# ЛАБОРАТОРНАЯ РАБОТА №7.
# РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

## СОДЕРЖАНИЕ

**Цель**

Ознакомиться с механизмами работы с почтовыми сервисами. Ознакомиться с применением регулярных выражений.

**Задание**

1. Создать ветку от ветки шестой лабораторной.
2. *Добавить работу с почтовым клиентом:*
   a. *Делать оповещение клиента (отправка письма с корпоративной почты) при смене статусов его заказов.*
   б. *Проверять корпоративную почту на предмет писем от клиентов.*
   в. *Дополнительно: у клиента добавить проверки, что в поле логин вводится адрес электронной почты (чтобы точно отправлять письма на электронный адрес), а пароль удовлетворяет ряду условий.*
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

**Проектирование**

Рассмотрим технологии, используемые для отправки/получения почтовых сообщений. Работа с почтой осуществляется через определенные протоколы передачи данных (наборы соглашений логического уровня об интерфейсах передачи данных). Для отправки сообщений в сетях TCP/IP предназначен сетевой протокол SMTP (англ. Simple Mail Transfer Protocol – простой протокол передачи почты). Он используется для отправки почты от пользователей к серверам и между серверами для дальнейшей пересылки к получателю.

Данный протокол получил распространение в начале 80-х годов. До этого использовался протокол UUCP, основным недостатком которого была необходимость знания полного маршрута от отправителя до получателя и явного указания этого маршрута в адресе получателя либо наличия прямого

коммутируемого или постоянного соединения между компьютерами отправителя и получателя.

Сервер SMTP представляет собой конечный автомат с внутренним состоянием. Клиент передает на сервер строку вида: «команда<пробел>параметры<перевод строки>». Сервер отвечает на каждую команду строкой, содержащей код ответа и текстовое сообщение, отделенное пробелом. Код ответа представляет собой число в диапазоне от 100 до 999, представленное в виде строки. Коды имеют следующие значения:

2XX – команда успешно выполнена

3XX – ожидаются дополнительные данные от клиента

4XX – временная ошибка, клиент должен произвести следующую попытку через некоторое время

5XX – неустранимая ошибка

Текстовая часть ответа носит справочный характер. Общение между клиентом и сервером может осуществляться через ряд портов, открытых на почтовом сервере.

Рассмотрим пример обычной SMTP-сессии. В данном примере C: – клиент, а S: – сервер (листинг 7.1).

S: (ожидает соединения)

C: (подключается к порту 25 сервера)

S:220 mail.MyCompany.ltd ESMTP is glad to see you!//подключение успешно

C:HELO //начать работу

S:250 domain name should be qualified //укажите отправителя

C:MAIL FROM: user1name@company.ru //адрес отправителя

S:250 user1name@company.ru sender accepted //адрес принят

C:RCPT TO:user2@company.ltd // адрес получателя

S:250 user2@company.ltd ok // адрес принят

C:RCPT TO: user3@company.lttd //еще адрес получателя

S:550 user3@company.lttd unknown user account //адреса не существует

C:DATA // передача данных письма

S:354 Enter mail, end with "." on a line by itself //ожидаются данные,

//по окончании ввода ожидается точка.

C:from: user1name@company.ru //текст письма

C:to: user2@company.ltd //текст письма

C:subject: tema //текст письма

C: //текст письма

C:Hi! //текст письма

C:. //текст письма

S:250 769947 message accepted for delivery//письмо принято для доставки

C:QUIT //клиент отключается

S:221 mail.company.tld ESMTP closing connection //сервер принял команду

S: (закрывает соединение)

Листинг 7.1 – Пример SMTP-сессии

В результате такой сессии письмо будет доставлено адресату user2@company.ltd, но не будет доставлено адресату user3@company.lttd, потому что такого адреса не существует.

Проверка почты. Получение писем.

Получение почтовых сообщений с сервера осуществляется посредством протокола POP3 (англ. Post Office Protocol Version 3 – протокол почтового отделения, версия 3). В данном протоколе предусмотрено 3 состояния сеанса:

- авторизация, когда клиент проходит процедуру аутентификации;
- транзакция, когда клиент получает информацию о состоянии почтового ящика, принимает и удаляет почту;

- обновление, когда сервер удаляет выбранные письма и закрывает соединение.

В таблице 7.1 приведены команды, поддерживаемые протоколом POP.

Таблица 7.1. Команды протокола POP

| Команда | Описание | Аргументы | Ограничения | Возможные ответы |
|---|---|---|---|---|
| APOP [имя] [digest] | Команда служит для передачи серверу имени пользователя и зашифрованного пароля (digest). | [имя] – строка, указывающая имя почтового ящика. [digest] — хеш-сумма временной метки, связанной с паролем пользователя, вычисленная по алгоритму MD5. Временная метка получается при соединении с сервером. | Ее поддержка не является обязательной. | +OK maildrop has n message; -ERR password suplied for [имя] is incorrect. |
| USER [имя] | Передает серверу имя пользователя. | [имя] – строка, указывающая имя почтового ящика. | – | +OK name is a valid mailbox; -ERR never heard of mailbox name. |
| DELE [сообщение] | Сервер помечает указанное сообщение для удаления. Такие сообщения реально удаляются только после закрытия транзакции (закрытие происходит по команде QUIT или иногда по истечении времени, установленного сервером). | [сообщение] – номер сообщения. | Доступна после успешной идентификации. | +OK message deleted; -ERR no such message. |
| PASS [пароль] | Передает серверу пароль почтового ящика. | [пароль] – пароль для почтового ящика. | Работает после успешной передачи имени почтового ящика. | +OK maildrop locked and ready; -ERR invalid password; -ERR unable to lock maildrop. |

