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. Теоретическая часть**

## 1.1 Java

Java – один из самых популярных и мощных языков программирования, широко используемый в различных областях разработки программного обеспечения. Он отличается высокой надежностью, переносимостью и масштабируемостью. Java обладает строгой типизацией, что обеспечивает безопасность и удобство разработки. Он поддерживает объектно-ориентированное программирование, что позволяет создавать модульные и гибкие приложения.

В области backend-разработки Java широко применяется для создания серверных приложений и веб-сервисов. С помощью Java и фреймворков, таких как Spring и Hibernate, разработчики могут создавать мощные и эффективные backend-решения. Spring Framework предоставляет инструменты для создания RESTful сервисов, управления зависимостями, обеспечения безопасности и других аспектов разработки. Hibernate позволяет работать с базами данных, обеспечивая удобство и эффективность взаимодействия с данными.

Backend-разработка на Java включает в себя проектирование и создание серверной части приложения, обработку запросов от клиентов, работу с базами данных, обеспечение безопасности и масштабируемости приложения. Java позволяет создавать высокопроизводительные и надежные backend-системы, способные обрабатывать большие объемы данных и обеспечивать стабильную работу веб-сервисов. Все это делает Java одним из наиболее предпочтительных языков для backend-разработки в современном программировании.

## 1.2 Spring Boot Framework

Spring Boot – это фреймворк, который значительно упрощает и ускоряет разработку приложений на Java. Он построен на основе популярного Spring Framework, но при этом избавляет разработчиков от рутинной работы по настройке и конфигурированию проекта.

Одним из ключевых принципов Spring Boot является "конвенция над конфигурацией". Это означает, что фреймворк сам предлагает разумные значения по умолчанию для большинства настроек, тем самым сокращая объем необходимого кода и ускоряя процесс разработки. Spring Boot автоматически настраивает компоненты Spring, такие как DataSource, JPA, Tomcat и другие, позволяя быстро запускать приложения без ручной настройки.

Еще одно важное преимущество – встроенный сервер Tomcat. Благодаря этому, приложения на Spring Boot можно развертывать как самостоятельные (standalone) программы, которые запускаются простой командой java -jar. Нет необходимости устанавливать и настраивать внешний сервер приложений.

Разработчикам также на руку приходятся многочисленные стартеры (starter), предоставляемые Spring Boot. Это зависимости Maven или Gradle, которые включают в себя все необходимые библиотеки для определенной функциональности - веб-разработки, доступа к базам данных, безопасности и т.д. Стартеры позволяют быстро добавлять нужные компоненты в проект и избавляют от рутинной настройки зависимостей.

## 1.3 Maven

Maven – это инструмент для управления сборкой и зависимостями проектов на Java. Он помогает автоматизировать процесс сборки, тестирования и развертывания приложений, значительно упрощая работу разработчиков.

Основная идея заключается в использовании стандартизированной структуры проекта и конфигурационного файла pom.xml. Этот файл содержит всю необходимую информацию о проекте, включая зависимости, плагины, профили сборки и другие настройки. Maven использует эти данные для управления жизненным циклом проекта - от компиляции исходного кода до создания дистрибутива.

Одним из его ключевых преимуществ – это управление зависимостями. Вместо ручного добавления библиотек в проект и настройки версий, Maven автоматически загружает необходимые артефакты из удаленных репозиториев, таких как Maven Central Repository. Это позволяет избежать конфликтов версий и обеспечивает согласованность сборки на разных машинах.

Также предоставляет стандартизированные фазы жизненного цикла сборки, такие как compile, test, package и deploy. Это позволяет выполнять одни и те же команды для сборки проекта на любой машине, не зависимо от используемой IDE или операционной системы.

Кроме того, поддерживает плагины, которые расширяют его функциональность. Существует множество готовых плагинов для различных задач - генерации отчетов, статического анализа кода, создания документации и т.д. Разработчики также могут создавать собственные плагины для автоматизации специфичных для проекта задач.

## 1.4 JPA

JPA (Java Persistence API) – это мощный инструмент, который помогает Java-разработчикам эффективно работать с реляционными базами данных. Он обеспечивает стандартизированный подход к отображению объектов Java на таблицы базы данных и наоборот, что упрощает взаимодействие с базой данных.

С его помощью, разработчики могут определить сущности - классы Java, которые соответствуют таблицам в базе данных. Затем он автоматически управляет жизненным циклом этих сущностей, выполняя операции CRUD (Create, Read, Update, Delete) при взаимодействии с базой данных.

JPA также предлагает удобный язык запросов JPQL (Java Persistence Query Language), который позволяет выполнять сложные запросы к базе данных, используя объектно-ориентированный подход. Работа с базами данных становится более интуитивно понятной и эффективной.

Кроме того, он обеспечивает абстракцию от конкретной реализации ORM-фреймворка, что означает, что разработчики могут легко переключаться между различными ORM-фреймворками без изменения кода приложения. Это делает JPA идеальным выбором для разработчиков, которые ищут гибкость и масштабируемость в работе с базами данных.

## 1.5 Spring Security

Spring Security – это мощный фреймворк, который обеспечивает комплексную защиту Java-приложений. Он предоставляет гибкие и настраиваемые механизмы аутентификации, авторизации и защиты от различных видов атак.

Одна из ключевых возможностей Spring Security – аутентификация пользователей. Фреймворк поддерживает множество способов аутентификации, включая форму входа, HTTP Basic, OAuth2 и другие. Разработчики могут легко настроить процесс аутентификации в соответствии с требованиями приложения.

Не менее важна и авторизация – определение прав доступа для аутентифицированных пользователей. Spring Security предоставляет гибкую систему ролей и разрешений, позволяющую точно контролировать доступ к ресурсам приложения. Разработчики могут задавать правила доступа на уровне URL, методов контроллеров или даже отдельных доменных объектов.

Кроме того, он защищает приложение от распространенных видов атак, таких как CSRF, XSS, Session Fixation и других. Фреймворк автоматически применяет меры защиты, не требуя дополнительных усилий от разработчиков.

Важно отметить, что Spring Security легко интегрируется с другими компонентами Spring, такими как Spring MVC и Spring Data. Это позволяет создавать комплексные защищенные приложения, используя единообразный подход.

В целом, он значительно упрощает реализацию безопасности в Java-приложениях. Он предоставляет надежные механизмы аутентификации и авторизации, защищает от распространенных атак и легко настраивается под требования конкретного проекта. Использование данного фреймворка позволяет разработчикам сосредоточиться на бизнес-логике, не отвлекаясь на рутинные задачи обеспечения безопасности.

## 1.6 СУБД MySQL

SQL (Structured Query Language) – это стандартный язык для работы с реляционными базами данных. Он позволяет эффективно управлять данными, выполняя операции создания, чтения, обновления и удаления (CRUD). SQL предоставляет мощные средства для фильтрации, сортировки, группировки и агрегирования данных.

MySQL – это одна из самых популярных и широко используемых систем управления реляционными базами данных (СУБД). Она отличается высокой производительностью, надежностью и простотой использования. MySQL поддерживает множество типов данных, транзакции, индексы и другие важные возможности.

Сочетание SQL и MySQL позволяет Java-разработчикам эффективно хранить и обрабатывать структурированные данные в своих приложениях. Они могут использовать SQL для выполнения запросов к базе данных и получения необходимой информации, а MySQL обеспечивает надежное и масштабируемое хранилище данных.

# **2. Описание предметной области**

Предметная область, в которой будет функционировать разрабатываемый REST-сервис, связана с управлением жилищно-коммунальными услугами (ЖКХ). Эта область включает в себя информацию о жильцах, зданиях, квартирах и владельцах квартир, а также учет данных о пользователях системы.

Основные понятия и принципы:

* Жилец (Person): представляет собой человека, проживающего в квартире. У жильца есть имя, фамилия и связь с квартирой через сущность PersonLinkFlat.
* Квартира (Flat): описывает жилое помещение, имеет номер, количество комнат, общую площадь и связь с зданием, в котором находится, через сущность Building.
* Здание (Building): представляет собой строение, включающее адрес и количество квартир. Здание содержит список квартир и устанавливает связь с ними.
* Пользователь (User): представляет собой пользователя системы, имеющего уникальное имя пользователя, пароль и уровень доступа (роль).

В предметной области ЖКХ основными принципами являются учет информации о жильцах, квартирах и зданиях, управление связями между ними, а также обеспечение безопасности и доступа к системе через учетные записи пользователей.

Разрабатываемый REST-сервис будет обеспечивать возможность управления данными о жильцах, квартирах, зданиях и пользователях, а также предоставлять функционал для установления связей между ними. Это позволит эффективно управлять жилищно-коммунальными услугами, обеспечивая удобство и прозрачность взаимодействия с данными и пользователями системы.

# **3. DTO**

DTO (Data Transfer Object) – это объекты, которые используются для передачи данных между слоями приложения или между приложением и внешним миром. Они представляют собой удобный способ инкапсуляции данных и обеспечивают структурированный способ передачи информации.

DTO используются для передачи данных между различными компонентами приложения, такими как контроллеры, сервисы и слой доступа к данным. Они позволяют изолировать внутреннюю структуру данных от внешних интерфейсов, обеспечивая гибкость и безопасность взаимодействия. Рассмотрим их в контексте REST-сервиса для управления ЖКХ:

PersonDTO:

* Представляет объект для передачи данных о жильце (Person).
* Содержит поля для идентификатора, имени и фамилии жильца.
* Конструктор принимает объект Person и инициализирует соответствующие поля.

FlatDTO:

* Используется для передачи информации о квартире (Flat).
* Включает атрибуты для идентификатора, номера квартиры, количества комнат и общей площади.
* Конструктор принимает объект Flat и устанавливает соответствующие значения.

BuildingDTO:

* Предназначен для передачи данных о здании (Building).
* Содержит поля для идентификатора, адреса и количества квартир в здании.
* Конструктор инициализирует атрибуты на основе объекта Building.

DTO облегчают передачу данных между различными частями приложения, упрощая взаимодействие и обеспечивая структурированный способ обмена информацией. Они помогают избежать избыточности данных, улучшают производительность и обеспечивают четкую структуру данных в приложении.

# **4. Маппинг**

Маппинг в контексте REST-сервисов – это процесс сопоставления данных и операций с ресурсами и действиями HTTP. Он позволяет устанавливать соответствие между структурой данных и URL-адресами, определяя, какие данные передаются и какие операции выполняются при обращении к определенным эндпоинтам.

Концепция маппинга в контексте REST-сервисов заключается в том, что каждый ресурс (например, объект, коллекция или действие) должен иметь уникальный идентификатор в виде URI (Uniform Resource Identifier). Клиенты используют эти URI для доступа к необходимым ресурсам, выполняя стандартные HTTP-методы (GET, POST, PUT, DELETE) для выполнения соответствующих операций. Например, запрос GET к эндпоинту /users возвращает список пользователей, а запрос POST к /users создает нового пользователя. Для различных типов данных маппинг может включать преобразование объектов Java в формат JSON или XML для передачи данных через сеть.

Рассмотрим маппинг для различных типов данных и операций в контексте REST-сервиса для управления ЖКХ:

Управление информацией о квартирах (FlatController) URI: /api/flat:

* GET /api/flat – получение списка всех квартир;
* POST /api/flat – создание новой квартиры;
* GET /api/flat/{id} – получение информации о конкретной квартире;
* PUT /api/flat – обновление информации о квартире;
* DELETE /api/flat/{id} – удаление квартиры;
* GET /api/flat/flat-persons/{id} – получение списка жильцов, проживающих в квартире;
* GET /api/flat/building/{id} – получение информации о здании, в котором находится квартира;
* GET /api/flat/count-tenants/{id} – получение количества жильцов, проживающих в квартире.

Основные принципы маппинга в REST-сервисах включают использование именованных ресурсов (nouns) вместо глаголов, применение стандартных HTTP-методов для CRUD-операций, использование иерархической структуры URI, возвращение соответствующих HTTP-статусов и предоставление возможности фильтрации, сортировки и пагинации списков ресурсов.

Следование этим принципам маппинга позволяет создавать REST-сервисы, которые обладают высокой степенью согласованности, масштабируемости и удобства использования. Кроме того, он должен учитывать особенности предметной области и требования бизнес-логики, обеспечивая удобный доступ к данным.

Маппинг является ключевым аспектом разработки REST-сервисов, определяющим их интерфейс и взаимодействие с клиентами. Следование принципам маппинга, а также учет специфики предметной области, позволяет создавать высококачественные и интуитивно понятные REST-сервисы.

# **5. Описание точек доступа**

Точки доступа (endpoints) являются ключевым элементом в архитектуре REST-сервисов. Они представляют собой URL-адреса, к которым клиенты обращаются для взаимодействия с сервисом, выполняя различные операции над ресурсами. Правильное проектирование и описание точек доступа играет важную роль в обеспечении согласованности, масштабируемости и удобства использования REST-сервиса.

В REST-сервисах они определяют, как клиенты могут взаимодействовать с ресурсами, предоставляемыми сервисом. Каждая точка доступа соответствует определенному ресурсу или набору ресурсов и поддерживает один или несколько HTTP-методов для выполнения операций над этими ресурсами. Рассмотрим на примере:

PersonController:

* GET /api/person – получение списка всех жильцов;
* POST /api/person – создание нового жильца;
* GET /api/person/{id} – получение информации о конкретном жильце;
* PUT /api/person – обновление информации о жильце;
* DELETE /api/person/{id} – удаление жильца;
* POST /api/person/person-link – добавление связи между жильцом и квартирой;
* GET /api/person/person-flats/{id} – получение списка квартир, принадлежащих жильцу;
* GET /api/person/count-flats/{id} – получение количества квартир, принадлежащих жильцу.

Эти точки доступа обеспечивают полный набор операций для управления данными о жильцах REST-сервисе. Они следуют принципам проектирования, обеспечивая согласованность, безопасность и удобство использования.

# **6. 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.

Сервис и его структура

Архитектура сервиса будет выглядеть следующим образом:

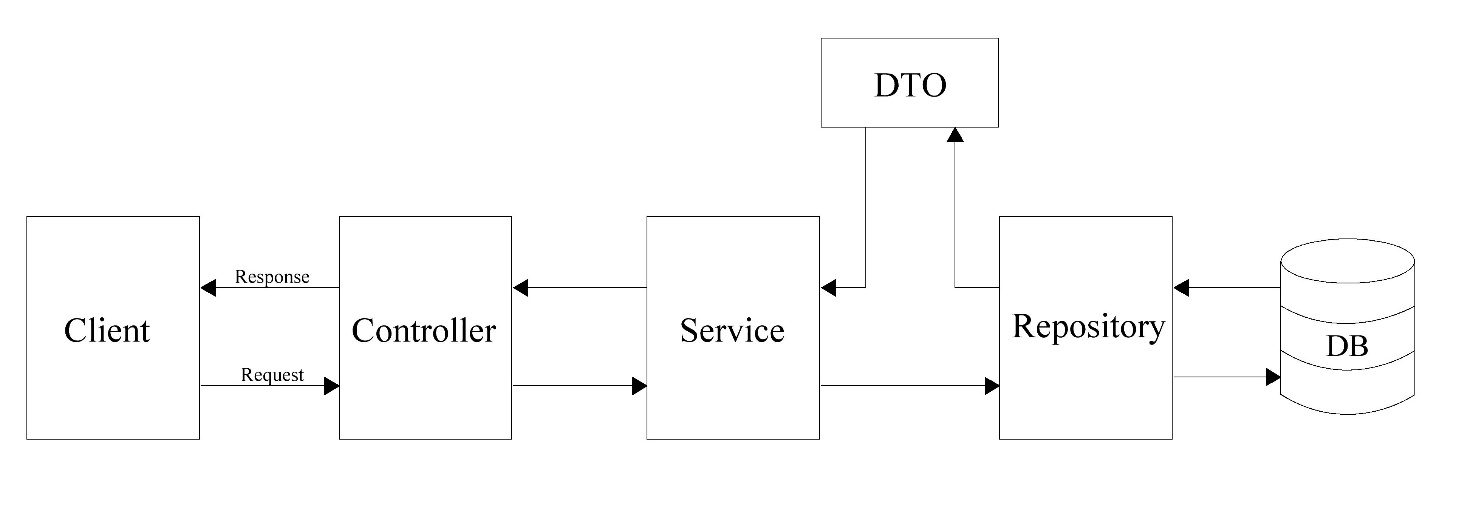


Рисунок 6.1 – Архитектура сервиса

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.

