МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО

«БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра «Информатика и программное обеспечение»

**Преддипломная практика**

Тема: «Проектирование программного комплекса управления надёжностью отправки электронных писем с реализацией основного механизма работы и пользовательского интерфейса»

Выполнил студент гр. о-17-ПРИ-рпс-б,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Сухарев Е.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Защищено с оценкой: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_к.т.н., доц. Лагерев Д.Г.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

БРЯНСК

2021

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Брянский Государственный Технический Университет**

*Кафедра «Информатика и программное обеспечение»*

**Задание**

на преддипломную практику

студенту гр. О-17-ПРИ-рпс-Б Сухареву Е.А.

Тема дипломной работы:

**Проектирование программного комплекса управления надёжностью отправки электронных писем с реализацией основного механизма работы и пользовательского интерфейса**

***Задачи на период прохождения практики:***

1. Оформить следующие разделы пояснительной записки в соответствии с требованиями методических указаний по выполнению ВКР:

* Техническое задание
  + Доработать функциональные требования.
* Проектирование и разработка программного продукта
  + Доработать архитектуру программного комплекса.
  + Доработать проектные модели программного комплекса.

1. Разработать следующие подсистемы (модули) программного комплекса:
   * Реализовать базу данных в СУБД MS SQL Server.

* Реализовать клиентскую часть
  + Все спроектированные интерфейсы.
  + Сервисы взаимодействия с web API.
* Реализовать серверную часть
  + Модуль Web API.
  + Модуль авторизации.
  + Модуль асинхронной отправки сообщений.

1. Выполнить тестирование разработанных подсистем (модулей).

Руководитель дипломного проектирования: к.т.н., доц. Трубаков А.О.

Руководитель преддипломной практики: к.т.н., доц. Лагерев Д.Г.

Студент гр. О-17-ПРИ-рпс-Б: Сухарев Е.А,

СОДЕРЖАНИЕ

[Введение 5](#_Toc71822664)

[1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 6](#_Toc71822665)

[1.1. Основание для разработки 6](#_Toc71822666)

[1.2. Назначение и область применения 6](#_Toc71822667)

[1.3. Требование к программному комплексу 6](#_Toc71822668)

[1.3.1. Требования к функциональным характеристикам 6](#_Toc71822669)

[1.3.2. Требования к надежности 9](#_Toc71822670)

[1.4. Условия эксплуатации 10](#_Toc71822671)

[1.4.1. Климатические условия эксплуатации 10](#_Toc71822672)

[1.4.2. Требования к квалификации и численности персонала 11](#_Toc71822673)

[1.4.3. Требования к составу и параметрам технических средств 11](#_Toc71822674)

[1.4.4. Требования к информационной и программной совместимости 11](#_Toc71822675)

[1.5. Программная документация 12](#_Toc71822676)

[1.5.1. Предварительный состав программной документации 12](#_Toc71822677)

[1.6. Стадии и этапы разработки 12](#_Toc71822678)

[1.6.1. Стадии разработки 12](#_Toc71822679)

[1.6.2. Содержание работ по этапам 12](#_Toc71822680)

[1.7. Порядок контроля и приемки 13](#_Toc71822681)

[1.7.1. Виды испытаний 13](#_Toc71822682)

[2. ПРОЕКТИРОВАНИЕ 14](#_Toc71822683)

[2.1. Клиентская часть 14](#_Toc71822684)

[2.2. Серверная часть 16](#_Toc71822685)

[2.3. Сервер СУБД 17](#_Toc71822686)

[3. РАЗРАБОТКА 19](#_Toc71822687)

[3.1. Клиентская часть 19](#_Toc71822688)

[3.2. Web API 21](#_Toc71822689)

[3.3. Асинхронная отправка сообщений 21](#_Toc71822690)

[4. ТЕСТИРОВАНИЕ 22](#_Toc71822691)

[ЗАКЛЮЧЕНИЕ 23](#_Toc71822692)

[СПИСОК ЛИТЕРАТУРЫ 24](#_Toc71822693)

[ПРИЛОЖЕНИЕ 26](#_Toc71822694)

# Введение

Учебная практика посвящена дальнейшей работе над дипломным проектом на **тему** «Проектирование программного комплекса управления надёжностью отправки электронных писем с реализацией основного механизма работы и пользовательского интерфейса».

**Цели** практики:

1. оформить пояснительную записку в соответствии с требованиями методических указаний по выполнению ВКР;
2. разработать основные модули программного комплекса;
3. выполнить тестирование разработанных подсистем.

**Задачи** практики:

1. доработать функциональные требования;
2. доработать архитектуру программного комплекса;
3. доработать проектные модели программного комплекса;
4. реализовать базу данных в СУБД MS SQL Server;
5. реализовать клиентскую часть;
6. реализовать серверную часть;
7. выполнить тестирование разработанных подсистем (модулей).

# ТЕХНИЧЕСКОЕ ЗАДАНИЕ

В рамках практики были доработаны функциональные требования разрабатываемого программного комплекса. Добавлены новые особенности:

Техническое задание включает в себя требования, необходимые для реализации программного продукта.

Название программного комплекса «Подсистема управления рассылкой электронных писем и их учёта».

## Основание для разработки

Основанием для разработки подсистемы управления надежностью отправки электронных писем является задание на дипломную работу, выданное доцентом Трубаковым А.О.

## Назначение и область применения

Разрабатываемый программный комплекс должен выполнять следующие задачи:

* асинхронная рассылка сообщений выбранным адресам с помощью выбранных сервисов доставки;
* обеспечение повышенной надежности доставки;
* возможность получения и просмотра подробной информации о статусе доставки сообщения.

## Требование к программному комплексу

### Требования к функциональным характеристикам

Реализуемый программный комплекс должен включать в себя нижеупомянутые функциональные особенности:

1. Авторизация пользователей в web-интерфейсе для администрирования:

Роли:

1. Оператор – может управлять рассылкой сообщений, просматривать соответствующую историю сообщений;
2. Администратор – имеет все возможности оператора, но также может просматривать список операторов, создавать новых операторов и удалять существующих.
3. После входа в Web-сервис пользователь может:
4. отправлять сообщения на электронные почтовые ящики;
5. выбирать сервисы доставки из списка доступных на данный момент;
6. прикреплять файлы к сообщению;
7. организовывать отправку сообщений по расписанию;
8. просматривать подробную информацию о доставке сообщений.
9. Разрабатываемый комплекс должен обладать следующими функциями:
10. разрабатываемый программный комплекс должен предоставлять возможность подключить свой почтовый ящик и отправлять сообщения через доступный SMTP-сервер;
11. если пользователь подключил свой почтовый ящик, то по умолчанию в первую очередь пытаться отправлять сообщение с помощью SMTP-сервера;
12. возможность повторной отправки недоставленного сообщения, но уже посредством другого доступного сервиса доставки при наличии Интернет-соединения;
13. возможность отправлять сообщения посредством минимум **трех** различных сервисов для отправки сообщений;
14. наличие API, с помощью которого подсистему можно будет встраивать в различные сервисы, использующие рассылку сообщений;
15. возможность прикреплять файлы к сообщению;
16. возможность хранить историю сообщений, т.е. хранить определенное число ранее отправленных сообщений и давать доступ к просмотру информации об их доставке;
17. возможность получения обратной связи - подсистема должна оповещать пользователя об удачной доставке или о причине неудачной отправки сообщения (*опционально:* уведомлять о прочтении доставленного сообщения), хранить эту информацию и давать возможность ее просматривать.
18. Разрабатываемый комплекс предоставляет асинхронную отправку сообщений. Когда оператор или администратор отправляет сообщение, сообщение отправляется в очередь на удаленном сервере. Система с настраиваемой периодичностью проверяет эту очередь и передает сообщения соответствующим сервисам доставки.
19. В целях улучшения надежности разрабатываемый комплекс должен включать в себя кластерную систему серверов.
20. Требования к интерфейсу

Реализуемый программный комплекс должен включать в себя нижеупомянутые функциональные особенности:

1. интерфейс для работы с подсистемой (интерфейс администратора) – web-страница, включает в себя список со всей информацией об отправленных сообщениях, предоставляет возможность вручную отправить сообщение через выбранный сервис, настраивать почтовые шлюзы;
2. необходимо обеспечить совместимость с основными браузерами (последние версии Chrome, Firefox, Safari, Opera; IE начиная с 10 версии). Верстка должна быть адаптивной и рассчитана на минимальное разрешение экрана 1280×720;
3. цветовая гамма приложения не должна быть излишне яркой, но сочетания цветов должны быть контрастными. Дизайн должен быть простым и понятным, нужно избегать непонятных иконок. Списки, с помощью которых также может быть реализовано меню, не должны превышать 5-9 элементов;
4. если время загрузки страницы составляет более 5с., необходимо обеспечить пользователю уверенность в том, что процесс действительно происходит при помощи индикатора выполнения процесса;
5. на длинных страницах необходимо применять ссылки, возвращающие пользователя в верхнюю часть страницы.