Продолжение таблицы 7.1

| Команда | Описание | Аргументы | Ограничения | Возможные ответы |
|---|---|---|---|---|
| LIST [сообщение] | Если был передан аргумент, сервер выдает информацию об указанном сообщении. Если аргумент не был передан, то сервер выдаёт информацию обо всех сообщениях, находящихся в почтовом ящике. Сообщения, помеченные для удаления, не перечисляются. | [сообщение] – номер сообщения (необязательный аргумент). | Доступна после успешной идентификации. | +OK scan listing follows;<br>-ERR no such message. |
| NOOP | Сервер ничего не делает, всегда отвечает положительно. Команда используется для поддержки соединения с сервером при длительном бездействии. | – | Доступна после успешной идентификации. | +OK. |
| RETR [сообщение] | Сервер передает сообщение с указанным номером. | [сообщение] – номер сообщения. | Доступна после успешной идентификации. | +OK message follows;<br>-ERR no such message. |
| RSET | Этой командой производится откат транзакций внутри сессии. Например, если пользователь случайно пометил на удаление какие-либо сообщения, он может убрать эти пометки, отправив эту команду. | – | Доступна после успешной идентификации. | +OK. |

Продолжение таблицы 7.1

| Команда | Описание | Аргументы | Ограничения | Возможные ответы |
|---|---|---|---|---|
| STAT | Сервер возвращает количество сообщений в почтовом ящике плюс размер, занимаемый этими сообщениями на почтовом ящике. | _ | Доступна после успешной идентификации. | +OK [количество] [размер]. |
| TOP [сообщение] [количество строк] | Сервер возвращает заголовки указанного сообщения, пустую строку и указанное количество первых строк тела сообщения. | [сообщение] – номер сообщения. [количество строк] – сколько строк нужно вывести. | Доступна после успешной идентификации. | +OK n octets; -ERR no such message. |
| QUIT | Закрытие соединения | _ | _ | +OK. |

По аналогии с сервером SMTP рассмотрим пример сессии с сервером POP3 (листинг 7.2).

```
S: <Сервер ожидает входящих соединений на порту 110>
C: <подключается к серверу>
S:   +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C:   APOP mrose c4c9334bac560ecc979e58001b3e22fb
S:   +OK mrose's maildrop has 2 messages (320 octets)
C:   STAT
S:   +OK 2 320
C:   LIST
S:   +OK 2 messages (320 octets)
S:   1 120
S:   2 200
S:   .
C:   RETR 1
S:   +OK 120 octets
S:   <сервер передает сообщение 1>
S:   .
C:   DELE 1
```

```
S:   +OK message 1 deleted
C:   RETR 2
S:   +OK 200 octets
S:   <сервер передает сообщение 2>
S:   .
C:   DELE 2
S:   +OK message 2 deleted
C:   QUIT
S:   +OK dewey POP3 server signing off (maildrop empty)
C:  <закрывает соединение>
S:  <продолжает ждать входящие соединения>
```

Листинг 7.2 – Пример сессии с сервером POP3

Само собой, редко кто напрямую «общается» с сервисами. В большинстве своем используются готовые решения как для отправки писем через почту, так и для получения писем с почтового адреса. Для того, чтобы иметь возможность гибко менять библиотеки для работы с почтой, сделаем абстракцию для работы с почтой, через которую будет подключаться к почтовому сервису, отправлять и получать письма через него. При реализации это абстракции в качестве библиотеки для отправки писем будем использовать встроенную в .Net библиотеку System.Net.Mail. А для проверки почты – библиотеку MailKit.

Необходимо определится где и когда будут вызываться методы отправки писем и проверки почты. Так как отправка писем будет происходить при создании заказа, либо смене его статусов, то логично поместить метод в бизнес-логику работы с заказом. Структура письма при этом будет следующей:

- кому отправляем (адрес почты берем от клиента);
- заголовок (заказ №dd);
- сообщение о смене статуса заказа, либо о создании заказа;

- от кого отправляем: адрес корпоративной почты (берем из настроек системы).

С проверкой почты сложнее. По-хорошему ее нужно делать периодически. Тут несколько вариантов:

1. Сделать отдельный сервис, который встроить в ОС, чтобы он постоянно там крутился.
2. В приложении (desktop или RestAPI) сделать метод, который бы периодически вызывался и проверял почту на предмет новых писем.

Остановимся на втором варианте. Заведем таймер и будем периодически вызывать в фоновом потоке метод для проверки почты, пока desktop-приложение запущено. Новые письма будет сохранять в системе, для чего заведем новую сущность «Информация о письме». В сущности, будем хранить информацию:

- кто прислал письмо (адрес почты отправителя, и, по возможности, найти клиента с таким адресом);
- когда было прислано письмо;
- заголовок письма:
- текст письма.

Для новой сущности потребуется сделать стандартный набор:

- модели;
- интерфейс и реализации логики (не забыть про поиск клиента по логину);
- интерфейс и 3 реализации для разных видов хранилищ;
- формы для вывода писем в desktop-приложении и в web-приложении.

Важный нюанс, в логике достаточно прописать методы получения списка и добавление новой записи. Методов получения отдельной записи, редактирования, удаления в интерфейсе для хранилища также не потребуются, так как будем только вытаскивать письма с почтового сервера и сохранять у

себя для вывода клиенту (только его письма, так что фильтрация нужна будет) и всего списка для вывода в desktop-приложении.

Для подключения к почтовому сервису потребуется где-то хранить адрес корпоративной почты и пароль к ней. Прописывать его жестко в код – плохая идея, так как в случае смены почты или ее пароля потребуется пересборка проекта и повторное ее разворачивание на серверах, что может занимать много ресурсов и времени. Правильнее будет в конфигурационных файлах проектов впивать требуемые данные, тем самым позволив их менять, когда будет необходимость без пересборки проекта. Для desktop-приложения это будет файл App.config (листинг 7.3), а для RestAPI-приложения файл appsettings.json (листинг 7.4).