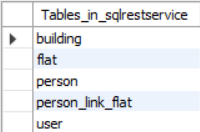


Рисунок 6.2 – Все таблицы в базе данных



Рисунок 6.3 – Таблица person



Рисунок 6.4 – Таблица flat



Рисунок 6.5 – Таблица building



Рисунок 6.6 – Таблица person\_link\_flat



Рисунок 6.7 – Таблица user

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.

Далее создаём 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.

Далее 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).

Далее 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.

Функционал REST-сервиса:

1. Управление информацией о жильцах:

* Получение списка всех жильцов – /api/person;
* Создание нового жильца – /api/person;
* Получение информации о конкретном жильце – /api/person/{id};
* Обновление информации о жильце – /api/person;
* Удаление жильца – /api/person/{id};
* Добавление связи между жильцом и квартирой – /api/person/person-link;
* Удаление связи между жильцом и квартирой – /api/person/person-link;
* Получение списка квартир, принадлежащих жильцу – /api/person/person-flats/{id};
* Получение количества квартир, принадлежащих жильцу – /api/person/count-flats/{id}.

1. Управление информацией о квартирах:

* Получение списка всех квартир – /api/flat;
* Создание новой квартиры – /api/flat;
* Получение информации о конкретной квартире – /api/flat/{id};
* Обновление информации о квартире – /api/flat;
* Удаление квартиры – /api/flat/{id};
* Получение списка жильцов, проживающих в квартире – /api/flat/flat-persons/{id};
* Получение информации о здании, в котором находится квартира – /api/flat/building/{id};
* Получение количества жильцов, проживающих в квартире – /api/flat/count-tenants/{id}.

1. Управление информацией о зданиях:

* Получение списка всех зданий – /api/building;
* Создание нового здания – /api/building;
* Получение информации о конкретном здании – /api/building/{id};
* Обновление информации о здании – /api/building;
* Удаление здания – /api/building/{id};
* Получение количества жильцов, проживающих в здании – /api/building/count-tenants/{id};
* Получение списка владельцев квартир в здании – /api/building/owners/{id}.

1. Управление информацией о пользователях:

* Создание нового пользователя – /api/user;
* Удаление пользователя – /api/user/{username}.

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.

