

Министерство образования Республики Беларусь
Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин

Дисциплина: Базы данных

ОТЧЕТ
по лабораторной работе №2
на тему

«Разработка серверной части прикладной программы»
Кинотеатр

Студент:

Р.Е. Власов

Преподаватель:

А.И. Крюков

МИНСК 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ.....	2
2 ВЫПОЛНЕНИЕ РАБОТЫ	2
2.1 Выбор языка программирования и дополнительных компонентов	2
2.2 Взаимодействие с базой данных.....	3
2.3 Основные части пользовательского интерфейса.....	4
2.4 Листинг кода	5
3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ	6
3.1 Развертывание приложения	6
3.2 Работа с приложением.....	7
4 ВЫВОД.....	7
ПРИЛОЖЕНИЕ А.....	8

1 ЦЕЛЬ РАБОТЫ

Цели данной лабораторной работы:

- разработка спецификаций серверной части (backend) программы;
- программирование серверной части с использованием прикладного интерфейса СУБД PostgreSQL;
- программирование клиентской части программы.

2 ВЫПОЛНЕНИЕ РАБОТЫ

2.1 Выбор языка программирования и дополнительных компонентов

В качестве языка программирования для реализации серверной и клиентской частей программы будет использоваться C#, а конкретнее:

- само приложение будет работать на платформе пользовательского интерфейса для создания разнообразных кроссплатформенных клиентских приложений рабочего стола с использованием .Net Framework 9.0;
- для взаимодействия с базой данных будет использован пакет NuGet Npgsql версии 9.0.0.

2.2 Взаимодействие с базой данных

Для подключения к базе данных в проекте «Кинотеатр» используется Entity Framework Core с использованием PostgreSQL в качестве базы данных. Подключение настраивается через строку подключения в формате: Host=(адрес расположения базы данных); Username=(имя пользователя базы данных); Password=(пароль пользователя); Database=(название базы данных).

В классе ApplicationDbContext реализуется контекст базы данных, где задаются все сущности, отражающие таблицы базы данных: клиенты, фильмы, залы, сеансы, билеты, места, сотрудники, отзывы, а также промежуточные таблицы для связи многие-ко-многим. Метод OnModelCreating настраивает составные ключи для промежуточных сущностей, таких как SessionEmployee и SessionMovie.

Для работы с данными реализуются контроллеры API, например, контроллер TicketsController, который обеспечивает CRUD-операции для сущности Ticket. Данные можно:

- Получать (GET) все билеты или один конкретный по ID;^{[1][2][3][4][5][6][7][8][9][10]}
- Добавлять новый билет через POST;
- Обновлять существующий билет через PUT;
- Удалять билет через DELETE.

При создании или изменении данных данные сохраняются в базе данных через метод SaveChangesAsync. Для асинхронного выполнения запросов используются методыToListAsync, FirstOrDefaultAsync и другие, чтобы обеспечить производительность и отзывчивость API.

Строка подключения задается в файле appsettings.json, что позволяет легко изменять параметры базы данных.

При развертывании приложения на сервере или локальной машине обеспечивается автоматическое создание и миграция базы данных при помощи Entity Framework Core, что упрощает управление схемой базы данных.

На клиентской стороне данные визуализируются в удобной форме, например, на странице TicketsPage, где отображается список билетов с такими параметрами, как ID, цена, время покупки, категория, ID сеанса, ID места и ID клиента. Пользователь может редактировать или добавлять данные через отдельную форму (TicketFormPage), где также реализованы валидация и удобный выбор клиента, места и сеанса.

Любые изменения синхронизируются с базой данных через метод SaveChangesAsync в контроллерах.

Такой подход обеспечивает гибкость и масштабируемость при разработке приложения для кинотеатра.

