

МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение высшего  
образования

«Национальный исследовательский университет

«Московский институт электронной техники»

Институт системной и программной инженерии и информационных технологий

Мясников Максим Александрович

Выпускная квалификационная работа

по направлению 09.03.04 «Программная инженерия»

Разработка веб-службы для доступа к электронной почте на основе двухфакторной аутен-  
тификации (ВС ДЭП)

Студент

\_\_\_\_\_Мясников М.А.

Научный руководитель

\_\_\_\_\_д.т.н., проф. Гагарина Л.Г.

Москва, 2022

## СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ .....	4
ВВЕДЕНИЕ.....	5
1 Исследовательский раздел .....	7
1.1 Актуальность предметной области.....	7
1.2 Изучение предметной области .....	8
1.2.1 Протоколы работы с электронной почты .....	8
1.2.2 Подходы к построению API веб-служб.....	9
1.2.3 Авторизация и аутентификация.....	12
1.2.4 Почтовые сервера .....	14
1.2.5 Алгоритм создания одноразовых паролей.....	16
1.3 Обзор и сравнение существующих аналогов.....	17
1.3.1 Microsoft Graph .....	17
1.3.2 Gmail API.....	18
1.3.3 Mail.ru для бизнеса .....	19
1.3.4 Сравнение аналогов .....	19
1.4 Цель и задачи разработки .....	20
1.5 Концептуальная модель предметной области .....	21
1.6 Структура входных и выходных данных .....	24
1.7 Требования к алгоритмам работы программы.....	25
Выводы по исследовательскому разделу .....	26
2 Конструкторский раздел.....	27
2.1 Выбор языка и среды программирования .....	27
2.1.1 Выбор языка программирования .....	27
2.1.2 Выбор среды разработки .....	31
2.2 Алгоритм и методы работы ВС ДЭП.....	33
2.2.1 Алгоритм работы ВС ДЭП .....	33

2.2.2	Логирование .....	34
2.2.3	Документирование API .....	36
2.2.4	Система управления базами данных и база данных .....	38
2.2.5	Внедрение зависимостей .....	40
2.2.6	Двухфакторная аутентификация .....	41
2.3	Пользовательский интерфейс ВС ДЭП .....	42
2.3.1	Технологии разработки пользовательского интерфейса .....	42
2.3.2	Переопределяемые компоненты .....	43
2.3.3	Форма авторизации .....	45
2.3.4	Компоненты просмотра писем .....	45
2.3.5	Компоненты работы с почтовыми серверами .....	46
	Выводы по конструкторскому разделу .....	47
3	Испытательный раздел .....	49
3.1	Средства отладки программы .....	49
3.1.1	Отладка с использованием статического анализатора кода .....	49
3.1.2	Отладка с использованием логгера и встроенного отладчика .....	51
3.2	Средства тестирования программы .....	53
3.2.1	Модульное тестирование .....	54
3.2.2	Реализация модульного тестирования .....	55
3.2.3	Результаты модульного тестирования .....	59
3.2.4	Интеграционное тестирование .....	61
	Выводы по испытательному разделу .....	64
	ЗАКЛЮЧЕНИЕ .....	65
	СПИСОК ЛИТЕРАТУРЫ .....	66
	ПРИЛОЖЕНИЕ 1	
	ПРИЛОЖЕНИЕ 2	
	ПРИЛОЖЕНИЕ 3	

## ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

ВС ДЭП – веб-служба для доступа к электронной почте на основе двухфакторной аутентификации

ООП – объектно-ориентированное программирование

ФП – функциональное программирование

API – Application Programming Interface (программный интерфейс приложения)

НМАС – hash-based message authentication code (код аутентификации сообщений, использующий хеш-функции)

НОТН – НМАС-Based One-Time Password Algorithm (основанный на НМАС алгоритм генерации одноразовых паролей)

ИМАР – Internet Message Access Protocol (протокол доступа к интернет-сообщениям)

JSON – JavaScript Object Notation (объектная нотация JavaScript)

ОТН – one-time password (одноразовый пароль)

РОР3 – Post Office Protocol Version 3 (протокол почтового отделения 3-й версии)

РЕСТ – Representational State Transfer (передача репрезентативного состояния)

RFC – Request for Comments (запрос на отзывы)

SOAP – Simple Object Access Protocol (простой протокол доступа к объектам)

SSL – Secure Sockets Layer (уровень защищённых сокетов)

ТОТН – Time-based One-Time Password Algorithm (базирующийся на времени алгоритм генерации одноразовых паролей)

W3C – World Wide Web Consortium (Консорциум Всемирной паутины)

XML – eXtensible Markup Language (расширяемый язык разметки)

## ВВЕДЕНИЕ

В современном мире сложно представить такого человека, который не имел бы своего электронного почтового ящика. Более того, люди имеют в своем распоряжении по несколько электронных почтовых адресов, которые могут относиться к одному или к разным почтовым серверам. Во втором случае доступ к письмам, хранящимся на сервере, обычно осуществляется с помощью сервисов, предоставляемых теми же компаниями, что и являются владельцами сервера, к которому относится адрес электронной почты. Так, для сервера yandex.ru используется сервис «Яндекс.Почта», для mail.ru, inbox.ru, bk.ru, internet.ru – «Почта Mail.ru», для gmail.com – «Gmail», для outlook.com – «Outlook.com» и т. п.

При наличии нескольких электронных почтовых адресов, их обход и проверка всех почтовых ящиков на наличие новых писем, превращается в сложную задачу. Также обладатель нескольких почтовых адресов, сталкивается с проблемой, поиска письма, принадлежность которого к какому-либо из почтовых ящиков он забыл. В таком случае ему придется проверять всех их в поисках потерянного письма, что является долгим процессом.

Решением этих проблем является введение веб-службы, позволяющей иметь доступ у различным почтовым ящикам.

Таким образом, целью выполнения ставится повышение скорости разработки веб-приложений, включающих работу с различными почтовыми серверами и почтовыми ящиками. В рамках данной работы для достижения поставленной цели производятся:

- исследование предметной области и актуализация изучаемой проблемы;
- обзор существующих аналогов и их сравнительный анализ;
- описание концептуальной модели предметной области
- разработка описания входных и выходных данных, форматов их внутреннего представления в программе;
- сравнительный анализ, выбор языка программирования и среды разработки;
- разработка и описание алгоритма и методов решения поставленных задач;
- программная реализация ВС ДЭП, ее пользовательского интерфейса;
- отладка и тестирование ВС ДЭП;
- разработка руководства программиста ВС ДЭП.

Пояснительная записка состоит из перечня используемых сокращений, введения, исследовательского раздела, конструкторского раздела, испытательного раздела, заключения,

списка использованной литературы и трех приложений: текста программы, технического задания и руководства программиста.

Исследовательский раздел включает исследование предметной области, выделение проблематики. Рассматривается актуальность выбранной темы, проводится изучение аналогичных решений. Формулируются цель и задачи исследования. Описывается концептуальная модель предметной области, а на ее основании описываются наборы входных и выходных данных, определяются требования к алгоритмам работы программы.

В конструкторском разделе проводится анализ существующих языков программирования, выбор подходящего. На основании выбранного языка проводится анализ сред разработки и выбор наиболее подходящей. Описывается процесс разработки алгоритма, реализация ВС ДЭП и ее пользовательского интерфейса.

В испытательном разделе рассматриваются способы отладки и технологии тестирования программ. Описывается процесс составления кейс-тестов. Описываются процессы отладки и тестирования, их результаты.

Список использованной литературы содержит сведения об источниках, использованных при составлении отчета.

Приложение 1 содержит код программы.

Приложение 2 содержит техническое задание на разработку ВС ДЭП.

Приложение 3 содержит руководство программиста.

## 1 Исследовательский раздел

### 1.1 Актуальность предметной области

Электронная почта является востребованной технологией, позволяющей вести переписку (личную, деловую), передавать как текстовые сообщения, так и файлы различных форматов; служащей для регистрации на различных сервисах.

Наличие у человека нескольких адресов электронной почты является распространённым явлением. Для облегчения доступа к различным электронным почтовым ящикам существуют различные решения: почтовые сервисы («Яндекс.Почта», «Почта Mail.ru», «Gmail» и т. д.) и настольные почтовые клиенты (такие как «Microsoft Outlook», «Mozilla Thunderbird», «The Bat!», «eM Client», «Mailbird» и т. д.).

Однако эти решения являются завершенными продуктами и зачастую не являются расширяемыми. Интегрировать функциональность из этих решений в свое также не представляется возможным. Если разработчику понадобится реализовать в своем продукте работу с почтовыми серверами и почтовыми ящиками, ему придется разрабатывать этот функционал самостоятельно. Так, различные разработчики тратят время на одни и те же действия в своих продуктах.

Конечно, некоторые компании предоставляют API для некоторых методов и функций для взаимодействия с их почтовым сервером, но не более. Это полезно, если нужно взаимодействовать только с одним почтовым сервером. Если же планируется работа с различными почтовыми серверами, то решение строится на подключении веб-служб для тех серверов, компании которых предоставляют такие службы; иначе необходимо реализовывать связь с сервером напрямую. Всё это сказывается на скорости разработки и перегруженности приложения. Эти проблемы решаются разработкой веб-службы, которая будет включать в себя методы взаимодействия с различными почтовыми серверами.

На рисунке 1.1 изображен принцип взаимодействия приложений с почтовыми серверами до и после введения ВС ДЭП.

