МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Московский авиационный институт (национальный исследовательский университет)»

Ступинский филиал МАИ

Кафедра «Моделирование систем и информационные технологии»

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «ООП»

**Модуль бронирования**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | | | *Лисиченко М.И.* |  |
| Группа | | | ТСО-305Б-21 |  |
| Руководитель | | | *Новиков Б.Б.* |  |
| Оценка |  | Дата защиты «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024г. | | |

**Ступино 2024**

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Московский авиационный институт (национальный исследовательский университет)»

Ступинский филиал МАИ

Кафедра «Моделирование систем и информационные технологии»

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. М. Мамонов

(И.О.Фамилия)

« \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_ г.

**З А Д А Н И Е**

на курсовой проект по дисциплине

«ООП»

Студент ТСО-305Б-21 Лисиченко М.И.

(№ группы, Ф. И. О.)

Тема База данных сервиса выбора жилья для путешествий

Перечень вопросов, подлежащих разработке в курсовой работе

1. Проектирование БД.
2. Описание работы web-интерфейса с БД
3. Описание веб апи, взаимодействие с инсомнией.
4. Описание панели администратора веб-интерфейса
5. Описание управления пользователями

Рекомендуемая литература

1. Кристиан В. Безопасность ASP. NET CORE / Изд-во: ДМК Пресс, 2023, 386 с.

2. Умрихин Е.Д. Разработка веб-приложений с помощью ASP.Net Core MVC. / Изд-во: БХВ, 2023,

416 с.

3. Евдокимов П.В. C# на примерах. Практика, практика и только практика/ Изд-во: Наука и техника, 2022, 320 с.

4. Арно Л. Проектирование веб-API / Изд-во: ДМК Пресс, 2020, 440 с.

5. Макеев Г. Обьектно-ориентированное программирование: с нуля к SOLID и MVC / Изд-во: БХВ, 2024, 272 с.

Задание выдано «\_\_\_»\_\_\_\_\_\_\_\_\_20\_\_ г.

Руководитель Новиков Б.Б.

(Ф. И. О., должность, подпись)

Студент Лисиченко М.И.

(подпись)

# **Содержание**

[**Содержание** 3](#_Toc169186496)

[**Введение** 4](#_Toc169186497)

[**Постановка задачи** 5](#_Toc169186498)

[**Исследовательская часть** 6](#_Toc169186499)

[**Практическая часть** 1](#_Toc169186508)2

[**1.** **Проектирование БД** 1](#_Toc169186509)2

[**1.2. Описание таблиц** 1](#_Toc169186510)2

[**6.** **Описание работы web-интерфейса с БД** 2](#_Toc169186511)2

[**Веб API** 22](#_Toc169186512)

[**Взаимодействие с swagger** 2](#_Toc169186513)3

[**Заключение** 2](#_Toc169186514)5

[**Список литературы** 2](#_Toc169186515)6

# **Введение**

В современном мире туризм является одной из ключевых отраслей экономики, а с развитием технологий, он все больше переходит в онлайн-формат. Сервисы по выбору жилья для путешествий становятся неотъемлемой частью этого процесса, предлагая пользователям удобный и эффективный способ найти подходящий вариант размещения.

Данная курсовая работа посвящена разработке back-end модуля бронирования для базы данных сервиса выбора жилья для путешествий. Модуль предоставляет функцию бронирования номеров, вывод информации и её редактирования.

# **Постановка задачи**

Цель курсовой работы по ООП и разработке модуля «Дронирования» — создание программного модуля, обеспечивающего управление бронированием номеров, получением, имением и удалением информации о бронировании.

**Задачи:**

* создание схемы БД по модулю и краткое описание таблиц;
* обеспечить взаимосвязь между таблицами данного модуля, и взаимосвязь с таблицами из других модулей;
* описание веб-апи;
* взаимодействие с swagger;
* описание панели администратора веб-интерфейса;
* описание управление пользователями;

Модуль «Бронирования» предназначен для взаимодействия и изменения информации о бронировании. Он позволяет отслеживать статус бронирования и дату изменения информации о бронировании, которая включает в себя данные: об оплате и заказанных основных бесплатны и дополнительных платных услугах, датах въезда и выезда, о пользователе, совершившим бронирование и способе оплаты брони.