2.3 Основные части пользовательского интерфейса

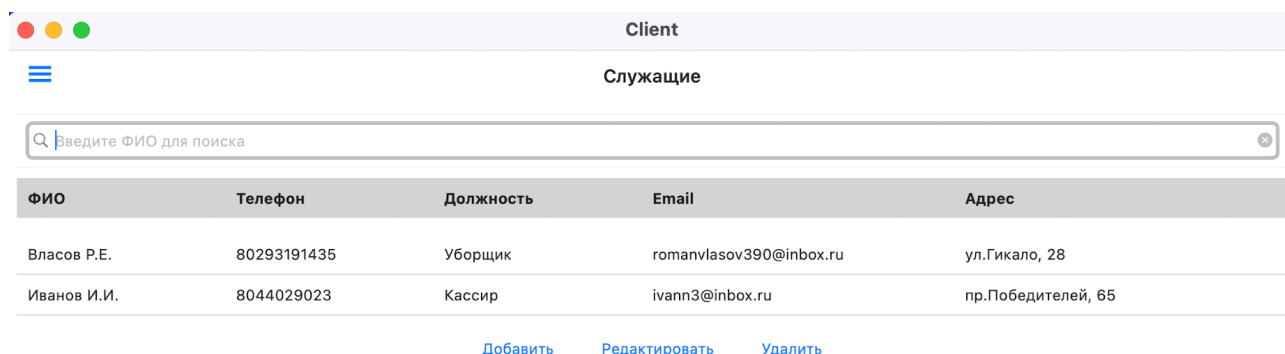
Пользовательский интерфейс приложения представлен в виде главного окна с бургер-меню (Рисунок 2.3.2), предоставляющего доступ ко всем таблицам базы данных кинотеатра. Основные функции интерфейса:

1. Бургер-меню слева в верхней части экрана: позволяет выбрать нужную таблицу (например, клиенты, сотрудники, фильмы, залы, билеты и т.д.).

2. Поисковая строка в верхней части окна: позволяет фильтровать данные в таблице, вводя ключевые слова или параметры.

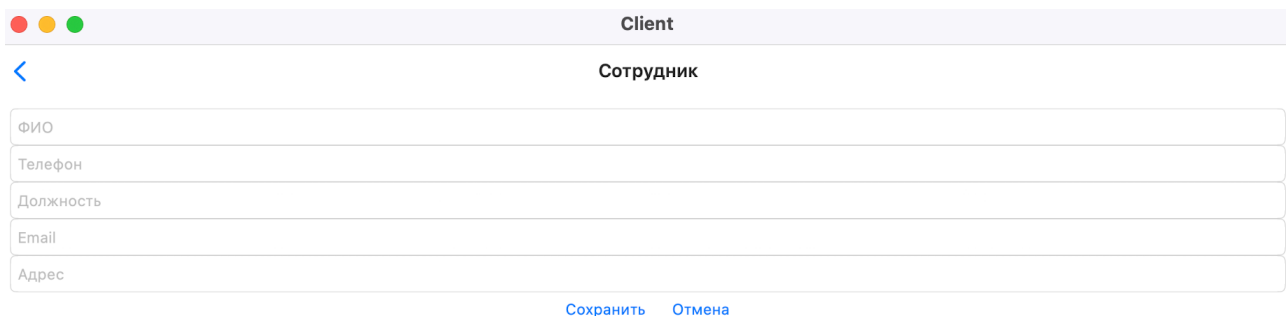
3. Основная рабочая область: отображает выбранную таблицу с данными в виде списка, организованного по колонкам. Для каждой записи в таблице можно увидеть все основные поля, такие как ФИО, телефон, должность, email и адрес для сотрудников.

4. Кнопки управления под таблицей: Добавить: открывает новое окно для ввода информации о новой записи. Редактировать: открывает отдельное окно для внесения изменений в выбранную запись. Удалить: позволяет удалить выбранную запись из базы данных.



