

Тема 11. Класс HttpClient. Деплой приложений.

Цель занятия:

**Ознакомиться с возможностями
класса `HttpClient` и с основами
деплой приложений.**

Учебные вопросы:

- 1. Выполнение HTTP-запросов. Класс HttpClient.**
- 2. Деплой приложений.**

1. Выполнение HTTP-запросов.

Класс `HttpClient`.

`HttpClient` — это основной класс для выполнения HTTP-запросов и получения ответов из веб-ресурсов. Он позволяет работать с протоколами HTTP/HTTPS, отправлять запросы и обрабатывать ответы в приложениях .NET.

HTTP (HyperText Transfer Protocol) — это протокол прикладного уровня для передачи данных в сети, основанный на модели «клиент-сервер».

Он используется для обмена информацией между веб-браузерами (клиентами) и веб-серверами.

Основные характеристики:

- Работает поверх TCP/IP — использует TCP-соединения для передачи данных.
- Запрос-ответ — клиент отправляет запрос, сервер возвращает ответ.
- Статус-коды — ответы сервера содержат коды (например, 200 — OK, 404 — Not Found).
- Методы запросов — GET (получить данные), POST (отправить данные), PUT (обновить), DELETE (удалить) и другие.
- Без состояния — каждый запрос обрабатывается независимо, без сохранения информации о предыдущих запросах (для этого используются куки или сессии).
- HTTPS — защищённая версия HTTP с шифрованием через SSL/TLS.

Как это работает?

- Пользователь вводит URL в браузере (например, `https://example.com`).
- DNS-запрос: браузер преобразует доменное имя (`example.com`) в IP-адрес сервера через систему DNS (Domain Name System).
- HTTP-запрос: браузер отправляет запрос на сервер по протоколу HTTP или HTTPS (защищённая версия).
- Сервер обрабатывает запрос: сервер находит запрашиваемый ресурс (например, HTML-страницу) и отправляет его обратно браузеру.
- Браузер отображает страницу: браузер получает HTML, CSS, JavaScript и другие файлы, затем отрисовывает страницу на экране.

HTTP-запросы — это способы взаимодействия клиента (например, браузера) с сервером. Основные типы запросов — GET, POST, PUT, DELETE.

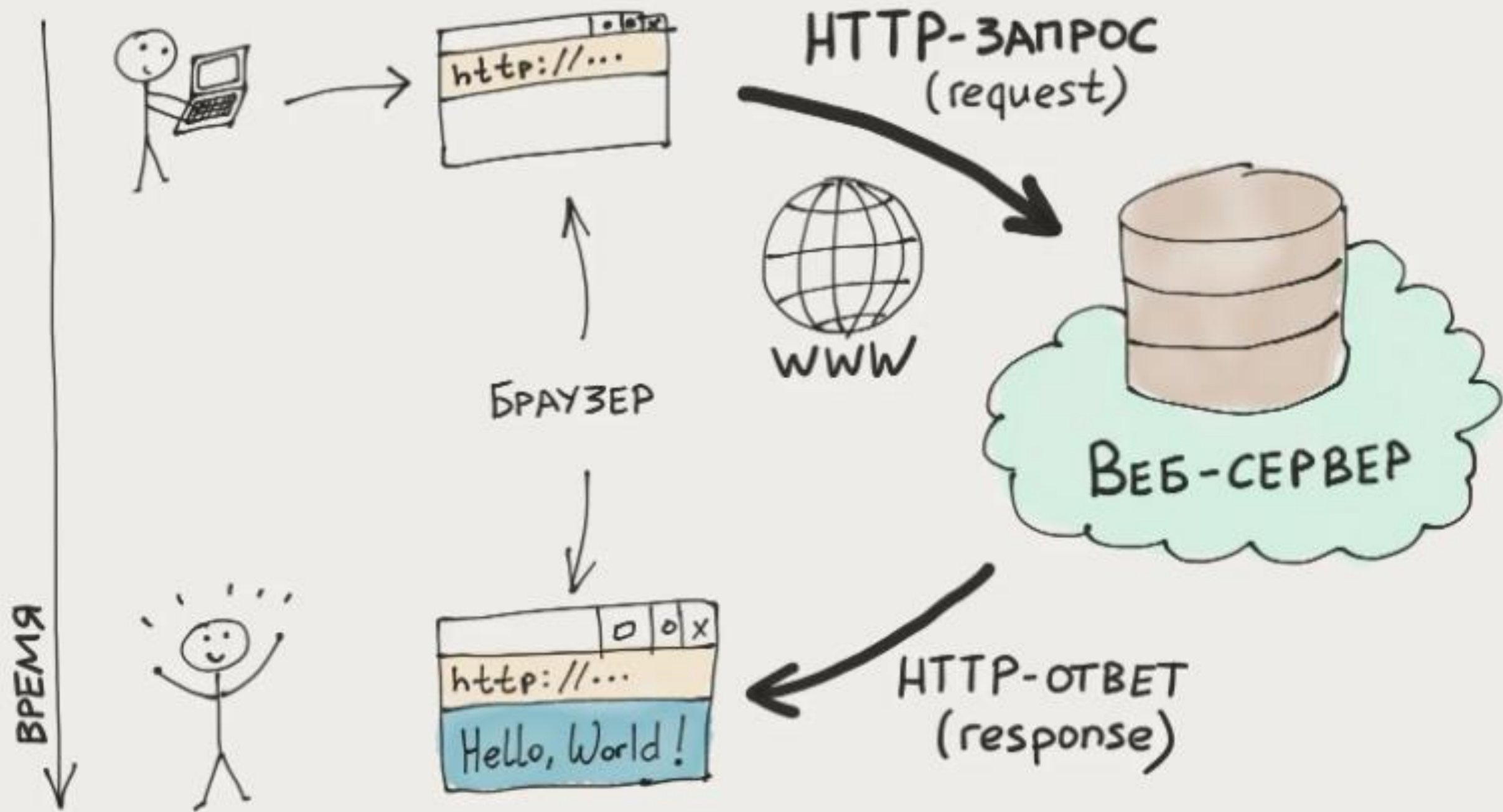
Они соответствуют операциям CRUD (Create, Read, Update, Delete) и используются для выполнения различных действий с данными на сервере.

Запрос состоит из трёх основных частей:

- Стартовая строка (Request Line). Определяет тип запроса (HTTP-метод), URL-адрес и версию HTTP.
- Заголовки (Headers). Дополнительная информация о запросе (кодировка, тип данных, авторизация и т. д.).
- Тело (Body) (только для некоторых запросов, например, POST, PUT). Содержит данные, отправляемые серверу.

Пример простого HTTP-запроса

```
GET /index.html HTTP/1.1 # Метод запроса (GET), запрашиваемый ресурс
                          # (/index.html) и версия протокола (HTTP/1.1)
Host: example.com        # Домен сервера, к которому отправляется запрос
User-Agent: Mozilla/5.0   # Информация о клиенте (браузер, приложение)
Accept: text/html        # Какие типы данных клиент хочет получить (HTML)
Connection: keep-alive    # Просьба не закрывать соединение сразу
```



HTTP-ответ (HTTP Response)— это сообщение, которое сервер отправляет клиенту (например, браузеру или приложению) в ответ на HTTP-запрос. Ответ содержит статус выполнения запроса, заголовки и, при необходимости, тело ответа.

