

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Отделение информационных технологий и вычислительной техники

Утверждаю

Руководитель
отделения ИТВТ _____

_____ Рощенко О.Е.
(подпись, фамилия, инициалы)

«___» _____ 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ДИПЛОМНЫЙ ПРОЕКТ

НА ТЕМУ

Разработка информационной системы учёта гостиничных номеров

Автор дипломного проекта _____

(подпись студента, выполнившего дипломный проект)

_____ Лепке Д.А. _____ Группа И - 82
(фамилия, инициалы студента) (в которой обучался студент)

_____ Институт социальных технологий

(факультет)

Специальность _____ 09.02.03 Программирование в компьютерных системах
(код и наименование специальности)

Руководитель дипломного проекта _____ Р.А. Сафаров
(подпись, инициалы, фамилия)

Новосибирск, 2022 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Отделение информационных технологий и вычислительной техники

Утверждаю

Руководитель
отделения ИТБТ _____

_____ Рощенко О.Е.
(подпись, фамилия, инициалы)
« ____ » _____ 2022 г.

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ

студенту _____ Лепке Д.А. _____ группы _____ И-82
(фамилия, инициалы) (обучения)

1. Тема _____ Разработка информационной системы учёта гостиничных номеров

Утверждена приказом по НГТУ № 4953/2 от «22» декабря 2021 г.

2. Дата представления проекта к защите « ____ » _____ 2022 г.

3. Цели проекта (исходные данные) _____ Разработка информационной системы учёта
гостиничных номеров, которая представляет собой систему управления всеми номерами в
гостинице, их состоянием, а также имеет встроенную систему бронирования номеров
постояльцами.

- 4. Содержание пояснительной записки
 - 4.1. Введение
 - 4.2. Глава 1. Описание предметной области
 - 4.3. Обзор существующих решений
 - 4.4. Техническое задание
 - 4.5. Технические требования к работе системы
 - 4.6. Инструментальные средства разработки
 - 4.7. Технологии разработки
 - 4.8. Глава 2. Разработка системы
 - 4.9. Структура проекта
 - 4.10. Схема базы данных
 - 4.11. Описание системы
 - 4.12. Настройка сервера Ubuntu и запуск API
 - 4.13. Документация по API
 - 4.14. Создание консольного клиента
 - 4.15. Создание консольного автотестера
 - 4.16. Демонстрация работы программ
 - 4.17. Заключение
 - 4.18. Список используемых источников
 - 4.19. Приложение

5. Перечень графического и (или) иллюстрационного материала

.....
.....
.....

Руководитель проекта

(подпись, дата)

Сафаров Р.А.

(фамилия, инициалы)

Задание принял к исполнению

(подпись студента, дата)

Лепке Д.А.

(фамилия, инициалы студента)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	5
ГЛАВА 1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ	7
1.1. Обзор существующих решений	7
1.1.1. Booking.com	7
1.1.2. Яндекс.Путешествия	9
1.1.3. Ostrovok.ru	10
1.2. Техническое задание	10
1.3. Технические требования к работе системы	14
1.4. Инструментальные средства разработки	15
1.5. Технологии разработки	16
ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ	18
2.1. Структура проекта	18
2.2. Схема базы данных	18
2.3. Описание системы	19
2.3.1. Создание проекта в Visual Studio	19
2.3.2. Создание моделей данных	19
2.3.3. Создание вспомогательных классов	19
2.3.4. Создание классов по работе с БД	21
2.3.5. Создание API-контроллеров	22
2.4. Настройка сервера Ubuntu и запуск API	23
2.5. Документация по API	23
2.5.1. Room Methods	24
2.5.2. Category Methods	26
2.5.3. Reserve Methods	28
2.6. Создание консольного клиента	31
2.6.1. Создание обёртки над API	31
2.6.2. Создание основной логики	33
2.7. Создание консольного автотестера	34
2.8. Демонстрация работы программ	36
2.8.1. Демонстрация работы клиента	36
2.8.2. Демонстрация работы автотестера	38
ЗАКЛЮЧЕНИЕ	40
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	41
Приложение	42

ВВЕДЕНИЕ

Гостиничный бизнес — одна из составляющих индустрии гостеприимства. Рынок гостиничных услуг, прежде всего, ориентирован на работу с клиентами, на создание безопасных и комфортных условий проживания и предоставление полного перечня современных информационных услуг. Уровень сервиса напрямую влияет на классность гостиничного объекта и определяет его конкурентные преимущества на рынке. Одну из главных функций в организации высококлассного сервиса в гостиничном комплексе выполняют системы автоматизации, отвечающие не только за инженерные и телекоммуникационные процессы в здании, но и за все этапы работы с клиентами, начиная бронированием номера и заканчивая созданием базы данных о предпочтениях постоянных гостей.

Системы автоматизации в гостиничных комплексах, равно как и в других объектах недвижимости, направлены на перевод многих операций в автоматический режим, что предполагает контроль всех рабочих процессов, обработку документов и данных при минимальном участии персонала. Каждая такая система состоит из программного комплекса систем управления, автоматизированных рабочих мест и т.д.

Гостиница — имущественный комплекс (дом, здание, часть здания, иные постройки) с меблированными комнатами («номерами») для временного проживания.

С точки зрения функционирования или структуры гостиницы, можно сказать, что гостиница располагает номерами с разным уровнем сервиса, комфортности и, соответственно, оплаты. Номера могут быть разных типов: люкс — многокомнатный номер с высоким уровнем сервиса, комфортности и обслуживания; полулюкс — номер меньшей, чем люкс, площади, но с достаточным уровнем сервиса и комфортности; обычный номер — с минимальным уровнем сервиса. В гостинице ведется учет состояния номеров.

Все прибывающие и размещаемые в гостинице клиенты при вселении должны заполнить карточку регистрации, в которой необходимо указать фамилию, имя, отчество, телефон, время заселения, время отъезда.

Любой номер гостиницы имеет номер, по которому ведется учет клиентов, проживающих в гостинице.

Также гостиница предоставляет возможность бронирования номеров.

Таким образом, в функционирование гостиницы входит:

1. Регистрация клиентов.
2. Учет состояния номеров.
3. Прием заявок на бронирование номеров.
4. Расчет стоимости проживания.

Учитывая все вышесказанное, становится понятна актуальность создания удобной информационной системы учета гостиничных номеров. Целью данной работы является разработка информационной системы учета гостиничных номеров, в которой будут храниться параметры и состояние номеров, а также будет возможность бронирования номеров постояльцами.

ГЛАВА 1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор существующих решений

Существует огромное количество сайтов и сервисов по подбору и бронированию номеров в множестве гостиниц мира, но, к сожалению, не удалось найти ни одного программного продукта, реализующего собственный учёт гостиничных номеров и собственную систему бронирования (чем, например, является данный дипломный проект). По этой причине автор решил рассмотреть преимущества и недостатки тех самых сайтов и сервисов по поиску и бронированию номеров, так как такие решения по своему функционалу схожи представленному в этом проекте.

1.1.1. Booking.com

Это система онлайн-бронирования, существующая с 1996 года. Проверенная временем, сегодня она лидирует по количеству проданных номеров. Ежемесячно более 30 млн туристов и бизнесменов со всего мира пользуются её услугами.

Booking.com представляет собой интернет-сервис, через который люди самостоятельно ищут и резервируют проживание в гостинице практически в любой точке мира. На данном ресурсе можно:

- Узнать об отелях в том городе, куда вы собрались поехать.
- Сравнить цены.
- Посмотреть фотографии гостиниц и окрестностей.
- Рассчитать расстояние до музея, ресторана или магазина.
- Забронировать номер в гостинице и многое другое.

База booking.com содержит данные об отелях более чем в 200 странах мира. Вы можете забронировать номер как в популярных странах: Турции, Италии и др., – так и в малопосещаемых с точки зрения туризма государствах: Анголе, на островах Кука, Палестинских территориях и пр.

