ЛАБОРАТОРНАЯ РАБОТА №5.
КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ

СОДЕРЖАНИЕ

Цель

Изучить работу клиент-серверных приложений.

Задание

1. Создать ветку от ветки четвертой лабораторной.
2. *Добавить клиента в приложение:*
    a. *Добавить сущность «Клиент». По клиенту необходимо хранить информацию: ФИО, логин (электронная почта) и пароль.*
    б. *В заказе фиксировать какой клиент сделал заказ.*
    в. *Создать приложение с RestAPI, которое будет предоставлять функционал для web-приложения для регистрации клиентов, создания ими заказов и просмотра ранее созданных заказов.*
    г. *Создать web-приложение для клиентов, чтобы они могли сами создавать заказы.*
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Проектирование

В первую очередь надо определится как новое приложение для клиента будет встроено в наш проект. Приложение будет разбито на 2 части: клиентскую и серверную. Клиентская часть будет из себя представлять web-приложение, которое будет слать запросы на сервер, реализованный в виде RestAPI-приложения. В свою очередь RestAPI-приложение будет использовать туже логику, что и имеющееся у нас desktop-приложение (рисунок 5.1). Таким образом, все наши приложения будут использовать единую логику, а также единую систему хранения данных.

Рисунок 5.1 – Обновленная архитектура проекта

Немного теории. Клиент-сервер – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между серверами и клиентами. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, работа с базами данных).

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

Каждая единица информации однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных. Т.е., например, третья книга с книжной полки будет иметь вид /book/3, а 35 страница в этой книге – /book/3/page/35. Отсюда и получается строго заданный формат.

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. Так вот, для HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия CRUD (Create-Read-Update-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

Вот как это будет выглядеть на примере:

- GET /book/ — получить список всех книг
- GET /book/3/ — получить книгу номер 3
- PUT /book/ — добавить книгу (данные в теле запроса)
- POST /book/3 – изменить книгу (данные в теле запроса)
- DELETE /book/3 – удалить книгу

Перейдем к сущности «Клиент». Как было сказано, у клиента будет 3 поля: ФИО (выводится в заказе), логин (электронная почта) и пароль. Первым делом в слое моделей надо будет объявить новый интерфейс модели-клиента. Важное ограничение: не должно быть 2-х клиентов с одинаковым логином.

Далее в слое контрактов объявить binding и view-модели для сущности «Клиент», интерфейс бизнес-логики и интерфейс работы с данными.

В слое бизнес-логики создать класс-реализацию IClientLogic, а в слое хранения данных сделать 3 модели для хранения данных и 3 реализации IClientStorage.

Также потребуется доработать сущность «Заказ». В модель добавить поле с идентификатором клиента, дополнить модели «Заказа» в слое контрактов, дополнить и доработать логику в моделях «Заказ» в слое хранения

данных, а также добавить в метод GetFilteredList новое условие выборки заказов по клиенту.

Перейдем к desktop-приложению. Тут потребуется добавить новую форму для отображения списка клиентов. При этом делать форму для создания и редактирования не потребуется, клиенты сами будут это делать в web-приложении. Останется только просмотр списка и удаление клиентов.

Потребуется доработать форму создания заказа, так как теперь у заказа появляется новое поле, клиент, делающий заказ, а также доработать логику вывода списка заказов на форму, чтобы отображалось ФИО клиента, создавшего заказ.

Следующий шаг – создание RestAPI-приложения. Это приложение позволит выполнять следующие действия:

Регистрация пользователя (клиента)

- Аутентификация пользователя
- Изменение данных пользователя
- Получение списка заказов пользователя
- Создание заказа

Для создания заказа потребуется также получать как минимум список изделий и конкретное изделие, чтобы вытаскивать его цену (либо хранить в web-приложении полученный список и оттуда брать цену).

Логично будет разбить описанный функционал на 2 части: отдельно работа по пользователю (регистрация, аутентификация, изменение данных) и отдельно работа по заказам, включая получение списка изделий.

И последнее – создание web-приложения. Оно будет включать в себя тот же набор функций, что и RestAPI-приложение, однако будет иметь интерфейс пользователя, чтобы он мог использовать этот набор.

Рассмотрим детальнее сценарии поведения пользователя, как он будет взаимодействовать с функциями приложения, какие формы будут вызываться и в какой последовательности:

1. Для не авторизованного пользователя предоставляется страница аутентификации с переходом на страницу регистрации по необходимости.
2. Для регистрации требуется заполнить все данные, после чего идет возврат на страницу аутентификации.
3. При верной авторизации следует переход на страницу со списком всех заказов клиента. На странице предусмотрены возможности перехода на страницу редактирования пользовательских данных и на страницу создания заказа.
4. На странице редактирования пользовательских данных пользователь может изменить свои данные, после чего идет возврат на страницу со списком заказов.
5. На странице создания заказа пользователь выбирает изделие из списка, указывает количество (сумма рассчитывается автоматически при наличии выбранного изделия и количества) и создает заказ, после чего идет возврат на страницу со списком заказов.

Перейдем к реализации всего выше описанного.

Реализация

Создадим новый интерфейс модели-клиента (листинг 5.1). Само собой, у клиента также будет идентификатор, так что интерфейс наследуем от IId.

```
namespace AbstractShopDataModels.Models
{
    public interface IClientModel : IId
    {
        string ClientFIO { get; }

        string Email { get; }

        string Password { get; }
    }
}
```

Листинг 5.1 – Интерфейс IClientModel

Далее создаем binding и view-модели для сущности «Клиент» (реализации интерфейса IClientModel, листинг 5.2, 5.3) интерфейс бизнес-

логики (листинг 5.4) и интерфейс работы с данными (листинг 5.5). Они полностью идентичны сущности «Компонент» (можно взять за основу).

```
using AbstractShopDataModels.Models;

namespace AbstractShopContracts.BindingModels
{
    public class ClientBindingModel : IClientModel
    {
        public int Id { get; set; }

        public string ClientFIO { get; set; } = string.Empty;

        public string Email { get; set; } = string.Empty;

        public string Password { get; set; } = string.Empty;
    }
}
```

Листинг 5.2 – Класс ClientBindingModel

```
using AbstractShopDataModels.Models;
using System.ComponentModel;

namespace AbstractShopContracts.ViewModels
{
    public class ClientViewModel : IClientModel
    {
        public int Id { get; set; }

        [DisplayName("ФИО клиента")]
        public string ClientFIO { get; set; } = string.Empty;

        [DisplayName("Логин (эл. почта)")]
        public string Email { get; set; } = string.Empty;

        [DisplayName("Пароль")]
        public string Password { get; set; } = string.Empty;
    }
}
```

Листинг 5.3 – Класс ClientViewModel

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.BusinessLogicsContracts
{
    public interface IClientLogic
    {
        List<ClientViewModel>? ReadList(ClientSearchModel? model);

        ClientViewModel? ReadElement(ClientSearchModel model);

        bool Create(ClientBindingModel model);

        bool Update(ClientBindingModel model);

        bool Delete(ClientBindingModel model);
    }
}
```

Листинг 5.4 – Интерфейс IClientLogic

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.StoragesContracts
{
    public interface IClientStorage
    {
        List<ClientViewModel> GetFullList();

        List<ClientViewModel> GetFilteredList(ClientSearchModel model);

        ClientViewModel? GetElement(ClientSearchModel model);

        ClientViewModel? Insert(ClientBindingModel model);

        ClientViewModel? Update(ClientBindingModel model);

        ClientViewModel? Delete(ClientBindingModel model);
    }
}
```

Листинг 5.5 – Интерфейс IClientStorage

Реализация IClientLogic довольно проста, можно взять за основу, например, реализацию IComponentLogic (разработать самостоятельно).

Тоже самое для моделей и реализаций сущности «Клиент» в проектах хранения данных. Разработать самостоятельно.