**Получение данных.** Система получает на вход:

1. тему;
2. тело сообщения;
3. прикрепленные файлы;
4. список адресов, на которые нужно отправить сообщения;
5. список сервисов доставки, посредством которых необходимо доставить сообщение;
6. запланированное время доставки или диапазоны времени, в пределах которых сообщение должно быть доставлено.

**Выходные данные.** Система предоставляет пользователю историю сообщений с информацией о доставке.

**Отправка данных.** Система обрабатывает полученные от пользователя данные через REST API и передает их указанным сервисам доставки.

### Требования к надежности

* + - 1. Требования к обеспечению надежного функционирования программы

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением Заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:

1. организацией бесперебойного питания технических средств;
2. использованием лицензионного программного обеспечения;
3. регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;
4. регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов.
   * + 1. Время восстановления после отказа

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать 30-ти минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

* + - 1. Отказы из-за некорректных действий оператора

Отказы программы возможны вследствие некорректных действий пользователя при взаимодействии с системой.

Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу оператора без предоставления ему административных привилегий, что реализовано посредством разделения ролей.

## Условия эксплуатации

### Климатические условия эксплуатации

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

### Требования к квалификации и численности персонала

Минимальное количество персонала, требуемого для работы программного комплекса, должно составлять не менее 1 человека: администратор, управляющий настройкой подсистемы.

### Требования к составу и параметрам технических средств

Подсистема функционирует в составе уже имеющегося сервиса и не предъявляет дополнительных требований к составу и параметру технических средств. Подсистема должна работать на тех же самых устройствах, что и вся система учета.

Сервер приложения:

* наличие веб-сервера IIS.
* процессор: 3,1 ГГц и выше;
* ОЗУ: 16Гб;
* SSD диск.

Сервер баз данных:

* процессор: 3,1 ГГц и выше;
* ОЗУ: 16Гб и выше;
* SSD диск.

### Требования к информационной и программной совместимости

В перечень программного обеспечения, работающего на стороне сервера входит следующее ПО:

* лицензионная версия операционной системы «Microsoft Windows Server 2019»;
* базы данных «Microsoft SQL Server 2019»;
* СУБД «SQL Server Management Studio»;
* web-браузер Google Chrome 89.0 или Safari 14, или Mozilla Firefox 91.6 ESRЮ, или Opera 12;
* антивирусное программное обеспечение.

Дополнительные требования к защите программного обеспечения и информации не предъявляются.

## Программная документация

### Предварительный состав программной документации

Состав программной документации:

* техническое задание;
* программа и методики испытаний;
* руководство оператора.

## Стадии и этапы разработки

### Стадии разработки

Разработка подсистемы состоит из следующих этапов:

* разработка технического задания;
* проектирование программного комплекса;
* разработка программного комплекса;
* тестирование разработанного программного комплекса;
* написание программной документации;
* внедрение разработанного сервиса.

### Содержание работ по этапам

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

* постановка задачи;
* определение и уточнение требований к техническим средствам;
* определение требований к программе;
* определение стадий, этапов и сроков разработки программы;
* согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.

Этап тестирования включает в себя выполнение тестов по указанным заранее шагам и сверку полученных результатов с указанными заранее ожидаемыми результатами.

Этап внедрения подразумевает развертывание разработанного программного комплекса на выделенном сервере с дальнейшим приведением подсистемы в стабильно работающее состояние. Результатом данного заключительного этапа является подписанный Заказчиком акт о внедрении.

## Порядок контроля и приемки

### Виды испытаний

Для проверки корректности работы разработанного программного комплекса должны быть проведены следующие виды испытаний:

1. Тестирование WEB-API методом черного ящика;
2. Модульное тестирование;
3. Кроссбраузерное тестирование.

В результате проведенного тестирования методом черного ящика и модульного тестирования все установленные заранее тесты должны быть пройдены успешно.

Кроссбраузерное тестирование должно подтвердить идентичное поведение разработанного программного комплекса в указанных в разделе «Требования к информационной и программной совместимости» браузерах.

Приемо-сдаточные испытания должны проводиться на объекте Заказчика в оговоренные сроки.

Приемо-сдаточные испытания программы должны проводиться согласно разработанной Исполнителем и согласованной Заказчиком Программы и методик испытаний.

Ход проведения приемо-сдаточных испытаний Заказчик и Исполнитель документируют в Протоколе проведения испытаний

# ПРОЕКТИРОВАНИЕ

В процессе проектирования было принято следующее: система должна быть централизованной, т.е. все данные должны располагаться в центральном хранилище. Система должна иметь трехуровневую архитектуру, состоящую из следующих уровней: первый - клиент, второй - сервер, третий – хранилище в связи с ее масштабируемостью, гибкой настройкой, высокой безопасностью и надежностью, а также наличием большого объема документации.

Схема архитектуры ПС представлена на рис. 2.1.