HTTP-ответ состоит из трёх основных частей:

- Стартовая строка (Status Line). Содержит версию HTTP, код состояния и его текстовое описание.
- Заголовки (Headers)
- Тело (Body) (не всегда присутствует)

Пример HTTP-ответа

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 11

Hello, world!
```

Где:

HTTP/1.1: Версия протокола HTTP.

200 OK: Запрос обработан успешно.

Content-Type: text/html: Тип содержимого — HTML.

Content-Length: 11: Длина содержимого в байтах.

Hello, world!: Тело ответа, содержащее текст.

Основные HTTP-статус-коды

- 1xx – Информационные (редко используются)
- **2xx** – Успешные ответы. Например:
 - 200 OK – Запрос успешно обработан.
 - 201 Created – Успешно создан новый ресурс.
- 3xx – Перенаправления
- **4xx** – Ошибки клиента
- **5xx** – Ошибки сервера

[Подробнее про статус-коды](#)

Основные HTTP-методы

1. GET

Назначение: **Получить** данные с сервера.

Использование: Запрос данных (например, веб-страницы, изображения, JSON).

Особенности:

- Данные передаются в URL (видимы в адресной строке).
- Не изменяет состояние сервера (идемпотентный).
- Имеет ограничение на длину URL.

2. POST

Назначение: **Отправить** данные на сервер для создания нового ресурса.

Использование: Отправка форм, загрузка файлов, создание записей в базе данных.

Особенности:

- Данные передаются в теле запроса (не видны в URL).
- Может изменять состояние сервера.
- Не идемпотентный (повторный запрос может создать дубликат).

3. PUT

Назначение: **Обновить** существующий ресурс или создать новый, если он не существует.

Использование: Редактирование данных (например, обновление профиля пользователя).

Особенности:

- Данные передаются в теле запроса.
- Идемпотентный (повторный запрос не изменяет результат).
- Обычно требует указания идентификатора ресурса.

4. DELETE

Назначение: **Удалить** ресурс на сервере.

Использование: Удаление данных (например, удаление пользователя).

Особенности:

- Обычно не требует тела запроса.
- Идемпотентный (повторный запрос не изменяет результат).

Основные методы HttpClient:

1. GetAsync

- Отправляет GET-запрос на указанный URI.
- Асинхронный метод.
- Возвращает **HttpResponseMessage**.

```
HttpResponseMessage response = await client.GetAsync("https://example.com");
```

2. GetStringAsync

- Отправляет GET-запрос и возвращает ответ в виде строки.
- Асинхронный метод.
- Удобен для получения текстового содержимого (например, HTML, JSON).

```
string content = await client.GetStringAsync("https://example.com");
```

3. **GetByteArrayAsync**

- Отправляет GET-запрос и возвращает ответ в виде массива байт.
- Асинхронный метод.
- Полезен для загрузки бинарных данных (например, изображений).

```
byte[] data = await client.GetByteArrayAsync("https://example.com/image.png");
```

4. GetStreamAsync

- Отправляет GET-запрос и возвращает ответ в виде потока (Stream).
- Асинхронный метод.
- Подходит для обработки данных по мере их поступления. Скачивание больших файлов. Поточковая обработка данных в реальном времени (аудио, видео).
- Возвращает асинхронную задачу (Task<Stream>), содержащую поток данных (Stream) нужно обрабатывать (например, передать в аудиоплеер или видеопроигрыватель)

```
Stream stream = await client.GetStreamAsync("https://example.com");
```

5. PostAsync

- Отправляет POST-запрос на указанный URI с передачей данных.
- Асинхронный метод.
- Возвращает **HttpResponseMessage**.

```
var content = new StringContent("{\"name\":\"John\"}", Encoding.UTF8, "application/json");  
HttpResponseMessage response = await client.PostAsync("https://example.com/api", content);
```

6. PutAsync

- Отправляет PUT-запрос на указанный URI с передачей данных.
- Асинхронный метод.
- Возвращает **HttpResponseMessage**.

```
var content = new StringContent("{\"name\":\"John\"}", Encoding.UTF8, "application/json");  
HttpResponseMessage response = await client.PutAsync("https://example.com/api/1", content);
```

7. DeleteAsync

- Отправляет DELETE-запрос на указанный URI.
- Асинхронный метод.
- Возвращает **HttpResponseMessage**.

```
HttpResponseMessage response = await client.DeleteAsync("https://example.com/api/1");
```

8. SendAsync

- Отправляет произвольный HTTP-запрос, описанный объектом `HttpRequestMessage`.
- Асинхронный метод.
- Возвращает **`HttpResponseMessage`**.
- Полезен для создания сложных запросов.

```
var request = new HttpRequestMessage(HttpMethod.Post, "https://example.com/api");  
request.Content = new StringContent("{ \"name\": \"John\" }", Encoding.UTF8, "application/json");  
HttpResponseMessage response = await client.SendAsync(request);
```


HttpResponseMessage — это класс в .NET, который представляет собой HTTP-ответ, полученный от сервера после выполнения HTTP-запроса с помощью HttpClient.

Этот класс содержит всю информацию о ответе: статус-код, заголовки, содержимое и другие данные.

Основные свойства HttpResponseMessage:

1. StatusCode

- Возвращает HTTP-статус код ответа (например, 200 OK, 404 Not Found, 500 Internal Server Error).
- Тип: HttpStatusCode.

```
HttpStatusCode statusCode = response.StatusCode;
```

2. **IsSuccessStatusCode**

- Возвращает true, если статус-код ответа находится в диапазоне 200-299 (успешные запросы).
- Тип: bool.

3. **Content**

- Возвращает содержимое ответа (тело ответа).
- Тип: `HttpContent`.
- Содержимое может быть прочитано различными способами: как строка, массив байт или поток.

Свойство **Content** имеет тип **HttpContent**. Это абстрактный класс, который предоставляет методы для чтения содержимого HTTP-ответа в различных форматах. Например:

- **ReadAsStringAsync()** — для чтения содержимого как строки.
- **ReadAsStreamAsync()** — для чтения содержимого как потока.
- **ReadAsByteArrayAsync()** — для чтения содержимого как массива байтов.

```
string content = await response.Content.ReadAsStringAsync();
```

4. Headers

- Возвращает коллекцию HTTP-заголовков ответа.
- Тип: `HttpResponseHeaders`.

```
foreach (var header in response.Headers)
{
    Console.WriteLine($"{header.Key}: {string.Join(", ", header.Value)}");
}
```

5. ReasonPhrase

- Возвращает фразу, связанную с статус-кодом (например, "ОК" для кода 200).
- Тип: string.

6. Version

- Возвращает версию HTTP-протокола, используемую в ответе (например, HTTP/1.1).
- Тип: Version.

7. RequestMessage

- Возвращает объект HttpRequestMessage, который был отправлен для получения этого ответа.
- Тип: HttpRequestMessage.

Пример

```
// Обработчик нажатия на кнопку
```

Ссылка: 1

```
async void btnSendRequest_Click(object sender, EventArgs e)
{
```

```
    // Очищаем текстовое поле перед новым запросом
    textBox.Clear();
```

```
    // Создаем экземпляр HttpClient
    using (HttpClient client = new HttpClient())
    {
```

```
        try
        {
```

```
            // Выполняем GET-запрос
            HttpResponseMessage response =
                await client.GetAsync("https://www.catec.kz/");
```

```
            // Получаем код ответа
            int statusCode = (int)response.StatusCode;
            textBox.AppendText($"Код ответа: {statusCode}\r\n");
```

