**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт сервиса и отраслевого управления
Кафедра бизнес – информатики и математики

# КУРСОВОЙ ПРОЕКТ

по предмету: Методы и средства проектирования компьютерных приложений
на тему: «Разработка автоматизированной информационной системы «Бронирование столиков»»

**Выполнил:**
Студент 2 курса группы НТм-23-1
направления 09.04.01
«Информатика и вычислительная техника»
Рогов Д. Ю.

**Руководитель:**
доцент, канд. пед. наук
Спирин И.С.

2025

## Техническое задание

Разработка автоматизированной информационной системы по предметной области «Бронирование столиков» на основе Blazor в среде разработки Microsoft Visual Studio. Основными функциями системы являются получение, обработка и хранение информации о занятости столиков в ресторане. Разработать систему со всеми нужными функциями для удобной работы с базой данных, создать интуитивный в использовании пользовательский интерфейс для приложения. Для разработки требуется:

- провести анализ предметной области;
- создать модель базы данных;
- разработать пользовательский интерфейс приложения;
- подключить созданную базу данных к приложению;
- разработать функционал, позволяющий добавлять, редактировать и удалять данные.

## Аннотация

**Цель** – Разработка автоматизированной информационной системы по предметной области «Бронирование столиков» на основе Blazor и разработка клиентского приложения.

**Объект работы** – клиентское приложение для работы с базой данных бронирования столиков.

**Предмет работы** – процесс разработки клиентского приложения для работы с базой данных бронирования столиков.

**Информационная база исследования:** статьи, учебники и научные работы в области создания АИС и работы с базами данных.

**Результат:** спроектированная база данных клиентов и бронирования столиков ресторана, разработанное клиентское приложение на основе Blazor с помощью языка программирования C#.

**Курсовой проект состоит** из введения, трёх глав, заключения, списка использованных источников и 6 приложений.

**Содержание**

## Введение

Современные заведения общественного питания стремятся автоматизировать как можно больше процессов, чтобы повысить уровень сервиса, сократить время обслуживания клиентов и исключить человеческий фактор в организационных вопросах. Одним из таких процессов является бронирование столиков. Традиционные способы — телефонные звонки или запись в бумажный журнал — становятся всё менее удобными как для клиентов, так и для персонала.

В условиях стремительного развития информационных технологий всё чаще внедряются автоматизированные системы, позволяющие пользователям через веб-интерфейс оформить бронирование в несколько кликов. Такие решения повышают лояльность клиентов, упрощают работу администратора и дают возможность гибко управлять загруженностью заведения.

В рамках данной курсовой работы была поставлена задача разработать автоматизированную информационную систему "Бронирование столиков", которая позволит пользователям выполнять бронирование онлайн, а администраторам — управлять заявками через удобный интерфейс. Система реализована с использованием современных технологий, включая язык программирования C# и фреймворк Blazor, который позволяет создавать интерактивные веб-приложения с единой логикой как на стороне клиента, так и сервера.

Цель проекта – разработка автоматизированной информационной системы, предназначенной для бронирования столиков в ресторане или кафе, с удобным веб-интерфейсом как для посетителей, так и для персонала.

Объект исследования – процесс бронирования столиков в заведениях общественного питания.

Предмет исследования – информационные технологии и методы, используемые для реализации автоматизированных систем бронирования.

Задачи проекта:

- проанализировать текущие подходы к бронированию столиков;
- сформулировать требования к функционалу будущей системы;
- спроектировать архитектуру приложения;
- реализовать пользовательский интерфейс и серверную часть с использованием C# и Blazor;
- организовать хранение данных о бронированиях;
- провести тестирование системы;
- подготовить инструкцию пользователя и отчет по выполненной работе.

## 1. Анализ предметной области

Бронирование столиков — важная составляющая работы заведений общественного питания, таких как кафе, рестораны, бары и другие. Этот процесс позволяет планировать количество гостей, избегать переполненности зала и обеспечивать высокий уровень обслуживания. В традиционном формате бронь оформляется устно — по телефону или при личном визите. Такой способ бронирования не лишён недостатков: высокая нагрузка на персонал, возможность ошибки при записи, отсутствие наглядной картины текущих броней, ограниченное рабочее время.

Основные проблемы традиционного подхода:

- человеческий фактор (ошибки при записи);
- отсутствие возможности быстро изменить или отменить бронь;
- недоступность записи в нерабочие часы;
- ограниченный контроль и аналитика по загруженности заведения.

С развитием информационных технологий стала возможной автоматизация этого процесса. Использование специализированных информационных систем позволяет клиенту самостоятельно выбрать дату, время и столик, а администратору — отслеживать брони в реальном времени.

Требования к информационной системе бронирования:

1. Удобный пользовательский интерфейс — простой и интуитивно понятный, доступный как с компьютера, так и с мобильного устройства.
2. Регистрация и идентификация клиентов (опционально) — для отслеживания истории бронирований.
3. Выбор даты, времени и количества гостей — основной функционал клиента.
4. Отображение доступных столиков — в режиме реального времени.