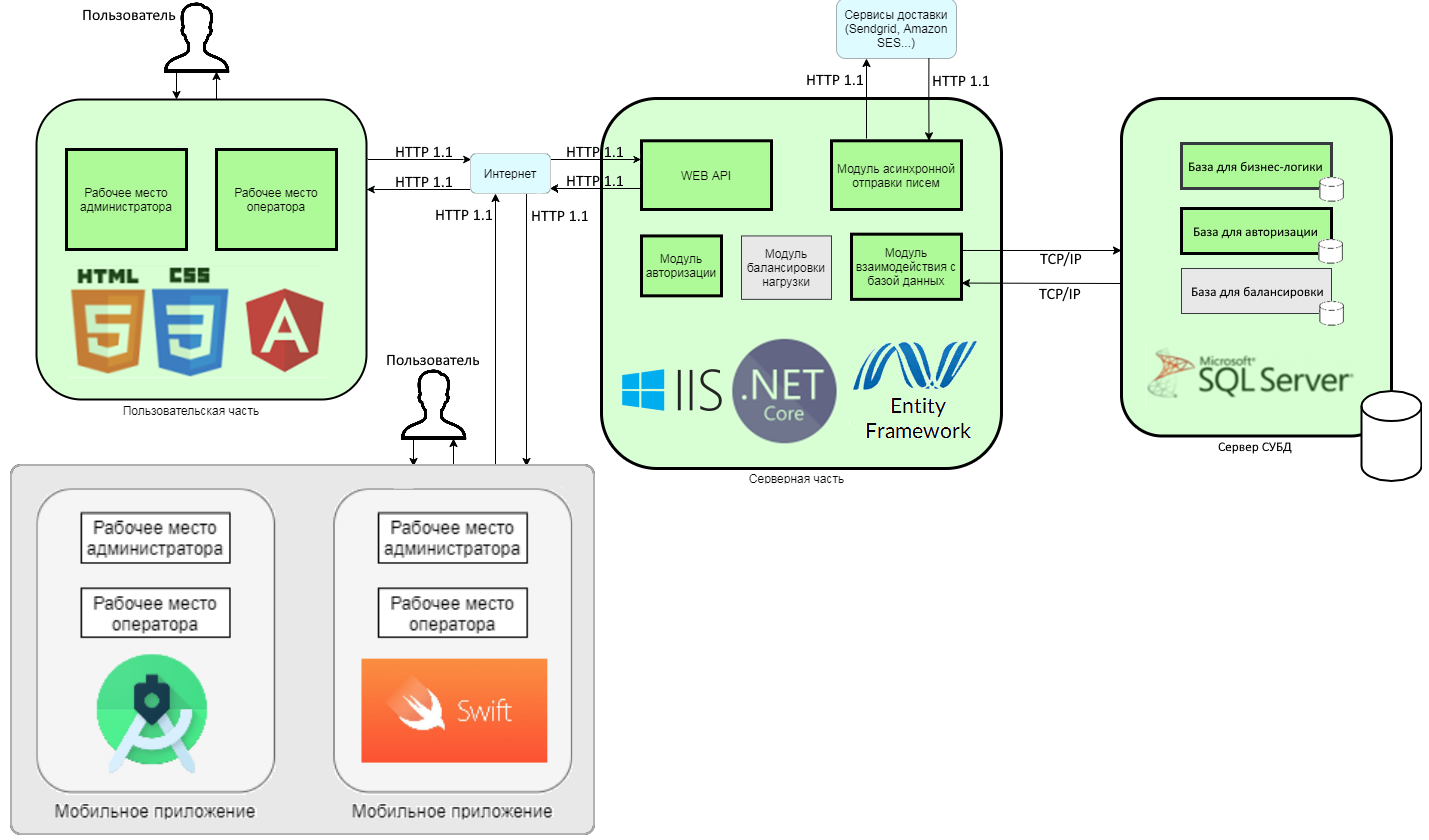
## Клиентская часть

В процессе анализа было рассмотрено несколько альтернативных инструментов разработки web-интерфейса:

1. React [25];
2. Angular [17];
3. Vue [26].

Среди выбранных фреймворков предпочтение было отдано Angular благодаря его декларативности, наличию развитого сообщества, модульности и MVC из коробки [17]. В качестве скриптового языка из-за ряда преимуществ над JavaScript был выбран TypeScript [19]. Для вёрстки будет применяться следующий набор инструментов: HTML + CSS [18], а также адаптивные компоненты Material и некоторые стили Bootstrap. Bootstrap и Material уже входят в Angular, поэтому отдельно подключать их не требуется.

|  |
| --- |
| ***Рис. 2.1. Схема архитектуры ПС*** |



## Серверная часть

Серверная часть включает в себя API, используемый для обработки запросов от клиентской части, набор функций для взаимодействия с хранилищем данных, модуль асинхронной отправки писем.

Серверная часть условно поделена на ряд модулей:

1. web API;
2. модуль авторизации;
3. модуль балансировки нагрузки;
4. модуль асинхронной отправки писем;
5. модуль взаимодействия с базой данных.

Web API служит для взаимодействия пользовательской части и серверной части. Он включает в себя методы получения, добавления, изменения и удаления пользователей, сообщений и других сущностей [8].

Модуль авторизации отвечает за процедуру подтверждения подлинности конкретного человека, блокировку пользователей, а также за управление ролями зарегистрированных в системе пользователей.

Модуль балансировки нагрузки обеспечивает корректную работу кластерной системы серверов: координирует работу нескольких серверов, позволяет проверять их работоспособность и эффективность их работы, назначать определенным единицам выбранные роли, собирает статистику запросов, позволяет задавать подходящий метод распределения нагрузки.

Модуль асинхронной отправки писем служит для взаимодействия с доступными сервисами доставки и организации отправки писем.

Модуль взаимодействия с базой данных предоставляет интерфейсы для взаимодействия с таблицами сущностей. Для разработки данного модуля был выбран Entity Framework Core.

В качестве инструмента разработки серверной части была выбрана технология .NET Core, благодаря ее кроссплатформенности и удобству языка C#, а также благодаря детальной технической документации от официального разработчика [9].

## Сервер СУБД

В ходе работы был проанализирован ряд различных СУБД, а именно:

1. MySQL [23];
2. MS SQL [22];
3. PostgreSQL [24].

Предпочтение было отдано MS SQL в связи с его масштабируемостью и надежностью, высокой скоростью создания решений, а также возможностью обработки вычислений в оперативной памяти (in-memory OLTP) [22].

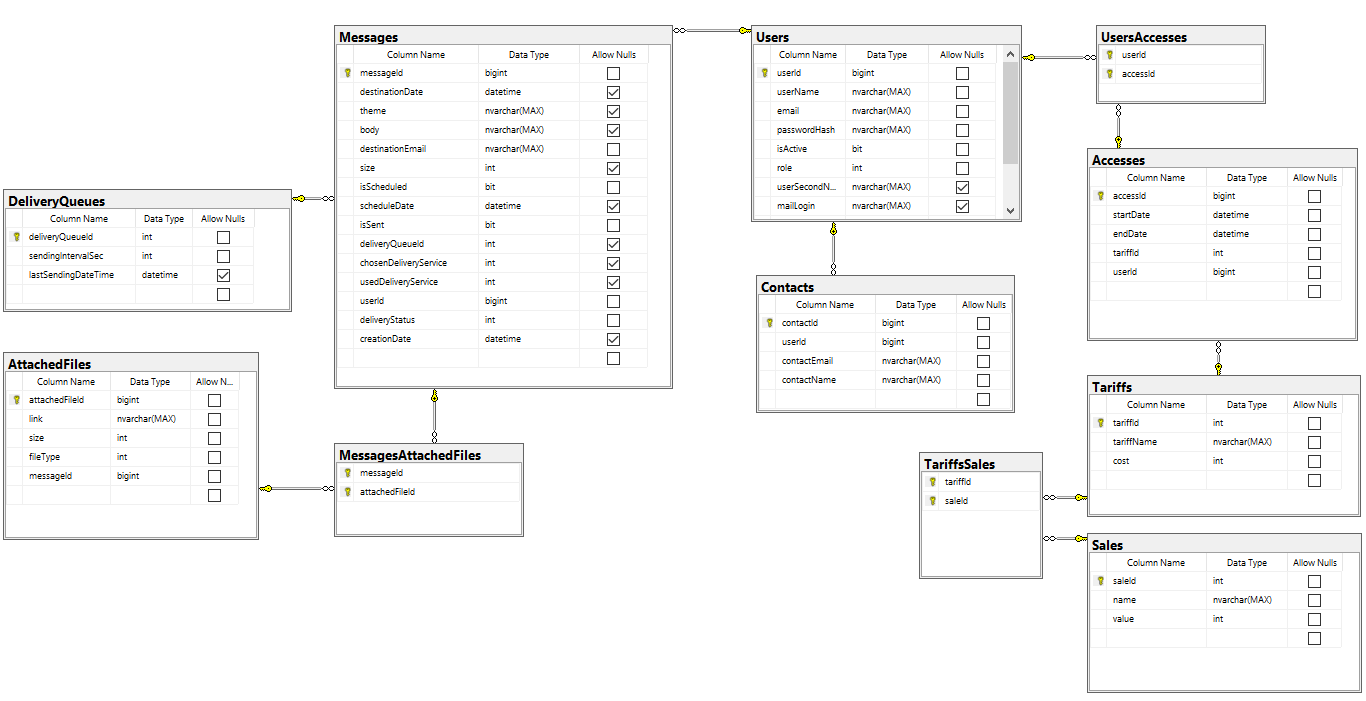
В базе данных будут храниться все основные сущности системы:

1. сообщения;
2. пользователи;
3. ссылки на прикрепленные файлы;
4. контакты пользователей;
5. сервисы доставки;
6. очереди доставки;
7. доступы пользователей;
8. тарифы;
9. скидки.

Также для осуществления отношения «многие ко многим» необходимы промежуточные таблицы для взаимодействия следующих таблиц:

* Сообщения – Файлы;
* Пользователи – Доступы;
* Тарифы – Скидки.