# **Исследовательская часть**

**Веб-API (Web API)** — это набор правил и спецификаций, определяющих, как приложения могут взаимодействовать друг с другом через интернет.

**Ключевые особенности веб-API:**

* **Интерфейс:** определяет, как приложения взаимодействуют с API.
* **Протокол HTTP:** используется для обмена данными между приложениями.
* **Форматы данных:** Данные передаются в форматах, таких как JSON или XML.
* **Ресурсы:** API предоставляет доступ к данным, функциям или сервисам.
* **Методы HTTP:** GET, POST, PUT и DELETE - используются для выполнения операций с ресурсами.

**Преимущества веб-API:**

* **Упрощенное взаимодействие:** Приложения могут взаимодействовать друг с другом, не зная подробностей реализации.
* **Гибкость:** API легко интегрируется с различными приложениями и платформами.
* **Масштабируемость:** API может обрабатывать большое количество запросов.
* **Обновления без перезапуска:** Изменения в API не требуют перезапуска приложений, использующих его.

**C#** — это объектно-ориентированный язык программирования, разработанный Microsoft.

**Ключевые характеристики C#:**

* **Объектно-ориентированный:** поддерживает принципы ООП, такие как инкапсуляция, наследование и полиморфизм.
* **Типизированный:** Переменные имеют определенный тип данных.
* **Управление памятью:** Автоматическое управление памятью упрощает разработку.
* **Современные функции:** Лямбда-выражения, LINQ, асинхронное программирование.
* **Платформа .NET:** C# тесно интегрирован с платформой .NET, предоставляющей обширный набор инструментов.

**Преимущества C#:**

* **Мощность и гибкость:** позволяет создавать сложные и эффективные приложения.
* **Простота изучения:** Ясный синтаксис и простота освоения.
* **Хорошая документация:** Обширные ресурсы и документация для C#.
* **Большое сообщество:** Активное сообщество разработчиков, предоставляющее поддержку и обмен опытом.

**ASP.NET Core** — это кроссплатформенная, высокопроизводительная платформа для создания современных веб-приложений.

**Ключевые особенности ASP.NET Core:**

* **Кроссплатформенность:** работает на Windows, macOS и Linux.
* **Высокая производительность:** Оптимизирован для максимальной производительности и масштабируемости.
* **Модульность:** позволяет выбирать только необходимые компоненты.
* **Открытый исходный код:** Разработчики могут вносить свой вклад и использовать его свободно.
* **Современные технологии:** Razor Pages, MVC, Blazor, Web API.
* **Интеграция с облачными платформами:** легко интегрируется с Azure, AWS и OpenAI Cloud Platform.

**Преимущества ASP.NET Core:**

* **Упрощенная разработка:** предоставляет инструменты и функции для облегчения разработки.
* **Высокая производительность:** Оптимизирован для максимальной производительности и масштабируемости.
* **Гибкость:** позволяет создавать различные типы веб-приложений.
* **Кроссплатформенность:** работает на различных платформах.
* **Активное сообщество:** предоставляет поддержку, документацию и примеры кода.

**Entity Framework** — это объектно-реляционное отображение (ORM), позволяющее разработчикам взаимодействовать с базами данных с использованием объектов .NET вместо SQL-запросов.

**Основные функции Entity Framework:**

* **ORM:** представляет данные из реляционных баз данных в виде объектов .NET.
* **Code First:** создает классы .NET, представляющие схему базы данных.
* **Database First:** создает классы .NET, представляющие схему существующей базы данных.
* **Model First:** генерирует классы .NET и базу данных на основе модели данных.
* **LINQ:** использует LINQ для запросов к базе данных, что делает запросы более удобными.
* **Изменения отслеживания:** отслеживает изменения в объектах и обновляет базу данных.