5. Исключение конфликтов по времени — невозможно забронировать уже занятый столик.

6. Обработка заявок администратором — подтверждение, изменение или удаление брони.

7. Хранение информации в базе данных — обеспечение устойчивости и сохранности данных.

8. Возможность масштабирования — в случае роста количества пользователей.

Целевая аудитория системы:

- клиенты (посетители заведения): нуждаются в простой и быстрой возможности забронировать столик без звонков и ожидания.
- персонал (администратор, управляющий): использует систему для контроля за бронированиями, аналитики посещений и оптимизации рабочего процесса.

Типовые сценарии использования:

- клиент заходит на сайт, выбирает дату и время, видит доступные столики и оформляет бронь;
- администратор просматривает список заявок, подтверждает или отклоняет бронь;
- клиент получает уведомление (на экране или через e-mail/SMS — опционально);
- администратор просматривает загруженность зала на конкретную дату и время.

Таким образом, автоматизация процесса бронирования столиков не только повышает удобство для клиента, но и оптимизирует внутреннюю работу заведения, позволяя снизить нагрузку на персонал и минимизировать ошибки. Это делает разработку подобной системы актуальной и востребованной задачей.

## 2. Этапы разработки, инструменты и методология

Разработка программного обеспечения включает в себя ряд последовательных этапов, каждый из которых влияет на конечное качество продукта. Для создания автоматизированной информационной системы «Бронирование столиков» использовалась итеративная (поэтапная) методология разработки**,** при которой проект создавался по частям, каждая из которых проходила цикл анализа, проектирования, реализации и тестирования. Такой подход позволил своевременно выявлять и устранять ошибки, а также гибко вносить изменения в структуру и функциональность приложения.

### 2.1 Методология разработки

Методология разработки базировалась на принципах итеративного и инкрементального подхода. Проект делился на логические блоки (например, интерфейс бронирования, обработка заявок, работа с базой данных), каждый из которых реализовывался отдельно, проходил проверку, после чего интегрировался в основную систему.

Преимущества выбранной методологии:

- постепенное наращивание функционала;
- возможность изменения требований на любом этапе;
- высокая управляемость процесса;
- удобство тестирования отдельных компонентов.

## 2.2 Инструменты разработки

Рисунок 2.1 – «логотип Blazor WebAssembly»

Blazor WebAssembly – это современный веб-фреймворк от Microsoft, позволяющий писать интерактивные пользовательские интерфейсы на языке C# вместо JavaScript. Blazor WebAssembly компилирует C#-код в WebAssembly, позволяя запускать приложение прямо в браузере клиента.

Преимущества использования Blazor:

- единый язык разработки (C#) как для клиентской, так и для серверной части;
- компонентный подход к построению интерфейса;
- высокая производительность и отзывчивость;
- хорошая интеграция с ASP.NET и Entity Framework;
- возможность повторного использования бизнес-логики между клиентом и сервером.

В рамках проекта на Blazor были реализованы все страницы приложения: главная, форма бронирования, список заявок, административная панель. Использовалась привязка данных (data binding), маршрутизация и компоненты Razor.


Рисунок 2.2 – «логотип ASP.NET Core»

ASP.NET Core — это кросс-платформенный веб-фреймворк от Microsoft, предназначенный для создания веб-приложений и API. Он обеспечивает высокую производительность, безопасность и модульность.

Основные возможности ASP.NET Core:

- создание REST API для взаимодействия между клиентом и сервером;
- обработка HTTP-запросов;
- middleware (промежуточная обработка запросов);
- аутентификация и авторизация (в случае необходимости);
- легкость в настройке и развертывании.

В рамках проекта ASP.NET Core использовался для создания REST API, обрабатывающего операции бронирования (создание, получение, удаление, проверка занятости столика). Он также обрабатывал связь между Blazor и базой данных через сервисный слой.


Рисунок 2.3 – «логотип Entity Framework Core»

Entity Framework Core — это ORM-библиотека от Microsoft, позволяющая работать с базой данных на уровне объектов C#, избегая написания SQL-запросов вручную.

Преимущества EFC:

- автоматическое создание базы данных и таблиц (Code First);
- работа с LINQ-запросами вместо SQL;
- поддержка миграций базы данных;
- упрощение CRUD-операций.

В рамках проекта с помощью EFC были созданы следующие модели:

- Table (Столик) — содержит номер, количество мест и статус;
- TableBusyness (Занятость стола) — включает дату, время, клиента, связанный столик;

Использовался In-Memory Database на этапе разработки и SQLite/SQL Server как основное хранилище. Все изменения структуры данных отслеживались через миграции.