Аналогично для добавления нового поля в сущность «Заказ». Самый сложный шаг – обновление базы данных, так как тут могут возникнуть сложности. В таблицу «Заказы» потребуется вставить новое поле. Если в таблице нет записей, то все пройдет нормально. Но, если записи есть, то возникнет конфликт, так как в каждую запись потребуется вставить новое поле, связанное с другой таблицей («Клиенты»), у которой нет записей еще. Тут существует несколько вариантов решения:

- Прописать значение по умолчанию и прописать код вставки клиента по умолчанию после создания таблицы «Клиенты» и до создания поля в таблице «Заказы», чтобы при установке значения по умолчанию не нарушалась связанность между таблицами.
- В классе «Заказ» указать, что поле идентификатора клиента может принимать значение NULL. В таком случае, в существующих записях заказов выставится значение «NULL».
- Удалить все записи из таблицы «Заказы».

Так как у нас проект находится на стадии разработки, то проще будет отчистить таблицу. После этого, можно обновлять схему БД.

Не забудем еще добавить новый фильтр в методы GetFilteredList для выборки заказов конкретного клиента.

Также для desktop-приложения самостоятельно сделать:

- Добавить форму вывода списка клиентов. Иметь возможность выводить полный список клиентов, а также удалять выбранных клиентов (функций редактирования и создания не требуется делать).
- В форму создания заказа добавить возможность выбора клиента. Добавить выпадающий список, подгружать в него список всех клиентов и запоминать выбранного при создании заказа.
- Выводить ФИО клиента в заказах на главной форме. Расширить логику вывода списка заказов новой колонкой с ФИО клиента, при это идентификатор клиента не отображать!

Перейдем к проекту RESTAPI. Добавим новый проект **AbstractShopRestApi** (рисунки 5.2, 5.3).

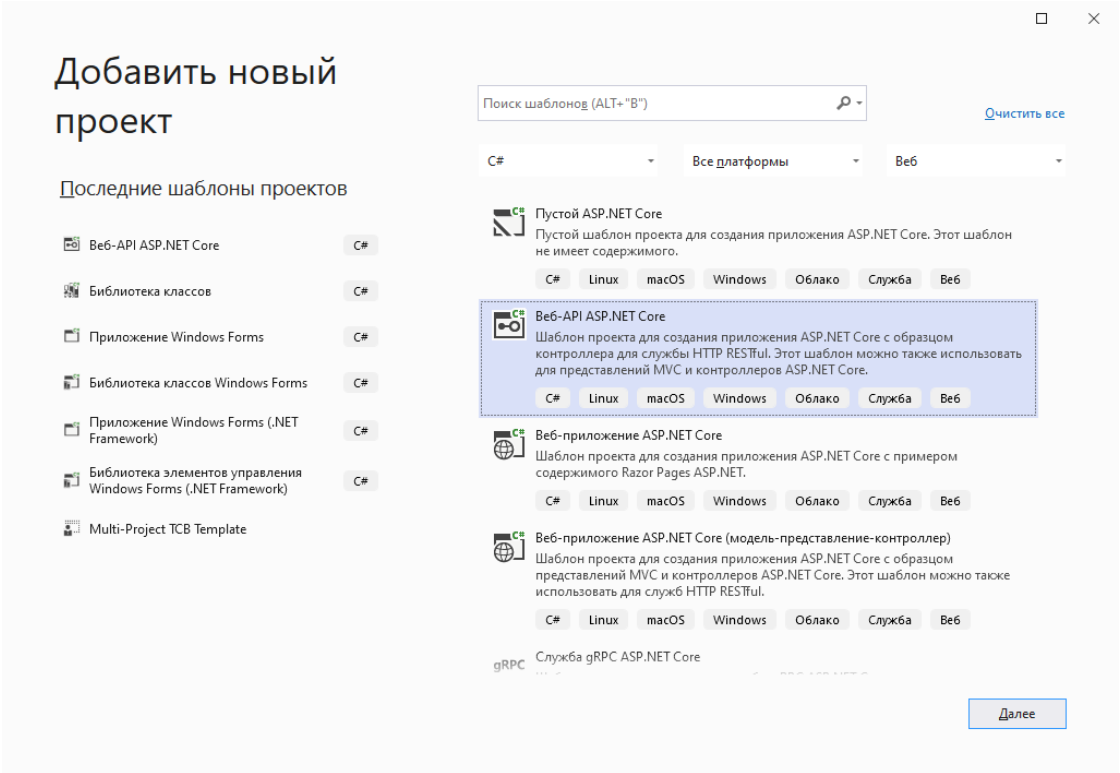


Рисунок 5.2 – Выбор типа проекта

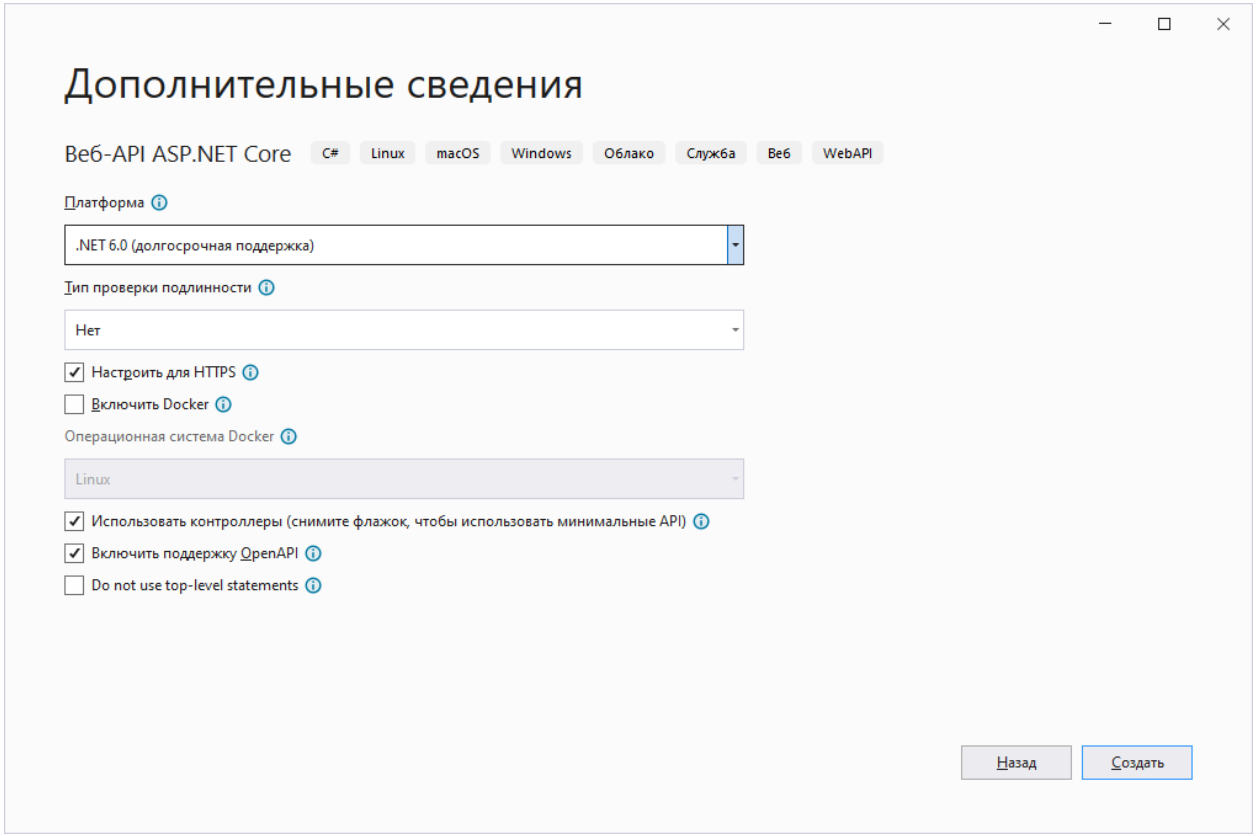


Рисунок 5.3 – Настройка проекта AbstractShopRestApi

Добавим ссылки на проекты:

- **AbstractShopBusinessLogic** (автоматически подтянется проект **AbstractShopContracts**)
- **AbstractShopDatabaseImplement**

