ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № _10_

Дисциплина: _Backend разработка

Тема:

_____

Выполнил(а): студент(ка) группы __231-336__

_____Канищев И.М_____
(Фамилия И.О.)

Дата, подпись _____ _____
(Дата) (Подпись)

Проверил: _______
_____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____ _____
(Дата) (Подпись)

Замечания:

Москва

2025

1. Описание изученных способов формирования ответов в ASP.NET Core

ASP.NET Core предоставляет множество способов формирования ответов для веб-приложений:

Основные типы ответов:

- **HTML-страницы** - через представления Razor
- **JSON данные** - для API и веб-сервисов
- **Файлы** - статические файлы или загрузка
- **Перенаправления** - редиректы на другие URL
- **Статусные коды** - HTTP статусы с сообщениями
- **Потоковые данные** - для больших объемов данных
- **Пользовательские форматы** - специализированные типы контента

2. Код примеров методов контроллеров

```
using Microsoft.AspNetCore.Mvc;
using System.Net.Mime;

namespace ResponseDemo.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ResponseController : ControllerBase
    {
        // 1. Возврат JSON данных
        [HttpGet("json")]
        public IActionResult GetJson()
        {
            var user = new
            {
                Id = 1,
                Name = "John Doe",
                Email = "john@example.com",
                CreatedAt = DateTime.UtcNow
            };

            return Ok(user); // Автоматическая сериализация в JSON
        }

        // 2. Возврат HTML контента
        [HttpGet("html")]
        public ContentResult GetHtml()
        {
            var htmlContent = @"
                <!DOCTYPE html>
                <html>
                <head>
                    <title>HTML Response</title>
                    <style>
                        body { font-family: Arial, sans-serif; margin: 40px;
}
                        .container { max-width: 800px; margin: 0 auto; }
                    </style>
                </head>
                <body>
                    <div class='container'>
                        <h1>Демонстрация HTML ответа</h1>
                        <p>Этот HTML был сгенерирован в контроллере ASP.NET
Core</p>
                        <ul>
```

```
                            <li>Пункт 1</li>
                            <li>Пункт 2</li>
                            <li>Пункт 3</li>
                        </ul>
                    </div>
                </body>
                </html>";

            return Content(htmlContent, "text/html");
        }

        // 3. Возврат файла
        [HttpGet("file")]
        public IActionResult GetFile()
        {
            // Создаем временный текстовый файл в памяти
            var fileContent = "Это содержимое текстового
файла.\nСгенерировано: " +
                              DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
            var byteArray = System.Text.Encoding.UTF8.GetBytes(fileContent);
            var stream = new MemoryStream(byteArray);

            return File(stream, "text/plain", "example.txt");
        }

        // 4. Возврат файла изображения
        [HttpGet("image")]
        public IActionResult GetImage()
        {
            // Создаем простое изображение программно
            var width = 200;
            var height = 100;
            var bitmap = new System.Drawing.Bitmap(width, height);

            using (var graphics = System.Drawing.Graphics.FromImage(bitmap))
            {
                graphics.Clear(System.Drawing.Color.LightBlue);
                graphics.DrawString("ASP.NET Core",
                    new System.Drawing.Font("Arial", 12),
                    System.Drawing.Brushes.DarkBlue,
                    new System.Drawing.PointF(10, 40));
            }

            var stream = new MemoryStream();
            bitmap.Save(stream, System.Drawing.Imaging.ImageFormat.Png);
            stream.Position = 0;

            return File(stream, "image/png", "generated-image.png");
        }

        // 5. Перенаправление
        [HttpGet("redirect")]
        public IActionResult RedirectExample()
        {
            // Перенаправление на внешний URL
            return Redirect("https://dotnet.microsoft.com/");
        }

        [HttpGet("redirect-internal")]
        public IActionResult RedirectInternal()
        {
            // Перенаправление на внутренний метод
            return RedirectToAction("GetJson");
        }
```

```
        // 6. Возврат статусных кодов
        [HttpGet("status/{code:int}")]
        public IActionResult GetStatus(int code)
        {
            return code switch
            {
                200 => Ok("Запрос успешно обработан"),
                201 => Created("/api/response/status/201", new { message =
"Ресурс создан" }),
                400 => BadRequest("Неверный запрос"),
                404 => NotFound("Ресурс не найден"),
                500 => StatusCode(500, "Внутренняя ошибка сервера"),
                _ => StatusCode(code, $"Статус код: {code}")
            };
        }

        // 7. Потоковые данные
        [HttpGet("stream")]
        public async Task<IActionResult> GetStream()
        {
            var stream = new MemoryStream();
            var writer = new StreamWriter(stream);

            // Генерируем данные постепенно
            for (int i = 1; i <= 10; i++)
            {
                await writer.WriteLineAsync($"Строка данных #{i} -
{DateTime.Now:HH:mm:ss.fff}");
                await writer.FlushAsync();
                await Task.Delay(500); // Имитация задержки обработки
            }

            stream.Position = 0;
            return File(stream, "text/plain", "stream-data.txt");
        }

        // 8. Пользовательский тип контента
        [HttpGet("csv")]
        public IActionResult GetCsv()
        {
            var csvData = @"Id,Name,Email,Date
1,Иван Иванов,ivan@example.com,2024-01-15
2,Петр Петров,petr@example.com,2024-01-16
3,Мария Сидорова,maria@example.com,2024-01-17";

            var byteArray = System.Text.Encoding.UTF8.GetBytes(csvData);
            var stream = new MemoryStream(byteArray);

            return File(stream, "text/csv", "users.csv");
        }

        // 9. Возврат представления Razor (требует настройки MVC)
        [HttpGet("view")]
        public IActionResult GetView()
        {
            ViewBag.Message = "Данные из контроллера";
            ViewBag.CurrentTime = DateTime.Now;
            return View(); // Будет искать View с именем "GetView"
        }
    }

    [Route("")]
    public class HomeController : Controller
```

```
    {
        [HttpGet]
        public IActionResult Index()
        {
            var menuItems = new[]
            {
                new { Url = "/api/response/json", Name = "JSON данные" },
                new { Url = "/api/response/html", Name = "HTML контент" },
                new { Url = "/api/response/file", Name = "Текстовый файл" },
                new { Url = "/api/response/image", Name = "Изображение" },
                new { Url = "/api/response/csv", Name = "CSV данные" },
                new { Url = "/api/response/stream", Name = "Потоковые данные"
},
                new { Url = "/api/response/redirect", Name =
"Перенаправление" },
                new { Url = "/api/response/status/200", Name = "Статус 200"
},
                new { Url = "/api/response/status/404", Name = "Статус 404" }
            };

            ViewBag.MenuItems = menuItems;
            return View();
        }
    }
}
```