### 2.3 Этапы разработки

1. Сбор и анализ требований: На данном этапе были определены потребности пользователей (гостей и администраторов заведения), составлен список функциональных требований к системе: возможность бронирования, отображение занятых и свободных столиков, редактирование и удаление заявок, предотвращение конфликтов по времени.
2. Проектирование архитектуры: Была выбрана модель клиент-сервер с использованием Blazor WebAssembly на клиенте и ASP.NET Core на сервере. Также спроектированы модели данных, отражающие реальные сущности предметной области: столик, бронь, пользователь. Другими словами, система была разделена на две основные части:

    a. frontend (пользовательский интерфейс на Blazor WebAssembly);
    b. backend (логика и обработка данных на ASP.NET Core).
3. Моделирование базы данных: Создана структура хранения информации о столиках, клиентах и бронированиях. Использовалась ORM-библиотека Entity Framework Core для работы с базой данных.
4. Реализация пользовательского интерфейса: Созданы страницы с возможностью выбора даты и времени, списком доступных столиков, формой бронирования и списком текущих заявок.
5. Реализация бизнес-логики: Добавлена проверка на доступность столиков, предотвращение наложения бронирований, сохранение информации в базу данных.
6. Тестирование: Проведено тестирование каждого компонента по отдельности и всего приложения в целом. Проверялись случаи корректного и некорректного ввода, а также конфликтные ситуации с пересечением времени.
7. Подготовка отчетной документации и инструкции пользователя.

## 3. Инструкция по использованию программы

Автоматизированная информационная система «Бронирование столиков» предоставляет пользователю простой и интуитивно понятный веб-интерфейс для оформления бронирования в заведении общественного питания. Взаимодействие с системой осуществляется через браузер без необходимости устанавливать дополнительное программное обеспечение.

Основные функции системы:

- поиск свободных столиков на указанную дату и время;
- бронирование столика;
- ввод контактных данных клиента;
- возможность отмены ранее сделанного бронирования.

### 3.1 Оформление бронирования

1. Бронирование стола не требует регистрации со стороны пользователя, поэтому вертикальный блок в левой части страницы содержит только ссылки на Домашнюю страницу («Home»), Бронирование и Таблицу занятости. Домашняя страница может содержать данные о самом ресторане, рекламировать их услуги, акции, предоставлять ссылки на контактную информацию и др.

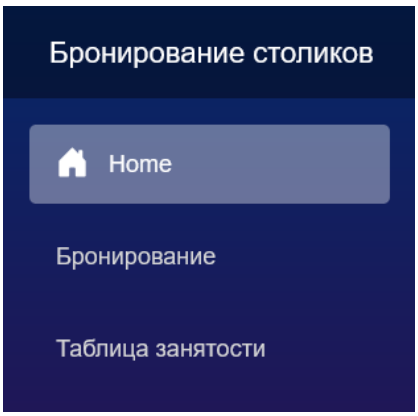


Рисунок 3.1 – «Меню веб-страницы – панель навигации по сайту»

2. Выбор параметров бронирования – на странице Бронирование пользователь видит форму, в которой необходимо указать:
   - количество человек;
   - желаемую дату бронирования;
   - предпочтительное время.

   Количество человек за столом должно быть не менее 1, и время бронирования по умолчанию стоит на 1 час, начиная с указанного времени. То есть – другой пользователь не сможет забронировать уже занятый стол.

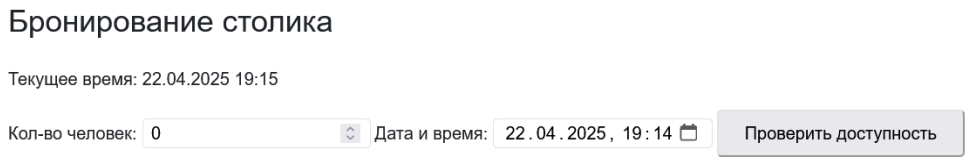


Рисунок 3.2 – «Поля бронирования столов»

3. Поиск подходящих столиков – после нажатия на кнопку "Проверить доступность", система проверяет базу данных и находит все свободные столики, которые соответствуют заданным параметрам (по вместимости и времени). Система автоматически предлагает первый подходящий по параметрам столик.

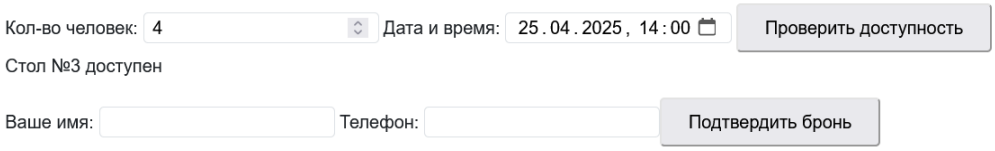


Рисунок 3.3 – «Поле подтверждения бронирования»

4. Если в системе есть свободный стол на указанное время и с подходящей вместимостью, то система предлагает ввести контактные данные – имя и номер телефона. Система выдает предупреждения в случае если пользователь вводит недоступные символы в поля регистрации (имя не

должно содержать цифр, номер не должен содержать букв). После ввода данных необходимо нажать кнопку "Подтвердить бронь".

