Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебный предмет
«Конструирование программ и языки программирования»

**Инструкция**
по выполнению лабораторной работы №38
«Исследование возможностей платформы ASP.NET»

Минск 2025 г.

## Лабораторная работа № 38

**Тема работы:** «Исследование возможностей платформы ASP.NET»

### 1 Цель работы

Сформировать умения использовать возможности платформы ASP.NET для решения профессиональных задач.

### 2 Задание

Номер варианта соответствует номеру по списку в журнале.

1. Реализовать обработку различных маршрутов с помощью делегатов. В классе Startup настройте маршрутизацию. В методе Configure используйте несколько методов Use для обработки различных URL (например, /hello, /goodbye). Для каждого маршрута создайте свой RequestDelegate, который будет возвращать различные сообщения.
2. Создать собственное Middleware для логирования запросов. В классе Startup создайте метод расширения для добавления вашего Middleware. В методе Configure подключите Middleware с помощью метода Use. Реализуйте Middleware, который будет логировать информацию о каждом входящем запросе (метод, URL, время).
3. Создать простое API для управления задачами (To-Do List). В классе Startup настройте необходимые службы (например, для хранения данных в памяти). В методе Configure создайте маршруты для операций CRUD (создание, чтение, обновление, удаление). Используйте делегаты RequestDelegate для обработки каждого из методов API.
4. Реализовать обработку ошибок в приложении. В классе Startup создайте Middleware для обработки исключений. В методе Configure подключите Middleware с помощью метода Use. Реализуйте логику, которая будет возвращать пользовательское сообщение об ошибке при возникновении исключения.
5. Настроить приложение для обслуживания статических файлов. В классе Startup в методе ConfigureServices добавьте необходимые службы для работы с статическими файлами. В методе Configure используйте метод UseStaticFiles. Создайте папку для статических файлов и добавьте туда несколько изображений или CSS-файлов. Убедитесь, что они доступны по URL.
6. Реализовать кэширование ответов для определенных маршрутов. В классе Startup добавьте необходимые службы для кэширования. В методе Configure настройте Middleware для кэширования ответов на определенные запросы. Реализуйте логику, которая будет кэшировать ответы и возвращать их при повторных запросах.
7. Реализовать Middleware, который будет проверять наличие определенного заголовка в запросе. В классе Startup создайте Middleware, который будет проверять наличие заголовка X-Requested-With. Если заголовок отсутствует, возвращайте ответ с кодом 400 (Bad

Request). В методе Configure подключите ваше Middleware с помощью метода Use.

8. Реализовать счетчик запросов, который будет отслеживать количество обращений к приложению. В классе Startup создайте Middleware, который будет увеличивать счетчик при каждом запросе. В методе Configure используйте Run, чтобы возвращать текущее значение счетчика по маршруту /count. Реализуйте метод, который будет сбрасывать счетчик по маршруту /reset.
9. Реализовать API, который возвращает информацию о пользователях. В классе Startup создайте и настройте службы для хранения данных о пользователях (например, в памяти). Используйте RequestDelegate для обработки запросов на получение списка пользователей по маршруту /api/users. Реализуйте возможность получения пользователя по ID по маршруту /api/users/{id}.
10. Настроить CORS (Cross-Origin Resource Sharing) для вашего приложения. В методе ConfigureServices добавьте настройки для CORS, разрешающие доступ с определенных доменов. В методе Configure используйте метод UseCors для применения настроек. Проверьте работу CORS, отправив запрос из другого домена (можно использовать Postman или браузер).

## 3 Оснащение работы

ПК, среда Visual Studio 2019, MSword.

## 4 Основные теоретические сведения

Класс Startup – это основная точка конфигурации приложения. Он состоит из двух основных методов: ConfigureServices и Configure.

**Метод ConfigureServices**

Этот метод используется для регистрации служб, необходимых приложению.

Пример:

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        // Регистрация контекста базы данных
        services.AddDbContext<MyDbContext>(options =>
            options.UseSqlServer("Connection_String_Here"));

        // Регистрация контроллеров
        services.AddControllers();

        // Регистрация кэширования
        services.AddMemoryCache();

        // Регистрация CORS
        services.AddCors(options =>
        {
            options.AddPolicy("AllowAll", builder =>
            {
                builder.AllowAnyOrigin()
```

```
                        .AllowAnyMethod()
                        .AllowAnyHeader();
                });
            });
        }
    }
```

**Метод Configure**

Метод Configure настраивает конвейер обработки HTTP-запросов.

Пример:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage(); // Страница ошибок в разработке
    }
    else
    {
        app.UseExceptionHandler("/Home/Error"); // Общая страница ошибок
        app.UseHsts(); // Защита с помощью HSTS
    }

    app.UseHttpsRedirection(); // Перенаправление на HTTPS
    app.UseStaticFiles(); // Обслуживание статических файлов

    app.UseRouting(); // Включение маршрутизации

    app.UseCors("AllowAll"); // Применение CORS политики

    app.UseAuthorization(); // Включение авторизации

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers(); // Маршрутизация контроллеров
    });
}
```

**Middleware**

Middleware – это компонент, который обрабатывает HTTP-запросы и ответы. Они могут выполнять различные задачи, такие как логирование, аутентификация и обработка ошибок.

Методы Use и Run

Метод Use добавляет middleware в конвейер.

Метод Run завершает конвейер и обрабатывает запрос.

