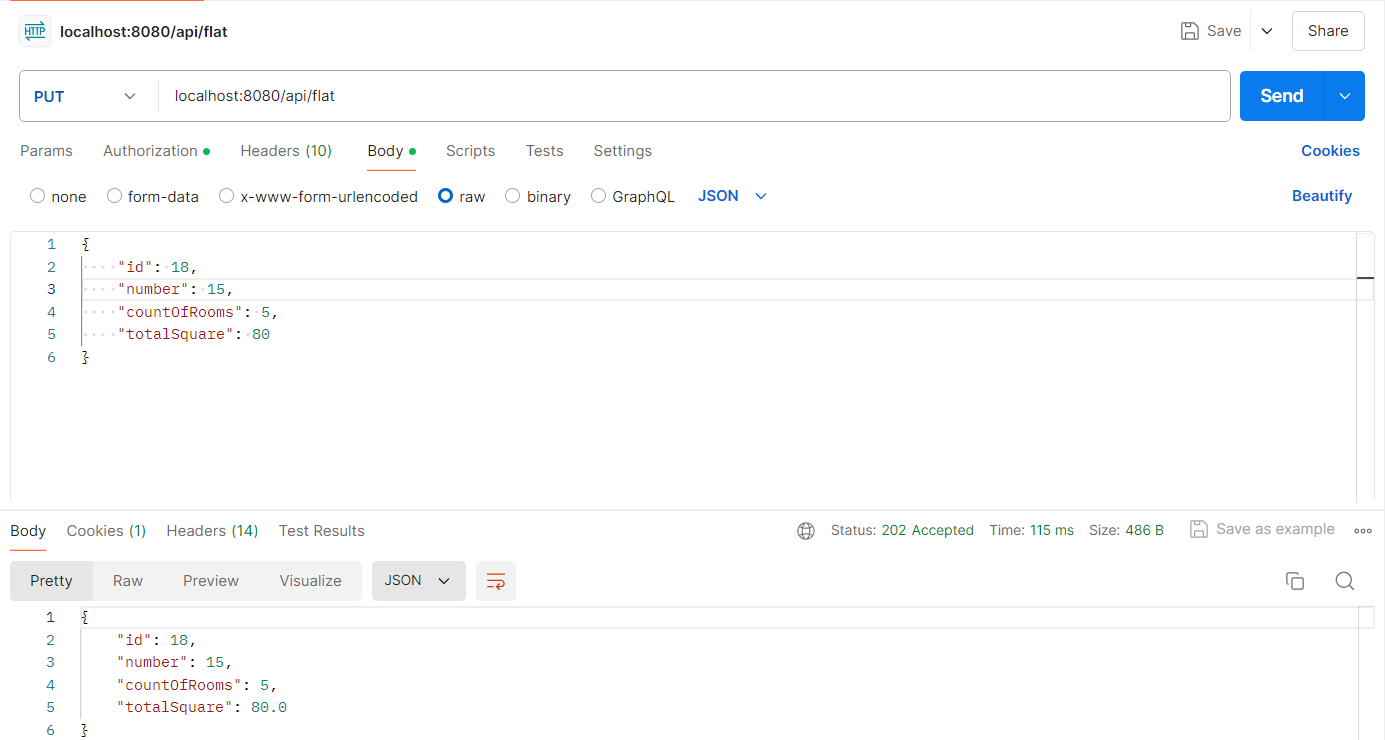


Рисунок 8.12 – Обновление информации о квартире

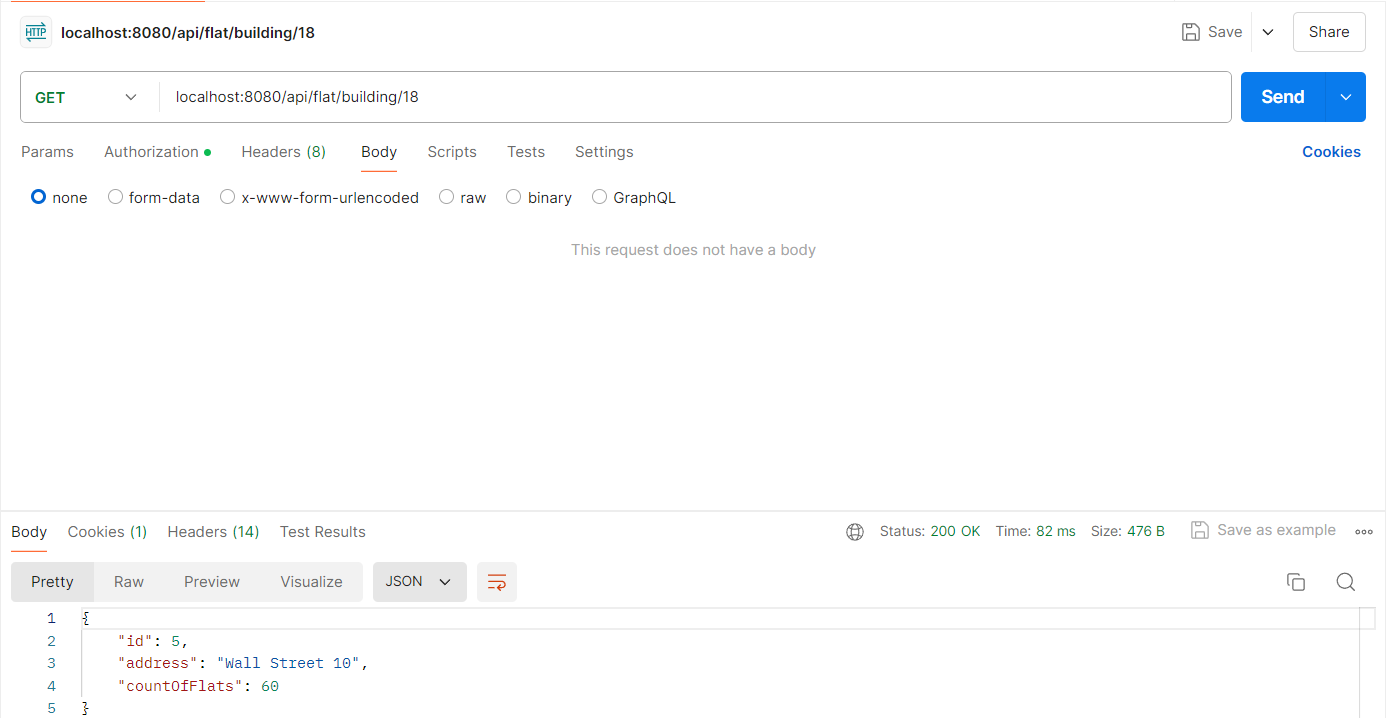


Рисунок 8.13 – Получение информации о здании, в котором находится квартира

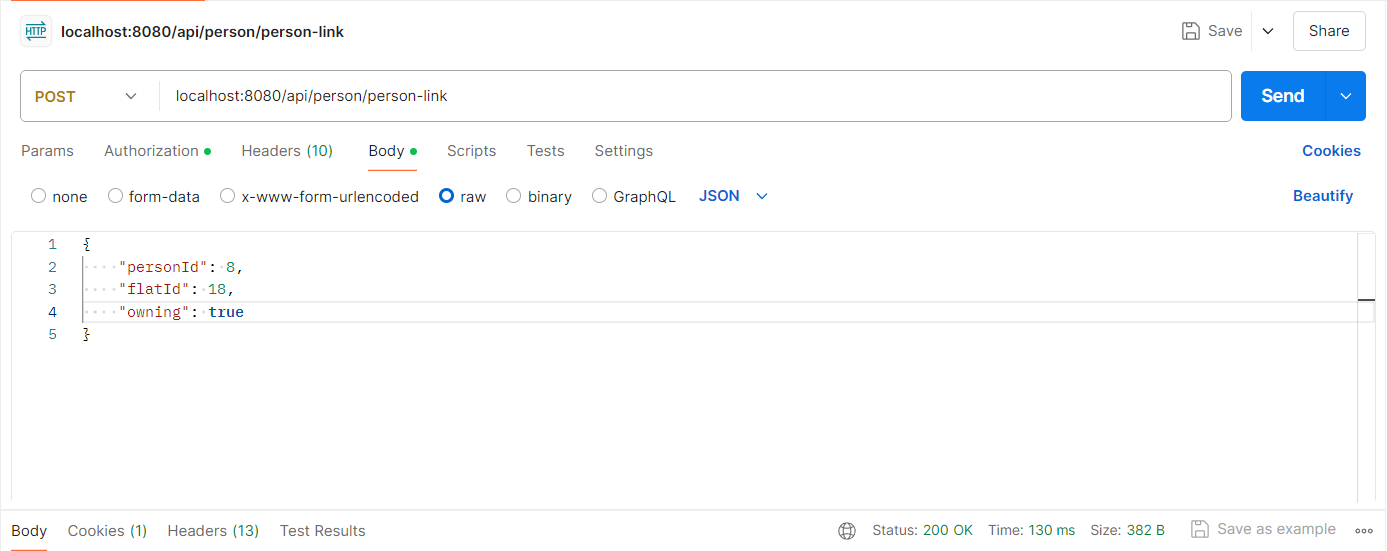


Рисунок 8.14 – Добавление связи между жильцом и квартирой