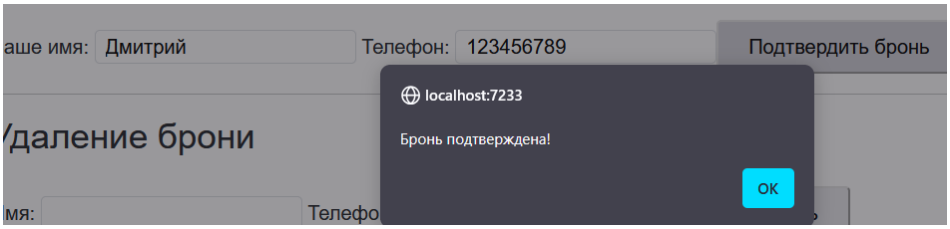


Рисунок 3.4 – «Сообщение о подтверждении бронирования»

5. Завершение бронирования – **с**истема сохраняет информацию в базе данных и уведомляет пользователя о том, что бронирование успешно оформлено.

### 3.2 Отмена бронирования

Если пользователь по каким-либо причинам хочет отменить бронь, он может воспользоваться соответствующей функцией:

1. В нижнем разделе страницы «Бронирование» находится секция для отмены бронирования.
2. Ввести имя и номер телефона, которые были указаны при бронировании.
3. Система проверяет наличие брони по введённым данным.
4. Если бронь найдена, пользователю будет предложено подтвердить удаление.

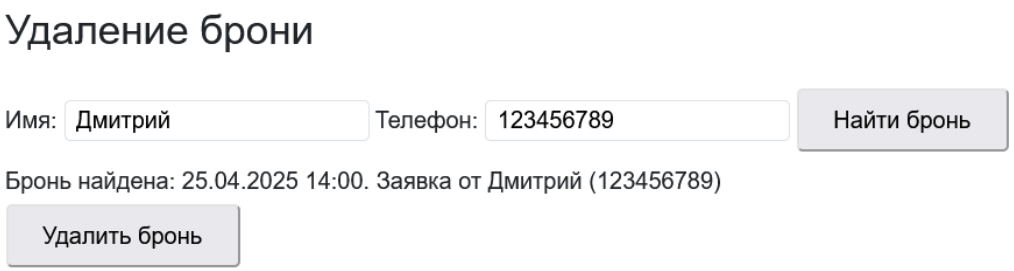


Рисунок 3.5 - «Функция отмены бронирования»

5. После подтверждения система удаляет запись из базы данных.

### 3.3 Примечания по использованию

- Время бронирования не может пересекаться с уже существующими бронями для одного и того же столика.
- Все данные пользователя используются исключительно для обработки бронирования и не сохраняются вне базы данных.
- Если не найдено подходящих столиков, система предложит изменить параметры запроса.

### 3.4 Таблица занятости

На этой странице сайта отображаются все столы, зарегистрированные в системе, включая их вместимости, номера и активные бронирования. Отсюда системный администратор сайта способен вручную отменить любую из броней в случае необходимости.

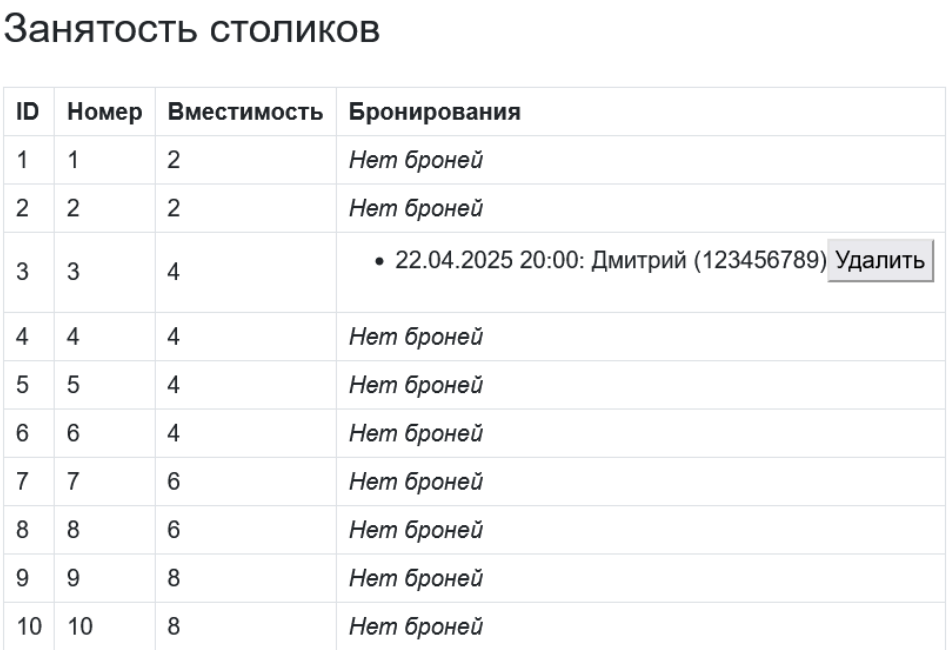


Рисунок 3.6 – «Таблица занятости столов»

## Заключение

В ходе разработки курсового проекта я проанализировал теоретическую составляющую предметной области, рассмотрел цели, преимущества, этапы создания автоматизированной информационной системы и ее структуру.