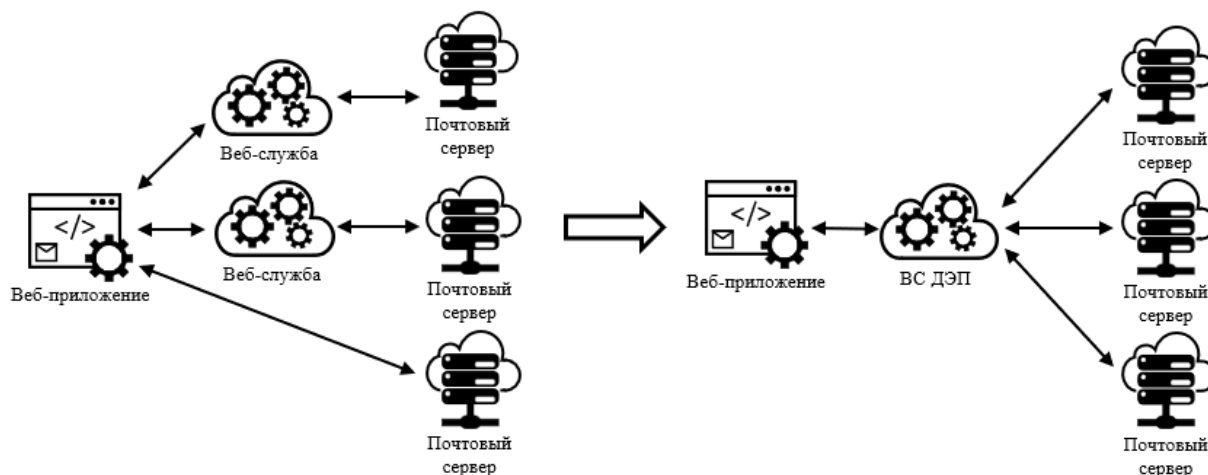


Рисунок 1.1 – Иллюстрация принципа взаимодействия приложений с почтовыми серверами до и после введения ВС ДЭП

Поэтому, принято решение разработать веб-службу с открытым интерфейсом для доступа к различным почтовым ящикам. В дальнейшем веб-службу можно будет использовать как самостоятельно, так и в силу открытого интерфейса встраивать в другие приложения.

## 1.2 Изучение предметной области

### 1.2.1 Протоколы работы с электронной почтой

На данный момент основными протоколами для получения электронных писем с почтового сервера являются POP3 и IMAP.

POP3 (Post Office Protocol Version 3 – 3-я версия протокола почтового отделения) – интернет-протокол прикладного уровня, используемый для получения электронной почты с удаленного почтового сервера. Принцип работы с протоколом заключается в проверке удаленного сервера на наличие новых писем, последующего их скачивания на устройство пользователя и удаление скачанных писем с сервера. При этом письмо скачивается целиком, со всеми вложениями. К преимуществам такого подхода получения писем можно отнести:

- экономия трафика, так как письма загружаются в локальное хранилище один раз, и хранятся на нем;
- быстрый доступ к вложениям, т.к. они загружаются вместе с письмами;
- сохранение незаполненным дискового пространства почтового сервера, выделенного под почтовый ящик.

Однако, также можно выделить и ряд недостатков данного подхода:



- риск потери данных при поломке локального носителя, на котором хранятся загруженные письма;
- невозможность синхронизации между несколькими компьютерами.

Таким образом, данный подход подходит пользователям, которым достаточно иметь доступ к электронной почте с одного устройства, при условии, что они будут периодически выполнять резервирование данных, с целью снижения риска потери загруженных писем.

IMAP (Internet Message Access Protocol – протокол доступа к интернет-сообщениям) – интернет-протокол прикладного уровня, также как и POP3, служащий для получения доступа к электронным письмам на удаленном почтовом сервере. Разработанный как альтернатива протоколу POP3, имеет ряд доработок, связанных с возможностью работы с почтой непосредственно на сервере, без загрузки их на локальное устройство. Так, к преимуществам протокола IMAP перед POP3 можно отнести:

- хранение писем на почтовом сервере, а как следствие меньший риск их потерять;
- возможность одновременного доступа к письмам с нескольких устройств;
- установка флагов письмам (прочитано, отвечено, черновик и т. п.).

Переход к постоянному хранению писем на почтовом сервере, помимо преимуществ имеет и несколько недостатков:

- необходимость постоянного интернет-соединения для доступа к почтовому ящику;
- хоть риск потери электронных писем и ниже, чем при работе с POP3 и хранением писем на локальном устройстве, вероятность поломки серверного оборудования также присутствует.

Исходя из выше написанного следует что протокол IMAP является более универсальным по сравнению с POP3. Если у пользователя нет проблемы с доступом в интернет и получение писем не ограничено одним единственным устройством, то следует использовать именно IMAP.

### 1.2.2 Подходы к построению API веб-служб

Основными подходами к построению API веб-служб являются RESTful API и SOAP API. Если первый подход основывается на архитектурном стиле REST (Representational State Transfer – передача репрезентативного состояния), то второй – на протоколе обмена сообщениями SOAP (Simple Object Access Protocol – простой протокол доступа к объекту). Оба подхода имеют существенные различия.

Рассмотрим, что из себя представляет протокол SOAP. Сообщение по протоколу SOAP передаются в формате XML, и имеют структуру:

- envelope – корневой элемент, являющийся обязательным;
- header – заголовок; необязательный элемент, содержащий атрибуты сообщения;
- body – тело; обязательный элемент содержащий непосредственно сообщение;
- fault – ошибки; необязательный элемент, в котором описываются ошибки.

В листинге 1.1 изображен пример сообщения SOAP-запроса, а в листинге 1.2 изображен пример сообщения SOAP-ответа. Примеры взяты из документа W3C Simple Object Access Protocol (SOAP) 1.1

#### Листинг 1.1 – Сообщение SOAP-запрос

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

#### Листинг 1.2 – Сообщение SOAP-ответ

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
```

```
<m:GetLastTradePriceResponse xmlns:m="Some-URI">
  <Price>34.5</Price>
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Строгость спецификации SOAP-сообщений, с одной стороны, является их преимуществом, так как представляет собой строгий стандарт, но тем самым усложняет реализацию.

К преимуществам SOAP также относится неразрывно идущий с этим протоколом WSDL (Web Service Description Language) – язык описания веб-сервисов. WSDL как и SOAP имеет XML-синтаксис и полностью описывает функции и процедуры веб-службы и способы доступа к ним.

В отличие от SOAP API подхода, использующего в своей основе протокол обмена сообщений, RESTful API подход основывается на архитектурном стиле REST. Иными словами, здесь нет строго стандарта, но есть набор правил, которым следует придерживаться:

- модель клиент-сервер;
- отсутствие хранения состояния клиента между запросами на сервере;
- кэширование ответов сервера;
- единообразие интерфейса;
- иерархическая структура сетей (слои);
- код по требованию (расширение функциональности клиента за счет получаемых с сервера сценариев).

Вызов функции или процедуры веб-службы при REST подходе представляет собой HTTP-запрос, определяющий к чему обратиться по конечной точке и HTTP методу. Особое внимание стоит уделить использованию методов HTTP – в RESTful API принято использовать:

- GET – для получения ресурса;
- POST – для создания ресурса;
- DELETE – для удаления ресурса;
- PUT – для обновления ресурса.

В таблице 1.1 приведен пример набора функций созданных для RESTful API для класса Customer (клиент).

Таблица 1.1 – Пример действий веб-службы для различных HTTP методов и конечных точек.

Ресурс	POST	GET	PUT	DELETE
/books	Создание новой книги	Получение всех книг	Обновление всех книг	Удаление всех книг
/books/1	-	Получение сведений о книге 1	Обновление сведений о книге 1	Удаление книги 1
/books/1/authors	Создание нового автора для книги 1	Получение всех авторов книги 1	Обновление авторов книги 1	Удаление всех авторов книги 1

При этом, в отличие от SOAP, REST не ограничивает сообщения определённым форматом представления данных. Хотя наибольшее распространение и имеет формат JSON (который является более гибким и удобочитаемым чем XML), в сообщения могут передаваться и в XML формате, а также в виде обычного текста или бинарных данных и т. п.

Если у SOAP есть WSDL как способ описания, в REST применяется спецификация открытого API (OAS – open API specification), представленная фреймворком Swagger.

### 1.2.3 Авторизация и аутентификация

Для обеспечения безопасности данных пользователей, учетные записи ограничиваются определенным набором возможных действий. Выдача прав на какое-либо действие подразумевает под собой процесс авторизации. Таким образом чтобы пользователь получил доступ к своим данным, он должен пройти процесс авторизации. Но чтобы пройти процесс авторизации, нужно сначала пройти два других процесса: идентификацию и аутентификацию. Во время идентификации пользователь однозначно определяется в системе по имени учетной записи. Аутентификация – это проверка подлинности субъекта, пытающегося авторизоваться. Чаще всего эта проверка происходит путем предоставления пароля учетной записи.

Однако проверки только по паролю часто недостаточно, т.к. злоумышленник различными способами может получить доступ к паролю от учетной записи какого-либо пользо-

вателя. Для повышения уровня защиты доступа к данным пользователей применяется многофакторная аутентификация, в том числе широко распространена двухфакторная аутентификация.

Многофакторная аутентификация представляет собой комбинацию нескольких факторов аутентификации, а именно:

- фактор знания (authentication by knowledge);
- фактор владения (authentication by ownership);
- фактор биометрических характеристик (authentication by characteristic).

Фактор знания (something they know – что-то известное) – пароль либо PIN-код (personal identification number – персональный идентификационный номер). Основным его преимуществом является возможность изменения и использования без специальных устройств.

Фактор владения (something they have – что-то чем обладают) – ключ-флешка или магнитная карта, OTP-токен (устройство генерирующее одноразовый пароль). Главным преимуществом этого фактора по сравнению с первым является сложность создания дубликата, а как следствие использования злоумышленниками. С другой стороны, физические носители могут быть просто украдены.

Фактор биометрических характеристик (something they are – что-то присущее) – отпечаток пальца, рисунок сетчатки глаз, голос. Данный фактор является наиболее сильным, но в тоже время и наиболее дорогим, т.к. производство периферийных устройств, эффективно распознающих биометрические характеристики, достаточно дорогое.

Таким образом, каждый из факторов имеет как сильные, так и слабые стороны, поэтому они применяются в связке, компенсируя друг друга. Часто можно встретить связку первого и второго фактора, с определенной модификацией: OTP-token может быть заменен, на приложение на телефоне, генерирующее одноразовые пароли.

Для одновременной генерации одноразовых паролей в системе, и на устройстве-генераторе пользователя, используются различные алгоритмы, такие как: TOTP (time-based one-time password – базируемый на времени одноразовый пароль) и HOTP (HMAC-based one-time password – базируемый на хеше одноразовый пароль; HMAC – hash-based message authentication code, базируемый на хеше код аутентификации сообщений).

#### 1.2.4 Почтовые сервера

Рассмотрим особенности взаимодействия с почтовыми серверами таких компаний как «VK» (до 12 октября 2021 г. «Mail.ru Group»), «Яндекс», «Google» и «Microsoft».

##### Почта Mail.ru («VK»)

Для доступа к почтовому ящику следует обращаться к серверам «imap.mail.ru» (IMAP-сервер) и «pop.mail.ru» (POP3-сервер). В силу использования протокола шифрования SSL/TLS, используемые порты: IMAP – 993, POP3 – 995.