# **8. Тестирование приложения**

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

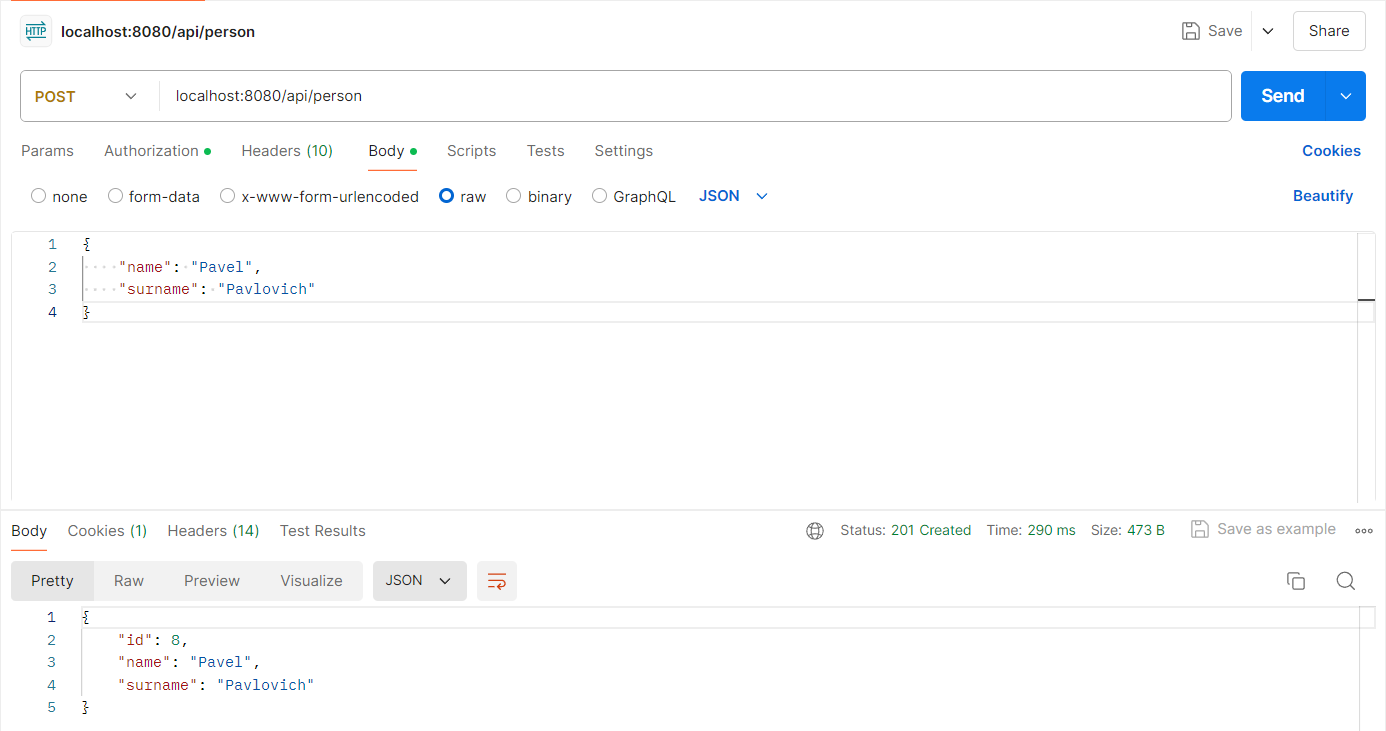


Рисунок 8.1 - Создание нового жильца

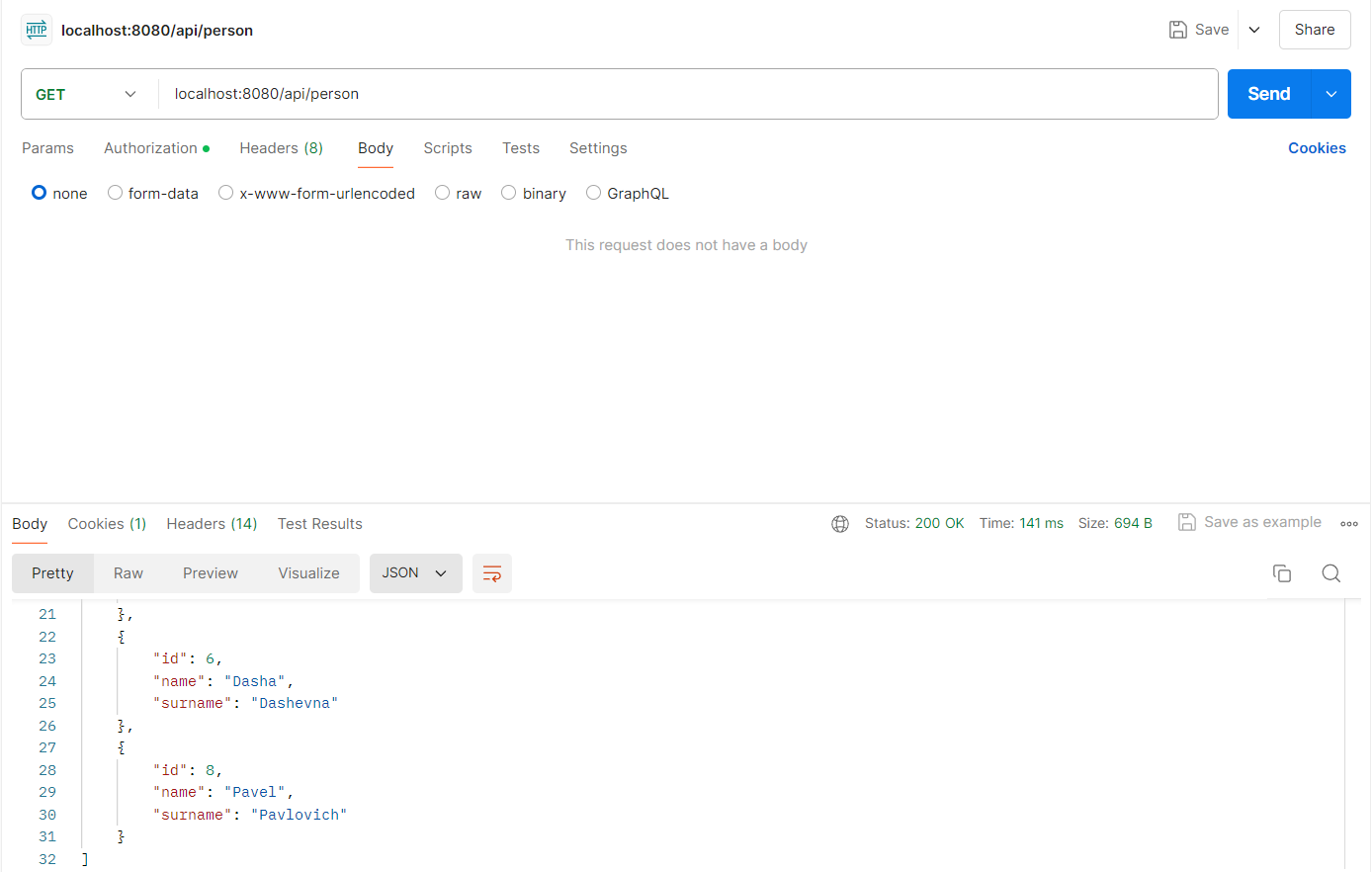


Рисунок 8.2 – Получение списка всех жильцов

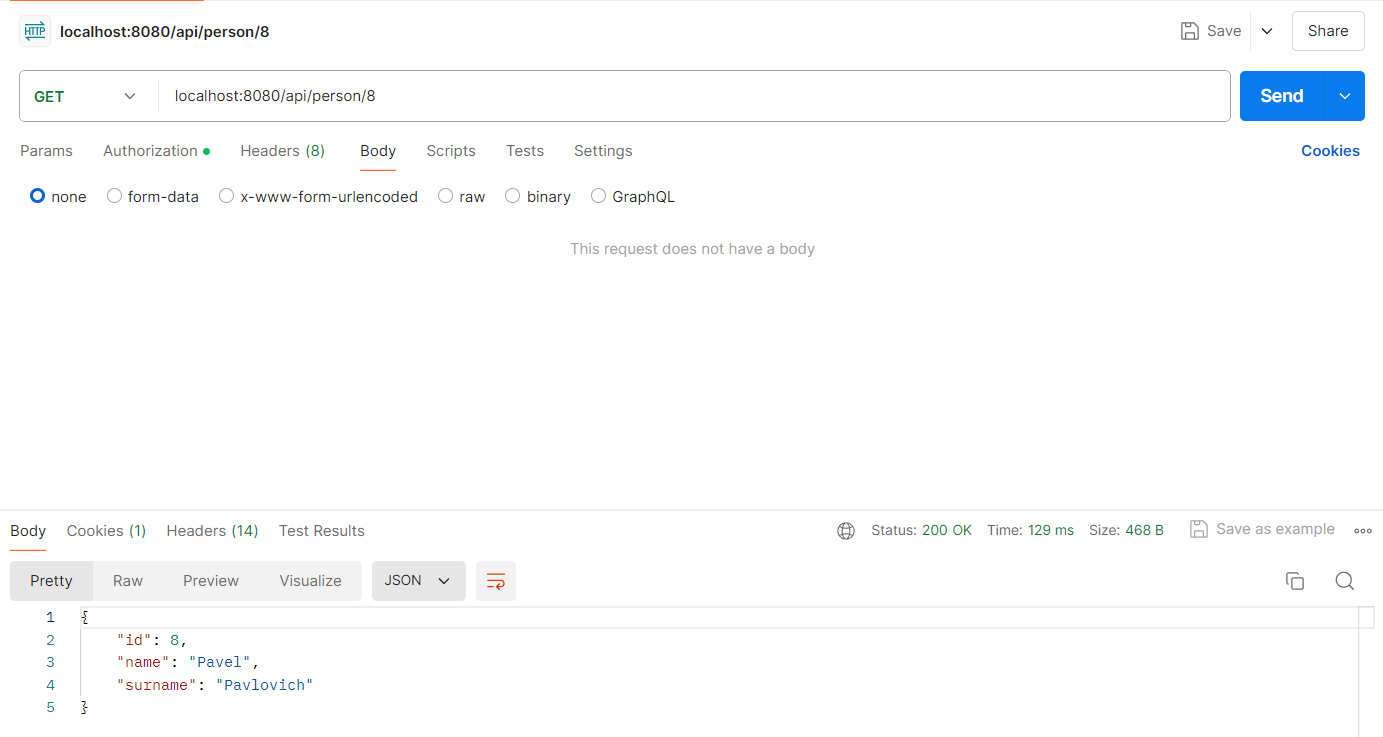


Рисунок 8.3 – Получение информации о конкретном жильце

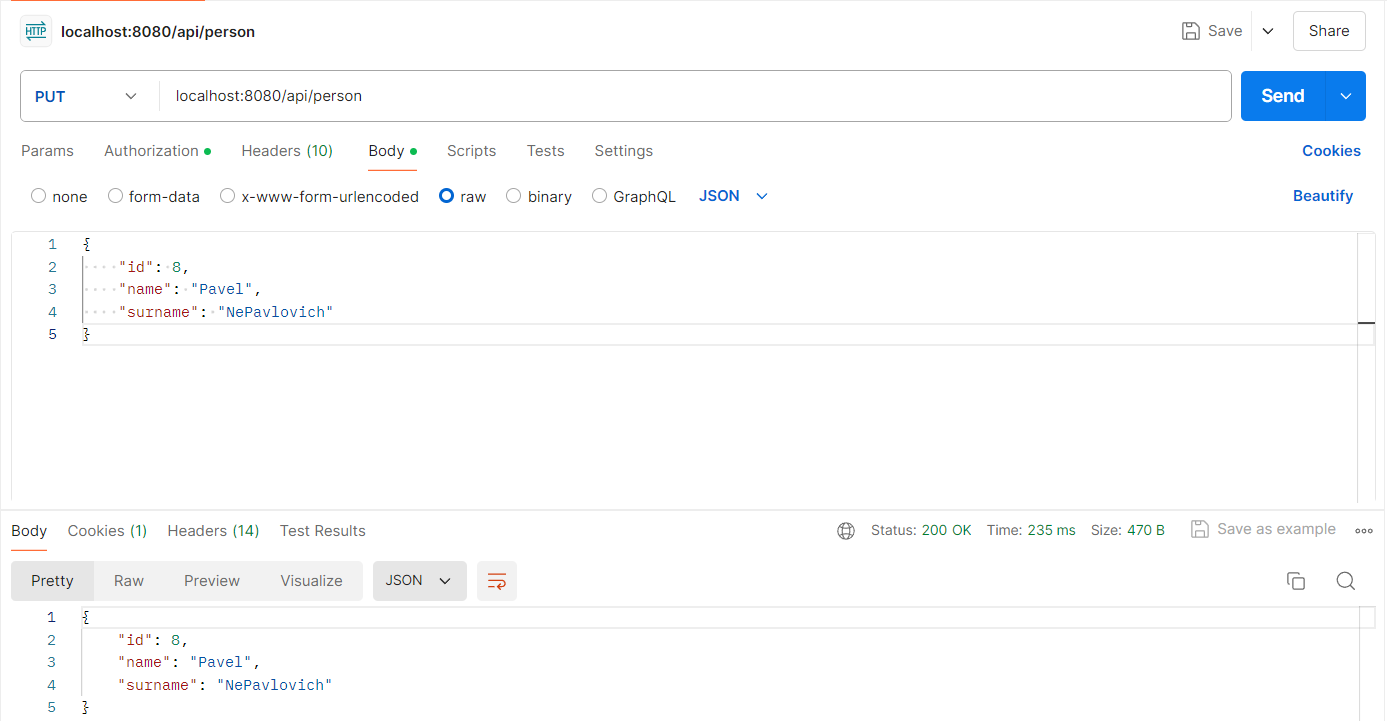


Рисунок 8.4 – Обновление информации о жильце

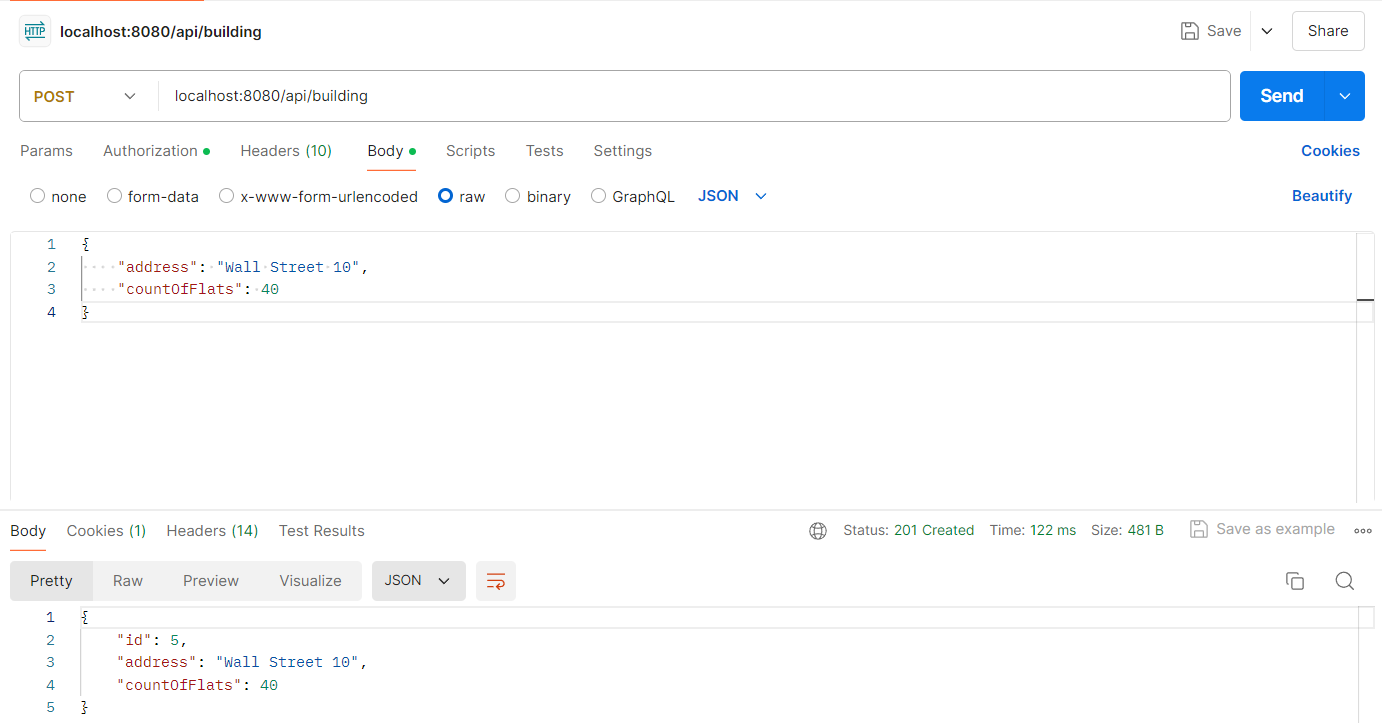


Рисунок 8.5 – Создание нового здания

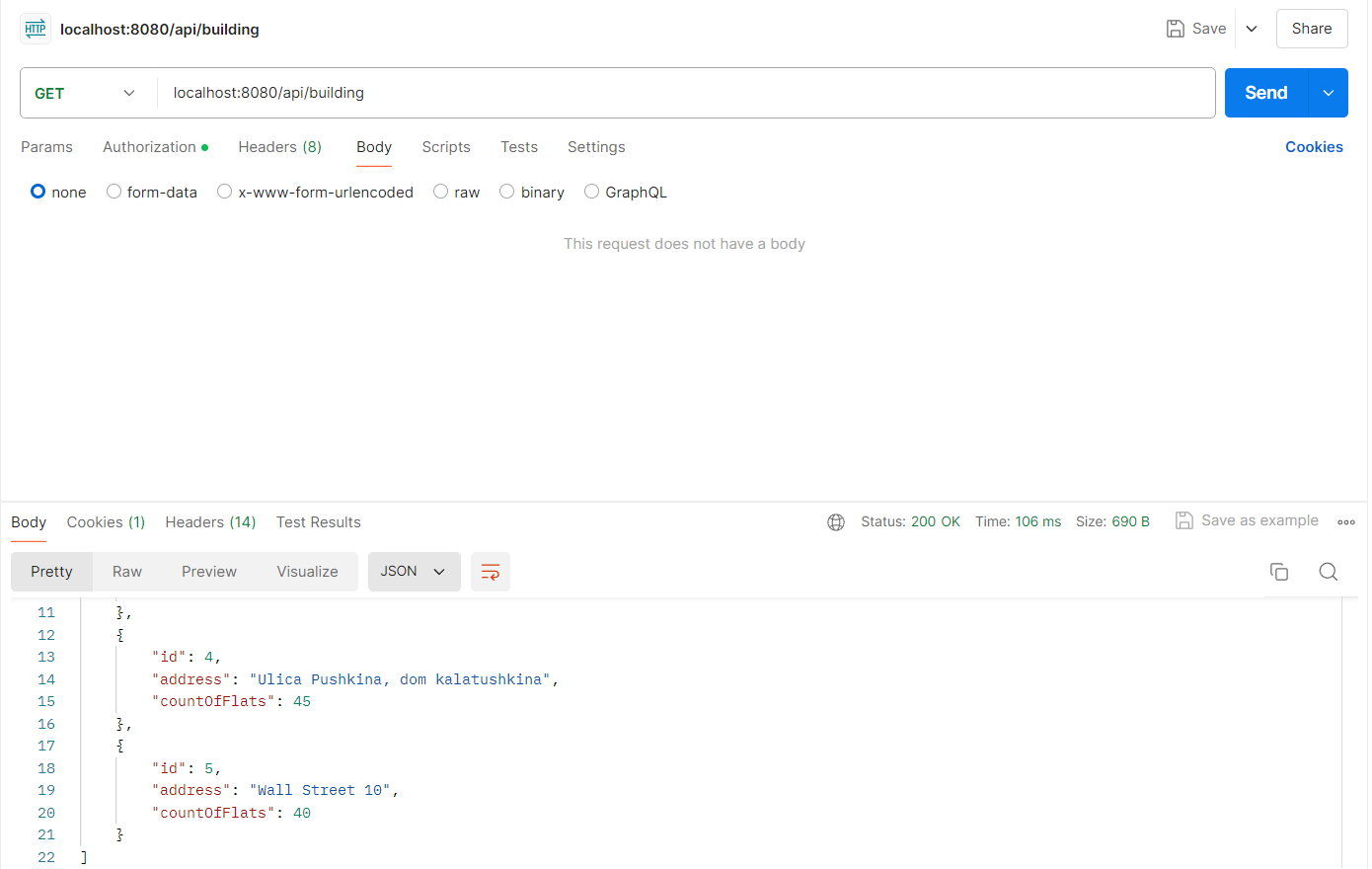


Рисунок 8.6 – Получение списка всех зданий