Были проанализированы существующие системы бронирования столов, их применение и задачи, выполняемые ими, а также причины применения информационных технологий для их достижения. Для создания информационной системы «Бронирование столиков» были рассмотрены и описаны технологии, использующиеся в их разработке.

Для правильного использования системы было написано руководство пользователя с описанием всех доступных функций системы.

## Список использованных источников


1. Автоматизированные системы управления (Лекция): [сайт]. – URL: https://hromatron.narod.ru/lekcii-po-asu/avtomatizirovannie-sistemi-upravleniya.htm (дата обращения: 04.03.2025) – Текст: электронный.
2. Blazor documentation [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/aspnet/core/blazor/, свободный. – Дата обращения: 15.04.2025.
3. Entity Framework Core documentation [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/ef/core/, свободный. – Дата обращения: 15.04.2025.
4. Хабр: Разработка веб-приложений на Blazor [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/microsoft/blog/516758/, свободный. – Дата обращения: 16.04.2025.
5. Использование чат-бота в качестве информационной системы ресторанного бизнеса // КиберЛенинка [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/ispolzovanie-chat-bota-v-kachestve-informatsionnoy-sistemy-restorannogo-biznesa (дата обращения: 16.04.2025).
6. Бронирование столов в ресторане: частые проблемы и решения – Saby [Электронный ресурс]. – Режим доступа: https://saby.ru/articles/presto/onlajn_bronirovanie_v_restorane (дата обращения: 16.04.2025).
7. Разработка информационной системы ресторана / Хайруллин Б. М. – М.: БиблиоКлуб, 2020. – 108 с.

## Приложение №1 «Классы проекта»

### Класс Table.cs:

```
namespace ToDoServerApp.Components.Data
{
    public class Table
    {
        public int ID { get; set; }
        public int Table_number {  get; set; }
        public int Capacity { get; set; }
        public List<TableBusyness> Busyness { get; set; } = [];
    }
}
```

### Класс TableBusyness.cs:

```
namespace ToDoServerApp.Components.Data
{
    public class TableBusyness
    {
        public int Id { get; set; }
        public required DateTime DateTime { get; set; }
        public string? Client { get; set; }

        public required int TableId { get; set; }
        public Table Table { get; set; }
    }
}
```

## Приложение №2 «Файл контекста»

```
using Microsoft.EntityFrameworkCore;

namespace ToDoServerApp.Components.Data
{
  public class AppDbContext(DbContextOptions<AppDbContext> options) :
DbContext(options)
  {
      public DbSet<Table> Tables { get; set; }
      public DbSet<TableBusyness> TableBusynesses { get; set; }

      protected override void OnModelCreating(ModelBuilder modelBuilder)
      {
          // Настройка связи один-ко-многим: Table -> TableBusyness
          modelBuilder.Entity<Table>()
              .HasMany(t => t.Busyness)
              .WithOne(tb => tb.Table)
              .HasForeignKey(tb => tb.TableId);

          // Первичный ключ
          modelBuilder.Entity<TableBusyness>()
              .HasKey(tb => tb.Id);

          // Уникальность: один стол не может быть занят дважды в одно и то же время
          modelBuilder.Entity<TableBusyness>()
              .HasIndex(tb => new { tb.TableId, tb.DateTime })
              .IsUnique();

          // Seed data для таблицы Table
          modelBuilder.Entity<Table>().HasData(new Table { ID = 1, Table_number = 1,
Capacity = 2 },
                                               new Table { ID = 2, Table_number = 2,
Capacity = 2 },
                                               new Table { ID = 3, Table_number = 3,
Capacity = 4 },
                                               new Table { ID = 4, Table_number = 4,
Capacity = 4 },
                                               new Table { ID = 5, Table_number = 5,
Capacity = 4 },
                                               new Table { ID = 6, Table_number = 6,
Capacity = 4 },
                                               new Table { ID = 7, Table_number = 7,
Capacity = 6 },
                                               new Table { ID = 8, Table_number = 8,
Capacity = 6 },
                                               new Table { ID = 9, Table_number = 9,
Capacity = 8 },
                                               new Table { ID = 10, Table_number =
10, Capacity = 8 });

          base.OnModelCreating(modelBuilder);
      }
  }
}
```