В проекте уже есть встроенный IoC-контейнер, так что не потребуется ничего дополнительно подключать. В классе Program пропишем связи, логгер, swagger (листинг 5.6).

```
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.StoragesContracts;
using AbstractShopDatabaseImplement.Implements;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Logging.SetMinimumLevel(LogLevel.Trace);
builder.Logging.AddLog4Net("log4net.config");

// Add services to the container.


builder.Services.AddTransient<IClientStorage, ClientStorage>();
builder.Services.AddTransient<IOrderStorage, OrderStorage>();
builder.Services.AddTransient<IProductStorage, ProductStorage>();

builder.Services.AddTransient<IOrderLogic, OrderLogic>();
```

```
builder.Services.AddTransient<IClientLogic, ClientLogic>();
builder.Services.AddTransient<IProductLogic, ProductLogic>();

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
      c.SwaggerDoc("v1", new OpenApiInfo { Title = "AbstractShopRestApi", Version
= "v1" });
});

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
"AbstractShopRestApi v1"));
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Листинг 5.6 – Класс Program

Сделаем 2 контроллера: один для работы с клиентами (листинг 5.7), а второй для работы с заказами (листинг 5.8).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;
using Microsoft.AspNetCore.Mvc;

namespace AbstractShopRestApi.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class ClientController : Controller
    {
        private readonly ILogger _logger;

        private readonly IClientLogic _logic;

        public ClientController(IClientLogic logic, ILogger<ClientController>
logger)
        {
            _logger = logger;
            _logic = logic;
        }

        [HttpGet]
        public ClientViewModel? Login(string login, string password)
        {
            try
```

```
            {
                return _logic.ReadElement(new ClientSearchModel
                {
                    Email = login,
                    Password = password
                });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка входа в систему");
                throw;
            }
        }

        [HttpPost]
        public void Register(ClientBindingModel model)
        {
            try
            {
                _logic.Create(model);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка регистрации");
                throw;
            }
        }

        [HttpPost]
        public void UpdateData(ClientBindingModel model)
        {
            try
            {
                _logic.Update(model);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка обновления данных");
                throw;
            }
        }
    }
}
```

Листинг 5.7 – Контроллер ClientController

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;
using Microsoft.AspNetCore.Mvc;

namespace AbstractShopRestApi.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class MainController : Controller
    {
        private readonly ILogger _logger;

        private readonly IOrderLogic _order;

        private readonly IProductLogic _product;
```

```
        public MainController(ILogger<MainController> logger, IOrderLogic order,
IProductLogic product)
        {
            _logger = logger;
            _order = order;
            _product = product;
        }

        [HttpGet]
        public List<ProductViewModel>? GetProductList()
        {
            try
            {
                return _product.ReadList(null);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка получения списка продуктов");
                throw;
            }
        }

        [HttpGet]
        public ProductViewModel? GetProduct(int productId)
        {
            try
            {
                return _product.ReadElement(new ProductSearchModel { Id =
productId });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка получения продукта по id={Id}",
productId);
                throw;
            }
        }

        [HttpGet]
        public List<OrderViewModel>? GetOrders(int clientId)
        {
            try
            {
                return _order.ReadList(new OrderSearchModel { ClientId = clientId
});
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка получения списка заказов клиента
id={Id}", clientId);
                throw;
            }
        }

        [HttpPost]
        public void CreateOrder(OrderBindingModel model)
        {
            try
            {
                _order.CreateOrder(model);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка создания заказа");
```

