



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки

09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 7

Дисциплина: «Backend»

Тема: *Логирование веб-приложения на основе ASP.NET Core*

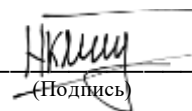
Выполнил: студент группы: 231-339

Карапетян Нвер Каренович

(Фамилия И.О.)

Дата, подпись: 16.04.25

(Дата)


(Подпись)

Проверил: _____

(Фамилия И.О., степень, звание)

(Оценка)

Дата, подпись _____

(Дата)

(Подпись)

Москва
2025

Цель:

Ознакомиться с методами логирования и их применением в веб-приложениях на платформе ASP.NET Core.

Задачи:

- Настроить логирование для веб-приложения на ASP.NET Core, используя различные провайдеры (например, Console, файлы, базы данных).
- Применить различные уровни логирования для различных компонентов приложения (например, информационные, отладочные, ошибки).
- Реализовать обработчики логов и пользовательские форматы сообщений при необходимости.
- Протестировать работу приложения, записывая и анализируя созданные логи.

Ход работы

Настройка логирования в ASP.NET Core

В файле `Program.cs` происходит первоначальная настройка логирования. Сначала удаляются стандартные провайдеры, затем добавляются провайдеры для вывода логов в консоль и отладчик. Для расширенных возможностей используется библиотека Serilog, которая позволяет настраивать вывод в текстовые файлы и в структурированном формате JSON.

Листинг 1. Настройка логирования в `Program.cs`.

```
// Удаление стандартных провайдеров логирования
builder.Logging.ClearProviders();

// Добавление провайдеров для консоли и отладчика
builder.Logging.AddConsole();
builder.Logging.AddDebug();

// Настройка Serilog для логирования с использованием различных провайдеров
Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Debug() // Устанавливаем минимальный уровень логирования
    .WriteTo.Console() // Логирование в консоль
    .WriteTo.File("Logs/log-.txt", rollingInterval: RollingInterval.Day) // Логиро-
// вание в обычный текстовый файл
```

```

        .WriteTo.File(new Serilog.Formatting.Json.JsonFormatter(), "Logs/structured-
.json", rollingInterval: RollingInterval.Day) // Логирование в формате JSON
        .CreateLogger();

// Интеграция Serilog в приложение
builder.Host.UseSerilog();

```

Реализация логирования в контроллере

В контроллере веб-приложения логирование используется для фиксации вызовов методов, регистрации успешных операций и обработки ошибок. Для этого логгер внедряется через **Dependency Injection** (`ILogger<T>`).

Листинг 2. Контроллер `AuthorizationController`.

```

[Route("api/[controller]")]
[ApiController]
public class AuthorizationController : ControllerBase
{
    private readonly IAuthService _authService;
    private readonly ILogger<AuthorizationController> _logger;

    public AuthorizationController(IAuthService authService, ILogger<Authoriza-
tionController> logger)
    {
        _authService = authService;
        _logger = logger;
    }

    [HttpPost("register")]
    public async Task<ActionResult<User>> Register(UserDto request)
    {
        _logger.LogInformation("Метод Register вызван для пользователя
{Username}", request.Username);
        var user = await _authService.RegisterAsync(request);
        if (user is null)
        {
            _logger.LogWarning("Попытка регистрации неуспешна. Пользователь с
именем {Username} уже существует", request.Username);
            return BadRequest("Username already exists!");
        }
        _logger.LogInformation("Пользователь {Username} успешно зарегистриро-
ван", request.Username);
        return Ok(user);
    }

    [HttpPost("login")]
    public async Task<ActionResult<TokenResponseDto>> Login(UserDto request)
    {

```

```

        _logger.LogDebug("Метод Login вызван для пользователя {Username}", request.Username);
        var response = await _authService.LoginAsync(request);
        if (response is null)
        {
            _logger.LogWarning("Неуспешная попытка входа пользователя {Username}. Неверное имя или пароль", request.Username);
            return BadRequest("Invalid username and/or password!");
        }
        _logger.LogInformation("Пользователь {Username} успешно вошёл в систему", request.Username);
        return Ok(response);
    }

    [HttpGet]
    [Authorize]
    public IActionResult AuthenticatedOnlyEndpoint()
    {
        _logger.LogDebug("Запрошен защищённый эндпоинт для аутентифицированного пользователя");
        return Ok("You are authenticated!");
    }

    [HttpGet("admin-only")]
    [Authorize(Roles = "Admin")]
    public IActionResult AdminOnlyEndpoint()
    {
        _logger.LogDebug("Запрошен эндпоинт администратора");
        return Ok("You are admin!");
    }

    [HttpPost("refresh-token")]
    public async Task<ActionResult<TokenResponseDto>> RefreshToken(RefreshTokenRequestDto request)
    {
        _logger.LogDebug("Метод RefreshToken вызван для пользователя с ID: {UserId}", request.UserId);
        var response = await _authService.RefreshTokensAsync(request);
        if (response is null || response.AccessToken is null || response.RefreshToken is null)
        {
            _logger.LogError("Ошибка при обновлении токенов для пользователя с ID: {UserId}", request.UserId);
            return Unauthorized("InvalidRefreshToken");
        }
        _logger.LogInformation("Токены для пользователя с ID: {UserId} успешно обновлены", request.UserId);
        return Ok(response);
    }
}