Система предлагает воспользоваться бронированием без комиссии и регулярными специальными предложениями. Вам доступны сотни тысяч

вариантов размещения: свыше 500 тыс. альтернатив, более 200 тыс. вариантов аренды для отпуска, более 70 тыс. направлений, свыше 200 стран.

Вы легко можете управлять бронированием онлайн: отменять или изменять бронь, отправлять особые пожелания администрации того объекта, где бронируете номер, и многое другое. Большинство номеров в отелях можно забронировать без предоплаты. Для гарантии бронирования все же номер карты придется оставить на сайте. Это безопасно и хранится в защищенной системе.

Здесь легко ознакомиться с 40 млн подлинных отзывов об отелях. Отзыв могут оставить только те, кто останавливался в отелях, что исключает вариант накрутки отзывов и оценок ботами.

Преимущества:

- Популярный международный сервис.
- Частые спецпредложения, помогающие сэкономить.
- Огромная база отелей и гостиниц по всему миру.
- Высокая надёжность сервиса, исходя из отзывов пользователей.
- Полностью автоматизированная система, к которой подключены партнёры сервиса, что позволяет отображать верную стоимость бронирования номеров и обновлять эту стоимость в реальном времени.
- Удобный и интуитивно понятный интерфейс сайта.

Недостатки:

- Некоторые отели приукрашивают состояние номеров, информацию о них, выставляют старые фотографии комнат, на которых помещения представлены в самом лучшем виде.
- В случае отмены брони возврат средств на карту не редко затягивается на месяц.
- В случае серьёзных проблем с номером, сервис часто заявляет, что он всего лишь посредник и человеку необходимо обратиться напрямую в отель, в котором был забронирован номер.
- За отмену брони бывают штрафы.

- Редко случаются проблемы с автоматизированной системой, из-за чего до гостиницы может не дойти заявка о брони, которую человек уже оплатил через сайт сервиса.

1.1.2. Яндекс.Путешествия

Возможно, вы будете удивлены, но популярный российский поисковик имеет собственную систему Yandex Travel, через которую можно забронировать и отель. База партнеров сервиса насчитывает свыше 270 тысяч отелей и гостиниц по всему миру.

Сервис Яндекс.Путешествия помогает путешественникам подбирать отели, билеты на самолёт, поезд и автобус. Пользователи сравнивают цены, просматривают отзывы и оформляют покупку за считанные минуты, не тратя время на блуждание по сторонним сайтам.

База сервиса включает более 3 млн гостиниц, санаториев, турбаз, хостелов и апартаментов от агрегаторов и владельцев отелей. Здесь же доступны актуальные акции и спецпредложения, описания, фотографии, рейтинг и отзывы от реальных клиентов.

- Часто цены ниже, чем у конкурентов.
- Большой выбор отелей, гостиниц, рейсов.
- Подробная информация.

Недостатки:

- Многие пользователи в своих отзывах отмечают очень слабую клиентскую поддержку.
- Могут не предупредить об отмене рейса, а иногда и брони в отеле или гостинице.
- Часто заявляет в случае возникновения каких-либо проблем у пользователя, что он всего лишь посредник, а поэтому человеку необходимо обращаться напрямую в отель/гостиницу, в котором он забронировал номер.

1.1.3. Ostrovok.ru

Перечислять регалии портала Ostrovok.ru можно очень долго. Авторитет этому сайту придает уже тот факт, что знаменитым National Geographic он признан лучшим сервисом для путешественников. В базе портала хранится подробнейшая информация более чем о 500000 различных отелей по всему миру, в том числе и в самых скромных городах.

Преимущества:

- Высококачественная круглосуточная поддержка клиентов.
- Огромное количество партнёров, а значит и отелей/гостиниц в базе.
- Гарантия лучшей цены - конкурентное преимущество “Островка”.
- Частые скидочные акции в соц. сетях.

Недостатки:

- При бронировании номера из-за границы множество пользователей отмечает, что информация о брони доходит до отеля/гостиницы с задержкой в несколько часов.

1.2. Техническое задание

Необходимо разработать информационную систему для учёта гостиничных номеров. Система требуется для следующих целей:

1. Централизованное управление гостиничными номерами, их параметрами, бронированием номеров.
2. Систематизация и структуризация данных о гостиничных номерах, посетителях.
3. Накопление и обработка данных гостиничных номеров, посетителей.

Полноценная система должна состоять из трёх частей:

1. Серверное (главное) ПО, т.е. API, с которым будут взаимодействовать приложения. API (программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface, API [эй-пи-ай]) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может

взаимодействовать с другой программой. Обычно входит в описание какого-либо интернет-протокола (например, RFC), программного каркаса (фреймворка) или стандарта вызовов функций операционной системы. Часто реализуется отдельной программной библиотекой или сервисом операционной системы. Используется программистами при написании всевозможных приложений.

2. Консольное приложение-клиент, с помощью которого можно будет взаимодействовать с серверной частью, обмениваться данными и управлять всей системой.
3. Консольное приложение-автотестер, которое в автоматическом режиме и при минимальном участии администратора протестирует всю функциональность ядра системы – серверной части.

Данная информационная система требуется для администратора гостиницы, т.е. функциональность разделения ролей и прав доступа не нужна, у администратора должны быть максимальные права изначально.

Данные информационной системы (база данных и побочные файлы/документы, генерируемые системой в ходе её работы) должны храниться тут же на сервере, по API-запросу обновляться/изменяться и выдаваться в виде JSON-ответа запрашивающим информацию приложениям. Консольные программы (клиент и автотестер) не должны хранить никаких данных локально, они должны в реальном времени взаимодействовать с API без кеширования результатов такого взаимодействия.

В серверном ПО должна быть реализована следующая функциональность:

1. Получение информации обо всех добавленных в базу данных гостиничных номерах, их категориях и статусах.
2. Добавление гостиничных номеров в базу данных с такими параметрами: номер комнаты, категория номера, статус номера (по умолчанию «свободен»).
3. Изменение параметров гостиничного номера: изменение категории номера, номера комнаты.

4. Удаление гостиничных номеров и всей хранящейся информации о них.
5. Получение информации обо всех добавленных категориях номеров с их именами и стоимостью аренды номера за сутки в каждой категории.
6. Добавление категорий в базу данных с такими параметрами: имя категории, стоимость суточной аренды номера в данной категории.
7. Изменение параметров категорий: изменение имени, стоимости суточной аренды номера в категории.
8. Удаление категории и всей информации о ней в базе данных.
9. Получение информации о всех забронированных номерах в гостинице с указанием полного имени и телефона человека, забронировавшего номер.
10. Бронирование номеров, где нужно вводить ФИО и телефон того, кто бронирует номер, указывать комнату, которую необходимо закрепить за данным лицом, определять количество дней аренды номера. Номер, который забронировали, должен получать статус «Забронирован» и быть в дальнейшем недоступным для повторного бронирования кем-либо другим до тех пор, пока не будет снята текущая бронь.
11. Снятие брони. При выполнении данной функциональности освобождающемуся номеру должен автоматически выставляться статус «свободен» в базе данных.

В консольной программе-клиенте должна быть реализована следующая минимальная функциональность:

1. Возможность запрашивания всей хранимой информации о добавленных гостиничных номерах, доступных категориях, бронированиях у серверного ПО и их немедленное отображение в программе.
2. Возможность добавлять новые гостиничные номера, категории и брони, формируя API-запрос к серверному ПО с требуемыми данными для добавления.

3. Возможность производить изменение параметров гостиничных номеров, категорий, формируя API-запрос к серверному ПО с необходимыми для этого действия данными, введенными администратором.
4. Возможность выполнять удаление добавленных ранее гостиничных номеров, категорий и броней, формируя соответствующий API-запрос к серверному ПО по требованию администратора.