```
                throw;
            }
        }
    }
}
```

Листинг 5.8 – Контроллер MainController

Логика в методах контроллеров будет простой: получить данные, если требуется и вызывать нужный метод из нужного интерфейса. Для клиента нам потребуется 3 действия: регистрация (добавление) новых клиентов, вход в систему и изменение данных клиента. Для заказов потребуются: получение списка заказов клиента, получение списка изделий (для выбора в заказ), получение изделия (для расчета стоимости) и создание заказа.

Протестируем разработанный сервис. Для этого запустим его. Это можно сделать 2 способами:

- назначить автозапускаемым;
- правой кнопкой щелкнуть по названию проекта, найти пункт «Отладка» и «Запустить новый экземпляр».

Выберем 2 вариант. В проекте по умолчанию добавлен swagger. По сути Swagger – это фреймворк для спецификации RESTful API. Он позволяет просматривать спецификацию (все контроллеры и их публичные методы), но и отправлять запросы (так называемый Swagger UI) к методам контроллеров для проверки их корректной работы. Если все сделано верно, то при обращении к методу GetProductList (данный метод выбран потому что не требует на вход никаких параметров, т.е. самый простой) контроллера MainController получим список изделий из БД (рисунки 5.4, 5.5).

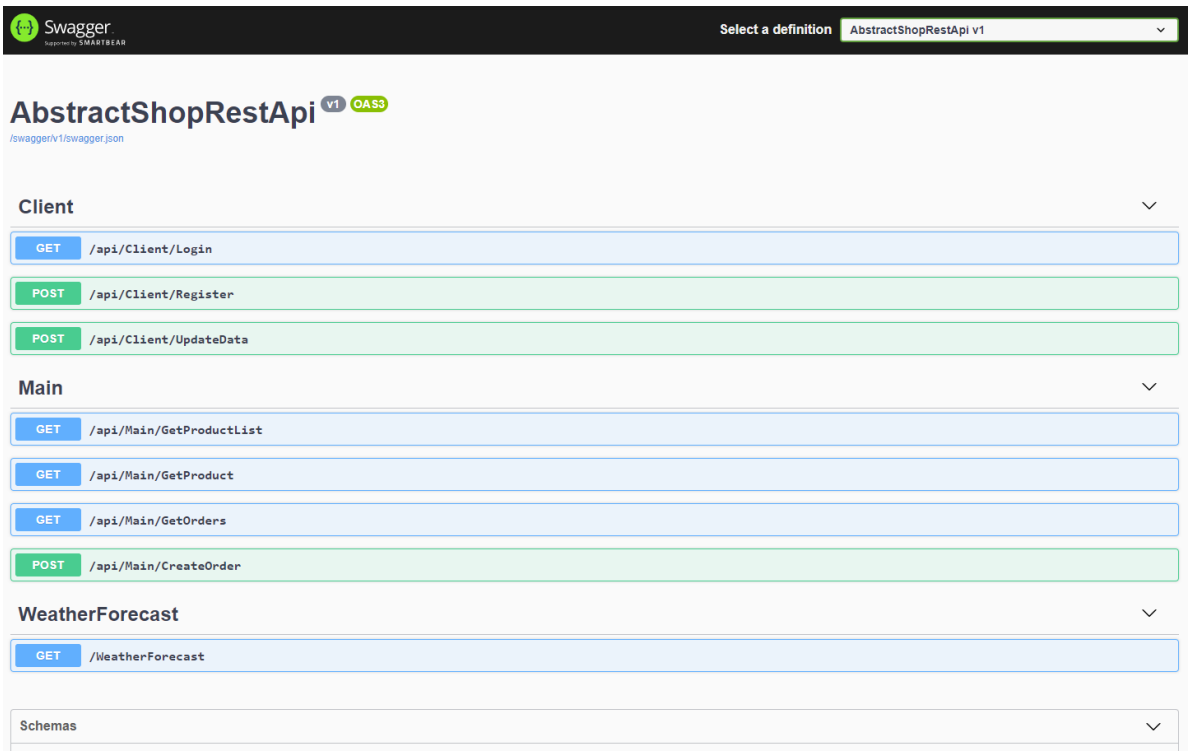


Рисунок 5.4 – Swagger приложения AbstractShopRestApi

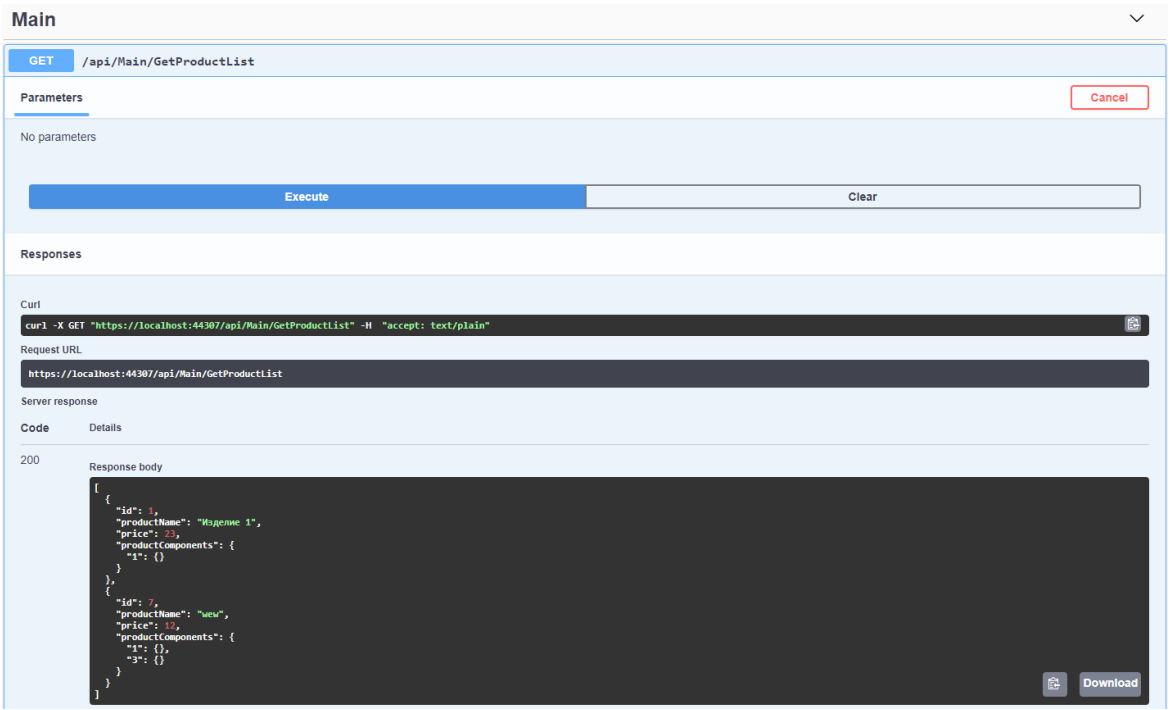


Рисунок 5.5 – Результат запроса для получения списка изделий

Последний шаг – создание приложения для клиентов. Создадим проект для клиентов (рисунок 5.6).

Создадим класс для подключения к серверу. Назовем его APIClient. Нам потребуется 3 метода: подключение к серверу, отправка get-запроса и отправка post-запроса (листинг 5.9).

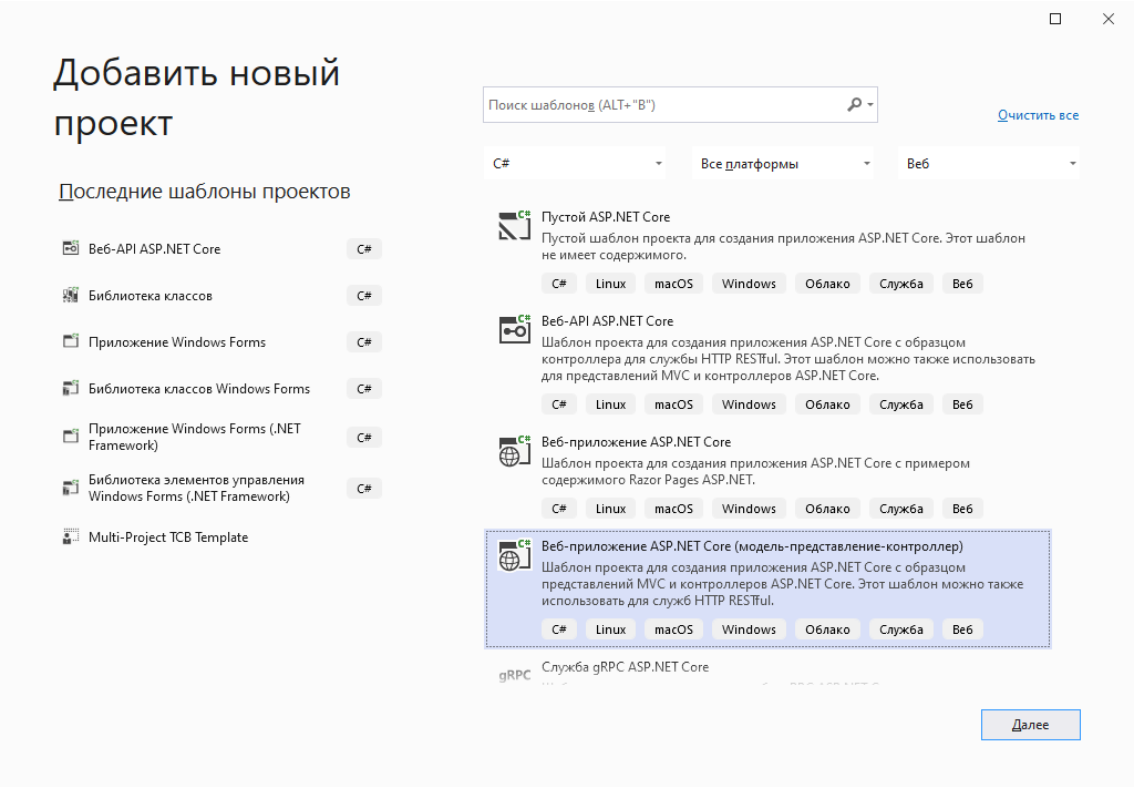


Рисунок 5.6 – Создание проекта AbstractShowClientApp

```
using AbstractShopContracts.ViewModels;
using Newtonsoft.Json;
using System.Net.Http.Headers;
using System.Text;

namespace AbstractShowClientApp
{
    public static class APIClient
    {
        private static readonly HttpClient _client = new();

        public static ClientViewModel? Client { get; set; } = null;

        public static void Connect(IConfiguration configuration)
        {
            _client.BaseAddress = new Uri(configuration["IPAddress"]);
            _client.DefaultRequestHeaders.Accept.Clear();
            _client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
        }

        public static T? GetRequest<T>(string requestUrl)
        {
            var response = _client.GetAsync(requestUrl);
            var result = response.Result.Content.ReadAsStringAsync().Result;
            if (response.Result.IsSuccessStatusCode)
            {
                return JsonConvert.DeserializeObject<T>(result);
            }
            else
            {
                throw new Exception(result);
            }
        }

        public static void PostRequest<T>(string requestUrl, T model)
```

```
        {
            var json = JsonConvert.SerializeObject(model);
            var data = new StringContent(json, Encoding.UTF8,
"application/json");

            var response = _client.PostAsync(requestUrl, data);

            var result = response.Result.Content.ReadAsStringAsync().Result;
            if (!response.Result.IsSuccessStatusCode)
            {
                throw new Exception(result);
            }
        }
    }
}
```

Листинг 5.9 – Класс APIClient

Строку подключения пропишем в настройках проекта в файле appsettings.json (листинг 5.10).

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",

  "IPAddress": "http://localhost:5159/"
}
```

Листинг 5.10 – Файл appsettings.json

Номер порта нужно взять из настроек проекта-RestAPI (рисунок 5.7).

В проекте есть HomeController и 2 представления, Index и Privacy. Создавать еще контроллеры не будем, а вот представлений создадим (папка Views\Home). В Index будем выводить список заказов, в Privacy форму для редактирования данных клиента. Создадим представление для входа в систему (Enter.cshtml), регистрации (Register.cshtml) и создания заказа (Create.cshtml). Все формы будут частичного представления, чтобы основное меню и тело страницы было одно для всех (_Layout.cshtml).