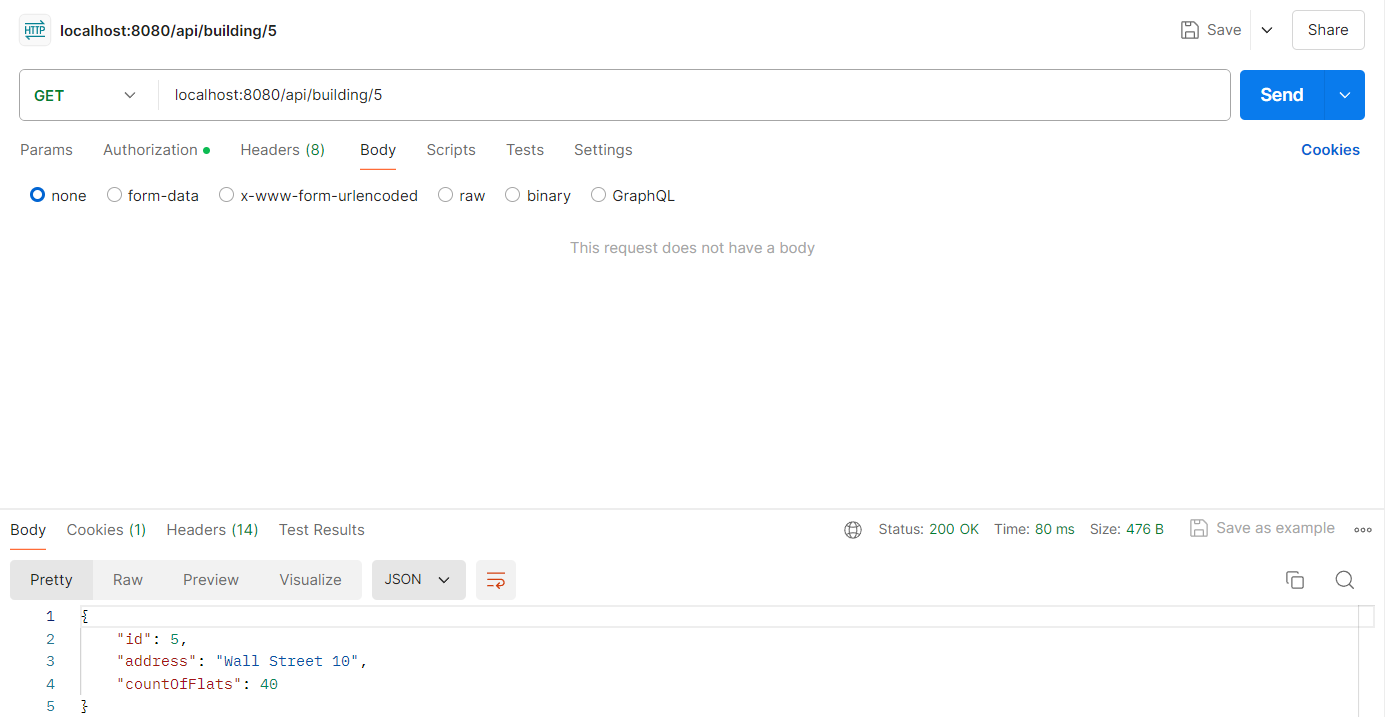


Рисунок 8.7 – Получение информации о конкретном здании

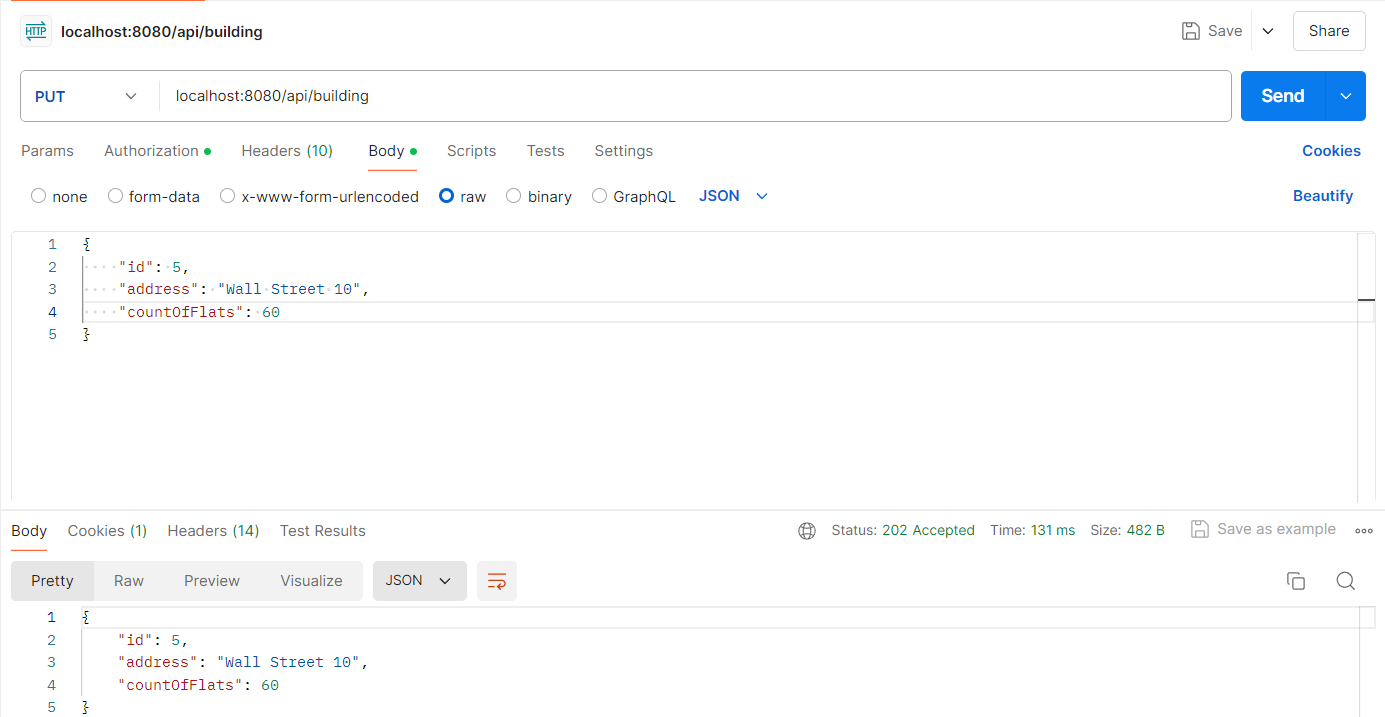


Рисунок 8.8 – Обновление информации о здании

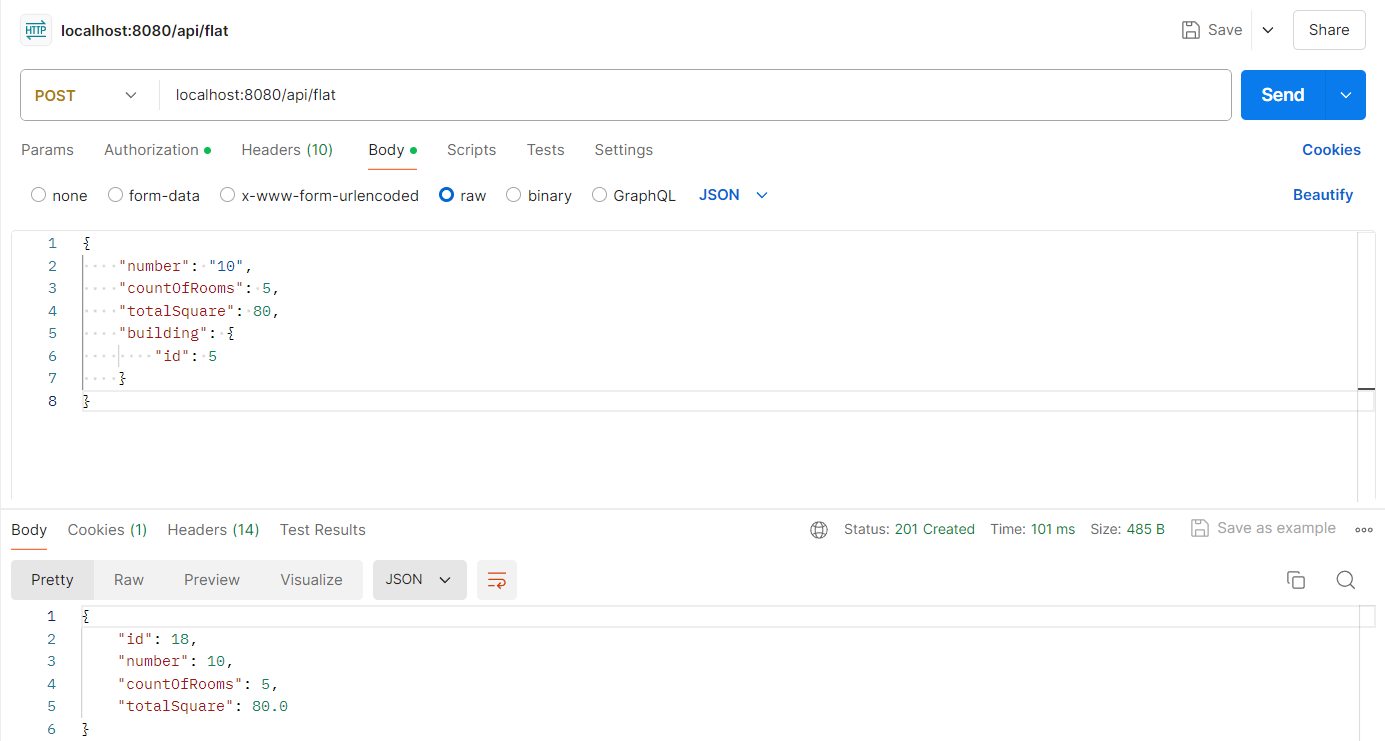


Рисунок 8.9 – Создание новой квартиры

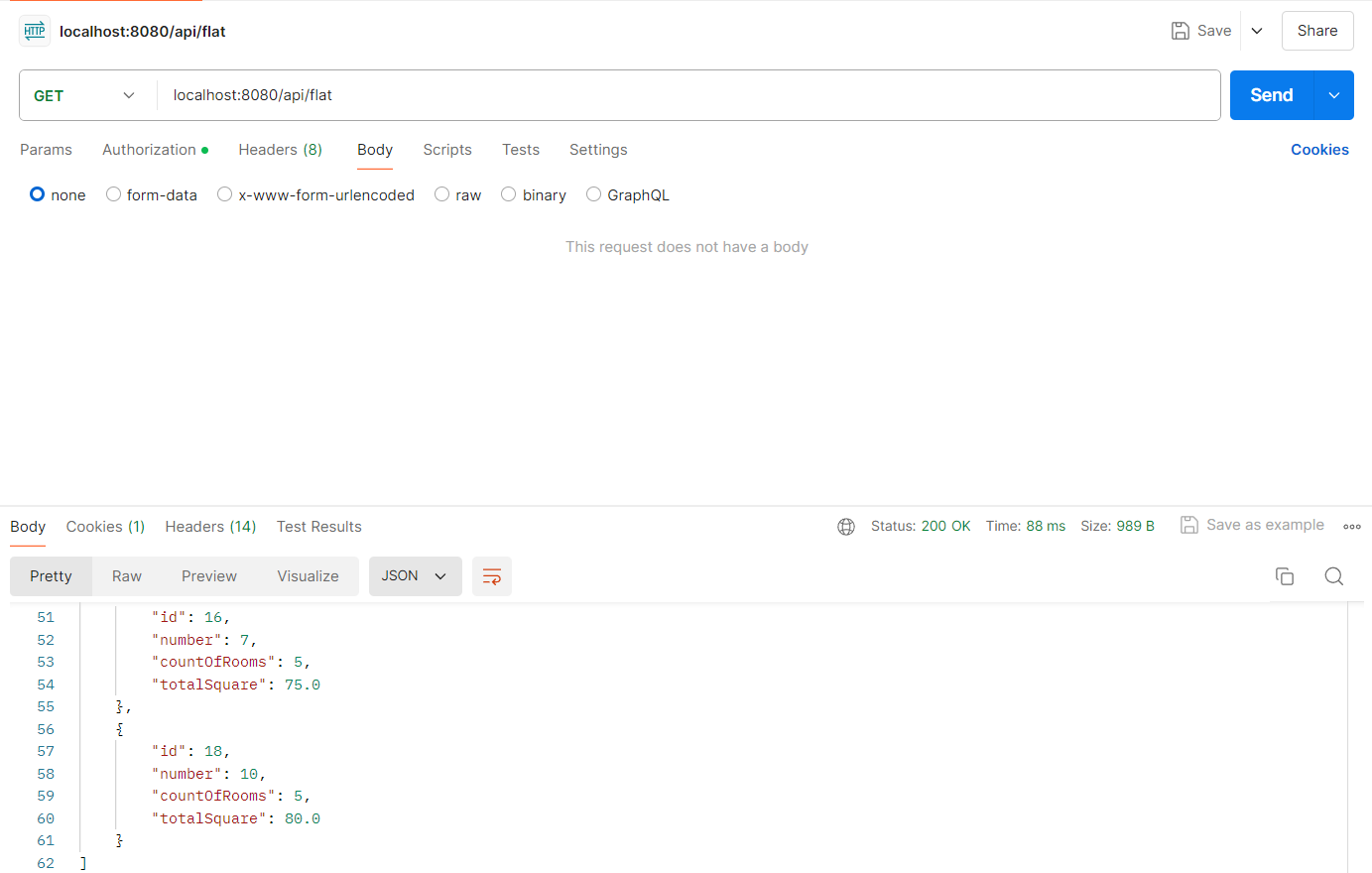


Рисунок 8.10 – Получение списка всех квартир

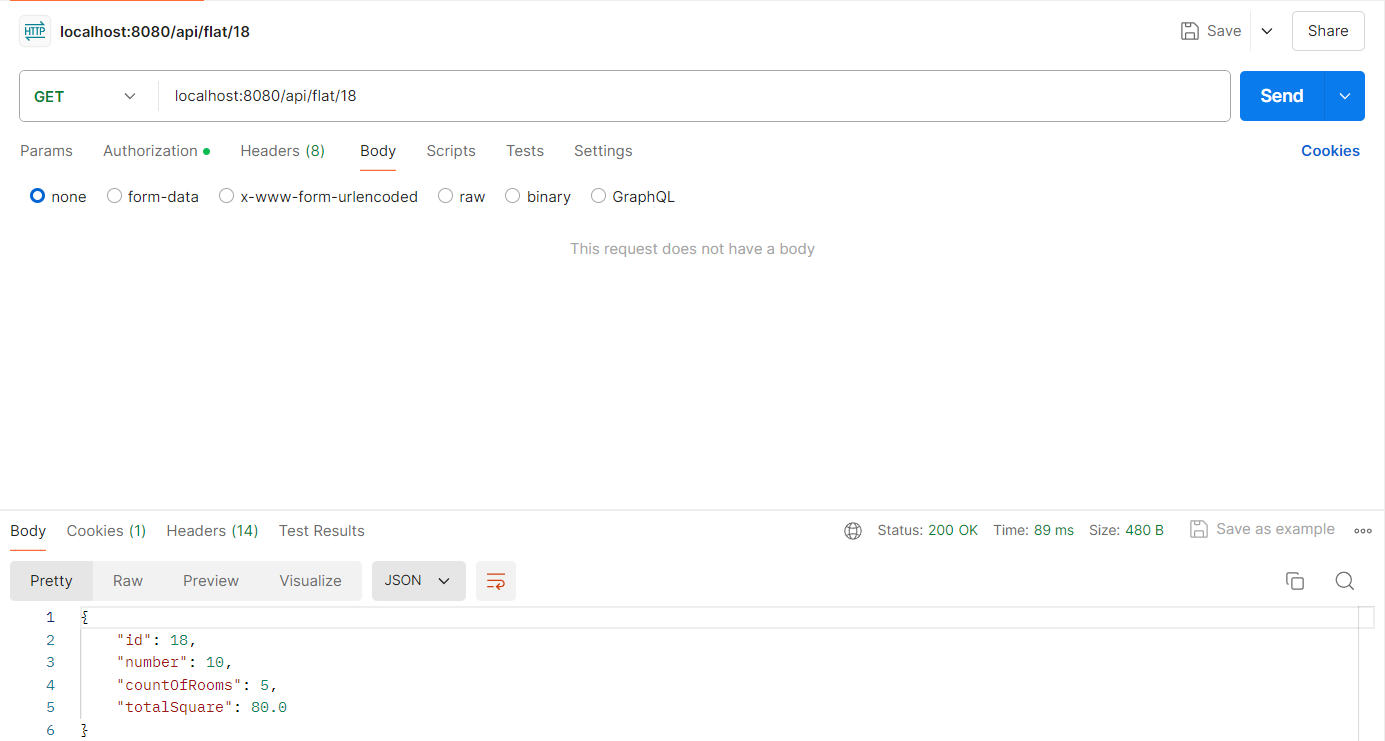


Рисунок 8.11 – Получение информации о конкретной квартире

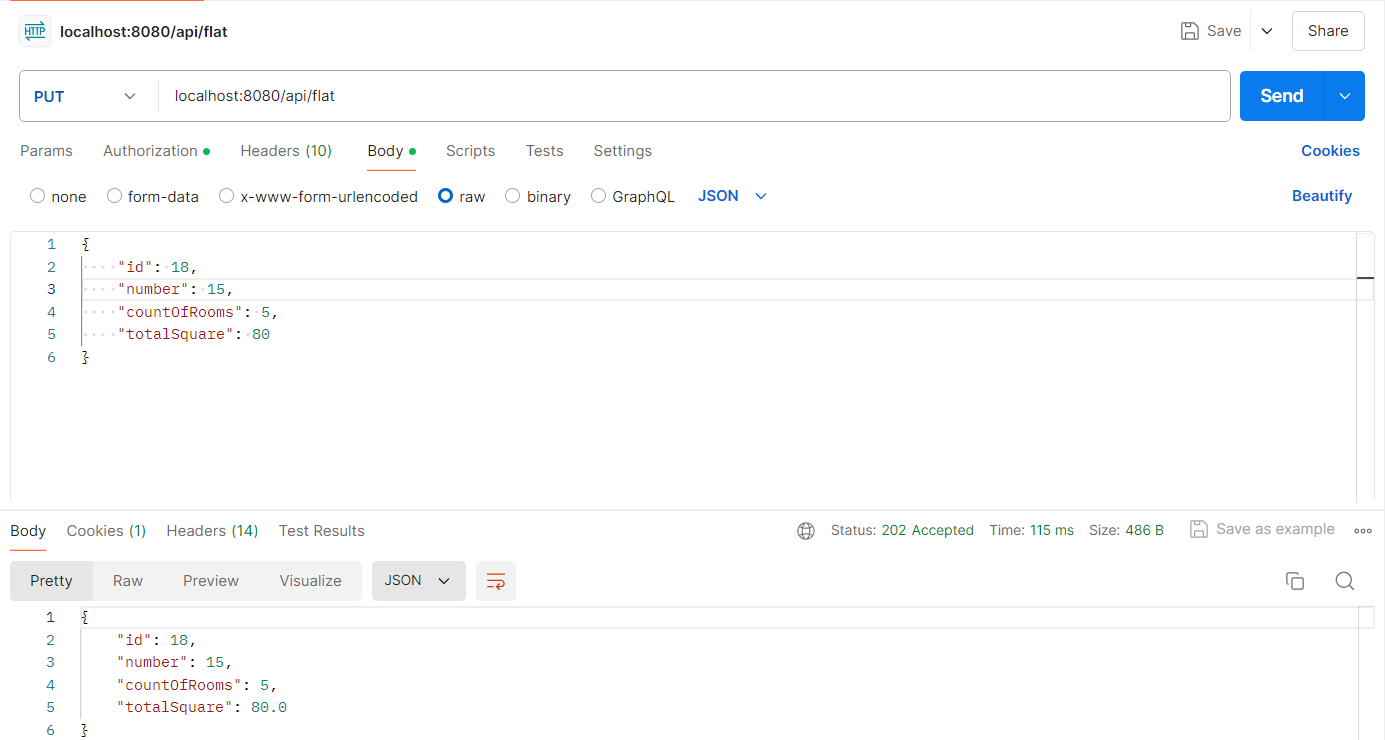


Рисунок 8.12 – Обновление информации о квартире

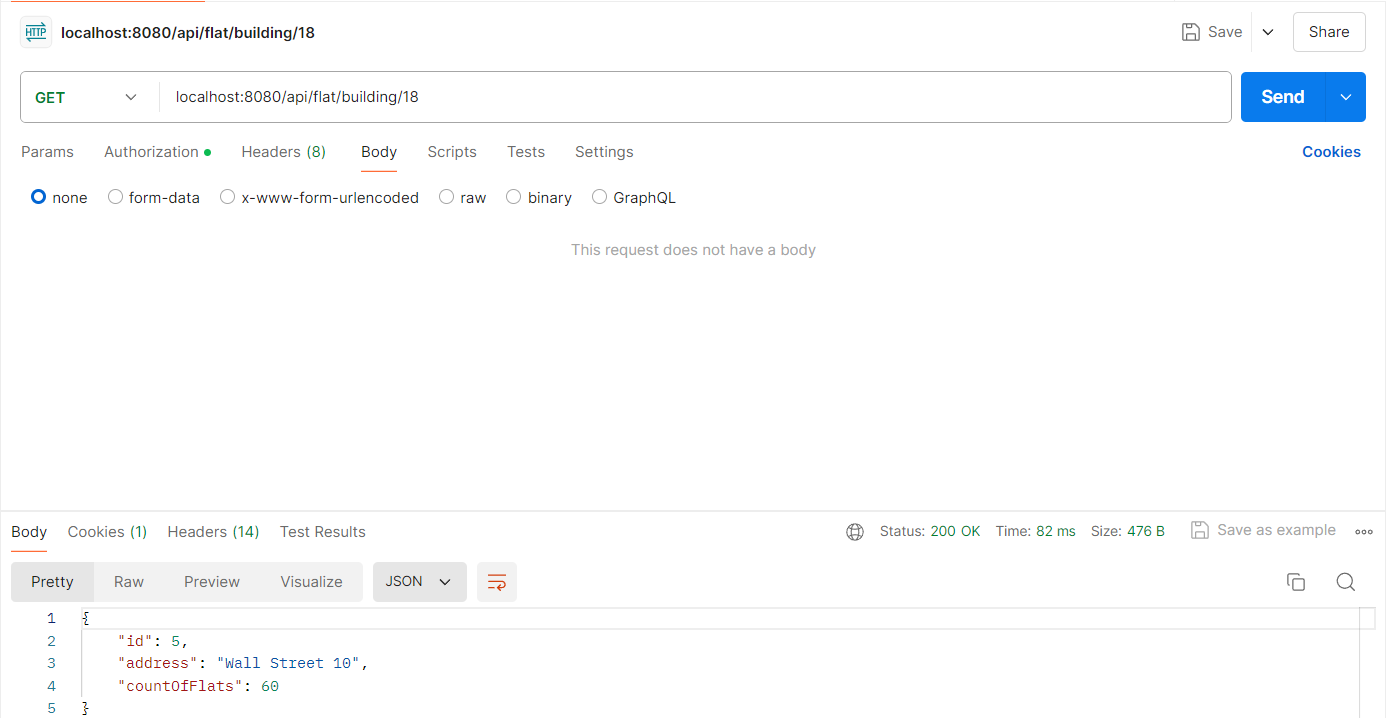