3. Представление Razor (Index.cshtml)

```
@{
    ViewData["Title"] = "Демонстрация ответов ASP.NET Core";
}

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewData["Title"]</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
            max-width: 1200px;
            margin: 0 auto;
            padding: 20px;
            background-color: #f5f5f5;
        }

        .header {
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
            padding: 2rem;
            border-radius: 10px;
            margin-bottom: 2rem;
            text-align: center;
        }

        .menu-grid {
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
            gap: 1.5rem;
            margin-bottom: 2rem;
        }
```

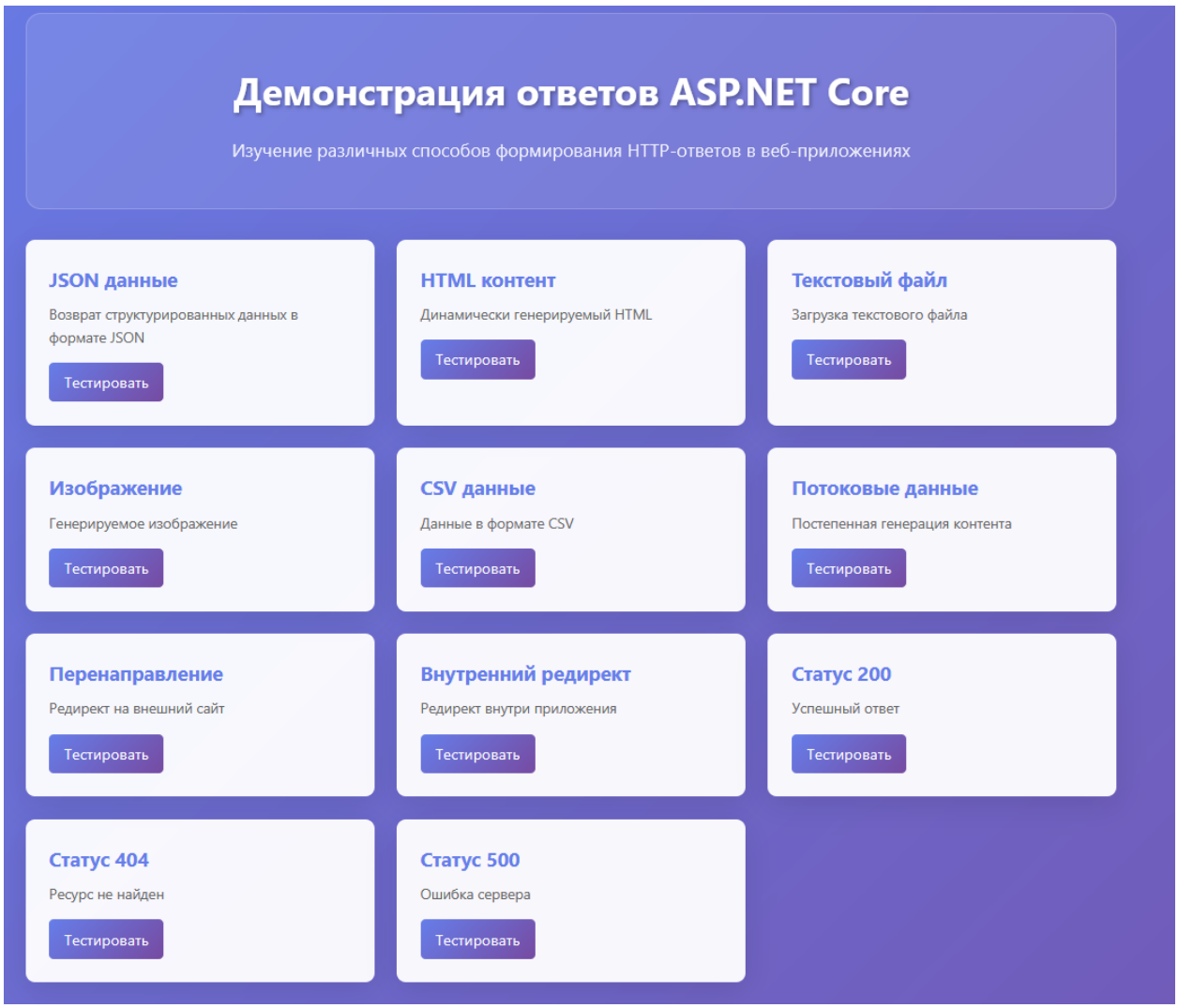
```
        .menu-card {
            background: white;
            padding: 1.5rem;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
            transition: transform 0.2s, box-shadow 0.2s;
            text-decoration: none;
            color: inherit;
            display: block;
        }

        .menu-card:hover {
            transform: translateY(-2px);
            box-shadow: 0 4px 20px rgba(0,0,0,0.15);
        }

        .menu-card h3 {
            color: #667eea;
            margin-top: 0;
            border-bottom: 2px solid #f0f0f0;
            padding-bottom: 0.5rem;
        }

        .response-info {
            background: white;
            padding: 1.5rem;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
            margin-top: 2rem;
        }

        .code-block {
            background: #f8f9fa;
            border: 1px solid #e9ecef;
            border-radius: 4px;
            padding: 1rem;
            overflow-x: auto;
            font-family: 'Consolas', monospace;
            font-size: 0.9rem;
        }
    </style>
</head>
<body>
    <div class="header">
        <h1>@ViewData["Title"]</h1>
        <p>Демонстрация различных способов формирования ответов в ASP.NET
Core</p>
    </div>

    <div class="menu-grid">
        @foreach (var item in ViewBag.MenuItems)
        {
            <a href="@item.Url" class="menu-card" target="_blank">
                <h3>@item.Name</h3>
                <p>Нажмите для тестирования ответа типа "@item.Name"</p>
            </a>
        }
    </div>

    <div class="response-info">
        <h2>О проекте</h2>
        <p>Это демонстрационное приложение показывает различные способы
формирования HTTP-ответов в ASP.NET Core:</p>
```

```
        <h3>Реализованные типы ответов:</h3>
        <ul>
            <li><strong>JSON данные</strong> - для API и веб-сервисов</li>
            <li><strong>HTML контент</strong> - динамически генерируемый
HTML</li>
            <li><strong>Файлы</strong> - текстовые файлы и изображения</li>
            <li><strong>CSV данные</strong> - пользовательский формат</li>
            <li><strong>Потоковые данные</strong> - постепенная генерация
контента</li>
            <li><strong>Перенаправления</strong> - внешние и внутренние
редиректы</li>
            <li><strong>Статусные коды</strong> - различные HTTP статусы</li>
        </ul>

        <h3>Пример кода для возврата JSON:</h3>
        <div class="code-block">
[HttpGet("json")]
public IActionResult GetJson()
{
    var user = new
    {
        Id = 1,
        Name = "John Doe",
        Email = "john@example.com"
    };
    return Ok(user);
}
        </div>
    </div>
</body>
</html>
```