Взаимодействие с консольной программой-клиентом должно осуществляться при помощи клавиатуры, в формате текстового меню, т.е. меню в командной строке. Результаты выполнения каждой тестируемой функции должны отображаться на очищенном от предыдущих результатов выполнения функций экране.

В консольной программе-автотестере должна быть реализована следующая функциональность:

1. Автоматическое тестирование функционала добавления номеров, изменения их параметров, удаления номеров из базы данных и получение всех ранее добавленных номеров.
2. Автоматическое тестирование функционала добавления новых категорий номеров, изменения их параметров, удаления категорий из базы данных и получение всех ранее добавленных категорий.
3. Автоматическое тестирование функционала системы бронирования: добавление новой брони, изменение параметров существующей брони, удаление брони из базы данных и вывод всех ранее добавленных броней.

Консольная программа-автотестер должна быть минималистичной по своей сути, по формату вывода информационных сообщений, по использованным решениям формирования пользовательского меню. Программа должна предполагать минимальное участие администратора в её работе – от администратора должно требоваться только единожды выбрать в автотестере, что ей нужно тестировать в системе. Далее программа должна выполнить тестирование всех выбранных компонентов и вывести на экран результат

тестирования как каждого компонента по отдельности со статусом выполненного теста этого компонента (удачно/неудачно), так и общий результат удачных/неудачных тестов за цикл тестирования.

1.3. Технические требования к работе системы

Основные требования к работе серверного (главного) ПО:

- Корректная обработка API-запросов приложений.
- Безошибочное выполнение запрошенных функций API-методов.
- Выдача корректных и ожидаемых JSON-ответов для дальнейшей обработки в приложениях.

Основные требования к работе консольной программы-клиента:

- Полнофункциональное консольное меню с корректной обработкой ввода и вывода данных.
- Реализация всех или почти всех (не менее 90% рабочих функций) функций серверного ПО и информационной системы.
- Корректная обработка JSON-ответов и вывод этих самых ответов от серверного ПО на консоль.

Основные требования к работе консольной программы-автотестера:

- Полнофункциональное консольное меню для стартового выбора компонентов для тестирования, с корректной обработкой ввода, а также с корректным выводом результирующих данных.
- Реализованные рабочие тесты для всех компонентов информационной системы, т.е. всех функций серверного ПО.
- Корректная обработка полученных ответов в JSON-формате от серверного ПО, их обработка и вывод на экран.

1.4. Инструментальные средства разработки

Visual Studio 2022 Community

Visual Studio 2022 Community — бесплатная комплексная IDE для разработчиков. Включает первоклассную поддержку разработки для сети, облака и игр, а также отличные инструменты для создания кроссплатформенных мобильных приложений.

Основные преимущества:

- Встроенный Web-сервер.
- Поддержка множества языков при разработке.
- Меньше кода для написания, т.е. присутствуют шаблоны кода для многих языков программирования.
- Интуитивный стиль кодирования.
- Возможность отладки программ.
- Более высокая скорость разработки в связи с наличием интеллектуальных функций дополнения и анализа кода.
- Огромное количество настраиваемых горячих клавиш.
- И многое другое.

Fiddler 4

Fiddler 4 — популярный прокси-сервер, служащий для тестирования сетевых запросов, перехвата и просмотра http(s)-трафика установленных программ.

Основные преимущества:

- Кроссплатформенность.
- Удобная панель инструментов с такими функциями как очищение сессии, редактирование и перевыполнение запроса, остановкой перехвата трафика и другими.
- Классификация запросов по хостам.
- Возможность экспортировать настройки.

- Возможность просматривать несколько сетевых сессий в соседних закладках.
- Возможность просмотра содержимого зашифрованных https-запросов.
- И многое другое.

1.5. Технологии разработки

C#

C# (произносится как "си шарп") — современный объектно-ориентированный и типобезопасный язык программирования. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript.

Основным языком программирования для разработки системы был выбран по следующим причинам:

- Удобный и элегантный синтаксис, который легко писать, читать и понимать.
- Лёгкое внедрение асинхронности (параллельности) в ПО.
- Обширная база как встроенных, так и сторонних библиотек, написанных на C#.

ASP.NET Core

ASP.NET Core является кроссплатформенной, высокопроизводительной средой с открытым исходным кодом для создания и выполнения современных облачных приложений, подключенных к Интернету. ASP.NET Core позволяет выполнять следующие задачи:

- Создавать веб-приложения и службы, приложения Интернета вещей (IoT) и серверные части для мобильных приложений.
- Выполнять развертывания в облаке или локальной среде.
- Использовать избранные средства разработки в Windows, macOS и Linux.

ASP.NET Core – своего рода веб-сервер, который легко выдерживает высокую нагрузку, одновременную обработку множества запросов (встроенная асинхронность). В системе на данной платформе создано и выполняется серверное ПО (API).

LiteDB

LiteDB — мощная, современная, потокобезопасная с открытым исходным кодом NoSQL база данных, написанная на C#.

Для API данное NoSQL-решение было выбрано по причинам своей популярности, удобства использования, встроенной поддержке асинхронности. С NoSQL базой данных никаких SQL-запросов писать не надо, достаточно всего лишь создать классы (модели данных) и отправить в LiteDB объект класса с данными, чтобы сохранить эти данные в базу. При получении данные тоже возвращаются в виде объекта нужного класса, из-за чего сохранять/получать данные из БД становится невероятно простым делом.

ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ

2.1. Структура проекта

Проект состоит из трёх частей:

- Серверное ПО (API), называется Hotel.API.
- Консольная программа-клиент, называется HotelAPIClient.
- Консольная программа-автотестер, называется HotelAPIAutotester.

Каждая часть расположена в отдельной папке и в отдельном GIT-репозитории.

2.2. Схема базы данных

База данных состоит из трёх таблиц (в NoSQL принято называть их коллекциями, т.к. внутри движка это преобразуется всё в коллекции C#):

- Коллекция Room используется для хранения комнат. Поля коллекции такие:
 - Id (int) – уникальный идентификатор записи в БД.
 - Number (int) – номер комнаты.
 - Category (Category) – объект категории, к которой относится данная комната (например, объект категории «люкс»).
 - RoomStatuses (enum) – перечисление статусов комнаты, в данном случае объект хранит текущий статус комнаты (например, Free – «свободен»).
- Коллекция Category используется для хранения категорий номеров.

Коллекция содержит такие поля:

- Id (int) – уникальный идентификатор категории в БД.
 - Name (string) – имя категории (например, «люкс»).
 - Price (int) – стоимость суточной аренды номера в данной категории.
- Коллекция Reserve используется для хранения броней. Содержит следующие поля:
 - Id (int) – уникальный идентификатор брони в БД.
 - Room (Room) – объект комнаты, которая была забронирована.

- `PrimaryPerson (Person)` – объект, представляющий человека, который забронировал номер. Объект `Person` содержит поля `Id`, `FullName` (ФИО) и `Phone` (телефон).
- `Guests (List<Person>)` – список объектов гостей.
- `StartDate (DateTime)` – объект, представляющий дату регистрации брони, т.е. начальную дату бронирования (с).
- `EndTime (DateTime)` – объект, представляющий конечную дату бронирования номера, т.е. дату, до которой человек забронировал номер.

2.3. Описание системы

2.3.1. Создание проекта в Visual Studio

В IDE Visual Studio создаём проект типа `.NET Core WEB API`, даём ему название `Hotel.API`. После создания проекта необходимо установить некоторые зависимости, для этого переходим в окно диспетчера пакетов `Nuget`, в строке поиска пакетов пишем «`LiteDB`» (движок базы данных) и устанавливаем его.