## Приложение №3 «Файл репозитория»

```
using Microsoft.EntityFrameworkCore;

namespace ToDoServerApp.Components.Data
{
    // Репозиторий для работы со столами и их занятостью
    public class TableRepository
    {
        private readonly AppDbContext _context;

        public TableRepository(AppDbContext context)
        {
            _context = context;
        }

        // Получение информации о конкретном столе по его ID, включая список броней
        public async Task<Table?> GetTableByIdAsync(int tableId)
        {
            return await _context.Tables
                .Include(t => t.Busyness)
                .FirstOrDefaultAsync(t => t.ID == tableId);
        }

        // Получение всех записей о занятости для определённого стола
        public async Task<List<TableBusyness>> GetBusynessByTableIdAsync(int
tableId)
        {
            return await _context.TableBusynesses
                .Where(tb => tb.TableId == tableId)
                .OrderBy(tb => tb.DateTime)
                .ToListAsync();
        }

        // Получение списка всех столов (без информации о занятости)
        public async Task<List<Table>> GetAllTablesAsync()
        {
            return await _context.Tables.ToListAsync();
        }

        // Проверка, занят ли стол в заданный временной интервал
        public async Task<bool> IsTableBusyAsync(int tableId, DateTime startTime,
DateTime endTime)
        {
            return await _context.TableBusynesses
                .Where(tb => tb.TableId == tableId &&
                             tb.DateTime < endTime &&
                             tb.DateTime.AddHours(1) > startTime &&
                             tb.Client != null) // Бронь существует только если
указан клиент
                .AnyAsync();
        }

        // Создание новой брони — удаляются пересекающиеся записи и добавляется
новая
        public async Task MakeReservationAsync(int tableId, DateTime startTime,
DateTime endTime, string clientName, string clientPhone)
        {
            var combinedName = $"{clientName} ({clientPhone})";

            // Удаление существующих пересекающихся броней
            var overlapping = await _context.TableBusynesses
                .Where(tb => tb.TableId == tableId &&
```

```
                            tb.DateTime < endTime &&
                            tb.DateTime.AddHours(1) > startTime)
                .ToListAsync();

            _context.TableBusynesses.RemoveRange(overlapping);

            // Добавление новой записи
            _context.TableBusynesses.Add(new TableBusyness
            {
                TableId = tableId,
                DateTime = startTime,
                Client = combinedName
            });

            await _context.SaveChangesAsync();
        }

        // Сдвигает клиентов и даты вперёд на один день
        public async Task ShiftClientsAndDatesAsync()
        {
            var allBusynesses = await _context.TableBusynesses.ToListAsync();

            if (allBusynesses.Count == 0)
                return;

            var earliestDate = allBusynesses.Min(tb => tb.DateTime.Date);
            var latestDate = allBusynesses.Max(tb => tb.DateTime.Date);

            // Получаем уникальные временные интервалы
            var timeSlots = allBusynesses
                .Select(tb => tb.DateTime.TimeOfDay)
                .Distinct()
                .OrderBy(t => t)
                .ToList();

            // Получаем уникальные ID столов
            var tableIds = allBusynesses.Select(tb => tb.TableId).Distinct();

            // Переносим брони на следующий день
            foreach (var tableId in tableIds)
            {
                for (var date = earliestDate; date < latestDate; date =
date.AddDays(1))
                {
                    foreach (var time in timeSlots)
                    {
                        var currentDateTime = date + time;
                        var nextDateTime = date.AddDays(1) + time;

                        var current = allBusynesses
                            .FirstOrDefault(tb => tb.TableId == tableId &&
tb.DateTime == currentDateTime);
                        var next = allBusynesses
                            .FirstOrDefault(tb => tb.TableId == tableId &&
tb.DateTime == nextDateTime);

                        if (current != null && next != null)
                        {
                            next.Client = current.Client; // копируем клиента на
следующий день
                        }
                    }
                }
            }
```

```
            // Очищаем клиентов на самой ранней дате
            foreach (var tb in allBusynesses.Where(tb => tb.DateTime.Date ==
earliestDate))
            {
                tb.Client = null;
            }

            await _context.SaveChangesAsync();
        }

        // Поиск брони по имени и номеру телефона
        public async Task<TableBusyness?> FindReservationAsync(string name, string
phone)
        {
            var combined = $"{name} ({phone})";

            return await _context.TableBusynesses
                .Where(tb => tb.Client != null && tb.Client == combined)
                .OrderBy(tb => tb.DateTime)
                .FirstOrDefaultAsync();
        }

        // "Мягкое" удаление брони: очищаем клиента, но оставляем слот
        public async Task DeleteReservationAsync(int id)
        {
            var entry = await _context.TableBusynesses.FindAsync(id);
            if (entry != null)
            {
                entry.Client = null;
                await _context.SaveChangesAsync();
            }
        }

        // Получение всех столов вместе с информацией о бронированиях
        public async Task<List<Table>> GetAllTablesWithBusynessAsync()
        {
            return await _context.Tables
                .Include(t => t.Busyness)
                .ToListAsync();
        }

        // "Жёсткое" удаление брони: удаляем запись из базы
        public async Task DeleteReservationByIdAsync(int id)
        {
            var record = await _context.TableBusynesses.FindAsync(id);
            if (record != null)
            {
                _context.TableBusynesses.Remove(record);
                await _context.SaveChangesAsync();
            }
        }
    }
}
```

