ЛАБОРАТОРНАЯ РАБОТА №8.
РЕФЛЕКСИЯ

СОДЕРЖАНИЕ

Цель

Ознакомиться с возможностями разбора классов и их методов.

Задание

1. Создать ветку от ветки седьмой лабораторной.
2. *Рефакторинг проекта:*
    a. *Разработать возможность создания бекапа данных. Сделать вариант бекапа в виде архива с данными из базы данных и файловой системы.*
    б. *Разработать методы для настройки табличного вывода данных (вывод в dataGridView) с помощью пользовательских атрибутов для классов ViewModels.*
    в. *Разработать механизм динамической настройки зависимостей для способа хранения данных без подключения библиотек к desktop-проекту.*
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Проектирование

И так, у нас 3 задачи, начнем по порядку усложнения. Самое простое – настройка табличного вывода. Для этого потребуется определить атрибут, который будет применяться к свойствам классов view-моделей. Через эти свойства будем задавать следующие параметры, которые важны нам при выводе через элемент DataGridView. Атрибут будет содержать следующие элементы:

- выводимый в шапке колонки заголовок;
- признак отображения или скрытия колонки;
- ширина колонки;
- авторазмер колонки;

- признак использования параметра авторазмера колонки для определения ширины колонки.

Данный набор никак нельзя считать строго обязательным. Он составлен исходя из сугубо наших потребностей в рамках этого проекта, для другого проекта набор может сильно отличаться. Даже этот набор, при желании, можно изменить, позволив достичь того же конечного результата. Например, можно убрать элемент «признак отображения или скрытия колонки», а определять, отображать колонку или нет по наличию текста в элементе «выводимый в шапке колонки заголовок», если текст есть, то колонку следует отображать, если нет, то колонку следует скрывать. Однако такой подход будет неочевидным, особенно для новых разработчиков, так что лучше для каждого значимого действия в построении DataGridView заводить отдельный элемент в атрибуте. Также у нас будет 2 элемента атрибута, отвечающие за настройку ширины колонки. Опять же, чтобы явно понимать, что будет использовать, заведем признак, отвечающий, в каких случаях будет использоваться элемент «авторазмер колонок», в остальных случаях будет использоваться элемент «ширина колонки».

После определения атрибута останется применить его к свойствам view-моделей, и сделать метод для формирования DataGridView на основе этих атрибутов. Метод сделаем через механизм расширения методов класса (для красоты). По логике все будет просто: подгружаем через свойство DataSource в DataGridView набор данных, далее через рефлексию вытаскиваем тип элементов набора данных, перечень свойств типа и атрибут для каждого свойства. Далее ищем в колонках DataGridView нужную колонку и настраиваем ее на основе данных атрибута. Если для свойства не задан атрибут, то будем вызывать ошибку, значит view-модель не настроена корректно, следует ее дополнить.

Перейдем к формированию бекапа. Так как предполагается формирования бекапа как минимум для 2 вариантов хранения данных, то пойдем стандартным путем, определим бизнес-логику формирования бекапа,

а получение данных из конкретного хранилища отдадим на откуп конкретной реализации. При сохранении будем активно использовать рефлексию. Это позволит нам избежать проблемы, если в будущем мы введем новые модели-классы. Выделим методы, которые будут специфичны для реализации:

- получение списка коллекция классов-моделей;
- получение типа-реализации по имени интерфейса-модели.

Выстроим следующую архитектуру:

- в слое контрактов объявим интерфейс создания бекапа и интерфейс получения данных для бекапа (первый будет использовать второй в процессе своей работы);
- в слое бизнес-логики сделаем реализацию интерфейса создания бекапа, в логике которого будем получать тип-реализацию класса-модели (сами классы вытащим из слоя моделей), получать списки по конкретному типу, сохранять их в формате json (сделаем через DataContractJsonSerializer, чтобы можно было настраивать, какие свойства сохранять, а какие игнорировать) и создавать архив из сохраненных данных;
- в слое хранения данных реализации интерфейс получения данных для бекапа (только для двух вариантов, хранения в файлах и хранение в БД).

Логику будем вызывать через пункт меню основной формы desktop-приложения.

Самое сложное – убрать зависимости в проектах. Для этого в слое контрактов объявим интерфейс, который будет определять зависимости. Реализации этого интерфейса будут в слое хранения данных. Также в слое контрактов объявим класс, который будет сохранять зависимости (IoC-контейнер будет объявлен в нем), создавать экземпляры через внутренний контейнер, а также подгружать зависимости из библиотеки. Загрузка зависимостей будет выполняться по следующему алгоритму:

- в проектах слоя хранения данных определить класс-реализацию интерфейса определения зависимостей;
- настроить выгрузку скомпилированной библиотеки проекта слоя хранения данных в определенную папку;
- в слое контрактов в классе хранения зависимостей сделать метод, который вытаскивает сборки из определенной папки и ищет там классы-реализации интерфейса определения зависимостей;
- если таких классов несколько, то смотрим свойство «Вес» у каждого такого класса и выбираем тот, у которого он больше;
- вызываем у найденного класса метод инициализации зависимостей для установления зависимостей интерфейсов хранения данных и их реализаций.

В desktop-приложение переделываем работу с IoC -контейнера на работу с классом определения зависимостей.