2.3.2. Создание моделей данных

Прежде чем приступать к написанию логики, нужно создать модели данных, с которыми мы будем работать в проекте: сохранять в БД, получать из неё и т.д.. Для этого в обозревателе решения Visual Studio создаём папку `Models`, в которой создаём классы, описывающие комнаты, категории, человека, бронь: `Room`, `Category`, `Person`, `Reserve`. Свойства данных моделей можно увидеть в п. 2.2.

Далее нам нужно создать модель ответа на API-запрос, а также вспомогательный класс `JsonParsingResult`, в который будут десериализовываться JSON, отправленные POST-запросами этому API, чтобы можно было работать с полученными данными как объектами.

2.3.3. Создание вспомогательных классов

Для полноценного оперирования данными нам не хватает нескольких классов, а именно необходимо:

- Создать папку `Enums` и в ней создать 2 класса-перечисления:

- RoomStatuses – перечисление статусов комнаты (номер свободен, занят, забронирован). Свойство с таким типом есть в модели данных Room.
- OperationStatus – общее перечисление статусов операции (Ok, Error). Перечисление активно используется в классе API-ответа и в классе, куда десериализуются данные из JSON. Перечисление указывает на успешность или неудачность той или иной операции, что позволяет приложениям, получающим ответ от API, сразу проверять поле «status» и понимать, успешно ли выполнен их запрос или нет.
- Создать папку Interfaces, в которой создать один интерфейс IOperation с такими свойствами:
 - Status типа OperationStatus – перечисление статусов выполнения операции.
 - ErrorInfo типа Exception – объект ошибки, в которой будет записана исходная ошибка, в случае, если OperationStatus будет равен Error. Тоже требуется для приложений, работающих с API, чтобы получать полную информацию о том, какая ошибка возникла и почему.
- Создать папку Extensions и в ней создать класс метода-расширения HttpExtensions. Методы, определённые в данном классе-расширении, позволяют облегчить выполнение некоторых манипуляций с http-запросом (объектом HttpRequest).
- Создать папку Middlewares, в которой будут храниться классы, представляющие промежуточные слои подпрограмм, с помощью которых можно на лету изменять объект запроса к API — HttpRequest, а также объект ответа от API — HttpResponse. В данном случае мы создаём там один класс-подпрограмму ErrorHandlingMiddleware, который будет на лету и в одном месте обрабатывать неожиданные ошибки, произошедшие во время выполнения

запроса к API, и выдавать информацию о них в удобочитаемом виде запрашивающему приложению.

2.3.4. Создание классов по работе с БД

В обозревателе решения создаём папку Database, в которой создаём следующие классы:

- Общий и самый главный статический класс LDB, который представляет собой агрегатора как всех методов по работе с БД, так и всех API-методов.
- Классы API-методов:
 - RoomMethods – класс, в котором определены методы по работе с комнатами:
 - GetAll – получить список всех комнат из БД.
 - Add – добавить новую комнату.
 - Edit – отредактировать данные добавленной ранее комнаты.
 - Delete – удалить комнату из базы.
 - CategoryMethods, в котором определены методы по работе с категориями:
 - GetAll – получить список всех категорий из базы.
 - Add – добавить новую категорию в БД.
 - Edit – редактировать параметры существующей категории в базе.
 - Delete – удалить категорию из базы.
 - ReserveMethods, в котором определены классы по работе с бронированиями:
 - GetAll – получить список всех существующих броней.
 - Add – добавить новую бронь.
 - Edit – редактировать данные существующей в БД брони.
 - Delete – удалить бронь из базы.

В дальнейшем, при ответе на API-запрос приложений, мы будем получать/отправлять данные в базу, используя лишь общий класс LDB (например, `LDB.Room.GetAll()`), что весьма удобно.

2.3.5. Создание API-контроллеров

Последними классами, в которых нуждается проект, являются классы контроллеров, которые, собственно, и обрабатывают API-запросы сторонних приложений. Для создания контроллеров в обозревателе решения Visual Studio создаём новую папку с именем `Controllers`, в которой создаём такие классы-контроллеры и сразу определяем в них необходимые API-методы:

- `RoomsController`, содержащий API-методы по работе с комнатами:
 - `GetAll` типа `HttpGet`.
 - `Add` типа `HttpPost`.
 - `Edit` типа `HttpPost`.
 - `Delete` типа `HttpPost`.
- `CategoriesController`, содержащий API-методы по работе с категориями номеров:
 - `GetAll` типа `HttpGet`.
 - `Add` типа `HttpPost`.
 - `Edit` типа `HttpPost`.
 - `Delete` типа `HttpPost`.
- `ReservesController`, содержащий API-методы по работе с системой бронирования:
 - `GetAll` типа `HttpGet`.
 - `Add` типа `HttpPost`.
 - `Edit` типа `HttpPost`.
 - `Delete` типа `HttpPost`.

Все API-методы, тип которых `HttpPost`, ожидают на вход JSON-объект соответствующей модели данных. Приложения обязаны знать эти модели и сериализовать JSON из верных моделей.

2.4. Настройка сервера Ubuntu и запуск API

Для запуска, созданного в предыдущем разделе проекта API, требуется кроссплатформенная среда выполнения dotnet. На Ubuntu её устанавливаем простой командой: `sudo apt install dotnet-sdk-2.1`

После установки скомпилированный ранее проект API запускаем не менее простой командой: `dotnet Hotel.API.dll`

Теперь проект работает на порту 7071, который, честно говоря, не хотелось бы светить в мир. Для решения этой проблемы необходимо установить веб-сервер, который будет слушать только допустимые для автора дипломного проекта порты, т.е. 443 SSL. Устанавливаем веб-сервер командой: `sudo apt install nginx`

После установки создаём виртуальный хост для нужного домена, т.е. в моём случае `hotel-api.tj.fyi`, который будет слушать исключительно 443 порт, для чего у автора дипломного проекта уже есть валидные SSL-сертификаты, которые он и указывает в конфиге хоста `nginx`.

Теперь API работает по `https` протоколу и на адресе `https://hotel-api.tj.fyi`

2.5. Документация по API

Демонстрационная версия API, запущенная на сервере автора: `https://hotel-api.tj.fyi`

Все методы возвращают один и тот же JSON-объект с такими полями:

- `response` – хранит запрошенный объект или массив объектов, может быть пустым в случае ошибки.
- `errorMsg` – хранит текстовое описание ошибки, если при выполнении или обработки запроса произошла ошибка. Если ошибок нет, то поле пустое.
- `status` – показывает статус успешности запроса: если нет ошибок, поле имеет значение 200, в случае ошибки -1. При обработке ответа от API это поле точно покажет вам статус вашего запроса, стоит ориентироваться на него в первую очередь.

2.5.1. Room Methods

Методы управления комнатами и их данными в системе.

GET /rooms.getAll

Получить список всех добавленных комнат.

Параметры

Метод не принимает параметров.

Результат

В случае успешного выполнения вернёт массив объектов комнат. Обратите внимание, что массив изначально не отсортирован по номеру комнаты, вам стоит сделать это самостоятельно.

Пример ответа

```
{"response":[{"id":1,"number":13,"category":{"id":2,"name":"Обычный","price":1300},"status":1},{"id":2,"number":15,"category":{"id":3,"name":"Люкс","price":2500},"status":0}],"errorMsg":null,"status":200}
```

POST /rooms.add

Добавить новую комнату в систему.

Параметры

Метод принимает JSON, представляющий объект новой комнаты с такими полями:

- Id [int] – уникальный идентификатор комнаты. Рекомендуется создавать новую комнату с нулевым идентификатором, тогда сервер корректно сможет изменить его на нужный.
- Number [int] – номер новой комнаты.
- Category [Category] – объект категории, которая будет присвоена новой комнате (предварительно необходимо запросить список всех категорий у сервера методом categories.getAll и в этот объект подставить объект нужной).