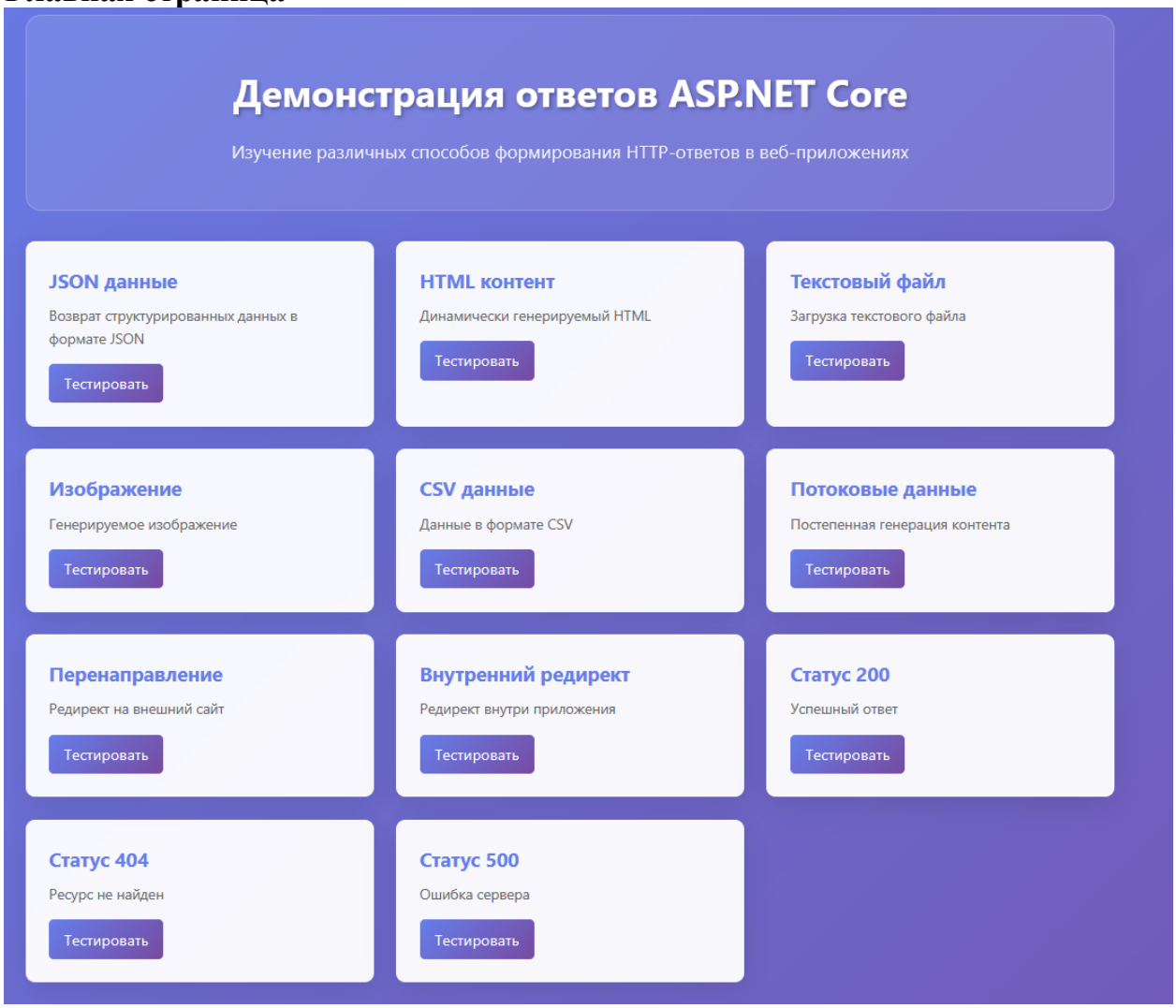
4. Программа (Program.cs)

```
using Microsoft.AspNetCore.Mvc;

var builder = WebApplication.CreateBuilder(args);

// Добавляем сервисы MVC
builder.Services.AddControllersWithViews();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();

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

app.Run();
```