Реализация

Для конфигурации табличного вывода в проекте **AbstractShopContracts** создадим класс-атрибут для конфигурации колонки. Для авторазмера потребуется перечисление. Создадим перечисление GridViewAutoSize. Для заполнения поступим просто, откроем перечисление DataGridViewAutoSizeColumnMode и скопируем оттуда значения (листинг 8.1).

```
namespace AbstractShopContracts.Attributes
{
    public enum GridViewAutoSize
    {
        NotSet = 0,

        None = 1,

        ColumnHeader = 2,

        AllCellsExceptHeader = 4,

        AllCells = 6,

        DisplayedCellsExceptHeader = 8,

        DisplayedCells = 10,
```

```
        Fill = 16
    }
}
```

Листинг 8.1 – Перечисление GridViewAutoSize

Создадим класс и унаследуем его от класса Attribute (листинг 8.2).

```
using System;

namespace AbstractShopContracts.Attributes
{
    [AttributeUsage(AttributeTargets.Property)]
    public class ColumnAttribute : Attribute
    {
        public ColumnAttribute(string title = "", bool visible = true, int width
= 0, GridViewAutoSize gridViewAutoSize = GridViewAutoSize.None, bool
isUseAutoSize = false)
        {
            Title = title;
            Visible = visible;
            Width = width;
            GridViewAutoSize = gridViewAutoSize;
            IsUseAutoSize = isUseAutoSize;
        }

        public string Title { get; private set; }

        public bool Visible { get; private set; }

        public int Width { get; private set; }

        public GridViewAutoSize GridViewAutoSize { get; private set; }

        public bool IsUseAutoSize { get; private set; }
    }
}
```

Листинг 8.2 – Класс ColumnAttribute

Так как помимо настроек самих колонок через свойства класса может быть еще важен порядок вывода свойств в таблице. Для этого у каждого класса нужно будет прописать метод возврата списка свойств.

Для всех основных view-моделей (за исключением тех, что используются для отчетов) зададим для свойств атрибуты (листинг 8.3). Остальные view-модели настраиваются аналогично (реализовать самостоятельно).

В desktop-проекте создадим класс, в котором пропишем расширение для DataGridView (листинг 8.4).

```
using AbstractShopDataModels.Models;

namespace AbstractShopContracts.BindingModels
{
    public class ClientBindingModel : IClientModel
    {
        public int Id { get; set; }

        public string ClientFIO { get; set; } = string.Empty;

        public string Email { get; set; } = string.Empty;

        public string Password { get; set; } = string.Empty;
    }
}
```

Листинг 8.3 – Обновленный класс ClientViewModel

```
using AbstractShopContracts.Attributes;

namespace AbstractShopView
{
    public static class DataGridViewExtension
    {
        public static void FillandConfigGrid<T>(this DataGridView grid, List<T>?
data)
        {
            if (data == null)
            {
                return;
            }
            grid.DataSource = data;

            var type = typeof(T);
            var properties = type.GetProperties();
            foreach (DataGridViewColumn column in grid.Columns)
            {
                var property = properties.FirstOrDefault(x => x.Name ==
column.Name);
                if (property == null)
                {
                    throw new InvalidOperationException($"В типе {type.Name} не
найдено свойство с именем {column.Name}");
                }
                var attribute =
property.GetCustomAttributes(typeof(ColumnAttribute), true)?.SingleOrDefault();
                if (attribute == null)
                {
                    throw new InvalidOperationException($"Не найден атрибут типа
ColumnAttribute для свойства {property.Name}");
                }
                // ищем нужный нам атрибут
                if (attribute is ColumnAttribute columnAttr)
                {
                    column.HeaderText = columnAttr.Title;
                    column.Visible = columnAttr.Visible;
                    if (columnAttr.IsUseAutoSize)
                    {
                        column.AutoSizeMode =
(DataGridViewAutoSizeColumnMode)Enum.Parse(typeof(DataGridViewAutoSizeColumnMode)
, columnAttr.GridViewAutoSize.ToString());
                    }
                    else
                    {
                        column.Width = columnAttr.Width;
                    }
```

```
                }
            }
        }
    }
}
```

Листинг 8.4 – Класс DataGridViewExtension

Остается переделать вывод табличных данных в логиках форм (листинг 8.5).

```
        private void LoadData()
        {
            try
            {
                dataGridView.FillandConfigGrid(_logic.ReadList(null));
                _logger.LogInformation("Загрузка клиентов");
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка загрузки клиентов");
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
```

Листинг 8.5 – Обновленный метод для вывода списка клиентов

Данную логику следует применить ко всем местам вывода списков сущностей в desktop-проекте (реализовать саомстоятельно).

Займемся бекапом. Пропишем для хранилища интерфейс получения данных по всем сущностям (листинг 8.6) и для логики интерфейс создания бекапа (листинг 8.7).

```
namespace AbstractShopContracts.StoragesContracts
{
    public interface IBackUpInfo
    {
        List<T>? GetList<T>() where T : class, new();

        Type? GetTypeByModelInterface(string modelInterfaceName);
    }
}
```

Листинг 8.6 – Интерфейс IBackUpInfo

```
using AbstractShopContracts.BindingModels;

namespace AbstractShopContracts.BusinessLogicsContracts
{
    public interface IBackUpLogic
    {
        void CreateBackUp(BackUpSaveBinidngModel model);
    }
}
```

Листинг 8.7 – Интерфейс IBackUpLogic

Второй интерфейс имеет метод, принимающий на вход информацию для сохранения всех данных. На данный момент информация будет в себя включать путь и имя файла под архив (листинг 8.8).

```
namespace AbstractShopContracts.BindingModels
{
    public class BackUpSaveBinidngModel
    {
        public string FolderName { get; set; } = string.Empty;
    }
}
```

Листинг 8.8 – Класс BackUpSaveBinidngModel

Далее делаем реализацию под хранилище БД (листинг 8.9).

```
using AbstractShopContracts.StoragesContracts;

namespace AbstractShopDatabaseImplement.Implements
{
    public class BackUpInfo : IBackUpInfo
      {
            public List<T>? GetList<T>() where T: class, new()
        {
                using var context = new AbstractShopDatabase();
                return context.Set<T>().ToList();
            }

            public Type? GetTypeByModelInterface(string modelInterfaceName)
        {
                var assembly = typeof(BackUpInfo).Assembly;
                var types = assembly.GetTypes();
                foreach (var type in types)
                {
                    if (type.IsClass &&
type.GetInterface(modelInterfaceName) != null)
                    {
                        return type;
                    }
                }
                return null;
            }
    }
}
```

Листинг 8.9 – Класс BackUpInfo

Реализация для хранения в файлах будем иметь идентичный метод получения типа по имени интерфейса, а логика получения списка будет следующей (реализовать самостоятельно):

1. Создать объект Type по типу, список которого хотим получить.
2. Получить ссылку на объект DataFileSingleton.
3. Создать объект Type по классу DataFileSingleton.
4. Пройтись по свойствам объекта Type от класса DataFileSingleton.

a. Если свойство является универсальным (generic) и в качестве типа-значений имеет тот же тип, по которому хотим получить список, то вернуть значение этого свойства для объекта класса DataFileSingleton.