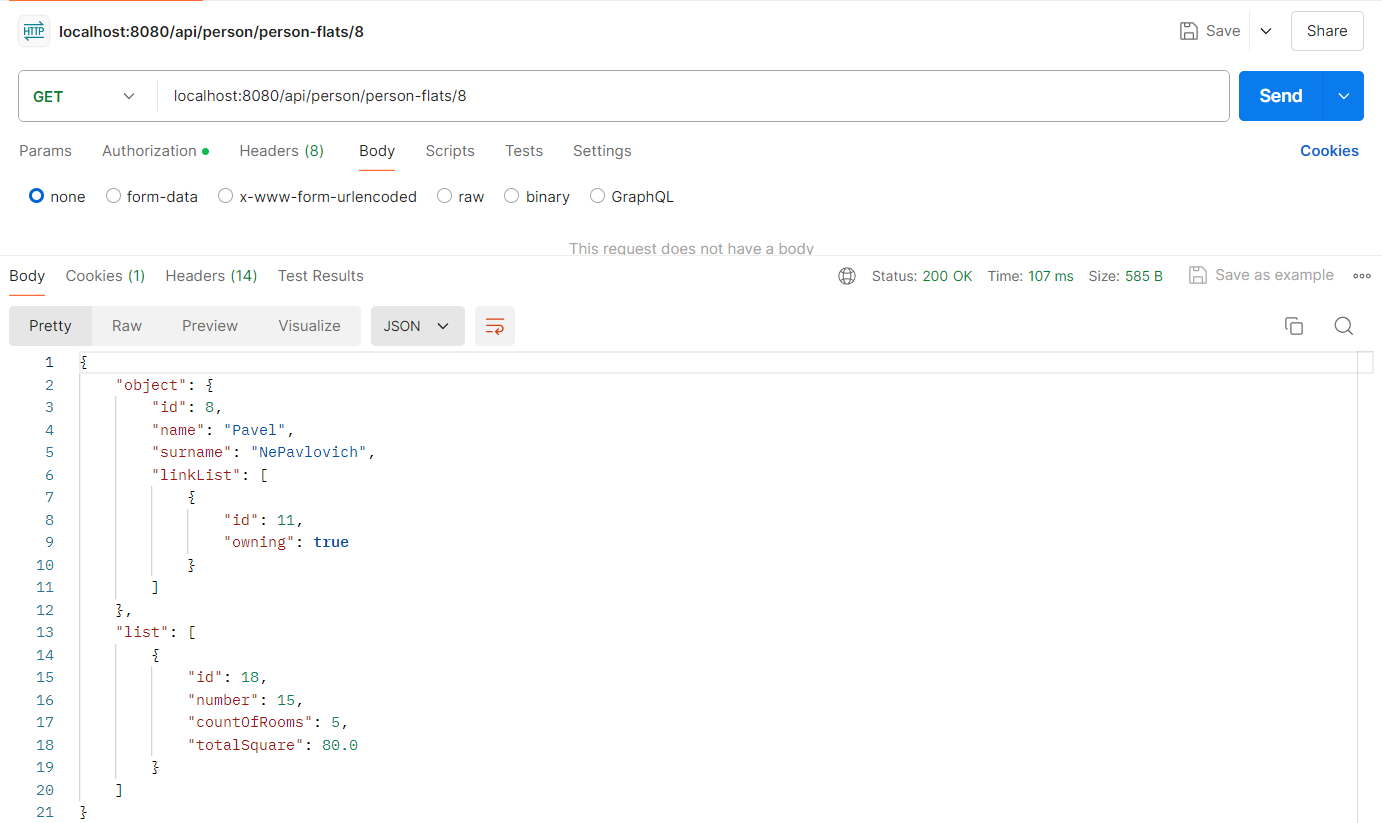


Рисунок 8.15 – Получение списка квартир, принадлежащих жильцу

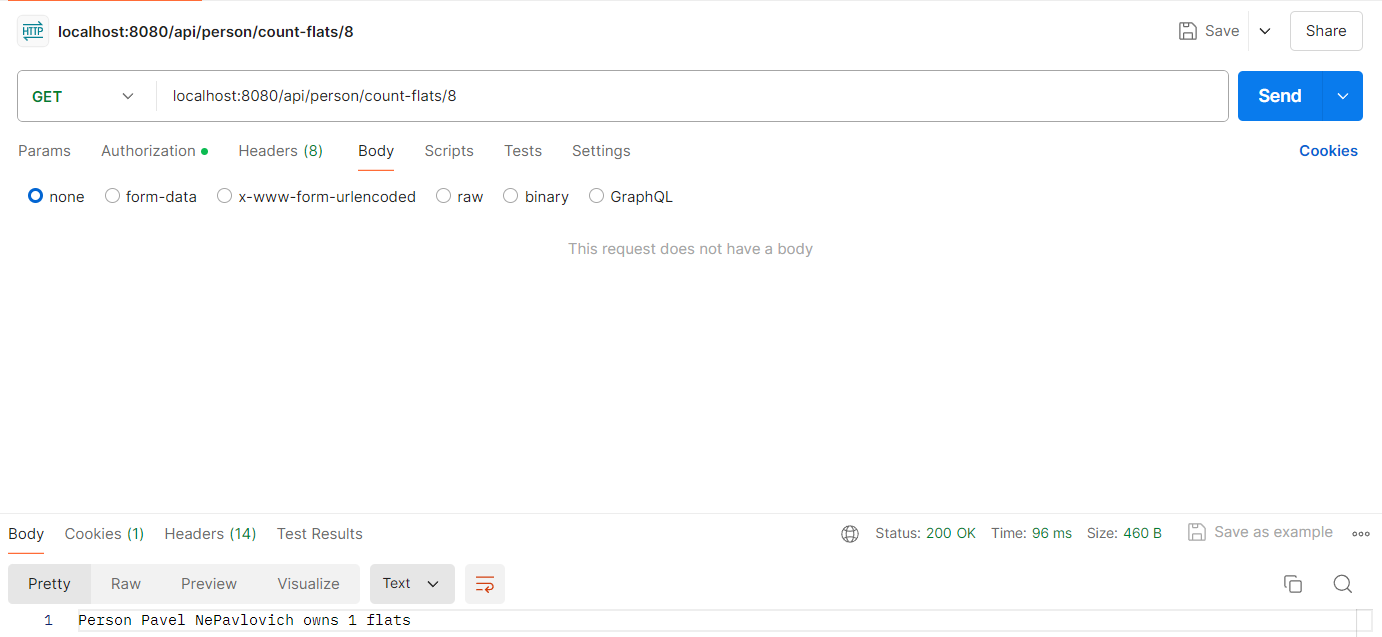


Рисунок 8.16 – Получение количества квартир, принадлежащих жильцу

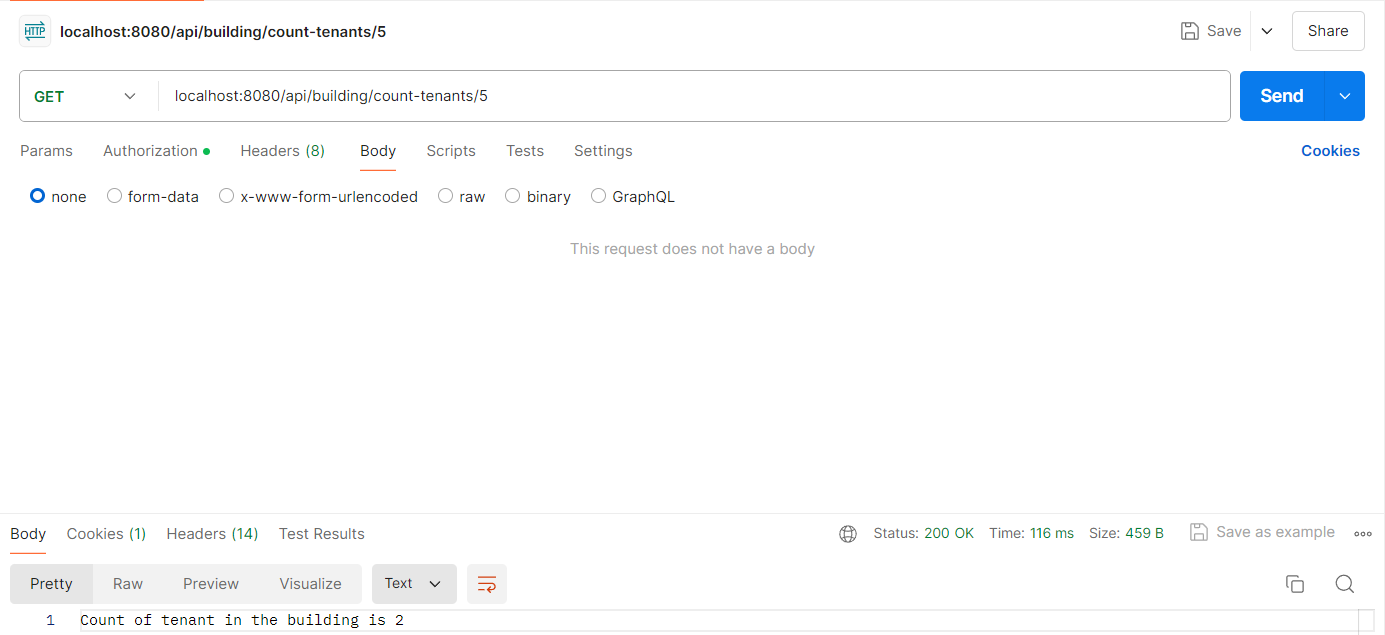


Рисунок 8.17 – Получение количества жильцов, проживающих в здании