Client				
Служащие				
<input type="text" value="Введите ФИО для поиска"/>				
ФИО	Телефон	Должность	Email	Адрес
Власов Р.Е.	80293191435	Уборщик	romanvlasov390@inbox.ru	ул.Гикало, 28
Иванов И.И.	8044029023	Кассир	ivann3@inbox.ru	пр.Победителей, 65
Добавить Редактировать Удалить				

Рисунок 2.3.1 – Пример пользовательского интерфейса для таблицы “Служащие”



Client

Сотрудник

ФИО

Телефон

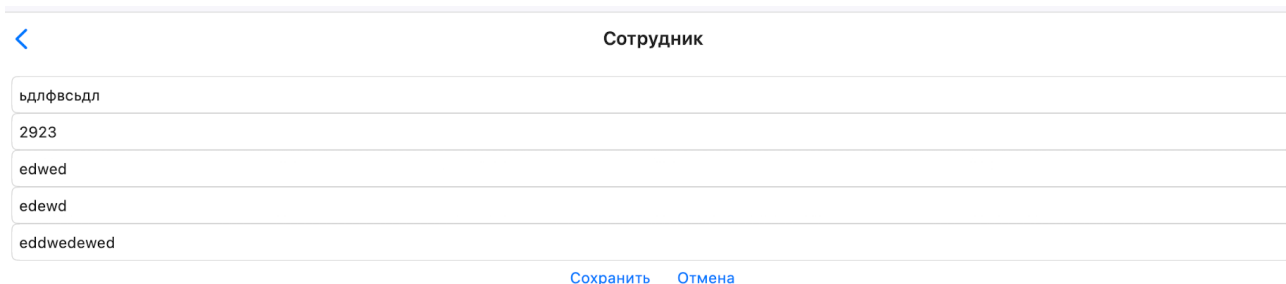
Должность

Email

Адрес

Сохранить Отмена

Рисунок 2.3.2 – Пример пользовательского интерфейса для добавления объекта “Служащие”