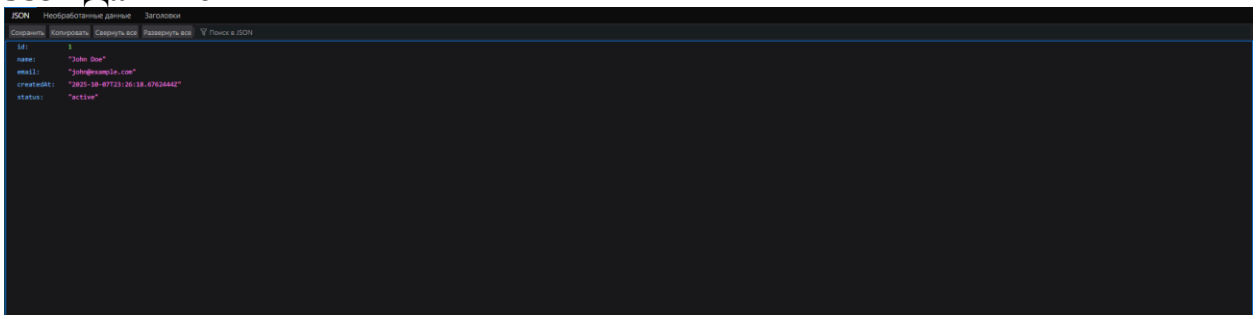
5. Скриншоты и примеры использования


Демонстрация ответов ASP.NET Core
Изучение различных способов формирования HTTP-ответов в веб-приложениях
JSON данные
Возврат структурированных данных в формате JSON
Тестировать
HTML контент
Динамически генерируемый HTML
Тестировать
Текстовый файл
Загрузка текстового файла
Тестировать
Изображение
Генерируемое изображение
Тестировать
CSV данные
Данные в формате CSV
Тестировать
Потоковые данные
Постепенная генерация контента
Тестировать
Перенаправление
Редирект на внешний сайт
Тестировать
Внутренний редирект
Редирект внутри приложения
Тестировать
Статус 200
Успешный ответ
Тестировать
Статус 404
Ресурс не найден
Тестировать
Статус 500
Ошибка сервера
Тестировать