Рисунок 8.13 – Получение информации о здании, в котором находится квартира

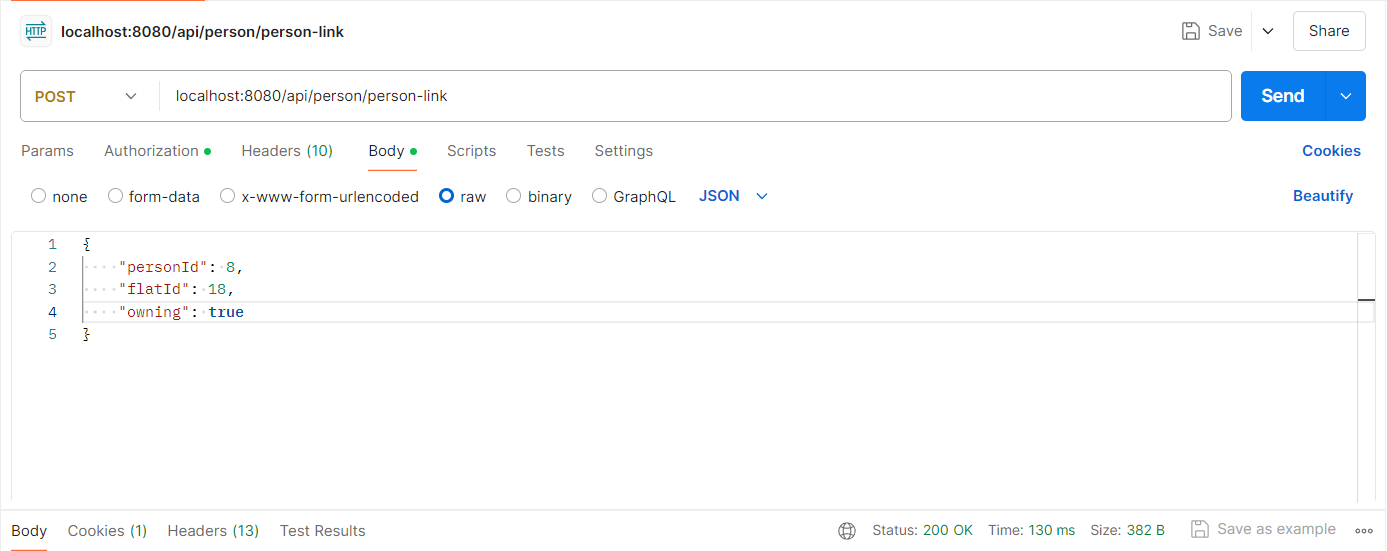


Рисунок 8.14 – Добавление связи между жильцом и квартирой

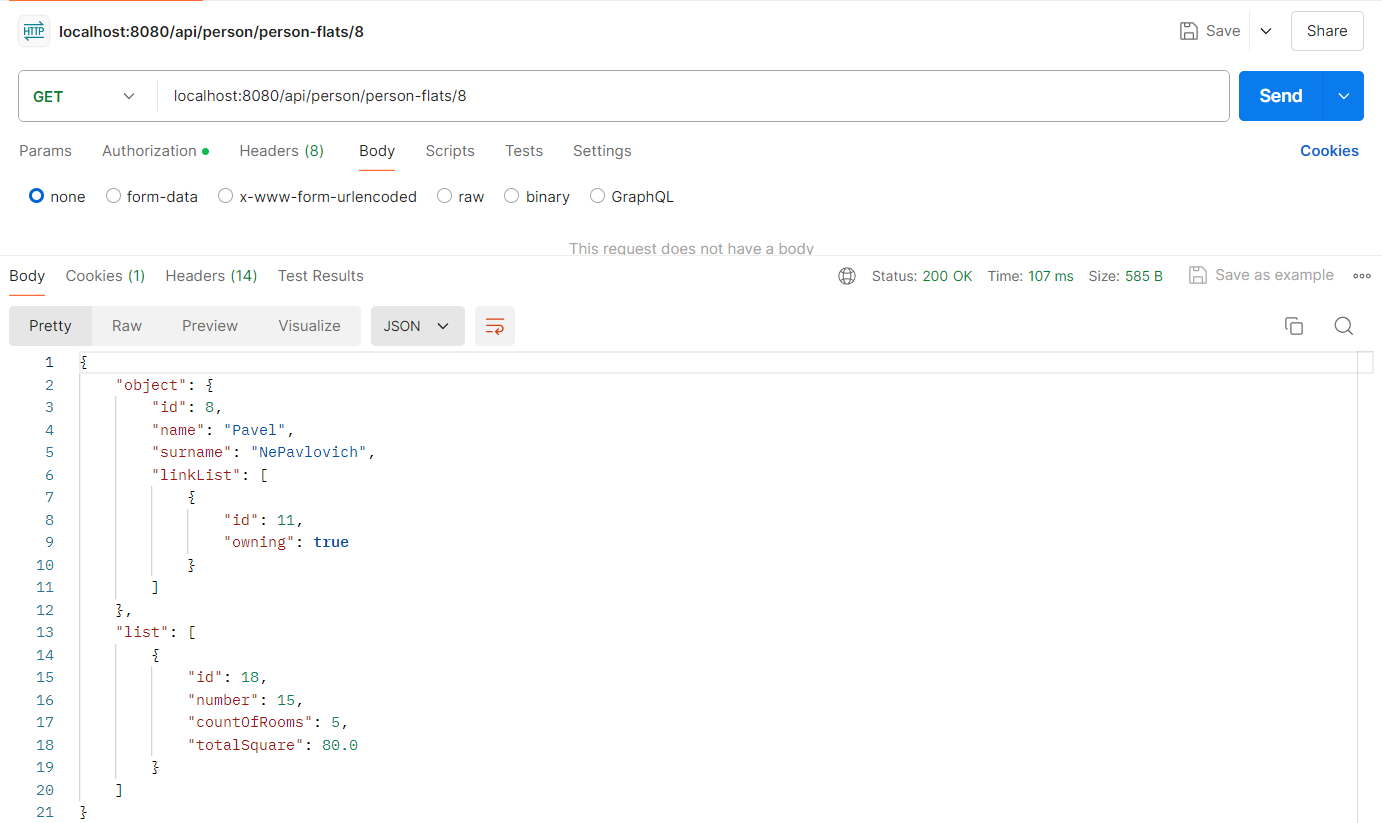


Рисунок 8.15 – Получение списка квартир, принадлежащих жильцу

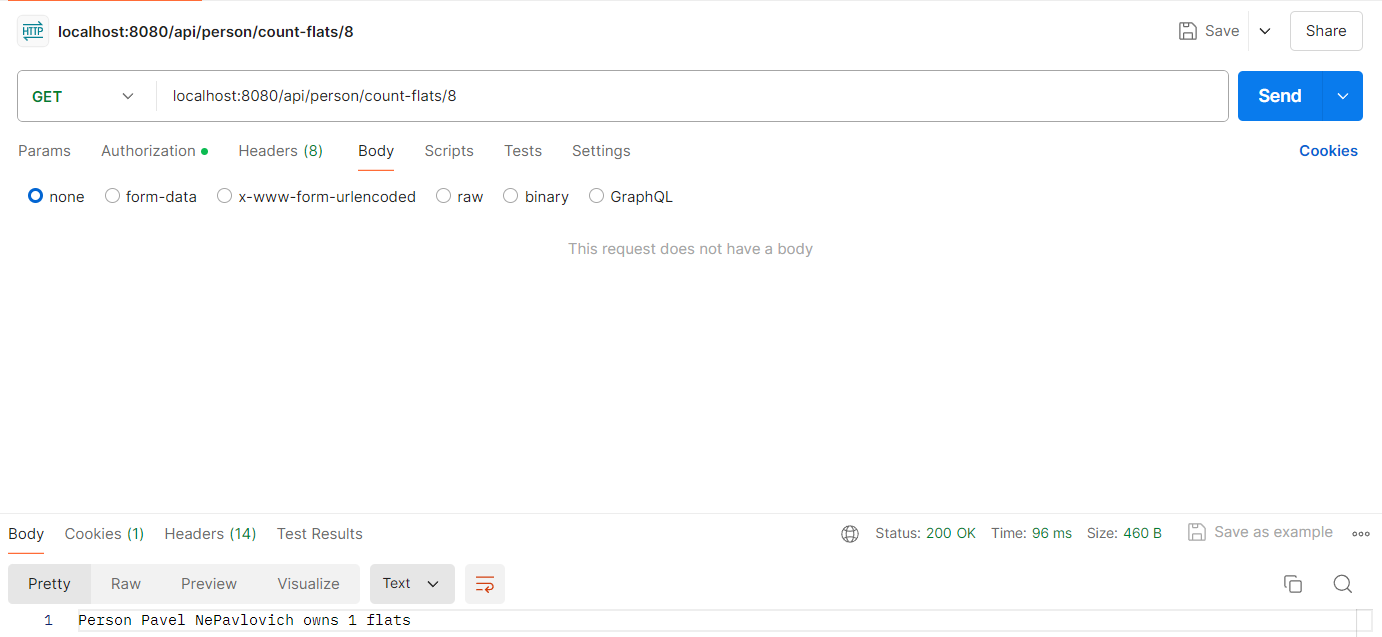


Рисунок 8.16 – Получение количества квартир, принадлежащих жильцу

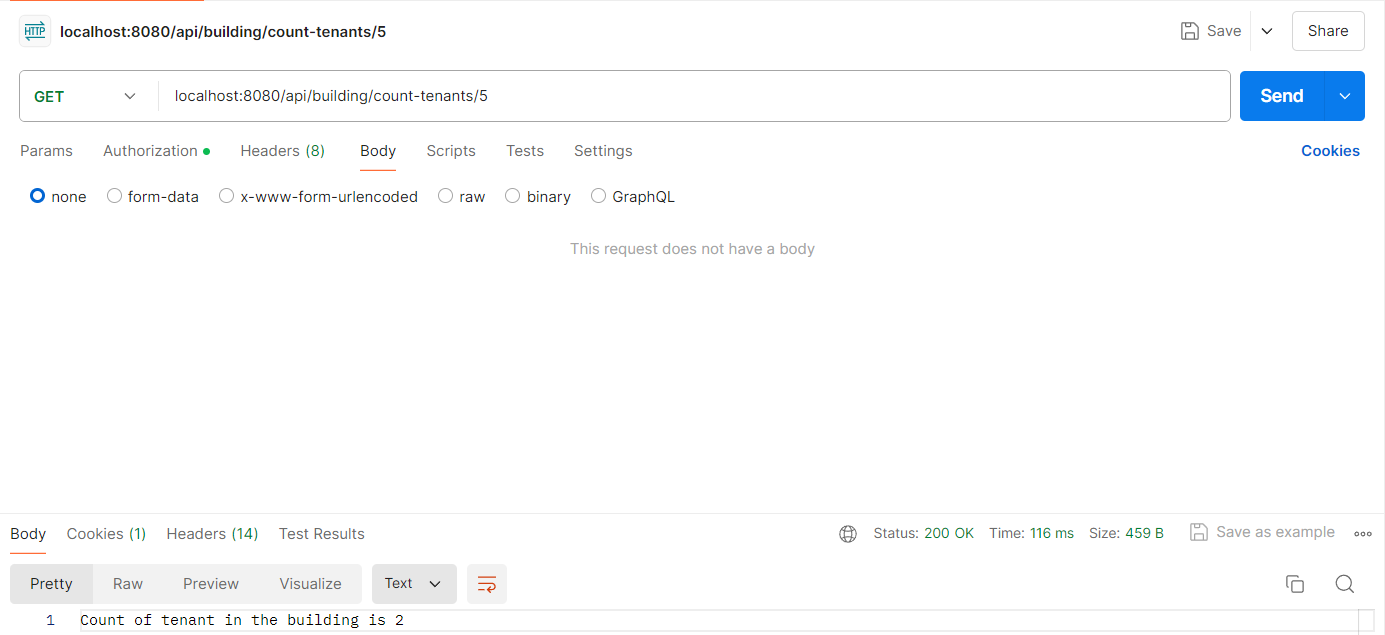


Рисунок 8.17 – Получение количества жильцов, проживающих в здании

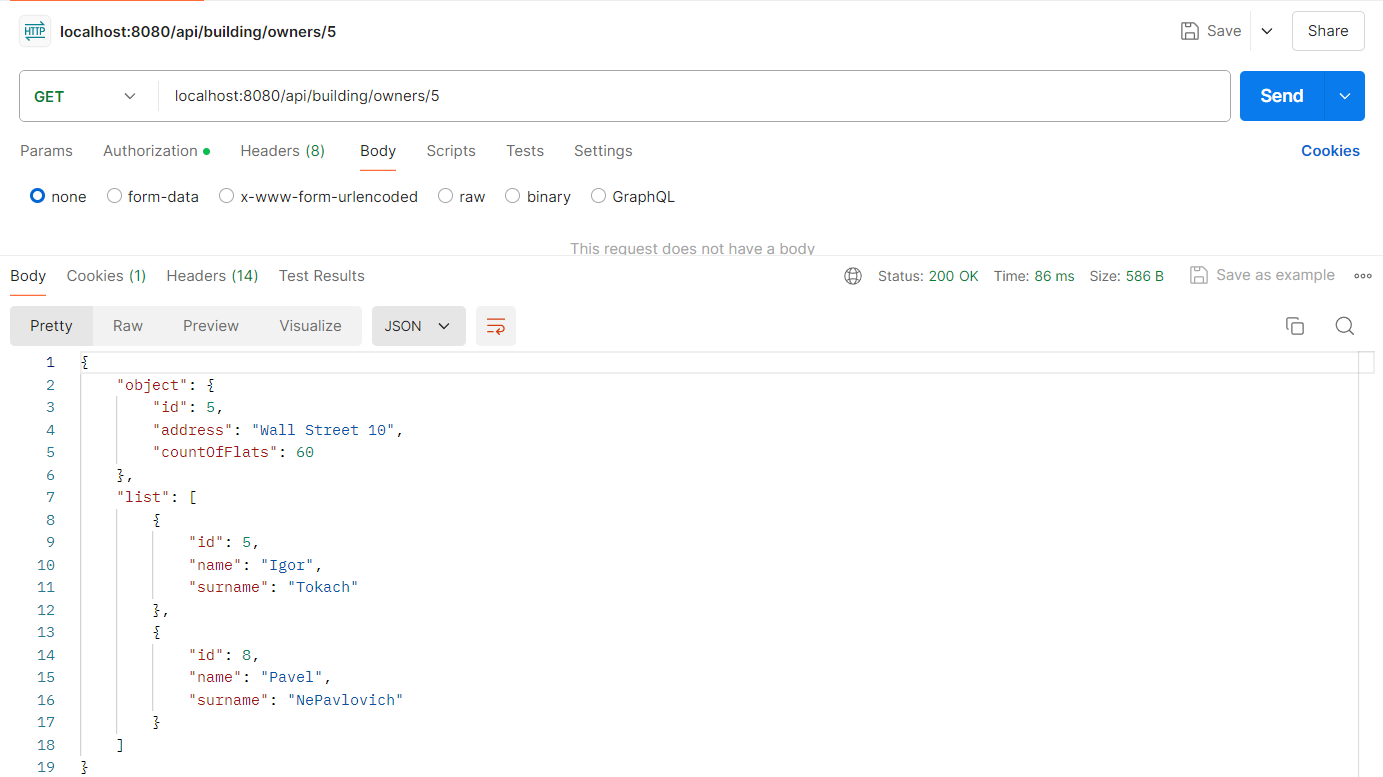


Рисунок 8.18 – Получение списка владельцев квартир в здании