- Status [int] – состояние гостиничного номера, может быть: 0 – свободен, 1 – забронирован, 2 – занят. Данное поле и его значение никак не влияет на действительное состояние новой комнаты, т.к. по умолчанию сервер присвоит новой комнате состояние «свободен».

Результат

В случае успешного добавления новой комнаты вернёт статус 200 в соответствующем поле JSON. В случае ошибки статус будет -1, а в поле errorMsg - описание ошибки.

Пример ответа

```
{"response":null,"errorMsg":null,"status":200}
```

POST /rooms.edit

Отредактировать параметры существующей комнаты.

Параметры

Метод принимает объект существующей комнаты с изменёнными параметрами. Вы можете изменить почти все параметры комнаты: номер, категорию, статус. Список всех комнат вы можете получить методом rooms.getAll, а список всех категорий методом categories.getAll. Обратите внимание, что нельзя изменять Id комнаты, т.к. именно по Id сервер сможет найти комнату и изменить её параметры на новые.

Результат

В случае успешного выполнения вернёт объект комнаты с изменёнными параметрами и статус 200 в соответствующем поле.

Пример ответа

```
{"response":{"id":12,"number":83,"category":{"id":4,"name":"Люкс (MAX)","price":3600},"status":1},"errorMsg":null,"status":200}
```

POST /rooms.delete

Удалить комнату из системы.

Параметры

Метод принимает JSON-объект комнаты, которую требуется удалить. Нужный объект вы можете получить, запросив список всех комнат методом `rooms.getAll`.

Результат

В случае успешного выполнения вернёт статус 200 в соответствующем поле.

Пример ответа

```
{"response":null,"errorMsg":null,"status":200}
```

2.5.2. Category Methods

Методы управления категориями номеров.

GET /categories.getAll

Получить список всех добавленных категорий.

Параметры

Метод не принимает параметров.

Результат

В случае успеха вернёт массив объектов категорий и статус 200 в соответствующем поле. В случае ошибки статус будет -1, а в поле `errorMsg` – описание ошибки.

Пример ответа

```
{"response":[{"id":2,"name":"Обычный","price":1300}, {"id":3,"name":"Люкс", "price":2500}, {"id":4,"name":"Люкс (MAX)", "price":3600}, {"id":5,"name":"Элитный", "price":4400}], "errorMsg":null, "status":200}
```

POST /categories.add

Добавить новую категорию в систему.

Параметры

Метод принимает JSON-объект новой категории с такими полями:

- Id [int] – уникальный идентификатор категории. Рекомендуется создавать новую категорию с нулевым идентификатором, чтобы сервер корректно мог изменить его на верный.
- Name [string] – название новой категории. Обратите внимание, что название новой категории не должно дублировать название уже существующей категории, в противном случае вы получите ошибку.
- Price [int] – стоимость суточной аренды номера в данной категории.

Результат

В случае успешного выполнения вернёт статус 200 в соответствующем поле. В случае ошибки статус будет -1, а в поле errorMsg – описание ошибки.

Пример ответа

```
{"response":null,"errorMsg":null,"status":200}
```

POST /categories.edit

Отредактировать параметры существующей категории.

Параметры

Метод принимает JSON-объект категории с изменёнными параметрами. Изменить можно почти всё: название, стоимость суточной аренды. Список всех категорий можно получить методом categories.getAll.

Обратите внимание, что нельзя менять Id категории, т.к. именно по Id сервер сможет найти категорию и изменить её параметры.

Результат

В случае успешного выполнения вернёт объект категории с изменёнными параметрами и статус 200. В случае ошибки статус будет -1, а в поле errorMsg – описание ошибки.

Пример ответа

```
{"response":{"id":6,"name":"Отдыхаюшка СУПЕР", "price":5655},
"errorMsg":null,"status":200}
```

POST /categories.delete

Удалить категорию из системы.

Параметры

Метод принимает JSON-объект категории, которую нужно удалить. Объект вы можете получить, запросив список всех категорий методом `categories.getAll`.

Результат

В случае успешного выполнения вернёт статус 200. В случае ошибки статус будет -1, а в поле `errorMsg` – описание ошибки.

Пример ответа

```
{"response":null,"errorMsg":null,"status":200}
```

2.5.3. Reserve Methods

Методы управления системой бронирования.

GET /reserves.getAll

Получить список всех броней.

Параметры

Метод не принимает параметров.

Результат

В случае успешного выполнения вернёт массив объектов броней и статус 200 в соответствующем поле. В случае отсутствия броней статус будет 200, а поле `response` пустое.

Пример ответа

```
{"response":[{"id":1,"room":{"id":5,"number":31,"category":{"id":2,"name":"Обычный","price":1300},"status":0},"primaryPerson":{"id":0,"fullName":"Гартив Артур Дмитриевич","phone":"+79123456789"},"guests":null,"startDate":"2021-12-15T23:41:40.819+03:00","endDate":"2021-12-24T23:41:40.819+03:00"}, {"id":2,"room":{"id":7,"number":55,"category":{"id":4,"name":"Люкс
```

```
(MAX)","price":3600},"status":0},"primaryPerson":{"id":0,"fullName":"Пупкин
Василий Акакиевич","phone":"+78005553535"},"guests":null,"startDate":"2021-
12-16T06:15:51.969+03:00","endDate":"2021-12-
19T06:15:51.969+03:00"},"id":3,"room":{"id":6,"number":50,"category":{"id":5,"n
ame":"Элитный","price":4400},"status":0},"primaryPerson":{"id":0,"fullName":"Ке
ймануров
Ивглид
Анурьевич","phone":"+79996665578"},"guests":null,"startDate":"2021-12-
16T06:35:09.613+03:00","endDate":"2021-12-
19T06:35:09.613+03:00"}],"errorMsg":null,"status":200}
```

POST /reserves.add

Создать новую бронь.

Параметры

Метод принимает JSON-объект брони, который содержит такие поля:

- Id [int] – уникальный идентификатор брони. Рекомендуется при создании новой брони делать его нулевым, чтобы сервер корректно изменил его на верный.
- Room [room] – объект свободной комнаты, которую нужно забронировать. Объект комнаты можно предварительно получить из списка всех комнат, запросив его методом `rooms.getAll`. Далее необходимо перебрать список и найти комнату с нулевым статусом, т.е. свободную, и вставить этот объект в данное поле.
- PrimaryPerson [person] – объект, представляющий человека, бронирующего номер:
 - Id [int] – идентификатор человека, сделайте его нулевым.
 - FullName [string] – ФИО человека, бронирующего номер.
 - Phone [string] – номер телефона человека, бронирующего гостиничный номер. Валидация телефона на сервере не производится, разработчику придётся сделать это на своей стороне.

- Guests [person array] – массив объектов гостей. Объекты гостей представляют собой объекты person, свойства которых вы можете посмотреть в предыдущем пункте.
- StartDate [DateTime] – действительная дата начала аренды номера.
- EndDate [DateTime] – дата окончания аренды номера, которая должна быть вычислена разработчиком на своей стороне.

Результат

В случае успешного выполнения вернёт статус 200. В случае ошибки статус будет -1, а в поле errorMsg – описание ошибки.

Пример ответа

```
{"response":null,"errorMsg":null,"status":200}
```

POST /reserves.edit

Отредактировать существующую бронь.

Параметры

Метод принимает JSON-объект брони с изменёнными параметрами. Менять можно почти всё, кроме поля Id, т.к. именно по Id сервер найдёт бронь и изменит её данные на нужные. Список всех броней можно получить методом reserves.getAll.

Результат

В случае успешного выполнения вернёт объект брони с новыми параметрами и статус 200. В случае ошибки статус будет -1, а в поле errorMsg – описание ошибки.

POST /reserves.delete

Удалить бронь из системы.

Параметры

Метод принимает JSON-объект существующей брони, который можно получить, например, запросив список всех броней методом reserves.getAll.