**Преимущества Entity Framework:**

* **Упрощение работы с базами данных:** Взаимодействие с базами данных с использованием объектов .NET.
* **Повышение продуктивности:** Сосредоточение на бизнес-логике, а не на написании SQL-запросов.
* **Лучшая читаемость кода:** Использование объектов .NET делает код более читаемым.
* **Поддержка различных баз данных:** поддерживает множество различных баз данных.

**JWT (JSON Web Token)** — это стандартный способ передачи информации между сторонами, обычно в виде безопасного токена.

**Основные компоненты JWT-токена:**

* **Заголовок (Header):** Информация о типе токена и алгоритме шифрования.
* **Полезная нагрузка (Payload):** Информация о пользователе, например, имя, идентификатор, роль.
* **Подпись (Signature):** Проверка подлинности токена, созданная с помощью секретного ключа.

**Как работает JWT-токен:**

1. **Авторизация:** Сервер создает JWT-токен с информацией о пользователе.
2. **Передача токена:** Сервер возвращает токен клиенту.
3. **Проверка токена:** Клиент отправляет токен на сервер, который проверяет его подпись.
4. **Авторизация:** Сервер использует информацию о пользователе для авторизации доступа.

**Преимущества JWT-токена:**

* **Безопасность:** Шифрование токена делает его устойчивым к подделке.
* **Стандартизация:** Совместимость с различными платформами и языками программирования.
* **Легкость в использовании:** Простой в генерации, отправке и проверке.
* **Самодостаточность:** содержит всю необходимую информацию для аутентификации и авторизации.

**Недостатки JWT-токена:**

* **Размер:** может быть достаточно большим, что влияет на производительность.
* **Хранение:** требует безопасного хранения.
* **Управление ключами:** Тщательное управление ключами для шифрования.

**Code First** — это один из подходов к разработке базы данных с использованием Entity Framework, который позволяет создавать модель данных с помощью классов в коде. Разработчики могут сосредоточиться на проектировании классов и их взаимосвязей, а Entity Framework автоматически создаст соответствующую базу данных.

**Архитектурный паттерн CSR (Controller Service Repository)** — это вариация паттерна MVC, которая часто применяется при создании веб-приложений и веб-API. CSR предлагает структуру, позволяющую разделить ответственности между компонентами приложения, обеспечивая более четкую организацию кода.

**Основные компоненты CSR:**

* **Controller:** отвечает за обработку входящих запросов, взаимодействие с клиентами и управление потоком выполнения.
* **Service:** содержит бизнес-логику приложения, выполняет операции с данными, бизнес-правила и другие действия.
* **Repository:** обеспечивает доступ к данным и управляет операциями базы данных.

**Преимущества CSR:**

* **Упрощение поддержки и развития приложения.**
* **Повышение модульности приложения.**
* **Возможность повторного использования кода.**

**PostgreSQL (Postgres)** — это мощная и надежная система управления реляционными базами данных (СУБД).

**Ключевые особенности PostgreSQL:**

* **Открытый исходный код:** Доступен для свободного использования, модификации и распространения.
* **Расширяемость:** предлагает множество встроенных функций, типов данных и возможностей, а также поддерживает расширения.
* **ACID-совместимость:** обеспечивает надежность и целостность данных.
* **Многоязычность:** поддерживает множество языков программирования.
* **Расширенные возможности:** Полнотекстовый поиск, географические запросы, поддержка JSON.
* **Сообщество и поддержка:** Активное сообщество разработчиков и пользователей.