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
      <appSettings>
            <add key="SmtpClientHost" value="smtp.gmail.com" />
            <add key="SmtpClientPort" value="587" />
            <add key="PopHost" value="pop.gmail.com" />
            <add key="PopPort" value="995" />
            <add key="MailLogin" value="labwork15kafis@gmail.com" />
            <add key="MailPassword" value="passlab15" />
      </appSettings>
</configuration>
```

Листинг 7.3 – Файл App.config

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",

  "SmtpClientHost": "smtp.gmail.com",
  "SmtpClientPort": "587",
  "PopHost": "pop.gmail.com",
  "PopPort": "995",
  "MailLogin": "labwork15kafis@gmail.com",
  "MailPassword": "passlab15"
}
```

Листинг 7.4 – Файл appsettings.json

Последнее, то потребуется сделать – ввести проверки для клиента. Проверяться будут логин и пароль. Для проверки логина достаточно в интернете найти пример шаблона проверки адреса электронной почты, это довольно распространенный шаблон, трудностей не должно возникнуть. Для

проверки почты потребуется самостоятельно составить шаблон. Необходимо проверять, что строка содержит именно все 3 необходимых типа символа (цифр, букв и небуквенных символов). Составим такое выражение (листинг 7.5)

```
^((\w+\d+\W+)|(\w+\W+\d+)|(\d+\w+\W+)|(\d+\W+\w+)|(\W+\w+\d+)|(\W+\d+\w+))[\w\d\W]*$
```

Листинг 7.5 – Регулярное выражение для поиска цифр, букв и небуквенных символов

В начале идет указание, что и поиск начинаем с начала строки (символ ^), далее перебираем возможные комбинации чередования типов символов, затем указание, что до конца строки также возможно (но не обязательно) что-то из этих символов.

Последнее, что требуется сделать – это определиться, где будет выполняться проверка. Есть 2 возможных кандидата:

- в web-приложении при регистрации клиента и изменении данных;
- в бизнес-логике при добавлении/изменении клиента.

Первый вариант позволяет экономить время, так как проверка будет выполнятся еще на стороне клиента и, если она не пройдет, то запрос в логику даже не отправится. Второй вариант позволит гарантировано не допустить сохранения не валидных данных, не важно с какого приложения они придут (если, например, в дальнейшем для клиента разработают иное приложение, типа мобильного). Остановимся на втором варианта и в логике обработки клиента добавим проверки.

**Реализация**

Создадим интерфейс модели-письма (листинг 7.6).

```
namespace AbstractShopDataModels.Models
{
    public interface IMessageInfoModel
    {
        string MessageId { get; }

        int? ClientId { get; }
```

```
        string SenderName { get; }

        DateTime DateDelivery { get; }

        string Subject { get; }

        string Body { get; }
    }
}
```

Листинг 7.6 – Интерфейс IMessageInfoModel

Далее все также, что для «Клиента» и «Исполнителя». Потребуется создать binding и view-модели, интерфейс бизнес-логики (листинг 7.7) и ее реализацию, интерфейс работы с данными (листинг 7.8) и 3 реализации, 3 модели для хранения данных, формы для desktop-приложения (разработать самостоятельно). Также потребуется в логике «Клиента» сделать получение записи по логину (разработать самостоятельно).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.BusinessLogicsContracts
{
    public interface IMessageInfoLogic
    {
        List<MessageInfoViewModel>? ReadList(MessageInfoSearchModel? model);

        bool Create(MessageInfoBindingModel model);
    }
}
```

Листинг 7.7 – Интерфейс IMessageInfoLogic

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;

namespace AbstractShopContracts.StoragesContracts
{
    public interface IMessageInfoStorage
    {
        List<MessageInfoViewModel> GetFullList();

        List<MessageInfoViewModel> GetFilteredList(MessageInfoSearchModel model);

        MessageInfoViewModel? GetElement(MessageInfoSearchModel model);

        MessageInfoViewModel? Insert(MessageInfoBindingModel model);
    }
}
```

Листинг 7.8 – Интерфейс IMessageInfoStorage

Далее сделаем класс с логикой отправки писем и проверки почты. При отправке писем и проверки почты будут ряд данных (хост для отправки, порт

для отправки, хост для проверки, порт для проверки, логин и пароль). Так что сделаем отдельный метод для получения этих данных. И отдельный метод для отправки письма (чтобы туда передавалось только адресат и текст письма). Получится два класса для передачи данных (листинг 7.9 – 7.10).

```
namespace AbstractShopContracts.BindingModels
{
    public class MailConfigBindingModel
    {
        public string MailLogin { get; set; } = string.Empty;

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

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

        public int SmtpClientPort { get; set; }

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

        public int PopPort { get; set; }
    }
}
```

Листинг 7.9 – Класс MailConfigBindingModel

```
namespace AbstractShopContracts.BindingModels
{
    public class MailSendInfoBindingModel
    {
        public string MailAddress { get; set; } = string.Empty;

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

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

Листинг 7.10 – Класс MailSendInfoBindingModel

Так как способов отправки письма и проверки почты существует много, то сделаем стандартный вариант, абстрактный класс с различными проверками и сохранением писем в хранилище и реализации для непосредственной отправки писем и получением новых писем (листинг 7.11).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using Microsoft.Extensions.Logging;

namespace AbstractShopBusinessLogic.MailWorker
{
    public abstract class AbstractMailWorker
    {
        protected string _mailLogin = string.Empty;

        protected string _mailPassword = string.Empty;

        protected string _smtpClientHost = string.Empty;
```

```
        protected int _smtpClientPort;

        protected string _popHost = string.Empty;

        protected int _popPort;

        private readonly IMessageInfoLogic _messageInfoLogic;

        private readonly ILogger _logger;

        public AbstractMailWorker(ILogger<AbstractMailWorker> logger,
IMessageInfoLogic messageInfoLogic)
        {
            _logger = logger;
            _messageInfoLogic = messageInfoLogic;
        }

        public void MailConfig(MailConfigBindingModel config)
        {
            _mailLogin = config.MailLogin;
            _mailPassword = config.MailPassword;
            _smtpClientHost = config.SmtpClientHost;
            _smtpClientPort = config.SmtpClientPort;
            _popHost = config.PopHost;
            _popPort = config.PopPort;
            _logger.LogDebug("Config: {login}, {password}, {clientHost},
{clientPOrt}, {popHost}, {popPort}", _mailLogin, _mailPassword, _smtpClientHost,
_smtpClientPort, _popHost, _popPort);
        }

        public async void MailSendAsync(MailSendInfoBindingModel info)
        {
            if (string.IsNullOrEmpty(_mailLogin) ||
string.IsNullOrEmpty(_mailPassword))
            {
                return;
            }

            if (string.IsNullOrEmpty(_smtpClientHost) || _smtpClientPort == 0)
            {
                return;
            }

            if (string.IsNullOrEmpty(info.MailAddress) ||
string.IsNullOrEmpty(info.Subject) || string.IsNullOrEmpty(info.Text))
            {
                return;
            }

            _logger.LogDebug("Send Mail: {To}, {Subject}", info.MailAddress,
info.Subject);
            await SendMailAsync(info);
        }

        public async void MailCheck()
        {
            if (string.IsNullOrEmpty(_mailLogin) ||
string.IsNullOrEmpty(_mailPassword))
            {
                return;
            }

            if (string.IsNullOrEmpty(_popHost) || _popPort == 0)
            {
                return;
```

```
            }

            if (_messageInfoLogic == null)
            {
                return;
            }

            var list = await ReceiveMailAsync();
            _logger.LogDebug("Check Mail: {Count} new mails", list.Count);
            foreach (var mail in list)
            {
                _messageInfoLogic.Create(mail);
            }
        }

        protected abstract Task SendMailAsync(MailSendInfoBindingModel info);

        protected abstract Task<List<MessageInfoBindingModel>>
ReceiveMailAsync();
    }
}
```