Сотрудник

ьдлфвсьдл

2923

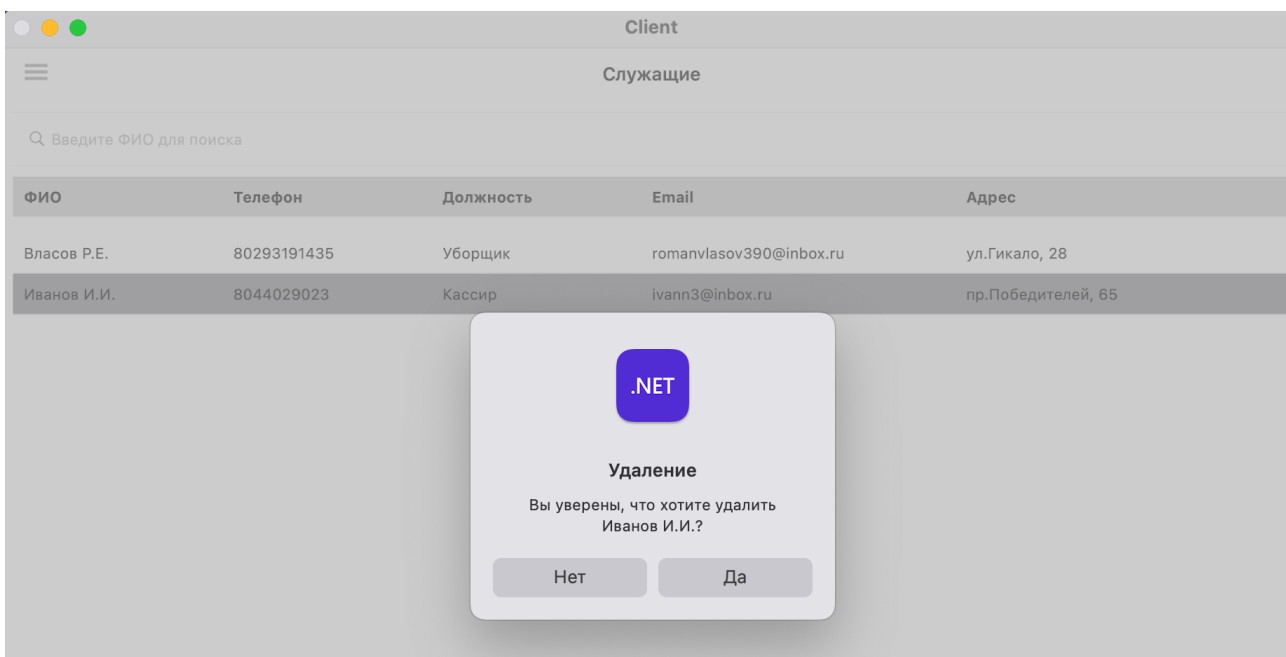
edwed

edewd

eddwedewd

Сохранить Отмена

Рисунок 2.3.3 – Пример пользовательского интерфейса для редактирования объекта “Служащие”



Client

Служащие

Введите ФИО для поиска

ФИО	Телефон	Должность	Email	Адрес
Власов Р.Е.	80293191435	Уборщик	romanvlasov390@inbox.ru	ул.Гикало, 28
Иванов И.И.	8044029023	Кассир	ivann3@inbox.ru	пр.Победителей, 65

.NET

Удаление

Вы уверены, что хотите удалить
Иванов И.И.?

Нет Да

Рисунок 2.3.4 – Пример пользовательского интерфейса для удаления объекта “Служащие”

2.4 Листинг кода

Листинг кода программы представлен в приложении А.

3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

3.1 Развертывание приложения

1. Установка и настройка:

1.1 Убедитесь, что у вас установлен JetBrains Rider. Если нет, скачайте и установите его с официального сайта JetBrains.

1.2 Убедитесь, что на вашем Mac установлены следующие зависимости:

- .NET SDK
- PostgreSQL

2. Создание нового проекта:

2.1 Запустите Rider.

2.2 В стартовом окне выберите “New Solution”.

2.3 Выберите шаблон “ASP.NET Core Web Application”.

2.4 Укажите имя проекта (например, “CinemaWebService”) и папку для сохранения. Нажмите “Create”.

2.5 В появившемся окне выберите шаблон “Web API” и убедитесь, что выбран .NET 9.0. Нажмите “Create”.

3. Добавление библиотеки Npgsql:

3.1 Откройте файл *.csproj вашего проекта.

3.2 Добавьте в секцию <ItemGroup> следующую строку для установки Npgsql:

```
<PackageReference Include="Npgsql.EntityFrameworkCore.PostgreSQL"
Version="9.*" />
```

3.3 Сохраните файл и выполните команду dotnet restore в терминале для загрузки всех зависимостей.

4. Запуск приложения:

4.1 В меню Rider нажмите "Run" или выберите конфигурацию запуска в правом верхнем углу и нажмите кнопку запуска (зеленая стрелка).

4.2 Приложение запустится локально, и вы сможете получить доступ к API через браузер по адресу <http://localhost:5242>.

При необходимости отладки используйте встроенные инструменты Rider для точек останова и анализа логов.

3.2 Работа с приложением

1. Для выбора таблицы из базы данных нажмите на иконку бургер-меню в верхнем левом углу экрана. В меню выберите нужную таблицу (например, “Клиенты”, “Сотрудники”, “Билеты” и т.д.). После выбора данные из выбранной таблицы появятся в основной рабочей области.

2. Для редактирования существующих записей нажмите кнопку “Редактировать” под таблицей. Откроется новое окно, где вы сможете внести изменения в выбранную запись.

3. Для удаления записей выберите нужную строку в таблице, а затем

нажмите кнопку “Удалить”. Приложение запросит подтверждение перед удалением данных.

4. Для добавления новой записи нажмите кнопку “Добавить” под таблицей. Откроется отдельное окно для ввода данных новой записи.

4 ВЫВОД

Цели данной лабораторной работы:

- разработка спецификаций серверной части (backend) программы;
- программирование серверной части с использованием прикладного интерфейса СУБД PostgreSQL;
- программирование клиентской части программы.