Пример:

```
app.Use(async (context, next) =>
{
    // Логирование запроса
    Console.WriteLine($"Request: {context.Request.Method}
{context.Request.Path}");
    await next.Invoke(); // Передача управления следующему middleware
    // Логирование ответа
    Console.WriteLine($"Response: {context.Response.StatusCode}");
});

// Завершение конвейера
app.Run(async context =>
{
    await context.Response.WriteAsync("Hello, World!");
});
```

**RequestDelegate**

RequestDelegate — это делегат для обработки HTTP-запросов.

Пример:

```
public delegate Task RequestDelegate(HttpContext context);

public class Startup
{
    // Пример использования RequestDelegate
    public void Configure(IApplicationBuilder app)
    {
        RequestDelegate myDelegate = async context =>
        {
            await context.Response.WriteAsync("Hello from
RequestDelegate!");
        };

        app.Run(myDelegate);
    }
}
```

**Конфигурация CORS**

CORS (Cross-Origin Resource Sharing) позволяет контролировать доступ к ресурсам на сервере из других доменов.

Пример:

```
services.AddCors(options =>
{
    options.AddPolicy("AllowSpecificOrigin", builder =>
    {
        builder.WithOrigins("https://example.com")
               .AllowAnyMethod()
               .AllowAnyHeader();
    });
});
```

**Логирование**

Логирование позволяет отслеживать события в приложении. ASP.NET Core предоставляет встроенные механизмы логирования.

Пример:

```
public void Configure(IApplicationBuilder app, ILogger<Startup> logger)
{
    app.Use(async (context, next) =>
    {
        logger.LogInformation($"Request: {context.Request.Method}
{context.Request.Path}");
        await next.Invoke();
        logger.LogInformation($"Response:
{context.Response.StatusCode}");
    });
}
```

**Кэширование**

Кэширование помогает улучшить производительность, сохраняя часто запрашиваемые данные.

Пример:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMemoryCache();
}

public void Configure(IApplicationBuilder app, IMemoryCache cache)
{
```

```
        app.Run(async context =>
        {
            string cachedValue;
            if (!cache.TryGetValue("key", out cachedValue))
            {
                cachedValue = "Cached Value";
                cache.Set("key", cachedValue, TimeSpan.FromMinutes(5)); //
Кэширование на 5 минут
            }
            await context.Response.WriteAsync(cachedValue);
        });
    }
```

Обработка ошибок

Вы можете создать middleware для обработки ошибок и возвращения соответствующих ответов.

Пример:

```
    app.Use(async (context, next) =>
    {
        try
        {
            await next.Invoke();
        }
        catch (Exception ex)
        {
            context.Response.StatusCode = 500;
            await context.Response.WriteAsync("An error occurred: " +
ex.Message);
        }
    });
```

Аутентификация и авторизация

ASP.NET Core поддерживает различные методы аутентификации, такие как JWT и ASP.NET Identity.

Пример:

```
    services.AddAuthentication("MyScheme")
        .AddJwtBearer("MyScheme", options =>
        {
            options.TokenValidationParameters = new
TokenValidationParameters
            {
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                // Настройки валидации
            };
        });
```

Паттерны проектирования

Dependency Injection (DI) позволяет управлять зависимостями и улучшает тестируемость кода.

Пример:

```
    csharp

    Copy
    public interface IMyService
    {
        string GetData();
    }

    public class MyService : IMyService
```

```
{
    public string GetData() => "Data from MyService";
}

public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IMyService, MyService>();
}

// Использование в контроллере
public class MyController : ControllerBase
{
    private readonly IMyService _myService;

    public MyController(IMyService myService)
    {
        _myService = myService;
    }

    public IActionResult GetData()
    {
        return Ok(_myService.GetData());
    }
}
```

**5. Порядок выполнения работы**

1. Выделить ключевые моменты задачи.
2. Построить алгоритм и теоретическую объектную модель решения задачи.
3. Запрограммировать полученные алгоритмы и объектную модель.

**6. Форма отчета о работе**

*Лабораторная работа № ___*

*Номер учебной группы* _____

*Фамилия, инициалы учащегося* _____

*Дата выполнения работы* _____

*Тема работы:* _____

*Цель работы:* _____

*Оснащение работы:* _____

*Результат выполнения работы:* _____

**7. Контрольные вопросы и задания**

1. Что такое класс Startup в ASP.NET и какую роль он играет в конфигурации приложения?
2. Опишите различия между методами ConfigureServices и Configure. Для чего они используются?
3. Как работает middleware в ASP.NET? Приведите пример его использования.
4. Что такое RequestDelegate, и как он связан с обработкой HTTP-запросов?

5. Как настроить CORS в ASP.NET, и почему это важно для веб-приложений?

6. Какие методы логирования доступны в ASP.NET Core? Как вы можете настроить логирование в приложении?

## 8. Рекомендуемая литература


**1. Рихтер, Дж**. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. СПб. : Изд-во Питер, 2021. 896 с.

**2. Прайс, М. Дж**. C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Дж. Прайс. СПб : Изд-во Питер, 2023. 848 с.

**3. Васильев, А.Н.** Программирование на C# для начинающих. Особенности языка / А.Н. Васильев. М. : Эксмо, 2022. 528 с.

**4. Фримен, А**. ASP.NET Core 3 с примерами на C# для профессионалов / А. Фримен. СПб. : Изд-во Вильямс, 2021. 1184 с.