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 № _1_

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)

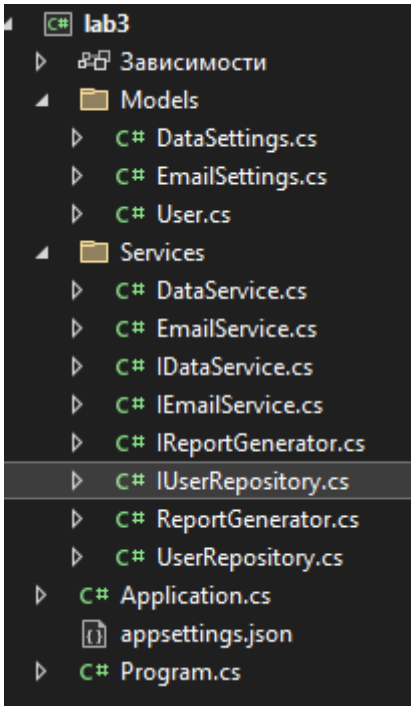
Замечания:

Москва

2025

Шаг 1: Создание проекта

Создан новый проект Console App с использованием .NET 6 и настроена конфигурация для внедрения зависимостей.



Шаг 2: Настройка зависимостей

В файле `Program.cs` настроены:

- Сервисы приложения (Transient, Scoped, Singleton)
- Конфигурация приложения
- Логирование

Шаг 3: Реализация сервисов

Созданы интерфейсы и их реализации:

- `IDataService` - сервис работы с данными
- `IEmailService` - сервис отправки email
- `IUserRepository` - репозиторий пользователей
- `IReportGenerator` - генератор отчетов

Шаг 4: Реализация основного приложения

Создан класс `Application` который использует все внедренные зависимости.

```
info: DataService[0]
      DataService создан
info: EmailService[0]
      EmailService создан с SMTP сервером: smtp.example.com
info: UserRepository[0]
      UserRepository создан с 3 пользователями
info: ReportGenerator[0]
      ReportGenerator создан с типом: Подробный отчет
info: Application[0]
      Application создан с внедренными зависимостями
info: Application[0]
      Запуск приложения
=== КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ С DI ===

1. ОПЕРАЦИИ С ПОЛЬЗОВАТЕЛЯМИ
============================
info: UserRepository[0]
      Получение всех пользователей
Найдено пользователей: 3
info: UserRepository[0]
      Добавление пользователя: Алексей Новый
Добавлен пользователь: Алексей Новый
info: UserRepository[0]
      Поиск пользователя с ID: 1
Найден пользователь: User[Id=1, Name=Иван Иванов, Email=ivan@example.com, Age=30]

2. ОПЕРАЦИИ С ДАННЫМИ
======================
info: DataService[0]
      Получение данных из DataService
Получено элементов данных: 5
info: DataService[0]
      Обработка данных: тестовые данные
Обработанные данные: Обработанные: ТЕСТОВЫЕ ДАННЫЕ
info: DataService[0]
      Сохранение данных: Новые данные из консоли
Данные сохранены

3. ОПЕРАЦИИ С EMAIL
===================
info: EmailService[0]
      Валидация email: test@example.com
Email валиден: True
info: EmailService[0]
      Отправка email пользователю: user@example.com, Тема: Тестовое сообщение
info: EmailService[0]
      Email успешно отправлен
Email отправлен: True

4. ГЕНЕРАЦИЯ ОТЧЕТОВ
====================
info: UserRepository[0]
      Получение всех пользователей
info: DataService[0]
```

```
info: DataService[0]
      Получение данных из DataService
info: ReportGenerator[0]
      Генерация отчета для 4 пользователей
info: ReportGenerator[0]
      Генерация отчета для 6 элементов данных
Отчет о пользователях:
=== ОТЧЕТ О ПОЛЬЗОВАТЕЛЯХ ===
Дата генерации: 07.10.2025 21:10:13
Количество пользователей: 4

ID: 1
Имя: Иван Иванов
Email: ivan@example.com
Возраст: 30
---
ID: 2
Имя: Петр Петров
Email: petr@example.com
Возраст: 25
---
ID: 3
Имя: Мария Сидорова
Email: maria@example.com
Возраст: 28
---
ID: 4
Имя: Алексей Новый
Email: alexey@example.com
Возраст: 35
---
=== КОНЕЦ ОТЧЕТА ===

Отчет о данных:
=== ОТЧЕТ О ДАННЫХ ===
Дата генерации: 07.10.2025 21:10:14
Количество элементов: 6

1. Данные 1
2. Данные 2
3. Данные 3
4. Данные 4
5. Данные 5
6. Новые данные из консоли
=== КОНЕЦ ОТЧЕТА ===
```