ПРИЛОЖЕНИЕ А
(обязательное)

Листинг кода

Файл ApplicationDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using CinemaWebService.Models;
using Microsoft.EntityFrameworkCore.Design;

namespace CinemaWebService.Data
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
            : base(options)
        {}

        public DbSet<Client> Clients { get; set; }
        public DbSet<Movie> Movies { get; set; }
        public DbSet<Hall> Halls { get; set; }
        public DbSet<Session> Sessions { get; set; }
        public DbSet<Ticket> Tickets { get; set; }
        public DbSet<Review> Reviews { get; set; }
        public DbSet<Employee> Employees { get; set; }
        public DbSet<Seat> Seats { get; set; }

        public DbSet<SessionEmployee> SessionEmployees { get; set; }
        public DbSet<SessionMovie> SessionMovies { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<SessionEmployee>()
                .HasKey(se => new { se.SessionId, se.EmployeeId });

            modelBuilder.Entity<SessionMovie>()
                .HasKey(sm => new { sm.SessionId, sm.MovieId });
        }
    }
}

namespace CinemaWebService.Data
{
    public class ApplicationDbContextFactory :
IDesignTimeDbContextFactory<ApplicationDbContext>
    {
        public ApplicationDbContext CreateDbContext(string[] args)
        {
            var configuration = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json")
                .Build();

            var optionsBuilder = new
DbContextOptionsBuilder<ApplicationDbContext>();
            var connectionString =
configuration.GetConnectionString("DefaultConnection");

            optionsBuilder.UseNpgsql(connectionString);

            return new ApplicationDbContext(optionsBuilder.Options);
        }
    }
}
```

Файл ClientsController.cs

```
using CinemaWebService.Data;
using CinemaWebService.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace CinemaWebService.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ClientsController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        public ClientsController(ApplicationDbContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Client>>> GetAllClients()
        {
            return await _context.Clients.ToListAsync();
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<Client>> GetClientById(int id)
        {
            var client = await _context.Clients.FindAsync(id);

            if (client == null)
            {
                return NotFound();
            }

            return client;
        }

        [HttpPost]
        public async Task<ActionResult<Client>> CreateClient(Client client)
        {
            _context.Clients.Add(client);
            await _context.SaveChangesAsync();

            // Возвращаем код 201 + путь к новому ресурсу
            return CreatedAtAction(nameof(GetClientById), new { id =
client.ClientId }, client);
        }

        [HttpPut("{id}")]
        public async Task<IActionResult> UpdateClient(int id, Client
updatedClient)
        {
            if (id != updatedClient.ClientId)
            {
                return BadRequest("Идентификаторы не совпадают");
            }

            _context.Entry(updatedClient).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
            }
        }
    }
}
```

```

        if (!ClientExists(id))
        {
            return NotFound();
        }
        throw;
    }

    return NoContent();
}

[HttpDelete("{id}")]
public async Task<IActionResult> DeleteClient(int id)
{
    var client = await _context.Clients.FindAsync(id);
    if (client == null)
    {
        return NotFound();
    }

    _context.Clients.Remove(client);
    await _context.SaveChangesAsync();

    return NoContent();
}

private bool ClientExists(int id)
{
    return _context.Clients.Any(e => e.ClientId == id);
}
}
}

```

Файл AppShell.xaml.cs

```

using CinemaClientService.Pages;
using Microsoft.Maui.Controls;

namespace CinemaClientService
{
    public partial class AppShell : Shell
    {
        public AppShell()
        {
            InitializeComponent();

            Routing.RegisterRoute(nameof(MainPage), typeof(MainPage));
            Routing.RegisterRoute(nameof(HallsPage), typeof(HallsPage));
            Routing.RegisterRoute(nameof(EmployeeFormPage),
typeof(EmployeeFormPage));
            Routing.RegisterRoute(nameof(HallFormPage), typeof(HallFormPage));
            Routing.RegisterRoute(nameof(ClientsPage), typeof(ClientsPage));

            this.Items.Add(new FlyoutItem
            {
                Title = "Сотрудники",
                Icon = "employees_icon.png",
                Route = nameof(MainPage),
                Items =
                {
                    new ShellContent
                    {
                        Title = "Список сотрудников",
                        ContentTemplate = new DataTemplate(typeof(MainPage))
                    }
                }
            });
        }
    }
}

```