Также в классах-моделях в проектах-хранилищах нужно проставить атрибуты DataContract для самих классов и DataMember для тех свойств этих классов, которые следует сохранять в бекап (реализовать самостоятельно).

Сделаем реализацию логики создания бекапа-архива с данными. Сделаем маленькую хитрость, чтобы вытаскивать все интерфейсы-сущности с проекта с моделями. Унаследуем их все от интерфейса IId (нужно только доработать IMessageInfoModel, при этом новое поле Id нигде не должно участвовать в классах-моделях от IMessageInfoModel, просто быть) и будем из всех типов проекта с моделями отбирать интерфейсы, которые наследуются от интерфейса IId (листинг 8.10).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.StoragesContracts;
using AbstractShopDataModels;
using Microsoft.Extensions.Logging;
using System.IO.Compression;
using System.Reflection;
using System.Runtime.Serialization.Json;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    public class BackUpLogic : IBackUpLogic
    {
        private readonly ILogger _logger;

        private readonly IBackUpInfo _backUpInfo;

        public BackUpLogic(ILogger<BackUpLogic> logger, IBackUpInfo backUpInfo)
        {
            _logger = logger;
            _backUpInfo = backUpInfo;
        }

        public void CreateBackUp(BackUpSaveBinidngModel model)
        {
            if (_backUpInfo == null)
            {
                return;
            }
            try
            {
                _logger.LogDebug("Clear folder");
                // зачистка папки и удаление старого архива
                var dirInfo = new DirectoryInfo(model.FolderName);
                if (dirInfo.Exists)
```

```
                {
                    foreach (var file in dirInfo.GetFiles())
                    {
                        file.Delete();
                    }
                }
                _logger.LogDebug("Delete archive");
                string fileName = $"{model.FolderName}.zip";
                if (File.Exists(fileName))
                {
                    File.Delete(fileName);
                }
                // берем метод для сохранения
                _logger.LogDebug("Get assembly");
                var typeIId = typeof(IId);
                var assembly = typeIId.Assembly;
                if (assembly == null)
                {
                    throw new ArgumentNullException("Сборка не найдена",
nameof(assembly));
                }
                var types = assembly.GetTypes();
                var method = GetType().GetMethod("SaveToFile",
BindingFlags.NonPublic | BindingFlags.Instance);
                _logger.LogDebug("Find {count} types", types.Length);
                foreach (var type in types)
                {
                    if (type.IsInterface && type.GetInterface(typeIId.Name) !=
null)
                    {
                        var modelType =
_backUpInfo.GetTypeByModelInterface(type.Name);
                        if (modelType == null)
                        {
                            throw new InvalidOperationException($"Не найден
класс-модель для {type.Name}");
                        }
                        _logger.LogDebug("Call SaveToFile method for {name}
type", type.Name);
                        // вызываем метод на выполнение
                        method?.MakeGenericMethod(modelType).Invoke(this, new
object[] { model.FolderName });
                    }
                }
                _logger.LogDebug("Create zip and remove folder");
                // архивируем
                ZipFile.CreateFromDirectory(model.FolderName, fileName);
                // удаляем папку
                dirInfo.Delete(true);
            }
            catch (Exception)
            {
                throw;
            }
        }

        private void SaveToFile<T>(string folderName) where T : class, new()
        {
            var records = _backUpInfo.GetList<T>();
            if (records == null)
            {
                _logger.LogWarning("{type} type get null list", typeof(T).Name);
                return;
            }
            var jsonFormatter = new DataContractJsonSerializer(typeof(List<T>));
```

```
                    using var fs = new FileStream(string.Format("{0}/{1}.json",
folderName, typeof(T).Name), FileMode.OpenOrCreate);
                    jsonFormatter.WriteObject(fs, records);
                }
    }
}
```

Листинг 8.10 – Класс BackUpLogic

В проекте **AbstractShopView** в классе Program в IoC-контейнере прописываем связки. На главной форме в меню добавляем пункт «Создать бекап». В логике создаем диалог для выбора папки для сохранения и вызываем метод создания бекапа (листинг 8.11).

```
        private void СоздатьБекапToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            try
            {
                if (_backUpLogic != null)
                {
                    var fbd = new FolderBrowserDialog();
                    if (fbd.ShowDialog() == DialogResult.OK)
                    {
                        _backUpLogic.CreateBackUp(new BackUpSaveBinidngModel {
FolderName = fbd.SelectedPath });
                        MessageBox.Show("Бекап создан", "Сообщение",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
```

Листинг 8.11 – Логика метода создатьБекапToolStripMenuItem_Click

Для реализации механизма динамической подгрузки зависимостей в проекте с контрактами зададим класс для хранения зависимостей. Чтобы этот класс был более универсальным и не зависел от конкретного способа хранения зависимостей (например, от IoC-контейнера) введем интерфейс для установки набора методов для задания связей между интерфейсами и их реализациями, и получения экземпляра нужной зависимости (листинг 8.12). И сделаем его реализацию с использованием ServiceProvider (реализовать самостоятельно).