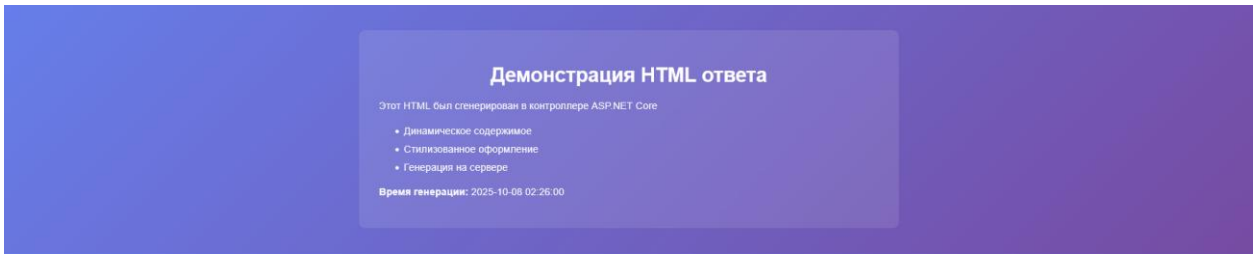
Главная страница



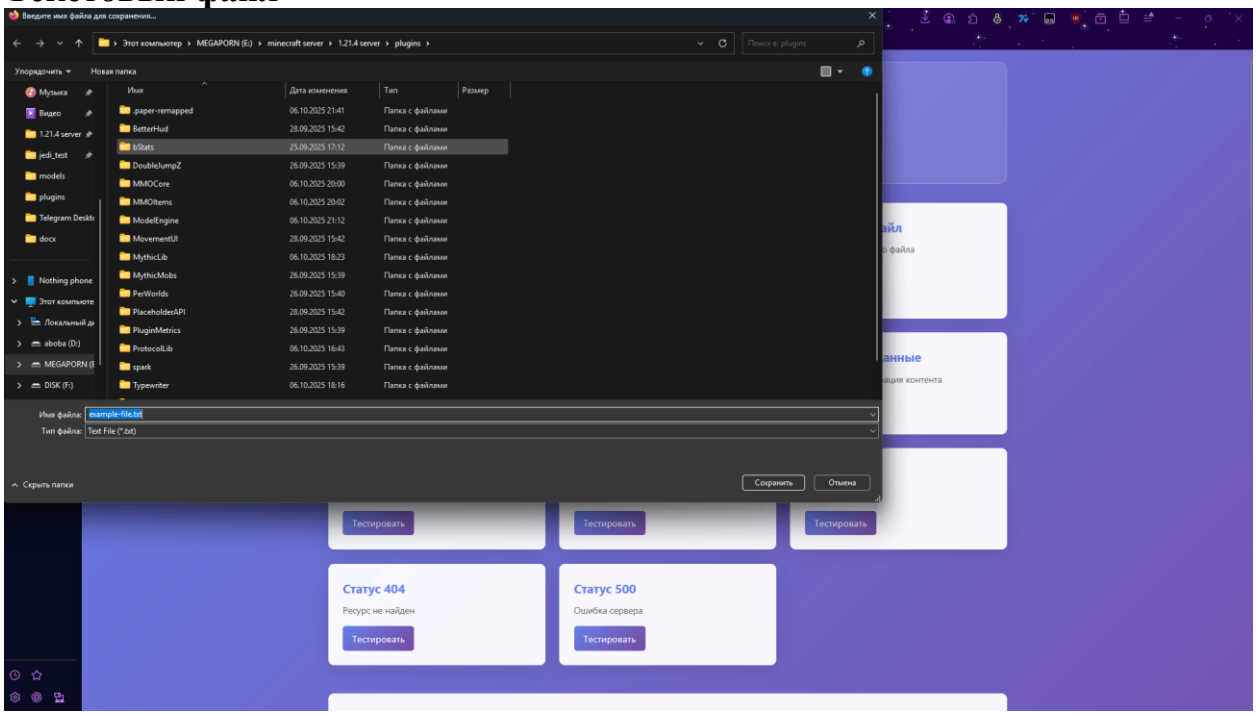
Json Данные



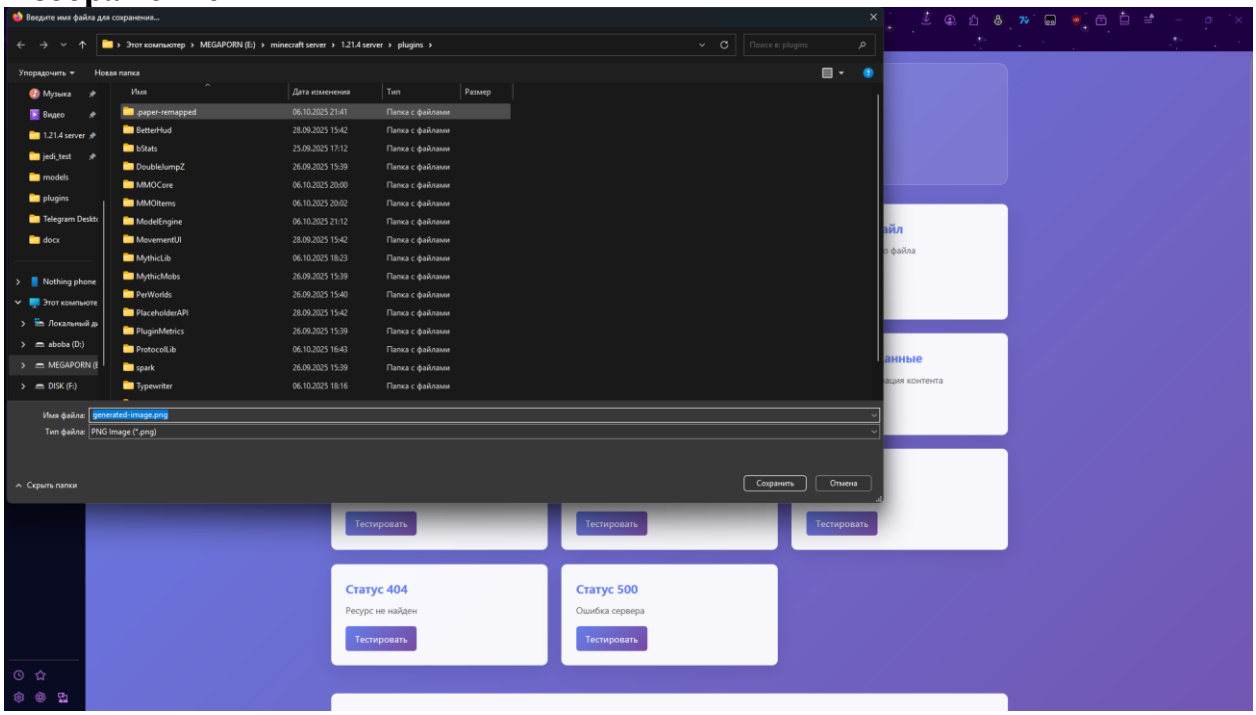
Html-ответ



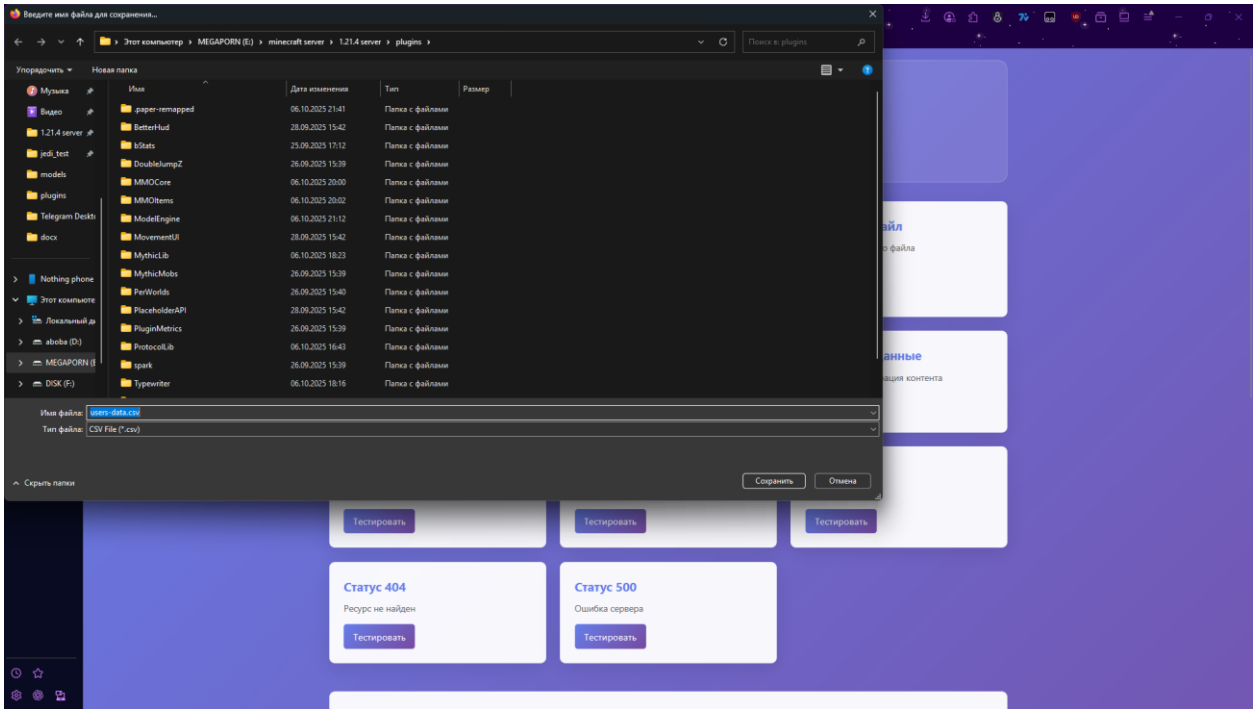
Текстовый файл



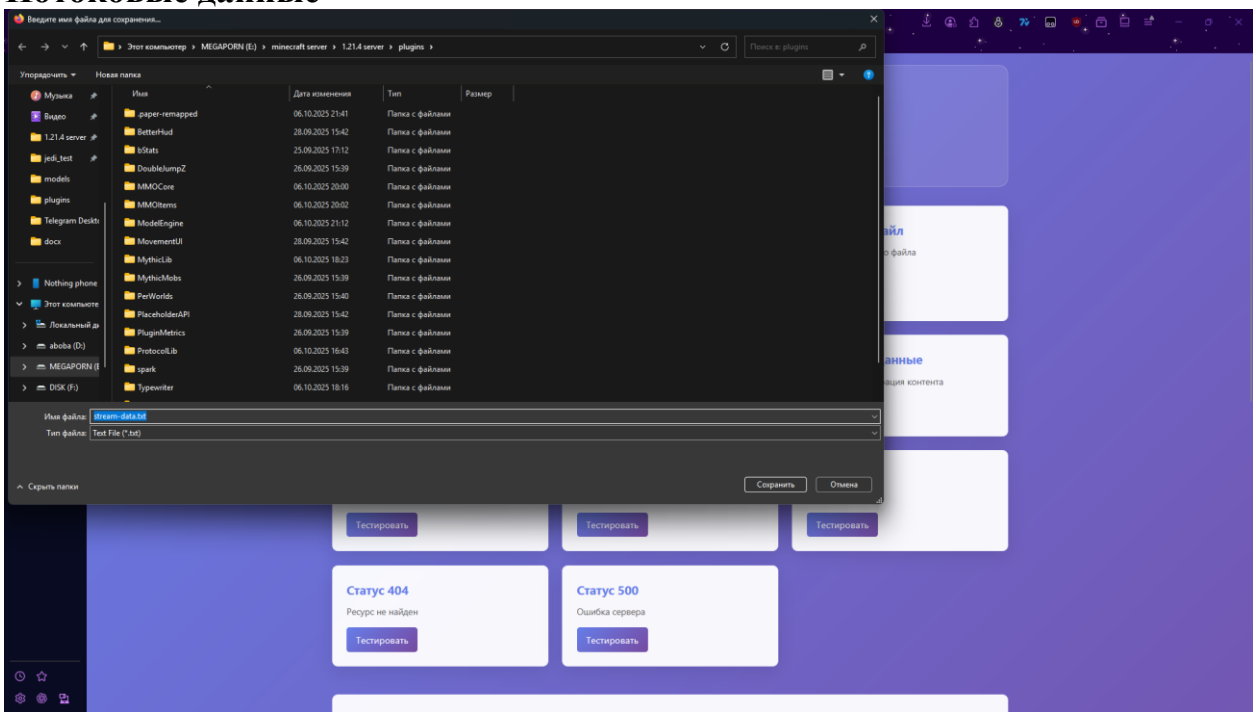
Изображение



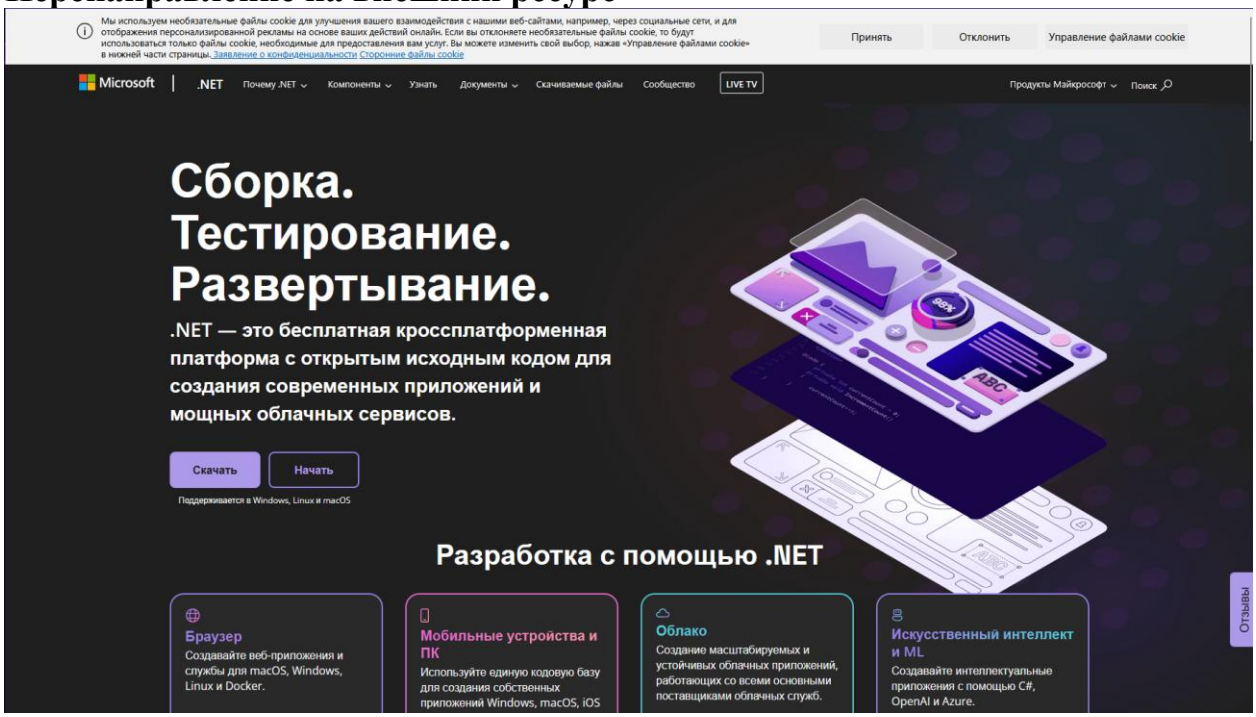
CSV



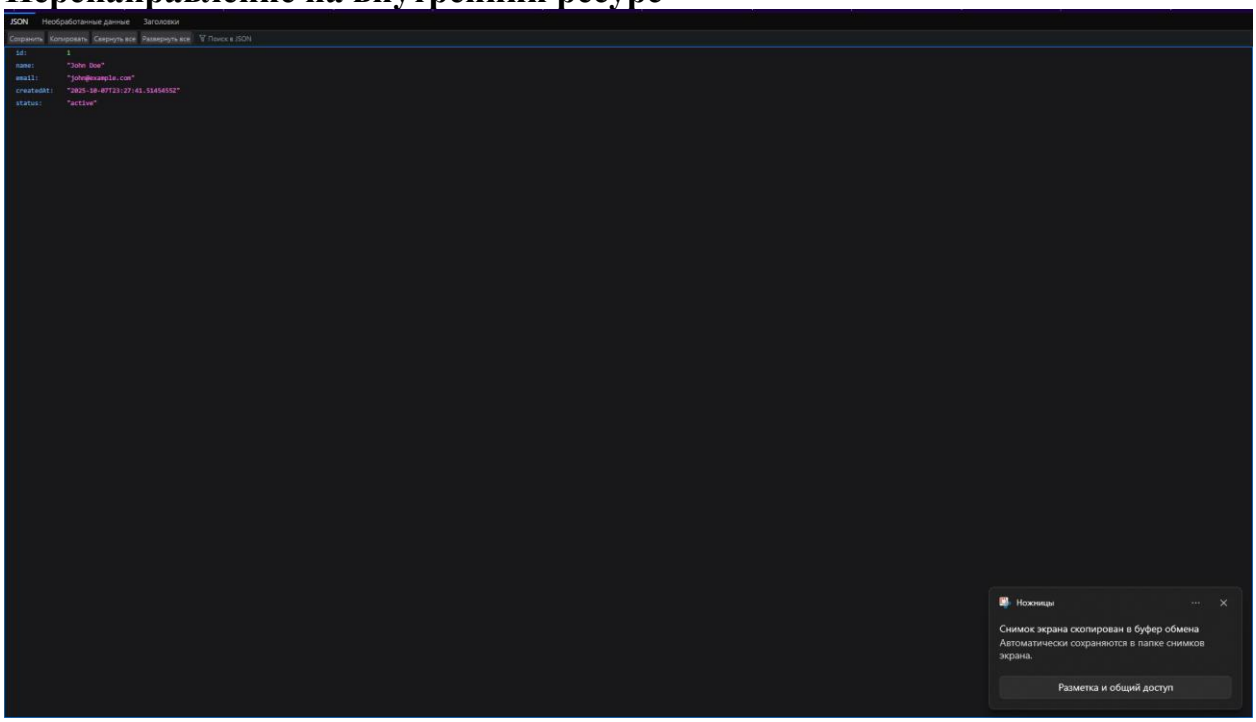
Потоковые данные



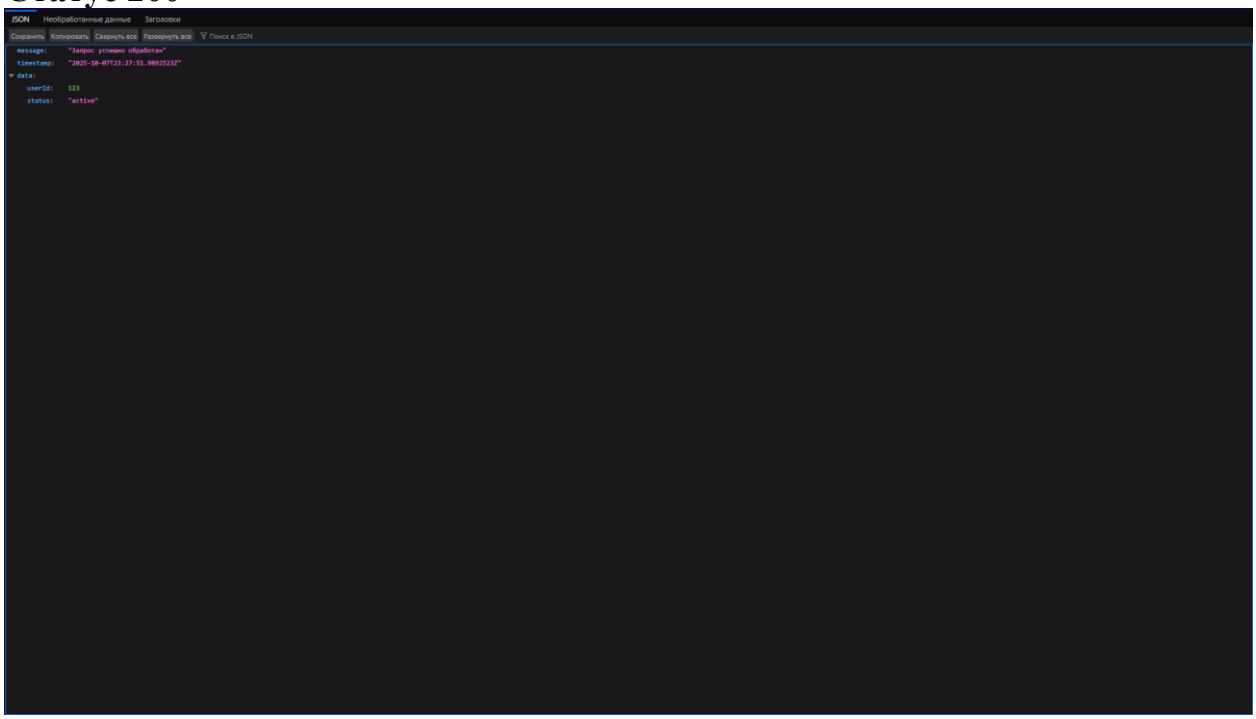
Перенаправление на внешний ресурс



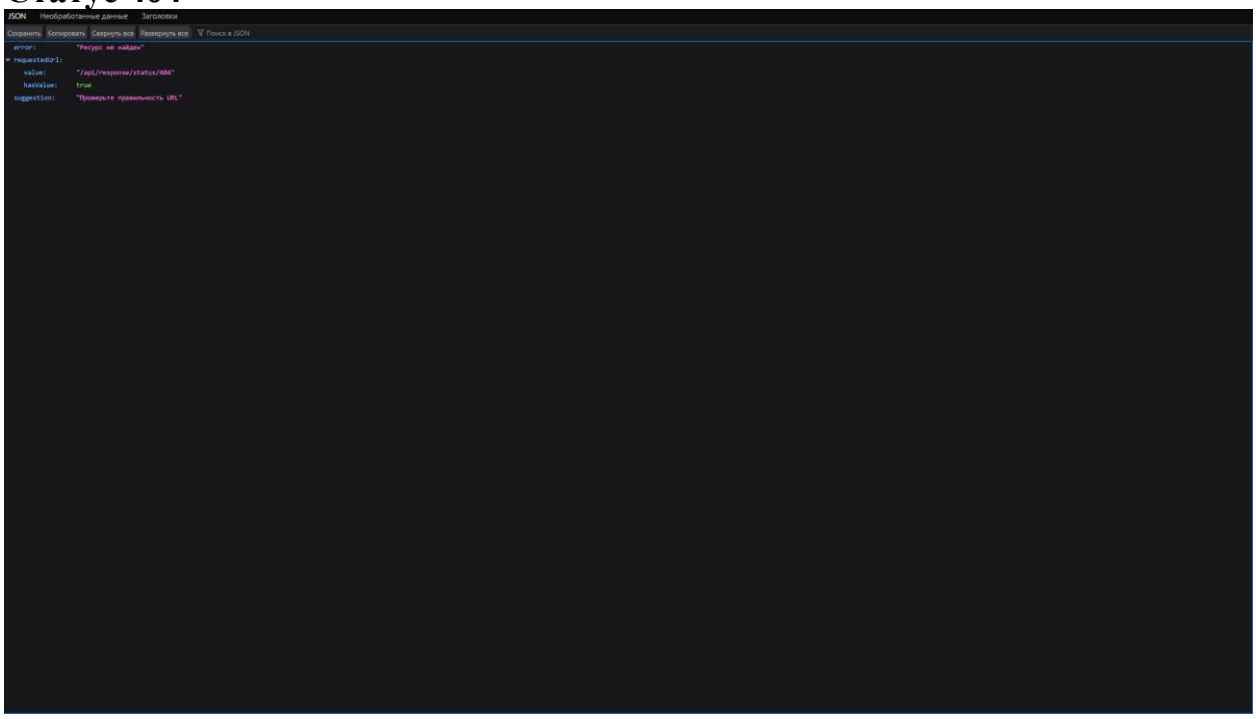
Перенаправление на внутренний ресурс



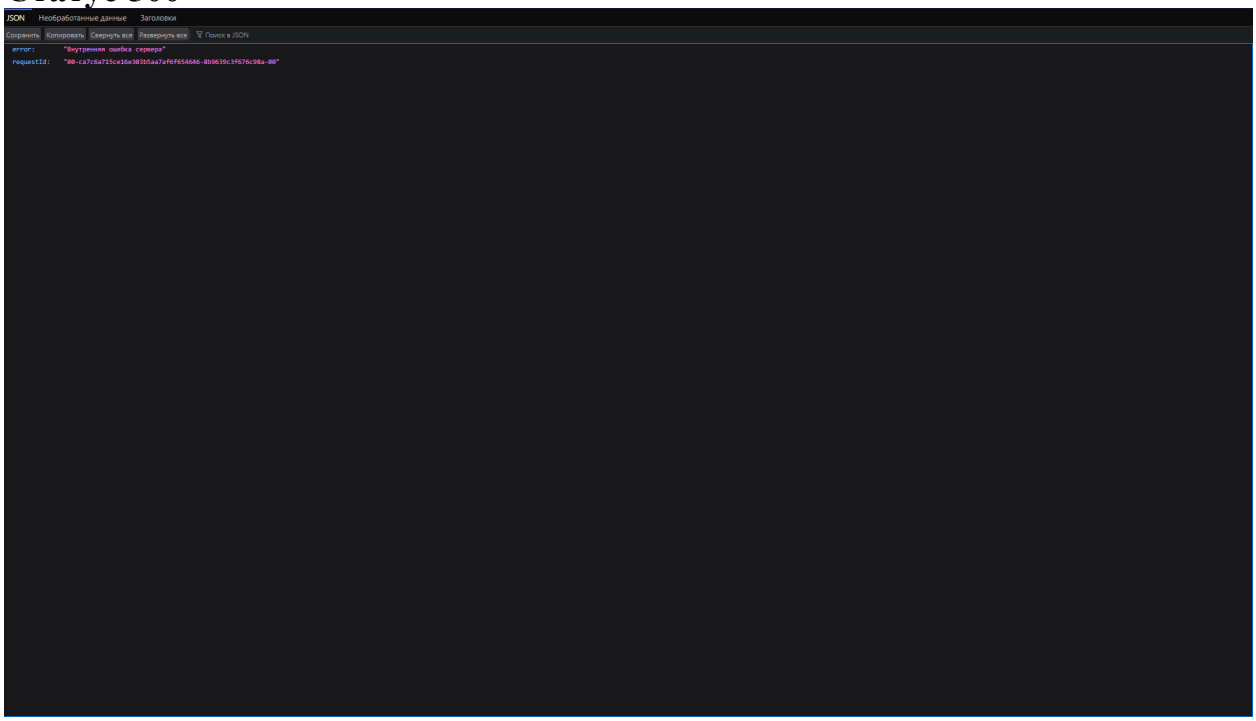
Статус 200



Статус 404



Статус 500



JSON ответ:

```
{
  "id": 1,
  "name": "John Doe",
  "email": "john@example.com",