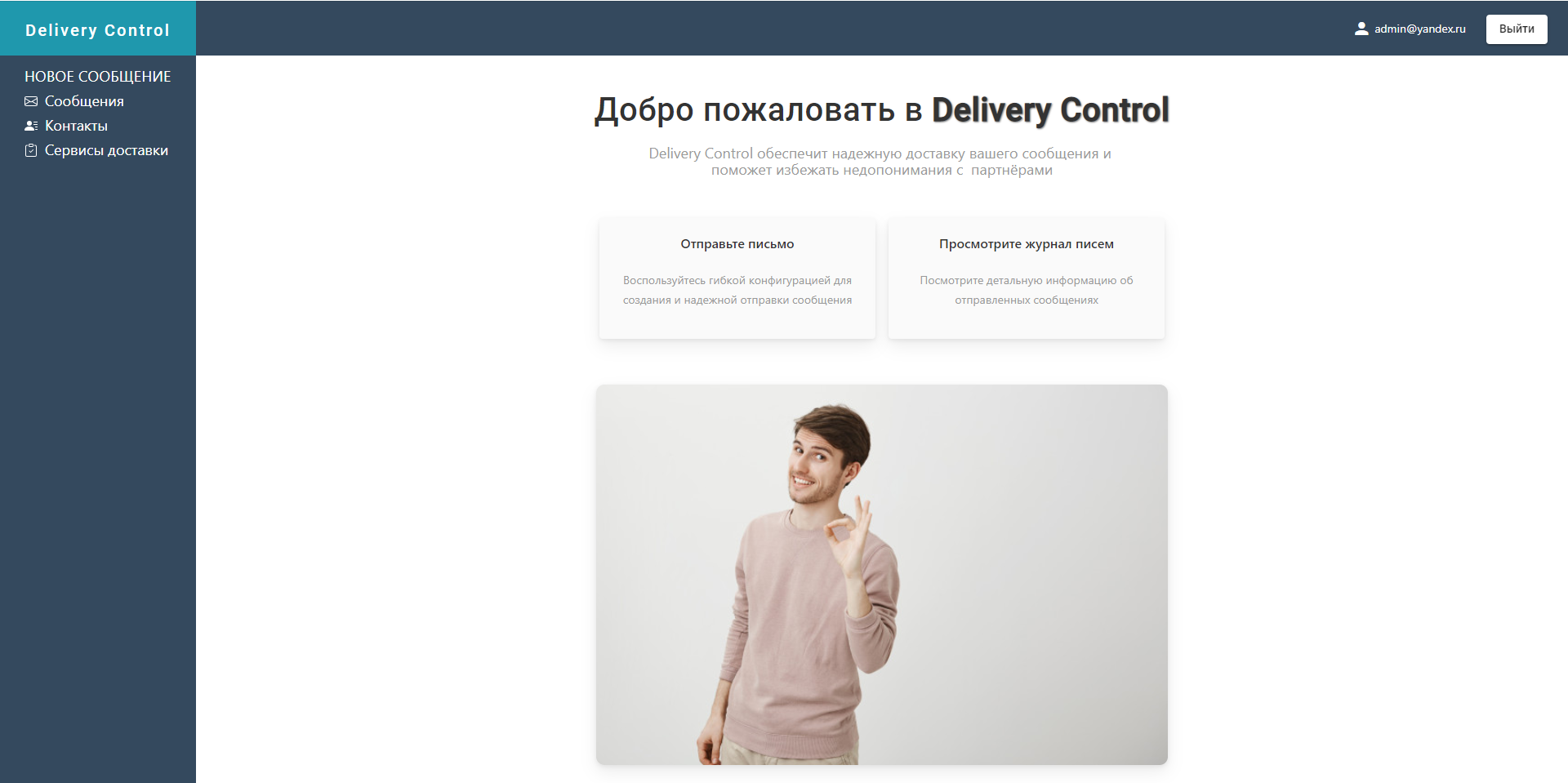
|  |
| --- |
| ***Рис. 2.2. Схема архитектуры ПС*** |