```

this.Items.Add(new FlyoutItem
{
    Title = "Фильмы",
    Icon = "movies_icon.png",
    Route = nameof(MoviesPage),
    Items =
    {
        new ShellContent
        {
            Title = "Список фильмов",
            ContentTemplate = new DataTemplate(typeof(MoviesPage))
        }
    }
});

// Добавление FlyoutItem для залов
this.Items.Add(new FlyoutItem
{
    Title = "Залы",
    Icon = "halls_icon.png",
    Route = nameof(HallsPage),
    Items =
    {
        new ShellContent
        {
            Title = "Список залов",
            ContentTemplate = new DataTemplate(typeof(HallsPage))
        }
    }
});

this.Items.Add(new FlyoutItem
{
    Title = "Отзывы",
    Icon = "reviews_icon.png",
    Route = nameof(ReviewsPage),
    Items =
    {
        new ShellContent
        {
            Title = "Список отзывов",
            ContentTemplate = new DataTemplate(typeof(ReviewsPage))
        }
    }
});

this.Items.Add(new FlyoutItem
{
    Title = "Клиенты",
    Icon = "clients_icon.png",
    Route = nameof(ClientsPage),
    Items =
    {
        new ShellContent
        {
            Title = "Список клиентов",
            ContentTemplate = new DataTemplate(typeof(ClientsPage))
        }
    }
});

this.Items.Add(new FlyoutItem
{
    Title = "Места",
    Icon = "reviews_icon.png",
    Route = nameof(SeatsPage),
    Items =
    {

```

```

        new ShellContent
        {
            Title = "Список мест",
            ContentTemplate = new DataTemplate(typeof(SeatsPage))
        }
    });

this.Items.Add(new FlyoutItem
{
    Title = "Сеансы",
    Icon = "sessions_icon.png",
    Route = nameof(SessionsPage),
    Items =
    {
        new ShellContent
        {
            Title = "Список сеансов",
            ContentTemplate = new DataTemplate(typeof(SessionsPage))
        }
    }
});

this.Items.Add(new FlyoutItem
{
    Title = "Билеты",
    Icon = "sessions_icon.png",
    Route = nameof(TicketsPage),
    Items =
    {
        new ShellContent
        {
            Title = "Список билетов",
            ContentTemplate = new DataTemplate(typeof(TicketsPage))
        }
    }
});

this.Items.Add(new FlyoutItem
{
    Title = "Сеанс_сотрудник",
    Icon = "sessions_icon.png",
    Route = nameof(SessionEmployeesPage),
    Items =
    {
        new ShellContent
        {
            Title = "Сеанс_сотрудник",
            ContentTemplate = new
DataTemplate(typeof(SessionEmployeesPage))
        }
    }
});

this.Items.Add(new FlyoutItem
{
    Title = "Сеанс_фильм",
    Icon = "sessions_icon.png",
    Route = nameof(SessionMoviesPage),
    Items =
    {
        new ShellContent
        {
            Title = "Сеанс_фильм",
            ContentTemplate = new
DataTemplate(typeof(SessionMoviesPage))
        }
    }
});

```

```

        });
    }
}

```

Файл ClientsPage.xaml.cs

```

using System;
using System.Collections.ObjectModel;
using System.Net.Http.Json;
using Microsoft.Maui.Controls;
using CinemaClientService.ViewModels;

namespace CinemaClientService.Pages
{
    public partial class ClientsPage : ContentPage
    {
        private readonly HttpClient _httpClient;
        public ObservableCollection<ClientView> Clients { get; set; }

        public ClientsPage(HttpClient httpClient)
        {
            InitializeComponent();
            _httpClient = httpClient;
            Clients = new ObservableCollection<ClientView>();
            ClientsCollection.ItemsSource = Clients; // ClientsCollection из
XAML
        }