```
using Microsoft.Extensions.Logging;

namespace AbstractShopContracts.DI
{
    /// <summary>
    /// Интерфейс установки зависмости между элементами
    /// </summary>
```

```
    public interface IDependencyContainer
      {
            /// <summary>
            /// Регистрация логгера
            /// </summary>
            /// <param name="configure"></param>
            void AddLogging(Action<ILoggingBuilder> configure);

            /// <summary>
            /// Добавление зависимости
            /// </summary>
            /// <typeparam name="T"></typeparam>
            /// <typeparam name="U"></typeparam>
            /// <param name="isSingle"></param>
            void RegisterType<T, U>(bool isSingle) where U : class, T where T :
class;

            /// <summary>
            /// Добавление зависимости
            /// </summary>
            /// <typeparam name="T"></typeparam>
            /// <param name="isSingle"></param>
            void RegisterType<T>(bool isSingle) where T : class;

            /// <summary>
            /// Получение класса со всеми зависмостями
            /// </summary>
            /// <typeparam name="T"></typeparam>
            /// <returns></returns>
            T Resolve<T>();
      }
}
```

Листинг 8.12 – Интерфейс IDependencyContainer

Далее объявим интерфейс, через который будут задаваться зависимости для проектов-хранилищ (листинг 8.13). Он будет состоять из свойства для указания приоритета реализации интерфейса и метода, внутри которого будут определятся зависимости между интерфейсами хранения данных и их реализациями в конкретном проекте.

```
namespace AbstractShopContracts.DI
{
    /// <summary>
    /// Интерфейс для регистрации зависимостей в модулях
    /// </summary>
    public interface IImplementationExtension
      {
            public int Priority { get; }
            /// <summary>
            /// Регистрация сервисов
            /// </summary>
            public void RegisterServices();
      }
}
```

Листинг 8.13 – Интерфейс IImplementationExtension

Для загрузки реализации этого интерфейса сделаем отдельный класс, в логике которого будем перебирать сборки из определенной папки и искать там реализации интерфейса IImplementationExtension, выбирая самую приоритетную (листинг 8.14).

```
using System.Reflection;

namespace AbstractShopContracts.DI
{
    /// <summary>
    /// Загрузчик данных
    /// </summary>
    public static partial class ServiceProviderLoader
       {
              /// <summary>
              /// Загрузка всех классов-реализаций IImplementationExtension
              /// </summary>
              /// <returns></returns>
              public static IImplementationExtension? GetImplementationExtensions()
              {
                     IImplementationExtension? source = null;
                     var files =
Directory.GetFiles(TryGetImplementationExtensionsFolder(), "*.dll",
SearchOption.AllDirectories);
                     foreach (var file in files.Distinct())
                     {
                            Assembly asm = Assembly.LoadFrom(file);
                            foreach (var t in asm.GetExportedTypes())
                            {
                                   if (t.IsClass &&
typeof(IImplementationExtension).IsAssignableFrom(t))
                                   {
                                          if (source == null)
                                 {
                                                 source =
(IImplementationExtension)Activator.CreateInstance(t)!;
                                          }
                                          else
                                 {
                                                 var newSource =
(IImplementationExtension)Activator.CreateInstance(t)!;
                                                 if (newSource.Priority >
source.Priority)
                                    {
                                                        source = newSource;
                                    }
                                          }
                                   }
                            }
                     }
                     return source;
              }

              private static string TryGetImplementationExtensionsFolder()
              {
                     var directory = new
DirectoryInfo(Directory.GetCurrentDirectory());
                     while (directory != null &&
!directory.GetDirectories("ImplementationExtensions",
SearchOption.AllDirectories).Any(x => x.Name == "ImplementationExtensions"))
                     {
                            directory = directory.Parent;
```

```
                }
                return $"{directory?.FullName}\\ImplementationExtensions";
            }
        }
}
```

Листинг 8.14 – Класс ServiceProviderLoader

И главный класс, который подгружает зависимости из реализаций хранилища, а также позволяет задать иные зависимости (листинг 8.15).

```
using Microsoft.Extensions.Logging;

namespace AbstractShopContracts.DI
{
    /// <summary>
    /// Менеджер для работы с зависимостями
    /// </summary>
    public class DependencyManager
       {
            private readonly IDependencyContainer _dependencyManager;

            private static DependencyManager? _manager;

            private static readonly object _locjObject = new();

            private DependencyManager()
            {
                _dependencyManager = new ServiceDependencyContainer();
            }

            public static DependencyManager Instance { get { if (_manager ==
null) { lock (_locjObject) { _manager = new DependencyManager(); } } return
_manager; } }

            /// <summary>
            /// Инициализация библиотек, в которых идут установки зависомстей
            /// </summary>
            public static void InitDependency()
            {
                var ext = ServiceProviderLoader.GetImplementationExtensions();
                if (ext == null)
                {
                    throw new ArgumentNullException("Отсутствуют компоненты
для загрузки зависимостей по модулям");
                }
                // регистрируем зависимости
                ext.RegisterServices();
            }

            /// <summary>
            /// Регистрация логгера
            /// </summary>
            /// <param name="configure"></param>
            public void AddLogging(Action<ILoggingBuilder> configure) =>
_dependencyManager.AddLogging(configure);

            /// <summary>
            /// Добавление зависимости
            /// </summary>
            /// <typeparam name="T"></typeparam>
            /// <typeparam name="U"></typeparam>
            public void RegisterType<T, U>(bool isSingle = false) where U :
class, T  where T : class => _dependencyManager.RegisterType<T, U>(isSingle);
```

```
            /// <summary>
            /// Добавление зависимости
            /// </summary>
            /// <typeparam name="T"></typeparam>
            /// <typeparam name="U"></typeparam>
            public void RegisterType<T>(bool isSingle = false) where T : class =>
_dependencyManager.RegisterType<T>(isSingle);

            /// <summary>
            /// Получение класса со всеми зависмостями
            /// </summary>
            /// <typeparam name="T"></typeparam>
            /// <returns></returns>
            public T Resolve<T>() => _dependencyManager.Resolve<T>();
        }
}
```

Листинг 8.15 – Класс DependencyManager

В данном случае, одна зависимость проставляется всегда жестко в коде – это реализация IDependencyContainer. Через нее идет регистрация зависимостей, а также получение объектов. Класс выполнен по паттерну Singleton, причем для многопоточного варианта реализации, так что у нас на все время жизни программы будет один экземпляр этого класса, и все зависимости будут там зарегистрированы.

Рассмотрим на примере хранилища в операционной памяти, как реализовывать IImplementationExtension (листинг 8.16). Также не забываем, что в настройках проектов-реализация способов хранения потребуется прописать, что по окончании сборки, полученные файлы-библиотеки надо переносить в определенную папку, из которой они потом будут считываться ServiceProviderLoader (чтобы не копировать ручками каждый раз после пересборки проектов).