Листинг 7.11 – Абстраткный класс AbstractMailWorker

Сделаем реализацию этого класса. Передаем туда хост и порт почтового сервера, логин и пароль почты для отправки, а также адрес получателя, заголовок письма и его текст. Обе операции будем выполнять асинхронно, чтобы не задерживать основную работу программы (листинг 7.12).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using MailKit.Net.Pop3;
using MailKit.Security;
using Microsoft.Extensions.Logging;
using System.Net;
using System.Net.Mail;
using System.Text;

namespace AbstractShopBusinessLogic.MailWorker
{
    public class MailKitWorker : AbstractMailWorker
    {
        public MailKitWorker(ILogger<MailKitWorker> logger, IMessageInfoLogic
messageInfoLogic) : base(logger, messageInfoLogic) { }

        protected override async Task SendMailAsync(MailSendInfoBindingModel
info)
        {
            using var objMailMessage = new MailMessage();
            using var objSmtpClient = new SmtpClient(_smtpClientHost,
_smtpClientPort);
            try
            {
                objMailMessage.From = new MailAddress(_mailLogin);
                objMailMessage.To.Add(new MailAddress(info.MailAddress));
                objMailMessage.Subject = info.Subject;
                objMailMessage.Body = info.Text;
                objMailMessage.SubjectEncoding = Encoding.UTF8;
                objMailMessage.BodyEncoding = Encoding.UTF8;

                objSmtpClient.UseDefaultCredentials = false;
                objSmtpClient.EnableSsl = true;
```

```
                objSmtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;
                objSmtpClient.Credentials = new NetworkCredential(_mailLogin,
_mailPassword);

                await Task.Run(() => objSmtpClient.Send(objMailMessage));
            }
            catch (Exception)
            {
                throw;
            }
        }

        protected override async Task<List<MessageInfoBindingModel>>
ReceiveMailAsync()
        {
            var list = new List<MessageInfoBindingModel>();
            using var client = new Pop3Client();
            await Task.Run(() =>
            {
                try
                {
                    client.Connect(_popHost, _popPort,
SecureSocketOptions.SslOnConnect);
                    client.Authenticate(_mailLogin, _mailPassword);
                    for (int i = 0; i < client.Count; i++)
                    {
                        var message = client.GetMessage(i);
                        foreach (var mail in message.From.Mailboxes)
                        {
                            list.Add(new MessageInfoBindingModel
                            {
                                DateDelivery = message.Date.DateTime,
                                MessageId = message.MessageId,
                                SenderName = mail.Address,
                                Subject = message.Subject,
                                Body = message.TextBody
                            });
                        }
                    }
                }
                catch (AuthenticationException)
                { }
                finally
                {
                    client.Disconnect(true);
                }
            });
            return list;
        }
    }
}
```

Листинг 7.12 – Класс MailKitWorker

Необходимо дополнить логику класса OrderLogic. Для отправки писем потребуется адрес электронной почты клиента. Для получения почты нам потребуется интерфейс для клиента, чтобы получить нужного клиента по идентификатору и вытащить адрес его электронной почты (разработать самостоятельно).

В проекте **AbstractShopView** делаем следующее:

1. В конфигурации (добавляем файл App.config, если его нет) прописываем настройки для подключения к почтовому сервису (листинг 7.3).

2. В классе Program пропишем логику для ввода настроек для отправки писем. Проверку почты будем осуществлять по таймеру (через определенное время), чтобы с одной стороны не вешать эту операцию на пользователя программы, а с другой, чтобы была возможность иметь актуальный список писем без перезапуска программы (листинг 7.13).

В методе ConfigureServices не забудем прописать новые интерфейсы и их реализации. Важный нюанс, для AbstractMailWorker выставим время жизни – один объект на все время программы (AddSingleton), чтобы в Program загрузить в него настройки из конфигурации и потом спокойно вызывать в OrderLogic. Для таймера сделаем метод, который он будет вызывать (проверка почты) и настроем периодичность вызова с интервалом в 100000 миллисекунд. Метод будет работать асинхронно, так что не скажется на работе основной программы.

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

namespace AbstractShopView
{
    internal static class Program
    {
            private static ServiceProvider? _serviceProvider;
            public static ServiceProvider? ServiceProvider => _serviceProvider;
            /// <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();
            ConfigureServices(services);
                  _serviceProvider = services.BuildServiceProvider();

                  try
                  {
                        var mailSender =
_serviceProvider.GetService<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 = _serviceProvider.GetService<ILogger>();
                        logger?.LogError(ex, "Ошибка работы с почтой");
                  }



      Application.Run(_serviceProvider.GetRequiredService<FormMain>());
             }

             private static void ConfigureServices(ServiceCollection services)
             {
                  services.AddLogging(option =>
                              {

      option.SetMinimumLevel(LogLevel.Information);
                                    option.AddNLog("nlog.config");
                              });
                  services.AddTransient<IComponentStorage, ComponentStorage>();
                  services.AddTransient<IOrderStorage, OrderStorage>();
                  services.AddTransient<IProductStorage, ProductStorage>();
                  services.AddTransient<IClientStorage, ClientStorage>();
                  services.AddTransient<IImplementerStorage,
ImplementerStorage>();
                  services.AddTransient<IMessageInfoStorage,
MessageInfoStorage>();

                  services.AddTransient<IComponentLogic, ComponentLogic>();
```

```
                    services.AddTransient<IOrderLogic, OrderLogic>();
                    services.AddTransient<IProductLogic, ProductLogic>();
                    services.AddTransient<IReportLogic, ReportLogic>();
                    services.AddTransient<IClientLogic, ClientLogic>();
                    services.AddTransient<IImplementerLogic, ImplementerLogic>();
                    services.AddTransient<IMessageInfoLogic, MessageInfoLogic>();

                    services.AddTransient<AbstractSaveToExcel, SaveToExcel>();
                    services.AddTransient<AbstractSaveToWord, SaveToWord>();
                    services.AddTransient<AbstractSaveToPdf, SaveToPdf>();

                    services.AddTransient<IWorkProcess, WorkModeling>();
                    services.AddSingleton<AbstractMailWorker, MailKitWorker>();

                    services.AddTransient<FormMain>();
                    services.AddTransient<FormComponent>();
                    services.AddTransient<FormComponents>();
                    services.AddTransient<FormCreateOrder>();
                    services.AddTransient<FormProduct>();
                    services.AddTransient<FormProductComponent>();
                    services.AddTransient<FormProducts>();
                    services.AddTransient<FormReportProductComponents>();
                    services.AddTransient<FormReportOrders>();
                    services.AddTransient<FormClients>();
                    services.AddTransient<FormImplementer>();
                    services.AddTransient<FormImplementers>();
                    services.AddTransient<FormMails>();
                }

                private static void MailCheck(object obj) =>
ServiceProvider?.GetService<AbstractMailWorker>()?.MailCheck();
        }
}
```