Результат

В случае успешного выполнения вернёт статус 200. В случае ошибки статус будет -1, а в поле errorMsg – описание ошибки.

Пример ответа

```
{"response":null,"errorMsg":null,"status":200}
```

2.6. Создание консольного клиента

Для взаимодействия с нашим серверным ПО (API) требуется программа-клиент, которая будет отправлять запросы к серверной части, получать ответы и выводить их в удобочитаемом виде администратору. Подобные клиенты можно сделать как графическими (построить их на формах в Windows Forms, или на окнах WPF), так и консольными, т.е. сделать их своего рода утилитами командной строки, когда есть только тёмный фон, белые буквы и строка ввода – именно последний вариант и выбрал автор данного дипломного проекта. Клиент получится функциональным и минималистичным.

В Visual Studio создаём новый проект типа «Консольное приложение .NET Framework», даём ему название HotelAPIClient, выбираем версию платформы .NET Framework 4.6.

После создания проекта первоначальное, что необходимо и является самым главным – создание обёртки над API.

2.6.1. Создание обёртки над API

В папке проекта создаём папку с именем API, в которой:

- Создаём Папку Models и из одноимённой папки проекта API, переносим модели данных комнат, категорий, человека, брони, а также вспомогательный класс JsonParsingResult, в который будем десериализовать JSON-ответы от API.
- Создаём папку Interfaces, куда также, как с папкой Models, копируем уже существующий интерфейс IOperation из проекта API. Интерфейс нужен для создания однообразных классов, в обязательном порядке содержащих статус и объект ошибки среди своих свойств.

- Создаём папку Extensions, в которой один единственный класс-расширение HotelAPIExtensions. Там определяем некоторые вспомогательные методы-расширения, которые упрощают взаимодействие с полученными от API данными.
- Создаём папку Methods, в которой 3 класса:
 - RoomMethods – в данном классе содержатся методы для запроса списка всех комнат от API, добавление новой комнаты, редактирование и удаление. Все запросы к API преобразуются в JSON нужной модели – Room.
 - CategoryMethods – в этом же классе содержатся методы по получению списка всех добавленных категорий по API, добавление новых, редактирование и удаление. Все запросы к API в данных методах преобразуются в JSON модели Category.
 - ReserveMethods – этот класс содержит методы по работе с бронированием, т.е. метод получения списка всех существующих броней, добавление новых, редактирование и удаление. Все запросы к API из этих методов преобразуются в JSON с моделью Reserve.
- Создаём папку Enums, в которую из проекта API переносим хорошо известное нам перечисление статусов операции (OperationStatus) и статусов комнат (RoomStatuses).
- И последним этапом создаём главный статический класс HotelAPI, в котором указываем URL, по которому находится API и связываем все добавленные ранее классы и методы по получению/отправки методов на сервер.

После выполнения действий выше, работа с обёрткой становится невероятно удобной. Отправлять нужные запросы и получать нужные данные можно будет так: HotelAPI.Room.GetAll() (получить с сервера список всех комнат), HotelAPI.Room.Add(Room r) (добавить новую комнату в базу на сервере) и т.д..

2.6.2. Создание основной логики

Обёртка над API уже есть, значит осталось определить только логику взаимодействия с консолью. Первым делом создаём цикл в главном методе Main, который будет работать бесконечно, пока значение переменной showMenu (показ меню/работа программы) не станет ложным.

В цикле мы вызываем метод MainMenu(). В методе MainMenu() с помощью встроенных в .NET функций работы с консолью мы информируем пользователя о том, какие функции и под какими номерами находятся и начинаем ждать ввода пользователя. Когда пользователь выбрал цифру той функциональности, которой он хочет воспользоваться, мы сравниваем в switch..case введённое значение с допустимыми, если выбранная функция существует, то в соответствующем case мы вызываем метод нужной функции.

Например, пользователь ввёл цифру 1, а это получение списка всех добавленных гостиничных номеров, значит мы выбираем case 1, а там вызывается метод GetAllRooms().

Во всех методах функций взаимодействие с API происходит через ранее определённый главный класс обёртки, т.е. HotelAPI. Ответы, полученные от API, проверяются на отсутствие ошибок и результат выполнения выводится пользователю на экран, который специально очищается для этого. Все методы асинхронные, т.е. главный поток программы никогда не блокируется и все операции выполняются параллельно.

Есть функции, в которых не требуется ввод пользователя, а есть те, в которых требуется. В этом случае в теле метода функции обязательно определены условия, отвечающие за проверку корректности ввода для той или иной функции API.

После получения результата запроса, на консоли появляется предложение вернуться в главное меню, нажав Enter.

Для более удобной работы с программой рекомендуется разворачивать командную строку во весь экран.

2.7. Создание консольного автотестера

Последней частью, составляющей систему учёта гостиничных номеров, является программа-автотестер. Данная программа при минимальном участии администратора в автоматическом режиме тестирует весь функционал информационной системы, т.е. её ядро/серверную часть (API). Подобный автотестер нужен, чтобы убедиться в работоспособности всех компонентов системы, в отсутствии каких-либо ошибок при обработке и выполнении. Автотестер было решено сделать по похожему с клиентом принципу, т.е. консольным и минималистичным.

В Visual Studio был создан проект типа «Консольное приложение .NET Framework», он был назван HotelAPIAutotester. Платформой, на которую нацелен проект, была выбрана версия .NET Framework 4.6.

После создания проекта логичным было бы как и в клиенте, создать обёртку над API, чтобы удобно отправлять запросы к серверной части, но её не понадобилось создавать повторно, так как у нас осталась функциональная и рабочая обёртка из HotelAPIClient, которая была успешно перенесена в проект автотестера.

Автотестер разработан по тем же принципам, что и клиент, т.е. всё тот же бесконечный цикл в методе Main(), всё тот же главный метод MainMenu() и switch..case, в котором сравнивается запрошенный администратором пункт и, в случае существования такого действия, вызывается нужная функция/метод. Единственное, в автотестере куда меньше пунктов меню, чем в клиенте, да и много пунктов здесь не нужно: тестирование функционала комнат, тестирование функционала категорий, тестирование функционала бронирования, выход.

Для упрощения запуска тестов, в проекте автотестера был создан большой по содержанию класс Tests всего с тремя методами: тестирование функционала комнат, тестирование функционала категорий, тестирование системы бронирования. Каждый метод содержит по 4 теста на компонент: получение всего списка, добавление, редактирование, удаление.

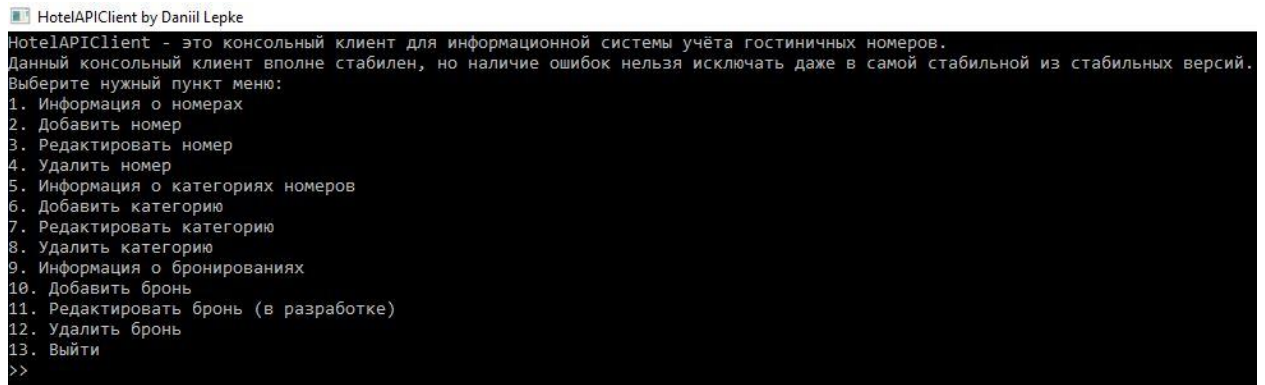
При запуске автотестера участие администратора минимально – ему нужно выбрать только, что тестировать, введя необходимую для этого цифру пункта меню. Далее система запускает тестирование и выполняет каждый тест, по окончании каждого теста показывается статус его выполнения. В конце тестирования на экран выводится общее количество тестов, общее количество удачных тестов и общее количество неудачных.

