**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 №3**

**Дисциплина:** BackEnd-разработка

**Тема:** Создание консольного приложения с внедренными зависимостями на основе ASP.NET Core

**Выполнил: студент группы 231-332**

**Чесноков Александр Евгеньевич**

(Фамилия И.О. обучающегося)

**Дата, подпись** 01.10.2025 

(Дата)(Подпись)

**Проверил:** \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_*

(Фамилия И.О., степень, звание) (Оценка)

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_ \_\_\_\_\_\_\_\_*

(Дата)(Подпись)

**Москва**

**2025**

**Цель:**

Освоить создание консольного приложения, использующего внедрение зависимостей в ASP.NET Core.

**Ход работы:**

Для реализации задачи был создан новый проект консольного приложения, в котором применён механизм регистрации сервисов через контейнер зависимостей.

Первым этапом было создание проекта консольного типа в VS Code. Далее в файл Program.cs был добавлен код, создающий хост с помощью класса Host и настраивающий коллекцию служб методом ConfigureServices. В данной лабораторной был зарегистрирован интерфейс IMessageService и его реализация MessageService, а также основной сервис приложения ApplicationService. Благодаря этому ASP.NET Core автоматически внедряет необходимые зависимости при создании экземпляров классов.

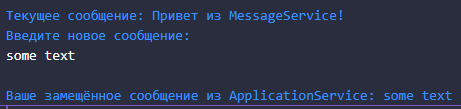
Интерфейс IMessageService определяет базовые операции с сообщениями - получение и сохранение строки. Класс MessageService реализует данный интерфейс и хранит сообщение в приватном поле. При вызове методов GetMessage и SaveMessage осуществляется доступ к текущему состоянию сообщения, что демонстрирует работу внедрённой зависимости.

Класс ApplicationService отвечает за основную логику приложения. В его конструкторе принимается IMessageService, внедрённый через DI-контейнер. В методе Run() приложение выводит текущее сообщение, запрашивает новое значение у пользователя через консоль и сохраняет его в сервисе. После этого результат выводится повторно, что подтверждает корректную работу внедрённой зависимости.

В результате выполнения программы можно наблюдать, как зависимости между классами настраиваются автоматически без явного создания экземпляров через оператор new. Это позволяет улучшить модульность и тестируемость кода. Программа успешно демонстрирует принципы инверсии управления и внедрения зависимостей в контексте консольного приложения на ASP.NET Core.

Таким образом, цель лабораторной работы достигнута — создано консольное приложение с внедрением зависимостей, реализованы интерфейсы и сервисы, произведена регистрация и тестирование работы в консольном режиме.

**Скриншоты работы приложения:**

****

**Листинг:**

**Program.cs:**

|  |
| --- |
| using Microsoft.Extensions.DependencyInjection;  using Microsoft.Extensions.Hosting;  namespace Lab1;  /// *Главный класс программы, настраивающий внедрение зависимостей и запускающий приложение*  class Program  {      static void Main(string[] args)      {          // *Создание хоста с настройкой служб*          var host = Host.CreateDefaultBuilder(args)              .ConfigureServices((context, services) =>              {                  // *Регистрация зависимостей*                  services.AddSingleton<IMessageService, MessageService>();                  services.AddSingleton<ApplicationService>();              })              .Build();          // *Получаем основной сервис приложения*          var app = host.Services.GetRequiredService<ApplicationService>();          app.Run();      }  } |

**IMessageService.cs:**

|  |
| --- |
| namespace Lab1;  /// *Интерфейс сервиса для работы с сообщениями*  public interface IMessageService  {      string GetMessage();      void SaveMessage(string message);  } |

**MessageService.cs:**

|  |
| --- |
| namespace Lab1;  /// *Реализация IMessageService, управляющая сообщениями*  public class MessageService : IMessageService  {      private string \_storedMessage = "Привет из MessageService!";      public string GetMessage()      {          return \_storedMessage;      }      public void SaveMessage(string message)      {          \_storedMessage = message;      }  } |

**ApplicationService.cs:**

|  |
| --- |
| namespace Lab1;  /// *Основной сервис приложения, использующий внедрённые зависимости*  public class ApplicationService  {      private readonly IMessageService \_messageService;      public ApplicationService(IMessageService messageService)      {          \_messageService = messageService;      }      public void Run()      {          Console.WriteLine("Текущее сообщение: " + \_messageService.GetMessage());          Console.WriteLine("Введите новое сообщение:");          var input = Console.ReadLine();          if (!string.IsNullOrWhiteSpace(input))          {              \_messageService.SaveMessage(input);          }          Console.WriteLine("Ваше замещённое сообщение из ApplicationService: " + \_messageService.GetMessage());      }  } |

**Ссылка на репозиторий:** <https://github.com/avoryyy/backend_mospoly>