**В целом PostgreSQL — это мощная и гибкая СУБД, подходящая для различных типов приложений.**

# **Практическая часть**

## **1.** **Проектирование БД**

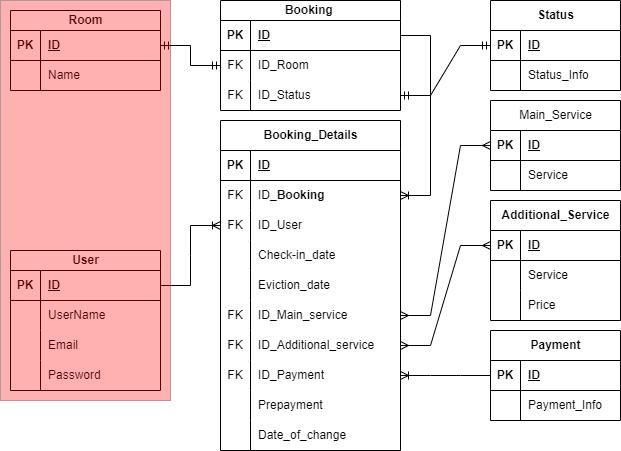


Рис. 1. Схема БД модуля “Бронирования”

## 

## **1.2. Описание таблиц**

**1) Booking**

Хранит информацию о статусе бронирования

Таблица содержит следующие столбцы:

* ID – первичный ключ, уникальный идентификатор;
* ID\_Room – вторичный ключ, таблицы Room;
* ID\_Status – вторичный ключ, таблицы Status;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Data;

public class Booking

{

public long ID { get; set; }

public Status Status { get; set; }

public long StatusID { get; set; }

public Room Room { get; set; }

public List<Booking\_Details> Booking\_DetailsBooking { get; set; } = [];

}

public class BookingMap

{

public BookingMap(EntityTypeBuilder<Booking> entityTypeBuilder)

{

entityTypeBuilder.HasKey(e => e.ID);

entityTypeBuilder

.HasOne(e => e.Status)

.WithMany(e => e.BookingStatus)

.HasForeignKey(e => e.StatusID);

entityTypeBuilder

.HasMany(e => e.Booking\_DetailsBooking)

.WithOne(e => e.Booking);

entityTypeBuilder

.HasOne(e => e.Room)

.WithOne(e => e.Booking)

.HasForeignKey<Room>(e => e.BookingRoomID);

}

}

**2) Booking\_Details**

Хранит информацию и историю изменения бронирования

Содержит следующие столбцы:

* ID – первичный ключ, уникальный идентификатор.
* ID\_Booking – внешний ключ к таблице Booking;
* ID\_User – внешний ключ к таблице User.;
* Chek-in\_date –дата заселения пользователя в номер;
* Eviction\_date – дата выселения пользователя из номера;
* ID\_Main\_service – внешний ключ к таблице Main\_Service;
* ID\_Additional\_service – внешний ключ к таблице Additional\_Service;
* ID\_Payment – внешний ключ к таблице Payment;
* Prepayment – состояние предоплаты за бронирование;
* Date\_of\_change – дата добавления изменений в бронирование;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Data;

public class Booking\_Details

{

public long ID { get; set; }

public DateTime Chek\_in\_date { get; set; }

public DateTime Eviction\_date { get; set; }

public bool Prepayment { get; set; }

public DateTime Date\_of\_change { get; set; }

public Payment Payment { get; set; }

public long PaymentID { get; set; }

public Booking Booking { get; set; }

public long BookingID { get; set; }

public User User { get; set; }

public long UserID { get; set; }

public List<Booking\_DetailsMain\_Service> Booking\_DetailsMain\_ServiceList { get; set; } = [];

public List<Booking\_DetailsAdditional\_Service> Booking\_DetailsAdditional\_ServiceList { get; set; } = [];

}

public class Booking\_DetailsMap

{

public Booking\_DetailsMap(EntityTypeBuilder<Booking\_Details> entityTypeBuilder)

{

entityTypeBuilder.HasKey(e => e.ID);

entityTypeBuilder.Property(e => e.Chek\_in\_date).IsRequired();

entityTypeBuilder.Property(e => e.Eviction\_date).IsRequired();

entityTypeBuilder.Property(e => e.Prepayment).IsRequired();

entityTypeBuilder.Property(e => e.Date\_of\_change).IsRequired();

entityTypeBuilder

.HasOne(e => e.Booking)