```
        }
```

```
        catch (Exception ex)
        {
```

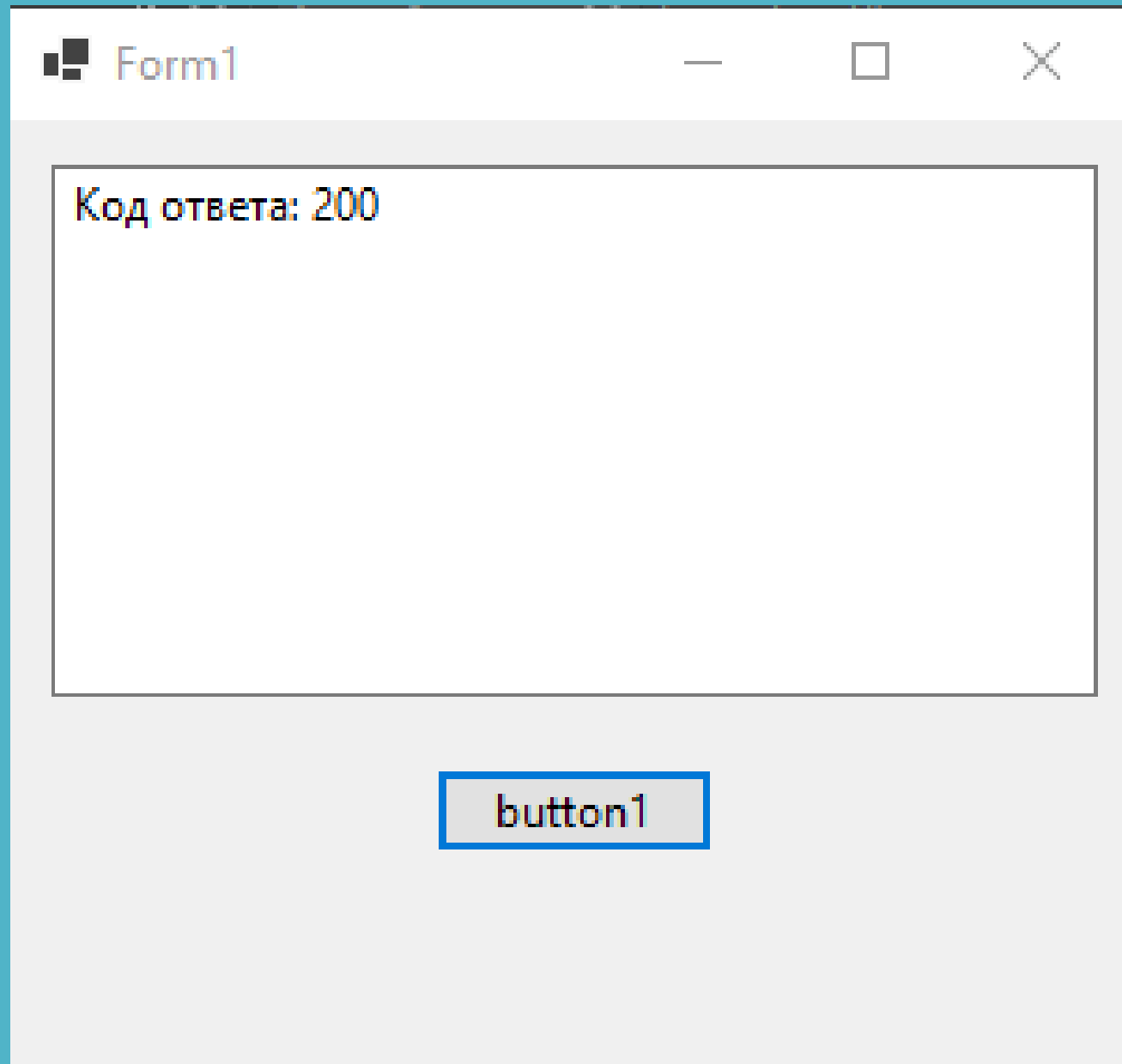
```
            textBox.AppendText(ex.Message);
```

```
        }
```

```
    }
```

```
}
```

Результат:



The image shows a screenshot of a Windows application window titled "Form1". The window has a standard Windows title bar with minimize, maximize, and close buttons. Inside the window, there is a large text area containing the text "Код ответа: 200". Below the text area, there is a button labeled "button1". The button has a blue border and a light gray background.

Десериализация Json. Класс System.Text.Json

JSON (JavaScript Object Notation) — это легковесный формат обмена данными, который используется для передачи структурированной информации между **клиентом и сервером**, а также для хранения данных. JSON легко читается и пишется как человеком, так и компьютером.

Основные принципы JSON:

- Структура JSON основана на парах **ключ-значение**.
- Данные записываются в виде объектов и массивов.
- JSON строго типизирован: значения могут быть определённых типов.

Типы данных в JSON

JSON поддерживает следующие типы данных:

1. Объекты:

- Представляют собой набор пар ключ-значение.
- Ключи — это строки, а значения могут быть любыми из допустимых типов.
- Объект заключается в фигурные скобки {}

```
{  
  "name": "John",  
  "age": 30,  
  "isEmployed": true  
}
```

2. Массивы:

Представляют собой упорядоченный список значений.
Заключаются в квадратные скобки [].

Пример:

```
{  
  "languages": ["JavaScript", "Python", "Java"]  
}
```

3. Значения:

Могут быть строками, числами, логическими значениями (true/false), null или другими объектами/массивами.

```
{  
  "price": 19.99,  
  "quantity": 3  
}
```

```
{  
  "message": "Hello, world!"  
}
```

JSON поддерживает вложение объектов и массивов друг в друга. Это позволяет создавать сложные структуры данных.

Пример:

```
{  
  "user": {  
    "id": 123,  
    "name": "John Doe",  
    "contacts": {  
      "email": "john.doe@example.com",  
      "phone": "+123456789"  
    },  
    "skills": ["JavaScript", "Python", "C++"]  
  }  
}
```

JSON активно используется в веб-разработке для передачи данных между клиентом (например, браузером) и сервером.

Пример использования в API.

Запрос к API для получения данных о пользователе:

```
{  
  "id": 101,  
  "name": "Alice",  
  "email": "alice@example.com",  
  "isActive": true  
}
```

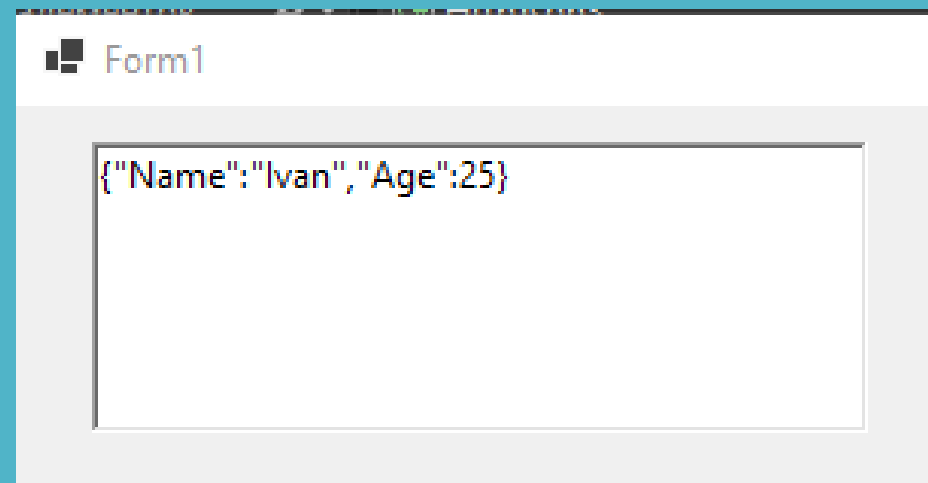
Сериализация и десериализация JSON

Сериализация – это процесс преобразования объекта в формат JSON (JavaScript Object Notation) для хранения или передачи данных.

Десериализация – это обратный процесс: преобразование JSON в объект.

Пример:

```
class Person
{
    Ссылка: 1
    public string Name { get; set; }
    Ссылка: 1
    public int Age { get; set; }
}
```



The screenshot shows a standard Windows application window titled "Form1". Inside the window is a RichTextBox control. The text displayed in the RichTextBox is a JSON object: {"Name":"Ivan","Age":25}. The text is in a monospaced font and is black on a white background.

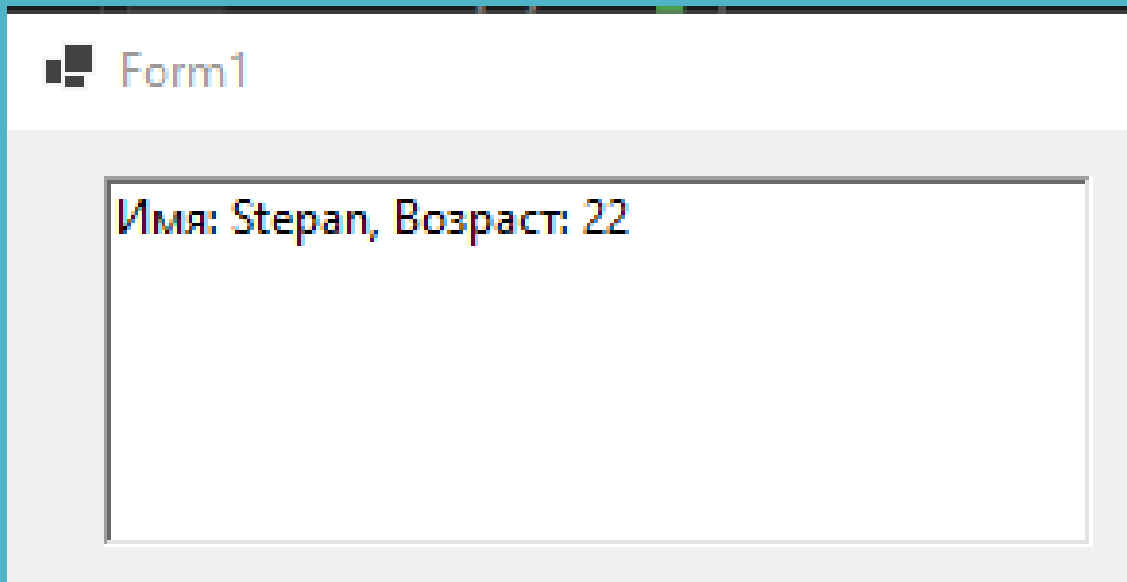
```
private void button1_Click(object sender, EventArgs e)
{
    var person = new Person { Name = "Ivan", Age = 25 };
    // Сериализация в JSON
    string json = JsonSerializer.Serialize(person);
    richTextBox1.Text = json;
}
```