        protected override async void OnAppearing()
        {
            base.OnAppearing();
            await LoadClients();
        }

        private async Task LoadClients()
        {
            try
            {
                var list = await
_httpClient.GetFromJsonAsync<List<ClientView>>("api/Clients");
                Clients.Clear();
                if (list != null)
                {
                    foreach (var c in list)
                        Clients.Add(c);
                }
            }
            catch (Exception ex)
            {
                await DisplayAlert("Ошибка", ex.Message, "OK");
            }
        }

        private async void OnAddClicked(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new ClientFormPage(_httpClient, null));
        }

        private async void OnEditClicked(object sender, EventArgs e)
        {
            if (ClientsCollection.SelectedItem is ClientView selected)
            {
                await Navigation.PushAsync(new ClientFormPage(_httpClient,
selected));
            }
        }
    }
}

```

```

        else
        {
            await DisplayAlert("Ошибка", "Выберите клиента для
редактирования", "OK");
        }
    }

    private async void OnDeleteClicked(object sender, EventArgs e)
    {
        if (ClientsCollection.SelectedItem is ClientView selected)
        {
            bool confirm = await DisplayAlert("Удаление",
                $"Удалить клиента {selected.FullName}?", "Да", "Нет");
            if (!confirm) return;

            try
            {
                var response = await
_httpClient.DeleteAsync($"api/Clients/{selected.ClientId}");
                if (!response.IsSuccessStatusCode)
                {
                    var msg = await response.Content.ReadAsStringAsync();
                    await DisplayAlert("Ошибка", msg, "OK");
                    return;
                }
                Clients.Remove(selected);
            }
            catch (Exception ex)
            {
                await DisplayAlert("Ошибка", ex.Message, "OK");
            }
        }
    }

    private void OnSearchTextChanged(object sender, TextChangedEventArgs e)
    {
        string query = e.NewTextValue?.ToLower() ?? string.Empty;
        var filtered = query == string.Empty
            ? Clients
            : new ObservableCollection<ClientView>(Clients.Where(c =>
c.FullName.ToLower().Contains(query)));
        ClientsCollection.ItemsSource = filtered;
    }
}

```

Файл ClientFormPage.xaml.cs

```

using System;
using System.Net.Http.Json;
using Microsoft.Maui.Controls;
using CinemaClientService.ViewModels;

namespace CinemaClientService.Pages
{
    public partial class ClientFormPage : ContentPage
    {
        private readonly HttpClient _httpClient;
        private ClientView _client;

        public ClientFormPage(HttpClient httpClient, ClientView client)
        {
            InitializeComponent();
            _httpClient = httpClient;
            _client = client;
        }
    }
}

```



```

        if (_client != null)
        {
            FullNameEntry.Text = _client.FullName;
            EmailEntry.Text = _client.Email;
            OrderHistoryEditor.Text = _client.OrderHistory;
        }
    }

    private async void OnSaveClicked(object sender, EventArgs e)
    {
        try
        {
            var newClient = new ClientView
            {
                ClientId = _client?.ClientId ?? 0,
                FullName = FullNameEntry.Text,
                Email = EmailEntry.Text,
                OrderHistory = OrderHistoryEditor.Text
            };

            HttpResponseMessage response;
            if (_client == null)
            {
                response = await _httpClient.PostAsJsonAsync("api/Clients",
newClient);
            }
            else
            {
                response = await
_httpClient.PutAsJsonAsync($"api/Clients/{newClient.ClientId}", newClient);
            }

            if (!response.IsSuccessStatusCode)
            {
                var msg = await response.Content.ReadAsStringAsync();
                await DisplayAlert("Ошибка", msg, "OK");
                return;
            }

            await DisplayAlert("Успешно", "Данные сохранены", "OK");
            await Navigation.PopAsync();
        }
        catch (Exception ex)
        {
            await DisplayAlert("Ошибка", ex.Message, "OK");
        }
    }

    private async void OnCancelClicked(object sender, EventArgs e)
    {
        await Navigation.PopAsync();
    }
}

```