.WithMany(e => e.Booking\_DetailsBooking)

.HasForeignKey(e => e.BookingID);

entityTypeBuilder

.HasMany(e => e.Booking\_DetailsMain\_ServiceList)

.WithOne(e => e.Booking\_Details);

entityTypeBuilder

.HasMany(e => e.Booking\_DetailsAdditional\_ServiceList)

.WithOne(e => e.Booking\_Details);

entityTypeBuilder

.HasOne(e => e.Payment)

.WithMany(e => e.Booking\_DetailsPay)

.HasForeignKey(e => e.PaymentID);

entityTypeBuilder

.HasOne(e => e.User)

.WithMany(e => e.Booking\_DetailsUser)

.HasForeignKey(e => e.UserID);

}

}

**3) Status**

Содержит информацию об статусах номеров.

Содержит следующие столбцы:

* ID – первичный ключ, уникальный идентификатор шаблона статуса;
* Status\_Info – содержит уникальную информацию о статусе номера;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Data;

public class Status

{

public long ID { get; set; }

public string Status\_Info { get; set; }

public List<Booking> BookingStatus { get; set; } = [];

}

public class StatusMap

{

public StatusMap(EntityTypeBuilder<Status> entityTypeBuilder)

{

entityTypeBuilder.HasKey(e => e.ID);

entityTypeBuilder.Property(e => e.Status\_Info).IsRequired();

entityTypeBuilder

.HasMany(e => e.BookingStatus)

.WithOne(e => e.Status);

}

}

**4) Main\_Service**

Содержит описание об обязательных услугах на выбор.

Содержит следующие столбцы:

* ID – первичный ключ, уникальный идентификатор каждой услуги;
* Service – описание бесплатной обязательной услуги включаемой в номер;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Data;

public class Main\_Service

{

public long ID { get; set; }

public string Service { get; set; }

public List<Booking\_DetailsMain\_Service> Booking\_DetailsMain\_ServiceList { get; set; } = [];

}

public class Main\_ServiceMap

{

public Main\_ServiceMap(EntityTypeBuilder<Main\_Service> entityTypeBuilder)

{

entityTypeBuilder.HasKey(e => e.ID);

entityTypeBuilder.Property(e => e.Service).IsRequired();

entityTypeBuilder

.HasMany(e => e.Booking\_DetailsMain\_ServiceList)

.WithOne(e => e.Main\_Service);

}

}

**5) Additional\_Service**

Содержит описание об платных дополнительных услугах.

Содержит следующие столбцы:

* ID – первичный ключ, уникальный идентификатор каждой услуги;
* Service – описание платной услуги включаемой в номер;
* Price – стоимость услуги;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Data;

public class Additional\_Service

{

public long ID { get; set; }

public string Service { get; set; }

public long Price { get; set; }

public List<Booking\_DetailsAdditional\_Service> Booking\_DetailsAdditional\_ServiceList { get; set; } = [];

}

public class Additional\_ServiceMap

{

public Additional\_ServiceMap(EntityTypeBuilder<Additional\_Service> entityTypeBuilder)

{

entityTypeBuilder.HasKey(e => e.ID);

entityTypeBuilder.Property(e => e.Service).IsRequired();

entityTypeBuilder.Property(e => e.Price).IsRequired();

entityTypeBuilder

.HasMany(e => e.Booking\_DetailsAdditional\_ServiceList)

.WithOne(e => e.Additional\_Service);

}

}

**6) Payment**

Содержит подробную информацию об оплате бронирования.