```
using AbstractShopContracts.DI;
using AbstractShopContracts.StoragesContracts;
using AbstractShopListImplement.Implements;

namespace AbstractShopListImplement
{
    public class ListImplementationExtension : IImplementationExtension
    {
        public int Priority => 0;

        public void RegisterServices()
        {
            DependencyManager.Instance.RegisterType<IClientStorage,
ClientStorage>();
            DependencyManager.Instance.RegisterType<IComponentStorage,
ComponentStorage>();
```

```
            DependencyManager.Instance.RegisterType<IImplementerStorage,
ImplementerStorage>();
            DependencyManager.Instance.RegisterType<IMessageInfoStorage,
MessageInfoStorage>();
            DependencyManager.Instance.RegisterType<IOrderStorage,
OrderStorage>();
            DependencyManager.Instance.RegisterType<IProductStorage,
ProductStorage>();
            DependencyManager.Instance.RegisterType<IBackUpInfo, BackUpInfo>();
        }
    }
}
```

Листинг 8.16 – Класс ListImplementationExtension

Последний шаг – в desktop-проекте заменить работу с IoC-контейнером на работу с DependencyManager. Например, в классе Program (листинг 8.16).

```
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.MailWorker;
using AbstractShopBusinessLogic.OfficePackage;
using AbstractShopBusinessLogic.OfficePackage.Implements;
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.DI;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using NLog.Extensions.Logging;

namespace AbstractShopView
{
    internal static class Program
    {
            /// <summary>
            ///  The main entry point for the application.
            /// </summary>
            [STAThread]
        static void Main()
        {
            // To customize application configuration such as set high DPI
settings or default font,
            // see https://aka.ms/applicationconfiguration.
            ApplicationConfiguration.Initialize();
            var services = new ServiceCollection();
                    InitDependency();

                    try
                    {
                            var mailSender =
DependencyManager.Instance.Resolve<AbstractMailWorker>();
                            mailSender?.MailConfig(new MailConfigBindingModel
                            {
                                    MailLogin =
System.Configuration.ConfigurationManager.AppSettings["MailLogin"] ??
string.Empty,
                                    MailPassword =
System.Configuration.ConfigurationManager.AppSettings["MailPassword"] ??
string.Empty,
                                    SmtpClientHost =
System.Configuration.ConfigurationManager.AppSettings["SmtpClientHost"] ??
string.Empty,
                                    SmtpClientPort =
Convert.ToInt32(System.Configuration.ConfigurationManager.AppSettings["SmtpClient
Port"]),
```

```
                                PopHost =
System.Configuration.ConfigurationManager.AppSettings["PopHost"] ?? string.Empty,
                                PopPort =
Convert.ToInt32(System.Configuration.ConfigurationManager.AppSettings["PopPort"])
                            });

                            // создаем таймер
                            var timer = new System.Threading.Timer(new
TimerCallback(MailCheck!), null, 0, 100000);
                    }
                    catch(Exception ex)
            {
                            var logger =
DependencyManager.Instance.Resolve<ILogger>();
                            logger?.LogError(ex, "Ошибка работы с почтой");
                    }



      Application.Run(DependencyManager.Instance.Resolve<FormMain>());
            }

            private static void InitDependency()
            {
                  DependencyManager.InitDependency();

                  DependencyManager.Instance.AddLogging(option =>
                  {
                        option.SetMinimumLevel(LogLevel.Information);
                        option.AddNLog("nlog.config");
                  });

                  DependencyManager.Instance.RegisterType<IComponentLogic,
ComponentLogic>();
                  DependencyManager.Instance.RegisterType<IOrderLogic,
OrderLogic>();
                  DependencyManager.Instance.RegisterType<IProductLogic,
ProductLogic>();
                  DependencyManager.Instance.RegisterType<IReportLogic,
ReportLogic>();
                  DependencyManager.Instance.RegisterType<IClientLogic,
ClientLogic>();
                  DependencyManager.Instance.RegisterType<IImplementerLogic,
ImplementerLogic>();
                  DependencyManager.Instance.RegisterType<IMessageInfoLogic,
MessageInfoLogic>();

                  DependencyManager.Instance.RegisterType<AbstractSaveToExcel,
SaveToExcel>();
                  DependencyManager.Instance.RegisterType<AbstractSaveToWord,
SaveToWord>();
                  DependencyManager.Instance.RegisterType<AbstractSaveToPdf,
SaveToPdf>();
                  DependencyManager.Instance.RegisterType<IWorkProcess,
WorkModeling>();
                  DependencyManager.Instance.RegisterType<AbstractMailWorker,
MailKitWorker>(true);
                  DependencyManager.Instance.RegisterType<IBackUpLogic,
BackUpLogic>();

                  DependencyManager.Instance.RegisterType<FormMain>();
                  DependencyManager.Instance.RegisterType<FormComponent>();
                  DependencyManager.Instance.RegisterType<FormComponents>();
                  DependencyManager.Instance.RegisterType<FormCreateOrder>();
                  DependencyManager.Instance.RegisterType<FormProduct>();
```

```
        DependencyManager.Instance.RegisterType<FormProductComponent>();
                    DependencyManager.Instance.RegisterType<FormProducts>();