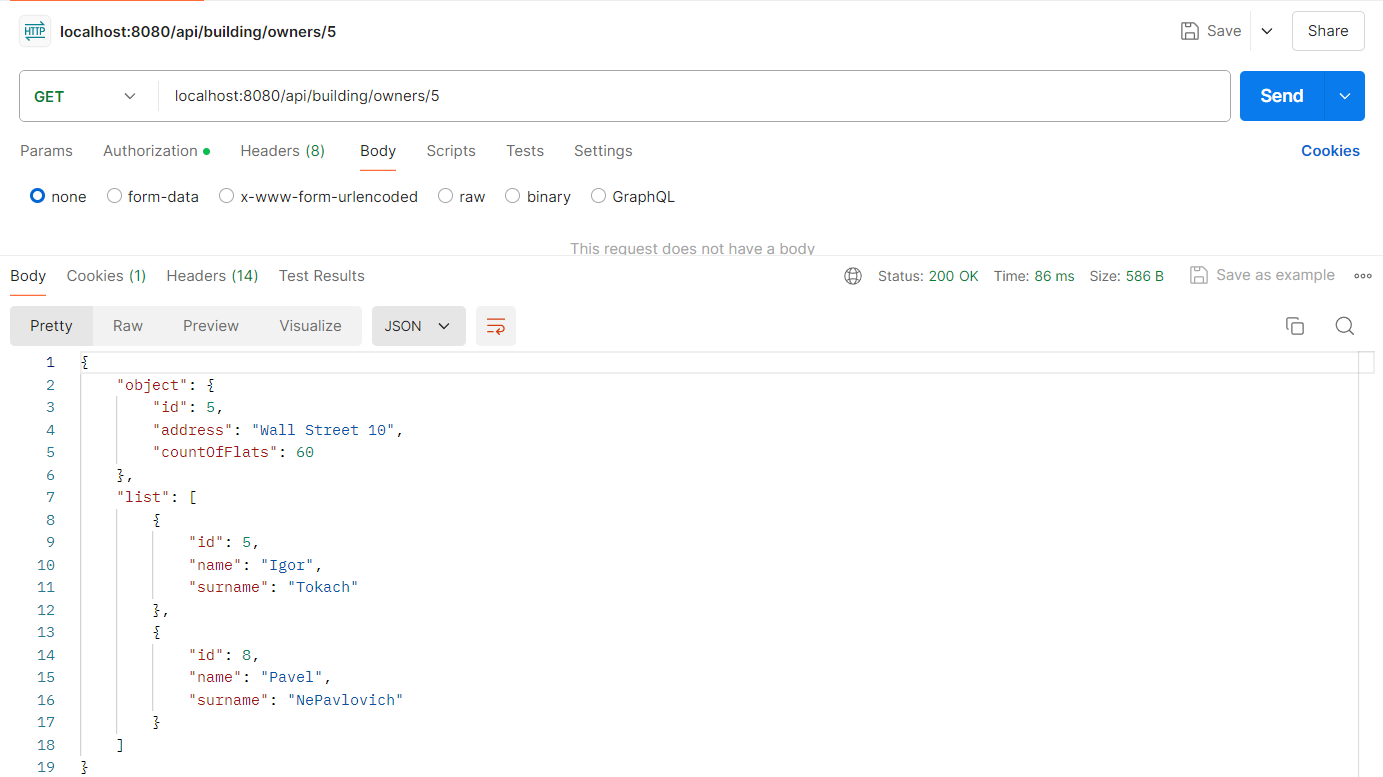


Рисунок 8.18 – Получение списка владельцев квартир в здании

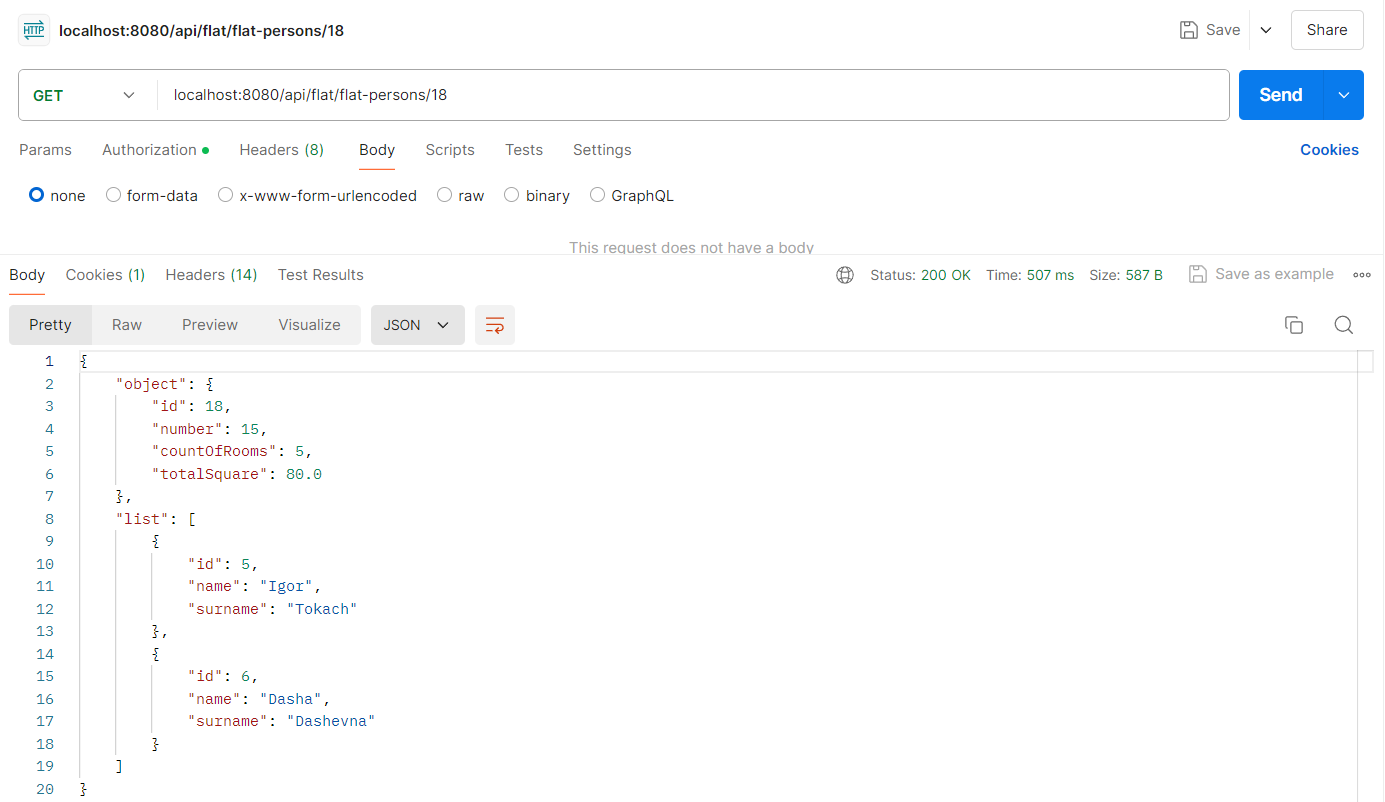


Рисунок 8.19 – Получение списка жильцов, проживающих в квартире

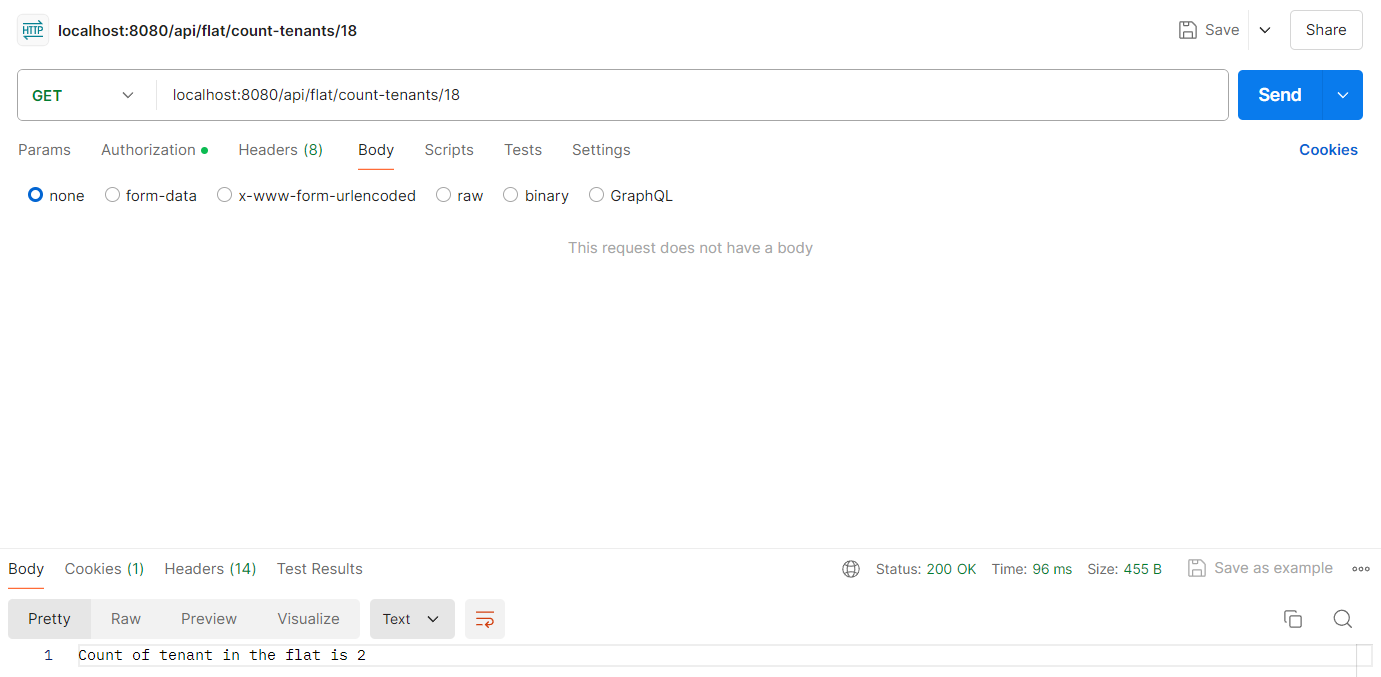


Рисунок 8.20 – Получение количества жильцов, проживающих в квартире