Содержит следующие столбцы:

* ID – первичный ключ, уникальный идентификатор каждого типа оплаты;
* Payment\_Info – информация об оплате бронирования;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Data;

public class Payment

{

public long ID { get; set; }

public string Payment\_Info { get; set; }

public List<Booking\_Details> Booking\_DetailsPay { get; set; } = [];

}

public class PaymentMap

{

public PaymentMap(EntityTypeBuilder<Payment> entityTypeBuilder)

{

entityTypeBuilder.HasKey(e => e.ID);

entityTypeBuilder.Property(e => e.Payment\_Info).IsRequired();

entityTypeBuilder

.HasMany(e => e.Booking\_DetailsPay)

.WithOne(e => e.Payment);

}

}

**7) User**

Таблица, в которой содержатся данные о пользователе.

Содержит следующие столбцы:

* UniqueID – первичный ключ, уникальный идентификатор;
* Name – Имя пользователя;
* Email – почта пользователя;
* Password – пароль пользователя;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using Microsoft.AspNetCore.Identity;

namespace Data;

public class User : IdentityUser

{

public string Gender { get; set; }

public List<Booking\_Details> Booking\_DetailsUser { get; set; } = [];

}

public class UserMap

{

public UserMap(EntityTypeBuilder<User> entityTypeBuilder)

{

entityTypeBuilder.HasKey(e => e.Id);

entityTypeBuilder

.HasMany(e => e.Booking\_DetailsUser)

.WithOne(e => e.User);

}

}

**8) Room**

Содержит подробную информацию об номерах отелей.

Содержит следующие столбцы:

* ID – первичный ключ, уникальный идентификатор каждого пользователя;
* Name – название номера;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Data;

public class Room

{

public long ID { get; set; }

public string Room\_Name { get; set; }

public long BookingRoomID { get; set; }

public Booking Booking { get; set; }

}

public class RoomMap

{

public RoomMap(EntityTypeBuilder<Room> entityTypeBuilder)

{

entityTypeBuilder.HasKey(e => e.ID);

entityTypeBuilder.Property(e => e.Room\_Name).IsRequired();

entityTypeBuilder

.HasOne(e => e.Booking)

.WithOne(e => e.Room)

.HasForeignKey<Room>(e => e.BookingRoomID).IsRequired();

}

}

1. **Описание работы web-интерфейса с БД**

Работа web-интерфейса с базой данных (БД) происходит следующим образом:

* Клиент формирует и отправляет запрос к серверу, который обрабатывает программу, связанную с БД.
* Сервер обрабатывает запрос, выполняет манипуляции с данными в БД и формирует результат.
* Сервер отправляет результат клиенту в формате HTML.
* Клиент отображает полученный результат на экране и ожидает действий пользователя.
* Цикл повторяется до тех пор, пока пользователь не завершит работу с сервером.

Основные функции web-интерфейса с БД включают обеспечение отображения интерфейса пользователя в формате HTML, формирование запросов к БД, аутентификацию пользователя и обработку запросов с возвратом результата в HTML-формате.

1. **Описание веб апи, взаимодействие с swagger.**

Веб API(Application Programming Interface) — это метод получения данных из запросов. В интернете API существуют в двух формах: клиентские и серверные. Клиентские API используют, например, в Fullscreen и Dialog API спецификации HTML5. Серверный API, например, есть у Netflix, он показывает пользователям бесплатные фильмы во время поиска.

Веб API имеют архитектуру состоящая из:

* Data – хранит в себе данные, которые будут присутствовать в таблице при ее создании.
* DTO - плоские структуры данных без бизнес-логики, предназначенные для переноса данных между слоями.
* Repository - интерфейс для работы с данными, предоставляющий методы для добавления, удаления и обновления данных.
* Service - реализация Repository, предоставляющая конкретные методы для работы с данными.
* Controller - они вызывают методы сервисов, обрабатывают возвращаемые данные и исключения, принимают и отвечают на запросы;