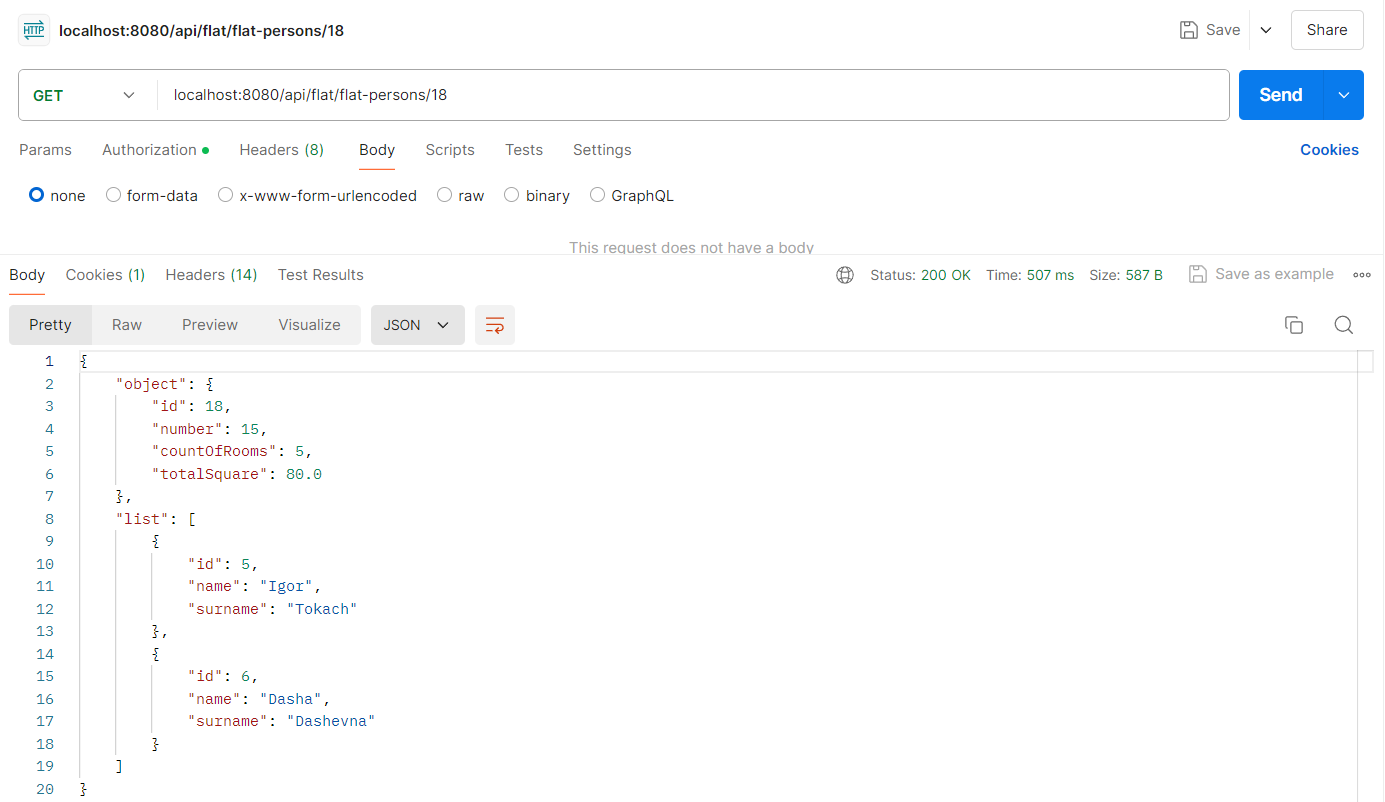


Рисунок 8.19 – Получение списка жильцов, проживающих в квартире

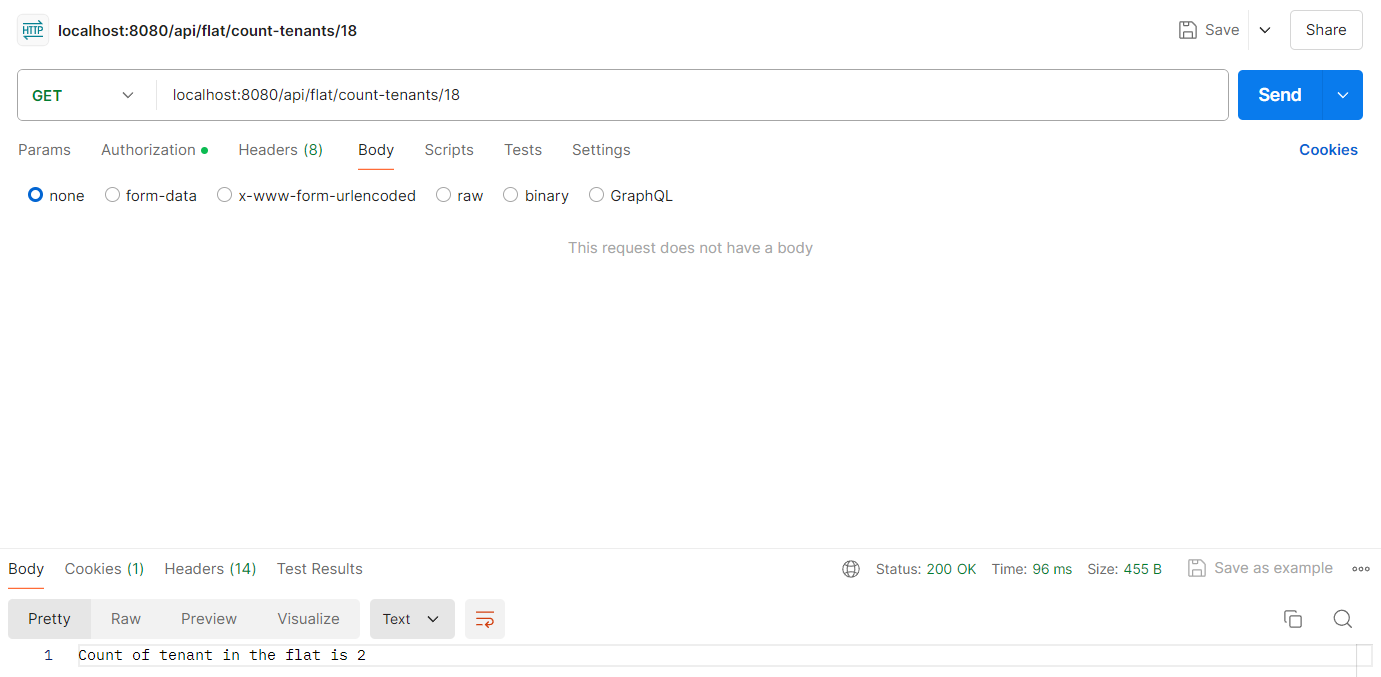


Рисунок 8.20 – Получение количества жильцов, проживающих в квартире

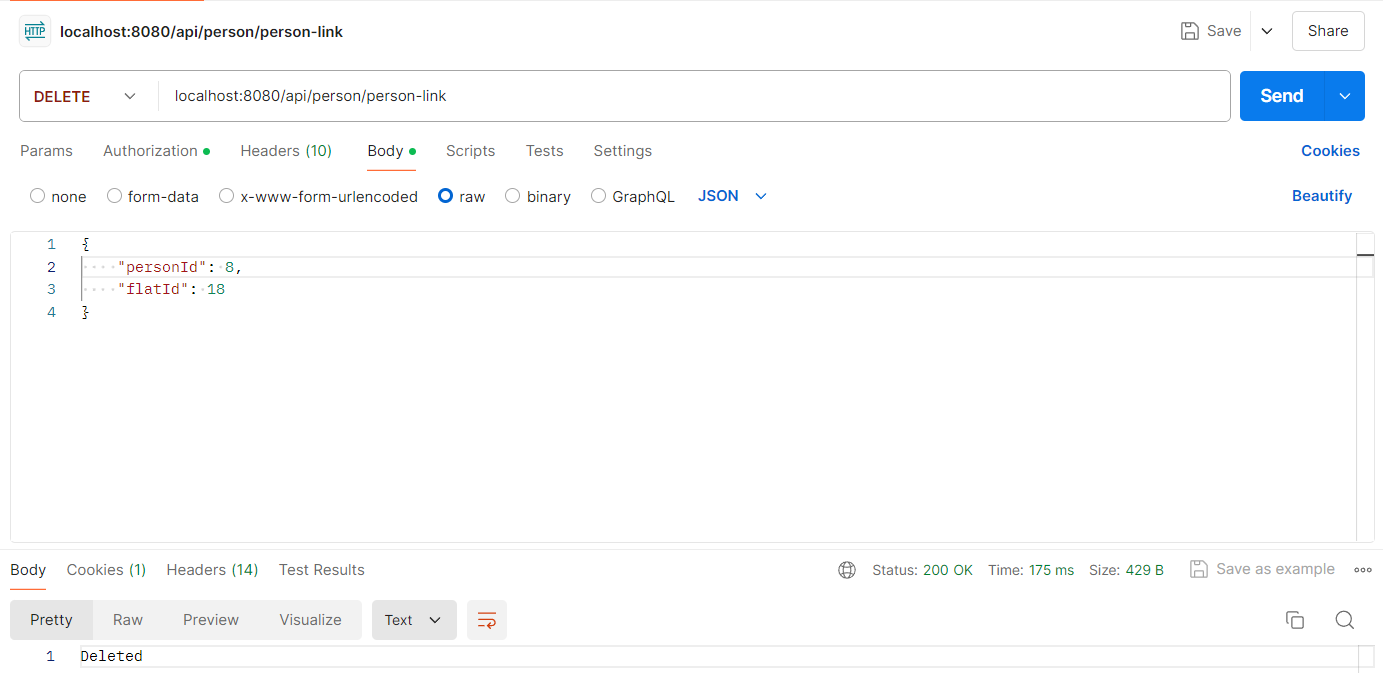


Рисунок 8.21 – Удаление связи между жильцом и квартирой

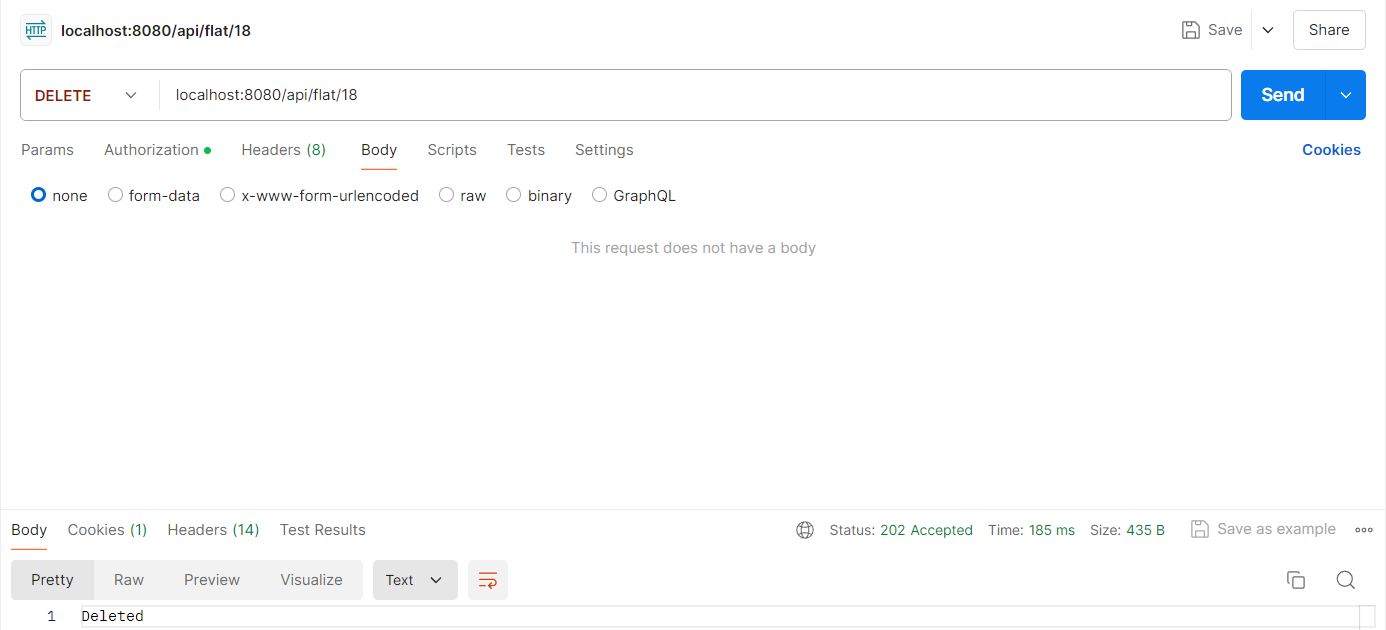


Рисунок 8.22 – Удаление квартиры

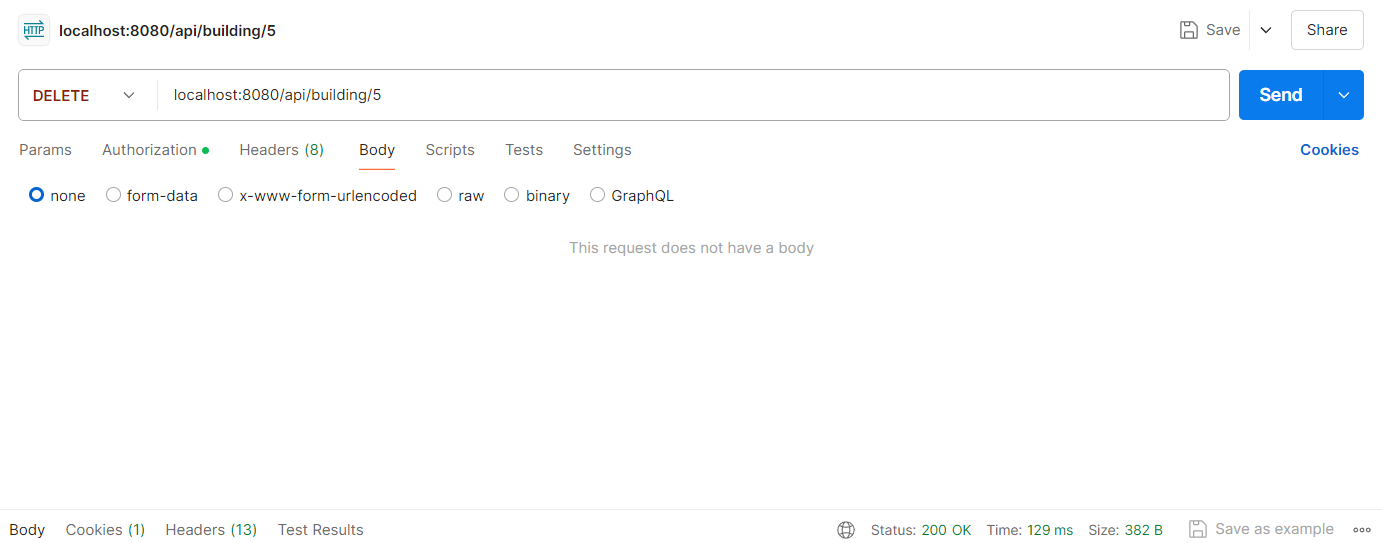


Рисунок 8.23 – Удаление здания

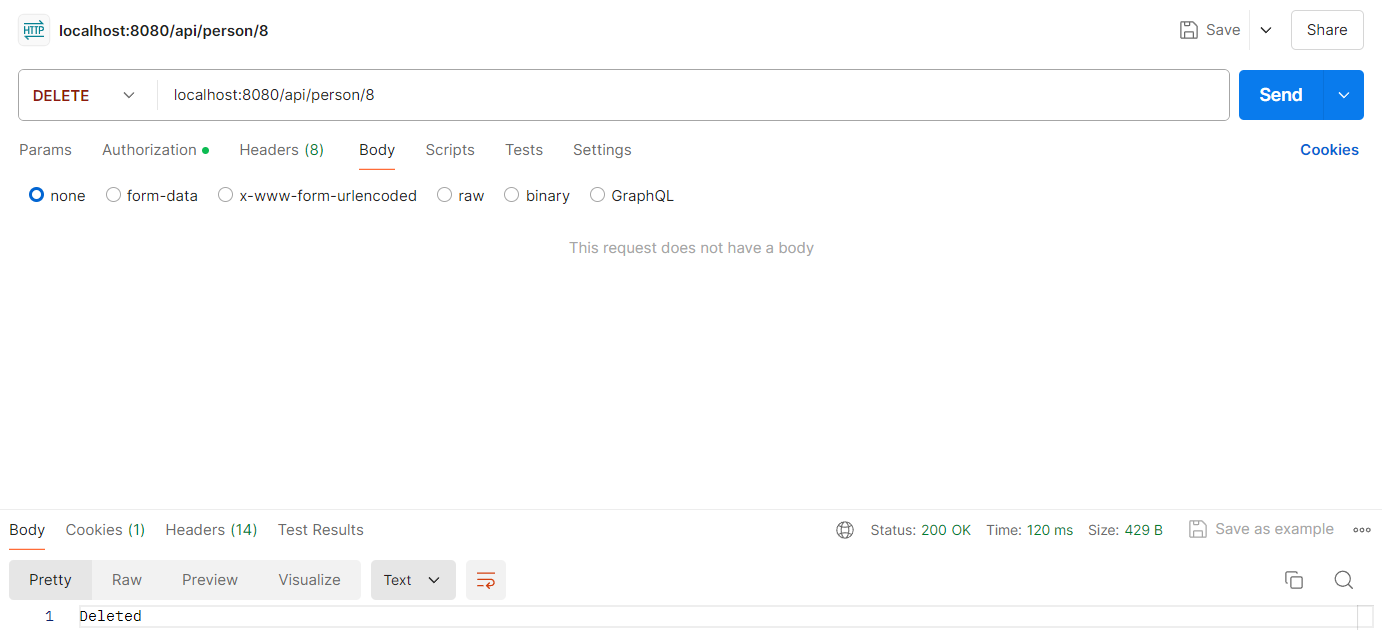


Рисунок 8.24 – Удаление жильца

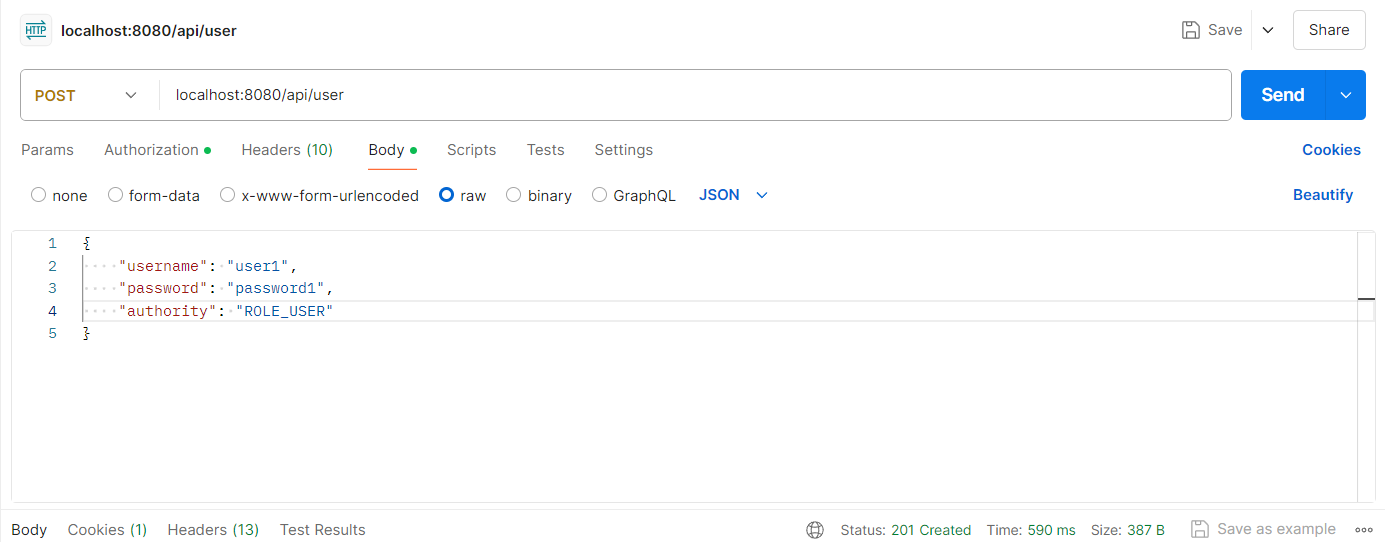