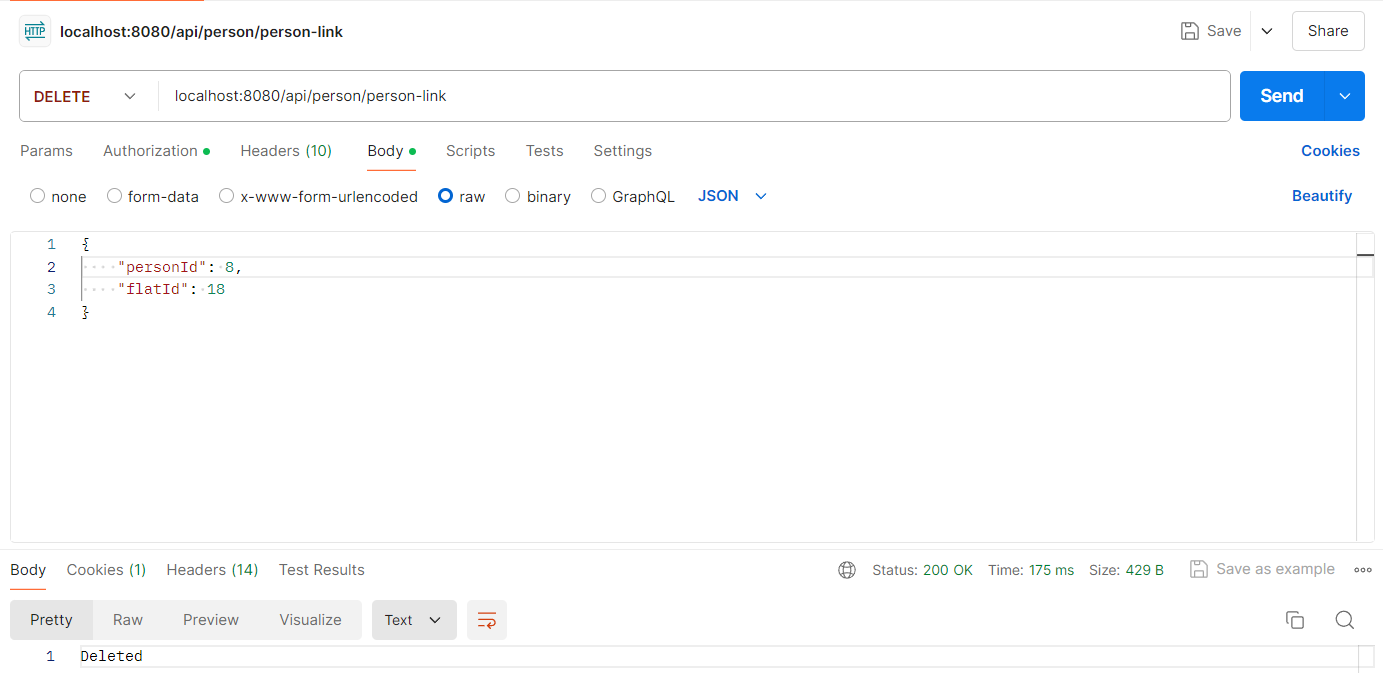


Рисунок 8.21 – Удаление связи между жильцом и квартирой

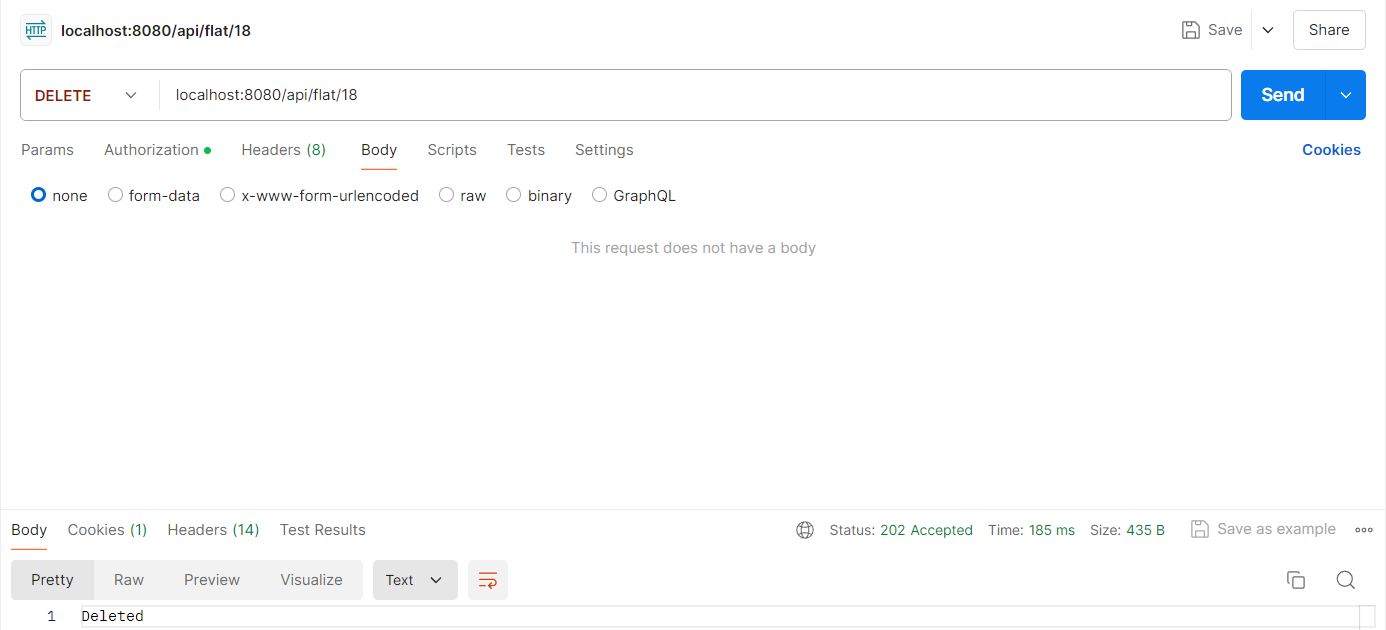


Рисунок 8.22 – Удаление квартиры

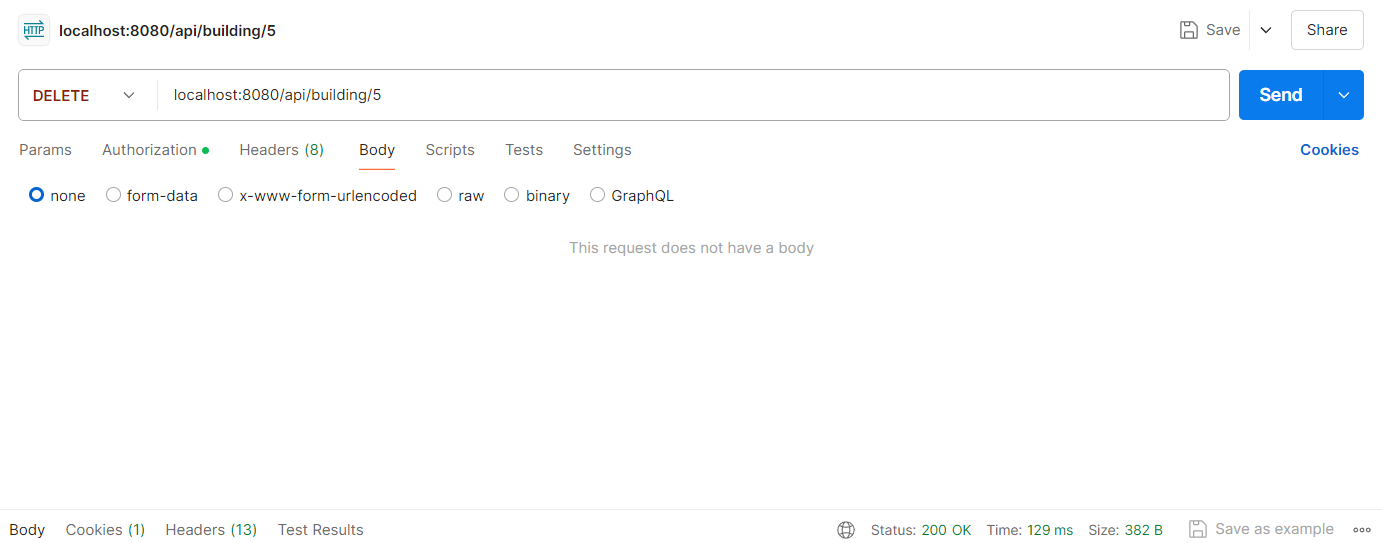


Рисунок 8.23 – Удаление здания

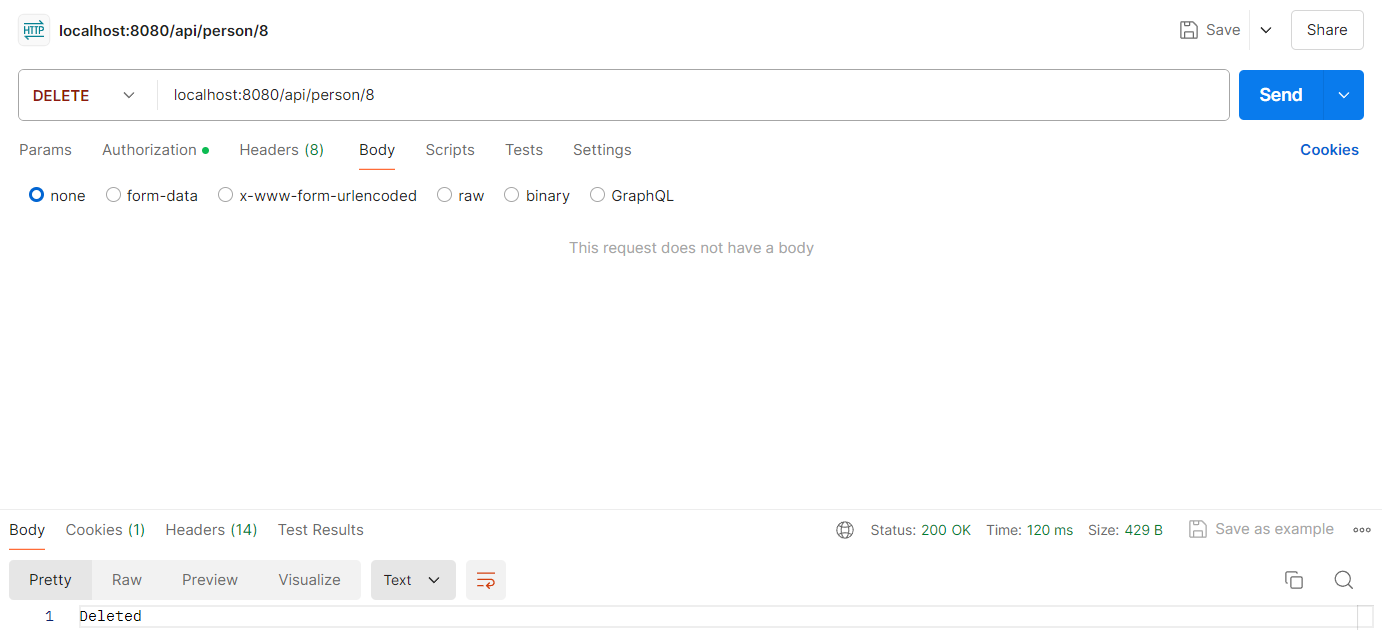


Рисунок 8.24 – Удаление жильца