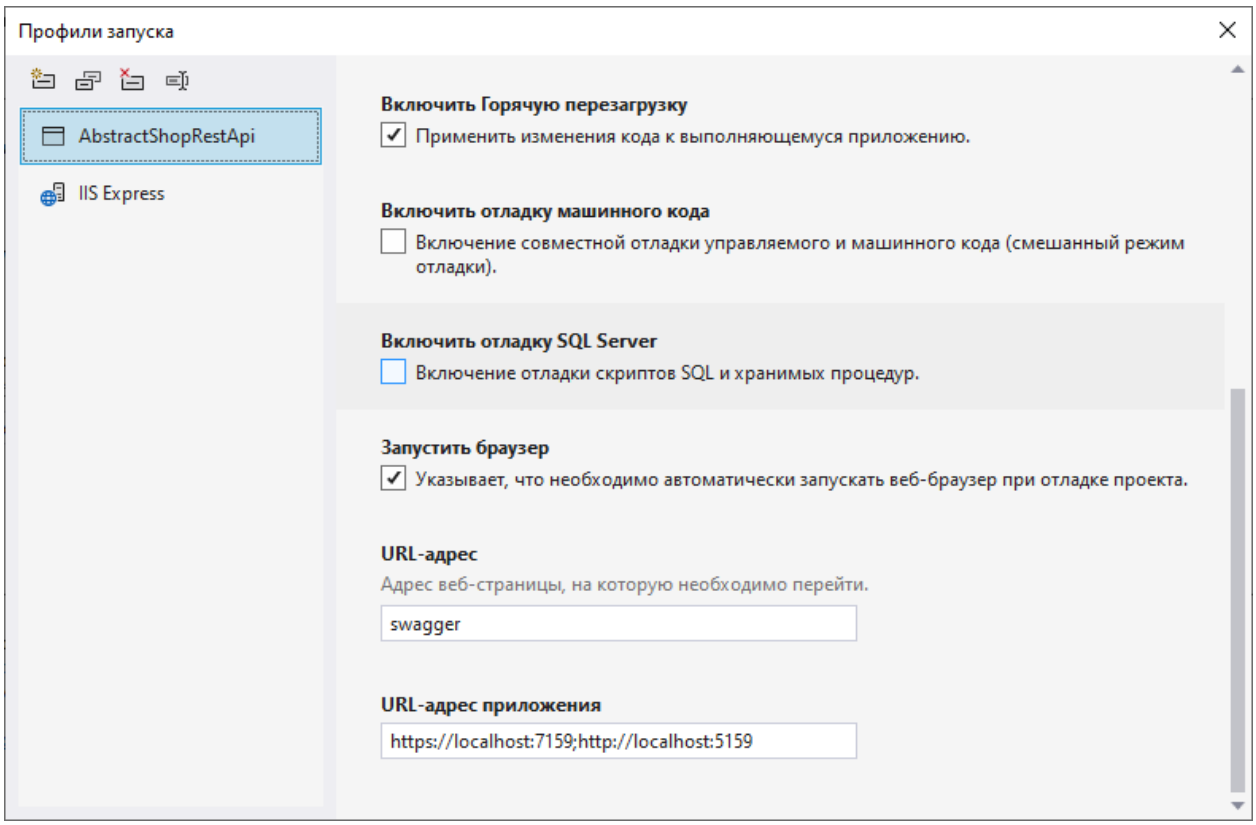


Рисунок 5.7 – Свойства проекта AbstractShopRestApi

После создания страниц перейдем в _Layout.cshtml и внесем туда несколько правок (листинг 5.11).

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="utf-8" />
      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
      <title>@ViewData["Title"] - AbstractShowClientApp</title>
      <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
      <link rel="stylesheet" href="~/css/site.css" />
      <script src="~/lib/jquery/dist/jquery.min.js"></script>
      <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-
white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">Абстрактный магазин</a>
                <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target=".navbar-collapse" aria-
controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-
row-reverse">
                                <ul class="navbar-nav flex-grow-1">
                                    <li class="nav-item">
                                        <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Index">Заказы</a>
```

```
                                        </li>
                                        <li class="nav-item">
                                            <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Privacy">Личные данные</a>
                                        </li>
                                        <li class="nav-item">
                                            <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Enter">Вход</a>
                                        </li>
                                        <li class="nav-item">
                                            <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Register">Регистрация</a>
                                        </li>
                                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <footer class="border-top footer text-muted">
            <div class="container">
                &copy; 2020 - Абстрактный магазин - <a asp-area="" asp-
controller="Home" asp-action="Privacy">Личные данные</a>
            </div>
    </footer>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @RenderSection("Scripts", required: false)
</body>
</html>
```

Листинг 5.11 – Представление _Layout.cshtml

Что сделали: добавили пункты меню, вписали названия на кириллице (это опционально) и подключение скриптов перенесли в header (потребуется при использовании javascript на одной из страниц).

Далее перейдем к представлениям и наполним их данными. Начнем с представления регистрации (листинг 5.12).

```
@{
      ViewData["Title"] = "Register";
}

<div class="text-center">
      <h2 class="display-4">Регистрация</h2>
</div>
<form method="post">
      <div class="row">
            <div class="col-4">Логин:</div>
            <div class="col-8"><input type="text" name="login" /></div>
      </div>
      <div class="row">
            <div class="col-4">Пароль:</div>
            <div class="col-8"><input type="password" name="password" /></div>
      </div>
      <div class="row">
            <div class="col-4">ФИО:</div>
```

```
            <div class="col-8"><input type="text" name="fio" /></div>
      </div>
      <div class="row">
            <div class="col-8"></div>
            <div class="col-4"><input type="submit" value="Регистрация"
class="btn btn-primary" /></div>
      </div>
</form>
```

Листинг 5.12 – Представление Register.cshtml

Тут просто форма с 3 полями (логин, пароль и ФИО) и кнопкой отправки данных на сервер.

Далее представление для входа в систему (листинг 5.13).

```
@{
      ViewData["Title"] = "Enter";
}

<div class="text-center">
      <h2 class="display-4">Вход в приложение</h2>
</div>
<form method="post">
      <div class="row">
            <div class="col-4">Логин:</div>
            <div class="col-8"><input type="text" name="login" /></div>
      </div>
      <div class="row">
            <div class="col-4">Пароль:</div>
            <div class="col-8"><input type="password" name="password" /></div>
      </div>
      <div class="row">
            <div class="col-8"></div>
            <div class="col-4"><input type="submit" value="Вход" class="btn btn-
primary" /></div>
      </div>
</form>
```

Листинг 5.13 – Представление Enter.cshtml

Представление для редактирования данных пользователя (листинг 5.14).

```
@using AbstractShopContracts.ViewModels

@model ClientViewModel

@{
      ViewData["Title"] = "Privacy Policy";
}
<div class="text-center">
      <h2 class="display-4">Личные данные</h2>
</div>
<form method="post">
      <div class="row">
            <div class="col-4">Логин:</div>
            <div class="col-8"><input type="text" name="login"
value="@Model.Email"/></div>
      </div>
      <div class="row">
            <div class="col-4">Пароль:</div>
            <div class="col-8"><input type="password" name="password"
value="@Model.Password"/></div>
      </div>
```

```
	<div class="row">
		<div class="col-4">ФИО:</div>
		<div class="col-8"><input type="text" name="fio"
value="@Model.ClientFIO"/></div>
	</div>
	<div class="row">
		<div class="col-8"></div>
		<div class="col-4"><input type="submit" value="Сохранить" class="btn
btn-primary" /></div>
	</div>
</form>
```

Листинг 5.14 – Представление Privacy.cshtml

Тут в представление будет передаваться модель и поля будут заполняться данными из модели.

Представление для создания заказа (листинг 5.15).