Рисунок 8.25 – Создание нового пользователя

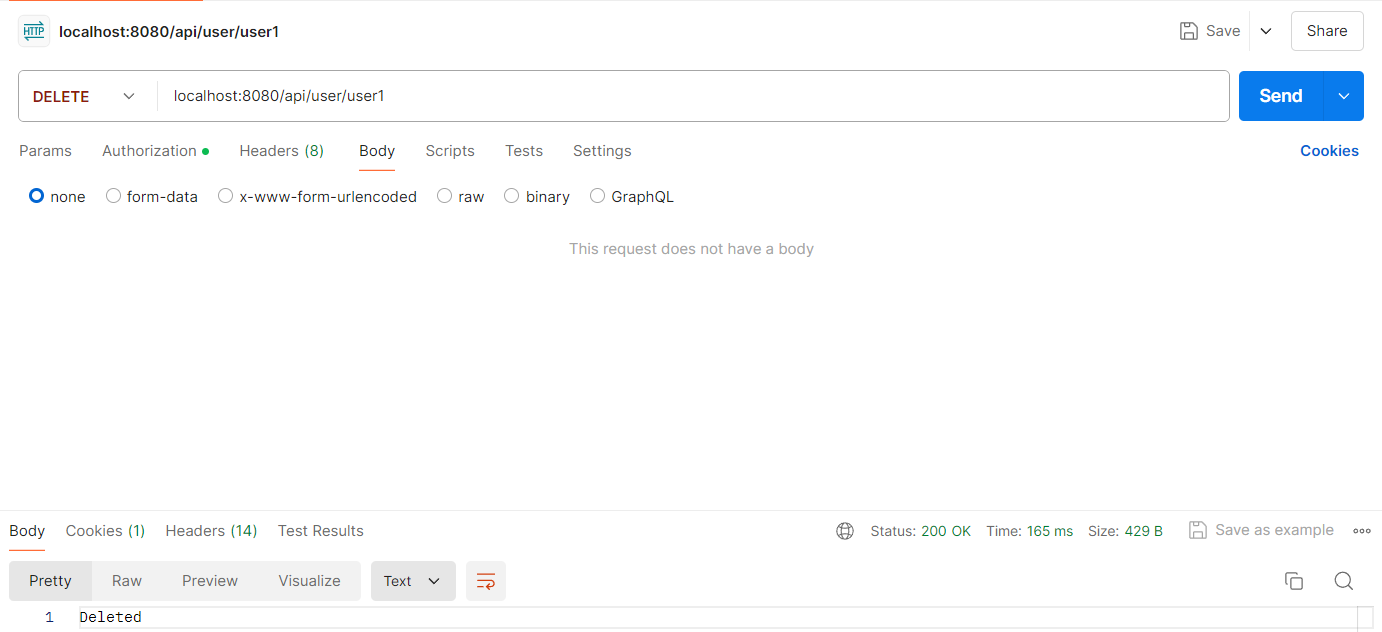


Рисунок 8.26 – Удаление пользователя

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

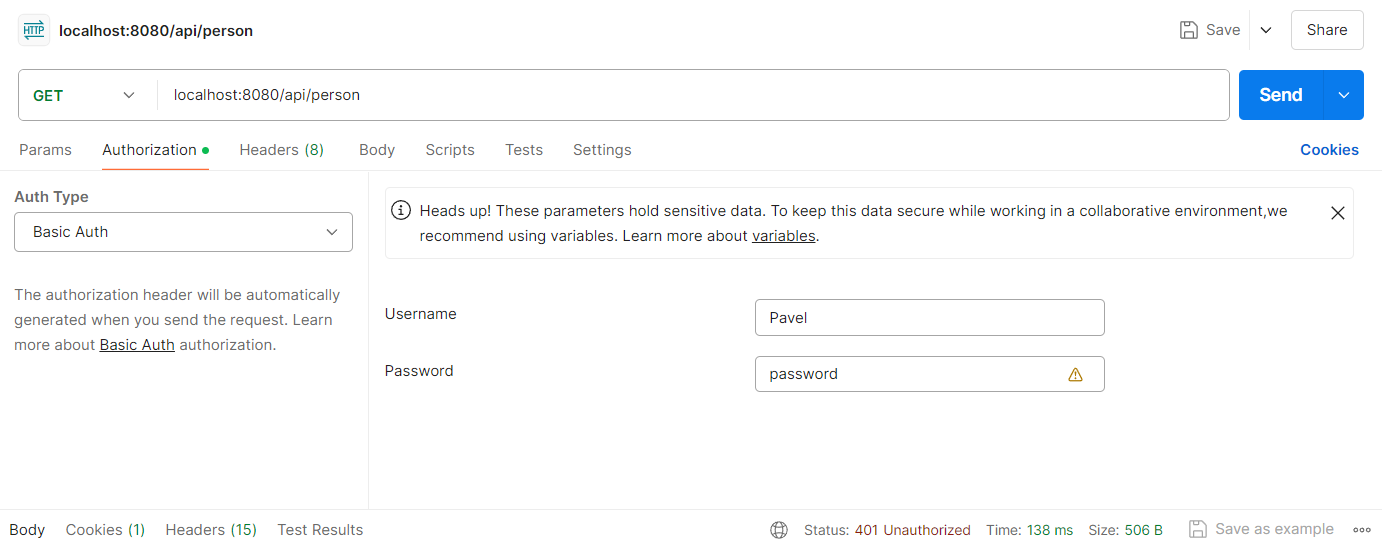


Рисунок 8.27 – Получение списка всех жильцов

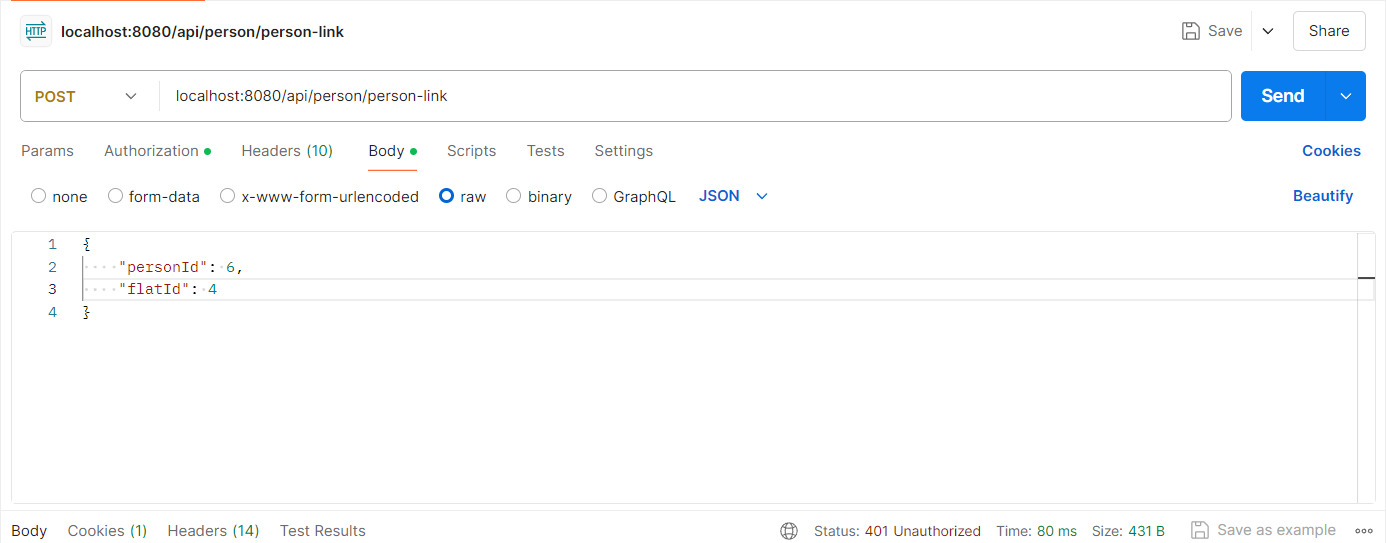


Рисунок 8.28 – Добавление связи между жильцом и квартирой

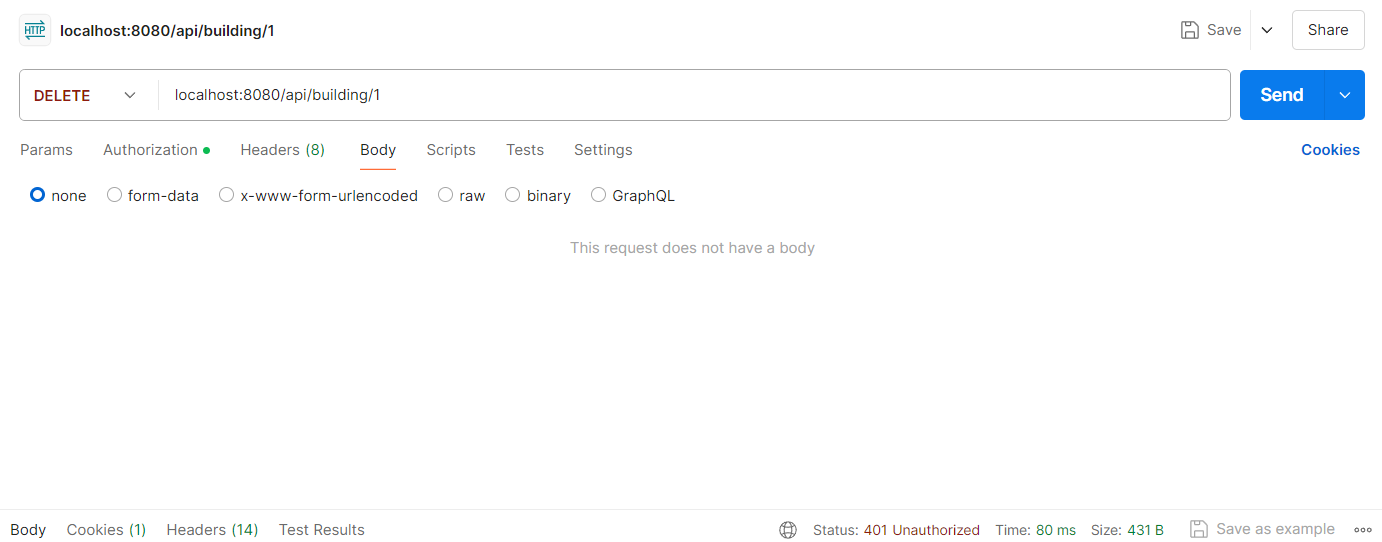


Рисунок 8.29 – Удаление здания

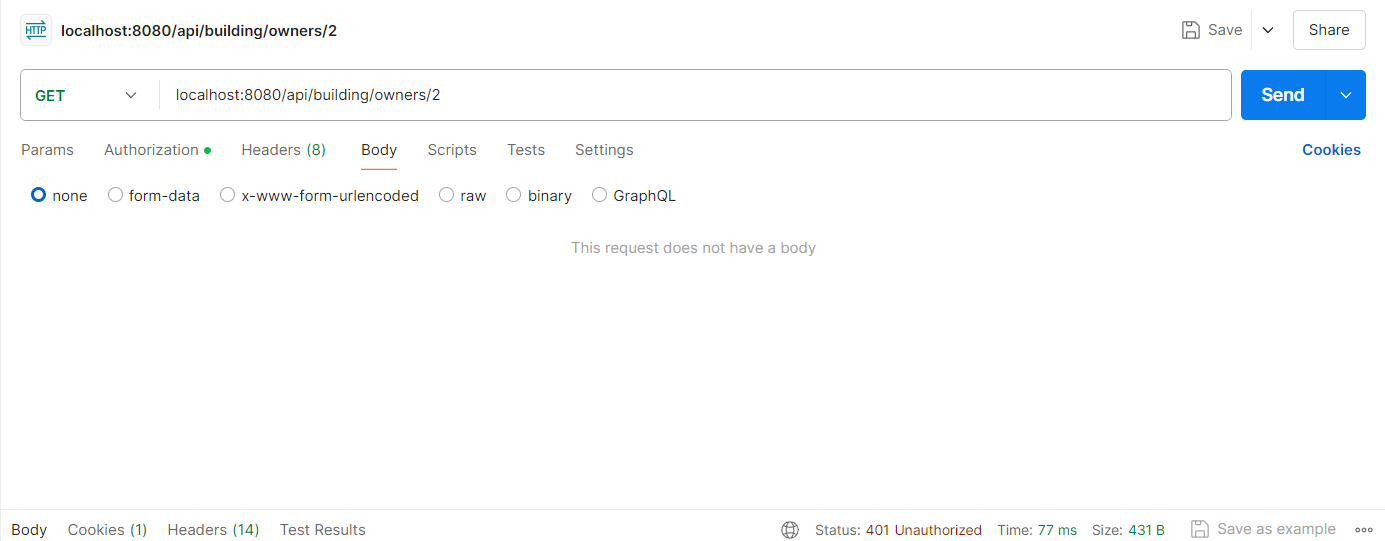


Рисунок 8.30 – Получение списка владельцев квартир в здании

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.