```

В данном контроллере использованы следующие уровни логирования:

- **LogDebug** — для начальных сообщений при входе в метод, что помогает отслеживать последовательность вызовов.
- **LogInformation** — для регистрации успешных операций.
- **LogWarning** — для фиксации неудачных попыток, например, при регистрации или входе с ошибочными данными.
- **LogError** — для регистрации критических ошибок, которые требуют внимания.

Тестирование и анализ логов

При тестировании приложения была произведена проверка корректности работы настроенной системы логирования путем запуска самого приложения и выполнения последовательности запросов к API. При старте приложения автоматически создавался каталог «Logs», в который записывались как текстовые логи, так и файлы в структурированном формате JSON. Это позволяло наблюдать за ходом работы в реальном времени и обеспечивало возможность последующего анализа записанных данных.

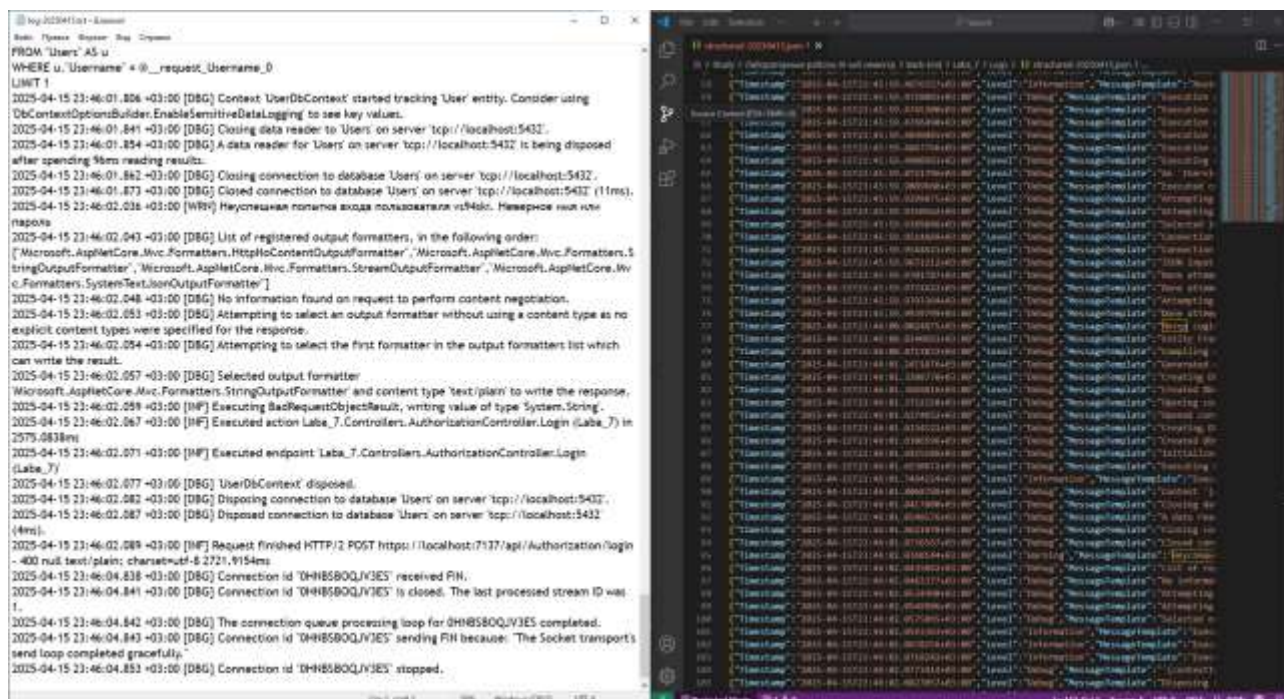


Рисунок 1. Логи в текстовом документе и в JSON-файле.

В ходе работы с приложением были выполнены такие действия, как регистрация нового пользователя, вход в систему, обновление токенов и обращение к защищённым эндпоинтам. При выполнении этих операций генерировались сообщения различных уровней — от отладочных до информационных, предупреждающих и критических. Все сообщения регистрировались как в консоли, так и в лог-файлах, что позволяло получить всестороннюю информацию о процессе обработки запросов.

Рисунок 2. Логи в консоли.

Проведенный анализ логов показал, что система корректно определяет уровни логирования для каждой операции. Например, при попытке входа с некорректными данными система фиксировала сообщение уровня Warning, а в случае критических сбоев регистрировались сообщения уровня Error.