        DependencyManager.Instance.RegisterType<FormReportProductComponents>();
                    DependencyManager.Instance.RegisterType<FormReportOrders>();
                    DependencyManager.Instance.RegisterType<FormClients>();
                    DependencyManager.Instance.RegisterType<FormImplementer>();
                    DependencyManager.Instance.RegisterType<FormImplementers>();
                    DependencyManager.Instance.RegisterType<FormMails>();
                }

                private static void MailCheck(object obj) =>
DependencyManager.Instance.Resolve<AbstractMailWorker>()?.MailCheck();
        }
}
```

Листинг 8.16 – Класс Program

Осталась определение зависимости от проекта-реализации бизнес-логики, а вот определение зависимости от проекта-хранилища ушла в вызов метода InitDependency класса DependencyManager.

А в desktop-проекте убрать зависимости от проектов-хранилищ.

Контрольный пример

Для проверки работоспособности нового механизма вывода и настройки DataGrigView достаточно просто пройтись по всем формам списков и посмотреть, что они отображаются корректно. Например, главная форма (рисунок 8.1) или форма с исполнителями (рисунок 8.2).

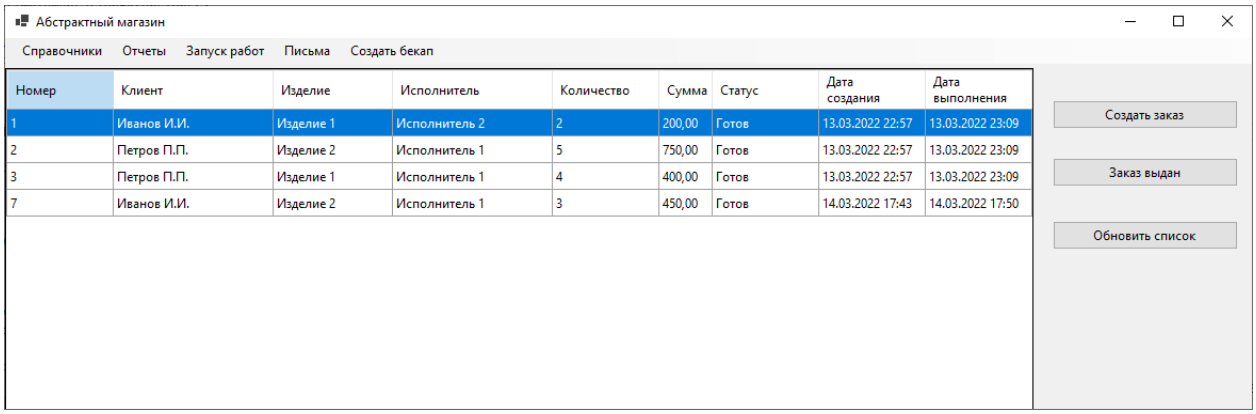

Рисунок 8.1 – Главная форма

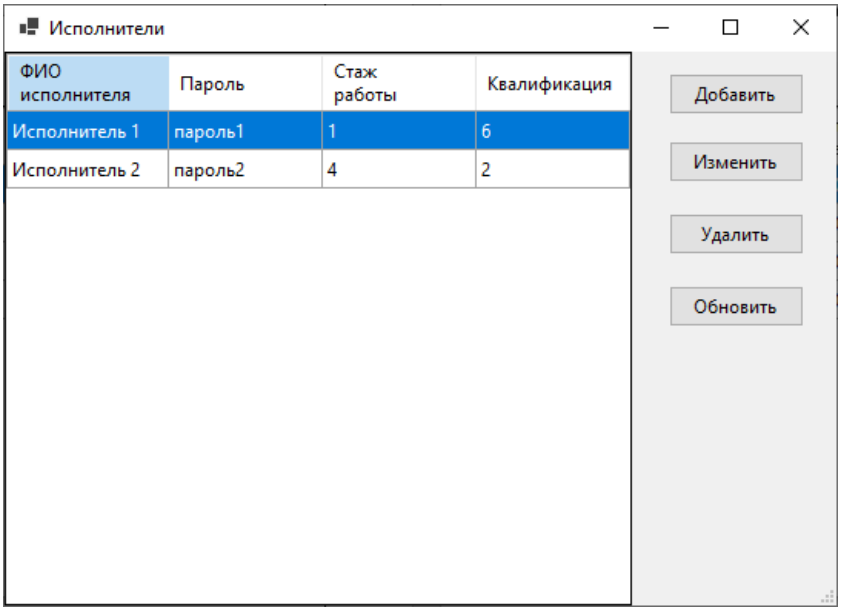

Рисунок 8.2 – Форма с исполнителями

Далее с главной формы вызовем метод создания бекапа и создадим под него папку «backup» (рисунок 8.3).

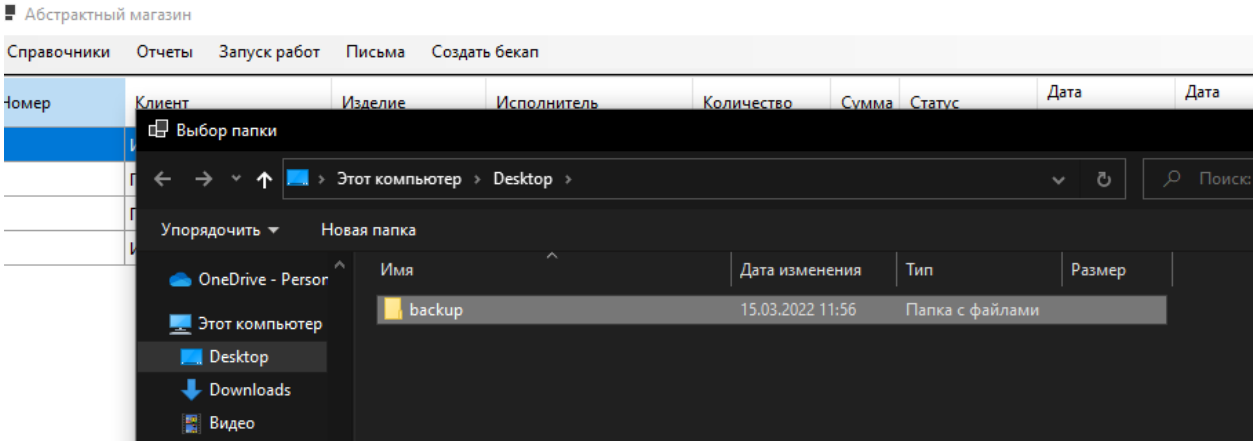

Рисунок 8.3 – Создание бекапа

По результату работы метода получится архив «backup.zip», в котором будут файлы, согласно сущностям нашего проекта (рисунок 8.4).

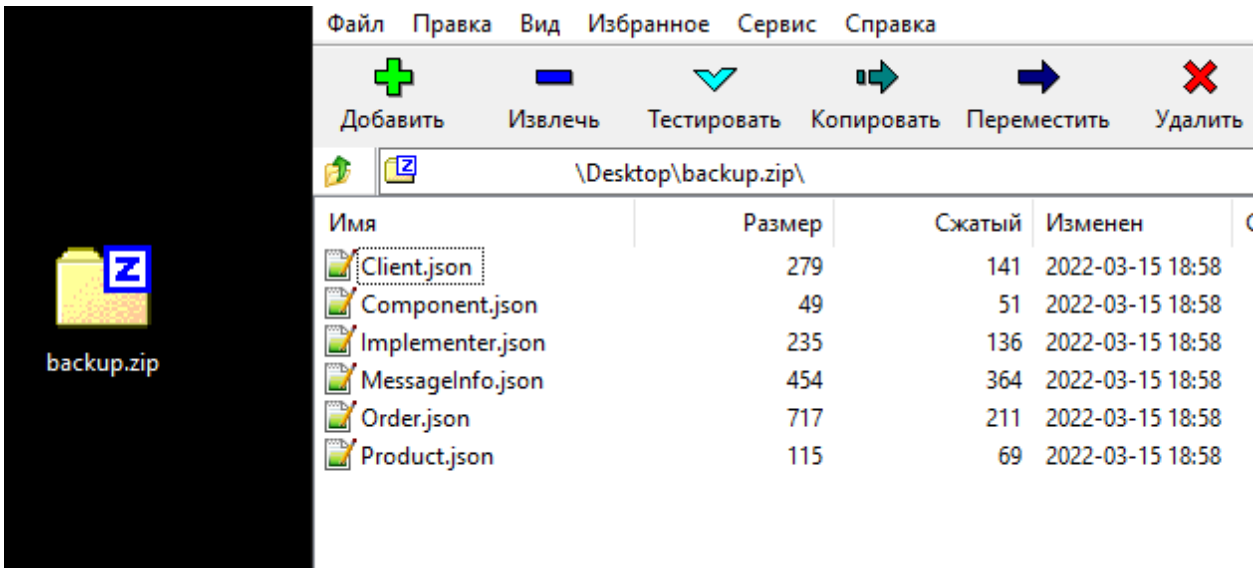

Рисунок 8.4 – Созданный бекап

Далее, закроем проект, перейдем в папку ImplementationExtensions в корне проекта и удалим оттуда библиотеку с реализаций хранения в БД. Запустим снова проект и увидим пустую форму (для второго варианта хранения в файлах у нас на этот момент нет файлов, так что списки будут пустые) (рисунок 8.5).

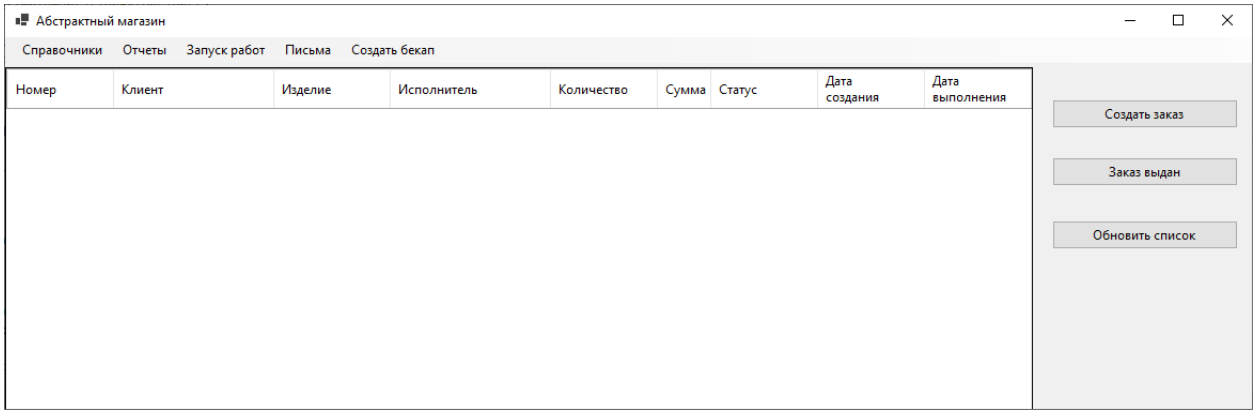

Рисунок 8.5 – Главная форма с пустым набором данных

Требования

1. Название проектов должны ОТЛИЧАТЬСЯ от названия проектов, приведенных в примере и должны соответствовать логике вашего задания по варианту.
2. Название форм, классов, свойств классов должно соответствовать логике вашего задания по варианту.

3. НЕ ИСПОЛЬЗОВАТЬ в названии класса, связанного с изделием слово «Product» (во вариантах в скобках указано название класса для изделия)!!!
4. Все элементы форм (заголовки форм, текст в label и т.д.) должны иметь подписи на одном языке (или все русским, или все английским).
5. Прописать во view-моделях атрибуты для табличного вывода.
6. Дополнить классы-модели в проектах-хранилищах атрибутами DataContract для классов и DataMember для тех свойств этих классов, которые следует сохранять в бекап.