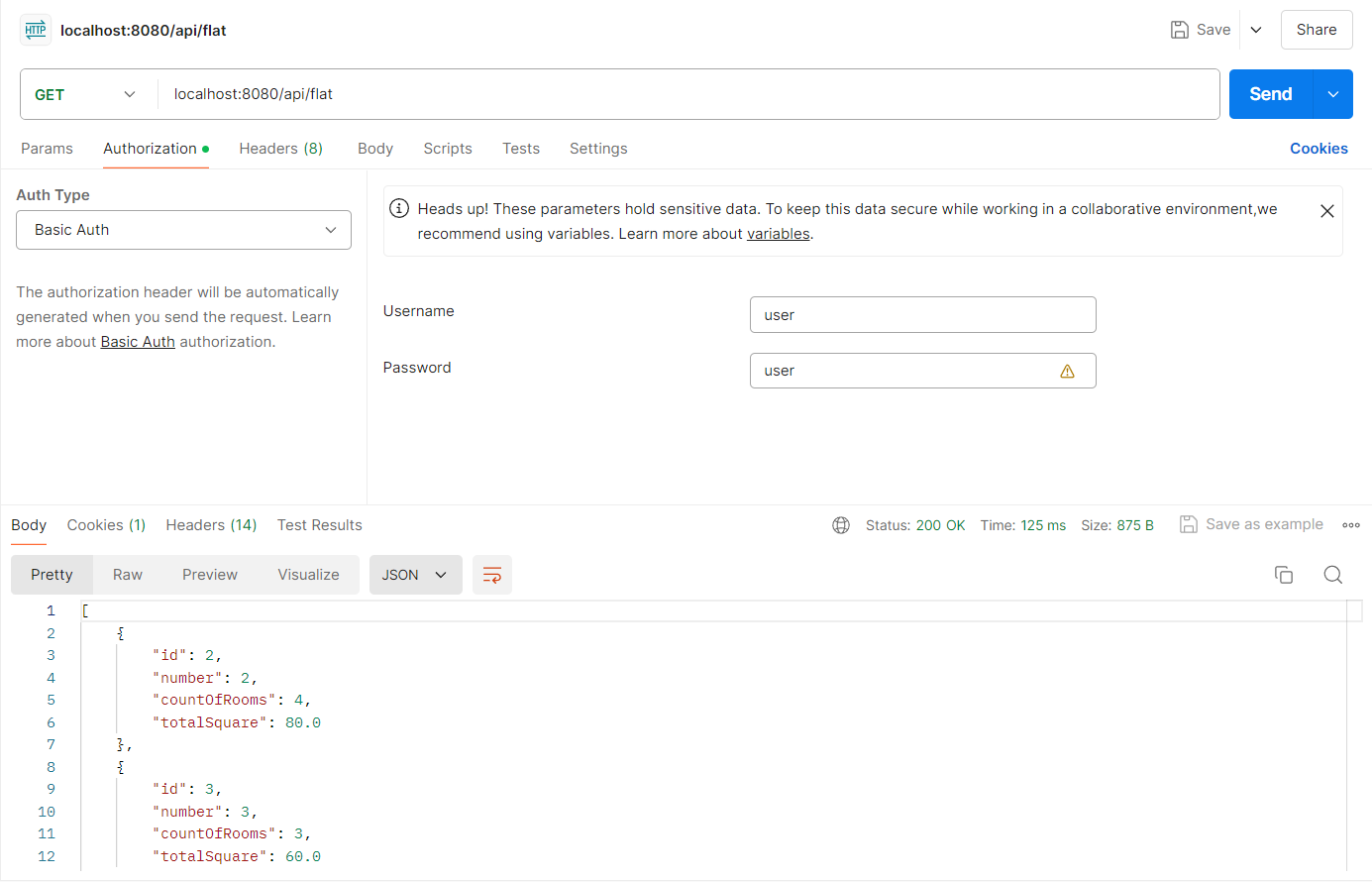


Рисунок 8.31 – Получение списка всех квартир

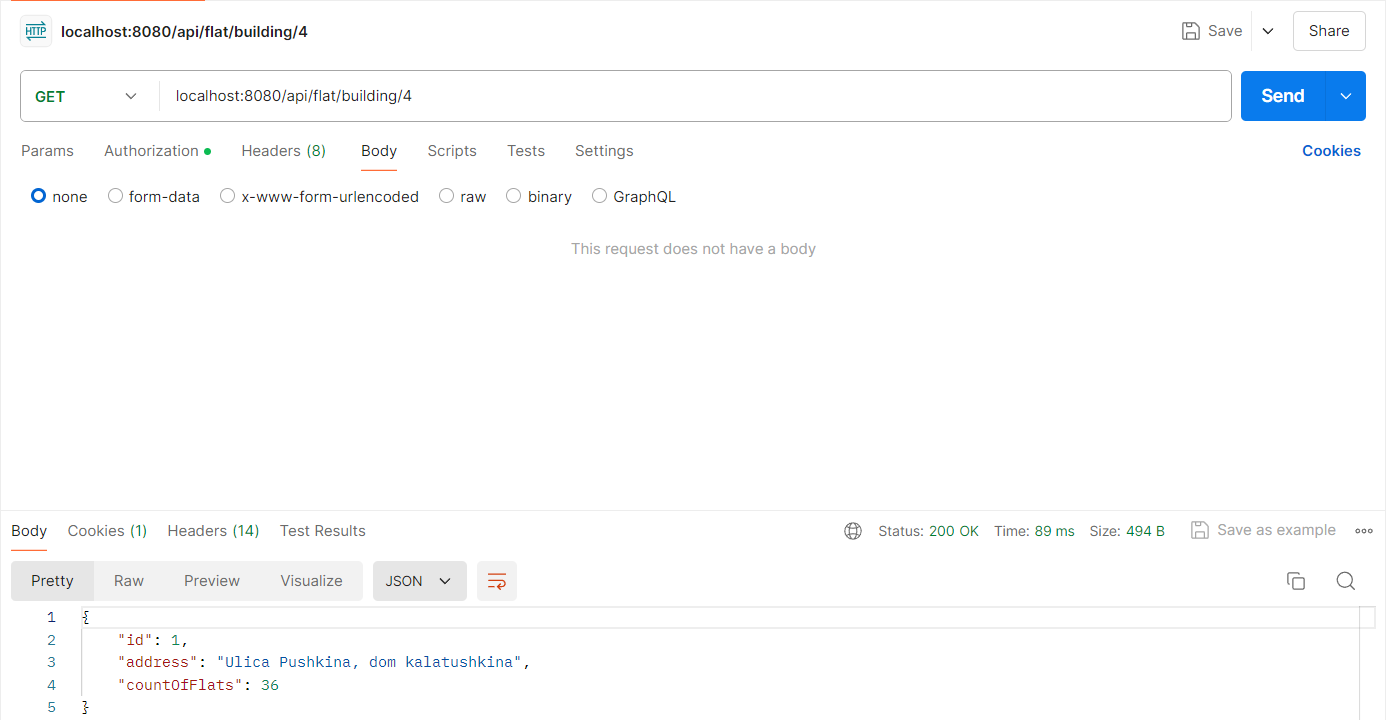


Рисунок 8.32 – Получение информации о здании, в котором находится квартира

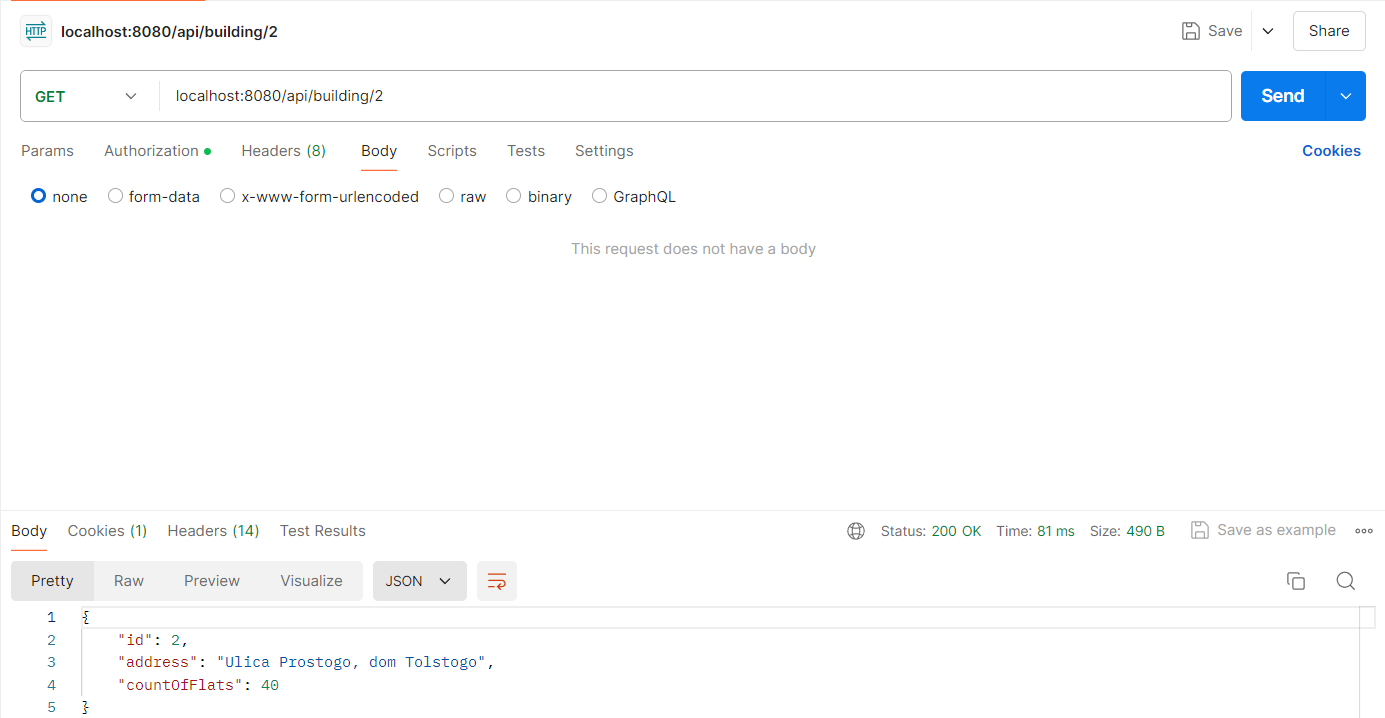


Рисунок 8.33 – Получение информации о конкретном здании

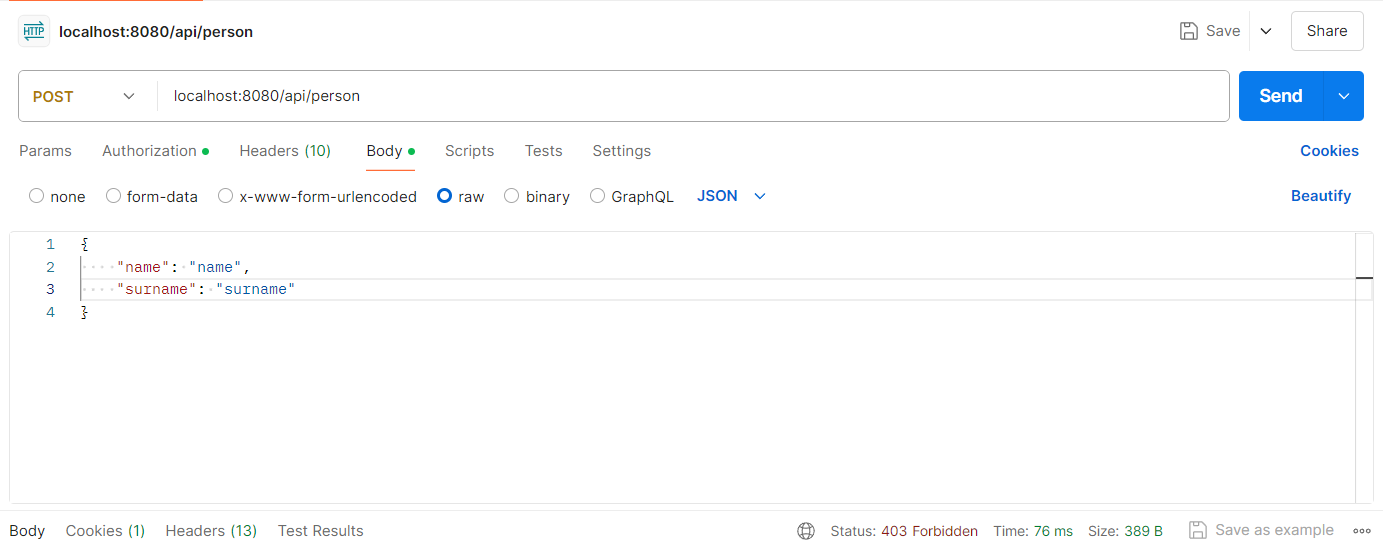


Рисунок 8.34 – Создание нового жильца

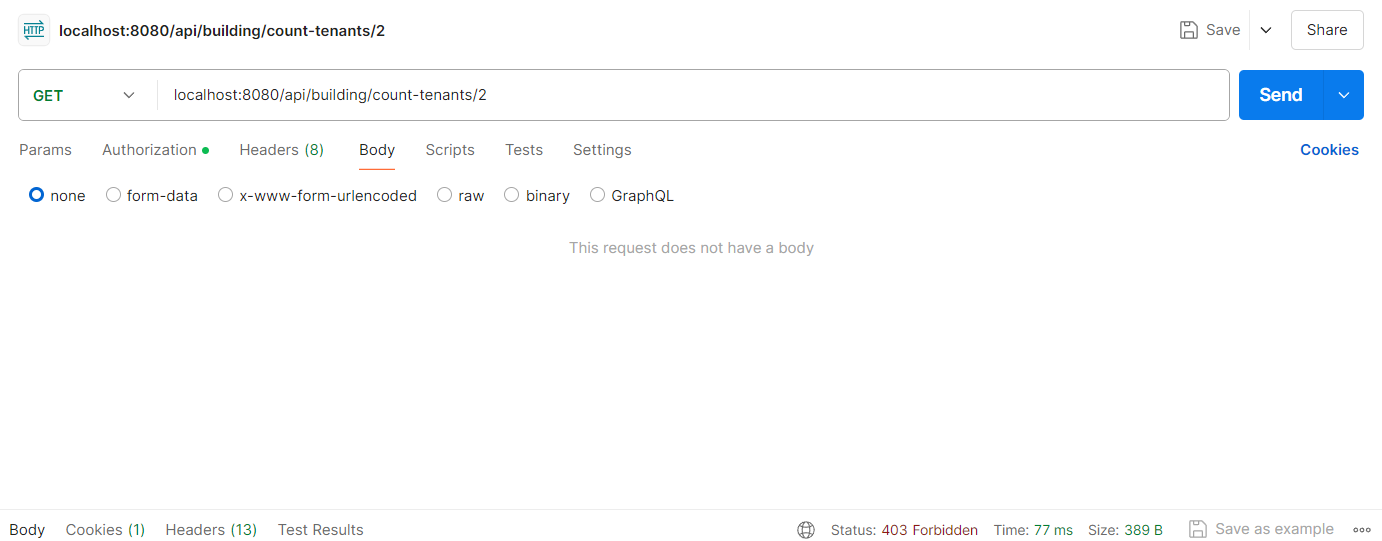


Рисунок 8.35 – Получение количества жильцов, проживающих в здании

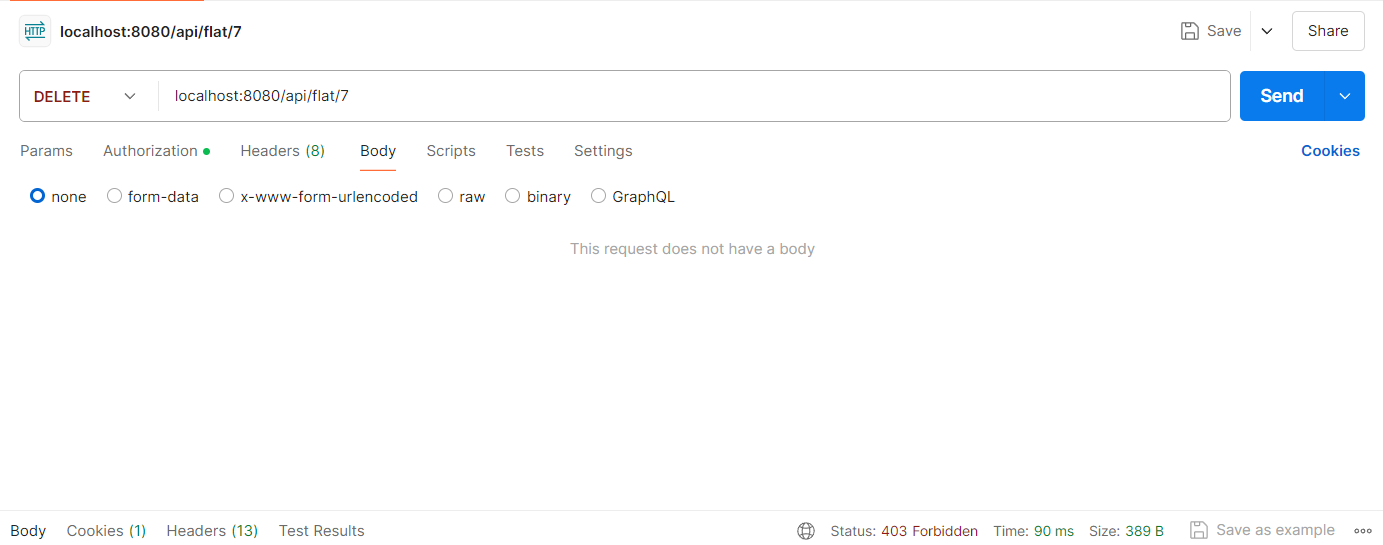


Рисунок 8.36 – Удаление квартиры

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.