Листинг 7.13 – Класс Program

3. Добавим форму для отображения всех писем. Там будет только dataGridView для вывода писем, без какого-либо дополнительного функционала (разработать самостоятельно). На главной форме сделать пункт меню для вызова новой формы (разработать самостоятельно).

В проекте **AbstractShopRestApi** в контроллере ClientController потребуется метод для получения списка писем клиента (листинг 7.14). В проекте **AbstractShowClientApp** создать представление для вывода писем клиента (листинг 7.15), метод в HomeController (листинг 7.16) для его вызова и в _Layout добавить новый пункт меню (листинг 7.17).

```
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
using AbstractShopContracts.SearchModels;
using AbstractShopContracts.ViewModels;
using Microsoft.AspNetCore.Mvc;

namespace AbstractShopRestApi.Controllers
{
```

```
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class ClientController : Controller
    {
        private readonly ILogger _logger;

        private readonly IClientLogic _logic;

        private readonly IMessageInfoLogic _mailLogic;

        public ClientController(IClientLogic logic, IMessageInfoLogic mailLogic,
ILogger<ClientController> logger)
        {
            _logger = logger;
            _logic = logic;
            _mailLogic = mailLogic;
        }

        [HttpGet]
        public ClientViewModel? Login(string login, string password)
        {
            try
            {
                return _logic.ReadElement(new ClientSearchModel
                {
                    Email = login,
                    Password = password
                });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка входа в систему");
                throw;
            }
        }

        [HttpPost]
        public void Register(ClientBindingModel model)
        {
            try
            {
                _logic.Create(model);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка регистрации");
                throw;
            }
        }

        [HttpPost]
        public void UpdateData(ClientBindingModel model)
        {
            try
            {
                _logic.Update(model);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка обновления данных");
                throw;
            }
        }

        [HttpGet]
```

```
        public List<MessageInfoViewModel>? GetMessages(int clientId)
        {
            try
            {
                return _mailLogic.ReadList(new MessageInfoSearchModel
                {
                    ClientId = clientId
                });
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Ошибка получения писем клиента");
                throw;
            }
        }
    }
}
```

Листинг 7.14 – Контроллер ClientController с новым методом

```
@using AbstractShopContracts.ViewModels

@model List<MessageInfoViewModel>

@{
	ViewData["Title"] = "Mails";
}

<div class="text-center">
	<h1 class="display-4">Заказы</h1>
</div>


<div class="text-center">
	@{
		if (Model == null)
		{
			<h3 class="display-4">Авторизируйтесь</h3>
			return;
		}

		<table class="table">
			<thead>
				<tr>
					<th>
						Дата письма
					</th>
					<th>
						Заголовок
					</th>
					<th>
						Текст
					</th>
				</tr>
			</thead>
			<tbody>
				@foreach (var item in Model)
				{
					<tr>
						<td>
							@Html.DisplayFor(modelItem =>
item.DateDelivery)
						</td>
						<td>
							@Html.DisplayFor(modelItem =>
item.Subject)
```

```
                                        </td>
                                        <td>
                                                @Html.DisplayFor(modelItem =>
item.Body)
                                        </td>
                                </tr>
                        }
                </tbody>
        </table>
    }
</div>
```

Листинг 7.15 – Представление для вывода писем

```
        [HttpGet]
        public IActionResult Mails()
        {
                if (APIClient.Client == null)
                {
                        return Redirect("~/Home/Enter");
                }
                return
View(APIClient.GetRequest<List<MessageInfoViewModel>>($"api/client/getmessages?cl
ientId={APIClient.Client.Id}"));
        }
```

Листинг 7.16 – Новый метод в HomeController

```
                        <ul class="navbar-nav flex-grow-1">
                                <li class="nav-item">
                                        <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Index">Заказы</a>
                                </li>
                                <li class="nav-item">
                                        <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Privacy">Личные данные</a>
                                </li>
                                <li class="nav-item">
                                        <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Mails">Письма</a>
                                </li>
                                <li class="nav-item">
                                        <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Enter">Вход</a>
                                </li>
                                <li class="nav-item">
                                        <a class="nav-link text-dark" asp-
area="" asp-controller="Home" asp-action="Register">Регистрация</a>
                                </li>
                        </ul>
```