```
@{
	ViewData["Title"] = "Create";
}
<div class="text-center">
	<h2 class="display-4">Создание заказа</h2>
</div>
<form method="post">
	<div class="row">
		<div class="col-4">Изделие:</div>
		<div class="col-8">
			<select id="product" name="product" class="form-control" asp-
items="@(new SelectList(@ViewBag.Products,"Id", "ProductName"))"></select>
		</div>
	</div>
	<div class="row">
		<div class="col-4">Количество:</div>
		<div class="col-8"><input type="text" name="count" id="count"
/></div>
	</div>
	<div class="row">
		<div class="col-4">Сумма:</div>
		<div class="col-8"><input type="text" id="sum" name="sum" readonly
/></div>
	</div>
	<div class="row">
		<div class="col-8"></div>
		<div class="col-4"><input type="submit" value="Создать" class="btn
btn-primary" /></div>
	</div>
</form>

<script>
	$('#product').on('change', function () {
		check();
	});
	$('#count').on('change', function () {
		check();
	});

	function check() {
		var count = $('#count').val();
		var product = $('#product').val();
		if (count && product) {
			$.ajax({
				method: "POST",
				url: "/Home/Calc",
```

```
                        data: { count: count, product: product  },
                        success: function (result) {
                                $("#sum").val(result);
                        }
                });
        };
    }
</script>
```

Листинг 5.15 – Представление Create.cshtml

Здесь будет выпадающий список с изделиями, а также через скрипт посылаться запрос на получение суммы на основе выбранного изделия и введенного количества (**ВНИМАНИЕ**, метод 'change' может не срабатывать в некоторых браузерах!).

И основное представление (листинг 5.16). В представление будет передаваться список заказов и выводиться в табличном виде. Также действие на создание заказа будет не в меню, а на этой форме.

```
@using AbstractShopContracts.ViewModels

@model List<OrderViewModel>

@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Заказы</h1>
</div>


<div class="text-center">
    @{
        if (Model == null)
        {
            <h3 class="display-4">Авторизируйтесь</h3>
            return;
        }

        <p>
            <a asp-action="Create">Создать заказ</a>
        </p>
        <table class="table">
            <thead>
                <tr>
                    <th>
                        Номер
                    </th>
                    <th>
                        Изделие
                    </th>
                    <th>
                        Дата создания
                    </th>
                    <th>
                        Количество
                    </th>
```

```
                                    <th>
                                          Сумма
                                    </th>
                                    <th>
                                          Статус
                                    </th>
                              </tr>
                        </thead>
                        <tbody>
                              @foreach (var item in Model)
                              {
                                    <tr>
                                          <td>
                                                @Html.DisplayFor(modelItem =>
item.Id)
                                          </td>
                                          <td>
                                                @Html.DisplayFor(modelItem =>
item.ProductName)
                                          </td>
                                          <td>
                                                @Html.DisplayFor(modelItem =>
item.DateCreate)
                                          </td>
                                          <td>
                                                @Html.DisplayFor(modelItem =>
item.Count)
                                          </td>
                                          <td>
                                                @Html.DisplayFor(modelItem =>
item.Sum)
                                          </td>
                                          <td>
                                                @Html.DisplayFor(modelItem =>
item.Status)
                                          </td>
                                    </tr>
                              }
                        </tbody>
                  </table>
            }
</div>
```

Листинг 5.16 – Представление Index.cshtml

Вся логика будет сосредоточена в HomeController (листинг 5.17).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.ViewModels;
using AbstractShowClientApp.Models;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;

namespace AbstractShowClientApp.Controllers
{
    public class HomeController : Controller
    {
            private readonly ILogger<HomeController> _logger;

            public HomeController(ILogger<HomeController> logger)
            {
                  _logger = logger;
            }

            public IActionResult Index()
```

```
        {
            if (APIClient.Client == null)
            {
                return Redirect("~/Home/Enter");
            }
            return
View(APIClient.GetRequest<List<OrderViewModel>>($"api/main/getorders?clientId={AP
IClient.Client.Id}"));
        }

        [HttpGet]
        public IActionResult Privacy()
        {
            if (APIClient.Client == null)
            {
                return Redirect("~/Home/Enter");
            }
            return View(APIClient.Client);
        }

        [HttpPost]
        public void Privacy(string login, string password, string fio)
        {
            if (APIClient.Client == null)
            {
                throw new Exception("Вы как суда попали? Суда вход
только авторизованным");
            }
            if (string.IsNullOrEmpty(login) ||
string.IsNullOrEmpty(password) || string.IsNullOrEmpty(fio))
            {
                throw new Exception("Введите логин, пароль и ФИО");
            }
            APIClient.PostRequest("api/client/updatedata", new
ClientBindingModel
            {
                Id = APIClient.Client.Id,
                ClientFIO = fio,
                Email = login,
                Password = password
            });

            APIClient.Client.ClientFIO = fio;
            APIClient.Client.Email = login;
            APIClient.Client.Password = password;
            Response.Redirect("Index");
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId =
Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }

        [HttpGet]
        public IActionResult Enter()
        {
            return View();
        }

        [HttpPost]
        public void Enter(string login, string password)
        {
```

```
                    if (string.IsNullOrEmpty(login) ||
string.IsNullOrEmpty(password))
                    {
                        throw new Exception("Введите логин и пароль");
                    }
                    APIClient.Client =
APIClient.GetRequest<ClientViewModel>($"api/client/login?login={login}&password={
password}");
                    if (APIClient.Client == null)
                    {
                        throw new Exception("Неверный логин/пароль");
                    }
                    Response.Redirect("Index");
                }

                [HttpGet]
                public IActionResult Register()
                {
                    return View();
                }

                [HttpPost]
                public void Register(string login, string password, string fio)
                {
                    if (string.IsNullOrEmpty(login) ||
string.IsNullOrEmpty(password) || string.IsNullOrEmpty(fio))
                    {
                        throw new Exception("Введите логин, пароль и ФИО");
                    }
                    APIClient.PostRequest("api/client/register", new
ClientBindingModel
                    {
                        ClientFIO = fio,
                        Email = login,
                        Password = password
                    });
                    Response.Redirect("Enter");
                    return;
                }

                [HttpGet]
                public IActionResult Create()
                {
                    ViewBag.Products =
APIClient.GetRequest<List<ProductViewModel>>("api/main/getproductlist");
                    return View();
                }

                [HttpPost]
                public void Create(int product, int count)
                {
                    if (APIClient.Client == null)
                    {
                        throw new Exception("Вы как суда попали? Суда вход
только авторизованным");
                    }
                    if (count <= 0)
                    {
                        throw new Exception("Количество и сумма должны быть
больше 0");
                    }
                    APIClient.PostRequest("api/main/createorder", new
OrderBindingModel
                    {
                        ClientId = APIClient.Client.Id,
```

```
                        ProductId = product,
                        Count = count,
                        Sum = Calc(count, product)
                    });
                    Response.Redirect("Index");
                }

                [HttpPost]
                public double Calc(int count, int product)
                {
                    var prod =
APIClient.GetRequest<ProductViewModel>($"api/main/getproduct?productId={product}"
);
                    return count * (prod?.Price ?? 1);
                }
            }
}
```

Листинг 5.17 – Класс HomeController

Для перехода между страницами используем команды Redirect. Для получение данных – запросы к RESTAPI-серверу.