При процессе авторизации, для идентификации и аутентификации используются имя пользователя (полное имя почтового ящика, включая логин, «@» и домен) и пароль (пароль для внешнего приложения). Внешними приложениями считаются все кроме сервисов Mail.ru: Агент, Облако, Почта, Мой Мир.

Для аутентификации через внешнее приложение необходимо заранее через сервисы Mail.ru создать пароль для внешних приложений. Это может быть как один пароль, так и несколько (рекомендуется использовать разные внешние пароли для разных приложений). Так, если злоумышленник получит доступ к внешнему паролю, у него все равно не будет полномочий ко всем возможностям учетной записи, а только к почтовому ящику. При такой утечке внешнего пароля он просто удаляется в настройках учетной записи.

На рисунке 1.2 приведена экранная форма «Пароль для внешнего приложения создан», отображаемая при создании нового пароля для внешних приложений.

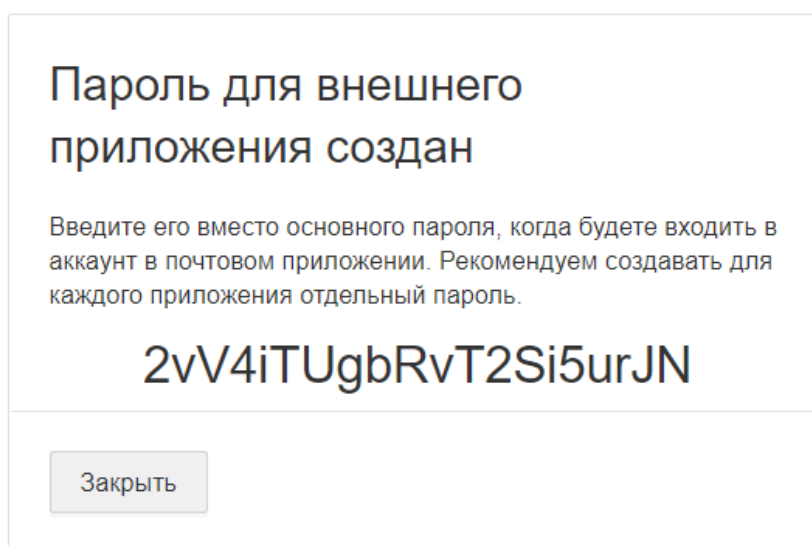


Рисунок 1.2 – Пароль для внешнего приложения

После закрытия данного окна повторный просмотр пароля для внешнего приложения становится недоступен.

### Яндекс.Почта («Яндекс»)

Для доступа к почтовому ящику следует обращаться к серверам «imap.yandex.ru» (IMAP-сервер) и «pop.yandex.ru» (POP3-сервер). В силу использования протокола шифрования SSL/TLS, используемые порты: IMAP – 993, POP3 – 995.

Также как и для доступа к почтовым ящикам Mail.ru, здесь используются полное имя почтового ящика и пароль для внешнего приложения. Особое внимание уделяется тому, что при создании пароля внешнего приложения его можно увидеть только один раз. При его потере рекомендуется удалить потерянный и создать новый.

На рисунке 1.3 представлена экранная форма «Пароли приложений», показывающая все имеющиеся пароли внешних приложений.

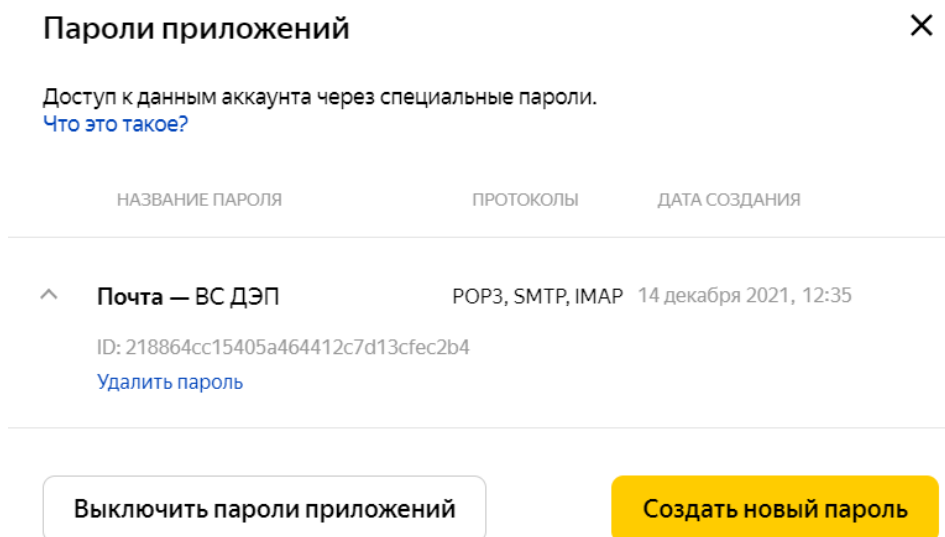


Рисунок 1.3 – Список паролей для внешних приложений

Все созданные пароли внешних приложений сбрасывается если происходит одно из следующих событий:

- смена основного пароля;
- включение/выключение двухфакторной аутентификации;
- восстановление доступа к учетной записи;
- «выход на всех устройствах».

После этого необходимо заново создавать пароли внешних приложений.

## Gmail («Google»)

Для доступа к почтовому ящику следует обращаться к серверам «imap.gmail.com» (IMAP-сервер) и «pop.gmail.com» (POP3-сервер). В силу использования протокола шифрования SSL/TLS, используемые порты: IMAP – 993, POP3 – 995.

Для получения доступа к почтовым ящикам Google для идентификации используется полное имя почтового ящика, но в отличие от mail.ru и yandex.ru, здесь нет обязательного пароля внешних приложений. Для аутентификации используется пароль Gmail (но, если в учетной записи Google включена двухфакторная аутентификация – используется пароль для внешних приложений).

Следует отметить, что во взаимодействии с почтовыми серверами «Google» существуют определённые ограничения. Так, превышение лимита по трафику для протокола IMAP (не более 2500 МБ в день на скачивание и не более 500 МБ в день на загрузку) может привести к блокировке учетной записи. Также для одного аккаунта разрешено не более 15 одновременных подключений по протоколу IMAP. По протоколу POP3 можно включить доступ только на одном клиенте.

## Outlook («Microsoft»)

Для доступа к почтовому ящику следует обращаться к серверу «outlook.office365.com». В силу использования протокола шифрования SSL/TLS, используемые порты: IMAP – 993, POP3 – 995.

Для подключения по протоколу IMAP для этого существует нерешенная проблема – может возникнуть ошибка подключения если подключено несколько клиентов к одной учетной записи.

### 1.2.5 Алгоритм создания одноразовых паролей

Для создания одноразовых паролей в 2005 году был представлен HOTP-алгоритм. Этот алгоритм получил развитие в виде TOTP алгоритма.

HOTP алгоритм – это расширение HOTP, определенного в спецификации RFC 4226, использующий вместо счетчика для генерации пароля – текущее время. Таким образом, производится вычисление, с помощью хеш-функции SHA-1 (описана в спецификации RFC 3174), подтверждающих данных с помощью секретного ключа и текущего времени как на стороне клиента, так и на стороне сервера, после чего эти значения сравниваются.



TOTP определяется как HOTP(K, T), где T - целое число и представляет собой количество временных шагов между начальным временем счетчика T0 и текущим временем Unix.  $T = (\text{Current Unix time} - T0) / X$ , где X – интервальный шаг. Реализация этого алгоритма должна поддерживать значение времени T больше 32-битного целого числа, когда оно выходит за пределы 2038 года.

В силу возможной рассинхронизации часов клиента и сервера, сервером должно быть предусмотрено ограничение в несколько временных шагов. Т.е. если ограничение по количеству временных шагов составляет 2 шага до и после времени проверки данных клиента, то сервер должен обеспечивать прохождение валидации для данных полученных по времени T, T-X, T-2\*X, T+X, T\*2X.

Пример эталонной реализации TOTP алгоритма приводится в спецификации RFC 6238.

### 1.3 Обзор и сравнение существующих аналогов

#### 1.3.1 Microsoft Graph

«Microsoft Graph» - представляет собой набор API, предоставляющий доступ к данным и средствам искусственного интеллекта в Microsoft 365. Благодаря этому обеспечивается единая модель программируемости, которую можно использовать для доступа к колоссальному объему данных в Microsoft 365, Windows 10 и Enterprise Mobility + Security.

Microsoft Graph предоставляет интерфейсы REST API и клиентские библиотеки для доступа к данным в различных облачных службах Майкрософт. Для версии API Microsoft Graph v1.0 конечной точкой является [graph.microsoft.com/v1.0](https://graph.microsoft.com/v1.0)

Microsoft Graph позволяет получать авторизованный доступ к данным почты Outlook в личной или корпоративной учетной записи.

Служба предоставляет доступ к следующему функционалу:

- создание, чтение, ответ, пересылка, отправка, обновление или удаление сообщений;
- запрос сообщений и их получение в папке поиска;
- получение содержимого сообщения или его вложения;
- добавление, получение или удаление вложений сообщения;
- получение настроек языка и часовых поясов для пользователя;

- получение или обновление автоматического ответа, языкового стандарта, часового пояса или рабочего времени пользователя;
- поиск и фильтрация сообщений;
- получение уведомлений об изменениях сообщений в папке;
- синхронизация сообщений или иерархии папок почты.

### 1.3.2 Gmail API

Gmail API представляет собой набор API предоставляемый сервисом `gmail.googleapis.com`. Данный сервис позволяет просматривать и управлять данными почтового ящика Gmail, такими как:

- сообщение – сообщение электронной почты, состоящее из отправителя, получателя, темы и тела письма. Созданное сообщение является неизменяемым.
- поток сообщений – набор связанных сообщений, формирующих переписку-беседу (когда получатель отвечает на сообщение своим собственным сообщением).
- метка – механизм организации сообщений и потоков, позволяющий разделить их на категории. Метки делятся на системные («INBOX» – входящие, «TRASH» – корзина, «SPAM» – спам) и пользовательские.
- черновик – неотправленное сообщение. В отличие от неизменяемых сообщений, черновик до его отправки и преобразования в сообщение может быть изменен.
- возможности, предоставляемые сервисом:
- извлечение данных для чтения, индексирование, резервное копирование;
- автоматическая и программная отправка сообщений;
- миграция учетной записи электронной почты;
- фильтрация и сортировка писем;
- стандартизация подписей электронной почты в организации.