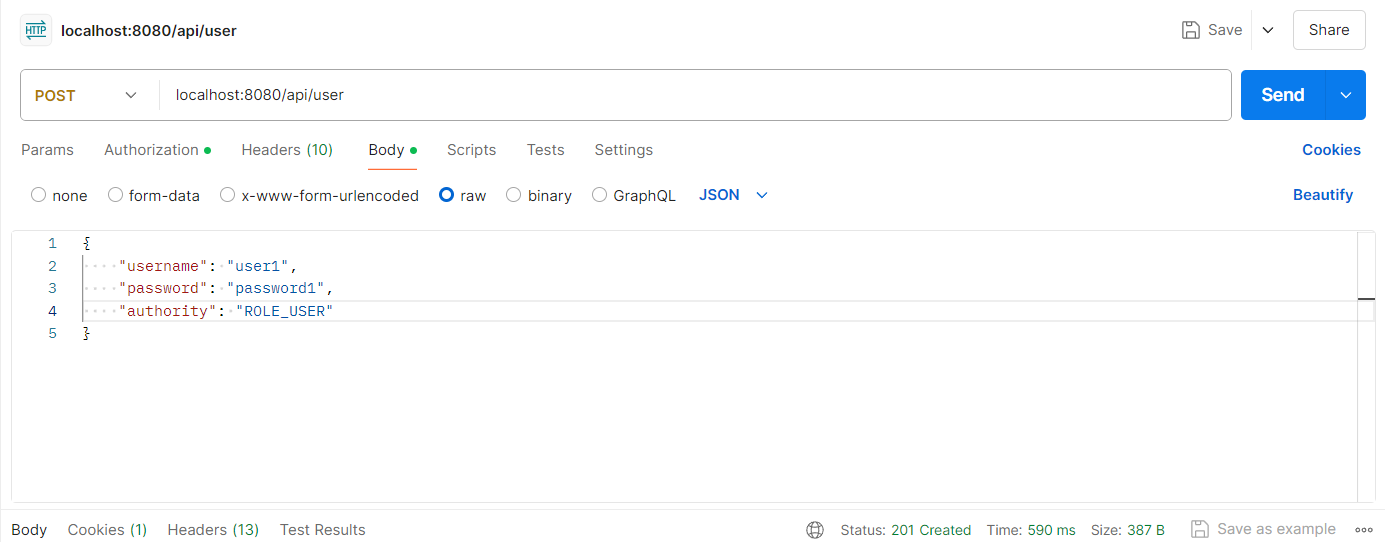


Рисунок 8.25 – Создание нового пользователя

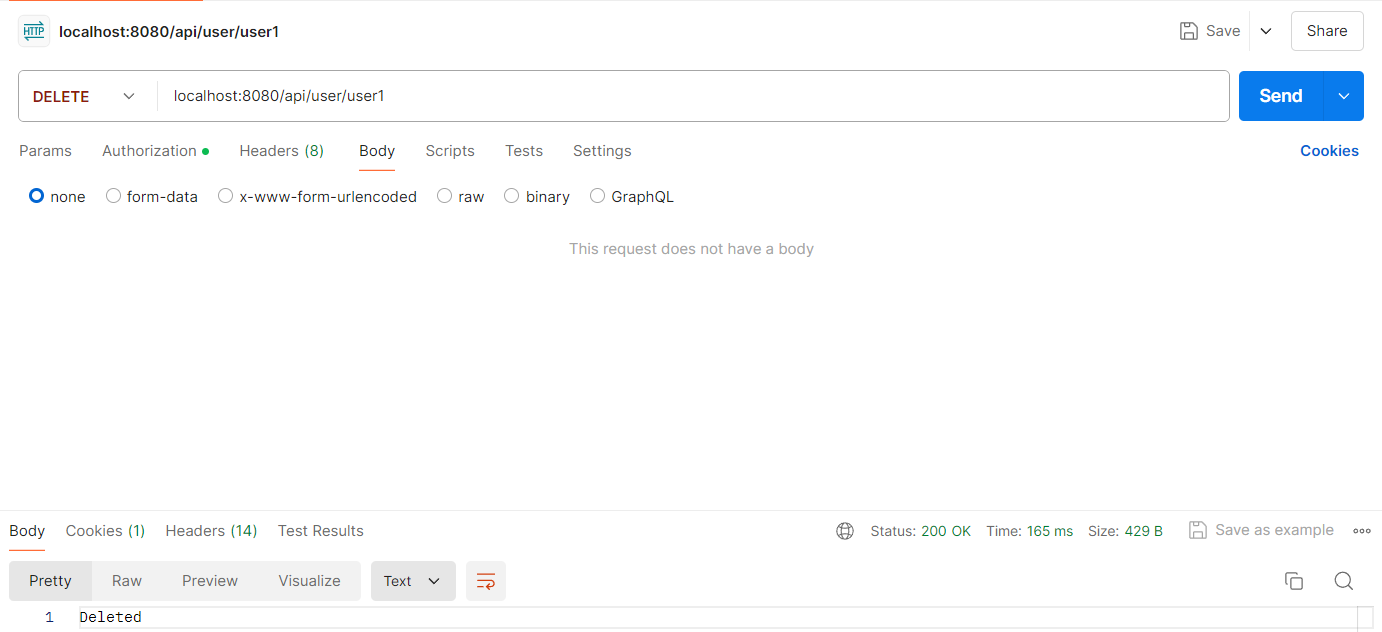


Рисунок 8.26 – Удаление пользователя

Теперь проверим работу Spring Security. Для этого в поле авторизации введём случайные значения и попробуем получить доступ к некоторым эндпоинтам.