Последний штрих, в классе Program прописать вызов метода Connect класса APIClient (листинг 5.18).

```
using AbstractShowClientApp;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

APIClient.Connect(builder.Configuration);

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for
production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

Листинг 5.18 – Класс Startup

Для корректной работы требуется запускать одновременно оба проекта, и RestAPI и web-приложение клиента.

Важно! Связка web-приложение с RestAPI-приложением не обязательна! Это просто пример! Web-приложение может спокойно работать с сервисами напрямую, без RestAPI-приложения (по сути, настройка связей между интерфейсами и реализациями и логика из RestAPI-приложения должна быть перенесена в web-приложение).

Контрольный пример

Запускаем RestAPI-приложение, затем web-приложение (рисунок 5.8).

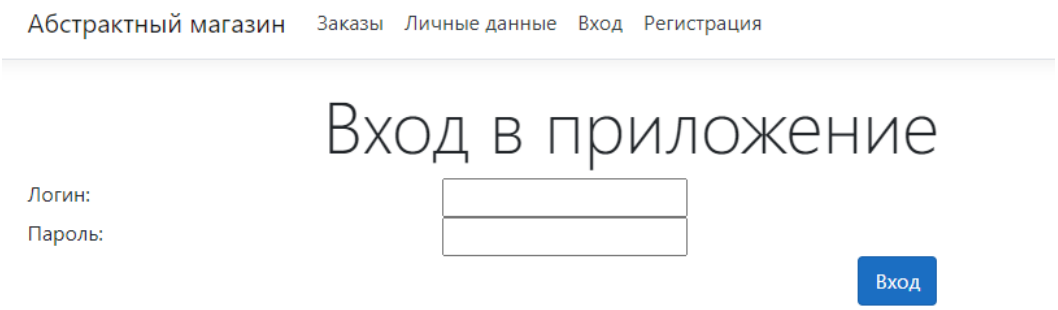


Рисунок 5.8 – Запущенное приложение

Выбираем пункт «Регистрация» и создаем нового пользователя (рисунок 5.9).

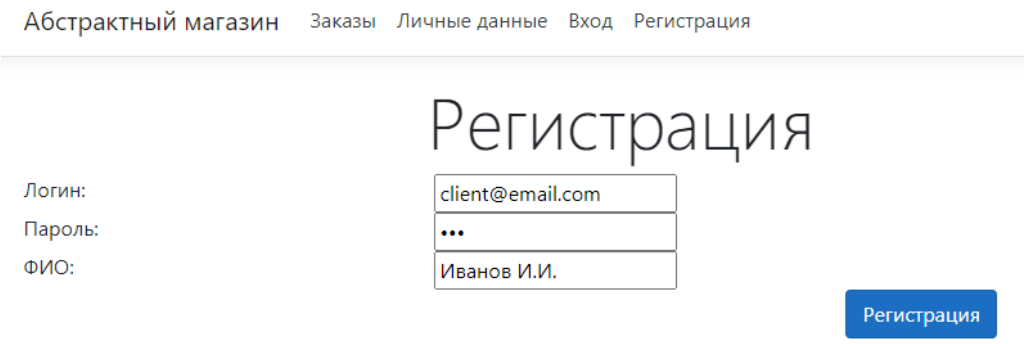


Рисунок 5.9 – Регистрация пользователя

Входим под созданным пользователем (рисунок 5.10).

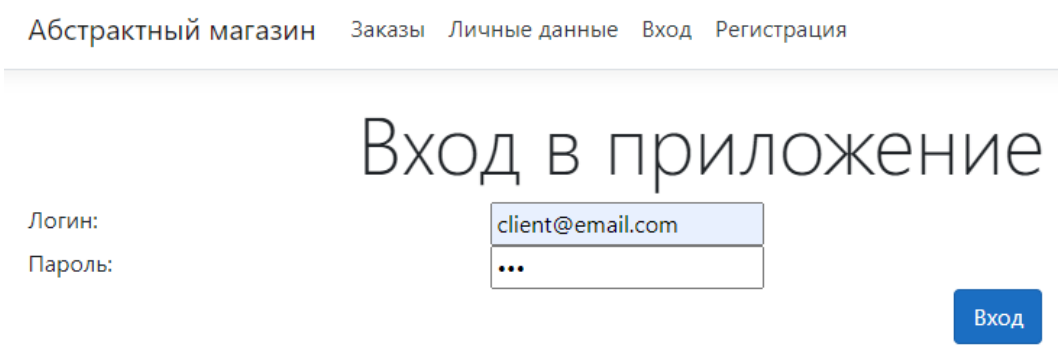


Рисунок 5.10 – Вход в систему под новым пользователем

Получаем главную страницу с пустым список заказов (рисунок 5.11).

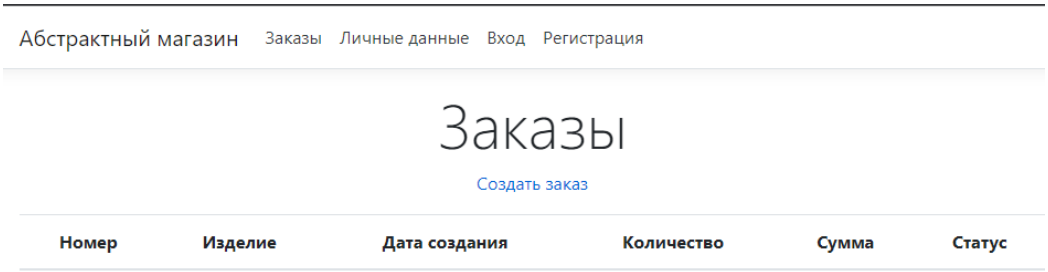
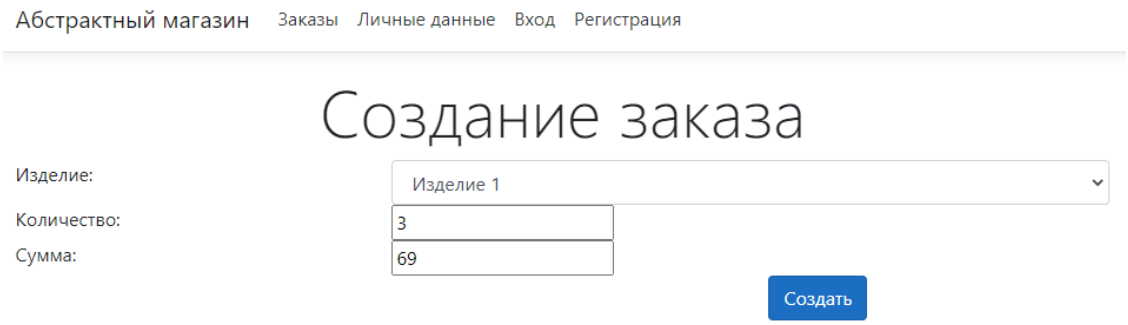


Рисунок 5.11 – Главная страница

Выберем пункт «Создать заказ» и перейдем на форму создания заказа (рисунок 5.12).



Заполним данные и нажмем кнопку «Создать». В результате будет создан заказ, мы вернемся на главную страницу и увидим созданный заказ (рисунок 5.13).

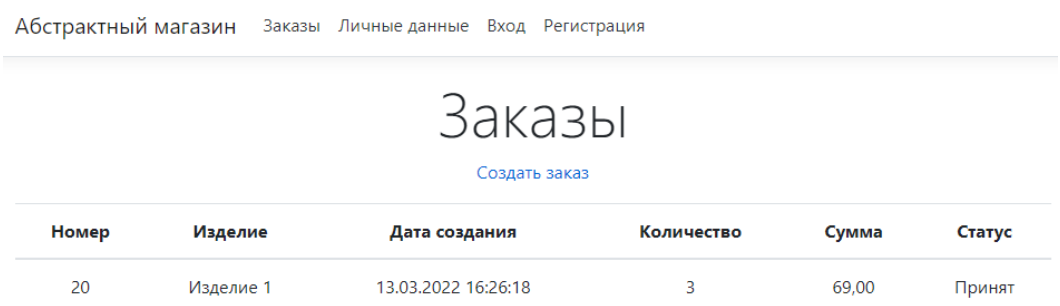


Рисунок 5.13 – Главная страница с созданным заказом

Также можем перейти на страницу личных данных и изменить их, если требуется (рисунок 5.14).

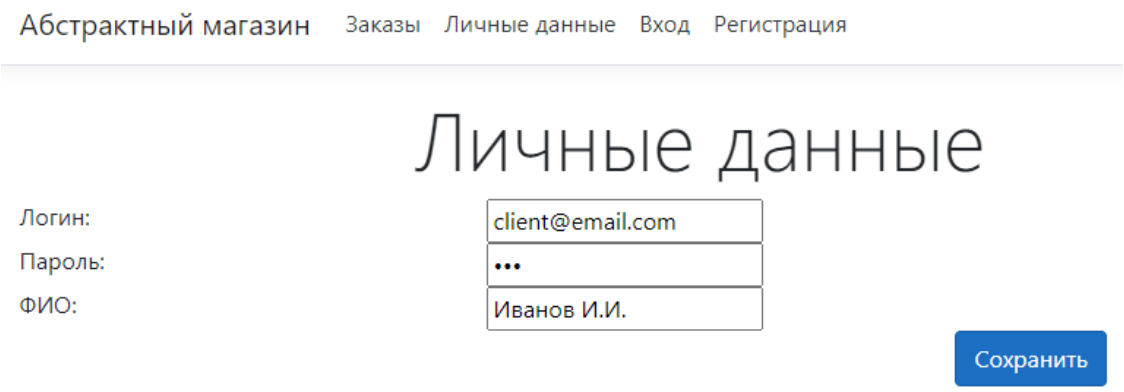


Рисунок 5.14 – Страница с личными данными клиента

Требования

1. Название проектов должны ОТЛИЧАТЬСЯ от названия проектов, приведенных в примере и должны соответствовать логике вашего задания по варианту.
2. Название форм, классов, свойств классов должно соответствовать логике вашего задания по варианту.
3. НЕ ИСПОЛЬЗОВАТЬ в названии класса, связанного с изделием слово «Product» (во вариантах в скобках указано название класса для изделия)!!!
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).

5. Сделать реализацию IClientLogic (не должно быть 2-х клиентов с одинаковым логином).
6. Сделать реализацию моделей и интерфейса IClientStorage для хранилищ.
7. Добавить в сущность «Заказ» новое поле-идентификатор клиента и в метод GetFilteredList новый вариант выборки.
8. В desktop-приложение добавить форму вывода списка клиентов.
9. В форму создания заказа добавить возможность выбора клиента.
10. Выводить ФИО клиента в заказах на главной форме.

Порядок сдачи базовой части

1. Предварительно создать 2-3 клиентов, у каждого создать по 2-3 заказа
2. Запустить 3 проекта (RestAPI, Web и Desktop)
3. В RestAPI-проекте показать получение списка изделий
4. В Web-проекте войти под одним из созданных ранее клиентом, создать заказ
5. Перейти в Desktop-проект, обновить список заказов, показать, что созданный заказ там появился
6. Ответить на вопрос преподавателя

Контрольные вопросы к базовой части

1. Как web-приложение «общается» с основной системой?
2. Как сущность «Клиент» встроена в систему?
3. Как работает RestAPI-приложение?

Усложненная лабораторная (необязательно)

1. Дополнить RestAPI-приложение контроллером для работы с магазинами.

2. В контроллере прописать методы для получения списка магазинов, создания магазина, редактирования магазина, удаление магазина и поставки изделий в магазин.
3. Создать web-приложение для работы с магазинами с использованием сервиса RestAPI.
4. Вход в приложение осуществлять через ввод пароля (пароль задать в файле конфигурации).
5. В приложении сделать представления для вывода списка магазинов (главная), для добавления, редактирования (отображать список изделий магазина), удаления и поставки изделий в магазин.

Порядок сдачи усложненной части

1. Запустить 3 проекта (RestAPI, Web и Desktop).
2. В RestAPI-проекте показать получение списка магазинов.
3. В Web-проекте создать магазин, пополнить его, показать, что изделия в магазине отображаются.
4. Перейти в Desktop-проект, показать обновленный список магазинов.
5. Ответить на вопрос преподавателя.

Контрольные вопросы к усложненной части

1. Как выполняется аутентификация?