## **Взаимодействие с swagger**

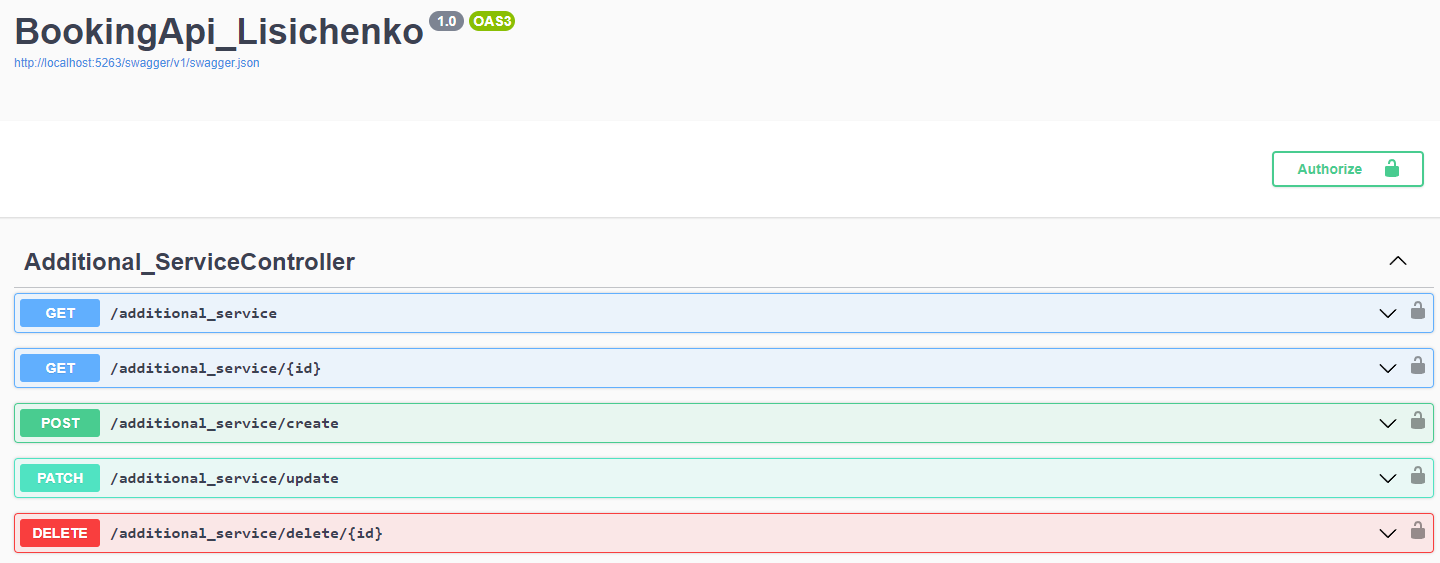
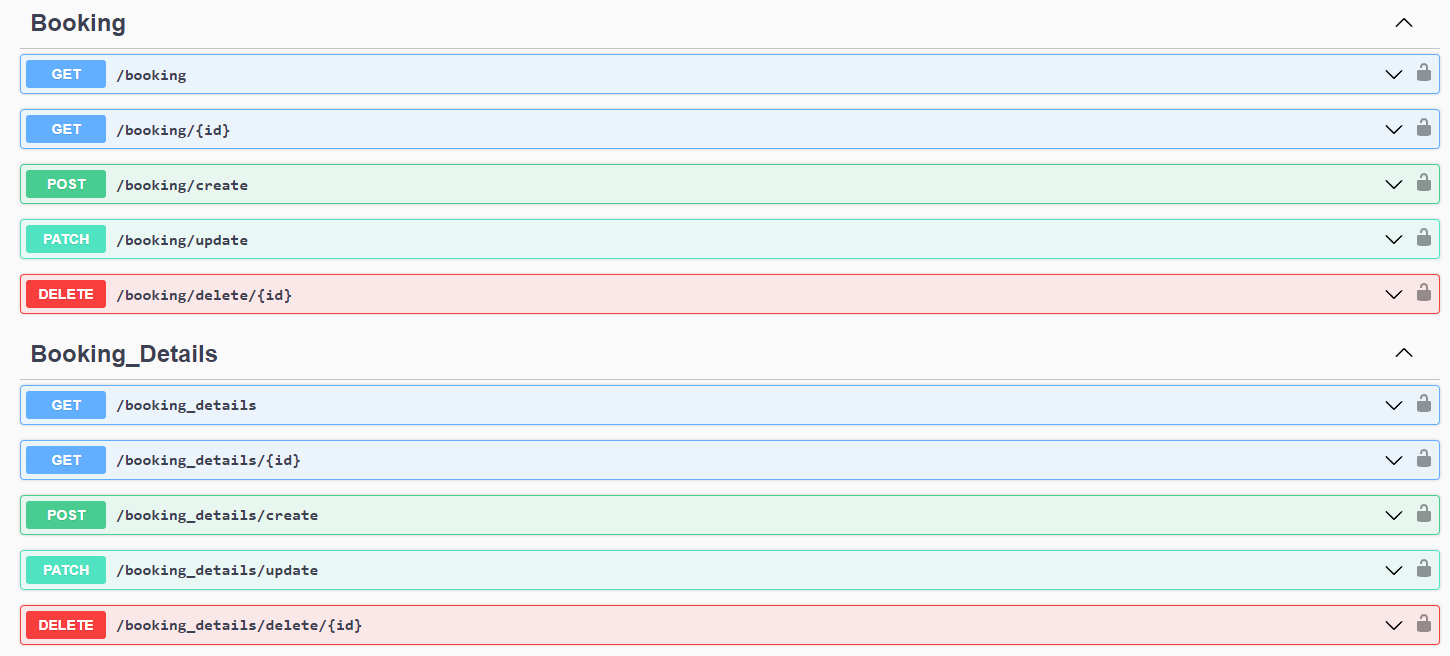
Как и было описано выше swagger работает с Http – запросами. Рассмотрим работу swagger.  