В проекте автотестера так же был создан класс RandomData, к которому обращаются методы из класса Tests, запрашивая для своих тестов случайные данные, чтобы тестировать систему на разных данных, а не на статичных.

2.8. Демонстрация работы программ

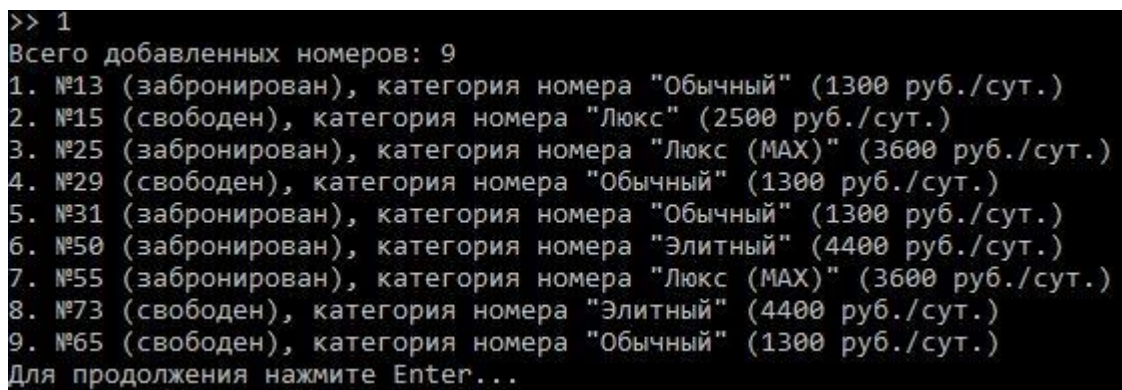
Ниже представлены скриншоты работы клиента и автотестера.

2.8.1. Демонстрация работы клиента



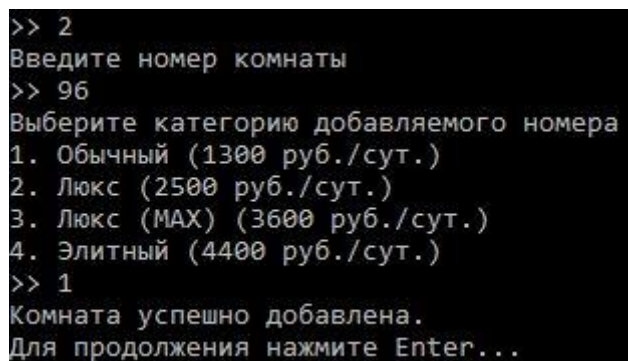
```
HotelAPIClient by Daniil Lepke
HotelAPIClient - это консольный клиент для информационной системы учёта гостиничных номеров.
Данный консольный клиент вполне стабилен, но наличие ошибок нельзя исключить даже в самой стабильной из стабильных версий.
Выберите нужный пункт меню:
1. Информация о номерах
2. Добавить номер
3. Редактировать номер
4. Удалить номер
5. Информация о категориях номеров
6. Добавить категорию
7. Редактировать категорию
8. Удалить категорию
9. Информация о бронированиях
10. Добавить бронь
11. Редактировать бронь (в разработке)
12. Удалить бронь
13. Выйти
>>
```

Рис. 1. Главное меню



```
>> 1
Всего добавленных номеров: 9
1. №13 (забронирован), категория номера "Обычный" (1300 руб./сут.)
2. №15 (свободен), категория номера "Люкс" (2500 руб./сут.)
3. №25 (забронирован), категория номера "Люкс (МАХ)" (3600 руб./сут.)
4. №29 (свободен), категория номера "Обычный" (1300 руб./сут.)
5. №31 (забронирован), категория номера "Обычный" (1300 руб./сут.)
6. №50 (забронирован), категория номера "Элитный" (4400 руб./сут.)
7. №55 (забронирован), категория номера "Люкс (МАХ)" (3600 руб./сут.)
8. №73 (свободен), категория номера "Элитный" (4400 руб./сут.)
9. №65 (свободен), категория номера "Обычный" (1300 руб./сут.)
Для продолжения нажмите Enter...
```

Рис. 2. Список всех комнат



```
>> 2
Введите номер комнаты
>> 96
Выберите категорию добавляемого номера
1. Обычный (1300 руб./сут.)
2. Люкс (2500 руб./сут.)
3. Люкс (МАХ) (3600 руб./сут.)
4. Элитный (4400 руб./сут.)
>> 1
Комната успешно добавлена.
Для продолжения нажмите Enter...
```

Рис. 3. Добавление новой комнаты

```
>> 1
Всего добавленных номеров: 11
1. №13 (забронирован), категория номера "Обычный" (1300 руб./сут.)
2. №15 (свободен), категория номера "Люкс" (2500 руб./сут.)
3. №25 (забронирован), категория номера "Люкс (МАХ)" (3600 руб./сут.)
4. №29 (свободен), категория номера "Обычный" (1300 руб./сут.)
5. №31 (забронирован), категория номера "Обычный" (1300 руб./сут.)
6. №50 (забронирован), категория номера "Элитный" (4400 руб./сут.)
7. №55 (забронирован), категория номера "Люкс (МАХ)" (3600 руб./сут.)
8. №73 (свободен), категория номера "Элитный" (4400 руб./сут.)
9. №65 (свободен), категория номера "Обычный" (1300 руб./сут.)
10. №95 (свободен), категория номера "Элитный" (4400 руб./сут.)
11. №96 (свободен), категория номера "Обычный" (1300 руб./сут.)
Для продолжения нажмите Enter...
```

Рис. 4. Новая комната добавлена

```
>> 5
Всего добавлено категорий: 4
1. Обычный (1300 руб./сут.)
2. Люкс (2500 руб./сут.)
3. Люкс (МАХ) (3600 руб./сут.)
4. Элитный (4400 руб./сут.)
Для продолжения нажмите Enter...
```

Рис. 5. Список всех категорий номеров

```
>> 6
Введите имя новой категории
>> Отдыхаюшка
Введите стоимость суточной аренды номера данной категории
>> 15000
Категория успешно добавлена.
```

Рис. 6. Добавление новой категории

```
>> 5
Всего добавлено категорий: 5
1. Обычный (1300 руб./сут.)
2. Люкс (2500 руб./сут.)
3. Люкс (МАХ) (3600 руб./сут.)
4. Элитный (4400 руб./сут.)
5. Отдыхаюшка (15000 руб./сут.)
Для продолжения нажмите Enter...
```

Рис. 7. Новая категория добавлена

```
>> 9
Всего броней: 4
1. Гартив Артур Дмитриевич забронировал(а) комнату №31 с 16.12.2021 3:41:40 до 25.12.2021 3:41:40, гости: 0
2. Пупкин Василий Акакиевич забронировал(а) комнату №55 с 16.12.2021 10:15:51 до 19.12.2021 10:15:51, гости: 0
3. Кеймануров Ивглид Ануриевич забронировал(а) комнату №50 с 16.12.2021 10:35:09 до 19.12.2021 10:35:09, гости: 0
4. Набудь Глеб Алексеевич забронировал(а) комнату №25 с 17.12.2021 0:39:22 до 21.12.2021 0:39:22, гости: 0
Для продолжения нажмите Enter...
```

Рис. 8. Список всех активных броней

```

>> 10
Выберите номер, который хотите забронировать
1. №13 (свободен), категория номера "Обычный" (1300 руб./сут.)
2. №15 (свободен), категория номера "Люкс" (2500 руб./сут.)
3. №29 (свободен), категория номера "Обычный" (1300 руб./сут.)
4. №73 (свободен), категория номера "Элитный" (4400 руб./сут.)
5. №65 (свободен), категория номера "Обычный" (1300 руб./сут.)
6. №95 (свободен), категория номера "Элитный" (4400 руб./сут.)
7. №96 (свободен), категория номера "Обычный" (1300 руб./сут.)
>> 7
Введите ФИО того, на кого будет забронирован номер
>> Кимрач Прохор Заирович
Введите телефон для связи
>> +73832405222
На сколько дней вы хотите забронировать номер?
>> 3
Подтвердите корректность данных брони
Комната №96
Бронирует Кимрач Прохор Заирович с номером телефона +73832405222
Бронь зарегистрирована 11.05.2022 19:54:02, а заканчивается 14.05.2022 19:54:02 = 3 дней
К оплате 3900 руб.

у/п >> у
Бронь успешно зарегистрирована.
Для продолжения нажмите Enter...