2. Какие методы были добавлены в RestAPI-приложение?
3. Как происходит работы с компонентами в web-приложении?

Варианты

1. Кондитерская. В качестве компонентов выступают различные виды шоколада и наполнители, типа орехов, изюма и т.п. Изделие – кондитерское изделие (pastry).
2. Автомастерская. В качестве компонентов выступают различные масла, смазки и т.п. Изделия – ремонт автомобиля (repair).

3. Моторный завод. В качестве компонентов выступают различные детали для производства двигателей. Изделия – двигатели (engine).
4. Суши-бар. В качестве компонентов выступают различные продукты для суши (рыба, водоросли, соусы). Изделия – суши (sushi).
5. Продажа компьютеров. В качестве компонентов выступают различные части для компьютеров (планки памяти, жесткие диски и т.п.). Изделия – компьютеры (computer).
6. Сборка мебели. В качестве компонентов выступают различные заготовки (ножки, спинки и т.п.). Изделия – мебель (furniture).
7. Рыбный завод. В качестве компонентов выступают различные виды рыб + дополнения к ним, типа соусов и т.п. Изделия – консервы (canned).
8. Установка ПО. В качестве компонентов выступают различное ПО. Изделия – пакеты установки, например, пакет установки офисных приложений, пакет разработчика и т.п. (package).
9. Ремонтные работы в помещении. В качестве компонентов выступают различные расходные материалы (клей, обои, краска, плитка, цемент и т.п.). Изделия – ремонтные работы в различных помещениях (repair).
10. Кузнечная мастерская. В качестве компонентов выступают различные болванки (заготовки), из которых изготавливаются подковы, кочерги и т.п. Изделия – кузнечные изделия (manufacture).
11. Пиццерия. В качестве компонентов выступают различные ингредиенты для пицц (тесто, соусы, паста и т.д.). Изделия – пиццы (pizza).
12. Завод ЖБИ. В качестве компонентов выступают различные виды бетона и металлоконструкций. Изделия – железобетонные изделия (reinforced).

13. Закусочная. В качестве компонентов выступают различные продукты для закусок (колбаса, сыр, хлеб и т.п.). Изделия – различные закуски (snack).
14. Пошив платьев. В качестве компонентов выступают различные ткани, нитки и т.п. Изделия – платья (dress).
15. Типография. В качестве компонентов выступают различные типы бумаг, тонер или чернила и т.п. Изделия – печатная продукция (листовки, брошюры, книги) (printed).
16. Автомобильный завод. В качестве компонентов выступают различные части для сборки автомобилей (кузов, двигатель, стекла и т.п.). Изделия – автомобили (car).
17. Юридическая фирма. В качестве компонентов выступают различные бланки для документов. Изделия – пакеты документов, например, для страховки или завещания (document).
18. Туристическая фирма. В качестве компонентов выступают различные условия поездки (отель проживания, туры в рамках поездок). Изделия – туристические путевки (travel).
19. Цветочная лавка. В качестве компонентов выступают различные цветы и украшения к ним. Изделия – цветочные композиции (flower).
20. Ювелирная лавка. В качестве компонентов выступают различные драгоценные камни и металлы. Изделия – драгоценности (jewel).
21. Авиастроительный завод. В качестве компонентов выступают различные части для сборки самолета (двигатели, крылья, фюзеляж и т.п.). Изделия – самолеты (plane).
22. Магазин подарков. В качестве компонентов выступают различные упаковочные материалы, ленты и подарки. Изделия – подарочные наборы (gift).
23. Система безопасности. В качестве компонентов выступают различные камеры, датчики и т.п. Изделия – базовые комплектации охраны, продвинутые, для предприятий, для частных и т.п. (secure).

24. Заказы еды. В качестве компонентов выступают различные блюда. Изделия – это наборы блюд (типа обеденный набор, или утренний набор, или набор для пикника) (dish).
25. Ремонт сантехники. В качестве компонентов выступают различные трубы, прокладки, смесители т.п. Изделия – замены смесителей, труб и т.п. (work).
26. Лавка с мороженым. В качестве компонентов выступают различные виды мороженного и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icecream).
27. Судостроительный завод. В качестве компонентов выступают различные части для сборки судов (корпуса, двигатели и т.п.). Изделия – суда (ship).
28. Столярная мастерская. В качестве компонентов выступают различные деревянные заготовки. Изделия – деревянные игрушки, утварь и т.п. (wood).
29. Бар. В качестве компонентов выступают различные ингредиенты для коктейлей. Изделия – коктейли (cocktail).
30. Швейная фабрика. В качестве компонентов выступают различные заготовки для штор, покрывал и т.п. (textile).