```
private void button2_Click(object sender, EventArgs e)
{
    string json = "{\"Name\":\"Stepan\",\"Age\":22}";

    // Десериализация JSON в объект
    Person person = JsonSerializer.Deserialize<Person>(json);

    richTextBox1.Text = $"Имя: {person.Name}, Возраст: {person.Age}";
}
```



The screenshot shows a Windows Form titled "Form1". Inside the form, there is a text box containing the text "Имя: Stepan, Возраст: 22".

Алгоритм десериализации JSON-ответа API

1. Определить модель данных (class)

- Описать только нужные поля (лишние игнорируются).
- Если JSON-ключи отличаются от имен свойств, использовать `[JsonPropertyName("json_key")]`.
- Если поля могут отсутствовать, использовать `Nullable<T>` (`int?`, `double?`, `DateTime?`).

2. Настроить JsonSerializerOptions

- `PropertyNameCaseInsensitive = true` → игнорирует регистр имен полей.
- `IgnoreUnknownProperties = true` → игнорирует неизвестные поля в JSON.

3. Создать HttpClient для отправки запроса к API

- Установить базовый URL API, если это потребуется для нескольких запросов.
- Добавить заголовки (headers), если API требует аутентификацию.

4. Выполнить GET-запрос и получить JSON-строку

- Использовать `HttpClient.GetStringAsync(url)`, если API просто отдает JSON.
- Или `HttpClient.GetAsync(url)`, если нужно обработать статус ответа (404, 500 и т. д.).
- 5. Использовать `JsonSerializer.Deserialize<T>()` для преобразования JSON в объект
- Передавать `JsonSerializerOptions`, чтобы не зависеть от регистра и игнорировать лишние поля.

6. Обработать возможные ошибки

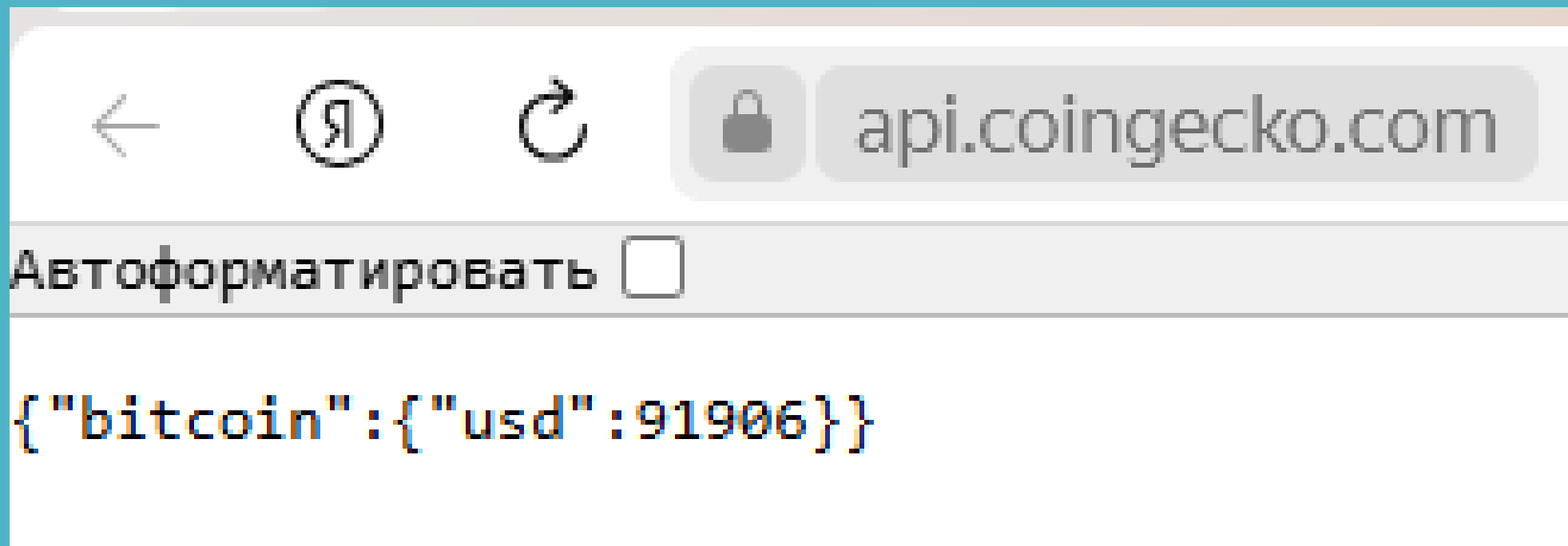
- `HttpRequestException` → если ошибка сети или API вернул ошибку.
- `JsonException` → если JSON некорректен или не соответствует модели.
- `TaskCanceledException` → если превышено время ожидания ответа.

7. Вывести результат или передать данные в приложение

- Вывести в консоль, UI или сохранить в базу данных.

Пример. Получения курса биткоина в USD с API CoinGecko.

API CoinGecko возвращает JSON следующего вида (https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=usd):



Создадим классы для десериализации:

```
// Класс для хранения цены BTC
class CryptoPrice
{
    public decimal Usd { get; set; } // Курс BTC в USD
}

// Основной класс для десериализации JSON-ответа
class CryptoResponse
{
    public CryptoPrice Bitcoin { get; set; }
}
```

Настройка JsonSerializerOptions

```
var options = new JsonSerializerOptions
{
    ....
    PropertyNameCaseInsensitive = true, // Игнорирует регистр (bitcoin = Bitcoin)
    DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull // Игнорирует ненужные поля
};
```

Создание HttpClient и подготовка URL запроса:

```
readonly HttpClient client = new HttpClient();  
readonly string url =  
    "https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=usd";
```

Отправляем запрос и получаем JSON:

```
string json = await client.GetStringAsync(url);
```

- GetStringAsync(url) скачивает JSON как строку.
- Если API вернет ошибку (404, 500 и т. д.), нужно обработать исключение.

Десериализация JSON в объект

```
var response = JsonSerializer.Deserialize<CryptoResponse>(json, options);
```

- JSON-строка преобразуется в объект CryptoResponse.
- Если JSON-структура не соответствует модели, возникнет JsonException

Обработка ошибок (HttpRequestException, JsonException)

```
catch (HttpRequestException ex)
{
    MessageBox.Show($"Ошибка сети: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    txtPrice.Text = "Ошибка сети";
}
catch (JsonException ex)
{
    MessageBox.Show($"Ошибка JSON: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    txtPrice.Text = "Ошибка данных";
}
```

- HttpRequestException → если API недоступно или нет интернета.
- JsonException → если JSON-ответ изменился или не соответствует модели.

Вывод результата