Подробное описание возможностей, предоставляемых сервисом, можно получить по адресу <https://developers.google.com/gmail/api/reference/rest>.

Существенным и самым явным недостатком Gmail API является возможность доступа исключительно к учетным записям почтового ящика Gmail.

### 1.3.3 Mail.ru для бизнеса

Сервис «Mail.ru для Бизнеса» предоставляет почтовый сервис для компаний, с возможностью миграции писем с других почтовых серверов на новые почтовые ящики, созданные для сотрудников компании-пользователя. Часть функционала данного сервиса, в том числе и миграция писем, доступна через API по адресу [biz.mail.ru/api/v1/](https://biz.mail.ru/api/v1/)

### 1.3.4 Сравнение аналогов

На основании проведенного обзора существующих аналогов сформированы параметры для их сравнения:

- возможность взаимодействия с различными почтовыми серверами;
- наличие у служб и сервисов документация для работы с ними;
- формат предоставляемых данных;
- возможность работы с несколькими почтовыми ящиками;
- возможность просматривать сообщения.

В таблице 1.2 отображено сравнение аналогов по основным параметрам.

Таблица 1.2 – Сравнение существующих аналогов

Параметры	Microsoft Graph	Gmail API	Mail.ru для Бизнеса	ВС ДЭП
1	2	3	4	5
Взаимодействие с различными почтовыми серверами	Нет, (только сервер Outlook)	Нет, (только сервер Google)	Производит миграцию писем с почтовых ящиков различных серверов, на свой сервер	Да
Наличие документации	Да, <a href="https://biz.mail.ru/developers/api.html">https://biz.mail.ru/developers/api.html</a>	Да, <a href="https://developers.google.com/gmail/api/reference/rest">https://developers.google.com/gmail/api/reference/rest</a>	Да, <a href="https://docs.microsoft.com/ru-ru/graph/outlook-mail-concept-overview">https://docs.microsoft.com/ru-ru/graph/outlook-mail-concept-overview</a>	Да

Продолжение таблицы 1.2

1	2	3	4	5
Формат данных	JSON	JSON	JSON	JSON, XML
Работа одновременно с несколькими почтовыми ящиками	Нет	Нет	Да	Да
Возможность просмотра сообщений	Да	Да	Нет	Да

Источники информации:

- <https://docs.microsoft.com/ru-ru/graph/>
- <https://developers.google.com/gmail/api>
- <https://biz.mail.ru/developer/api.html>

На основании сведений, приведенных в сравнительной таблице, сделан вывод, что несмотря на наличие решений для взаимодействия с почтовыми серверами, каждое из них не отвечает части из выделенных требований.

Таким образом является обоснованным разработка ВС ДЭП, реализующая взаимодействие с несколькими почтовыми серверами и почтовыми ящиками, поддерживающая более чем один формат данных, с описанной подробной документацией.

#### 1.4 Цель и задачи разработки

Целью данной работы является повышение скорости разработки веб-приложений, включающих работу с различными почтовыми серверами и почтовыми ящиками.

Задачами разрабатываемой веб-службы являются:

- доступ к электронным письмам посредством протокола POP3;
- доступ к электронным письмам посредством протокола IMAP;
- взаимодействие с часто используемыми почтовыми серверами;
- взаимодействие с иными почтовыми серверами по указанию пользователя;
- регистрация учетных записей в службе;

- подключение учетной записи к нескольким электронным почтовым ящикам;
- обеспечение дополнительной защиты учетных записей пользователей двухфакторной системой аутентификации.

### 1.5 Концептуальная модель предметной области

На основе проведенного исследования предметной области и сформированных задач, составим концептуальную модель предметной области.

На рисунке 1.4 представлена инфологическая модель предметной области, наиболее подробно отображающая концептуальную модель.

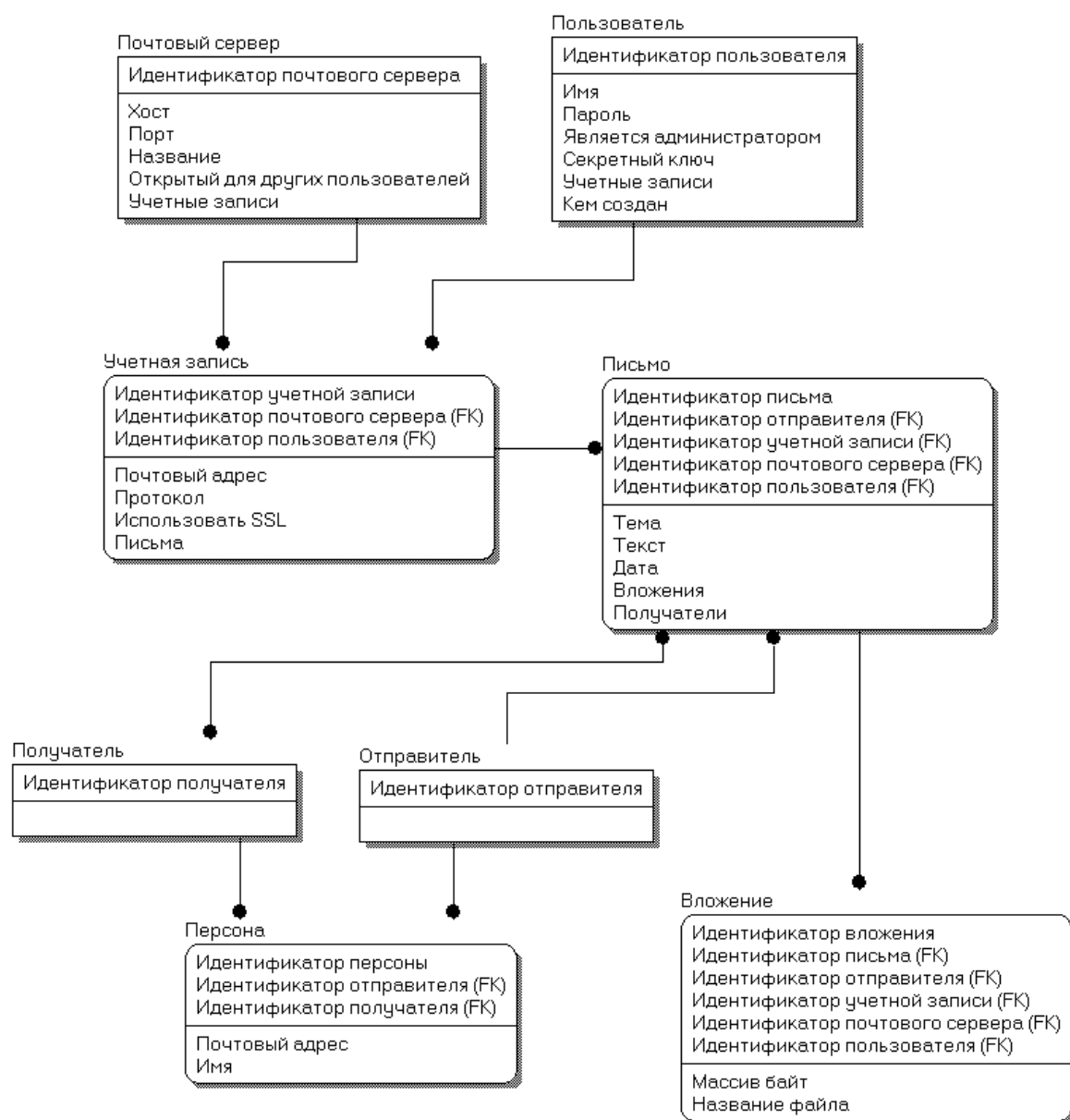


Рисунок 1.4 – Инфологическая модель предметной области

Опишем подробнее назначение сущностей и их атрибутов.

Сущность «Пользователь» представляет собой учетную запись ВС ДЭП. Включает в себя атрибуты имени и пароля, секретный ключ, необходимые чтобы авторизоваться; пользователь также может являться администратором, что дает ему дополнительные полномочия (создание новых пользователей, сброс паролей, настройка почтовых серверов). Атрибуты сущности «Пользователь»:

- S Идентификатор пользователя (целое число больше 0);
- S Имя (строка до 255 символов);
- S Пароль (строка до 255 символов);
- S Является администратором (логический тип);
- D Секретный ключ (строка до 255 символов);
- S Кем создан (целое число больше 0);
- DN Учетные записи (список «Учетная запись»).

Сущность «Учетная запись» хранит в себе информацию об учетных записях почтовых серверов пользователей, а именно: почтовый адрес пользователя, тип протокола подключения, необходимо ли при подключении использовать SSL и ссылку на почтовый сервер. Атрибуты сущности «Учетная запись»:

- S Идентификатор учетной записи (целое число больше 0);
- S Идентификатор пользователя (целое число больше 0);
- S Идентификатор почтового сервера (целое число больше 0);
- S Почтовый адрес (строка до 255 символов);
- S Протокол (строка до 255 символов);
- S Использовать SSL (логический тип);
- DN Письма (список «Письмо»).

Сущность «Почтовый сервер» — это информация необходимая для подключения к почтовому серверу: хост и порт, также хранимые почтовые сервера имеют название и параметр определяющий будет ли отображаться другим пользователям почтовый сервер, предложенный пользователем. Атрибуты сущности «Почтовый сервер»:

- S Идентификатор почтового сервера (целое число больше 0);
- S Хост (строка до 255 символов);
- S Порт (целое число больше 0);
- D Название (строка до 255 символов);

- D Открытый для других пользователей (логический тип);
- DN Учетные записи (список «Учетная запись»).

Сущность «Письмо» хранит в себе информацию, передаваемую в электронном письме. Атрибутами письма являются тема, дата, текст письма, а также отправитель, получатели и вложения. Атрибуты сущности «Письмо»:

- S Идентификатор письма (целое число больше 0);
- S Идентификатор отправителя (целое число больше 0);
- S Идентификатор учетной записи (целое число больше 0);
- S Идентификатор пользователя (целое число больше 0);
- S Тема (строка до 255 символов);
- S Текст (текст);
- S Дата (тип даты-времени);
- S Вложение («Вложение»);
- S Получатели (список «Получатели»).