Листинг 7.17 – Обновленный вид меню в Layout.cshtml

Также, для отправки письма при создании заказа через **AbstractShowClientApp** надо прописать настройки почты в проекте **AbstractShopRestApi**. Сами настройки прописываются в файле appsettings.json (листинг 7.4), а их подгрузка в классе Program (листинг 7.18).

```
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.MailWorker;
using AbstractShopContracts.BindingModels;
using AbstractShopContracts.BusinessLogicsContracts;
```

```
using AbstractShopContracts.StoragesContracts;
using AbstractShopDatabaseImplement.Implements;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Logging.SetMinimumLevel(LogLevel.Trace);
builder.Logging.AddLog4Net("log4net.config");

// Add services to the container.


builder.Services.AddTransient<IClientStorage, ClientStorage>();
builder.Services.AddTransient<IOrderStorage, OrderStorage>();
builder.Services.AddTransient<IProductStorage, ProductStorage>();
builder.Services.AddTransient<IImplementerStorage, ImplementerStorage>();
builder.Services.AddTransient<IMessageInfoStorage, MessageInfoStorage>();

builder.Services.AddTransient<IOrderLogic, OrderLogic>();
builder.Services.AddTransient<IClientLogic, ClientLogic>();
builder.Services.AddTransient<IProductLogic, ProductLogic>();
builder.Services.AddTransient<IImplementerLogic, ImplementerLogic>();
builder.Services.AddTransient<IMessageInfoLogic, MessageInfoLogic>();

builder.Services.AddTransient<AbstractMailWorker, MailKitWorker>();

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo { Title = "AbstractShopRestApi", Version =
"v1" });
});


var app = builder.Build();
var mailSender = app.Services.GetService<AbstractMailWorker>();
mailSender?.MailConfig(new MailConfigBindingModel
{
    MailLogin = builder.Configuration?.GetSection("MailLogin")?.Value?.ToString()
?? string.Empty,
    MailPassword =
builder.Configuration?.GetSection("MailPassword")?.Value?.ToString() ??
string.Empty,
    SmtpClientHost =
builder.Configuration?.GetSection("SmtpClientHost")?.Value?.ToString() ??
string.Empty,
    SmtpClientPort =
Convert.ToInt32(builder.Configuration?.GetSection("SmtpClientPort")?.Value?.ToStr
ing()),