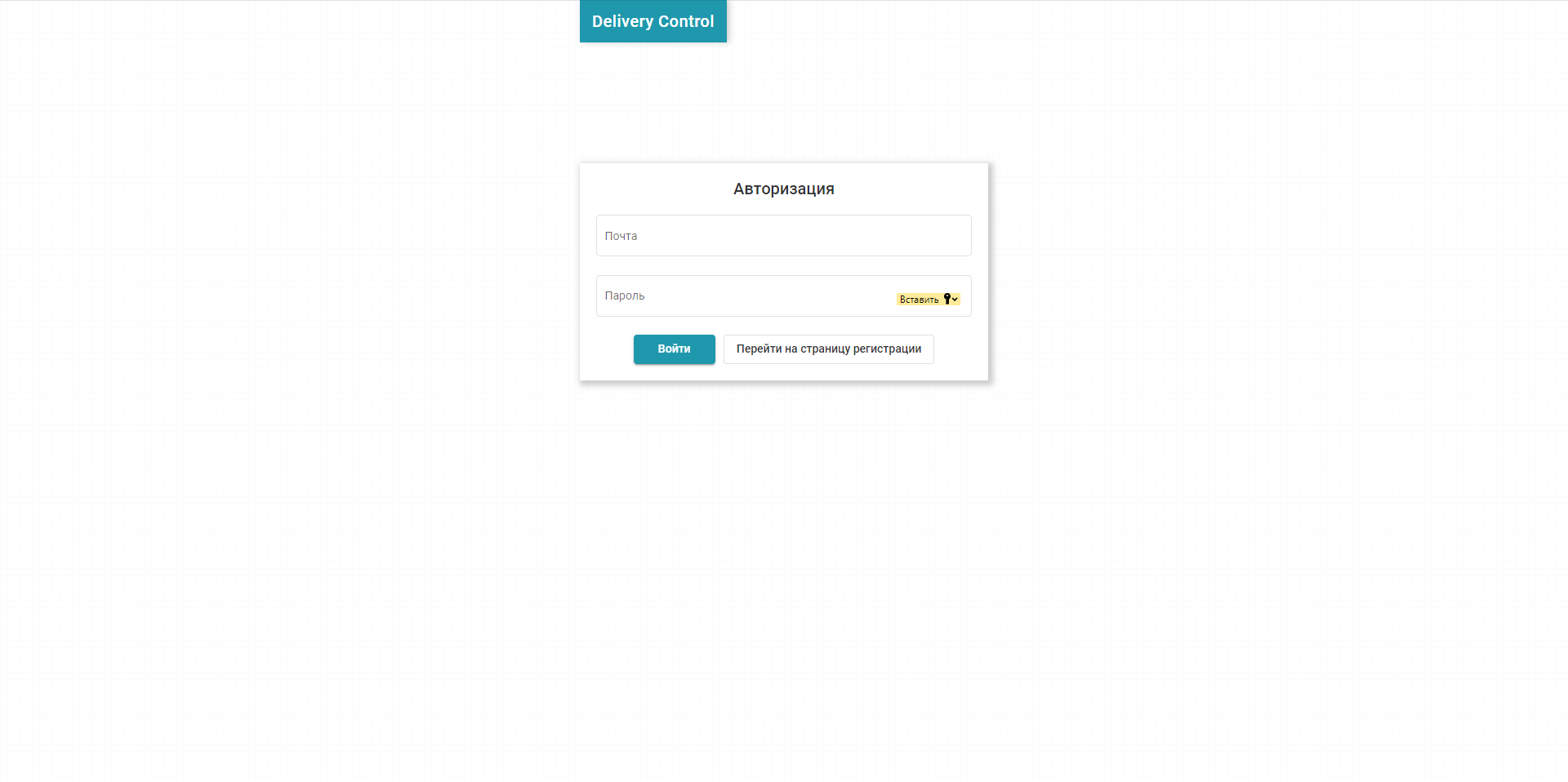
# РАЗРАБОТКА

## Клиентская часть

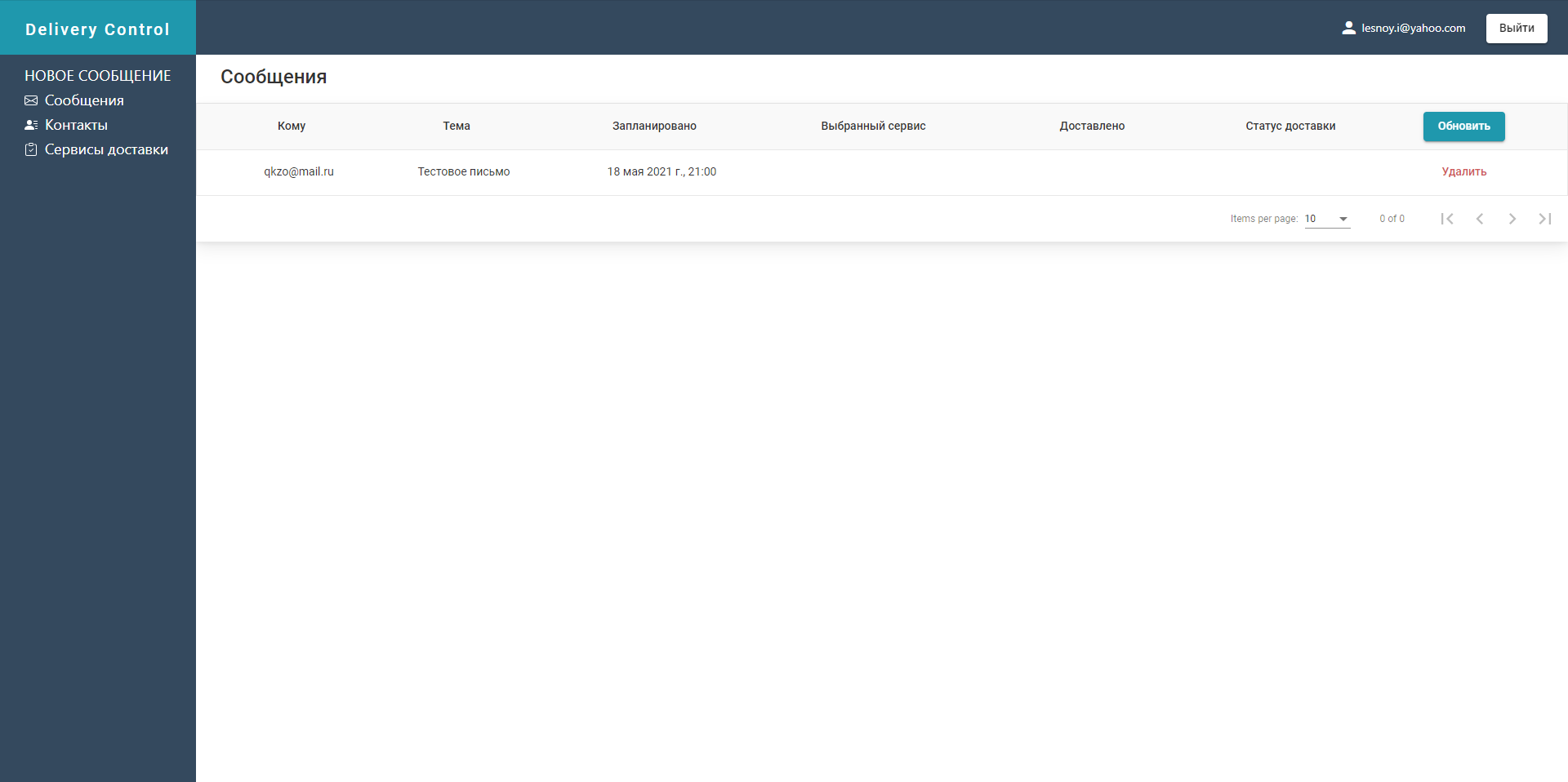
Спроектированы и разработаны основные интерфейсы программного комплекса.