```

Рис. 9. Добавление новой брони

```

>> 9
Всего броней: 5
1. Гартив Артур Дмитриевич забронировал(а) комнату №31 с 16.12.2021 3:41:40 до 25.12.2021 3:41:40, гости: 0
2. Пупкин Василий Акакиевич забронировал(а) комнату №55 с 16.12.2021 10:15:51 до 19.12.2021 10:15:51, гости: 0
3. Кеймануров Ивгилд Анурьевич забронировал(а) комнату №50 с 16.12.2021 10:35:09 до 19.12.2021 10:35:09, гости: 0
4. Набудь Глеб Алексеевич забронировал(а) комнату №25 с 17.12.2021 0:39:22 до 21.12.2021 0:39:22, гости: 0
5. Кимрач Прохор Заирович забронировал(а) комнату №96 с 11.05.2022 19:54:02 до 14.05.2022 19:54:02, гости: 0
Для продолжения нажмите Enter...

```

Рис. 10. Новая бронь добавлена

2.8.2. Демонстрация работы автотестера

```

HotelAPIAutotester by Daniil Lepke

Данное ПО тестирует в автоматическом режиме критический функционал системы учёта гостиничных номеров.
Так как ядром системы является серверная часть (API), для работы автотестера требуется интернет-соединение.
=====
Что будем тестировать?
1. Функционал комнат (Room Methods)
2. Функционал категорий номеров (Category Methods)
3. Функционал системы бронирования (Reserve Methods)
4. Ничего, хочу выйти
>>

```

Рис. 11. Главный экран автотестера


```

>> 1
Запущено тестирование API-методов управления гостиничными номерами.
Всего тестов: 4
Тест №1. Получить список всех гостиничных номеров
Успех! Список получен. Всего номеров в гостинице: 11
Тест №2. Добавить новый гостиничный номер на сервер
Тестовые данные:
Номер комнаты: 155
Категория Обычный
Успех! Комната добавлена на сервер.
Тест №3. Редактировать параметры ранее добавленной комнаты 155 на сервере
Успех! Параметры комнаты 155 изменены.
Тест №4. Удалить ранее добавленную комнату 155 на сервере
Успех! Тестовая комната удалена с сервера.
=====
Тестирование завершено.
Успешных тестов: 4
Неудачных тестов: 0
Для продолжения нажмите Enter...

```

Рис. 12. Тестирование функционала комнат

```

>> 2
Запущено тестирование API-методов управления категориями гостиничных номеров.
Всего тестов: 4
Тест №1. Получить список всех добавленных категорий
Успех! Список категорий получен. Всего категорий: 5
Тест №2. Добавить новую категорию на сервер
Тестовые данные:
Имя категории: Царский
Цена аренды за сутки: 15344
Успех! Новая категория создана.
Тест №3. Редактировать параметры ранее добавленной тестовой категории
Успех! Данные ранее добавленной новой категории отредактированы на сервере.
Тест №4. Удалить ранее добавленную тестовую категорию
Успех! Категория была удалена с сервера.
=====
Тестирование завершено.
Успешных тестов: 4
Неудачных тестов: 0
Для продолжения нажмите Enter...

```

Рис. 13. Тестирование функционала категорий

```

>> 3
Запущено тестирование API-методов системы бронирования.
Всего тестов: 4
Тест №1. Получить список всех существующих броней
Успех! Список броней получен. Всего номеров забронировано: 5
Тест №2. Добавить новую бронь
Тестовые данные:
Бронируемый номер: 13, категория: Обычный
ФИО бронирующего: Орлов Марк Григорьевич
Успех! Новая бронь создана.
Тест №3. Редактировать ранее добавленную тестовую бронь
Успех! Данные тестовой брони отредактированы.
Тест №4. Удалить ранее добавленную тестовую бронь
Успех! Тестовая бронь удалена с сервера.
=====
Тестирование завершено.
Успешных тестов: 4
Неудачных тестов: 0
Для продолжения нажмите Enter...

```

Рис. 14. Тестирование системы бронирования

ЗАКЛЮЧЕНИЕ

В результате выполнения работы, целью которой являлось создание информационной системы учёта гостиничных номеров, были сформулированы требования к системе, изучена информация о данной предметной области, технологиях, с помощью которых спроектировано требуемое. С успехом была создана и запущена система, состоящая из трёх частей:

- Серверное ПО (API).
- Консольная программа-клиент.
- Консольная программа-автотестер.

Исходный код серверного ПО (API):

<https://github.com/Dyaminigo/Hotel.API>

Исходный код консольной программы-клиента:

<https://github.com/Dyaminigo/HotelAPIClient>

Исходный код консольной программы-автотестера:

<https://github.com/Dyaminigo/HotelAPIAutotester>

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Введение в ASP.NET Core: Microsoft Docs [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0> - Дата доступа: 11.05.2022
2. Учебник по языку C# 10 и платформе .NET 6: Metanit [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/tutorial/> - Дата доступа: 11.05.2022
3. ASP.NET Core | Полное руководство: Metanit [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/aspnet5/> - Дата доступа: 03.05.2022
4. Гостиница: Википедия [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/%D0%93%D0%BE%D1%81%D1%82%D0%B8%D0%BD%D0%B8%D1%86%D0%B0> - Дата доступа: 11.05.2022
5. LiteDB :: A .NET embedded NoSQL database: LiteDB [электронный ресурс]. – Режим доступа: <http://www.litedb.org/> - Дата доступа: 11.05.2022
6. Учебник. Создание веб-приложения ASP.NET Core на языке C# в Visual Studio: Microsoft Docs [электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/get-started/csharp/tutorial-aspnet-core?view=vs-2022> – Дата доступа: 11.05.2022

Модели данных Room

```
namespace Hotel.API
{
    public class Room
    {
        public int Id { get; set; }
        public int Number { get; set; }
        public Category Category { get; set; }
        public RoomStatuses Status { get; set; }
    }
}
```

Category

```
namespace Hotel.API
{
    public class Category
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Price { get; set; }
    }
}
```

Person

```
namespace Hotel.API
{
    public class Person
    {
        public int Id { get; set; }
        public string FullName { get; set; }
        public string Phone { get; set; }
    }
}
```

Reserve

```
using System;
using System.Collections.Generic;

namespace Hotel.API
{
    public class Reserve
```

```

    {
        public int Id { get; set; }
        public Room Room { get; set; }
        public Person PrimaryPerson { get; set; }
        public List<Person> Guests { get; set; }
        public DateTime StartDate { get; set; }
        public DateTime EndDate { get; set; }
    }
}

```

RoomStatuses

```

namespace Hotel.API
{
    public enum RoomStatuses
    {
        Free,
        Reserved,
        Occupied
    }
}

```