```
txtPrice.Text = $"{response?.Bitcoin?.Usd ?? 0m} USD";
```

? (Null-conditional operator) — оператор, который проверяет, не является ли объект слева от него **null**. Если объект равен **null**, выражение сразу возвращает **null**, не вызывая исключения.

?? (Null-coalescing operator) возвращает значение слева от него, если оно не равно **null**. Если значение слева равно **null**, используется значение справа.

Полный код программы прилагается.

2. Деплой приложений.

Деплой (от английского deploy — развертывание) — это процесс переноса разработанного приложения, сервиса или системы из среды разработки в рабочую среду, где оно становится доступным для конечных пользователей.

Деплой включает в себя все этапы, необходимые для того, чтобы приложение начало работать на целевой платформе (например, на сервере, компьютере пользователя или в облаке).

Основные аспекты деплоя:

Подготовка приложения:

- Сборка проекта (компиляция кода, упаковка файлов).
- Проверка зависимостей (например, библиотеки, фреймворки, базы данных).

Перенос на целевую среду:

- Копирование файлов на сервер или устройство пользователя.
- Настройка окружения (например, установка необходимого ПО, настройка переменных среды).

Запуск и настройка:

- Запуск приложения.
- Настройка параметров (например, подключение к базе данных, настройка сети).

Тестирование:

- Проверка работоспособности приложения в новой среде.
- Устранение возможных ошибок.

Обслуживание:

- Обновление приложения.
- Мониторинг и устранение проблем.

Создание исполняемого файла.

Как сделать что бы .exe файл включал все зависимости?

Это особенно полезно, если вы хотите распространять приложение без необходимости установки дополнительных компонентов (например, .NET Runtime) на целевом компьютере. В .NET это достигается с помощью автономной публикации (Self-contained deployment).

Подготовка проекта Windows Forms в Visual Studio к публикации.

1. Проверка конфигурации сборки

Переключитесь на конфигурацию "Release":

Откройте меню Build (Построение) → выберите Configuration Manager.

Убедитесь, что для вашего проекта выбрана конфигурация Release (она оптимизирована для конечного использования, в отличие от Debug).

2. Настройка целевой платформы

В списке Platform выберите нужную платформу (например, x86, x64 или Any CPU).

Если у вас выбрано Any CPU, приложение будет работать на любой платформе, но для 64-разрядных систем оно будет автоматически работать в 64-разрядном режиме.

Проверьте Target Framework:

В Solution Explorer щелкните правой кнопкой мыши на проект → Properties.

Перейдите на вкладку Application.

Убедитесь, что в поле Target Framework выбрана нужная версия .NET (например, .NET Framework 4.7.2).

3. Добавление внешних файлов (если есть)

Для каждого файла:

Щелкните правой кнопкой мыши по файлу → Properties.

Установите:

Build Action: Content.

Copy to Output Directory: Copy if newer или Copy always.

4. Соберите проект:

Нажмите Build → Build Solution (Ctrl+Shift+B).

Visual Studio сгенерирует исполняемый файл (.exe) и связанные файлы в папке bin\Release

Публикация проекта (Deploy)

Откройте проект в Visual Studio:

- Щелкните правой кнопкой мыши на проекте и выберите Publish.

Настройте публикацию:

- Выберите Folder как целевой вариант публикации.
- В разделе Deployment Mode выберите Self-contained.
- В разделе Target Runtime выберите целевую платформу (например, win-x64 для 64-битной Windows или win-x86 для 32-битной Windows).
- Нажмите Configure.

Опубликуйте приложение:

- Нажмите Publish.
- Visual Studio создаст папку с вашим приложением и всеми необходимыми зависимостями.

Как изменить иконку .exe файла?

Подготовьте иконку:

- Убедитесь, что у вас есть файл иконки (обычно с расширением .ico). Вы можете создать его с помощью онлайн-конвертеров или графических редакторов.

Добавьте иконку в проект:

- Перетащите файл иконки в папку проекта в Solution Explorer.
- Или щелкните правой кнопкой мыши на проекте, выберите Add -> Existing Item и добавьте файл иконки.

Установите иконку для приложения:

- Щелкните правой кнопкой мыши на проекте в Solution Explorer и выберите Properties.
- Перейдите в раздел Application.
- В поле Icon and manifest выберите вашу иконку из выпадающего списка.
- Если иконки нет в списке, нажмите Browse и выберите файл .ico.

Установите иконку для главной формы:

- Откройте главную форму (например, MainForm.cs) в дизайнере.
- В окне Properties найдите свойство Icon и выберите вашу иконку.

Пересоберите проект:

- Нажмите Build -> Build Solution, чтобы применить изменения.

Создание установщика (MSI или EXE)

Для создания установщика можно использовать Microsoft Visual Studio Installer Projects (расширение для Visual Studio).

Для установки расширений используется диспетчер расширений. (Расширения => Управление расширениями)

1. Установка расширения "Microsoft Visual Studio Installer Projects"

Если у вас еще не установлено это расширение:

- Откройте Visual Studio.
- Перейдите в меню Extensions -> Manage Extensions.
- В поиске найдите Microsoft Visual Studio Installer Projects.
- Установите расширение и **перезапустите** Visual Studio.

Диспетчер расширений

Form1.cs*

Form1.cs [Конструирование]*

Обзор

Установлен

Обновлен

Роуминг

installer

Microsoft Visual Studio Installer Projects 2022

328 218

Установить

Visual & Installer

55 408

Установить

Package Installer

32 520

Установить

Microsoft Visual Studio

Microsoft

Установить

Просмотреть в браузере

Сообщить

ARM64 NOTE: If you are running Visual Studio on an ARM64 OS use this version instead.

This extension provides the same functionality that currently exists in Visual Studio 2019 for Visual Studio Installer projects. To use this extension, you can either open the Extensions and Updates dialog, select the online node, and search for "Visual Studio Installer Projects," or you can download directly from this page.

2. Создание проекта установщика.