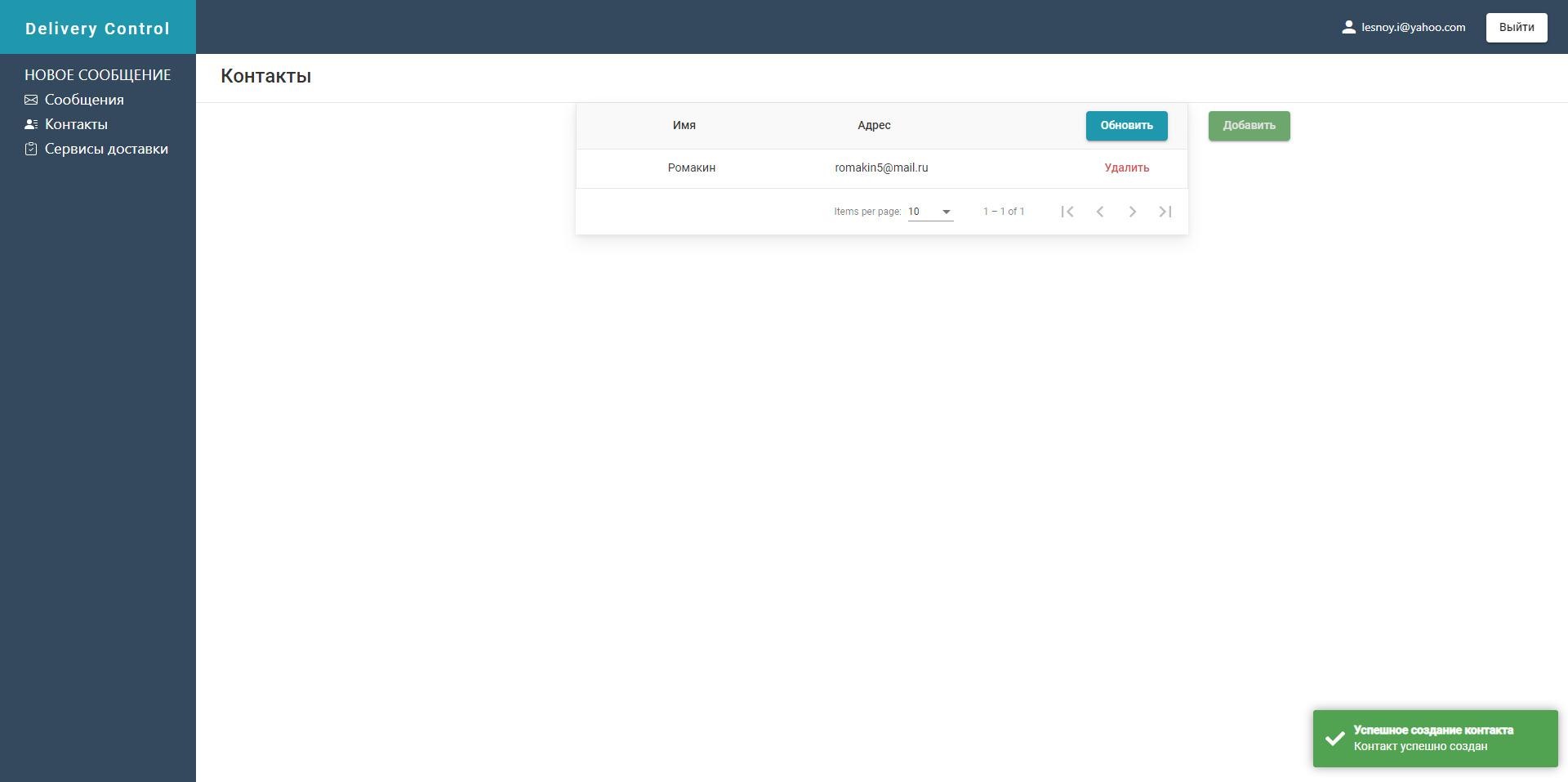
**Рис. 3.1. Главная страница**



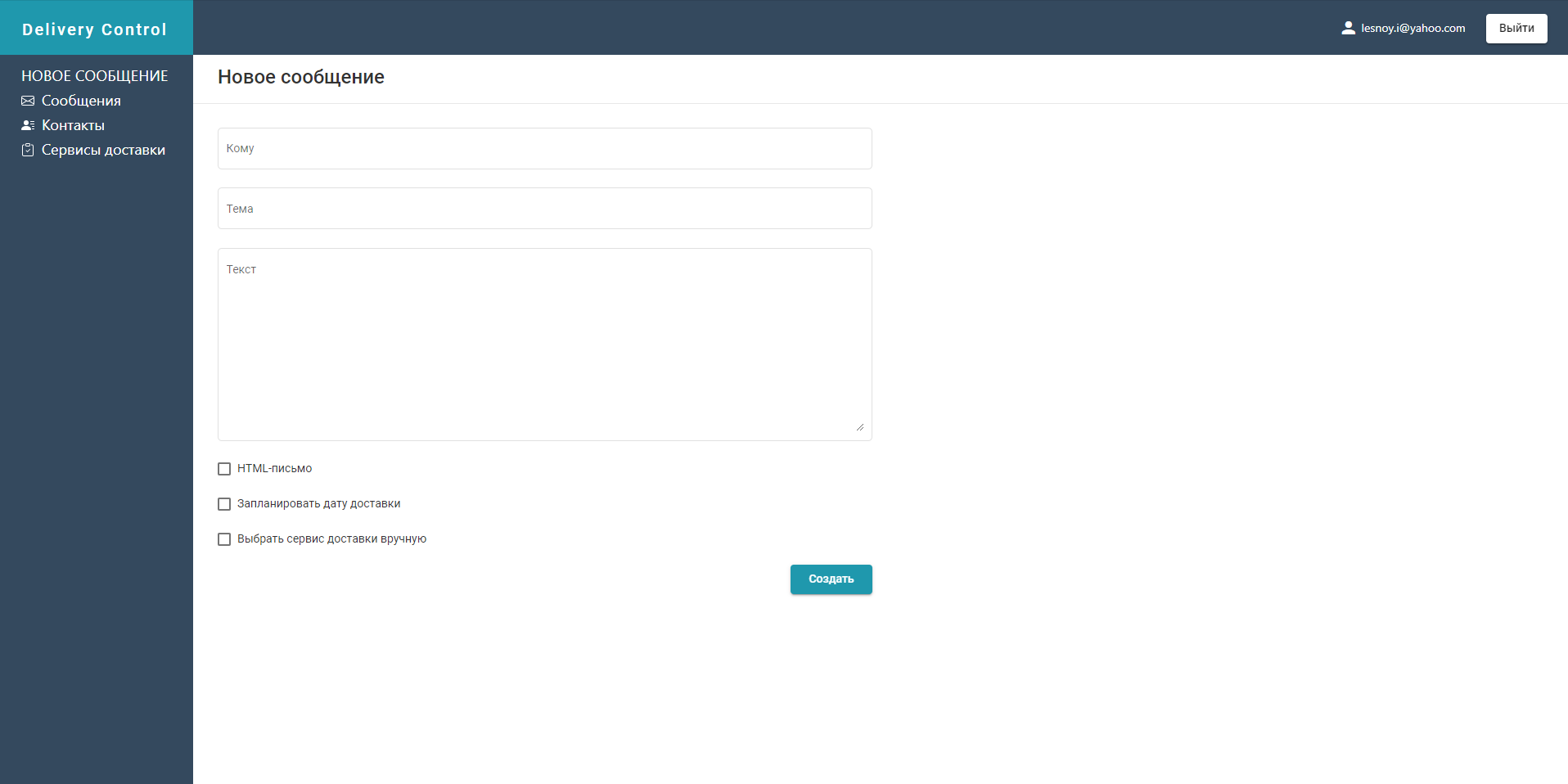
**Рис. 3.2. Страница регистрации и авторизации**



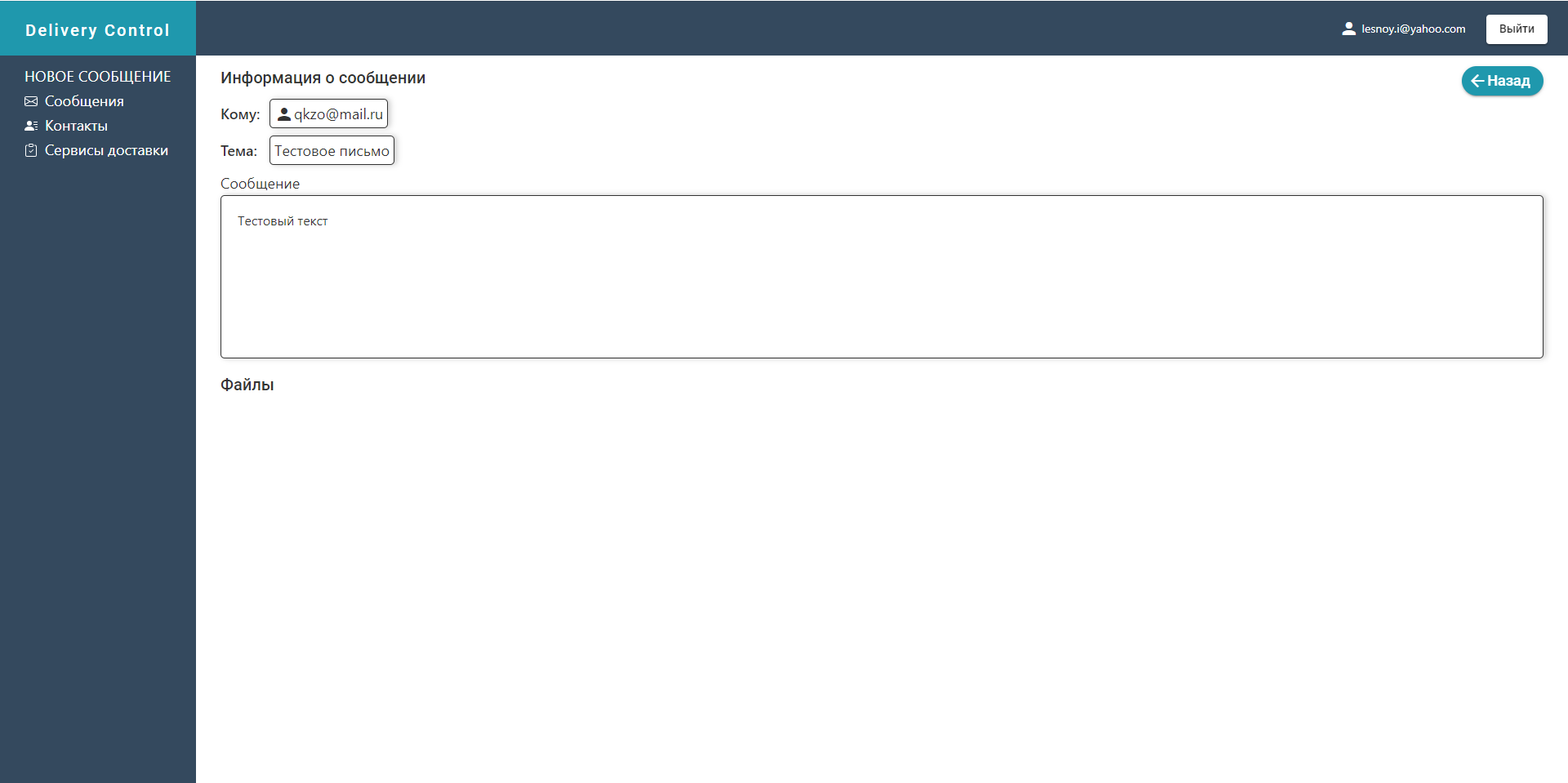
**Рис. 3.3. Список сообщений**



**Рис. 3.4. Страница контактов**



**Рис. 3.5. Создание сообщения**



**Рис. 3.6. Детали сообщения**

## Web API

Для взаимодействия клиентской и серверной части разработан Web API. Пример одного из контроллеров представлен в приложении в листинге 1.

А реализация контроллера авторизации представлен в листинге 2.

## Асинхронная отправка сообщений

Для асинхронной отправки сообщений была разработана фоновая служба. Код представлен в листинге 3.

# ТЕСТИРОВАНИЕ

В рамках практики для проверки корректности работы серверной части были разработаны модульные тесты.

В проекте Delivery.Domain.ServiceLayer.Tests содержатся тесты для методов класса Hash и других вспомогательных классов, таких как AutoMapper. Пример тестов представлен в листинге 4.

# ЗАКЛЮЧЕНИЕ

В рамках практики была осуществлена работа над дипломом на тему «Проектирование программного комплекса управления надёжностью отправки электронных писем с реализацией основного механизма работы и пользовательского интерфейса».

В результате проведенной работы были выполнены следующие задачи:

1. доработать функциональные требования;
2. доработать архитектуру программного комплекса;
3. доработать проектные модели программного комплекса;
4. реализовать базу данных в СУБД MS SQL Server;
5. реализовать клиентскую часть;
6. реализовать серверную часть;
7. выполнить тестирование разработанных подсистем (модулей).