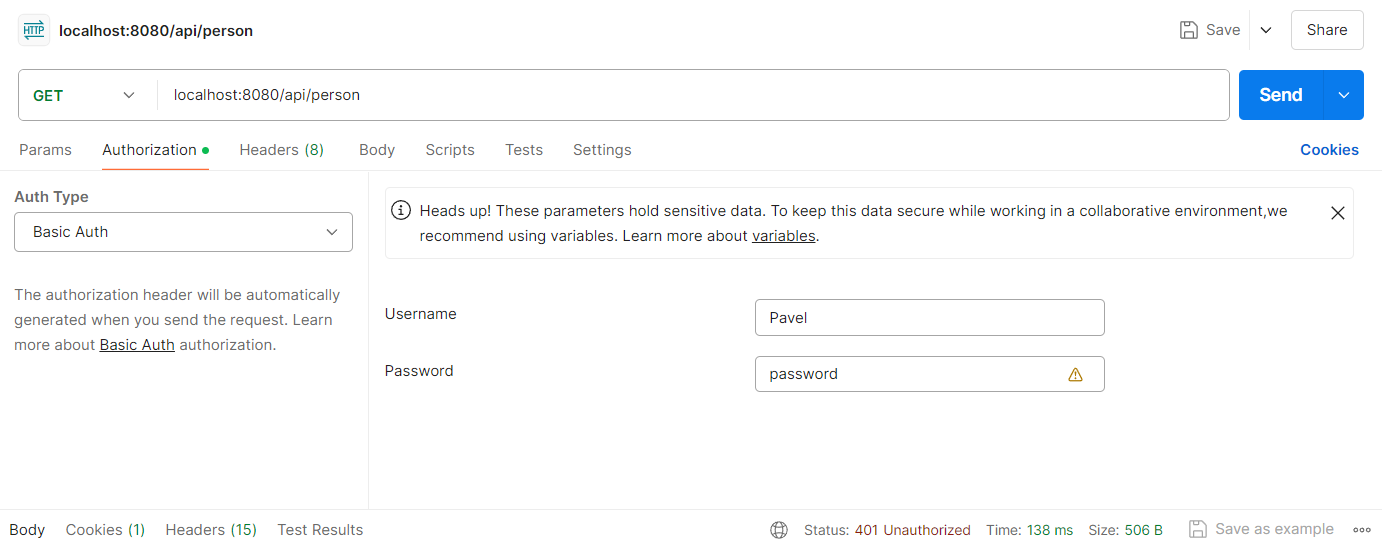


Рисунок 8.27 – Получение списка всех жильцов

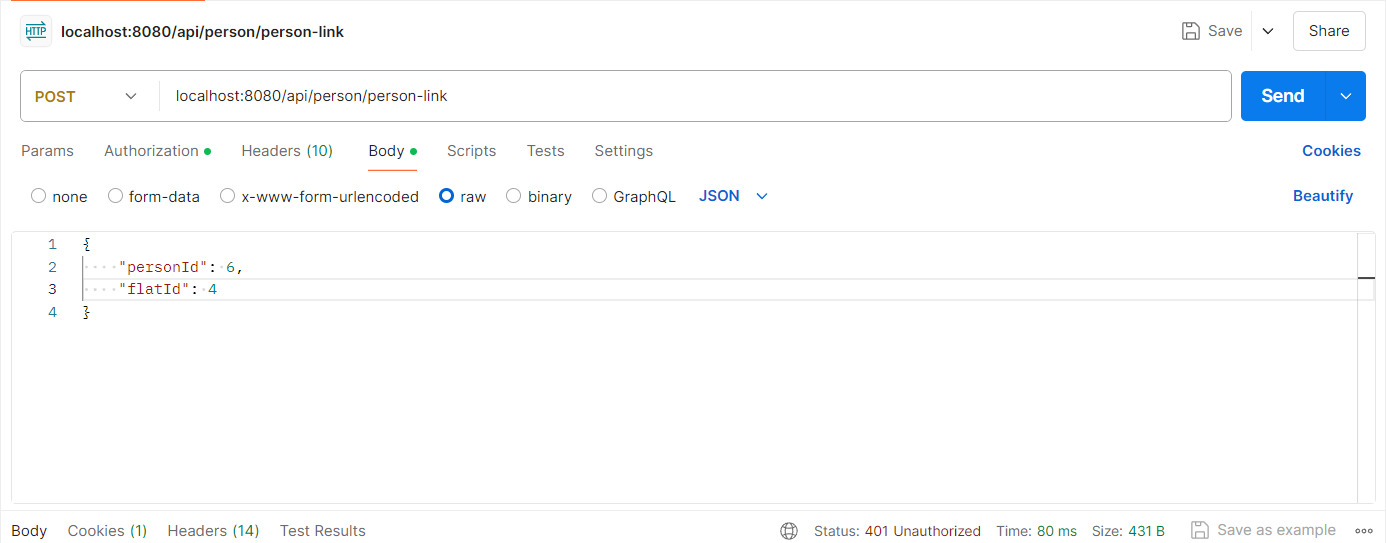


Рисунок 8.28 – Добавление связи между жильцом и квартирой

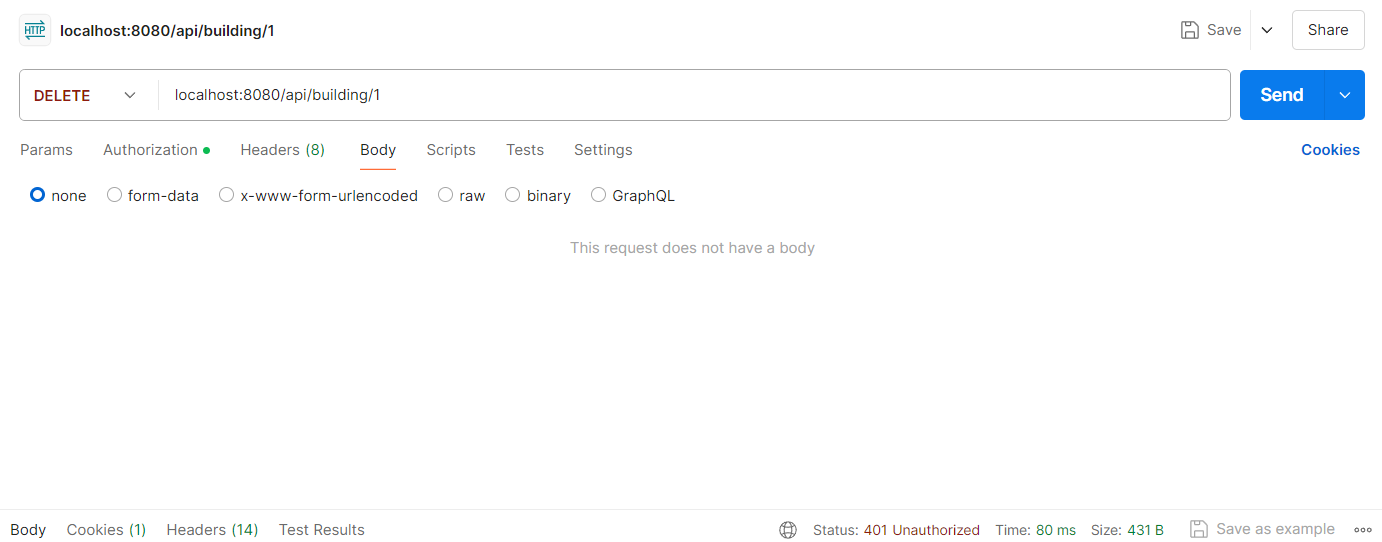


Рисунок 8.29 – Удаление здания

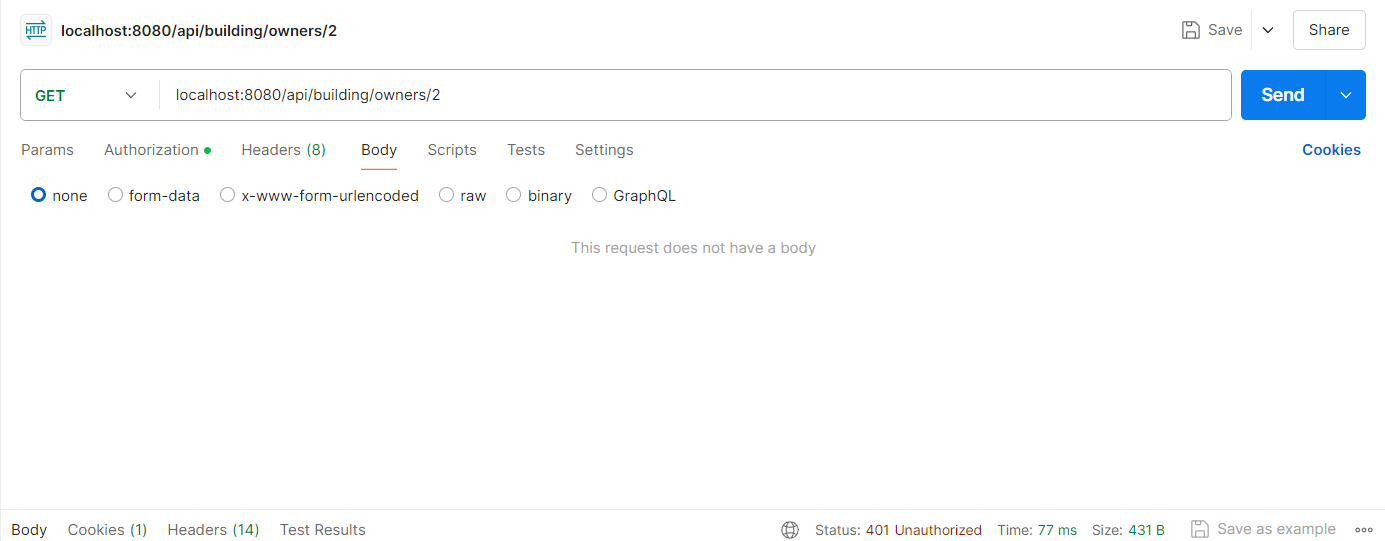


Рисунок 8.30 – Получение списка владельцев квартир в здании

Так как введённых username и password нет в базе данных, поэтому ответ сервера 401 Unauthorized.

Теперь авторизуемся как user, информация о котором уже занесена в базу данных. У него ограниченный список доступных эндпоинтов.