    PopHost = builder.Configuration?.GetSection("PopHost")?.Value?.ToString() ??
string.Empty,
    PopPort =
Convert.ToInt32(builder.Configuration?.GetSection("PopPort")?.Value?.ToString())
});

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
"AbstractShopRestApi v1"));
}
```

```
app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Листинг 7.18 – Класс Program

Остается сделать проверку данных при регистрации новых клиентов, либо смене их данных. Проверку будем делать в классе-логике по работе с клиентами (разработать самостоятельно).

Для проверки работоспособности потребуется клиент с почтой, к которой у вас есть доступ. На нее должны приходить сообщения, об изменениях. Также должна быть проверка, что письма от клиентов приходят и распознаются.

**Контрольный пример**

Заводим клиента в web-приложении с реальной почтой. Создаем заказ от имени этого клиента (рисунок 7.1).

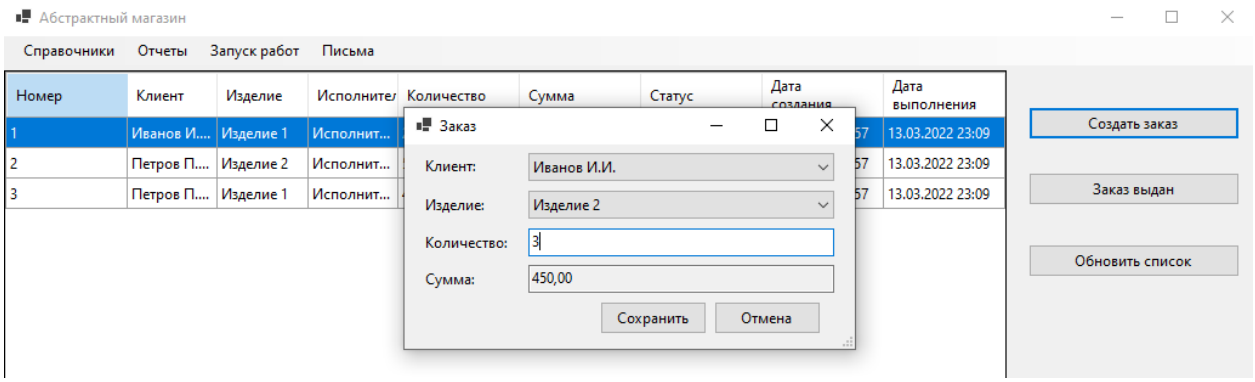


Рисунок 7.1 – Создание заказа

Создается новый заказ (рисунок 7.2).

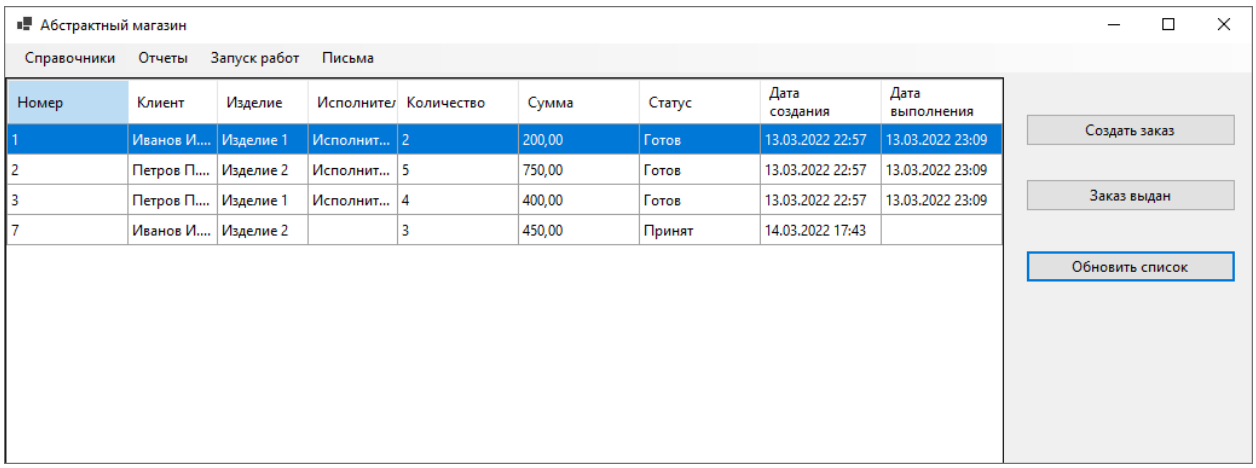


Рисунок 7.2 – Список с новым заказом

А на почту приходить письмо (рисунок 7.3).

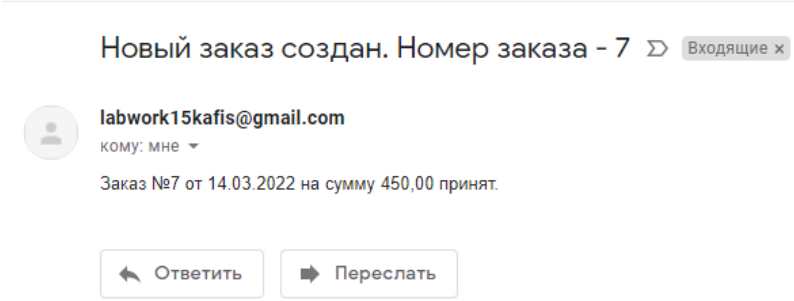


Рисунок 7.3 – Письмо о создании заказа на почте

Далее запускаем имитацию работы, после которой заказ оказывается в статусе «Готов» (рисунок 7.4).

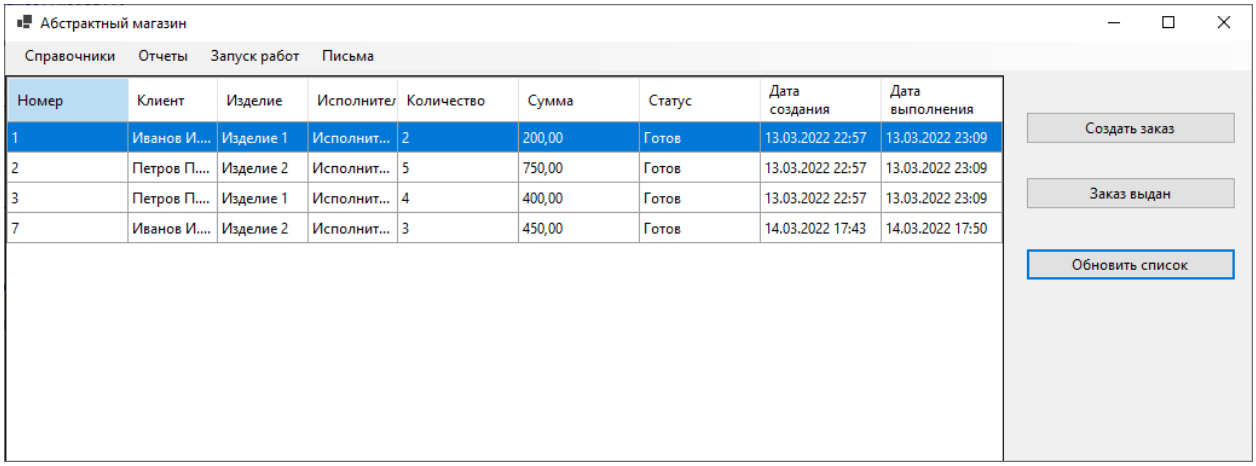


Рисунок 7.4 – Список с готовым заказом

После чего на почте оказываются новые письма (рисунок 7.5).

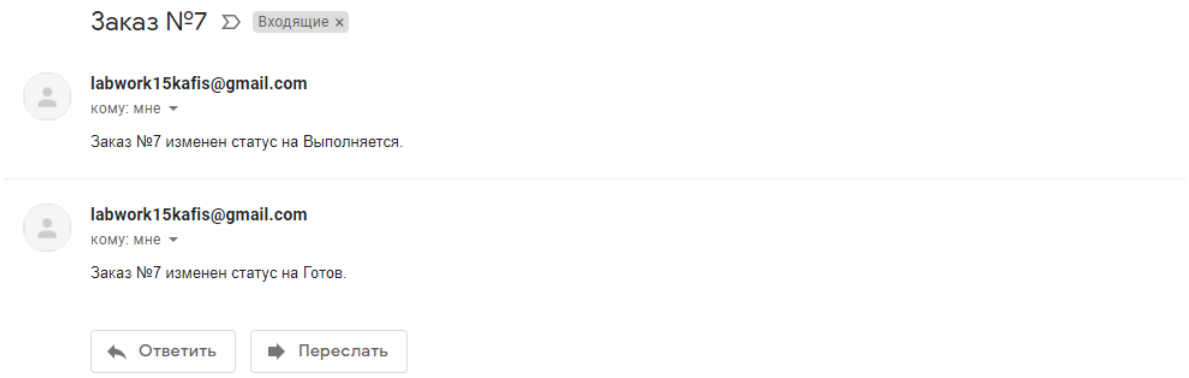


Рисунок 7.5 – Новые письма о смене статуса заказа на почте

Далее отправим с этой почты письмо-ответ (рисунок 7.6).

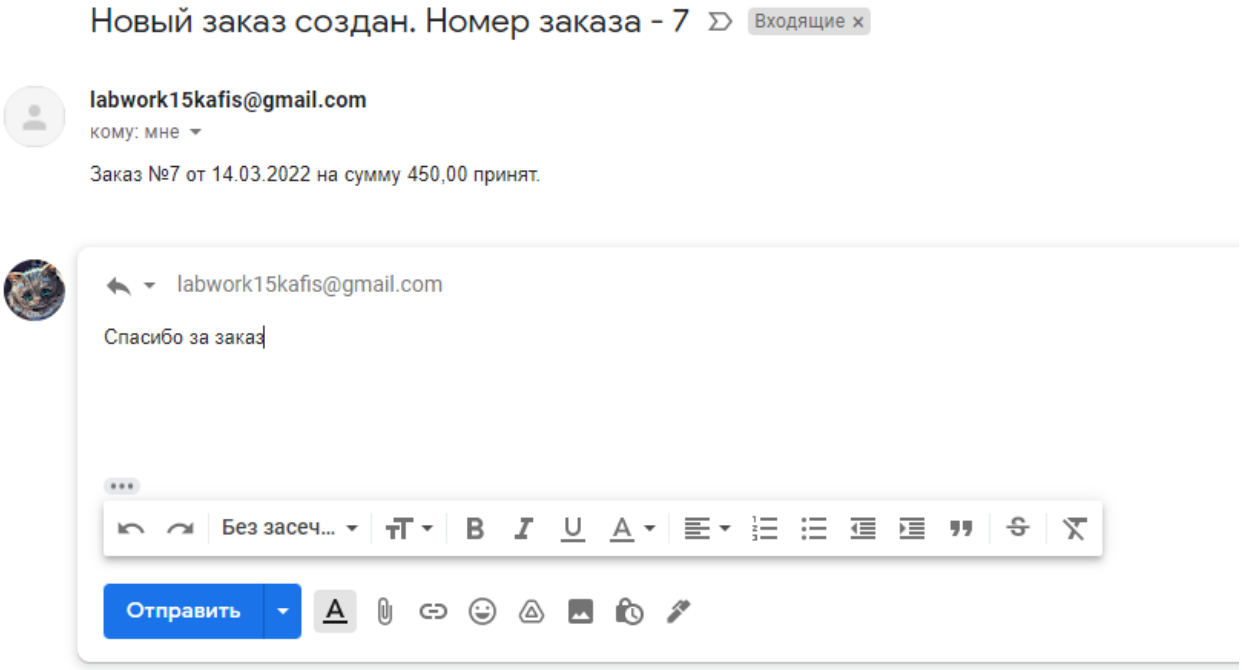


Рисунок 7.6 – Ответное письмо

Проверяем почту в desktop-приложении (рисунок 7.7).

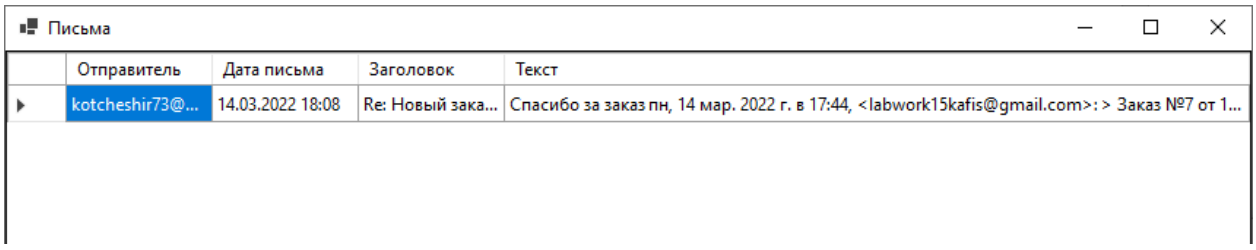


Рисунок 7.7 – Полученное письмо

И проверяем почту в web-приложении (рисунок 7.8).

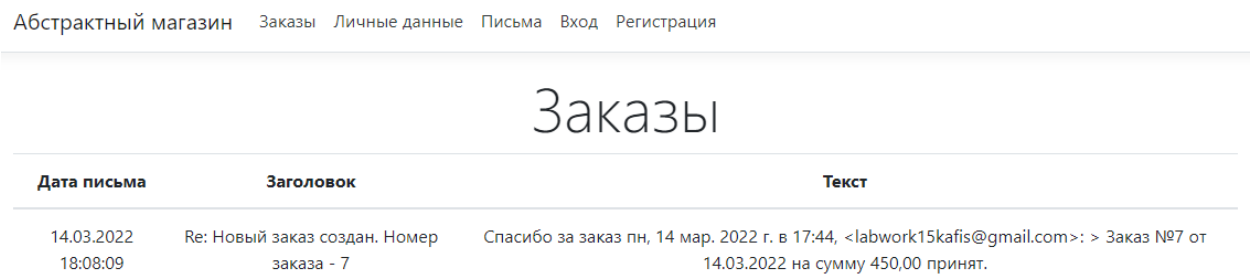


Рисунок 7.7 – Полученное письмо в web-приложении

Как видим, письмо не только успешно добавлено в приложение, но и привязано к конкретному пользователю.

**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. Сделать binding и view-модели для информации о письмах, интерфейс бизнес-логики и интерфейс работы с данными.
6. Сделать реализацию IMessageInfoLogic.