Код приложения

DataSettings.cs

```
public class DataSettings
{
```

```
    public string ConnectionString { get; set; } =
"Server=localhost;Database=TestDB;";
    public int TimeoutSeconds { get; set; } = 30;
}
```

EmailSettings.cs

```
public class EmailSettings
{
    public string SmtpServer { get; set; } = "smtp.example.com";
    public int Port { get; set; } = 587;
    public string Username { get; set; } = "username";
    public string Password { get; set; } = "password";
}
```

User.cs

```
public class User
{
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
    public string Email { get; set; } = string.Empty;
    public int Age { get; set; }

    public override string ToString()
    {
        return $"User[Id={Id}, Name={Name}, Email={Email}, Age={Age}]";
    }
}
```

DataService.cs

```
using Microsoft.Extensions.Logging;

public class DataService : IDataService
{
    private readonly ILogger<DataService> _logger;
    private readonly List<string> _mockData;

    public DataService(ILogger<DataService> logger)
    {
        _logger = logger;
        _mockData = new List<string>
        {
            "Данные 1", "Данные 2", "Данные 3", "Данные 4", "Данные 5"
        };

        _logger.LogInformation("DataService создан");
    }

    public async Task<List<string>> GetDataAsync()
    {
        _logger.LogInformation("Получение данных из DataService");
        await Task.Delay(100); // Имитация асинхронной операции
        return _mockData;
    }

    public async Task<bool> SaveDataAsync(string data)
    {
        _logger.LogInformation("Сохранение данных: {Data}", data);
        await Task.Delay(50);
        _mockData.Add(data);
        return true;
    }

    public async Task<string> ProcessDataAsync(string input)
    {
        _logger.LogInformation("Обработка данных: {Input}", input);
        await Task.Delay(80);
        return $"Обработанные: {input.ToUpper()}";
    }
```

```
}
```

EmailService.cs

```
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;

public class EmailService : IEmailService
{
    private readonly ILogger<EmailService> _logger;
    private readonly EmailSettings _settings;

    public EmailService(ILogger<EmailService> logger, IOptions<EmailSettings>
options)
    {
        _logger = logger;
        _settings = options.Value;
        _logger.LogInformation("EmailService создан с SMTP сервером: {SmtpServer}",
_settings.SmtpServer);
    }

    public async Task<bool> SendEmailAsync(string to, string subject, string body)
    {
        _logger.LogInformation("Отправка email пользователю: {To}, Тема: {Subject}",
to, subject);

        // Имитация отправки email
        await Task.Delay(200);

        var success = !string.IsNullOrEmpty(to) && to.Contains("@");
        if (success)
        {
            _logger.LogInformation("Email успешно отправлен");
        }
        else
        {
            _logger.LogWarning("Не удалось отправить email");
        }

        return success;
    }

    public async Task<bool> ValidateEmailAsync(string email)
    {
        _logger.LogInformation("Валидация email: {Email}", email);
        await Task.Delay(50);

        return !string.IsNullOrEmpty(email) &&
               email.Contains("@") &&
               email.Contains(".");
    }
}
```