# СПИСОК ЛИТЕРАТУРЫ

1. Что такое web-интерфейс: [Электронный ресурс]. URL: <https://semantica.in/blog/veb-interfejs.html/>.
2. Что такое API: [Электронный ресурс]. URL:  <https://skillbox.ru/media/code/chto_takoe_api/>.
3. Виды электронных подписей: [Электронный ресурс]. URL:  <http://www.consultant.ru/document/cons_doc_LAW_112701/d9cd621c949a3c9efef51c2884c247e18ab9908b/>.
4. Сертификат ключа подписи: [Электронный ресурс]. URL:  https://uc.iitrust.ru/articles/chto-takoe-sertifikat-klyucha-proverki-elektronnoy-podpisi/.
5. Официальный сайт Amazon SES: [Электронный ресурс]. URL:  https://aws.amazon.com/ru/ses/.
6. Официальный сайт Sendgrid: [Электронный ресурс]. URL:  https://sendgrid.com.
7. Проект Email queue пользователя tin-cat на Github: [Электронный ресурс]. URL:  <https://github.com/tin-cat/emailqueue>.
8. Введение в Web API: [Электронный ресурс]. URL: <https://metanit.com/sharp/mvc/12.1.php>.
9. Документация по ASP.NET: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/aspnet/core/?view=aspnetcore-3.1>.
10. Нормализация сущностей: [Электронный ресурс]. URL: <https://habr.com/ru/post/254773>.
11. Основы UML – диаграммы использования (use-case): [Электронный ресурс]. URL: <https://pro-prof.com/archives/2594>.
12. Тестирование методом черного ящика: [Электронный ресурс]. URL:  <https://habr.com/ru/post/462837/>.
13. Модульное тестирование или юнит-тестирование: [Электронный ресурс]. URL:  <https://habr.com/ru/post/169381/>.
14. Кросс-браузерное тестирование: [Электронный ресурс]. URL:  <https://imajor.ru/razrabotka/verstka/krossbrauzernost-sayta>.
15. Язык UML. Руководство пользователя: [Текст] / Буч Г., Рамбо Д., Якобсон И. – 2-е издание: Пер. с англ Мухин Н., М.: ДМК Пресс. – 496 с.: ил.
16. Документация по C#: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>
17. Официальная страница Angular: [Электронный ресурс]. URL: <https://angular.io/?developerstash>.
18. Большая книга CSS: [Текст] / Дэвид Макфарланд - СПб.: Питер, 2016.
19. Современный учебник JavaScript: [Электронный ресурс]. URL: <https://learn.javascript.ru/>
20. Язык программирования C# 7 и платформы .NET и .NET Core [Текст] / Троелсен, Джепикс – 2018.
21. Software Engineering for Internet Applications by Eve Andersson [Текст] / Philip Greenspun, Andrew Grumet – 2006.
22. Техническая документация по SQL Server: [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver15>
23. Техническая документация по MySQL: [Электронный ресурс]. URL: <https://dev.mysql.com/doc/>
24. Техническая документация по PostgreSQL: [Электронный ресурс]. URL: <https://postgrespro.ru/docs/postgresql>
25. React в действии [Текст] / Тиленс Томас Марк – ООО Издательство «Питер», 2019.
26. Vue.js в действии [Текст] / Эрик Хэнчетт, Бенджамин Листоун – ООО Издательство «Питер», 2019.
27. Чистый код. Создание, анализ и рефакторинг [Текст] / Роберт Мартин – ООО Издательство «Питер», 2013.

# ПРИЛОЖЕНИЕ

Листинг 1

Реализация UsersController и MessagesController

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using AutoMapper;  using DeliveryRely.Api.Models.Shared;  using DeliveryRely.Domain.Models.Users;  using DeliveryRely.Domain.Repository;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using Microsoft.EntityFrameworkCore;  namespace DeliveryRely.Api.Controllers  {  [Authorize]  [Route(Constants.ApiPrefix + "users")]  public class UsersController : ControllerBase  {  private readonly IRepository \_repository;  private readonly IMapper \_mapper;  public UsersController(IRepository repository, IMapper mapper)  {  \_repository = repository;  \_mapper = mapper;  }  // GET api/users/list  [HttpGet("list")]  public async Task<ApiResponse<List<User>>> GetList()  {  try  {  var users = await \_repository.Users  .AsNoTracking()  .ToListAsync();  return ApiResponse.Successful(users);  }  catch (Exception exception)  {  return ApiResponse.Failed<List<User>>(exception.Message);  }  }  // GET api/users/id  [Route("{id}")]  public async Task<ApiResponse<User>> Get(long id)  {  try  {  var foundUser = await \_repository.Users  .AsNoTracking()  .Where(user => user.UserId == id)  .FirstOrDefaultAsync();  if (foundUser == null)  {  throw new Exception("User not found");  }  return ApiResponse.Successful(foundUser);  }  catch (Exception exception)  {  return ApiResponse.Failed<User>(exception.Message);  }  }  // PUT api/users/id  [HttpPut]  [Route("{id}")]  public async Task<ApiResponse<User>> Put(long id, [FromBody]User user)  {  try  {  var updatedUser = await \_repository.UpdateUser(user, id);  return ApiResponse.Successful(updatedUser);  }  catch (Exception exception)  {  return ApiResponse.Failed<User>(exception.Message);  }  }  // DELETE api/users/id  [HttpDelete]  [Route("{id}")]  public async Task<bool> Delete(int id) =>  await \_repository.DeleteUser(id) != null;  }  }  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  using System;  using System.Collections.Generic;  using System.Linq;  using System.Threading.Tasks;  using AutoMapper;  using DeliveryRely.Api.Models.Shared;  using DeliveryRely.Domain.Models.Messages;  using DeliveryRely.Domain.Repository;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using Microsoft.EntityFrameworkCore;  namespace DeliveryRely.Api.Controllers.Messages  {  [Authorize]  [Route(Constants.ApiPrefix + "messages")]  public class MessagesController : ControllerBase  {  private readonly IRepository \_repository;  private readonly IMapper \_mapper;  public MessagesController(IRepository repository, IMapper mapper)  {  \_repository = repository;  \_mapper = mapper;  }  // GET api/messages/list  [Route("list")]  [HttpGet]  public async Task<ApiResponse<IEnumerable<MessageProjection>>> GetList()  {  try  {  var messageProjections = await \_repository.MessagesIncludeAll  .Select(message => \_mapper.Map<MessageProjection>(message))  .ToListAsync();  return ApiResponse.Successful<IEnumerable<MessageProjection>>(messageProjections);  }  catch (Exception exception)  {  return ApiResponse.Failed<IEnumerable<MessageProjection>>(exception.Message);  }  }  // GET api/messages/id  [Route("{id}")]  [HttpGet]  public async Task<ApiResponse<MessageProjection>> Get(long id)  {  try  {  var message = await \_repository.MessagesIncludeAll  .Where(m => m.MessageId == id)  .FirstOrDefaultAsync();    var messageProjection = \_mapper  .Map<MessageProjection>(message ?? throw new Exception("Message not found"));  return ApiResponse.Successful(messageProjection);  }  catch (Exception exception)  {  return ApiResponse.Failed<MessageProjection>(exception.Message);  }  }  // GET api/messages/queue/id  [Route("queue/{id}")]  [HttpGet]  public async Task<ApiResponse<IEnumerable<MessageProjection>>> GetMessageForQueue(int id)  {  try  {  var messages = await \_repository.GetMessagesForQueue(id);  var messageProjections = messages  .Select(m => \_mapper.Map<MessageProjection>(m));  return ApiResponse.Successful(messageProjections);  }  catch (Exception exception)  {  return ApiResponse.Failed<IEnumerable<MessageProjection>>(exception.Message);  }  }  // GET api/messages/user/id  [Route("user/{id}")]  [HttpGet]  public async Task<ApiResponse<IEnumerable<MessageProjection>>> GetListForUser(long id)  {  try  {  var messageProjections = (await \_repository  .GetMessagesForUser(id))  .Select(m => \_mapper.Map<MessageProjection>(m));  return ApiResponse.Successful(messageProjections);  }  catch (Exception exception)  {  return ApiResponse.Failed<IEnumerable<MessageProjection>>(exception.Message);  }  }  // GET api/messages/next  [Route("next")]  [HttpGet]  public async Task<ApiResponse<Message>> Get()  {  try  {  var message = await \_repository.GetCurrentMessageFromBusiestQueue();  if (message == null)  {  throw new Exception("All queues are empty");  }  return ApiResponse.Successful(message);  }  catch (Exception exception)  {  return ApiResponse.Failed<Message>(exception.Message);  }  }  // POST api/messages  [HttpPost]  public async Task<ApiResponse<MessageProjection>> Post([FromBody] Message message)  {  try  {  var messageProjection = \_mapper.Map<MessageProjection>(await \_repository.InsertMessage(message));  return ApiResponse.Successful(messageProjection);  }  catch (Exception exception)  {  return ApiResponse.Failed<MessageProjection>(exception.Message);  }  }  // PUT api/messages/id  [Route("{id}")]  [HttpPut]  public async Task<ApiResponse> Put(long id, [FromBody] Message message)  {  try  {  await \_repository.UpdateMessage(message, id);  return ApiResponse.Successful();  }  catch (Exception exception)  {  return ApiResponse.Failed(exception.Message);  }  }  // DELETE api/messages/id  [Route("{id}")]  [HttpDelete]  public async Task<ApiResponse> Delete(long id)  {  try  {  await \_repository.DeleteMessage(id);  return ApiResponse.Successful();  }  catch (Exception exception)  {  return ApiResponse.Failed(exception.Message);  }  }  }  } |