7. Сделать реализацию моделей и интерфейса IMessageInfoStorage для хранилищ.
8. Дополнить логику OrderLogic отправкой писем клиенту.
9. Добавить условие выборки клиента по логину.
10. Сделать форму для вывода писем в desktop-приложении.

11. Добавить проверки логина и пароля при сохранении клиента.

**Порядок сдачи базовой части**

1. Запустить Web-проект, создать там заказ.
2. Открыть почту и показать, что письмо с информацией о созданном заказе пришло.
3. Запустить Desktop-проект, вызвать «Запуск работ».
4. Вернуться в почту и показать, что письмо с информацией об изменении статуса заказа пришло.
5. Показать на почте письмо, отосланное на корпоративную почту.
6. В Desktop-проекте показать, что это письмо было обработано и добавлено в базу.
7. Ответить на вопрос преподавателя.

**Контрольные вопросы к базовой части**

1. Как реализована отправка писем при смене статусов заказа?
2. Как реализована проверка новых писем?
3. Как реализован выводи писем для конкретного клиента?

**Усложненная лабораторная (необязательно)**

1. Реализовать пагинацию для писем в desktop-проекте.
2. Реализовать пагинацию для писем в web-проекте.
3. У письма добавить 2 поле: отметка, что оно прочитано и ответ на письмо.
4. В интерфейсе IMessageInfoStorage добавить метод для редактирования письма (и в 3 реализациях его определить).
5. В desktop-проекте сделать форму для открытия письма (при открытии письмо должно помечаться, что оно прочитано) и возможности отправки на почту ответа на письмо (ответ должен сохраняться в письме).

**Порядок сдачи усложненной части**

1. Запустить Web-проект, показать там пагинацию.
2. Запустить Desktop-проект, показать там пагинацию.
3. Открыть непрочитанное письмо, написать на него ответ и закрыть письмо.
4. Показать, что у письма меняется отметка о том, что оно прочитано.
5. Показать на поте, что приходит ответ на письмо.
6. Ответить на вопрос преподавателя.

**Контрольные вопросы к усложненной части**

1. Как реализована пагинация в desktop-проекте?
2. Как реализована функция отметки о прочтении письма?
3. Как реализована пагинация в web-проекте?

**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).