Рис. 1. Рабочее пространство в swagger.

Get — это метод HTTP-запроса, который используется для получения и просмотра данных.

Create — это метод HTTP-запроса, который используется для создания новых строк в таблицу данных.

Update — это метод запроса в HTTP, который используется для изменения и обновления уже существующих данных.

Delete — это метод HTTP-запроса, который используется для удаления данных с сервера.

Добавление, Изменение и Удаление данных с сервера доступно только авторизованным пользователям.

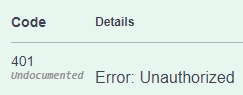


Рис. 2. Отказ не авторизированному пользователю в swagger.

1. **Описание управления пользователями**

Регистрация новых пользователей организуется по пути Post: /users/create и производится согласно следующему алгоритму:

* UserController передает входные данные в метод userService.InsertUser, который в свою очередь передаёт данные в метод UserRepository.InsertUser.
* В UserRepository.InsertUser у нас происходит добавление в базу данных.

# **Заключение**

Разработанный модуль «Бронирования» обеспечивает комплексное управление информацией о бронировании, позволяя отслеживать его статус и историю изменений. Модуль предоставляет доступ к данным об оплате, заказанных услугах (как бесплатных, так и платных), датах въезда и выезда, а также сведениям о пользователе, совершившем бронирование, и способе оплаты. Благодаря этому, система обеспечивает прозрачность и эффективность процесса бронирования, предоставляя полную информацию как клиентам, так и администраторам.

# **Список литературы**

«Entity Framework Core в действии» Джона Смита.

«Программирование Entity Framework: DbContext» Джулии Лерман и Роуэна Миллера.

«Entity Framework 6 Recipes» Брайана Дрисколла и Нитина Гупты.

«Pro Entity Framework Core 2 для ASP.NET Core MVC» Адама Фримена.

«Entity Framework Core Cookbook — второе издание» Рикардо Переса.

«Программирование Entity Framework: создание приложений с центром данных с помощью ADO.NET Entity Framework» Джулии Лерман.

«Microsoft Entity Framework 4.0 Cookbook» Тима Маллалью и Брайана Найта.

«Программирование Entity Framework: Code First» Джулии Лерман и Роуэна Миллера.

«Entity Framework Core в действии, второе издание» Джона Смита.