## Приложение №4 «Меню навигации»

```
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">Бронирование столиков</a>
    </div>
</div>

<input type="checkbox" title="Navigation menu" class="navbar-toggler" />

<div class="nav-scrollable" onclick="document.querySelector('.navbar-
toggler').click()">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="bi bi-house-door-fill-nav-menu" aria-
hidden="true"></span> Home
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink href="reservation" class="nav-link" Match="NavLinkMatch.All">
                <span class="oi oi-calendar"></span> Бронирование
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink href="/busy" class="nav-link">
                <span class="oi oi-list-rich"></span> Таблица занятости
            </NavLink>
       </div>
    </nav>
</div>
```

## Приложение №5 «Бронирование»

```
@page "/reservation"
@rendermode InteractiveServer
@using ToDoServerApp.Components.Data
@inject IJSRuntime JS
@inject TableRepository TableRepository

<!-- Заголовок страницы -->
<h3 class="text-xl font-bold mb-4">Бронирование столика</h3>
<p class="text-sm text-gray-600">Текущее время: @now.ToString("g")</p>

<!-- Блок ввода параметров бронирования -->
<div class="space-y-4 mb-6">
    <label>
        Кол-во человек:
        <input type="number" @bind="peopleCount" min="1" class="border rounded px-2"
/>
    </label>
    <label>
        Дата и время:
        <input type="datetime-local" @bind="reservationTime" class="border rounded
px-2" />
    </label>
    <button @onclick="CheckAvailability" class="bg-blue-500 text-black px-4 py-2
rounded">
        Проверить доступность
    </button>
</div>

@* Проверка доступности столов и отображение результата *@
@if (availabilityChecked)
{
    @if (selectedTable is not null)
    {
                <div class="text-green-600 mb-2">Стол №@selectedTable.Table_number
доступен</div>

                <label>
                    Ваше имя:
                    <input type="text" @bind="clientName" class="border rounded px-
2" />
                </label>
                <label>
                    Телефон:
                    <input type="text" @bind="clientPhone" class="border rounded px-
2" />
                </label>

                <button @onclick="ConfirmReservation" class="bg-green-500 text-black
px-4 py-2 rounded mt-2">
                    Подтвердить бронь
                </button>
    }
    else
    {
                <div class="text-red-600">Нет свободных столов для @peopleCount
человек на @reservationTime</div>
    }
}

@if (!string.IsNullOrEmpty(successMessage))
{
```

```
        <div class="mt-4 text-green-600 font-semibold">@successMessage</div>
}

<hr class="my-8" />

<!-- Блок удаления брони -->
<h3 class="text-xl font-bold mb-4">Удаление брони</h3>
<div class="space-y-2">
    <label>
        Имя:
        <input type="text" @bind="deleteClientName" class="border rounded px-2" />
    </label>
    <label>
        Телефон:
        <input type="text" @bind="deleteClientPhone" class="border rounded px-2" />
    </label>
    <button @onclick="FindReservationToDelete" class="bg-red-400 text-black px-4 py-
2 rounded">
        Найти бронь
    </button>

    @if (reservationToDelete != null)
    {
            <div class="text-yellow-700 mt-2">
                Бронь найдена: @reservationToDelete.DateTime.ToString("g"). Заявка
от @reservationToDelete.Client<br />
                <button @onclick="DeleteReservation" class="bg-red-600 text-red px-4
py-2 rounded mt-1">Удалить бронь</button>
            </div>
    }
    else if (deleteChecked)
    {
            <div class="text-gray-600 mt-2">Бронь не найдена.</div>
    }
</div>

<h4 class="mt-6 font-semibold">Тестовое изменение времени (для отладки)</h4>
<label>
    Новое время:
    <input type="datetime-local" @bind="now" class="border rounded px-2" />
</label>

@code {
    // Данные бронирования
    private int peopleCount;
    private DateTime reservationTime = DateTime.Now;
    private string? clientName;
    private string? clientPhone;
    private Table? selectedTable;
    private bool availabilityChecked = false;
    private string? successMessage;

    // Данные для удаления
    private string? deleteClientName;
    private string? deleteClientPhone;
    private TableBusyness? reservationToDelete;
    private bool deleteChecked = false;

    // Текущее время
    private DateTime now = DateTime.Now;
    private System.Timers.Timer? timer;

    // Инициализация таймера для обновления текущего времени
    protected override void OnInitialized()
    {
```

```
        timer = new System.Timers.Timer(1000); // 1 секунда
        timer.Elapsed += (s, e) =>
        {
            now = DateTime.Now;
            InvokeAsync(StateHasChanged);
        };
        timer.Start();
    }

    // Валидация имени и телефона
    private bool IsNameValid => !string.IsNullOrWhiteSpace(clientName) &&
                                clientName.All(c => char.IsLetter(c) ||
char.IsWhiteSpace(c));
    private bool IsPhoneValid => !string.IsNullOrWhiteSpace(clientPhone) &&
                             clientPhone.All(c => char.IsDigit(c) || "+-()
".Contains(c));

    public void Dispose()
    {
        timer?.Stop();
        timer?.Dispose();
    }

    // Проверка доступности столов
    private async Task CheckAvailability()
    {
        if (peopleCount < 1)
        {
            await JS.InvokeVoidAsync("alert", "Укажите хотя бы одного человека!");
            return;
        }

        availabilityChecked = true;
        selectedTable = null;

        var tables = await TableRepository.GetAllTablesAsync();
        var reservationEndTime = reservationTime.AddHours(1); // Стол считается
занятым на 1 час

        foreach (var table in tables.OrderBy(t => t.Capacity))
        {
            if (table.Capacity >= peopleCount)
            {
                var isBusy = await TableRepository.IsTableBusyAsync(table.ID,
reservationTime, reservationEndTime);
                if (!isBusy)
                {
                    selectedTable = table;
                    break;
                }
            }
        }
    }

    // Подтверждение бронирования
    private async Task ConfirmReservation()
    {
        if (selectedTable == null)
        {
            await JS.InvokeVoidAsync("alert", "Выберите доступный столик.");
            return;
        }

        if (!IsNameValid)
        {
```

```
            await JS.InvokeVoidAsync("alert", "Поле <Имя> должно содержать только
буквы.");
            return;
        }

        if (!IsPhoneValid)
        {
            await JS.InvokeVoidAsync("alert", "Поле <Телефон> должно содержать
только цифры и символы (+, -, (, )).");
            return;
        }

        var reservationEnd = reservationTime.AddHours(1); // Бронь длится час
        await TableRepository.MakeReservationAsync(selectedTable.ID,
reservationTime, reservationEnd, clientName, clientPhone);

        // Очистка после бронирования
        peopleCount = 0;
        clientName = string.Empty;
        clientPhone = string.Empty;
        availabilityChecked = false;
        selectedTable = null;

        await JS.InvokeVoidAsync("alert", "Бронь подтверждена!");
    }

    // Поиск и удаление брони
    private async Task FindReservationToDelete()
    {
        deleteChecked = true;
        reservationToDelete = await
TableRepository.FindReservationAsync(deleteClientName!, deleteClientPhone!);
    }

    private async Task DeleteReservation()
    {
        if (reservationToDelete != null)
        {
            await
TableRepository.DeleteReservationByIdAsync(reservationToDelete.Id);
            reservationToDelete = null;
            successMessage = "Бронь удалена.";
        }
    }
}
```