Сущности «Получатель» и «Отправитель» связывают «Письмо» и сущностью «Персона», которая хранит почтовый адрес получателей и отправителей писем. Атрибуты сущности «Персона»:

- S Идентификатор персоны (целое число больше 0);
- S Идентификатор отправителя (целое число больше 0);
- S Идентификатор получателя (целое число больше 0);
- S Почтовый адрес (строка до 255 символов);
- S Имя (строка до 255 символов).

Сущность «Вложение» необходима для хранения пришедших с письмом файлов: их названий и массива байт. Атрибуты сущности «Вложение»:

- S Идентификатор вложения (целое число больше 0);
- Массив байт (массив байт)
- Название файла (строка до 255 символов).

На основе описанной инфологической модели построим диаграммы «сущность-связь» (диаграмма ER-типа). Диаграмма «сущность-связь» изображена на рисунке 1.5.

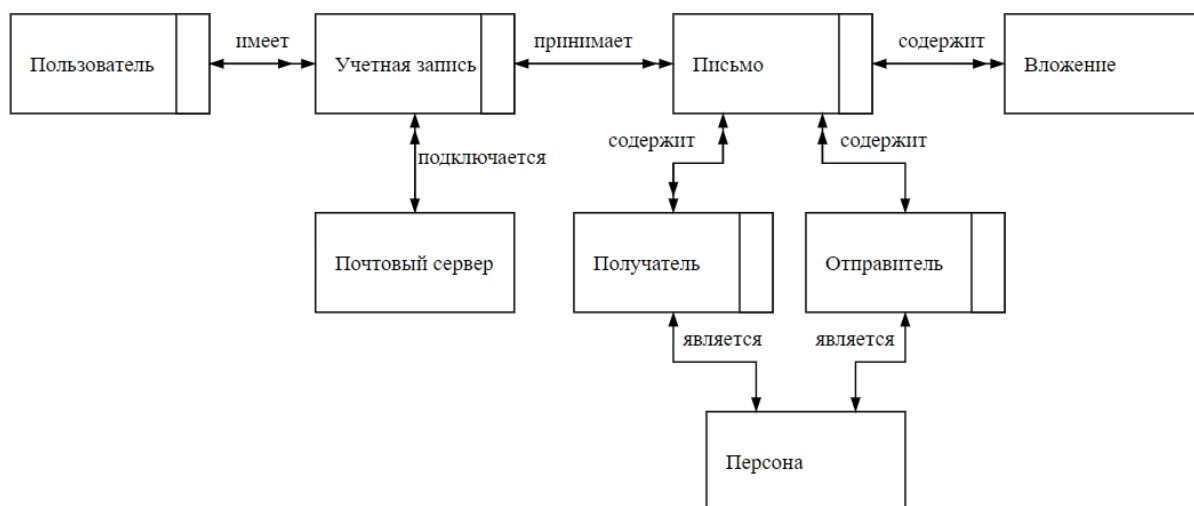


Рисунок 1.5 – Диаграмма «сущность-связь»

Эта диаграмма наглядно показывает, как сущности (entities) соотносятся и взаимодействуют между собой посредством связей (relationships). Связи в такой диаграмме могут иметь тип: «один-к-одному», «один-ко-многим», «многие-ко-многим».

## 1.6 Структура входных и выходных данных

Взаимодействие со службой производится при помощи HTTP-запросов, формируемых на клиентской стороне. Входные данные поступают в формате JSON (либо XML) и представляют из себя:

- данные авторизации;
- запрос на действие с письмом или письмами;
- запрос настройки учетной записи.

Выходными данными является HTTP-ответ. Данные в нем представлены в JSON-формате (либо XML-формате) и являются ответами на запросы:

- сообщение о результате выполнении запроса;
- запрошенным письмом или письмами.

Наглядное отображение преобразование входных данных во внутренне представление и данных во внутреннем представлении в выходные данные изображено на рисунке 1.6.





Рисунок 1.6 – Схема данных ВС ДЭП

В процессе обработки запросов и подготовки ответов данные в ВС ДЭП представлены классами данных, описанных в концептуальной модели. Таким образом при получении запроса и отправки ответа происходит десериализация из формата JSON (или XML) и сериализация этих данных в формат JSON (или XML) соответственно.

### 1.7 Требования к алгоритмам работы программы

На основании информационных потребностей пользователей определены следующие требования к алгоритмам работы программы:

- по протоколу POP3 должны выполняться следующие действия:
  - загрузка писем на устройство пользователя;
  - сохранение писем в базе данных связано с учетной записью пользователя;
- по протоколу IMAP должны выполняться следующие действия:
  - загрузка писем на устройство пользователя (без вложений, с вложениями);

- сохранение писем в базе данных связано с учетной записью пользователя (без вложений, с вложениями);
- установка флагов сообщений;
- удаление сообщений;
- сортировка и фильтрации сообщений при их поиске;
- должно быть реализовано взаимодействие со следующими почтовыми серверами:
  - mail.ru (imap.mail.ru, pop.mail.ru);
  - yandex.ru (imap.yandex.ru, pop.yandex.ru);
  - gmail.com (pop.gmail.com, imap.gmail.com);
  - outlook.com (outlook.office365.com).

#### Выводы по исследовательскому разделу

В исследовательском разделе проведено исследование предметной области, определена актуальность изучаемой проблемы, рассмотрены существующие аналоги (т.е. аналогичных программных средств и методов – программных технологических решений изучаемой проблемы). На основе этого поставлены цель и задачи разработки веб-службы, отвечающей критериям, выявленным при изучении аналогов. После этого описана концептуальная модель предметной области, на основании которой описаны наборы входных и выходных данных, форматы внутреннего представления данных в программе. На основании информационных потребностей пользователей определены требования к алгоритмам работы программы.

## 2 Конструкторский раздел

### 2.1 Выбор языка и среды программирования

Для разработки ВС ДЭП проведен анализ языков программирования, их сравнение. После выбора языка программирования также изучены среды разработки, предназначенные для выбранного языка. Анализ и сравнение языков программирования, а в последствии сред разработки, позволяет выбрать оптимальный вариант для решения поставленных задач.

#### 2.1.1 Выбор языка программирования

Для выбора, наиболее подходящего для реализации задач языка программирования, выделим критерии для анализа.

Прежде всего, язык должен поддерживать парадигму объектно-ориентированного программирования (ООП). В основе данной методологии лежит понятия объектной модели, формируемой с помощью системы классов и объектов. Основными принципами ООП являются: абстракция, инкапсуляция, наследование и полиморфизм.

Объектно-ориентированный подход, помимо прочего, помогает упростить взаимодействие с базами данных, предоставляя удобную возможность для проектирования сущностей. При проектировании ВС ДЭП не обойтись без таких сущностей как «Пользователь», «Письмо».

Однако существуют процессы, которые не должны хранить состояния, но сопоставлять и передавать данные. Процесс авторизации будет более уязвимым, если данные, участвующие в этом процессе, будут открыты к внешним изменениям. Для решения подобных задач применима функциональная парадигма программирования (ФП). Основным принципом ФП можно назвать утверждения:

- функция от тех же аргументов дает те же значения;
- функция при выполнении не изменяет внешних элементов.

Сочетание ООП и ФП позволит спроектировать гибкое и безопасное приложение.

Полезным при разработке будет наличие, как статической, так и динамической типизации данных. С одной стороны, статическая типизация является более безопасной в плане разработки кода, явно помогая обнаружить несоответствие данных на этапе компиляции. С

другой стороны, использование JSON для передачи данных облегчается наличием динамической типизации, упрощающей процесс преобразования данных: сериализации и десериализации.

Во время выполнения программы могут возникать различные ошибки и исключения, как возникающие в самой программе, так и внешние (например, ошибки базы данных). Не все из них можно предугадать и не все возможно устранить. Поэтому для корректной работы программы и избежания её аварийного завершения, необходимо предусмотреть механизмы обработки исключений и ошибок.

В силу использования веб-службы большим числом пользователей необходимо позаботиться о единовременной работе со всеми. Введение многопоточности, несомненно, усложнит архитектуру приложения, но обеспечит более удобную работу с пользователями, выделяя на запросы от каждого отдельные потоки. Но так как потоки не могут выделяться бесконечно, т.е. количество потоков в пуле ограничено, необходимо также предусмотреть возможность асинхронного выполнения операций, которые могут блокировать работу приложения.

Таким образом можно выделить семь основных критериев для выбора языка программирования:

- поддержка объектно-ориентированной парадигмы программирования (ООП);
- поддержка функциональной парадигмы программирования (ФП);
- статическая типизация данных;
- поддержка динамической типизации данных;
- поддержка обработки исключений;
- реализация многопоточности;
- реализация асинхронности.

Наиболее подходящими под эти критерии являются такие языки как Java, C#, JavaScript, Python, PHP, Ruby.

Java – объектно-ориентированный язык программирования со статической типизацией, разработанный изначально для программирования бытовой техники. Сейчас же сфера применения языка гораздо шире: микроэлектроника, клиентские приложения, разработка веб-служб, мобильная разработка. Однако язык не обладает полным набором средств для функционального программирования, а также не поддерживает динамическую типизацию.

C# – объектно-ориентированный язык программирования со статической типизацией. Имеет широкий спектр применения: настольные приложения, разработка игр, веб-сфера, машинное обучение. В последние годы стремительно развивается, чем привлекает к себе больше новых разработчиков, расширяя комьюнити. Язык обладает большим количеством встроенных и open-source библиотек что упрощает процесс разработки, а совместимость языка с другими языками (Managed C++, Visual Basic, F# и прочих .NET-совместимых языков) позволяет существенно расширить возможности программирования. А наличие платформы ASP.NET позволяет создавать веб-приложения и веб-службы различных шаблонов (веб-приложение, веб-приложение с шаблоном модель-представление-контроллер, веб-API, службы RPC).

JavaScript – язык сочетающий в себе объектно-ориентированную и функциональную парадигмы, тесно связанный с веб-разработкой. Он может использоваться как на стороне веб-сервера, для чего в основном используется, платформа Node.js, так и на клиентской стороне: библиотека React.js и встраиваемые в html JS-скрипты. Основными недостатками разработки на JavaScript является отсутствие статической типизация, что увеличивает вероятность возникновения ошибок во время выполнения программы; однопоточная архитектура языка, что вынуждает тратить дополнительное время на эмулирование многопоточности.