7. Создать реализацию IBackUpInfo для проекта хранения данных в файлах.
8. Переделать вывод всех табличных форм в desktop-проекте.
9. Сделать реализацию IDependencyManager с Unity-контейнером.

Порядок сдачи базовой части

1. Запустить Desktop-проект, показать 2-3 формы со списками по разным сущностям, убедиться, что есть записи по всем сущностям.
2. Создать бекап.
3. Открыть бекап, открыть любой из файлов, показать содержимое.
4. Закрыть приложение, удалить библиотеку с реализацией хранения в БД, запустить проект.
5. Ответить на вопрос преподавателя.

Контрольные вопросы к базовой части

1. Как реализован метод настройки DataGridView?
2. Как реализована архивация данных?
3. Как реализован новый механизм настройки зависимостей?

Усложненная лабораторная (необязательно)

1. Дополнить настройки колонок, чтобы был форматированный вывод данных (например, для дат, чисел).
2. Сделать настройку зависимостей для бизнес-логики по аналогии с настройкой зависимостей для хранения данных.

Порядок сдачи усложненной части

1. Запустить Desktop-проект, показать вывод дат, чисел.
2. Ответить на вопрос преподавателя.

Контрольные вопросы к усложненной части

1. Как реализован форматированный вывод данных в DataGridView?
2. Как реализован механизм настройки зависимостей для бизнес-логики?
3. Как поменялся алгоритм архивации данных с учетом появления новой сущности «Магазин»?

Варианты

1. Кондитерская. В качестве компонентов выступают различные виды шоколада и наполнители, типа орехов, изюма и т.п. Изделие – кондитерское изделие (pastry).
2. Автомастерская. В качестве компонентов выступают различные масла, смазки и т.п. Изделия – ремонт автомобиля (repair).
3. Моторный завод. В качестве компонентов выступают различные детали для производства двигателей. Изделия – двигатели (engine).
4. Суши-бар. В качестве компонентов выступают различные продукты для суши (рыба, водоросли, соусы). Изделия – суши (sushi).
5. Продажа компьютеров. В качестве компонентов выступают различные части для компьютеров (планки памяти, жесткие диски и т.п.). Изделия – компьютеры (computer).

6. Сборка мебели. В качестве компонентов выступают различные заготовки (ножки, спинки и т.п.). Изделия – мебель (furniture).
7. Рыбный завод. В качестве компонентов выступают различные виды рыб + дополнения к ним, типа соусов и т.п. Изделия – консервы (canned).
8. Установка ПО. В качестве компонентов выступают различное ПО. Изделия – пакеты установки, например, пакет установки офисных приложений, пакет разработчика и т.п. (package).
9. Ремонтные работы в помещении. В качестве компонентов выступают различные расходные материалы (клей, обои, краска, плитка, цемент и т.п.). Изделия – ремонтные работы в различных помещениях (repair).
10. Кузнечная мастерская. В качестве компонентов выступают различные болванки (заготовки), из которых изготавливаются подковы, кочерги и т.п. Изделия – кузнечные изделия (manufacture).