## Приложение №6 «Таблица занятости»

```
@page "/busy"
@rendermode InteractiveServer
@using ToDoServerApp.Components.Data
@inject TableRepository TableRepository

<!-- Заголовок страницы -->
<h3 class="text-xl font-bold mb-4">Занятость столиков</h3>

@* Отображаем "Загрузка...", пока данные не получены *@
@if (tableData == null)
{
        <p>Загрузка...</p>
}
else
{
        <!-- Таблица занятости столов -->
        <table class="table-auto border w-full">
            <thead class="bg-gray-100">
                <tr>
                    <th class="px-2 py-1 border">ID</th>
                    <th class="px-2 py-1 border">Номер</th>
                    <th class="px-2 py-1 border">Вместимость</th>
                    <th class="px-2 py-1 border">Бронирования</th>
                </tr>
            </thead>
            <tbody>
            @foreach (var table in tableData)
            {
                    <tr>
                        <!-- ID столика -->
                        <td class="border px-2 py-1">@table.ID</td>
                        <!-- Номер столика -->
                        <td class="border px-2 py-1">@table.Table_number</td>
                        <!-- Вместимость -->
                        <td class="border px-2 py-1">@table.Capacity</td>

                        <!-- Список бронирований -->
                        <td class="border px-2 py-1">
                        @if (table.Busyness.Count == 0)
                        {
                                    <i>Нет броней</i>
                        }
                        else
                        {
                                    <ul>
                                @foreach (var b in table.Busyness.OrderBy(b =>
b.DateTime))
                                {
                                            <li>
                                                <!-- Отображение времени и
клиента -->
                                                @b.DateTime.ToString("g"):
                                                @(string.IsNullOrEmpty(b.Client) ?
"Свободно" : b.Client)

                                                @* Если есть клиент – показываем кнопку
удаления *@
                                                @if (!string.IsNullOrEmpty(b.Client))
                                                {
                                                    <button @onclick="() =>
DeleteBusyness(b.Id)"
```

```
                                                                    class="text-red-
500 ml-2 underline">
                                                                        Удалить
                                                                    </button>
                                                }
                                                        </li>
                                            }
                                                </ul>
                                        }
                                        </td>
                                    </tr>
                            }
                            </tbody>
                        </table>
}

@code {
    // Список столов с информацией о занятости
    private List<Table>? tableData;

    // Получаем данные при загрузке компонента
    protected override async Task OnInitializedAsync()
    {
        tableData = await TableRepository.GetAllTablesWithBusynessAsync();
    }

    // Удаление конкретной брони по ID и обновление данных
    private async Task DeleteBusyness(int id)
    {
        await TableRepository.DeleteReservationByIdAsync(id);
        tableData = await TableRepository.GetAllTablesWithBusynessAsync();
    }
}
```