ReportGenerator.cs

```
using Microsoft.Extensions.Logging;
using System.Text;

public class ReportGenerator : IReportGenerator
{
    private readonly ILogger<ReportGenerator> _logger;
    public string ReportType => "Подробный отчет";

    public ReportGenerator(ILogger<ReportGenerator> logger)
    {
        _logger = logger;
```

```
        _logger.LogInformation("ReportGenerator создан с типом: {ReportType}",
ReportType);
    }

    public async Task<string> GenerateUserReportAsync(List<User> users)
    {
        _logger.LogInformation("Генерация отчета для {Count} пользователей",
users.Count);
        await Task.Delay(300);

        var report = new StringBuilder();
        report.AppendLine("=== ОТЧЕТ О ПОЛЬЗОВАТЕЛЯХ ===");
        report.AppendLine($"Дата генерации: {DateTime.Now}");
        report.AppendLine($"Количество пользователей: {users.Count}");
        report.AppendLine();

        foreach (var user in users)
        {
            report.AppendLine($"ID: {user.Id}");
            report.AppendLine($"Имя: {user.Name}");
            report.AppendLine($"Email: {user.Email}");
            report.AppendLine($"Возраст: {user.Age}");
            report.AppendLine("---");
        }

        report.AppendLine("=== КОНЕЦ ОТЧЕТА ===");

        return report.ToString();
    }

    public async Task<string> GenerateDataReportAsync(List<string> data)
    {
        _logger.LogInformation("Генерация отчета для {Count} элементов данных",
data.Count);
        await Task.Delay(200);

        var report = new StringBuilder();
        report.AppendLine("=== ОТЧЕТ О ДАННЫХ ===");
        report.AppendLine($"Дата генерации: {DateTime.Now}");
        report.AppendLine($"Количество элементов: {data.Count}");
        report.AppendLine();

        for (int i = 0; i < data.Count; i++)
        {
            report.AppendLine($"{i + 1}. {data[i]}");
        }

        report.AppendLine("=== КОНЕЦ ОТЧЕТА ===");

        return report.ToString();
    }
}
```

UserRepository.cs

```
using Microsoft.Extensions.Logging;

public class UserRepository : IUserRepository
{
    private readonly ILogger<UserRepository> _logger;
    private readonly List<User> _users;

    public UserRepository(ILogger<UserRepository> logger)
    {
        _logger = logger;
        _users = new List<User>
        {
```

```
            new User { Id = 1, Name = "Иван Иванов", Email = "ivan@example.com", Age
= 30 },
            new User { Id = 2, Name = "Петр Петров", Email = "petr@example.com", Age
= 25 },
            new User { Id = 3, Name = "Мария Сидорова", Email = "maria@example.com",
Age = 28 }
        };

        _logger.LogInformation("UserRepository создан с {Count} пользователями",
_users.Count);
    }

    public async Task<User> GetUserByIdAsync(int id)
    {
        _logger.LogInformation("Поиск пользователя с ID: {Id}", id);
        await Task.Delay(100);

        return _users.FirstOrDefault(u => u.Id == id);
    }

    public async Task<List<User>> GetAllUsersAsync()
    {
        _logger.LogInformation("Получение всех пользователей");
        await Task.Delay(150);

        return _users;
    }

    public async Task<bool> AddUserAsync(User user)
    {
        _logger.LogInformation("Добавление пользователя: {Name}", user.Name);
        await Task.Delay(80);

        user.Id = _users.Max(u => u.Id) + 1;
        _users.Add(user);
        return true;
    }

    public async Task<bool> UpdateUserAsync(User user)
    {
        _logger.LogInformation("Обновление пользователя с ID: {Id}", user.Id);
        await Task.Delay(80);

        var existingUser = _users.FirstOrDefault(u => u.Id == user.Id);
        if (existingUser != null)
        {
            existingUser.Name = user.Name;
            existingUser.Email = user.Email;
            existingUser.Age = user.Age;
            return true;
        }

        return false;
    }

    public async Task<bool> DeleteUserAsync(int id)
    {
        _logger.LogInformation("Удаление пользователя с ID: {Id}", id);
        await Task.Delay(80);

        var user = _users.FirstOrDefault(u => u.Id == id);
        if (user != null)
        {
            _users.Remove(user);
            return true;
```

```
        }

        return false;
    }
}
```

IDataService.cs

```
public interface IDataService
{
    Task<List<string>> GetDataAsync();
    Task<bool> SaveDataAsync(string data);
    Task<string> ProcessDataAsync(string input);
}
```

IEmailService.cs

```
public interface IEmailService
{
    Task<bool> SendEmailAsync(string to, string subject, string body);
    Task<bool> ValidateEmailAsync(string email);
}
```

IReportGenerator.cs

```
public interface IReportGenerator
{
    Task<string> GenerateUserReportAsync(List<User> users);
    Task<string> GenerateDataReportAsync(List<string> data);
    string ReportType { get; }
}
```

IUserRepository.cs

```
public interface IUserRepository
{
    Task<User> GetUserByIdAsync(int id);
    Task<List<User>> GetAllUsersAsync();
    Task<bool> AddUserAsync(User user);
    Task<bool> UpdateUserAsync(User user);
    Task<bool> DeleteUserAsync(int id);
}
```