- Откройте ваш проект (например, Windows Forms).
- Щелкните правой кнопкой мыши на решении (Solution) в Solution Explorer.
- Выберите Add -> New Project.
- В появившемся окне найдите и выберите Setup Project (или Installer Project) и нажмите “Далее”.
- Назовите проект (например, MyAppInstaller) и нажмите “Создать” (Create).

Кликнуть ПКМ

Обозреватель решений



Обозреватель решений — п

Решение "WinFormsApp2"

winformsapp2

Properties

Зависимости

pics

Form1.cs

Обозрев...

Измене...

Предста...











Свойства

WinFormsApp2 Свойства решен



(14м)



WinFormsApp2

-  Собрать решение Ctrl+ Shift+ B
- Пересобрать решение
- Очистить решение
- Анализ и очистка кода ▶
- Пакетная сборка...
- Диспетчер конфигураций...
-  Управление пакетами NuGet для решения...
-  Восстановить пакеты NuGet
-  Свернуть всех потомков Ctrl+ Стрелка влево
-  Новое представление: Обозреватель решений
- Вложение файлов ▶
- Добавить** ▶
- Синхронизация пространства имен
-  Настройка начальных проектов...
- Git ▶
-  Вставить Ctrl+V
-  Переименовать F2
-  Копировать полный путь
-  Открыть папку в проводнике


Создать проект...

Существующий проект...

Существующий веб-сайт...




 Создать элемент... Ctrl+ Shift+ A Существующий элемент... Shift+ Alt+ A Создать папку решения

Файл конфигурации установки

 New EditorConfig

Добавить новый проект

Последние шаблоны проектов

-  Приложение Windows Forms (Майкрософт) C#
-  Консольное приложение (Майкрософт) C#
-  Консольное приложение (.NET Framework) C#

[Очистить](#)

C#

Все платформы

Все типы проектов

Точных соответствий не найдено.

Другие результаты на основе вашего поиска



Setup Project

Create a Windows Installer project to which files can be added

Создать



Web **Setup** Project

Create a Windows Installer web project to which files can be added

Создать



Setup Wizard

Create a Windows Installer project with the aid of a wizard.

Создать

Не нашли то, что искали?
[Установка других средств и компонентов](#)

Настроить новый проект

Setup Project

Имя проекта

DiceSetup

Расположение

D:\RepoColledge\csharp\dice

Проект будет создан в "D:\RepoColledge\csharp\dice\DiceSetup\"

Назад

Создать

3. Настройка проекта установщика.

3.1.Добавление выходных файлов приложения:

- Щелкните ЛКМ на папке “Application Folder”.
- В правом окне нажмите ПКМ, затем “Add” => “Выходной элемент проекта” и нажать “ОК”

File System (DiceSetup)

Form1.cs*

Form1.cs [Конструирование]*



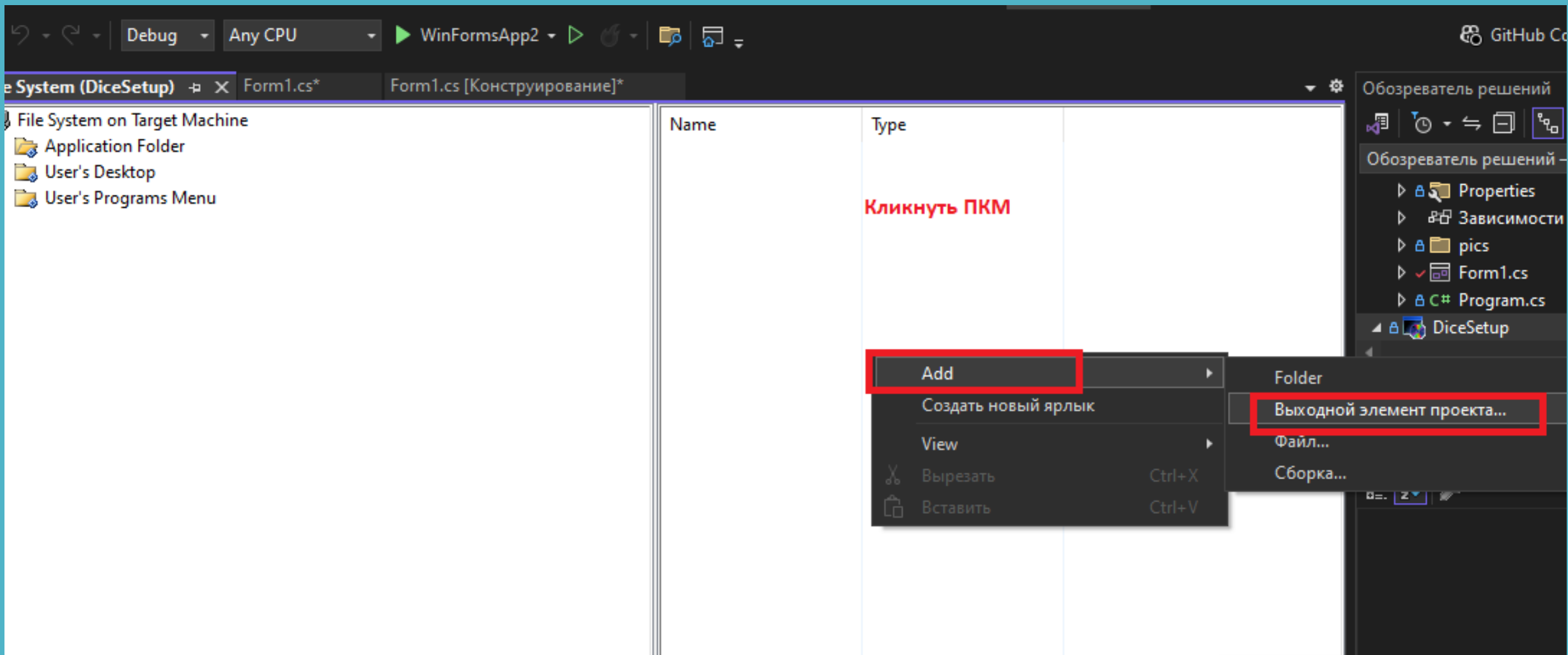
File System on Target Machine

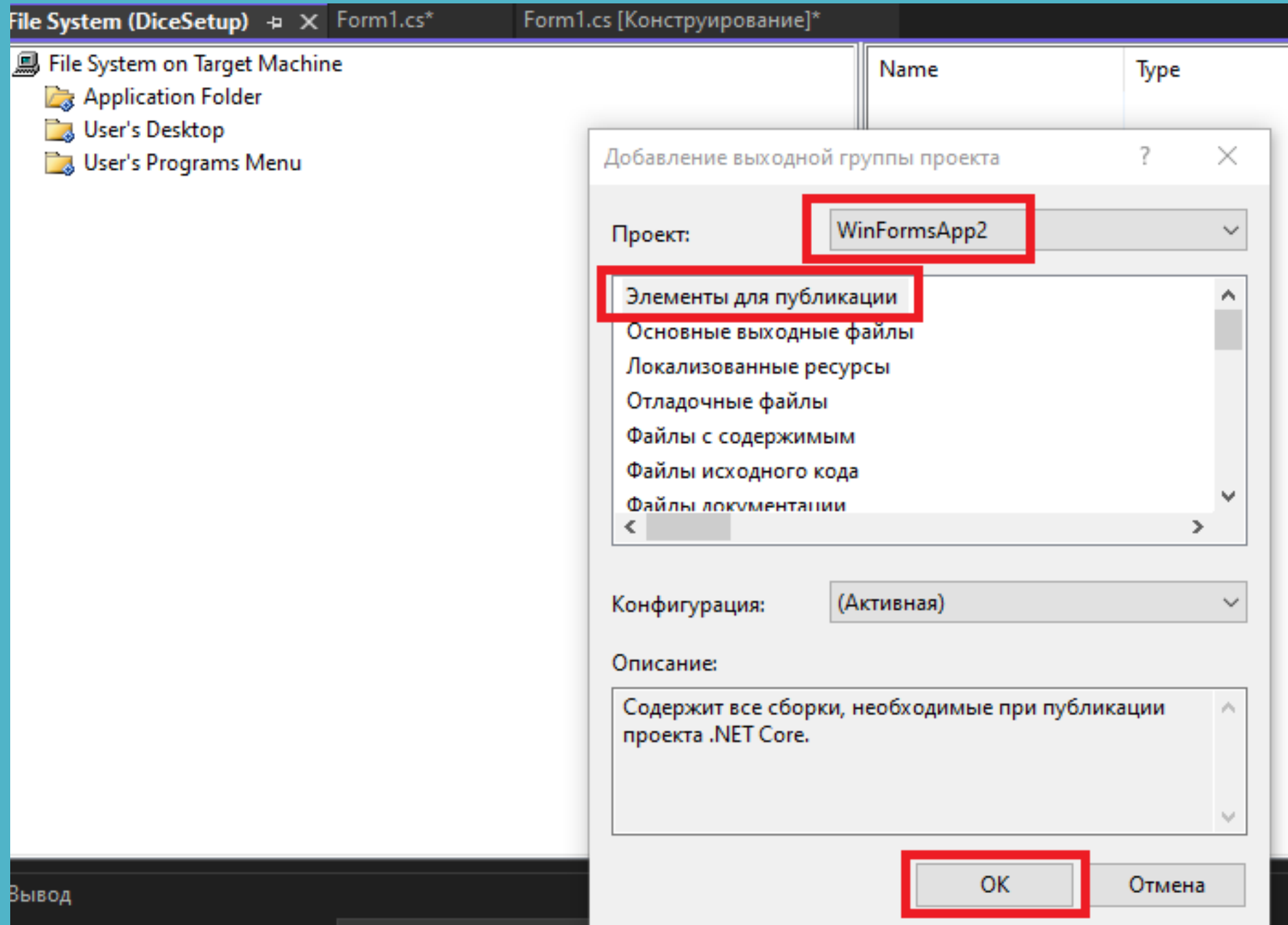
Application Folder

User's Desktop

User's Programs Menu

Name	Type
Application Folder	Folder
User's Desktop	Folder
User's Programs Me...	Folder








File System (DiceSetup)


Form1.cs*


Form1.cs [Конструирование]*

 File System on Target Machine

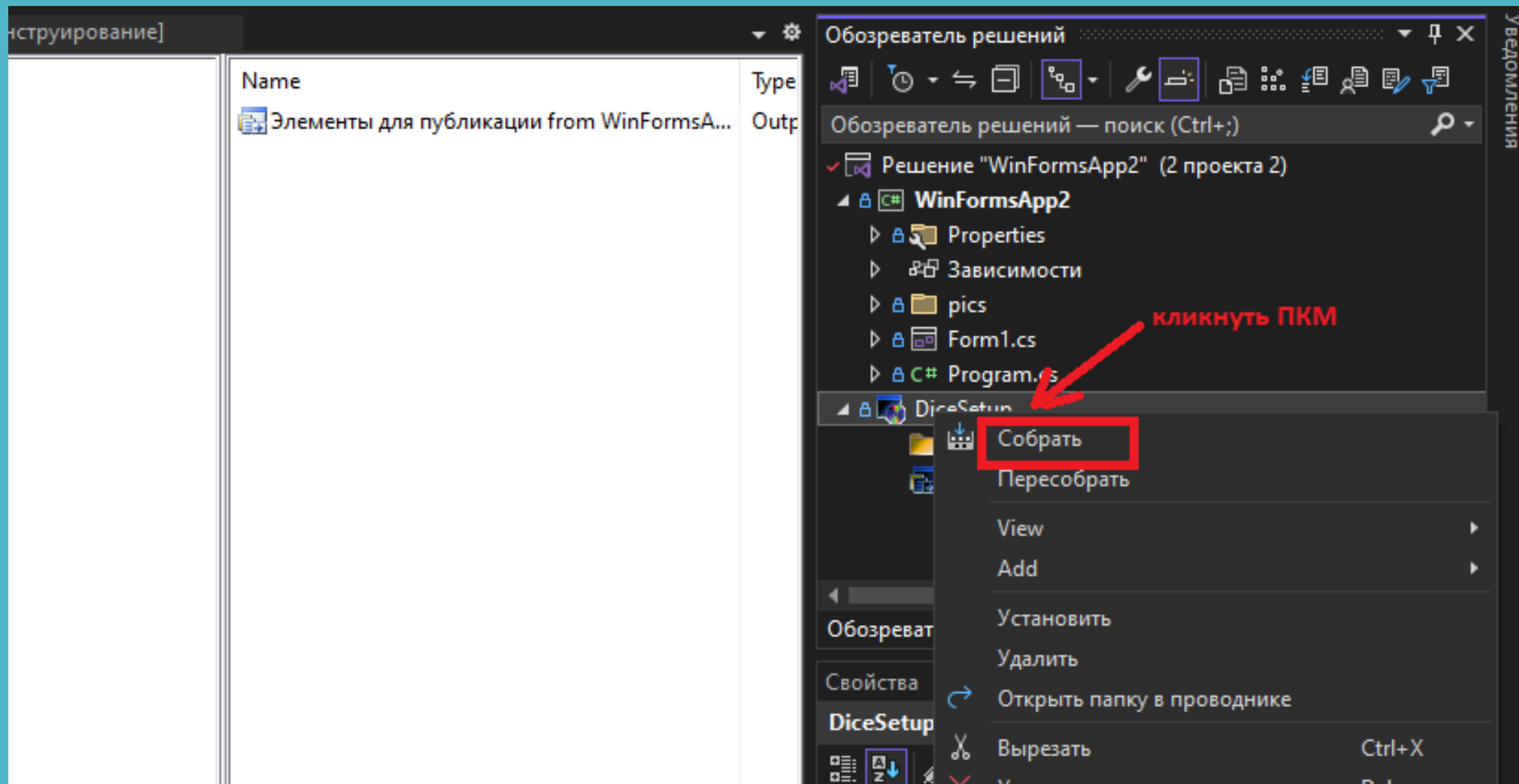
 Application Folder

 User's Desktop

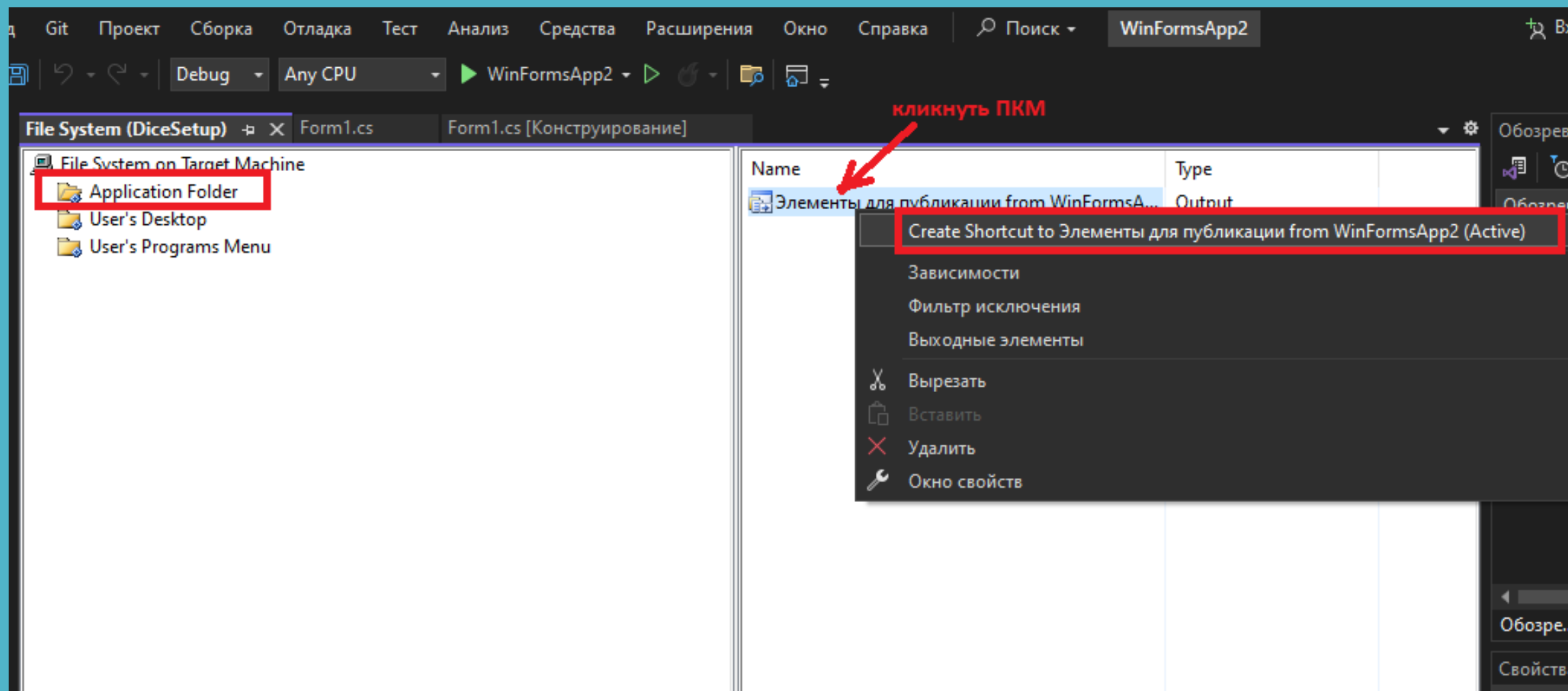
 User's Programs Menu