Python – язык активно развивающийся в сфере веб разработки благодаря такому фреймворку как Django. Иным направлением языка является машинное обучение, в чем он имеет большую популярность. Преимуществом языка является большое количество библиотек, позволяющих упростить процесс написания кода за счет использования множества готовых решений. Однако существенным недостатком являются отсутствие статической типизации и то, что язык является интерпретируемым, что существенно осложняет своевременное нахождение ошибок в программе.

PHP изначально разрабатывался, как язык для создания персональных веб-страниц. Также как и python является интерпретируемым, с динамической типизацией, что и определяет его недостатки. А ограниченность лишь сферой разработки веб-страниц ограничивает его распространенность в комьюнити, что осложняет процесс его изучения.

Ruby – интерпретируемый динамический язык реализующий в себе парадигму функционального программирования. Для создания веб-приложений и служб создан фреймворк Ruby on Rails. Однако язык и фреймворк не смогли получить сильного распространения в силу малого количества обучающих ресурсов и медленной разработки и развития.

Результаты анализа изучения языков программирования отображены в таблице 2.1.

Таблица 2.1 – Выбор языка программирования

Критерий	Java	C#	JavaScript	Python	PHP	Ruby
Поддержка ООП	Да	Да	Да	Да	Да	Да
Поддержка ФП	Нет	Да	Да	Да	Да	Да
Статическая типизация	Да	Да	Нет	Нет	Нет	Нет
Поддержка динамической типизации	Нет	Да	Да	Да	Да	Да
Обработка исключений	Да	Да	Да	Да	Да	Да
Многопоточность	Да	Да	Нет	Да	Да	Да
Асинхронность	Да	Да	Да	Да	Да	Да

Источники информации:

- <https://docs.oracle.com/en/java/>
- <https://docs.microsoft.com/ru-ru/dotnet/csharp/>
- <https://learn.javascript.ru/>
- <https://www.python.org/>
- <https://www.php.net/>
- <https://www.ruby-lang.org/ru/>

Как видно из проведенного анализа, наиболее подходящим по критериям языком программирования для разработки веб-службы является C#. Для написания веб-службы на языке C# поможет платформа ASP.NET Core.

ASP.NET Core – это кроссплатформенная среда с открытым исходным кодом, предназначенная для создания веб-приложений и веб-служб. Использование платформы .NET Core открывает доступ к ее функциональности и преимуществам, таким как: встроенные механизмы логирования, конфигурирования и внедрения зависимостей, доступ к технологии Entity Framework Core (объектно-ориентированная технология доступа к данным, позволяющая работать с данными базы данных на уровне объектов).

Текущей версией являются C# 10.0 и ASP.NET Core 6.0, вышедшие совместно с платформой .NET 6.0, являющийся развитием платформы .NET Core (начиная с версии 5.0 «Core» убрано из названия).

### 2.1.2 Выбор среды разработки

Наиболее важными критериями выбора среды разработки являются:

- бесплатность распространения;
- наличие автоматического дополнения кода;
- наличие статического анализатора кода;
- наличие отладчика;
- возможность проведения тестирования;
- наличие профилировщиков;
- возможность кроссплатформенной сборки;
- возможность создания проектов по шаблонам.

Такие возможности, как автоматическое дополнение кода и создание проектов по шаблонам, позволят писать код быстрее, не отвлекаясь на детали, общие для всех различных проектов. Отладчик и статический анализатор кода служат для поиска и устранения ошибок в программе. А тестировщик и профилировщик необходимы для гарантии корректных выполнения кода, работы с выделенными ресурсами (память, потоки и т. п.). Также должна быть предусмотрена возможность кроссплатформенной сборки, для последующего запуска на различных платформах.

Наиболее распространёнными средствами разработки на языке C# и платформе .Net являются Visual Studio, Rider, MonoDevelop, SharpDevelop, Eclipse aCute. Отобразим в таблице 2.2 сравнительного анализа этих средств согласно вышеописанным критериям.

Таблица 2.2 – Выбор среды программирования

Критерий	Visual Studio	Rider	MonoDevelop	SharpDevelop	Eclipse aCute
Бесплатность	Да	Бесплатно для студентов	Да	Да	Да
Автоматическое дополнение кода	Да	Да	Да	Да	Да
Статический анализатор кода	Да	Да	Да	Да	Да
Отладчик	Да	Да	Да	Да	Да
Тестирование	Да	Да	Да	Да	Да
Профилирование	Да	Необходимы расширения	Нет	Да	Необходимы расширения
Шаблоны проектов	Да	Да	Нет	Нет	Нет
Кроссплатформенная сборка	Да	Да	Да	Нет	Да

Источники информации:

- <https://visualstudio.microsoft.com/ru/>
- <https://www.jetbrains.com/ru-ru/rider/>
- <https://www.monodevelop.com/>
- <https://ru.wikipedia.org/wiki/SharpDevelop>
- <https://github.com/eclipse/aCute>

Хоть Visual Studio и Rider оба удовлетворяют выделенным критериям, у Visual Studio есть преимущество, заключающееся в том, что она может получить функционал Rider через расширения. Rider расширений с функционалом Visual Studio не имеет. Таким образом наиболее подходящей средой разработки является Visual Studio.



Visual Studio – интегрированная среда разработки, разрабатываемая компанией Microsoft, что идеально подходит для языка C#, разработанного также компанией Microsoft. Язык C# и платформа .NET активно развиваются, а в месте с ними и среда разработки Visual Studio, синхронизирующая свои возможности с новыми возможностями языка быстрее, других сред разработки.

## 2.2 Алгоритм и методы работы ВС ДЭП

### 2.2.1 Алгоритм работы ВС ДЭП

Алгоритм работы представляет собой взаимодействие с письмами на электронных почтовых серверах по протоколам IMAP и POP3. Изобразим алгоритм работы ВС ДЭП в виде диаграммы на рисунке 2.1.

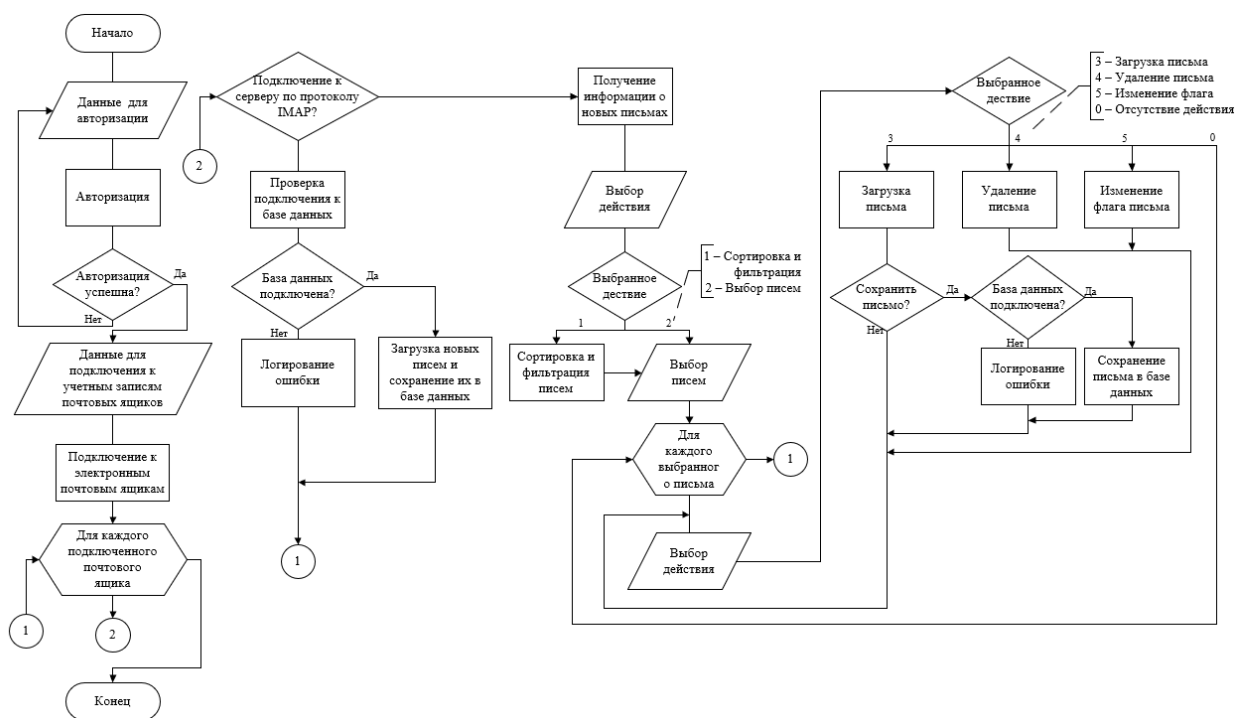


Рисунок 2.1 – Схема алгоритма ВС ДЭП

В первую очередь происходит авторизация пользователя с помощью логина, пароля и одноразового пароля-ключа. После этого пользователь подключается к почтовым серверам. Для этого он передает учетные данные, которые сохраняются в базе и при последующих входах в учетную запись ВС ДЭП, подключение к почтовым серверам будет происходить автоматически. В зависимости от выбранного протокола происходит подключение к IMAP или POP3 серверу. После этого пользователь получает доступ к письмам и функциям работы с ними. Письма с сервера POP3 могут быть загружены и сохранены в базе данных.

Для протокола ИМАР кроме загрузки, сохранения и удаления доступны функции фильтрации писем, их сортировки, редактирования флагов писем. В случае возникновения ошибок на каком-либо из этапов они обрабатываются и логируются.

### 2.2.2 Логирование

В случае возникновения ошибок во время работы программы, помимо их обработки, необходимо сохранять сообщения об ошибках с целью дальнейшей их диагностики и устранения причины ошибок. Одной из наиболее популярных библиотек логирования для платформы .Net является Serilog (более 400 миллионов скачиваний). Для сравнения две другие популярные библиотеки логирования log4net и NLog имеют 113 и 142 миллиона скачиваний соответственно.

Serilog предоставляет различные уровни логирования событий, такие как: verbose, debug, information, warning, error и fatal. Журнал можно вести в консоль, файл, локальный или облачный сервер, базу данных, очередь сообщений либо свой собственный приемник. При этом не обязательно ограничиваться лишь одним приемником – можно использовать несколько, определив для каждого свою конфигурацию. При записи в файл в конфигурации определяются такие параметры как предельный размер для файла (после чего запись идет в новый файл), либо срок записи в один файл (по истечении которого создается новый файл, например 1 день). Записи логов также сопровождаются вспомогательной контекстной информацией: дата, время, поток в котором вызвано событие, уровень события и т. д.).