Листинг 2

Реализация AuthController

|  |
| --- |
| using DeliveryRely.Api.Models.Shared;  using DeliveryRely.Api.Models.Users;  using DeliveryRely.ServiceLayer;  using Domain.Models.Users;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using Microsoft.EntityFrameworkCore;  using Microsoft.IdentityModel.Tokens;  using System;  using System.Collections.Generic;  using System.IdentityModel.Tokens.Jwt;  using System.Linq;  using System.Security.Claims;  using System.Threading.Tasks;  using AutoMapper;  using DeliveryRely.Domain.Models.Users;  using DeliveryRely.Domain.Repository;  namespace DeliveryRely.Api.Controllers.Auth  {  [Route(Constants.ApiPrefix + "auth")]  public class AuthController : Controller  {  private readonly IRepository \_repository;  private readonly IMapper \_mapper;  public AuthController(IRepository repository, IMapper mapper)  {  \_repository = repository;  \_mapper = mapper;  }  [HttpPost("sign-in")]  public async Task<ApiResponse<AuthToken>> SignIn([FromBody] UserSpecification specification)  {  try  {  var foundUser = await TryFindUserAsync(specification.Email, specification.Password);  // memorize info about an authorized user  AuthOptions.CurrentUser = foundUser ?? throw new Exception("Invalid username or password");  // generate JWT  var authToken = GenerateAuthToken(foundUser);  return ApiResponse.Successful(authToken);  }  catch (Exception exception)  {  return ApiResponse.Failed<AuthToken>(exception.Message);  }  }  private AuthToken GenerateAuthToken(UserBase foundUser)  {  var identity = GetIdentity(foundUser);  var now = DateTime.UtcNow;  var jwt = new JwtSecurityToken(  notBefore: now,  claims: identity.Claims,  expires: now.Add(TimeSpan.FromMinutes(AuthOptions.LifetimeInMinutes)),  signingCredentials: new SigningCredentials(AuthOptions.GetSymmetricSecurityKey(), SecurityAlgorithms.HmacSha256));  var authToken = new AuthToken  {  Token = AuthOptions.TokenPrefix + new JwtSecurityTokenHandler().WriteToken(jwt),  User = \_mapper.Map<UserProjection>(foundUser)  };  return authToken;  }  private async Task<User> TryFindUserAsync(string email, string password = null)  {  var passwordHash = password != null  ? new HashSHA512().MakeHash(password)  : null;  // If password is empty, search only using Email  return await \_repository.Users  .Where(u => u.Email == email && (passwordHash == null || u.PasswordHash == passwordHash))  .FirstOrDefaultAsync();  }  private static ClaimsIdentity GetIdentity(UserBase user)  {  var claims = new List<Claim>  {  new(ClaimsIdentity.DefaultNameClaimType, user.Email),  new(ClaimsIdentity.DefaultRoleClaimType, user.Role.ToString())  };  return new ClaimsIdentity(claims, "Token",  ClaimsIdentity.DefaultNameClaimType, ClaimsIdentity.DefaultRoleClaimType);  }  [HttpPost("sign-up")]  public async Task<ApiResponse<AuthToken>> SignUp([FromBody] UserSpecification specification)  {  try  {  var foundUser = await TryFindUserAsync(specification.Email);  if (foundUser != null)  {  throw new Exception("User with current data already exists");  }  var insertedUser = await \_repository.InsertUser(new User  {  Email = specification.Email,  PasswordHash = new HashSHA512().MakeHash(specification.Password),  UserName = specification.UserName,  UserSecondName = specification.UserSecondName,  IsActive = specification.IsActive  });  // memorize info about an authorized user  AuthOptions.CurrentUser = insertedUser;  var authToken = GenerateAuthToken(insertedUser);  return ApiResponse.Successful(authToken);  }  catch (Exception exception)  {  return ApiResponse.Failed<AuthToken>(exception.Message);  }  }  [HttpGet("sign-out")]  // public "new" - because method "SignOut" already exists in  // parent class but it returns type different to ApiResponse  public new ApiResponse SignOut()  {  try  {  // Remove info about current user  AuthOptions.CurrentUser = null;  return ApiResponse.Successful();  }  catch (Exception exception)  {  return ApiResponse.Failed(exception.Message);  }  }    [Authorize]  [HttpGet("check")]  public ApiResponse Check()  {  return ApiResponse.Successful();  }  }  } |

Листинг 3

Служба WorkerMain

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Threading;  using System.Threading.Tasks;  using DeliveryRely.Domain.Models.Messages;  using DeliveryRely.Domain.Repository;  using DeliveryRely.SendingService.Senders;  using DeliveryRely.SendingService.Senders.SendPulse;  using DeliveryRely.SendingService.Senders.Smtp;  using DeliveryRely.SendingService.SenderServices;  using Microsoft.Extensions.Hosting;  using Microsoft.Extensions.Logging;  namespace DeliveryRely.SendingService  {  public class WorkerMain : BackgroundService  {  private readonly IRepository \_repository;  private readonly ILogger<WorkerMain> \_logger;  private readonly ISenderService \_senderService;  private readonly IEnumerable<ISender> \_deliveryTypesList = new List<ISender>()  {  new SmtpSender(),  new SendPulseService(),  new SendgridSender()  };  private int DelayInMs { get; } = 10 \* 1000;    public WorkerMain(IRepository repository, ILogger<WorkerMain> logger)  {  \_repository = repository;  \_logger = logger;  \_senderService = new SenderService(\_deliveryTypesList, \_logger);  }  protected override async Task ExecuteAsync(CancellationToken stoppingToken)  {  while (!stoppingToken.IsCancellationRequested)  {  Thread.Sleep(DelayInMs);  try  {  \_logger.LogInformation("Try send another message");  var message = await \_repository  .GetCurrentMessageFromBusiestQueue()  .ConfigureAwait(false);  var result = await \_senderService  .TrySendByEachService(message)  .ConfigureAwait(false);    if (result.IsSuccess)  {  message.MarkSuccessful(result.DeliveryType);  message.DeleteFromQueue();  await UpdateMessage(message)  .ConfigureAwait(false);  \_logger.LogInformation("Message successfully sent by {Service}", result.DeliveryType);  continue;  }  message.MarkFailed(result.ErrorMessage);  message.DeleteFromQueue();  await UpdateMessage(message)  .ConfigureAwait(false);  \_logger.LogInformation("Sending via each delivery service failed");  }  catch (Exception exception)  {  \_logger.LogError("{Message}", exception.Message);  }  }  \_logger.LogInformation("Worker stopped");  }  private async Task UpdateMessage(Message message) =>  await \_repository  .UpdateMessage(message, message.MessageId)  .ConfigureAwait(false);  }  } |

Листинг 4

Пример модульного теста

|  |
| --- |
| using DeliveryRely.Domain.Models.Contacts;  using Domain.Models.Users;  using Microsoft.VisualStudio.TestTools.UnitTesting;  namespace DeliveryRely.Domain.Tests  {  [TestClass]  public class ContactTest  {  [TestMethod]  public void FieldsTest()  {  var contact = new Contact  {  ContactId = 1,  ContactName = "contact",  ContactEmail = "contact@yandex.ru",  UserId = 4,  };  Assert.AreEqual("contact", contact.ContactName);  Assert.AreEqual("contact@yandex.ru", contact.ContactEmail);  Assert.AreEqual(4, contact.UserId);  }  }  } |