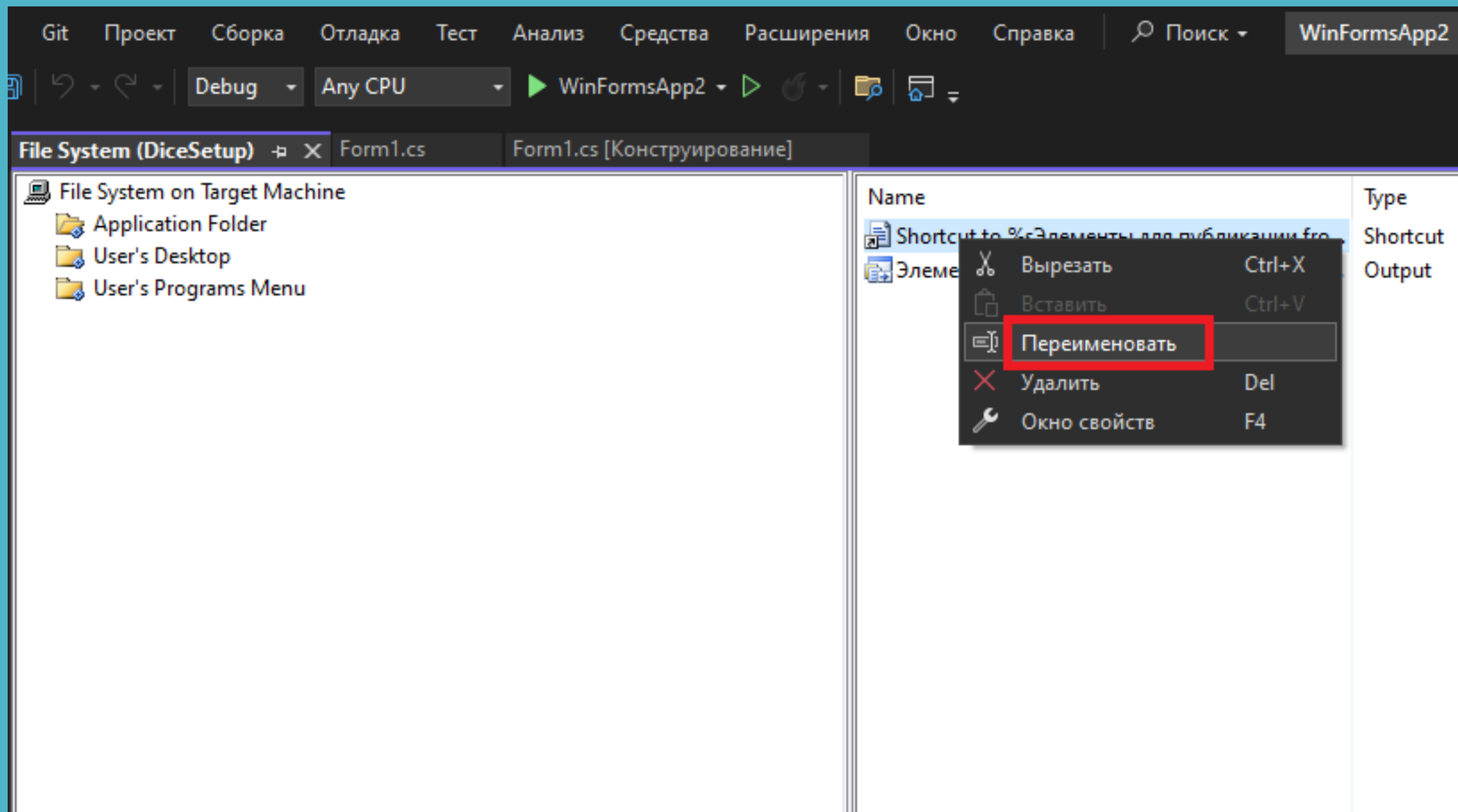
Name	Type
 Элементы для публикации from WinFormsA...	Output

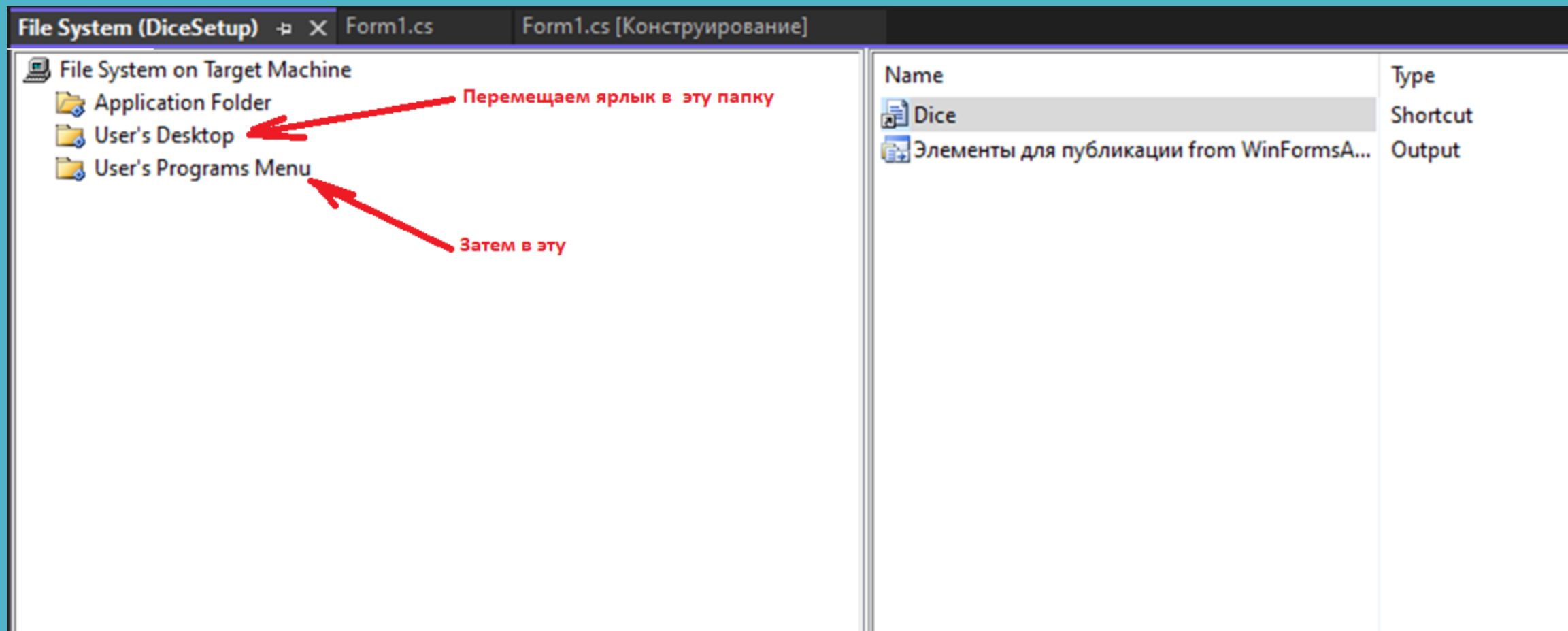
- Далее можно собрать проект, нажав ПКМ на проекте в обозревателе решений и выбрав “Собрать”



Или можно перед сборкой настроить проект, например создать ярлыки в меню и на рабочем столе.







- Далее можно настроить свойства проекта, такие как:
- Author — автор приложения.
- Manufacturer — название компании или разработчика.
- ProductName — название продукта.
- Version — версия установщика.
- Вы также можете настроить иконку для установщика и другие параметры.

...
1'...

ешно - 2, неудачно - 0, пропущено - 0 =====
о в 12:23 и заняло 24,590 с =====

Properties

Зависимости

pics

Form1.cs

Program.cs

DiceSetup

Detected Dependencies

Элементы для публикации from WinFormsApp2 (Active)

Обозреватель решений

Изменения Git

Представление классов

Свойства

DiceSetup Deployment Project Properties

AddRemoveProgramsIcon	(None)
Author	Catec and Co
BackwardCompatibilityGenerator	False
Description	
DetectNewerInstalledVersion	True
InstallAllUsers	False
Keywords	
Localization	Russian
Manufacturer	Catec and Co
ManufacturerId	

Manufacturer

Specifies the name of the manufacturer of an application or component

Сборка установщика

- Выберите конфигурацию сборки (например, Release).
- Щелкните правой кнопкой мыши на проекте установщика и выберите Build.
- После успешной сборки в папке bin\Release (или bin\Debug) проекта установщика вы найдете файл .msi — это ваш установочный пакет.

Подробнее можно посмотреть [тут](#)

Список литературы:

1. [Видеокурс C#.](#)
2. <https://metanit.com/sharp/windowsforms/3.1.php>
3. [Как использовать HttpClient](#)
4. [Парсинг сайтов с использованием HttpClient](#)
5. [Как настроить сериализацию с System.Text.Json в C#](#)

Материалы лекций:

<https://github.com/ShViktor72/Education2025>

Задание на дом:

Задание 1.

Создайте приложение Windows Forms. Добавьте на форму:

- кнопку (button)
- текстовое поле (textBox)
- текстовое поле (richTextBox)

При нажатии кнопки должен считываться адрес сайта из поля textBox и отправляться на сайт Get-запрос.

В поле richTextBox выведите:

- статус ответа сервера
- заголовок ответа
- тело ответа
- версию протокола

Задание 2. Получение данных о погоде

Создайте приложение Windows Forms. Добавьте на форму:

- кнопку (button)
- текстовое поле (textBox)
- текстовое поле (richTextBox)

В поле textBox вводится название города.

При нажатии кнопки выполняется GET-запрос к API OpenWeatherMap для получения текущей температуры и погодных условий

Полученные данные выводятся в поле richTextBox.

Реализуйте обработку ошибок.

Задание 3. Загрузка файлов.

Разработайте приложение, которое:

- Принимает от пользователя URL изображения
- Скачивает это изображение с использованием HttpClient
- Сохраняет его локально с оригинальным именем файла
- Обрабатывает различные ошибки (недоступность файла, неверный формат и т.д.)