Так, в ВС ДЭП ведется логирование в консоль, для чтения сообщений и ошибок в реальном времени, и в файл, для последующего анализа. Имя файлов и предельный размер конфигурируются пользователем, а интервал записи в файл – 1 день. В листинге 2.1 приведен код настройка конфигурации логирования в ВС ДЭП.

#### Листинг 2.1 – Конфигурация логирования в ВС ДЭП

```
#region serilog configuration

var logTemplateConsole = "[{Level:u3}] <{ThreadId}> :: {Message:l}{New-
Line}{Exception}";
var logTemplateFile =
    "{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} [{Level:u3}] <{ThreadId}> ::
{Message:l}{NewLine}{Exception}";
```

```

if (!Directory.Exists(config.Logger.FilePath))
    try
    {
        Directory.CreateDirectory(config.Logger.FilePath);
        log.Information($"create directory {config.Logger.FilePath} for
logs");
    }
    catch
    {
        log.Error("Can't find and create directory for logs");
        return;
    }

builder.Host.UseSerilog((context, services, configuration) => configuration
    .ReadFrom.Configuration(context.Configuration)
    .ReadFrom.Services(services)
    .Enrich.FromLogContext()
    .Enrich.WithThreadId()
    .WriteTo.Console(outputTemplate: logTemplateConsole)
    .WriteTo.File(
        outputTemplate: logTemplateFile,
        path: Path.Combine(config.Logger.FilePath, config.Logger.FileName),
        shared: true,
        rollingInterval: RollingInterval.Day,
        fileSizeLimitBytes: config.Logger.LimitFileSize
    )
);
#endregion

```

Фрагмент возможного файла-лога и консоли-лога изображены на рисунке 2.2. и рисунке 2.3 соответственно

```
Iris20220503.log X
D:\Study MIET\7 семестр\VCR\Iris\logs> iris 20220503.log
102 at System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler.ValidateJwtS(String token, TokenValidationParameters validationParameters, BaseConf
103 --- End of stack trace from previous location ---
104 at System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler.ValidateToken(String token, JwtSecurityToken outerToken, TokenValidationParameters
105 at System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler.ValidateToken(String token, TokenValidationParameters validationParameters, Secur
106 at Microsoft.AspNetCore.Authentication.JwtBearer.JwtBearerHandler.HandleAuthenticateAsync()
107 2022-05-03 10:13:44.501 +03:00 [INF] <10> :: Bearer was not authenticated. Failure message: IDX10223: Lifetime validation failed. The token is e
108 2022-05-03 10:13:44.501 +03:00 [INF] <10> :: Bearer was not authenticated. Failure message: IDX10223: Lifetime validation failed. The token is e
109 2022-05-03 10:13:44.502 +03:00 [INF] <10> :: Executing endpoint 'Iris.Api.Controllers.AuthControllers.AuthController.IsAuth (Iris)'
110 2022-05-03 10:13:44.503 +03:00 [INF] <10> :: Route matched with {action = "IsAuth", controller = "Auth"}. Executing controller action with signal
111 2022-05-03 10:13:44.506 +03:00 [INF] <10> :: Executing OkObjectResult, writing value of type 'System.Boolean'.
112 2022-05-03 10:13:44.508 +03:00 [INF] <10> :: Executed action 'Iris.Api.Controllers.AuthControllers.AuthController.IsAuth (Iris)' in 3.4768ms
113 2022-05-03 10:13:44.508 +03:00 [INF] <10> :: Executed endpoint 'Iris.Api.Controllers.AuthControllers.AuthController.IsAuth (Iris)'
114 2022-05-03 10:13:44.509 +03:00 [INF] <10> :: HTTP GET /api/authorize/isauth responded 200 in 1239.5717 ms
115 2022-05-03 10:13:44.510 +03:00 [INF] <10> :: Request finished HTTP/1.1 GET https://localhost:44468/api/authorize/isauth - - 200 - application/
116 2022-05-03 10:14:02.862 +03:00 [INF] <4> :: Request starting HTTP/1.1 PUT https://localhost:7208/api/authorize/ZDY4YTk0NzAtYzVlMi00YmESLThYmItM
117 2022-05-03 10:14:02.865 +03:00 [INF] <4> :: Executing endpoint 'Iris.Api.Controllers.AuthControllers.AuthController.ExecuteAuthorization (Iris)'
118 2022-05-03 10:14:02.876 +03:00 [INF] <4> :: Route matched with {action = "ExecuteAuthorization", controller = "Auth"}. Executing controller acti
119 2022-05-03 10:14:06.636 +03:00 [ERR] <4> :: Ошибка авторизации пользователя admin
120 2022-05-03 10:14:16.429 +03:00 [INF] <4> :: Executed action 'Iris.Api.Controllers.AuthControllers.AuthController.ExecuteAuthorization (Iris)' in 1
121 2022-05-03 10:14:16.431 +03:00 [INF] <4> :: Executed endpoint 'Iris.Api.Controllers.AuthControllers.AuthController.ExecuteAuthorization (Iris)'
122 2022-05-03 10:14:16.457 +03:00 [ERR] <4> :: HTTP PUT /api/authorize/ZDY4YTk0NzAtYzVlMi00YmESLThYmItMjBkMTM4MGMzNDky responded 500 in 13592.9039
123 Iris.Exceptions.AuthException: Iris process error (Неверное имя пользователя или пароль)
124 at Iris.Services.AuthService.AuthService.Authenticate(AuthRequestOperation operation, AuthRequestContract authRequest) in D:\Study MIET\7 семестр
125 at Iris.Api.Controllers.AuthControllers.AuthController.ExecuteAuthorization(String id, AuthRequestContract authRequest) in D:\Study MIET\7 ce
126 at lambda_method70(Closure , Object , Object[])
127 at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor.SyncActionResultExecutor.Execute(IActionResultTypeMapper mapper, ObjectMethod
128 at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeActionMethodAsync()
129 at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(State& next, Scope& scope, Object& state, Boolean& isCompleted)
130 at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeNextActionFilterAsync()
131 --- End of stack trace from previous location ---
132 at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Rethrow(ActionExecutedContextSealed context)
133 at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(State& next, Scope& scope, Object& state, Boolean& isCompleted)
```

Рисунок 2.2 – Пример фрагмента файла лога

```
Консоль отладки Microsoft Visual Studio
[INF] <7> :: Executed action 'Iris.Api.Controllers.AuthControllers.AuthController.InitAuth (Iris)' in 512.1827ms
[INF] <7> :: Executed endpoint 'Iris.Api.Controllers.AuthControllers.AuthController.InitAuth (Iris)'
[INF] <7> :: HTTP POST /api/authorize responded 201 in 516.4065 ms
[INF] <7> :: Request finished HTTP/1.1 POST https://localhost:44468/api/authorize - 0 - 201 - text/plain; charset=utf-8
518.8165ms
[INF] <7> :: Request starting HTTP/1.1 PUT https://localhost:44468/api/authorize/MDZhNTFkZGItMWQ3ZS00MGQyLWFLYmQtZWVhZmVhYmE5ZTkx
application/json; charset=utf-8 37
[INF] <7> :: Executing endpoint 'Iris.Api.Controllers.AuthControllers.AuthController.ExecuteAuthorization (Iris)'
[INF] <7> :: Route matched with {action = "ExecuteAuthorization", controller = "Auth"}. Executing controller action with
signature Microsoft.AspNetCore.Mvc.IActionResult ExecuteAuthorization(System.String, Iris.Api.Controllers.AuthController
AuthRequestContract) on controller 'Iris.Api.Controllers.AuthControllers.AuthController (Iris)'.
[ERR] <7> :: Ошибка авторизации пользователя admin
[INF] <7> :: Executed action 'Iris.Api.Controllers.AuthControllers.AuthController.ExecuteAuthorization (Iris)' in 2835.652
ms
[INF] <7> :: Executed endpoint 'Iris.Api.Controllers.AuthControllers.AuthController.ExecuteAuthorization (Iris)'
[ERR] <7> :: HTTP PUT /api/authorize/MDZhNTFkZGItMWQ3ZS00MGQyLWFLYmQtZWVhZmVhYmE5ZTkx responded 500 in 2859.9747 ms
Iris.Exceptions.AuthException: Iris process error (Неверное имя пользователя или пароль)
at Iris.Services.AuthService.AuthService.Authenticate(AuthRequestOperation operation, String login, String password) in
D:\Study MIET\7 семестр\VCR\Iris\Iris\Services\AuthService\AuthService.cs:line 32
at Iris.Api.Controllers.AuthControllers.AuthController.ExecuteAuthorization(String id, AuthRequestContract authRequest
t) in D:\Study MIET\7 семестр\VCR\Iris\Iris\Api\Controllers\AuthControllers\AuthController.cs:line 74
at lambda_method65(Closure , Object , Object[])
at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor.SyncActionResultExecutor.Execute(IActionResultTypeMap
per mapper, ObjectMethodExecutor executor, Object controller, Object[] arguments)
at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeActionMethodAsync()
at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(State& next, Scope& scope, Object& state, Boo
lean& isCompleted)
at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.InvokeNextActionFilterAsync()
--- End of stack trace from previous location ---
at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Rethrow(ActionExecutedContextSealed context)
```

Рисунок 2.3 – Пример фрагмента консоли лога

Логирование в консоль используется на этапе разработке программы и ее отладки. Поэтому при логировании в консоль нет необходимости указывать дату и время записи, т.к. записи наблюдаются в режиме реального времени.

### 2.2.3 Документирование API

Для создания документации по API в C# используются структурированные XML-комментарии. На основе этих комментариев компилятором создается XML-файл, в котором содержатся структурированные данные: комментарии и сигнатуры API. В дальнейшем

этот файл обрабатывается другими средствами для преобразования в удобочитаемую форму.

Основные теги XML-комментариев документации:

- `<summary>` – описание типа или метода;
- `<param>` – описание параметра метода;
- `<returns>` – описание возвращаемого значения;
- `<exception>` – описание возможных исключений;
- `<example>` – пример использования элемента;
- `<code>` – указание нескольких строк кода;
- `<inheritdoc>` – ссылка на переиспользование текста документации.

Одним из способов описания REST API документация является Swagger использующий спецификацию OpenAPI3.0. Для платформы .NET реализацией Swagger является Swashbuckle.