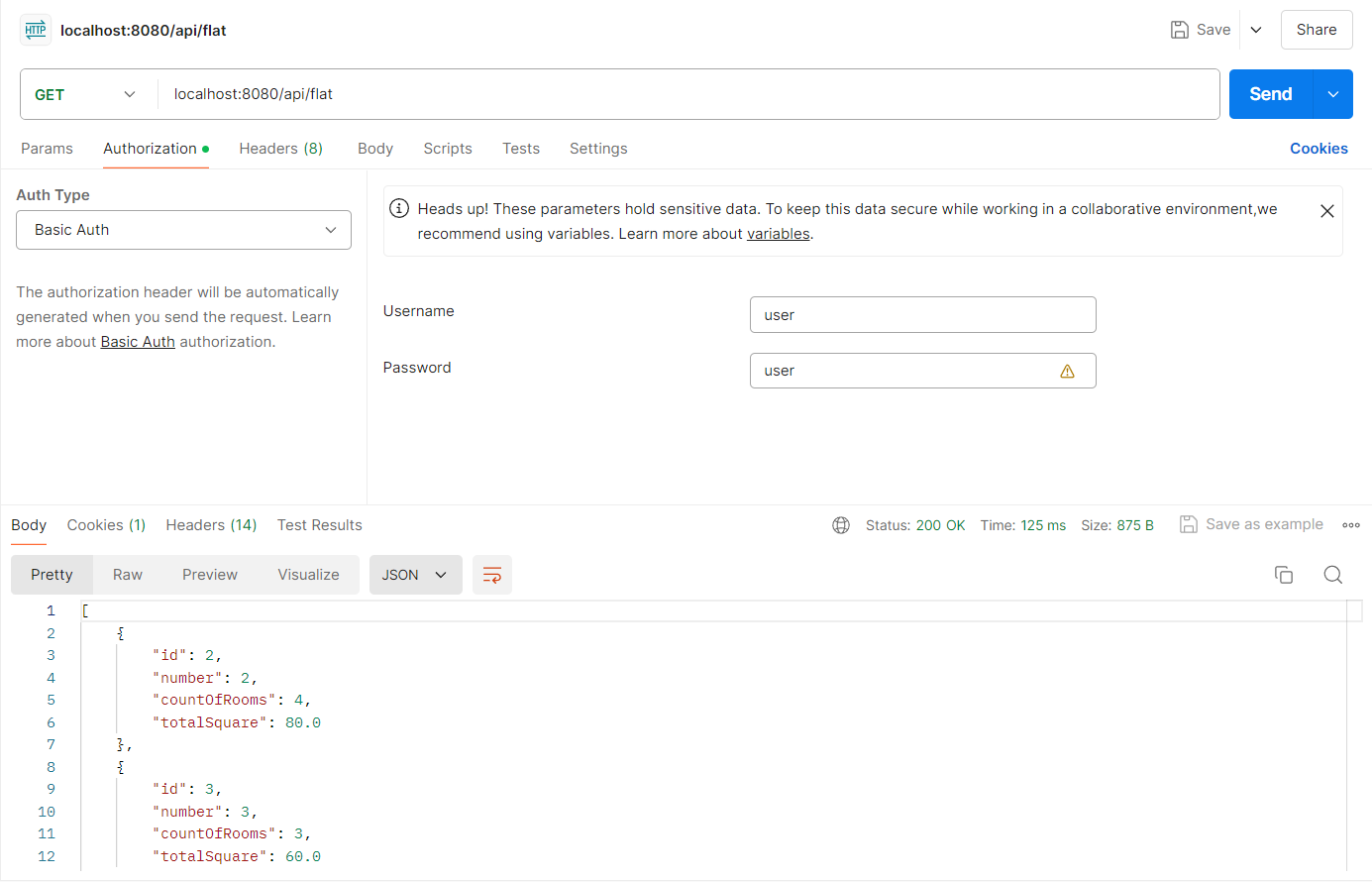


Рисунок 8.31 – Получение списка всех квартир

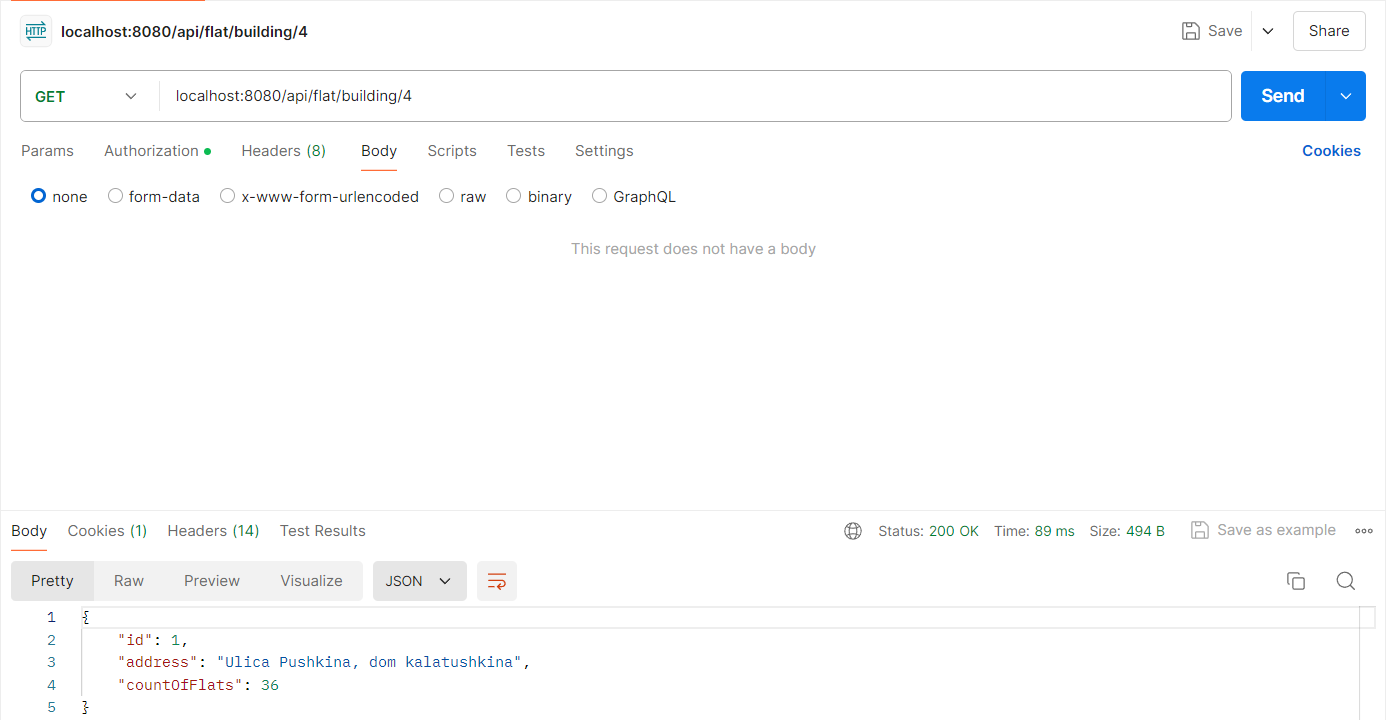


Рисунок 8.32 – Получение информации о здании, в котором находится квартира

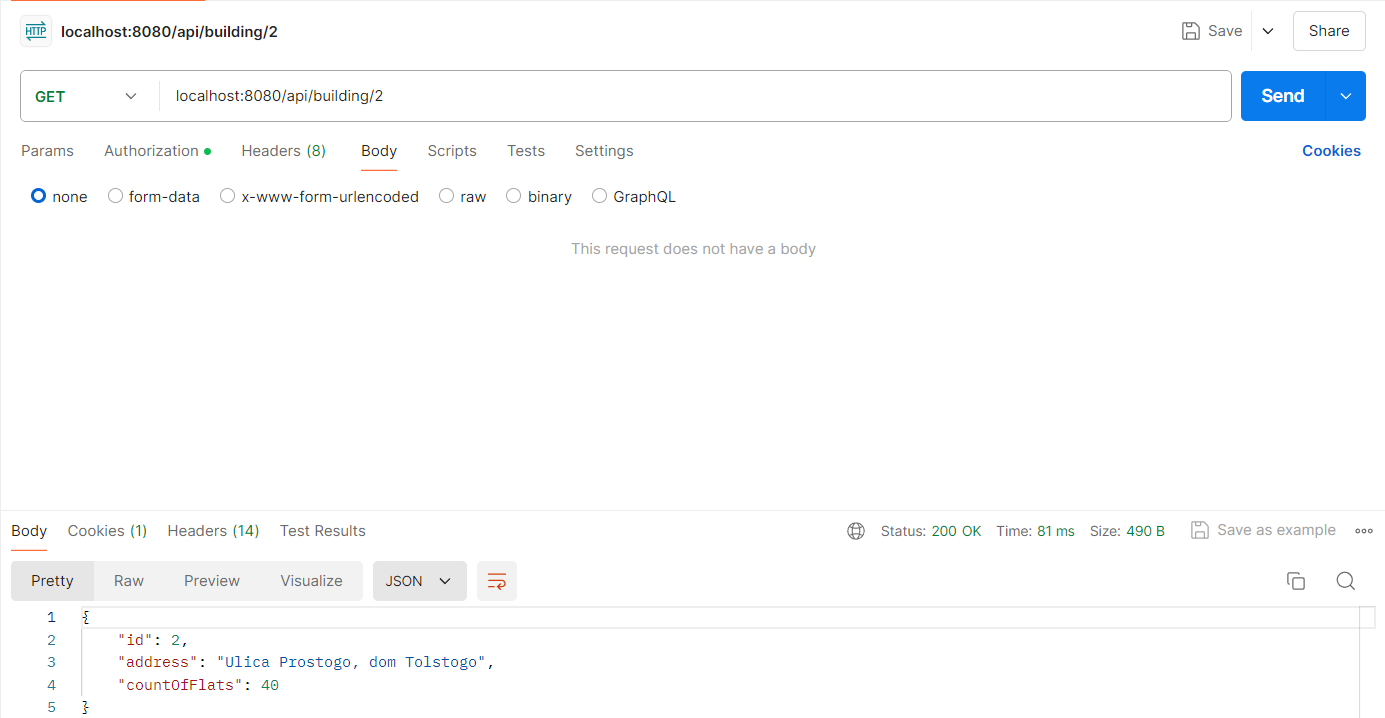


Рисунок 8.33 – Получение информации о конкретном здании

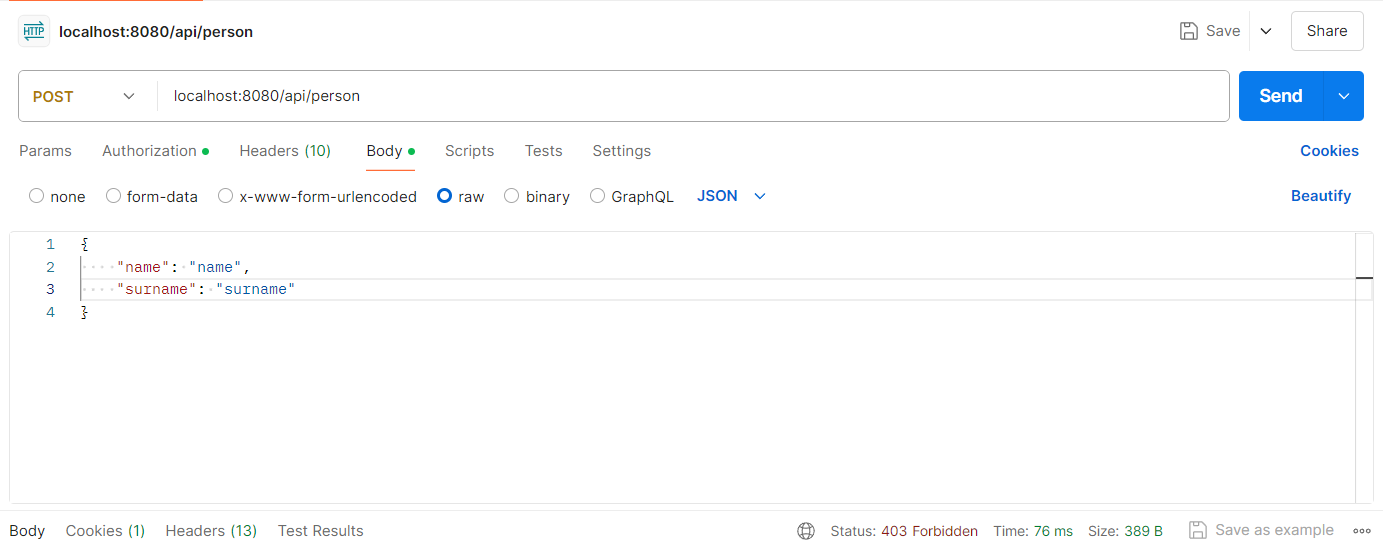


Рисунок 8.34 – Создание нового жильца

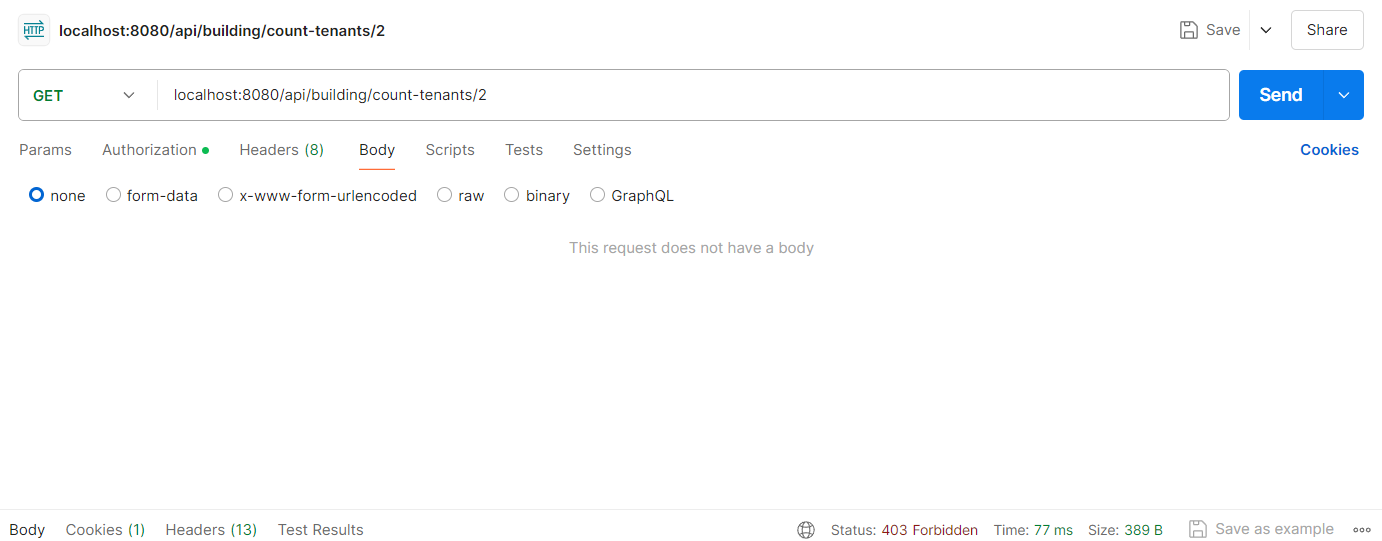


Рисунок 8.35 – Получение количества жильцов, проживающих в здании

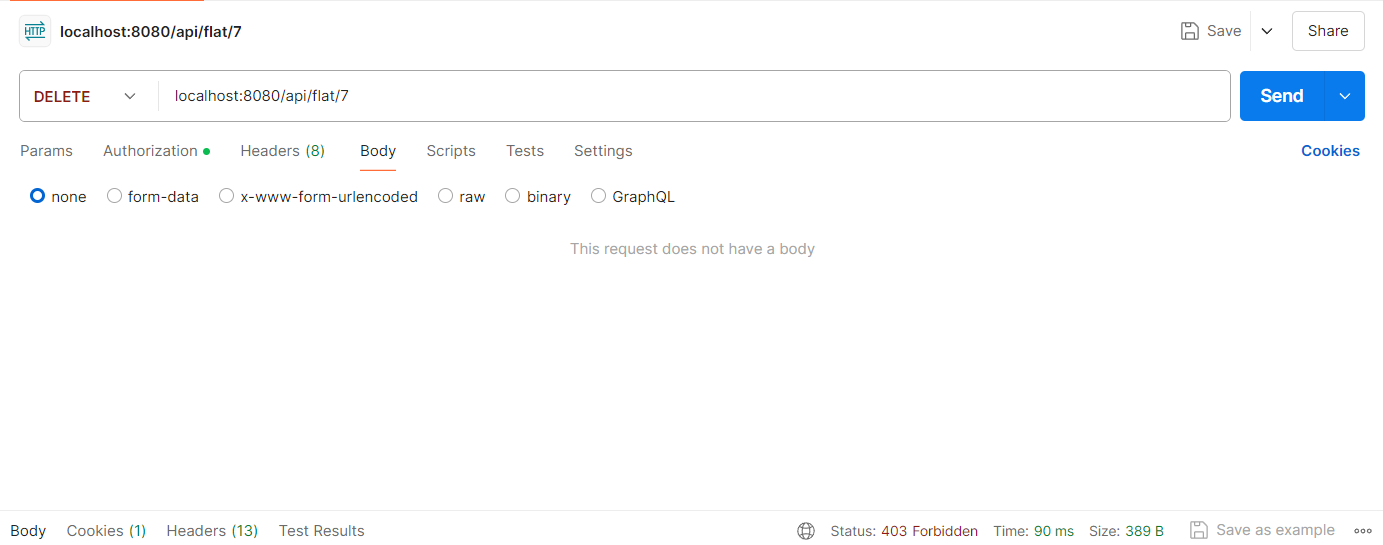


Рисунок 8.36 – Удаление квартиры

Некоторые эндпоинты выдают ответ 403 Forbidden. Это означает, что пользователь прошёл аутентификацию, но доступа к ним не имеет.

# **Заключение**

В ходе проекта был разработан REST-сервис для управления жилищно-коммунальными услугами (ЖКХ). Он обеспечивает полный набор операций для управления данными о жильцах, квартирах, зданиях и пользователях, а также обеспечивает безопасность и масштабируемость приложения. В нем использовались современные технологии и инструменты, такие как Spring Boot, Hibernate и RESTful API, Spring Security, чтобы создать эффективное и масштабируемое приложение.

Это была интересная задача, так как в современном мире большинство процессов и взаимодействий действительно происходит через интернет. Создание данного сервиса позволяет обеспечить удобство пользователей и эффективное взаимодействие с данными.