  "createdAt": "2024-01-15T10:30:45.123Z"
}
```

HTML ответ:

```
<!DOCTYPE html>
<html>
<head>
    <title>HTML Response</title>
</head>
<body>
    <div class='container'>
        <h1>Демонстрация HTML ответа</h1>
        <p>Этот HTML был сгенерирован в контроллере ASP.NET Core</p>
    </div>
</body>
</html>
```

Файловый ответ:

- Загрузка текстового файла "example.txt"
- Загрузка изображения "generated-image.png"
- Загрузка CSV файла "users.csv"

6. Сравнительный анализ преимуществ и ограничений

Тип ответа	Преимущества	Ограничения	Лучшие сценарии использования
JSON	Стандартный формат для API, легковесный, легко парсится	Не подходит для человеко-читаемого контента	REST API, мобильные приложения, SPA

Тип ответа	Преимущества	Ограничения	Лучшие сценарии использования
HTML	Прямое отображение в браузере, богатые возможности стилизации	Тяжелее JSON, сложнее кэшировать	Веб-страницы, порталы, CMS
Файлы	Универсальность, поддержка любых форматов	Требует больше ресурсов, безопасность	Загрузка документов, медиа-файлов
CSV	Простота, совместимость с Excel	Ограниченная структура данных	Экспорт данных, отчеты
Поток	Эффективность для больших данных, начало отдачи без полной загрузки	Сложность обработки ошибок	Большие файлы, реальное время
Редирект	Навигация, SEO, обработка форм	Дополнительный запрос	Аутентификация, POST-Redirect-GET
Статусы	Стандартизация, обработка ошибок	Требует обработки на клиенте	API, обработка ошибок

7. Выводы о наилучших практиках

Рекомендации по использованию:

1. **JSON** - для всех API endpoints, мобильных приложений
2. **HTML** - для серверного рендеринга веб-страниц
3. **Файлы** - для статического контента и загрузок
4. **Потоковые ответы** - для больших данных и реального времени
5. **Редиректы** - после операций изменения состояния (POST)
6. **Статусные коды** - всегда возвращать корректные HTTP статусы

Критические аспекты:

- **Производительность**: JSON и потоковые ответы наиболее эффективны
- **Безопасность**: Валидация всех входных данных, особенно для файлов
- **Кэширование**: Правильная настройка заголовков Cache-Control
- **Обработка ошибок**: Единообразная система статусных кодов и сообщений