11. Пиццерия. В качестве компонентов выступают различные ингредиенты для пицц (тесто, соусы, паста и т.д.). Изделия – пиццы (pizza).
12. Завод ЖБИ. В качестве компонентов выступают различные виды бетона и металлоконструкций. Изделия – железобетонные изделия (reinforced).
13. Закусочная. В качестве компонентов выступают различные продукты для закусок (колбаса, сыр, хлеб и т.п.). Изделия – различные закуски (snack).
14. Пошив платьев. В качестве компонентов выступают различные ткани, нитки и т.п. Изделия – платья (dress).
15. Типография. В качестве компонентов выступают различные типы бумаг, тонер или чернила и т.п. Изделия – печатная продукция (листовки, брошюры, книги) (printed).

16. Автомобильный завод. В качестве компонентов выступают различные части для сборки автомобилей (кузов, двигатель, стекла и т.п.). Изделия – автомобили (car).
17. Юридическая фирма. В качестве компонентов выступают различные бланки для документов. Изделия – пакеты документов, например, для страховки или завещания (document).
18. Туристическая фирма. В качестве компонентов выступают различные условия поездки (отель проживания, туры в рамках поездок). Изделия – туристические путевки (travel).
19. Цветочная лавка. В качестве компонентов выступают различные цветы и украшения к ним. Изделия – цветочные композиции (flower).
20. Ювелирная лавка. В качестве компонентов выступают различные драгоценные камни и металлы. Изделия – драгоценности (jewel).
21. Авиастроительный завод. В качестве компонентов выступают различные части для сборки самолета (двигатели, крылья, фюзеляж и т.п.). Изделия – самолеты (plane).
22. Магазин подарков. В качестве компонентов выступают различные упаковочные материалы, ленты и подарки. Изделия – подарочные наборы (gift).
23. Система безопасности. В качестве компонентов выступают различные камеры, датчики и т.п. Изделия – базовые комплектации охраны, продвинутые, для предприятий, для частных и т.п. (secure).
24. Заказы еды. В качестве компонентов выступают различные блюда. Изделия – это наборы блюд (типа обеденный набор, или утренний набор, или набор для пикника) (dish).
25. Ремонт сантехники. В качестве компонентов выступают различные трубы, прокладки, смесители т.п. Изделия – замены смесителей, труб и т.п. (work).

26. Лавка с мороженым. В качестве компонентов выступают различные виды мороженного и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icecream).
27. Судостроительный завод. В качестве компонентов выступают различные части для сборки судов (корпуса, двигатели и т.п.). Изделия – суда (ship).
28. Столярная мастерская. В качестве компонентов выступают различные деревянные заготовки. Изделия – деревянные игрушки, утварь и т.п. (wood).
29. Бар. В качестве компонентов выступают различные ингредиенты для коктейлей. Изделия – коктейли (cocktail).
30. Швейная фабрика. В качестве компонентов выступают различные заготовки для штор, покрывал и т.п. (textile).