Особенностью интерфейса Swagger-документации является возможность запустить и протестировать каждый из методов, что добавляет документации большую наглядность. Пример фрагмента сформированного интерфейса документации изображен на рисунке 2.4.

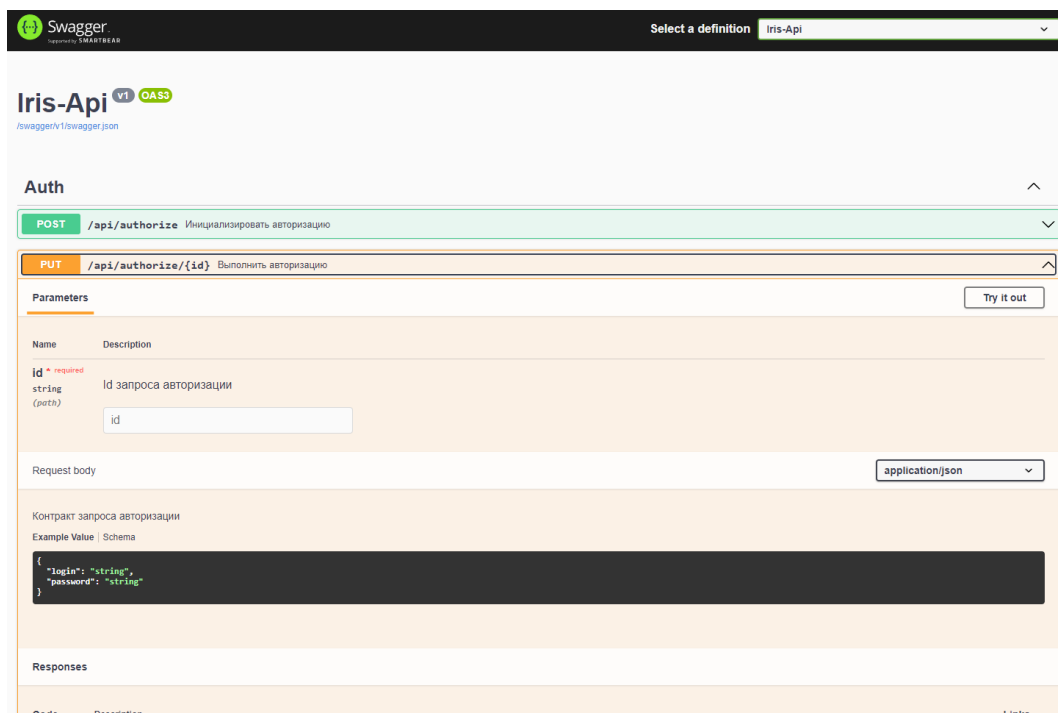


Рисунок 2.4 – Пользовательский интерфейс Swagger-документации.

Конфигурация Swagger в реализации Swashbuckle предлагает к настройке следующие параметры:

- заголовок документации;
- описание;
- версия API;
- контакты разработчика;
- лицензия.

В листинге 2.2 приведен код настройки генерации интерфейса документации.

#### Листинг 2.2. – Настройка Swagger

```
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Title = "Iris-API",
        Version = "v1"
    });

    var executingLocation = Assembly.GetExecutingAssembly().Location;
    var xmlName = $"{Path.GetFileNameWithoutExtension(executingLocation)}.xml";
    var xmlPath = Path.Combine(Path.GetDirectoryName(executingLocation),
xmlName);
    c.IncludeXmlComments(xmlPath);
});
```

Текущая версия документации для ВС ДЭП – «v1», ее название – «Iris-API».

#### 2.2.4 Система управления базами данных и база данных

Для использования базы данных в ВС ДЭП используется встраиваемая СУБД SQLite. Это легко-встраиваемая в программы СУБД, не имеющая отдельной программы-сервера, а существующая как библиотека и взаимодействующая с программами с помощью своего специального API. В силу отсутствия сервера вся база данных хранится в едином файле на компьютере, на котором установлена ВС ДЭП. Таким образом уменьшается время отклика базы данных и расходы на транзакции.

Так как SQLite является составной частью ВС ДЭП, она не требует от пользователя наличия отдельно установленных специализированных программ.

SQLite является кроссплатформенной, а для языка программирования C# провайдер поставляется библиотекой Microsoft.EntityFrameworkCore.Sqlite. Для администрирования базы данных созданной СУБД SQLite существуют такие программы, как: DB Browser, представляющий минимальный действий с базой данных (создание баз данных и таблиц, их редактирование; добавление, редактирование и удаление данных) и DataGrip, позволяющий также визуализировать диаграмму связей между таблицами (изображена на рисунке 2.5)

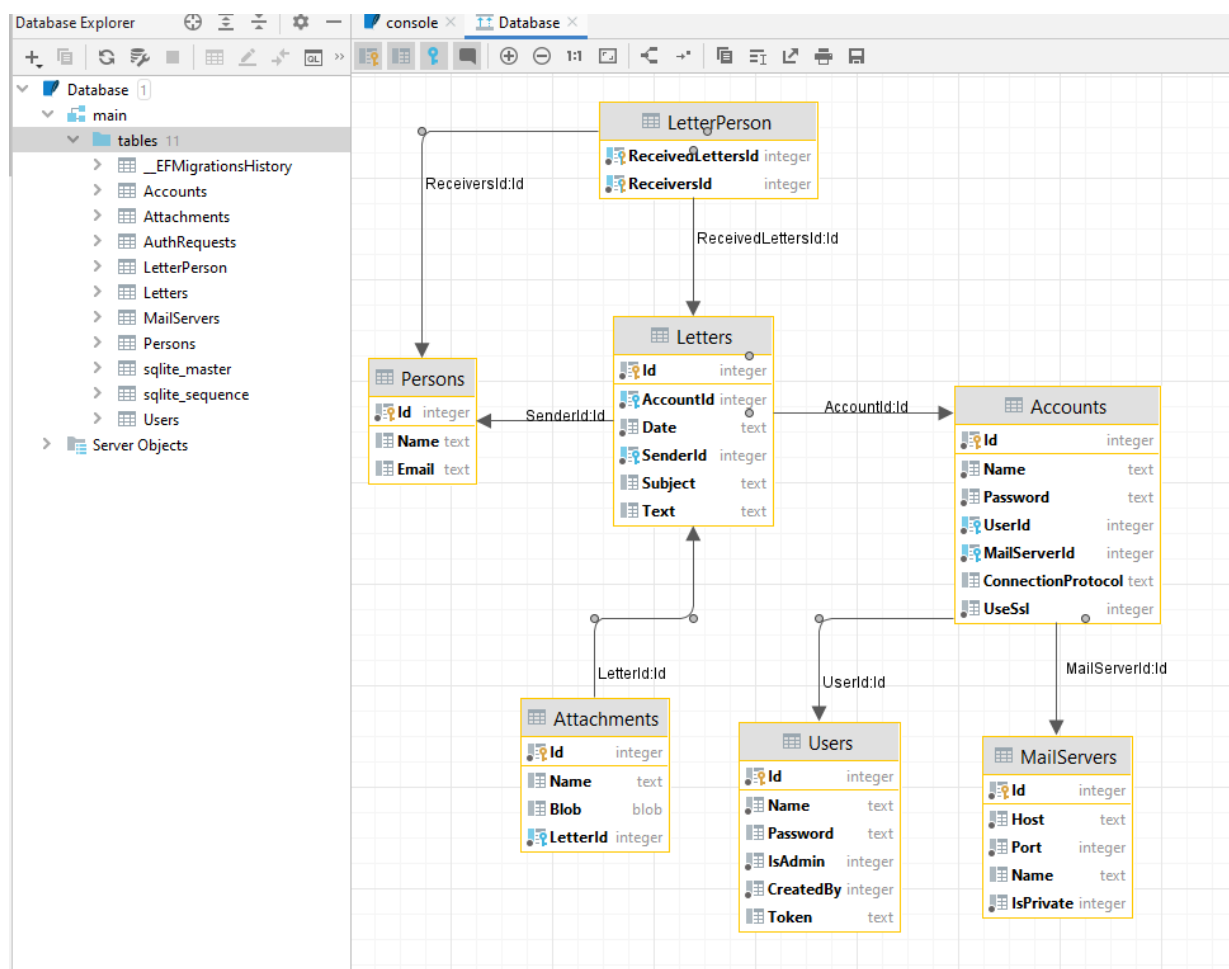


Рисунок 2.5 – Диаграмма связей между таблицами базы данных ВС ДЭП

Для работы с таблицами базы данных и представлением их в виде классов, присущих ООП применяется технология ORM (Object-Relational Mapping) – объектно-реляционное отображение. Для проекта ВС ДЭП использована ORM-технология платформы .Net – Entity Framework Core (EF Core). Эта технология представляет таблицы как классы, а атрибуты –

свойствами классов, что позволяет взаимодействовать с данными стандартными ООП-методами. Существует два подхода для связи таблиц с классами в программе:

- code-first (использованный в ВС ДЭП), когда сначала описываются классы в коде программы а затем EF Core создает базу данных с таблицами; если же в коде описание классов будет обновлена, EF Core предоставляет механизм миграций – последовательного применения изменений схемы к базе данных;
- database-first, когда таблицы создаются при помощи СУБД, а затем EF Core генерирует классы в коде.

### 2.2.5 Внедрение зависимостей

Одним из архитектурных принципов ООП является концепция инверсии управления (Inversion of Control). Принцип завязан на слабом связывании классов. Для этого проектируемые классы должны отвечать за независимые задачи и не должны зависеть от конкретной реализации друг друга. Одной из реализаций инверсии управления является процесс внедрения зависимостей, при которой:

- модули более высокого уровня не зависят от модулей нижнего уровня, но все модули зависят от абстракций;
- абстракции не зависят от конкретных реализаций.

Таким образом достигается гибкость динамической смены реализаций.

Платформа ASP .NET Core поддерживает механизм внедрения зависимостей через конструктора. В такой реализации для абстрагирования используются интерфейсы; зависимости реализаций с абстракцией регистрируются в специально предоставляемом контейнере служб, представленном интерфейсом `IServiceProvider`; службы, добавленные через провайдера в коллекцию `IServiceCollection` внедряются через конструктор в классы в которых необходима их реализация. Жизненный цикл создаваемых служб представляется тремя вариантами:

- «единственный» (singleton): экземпляр создается один раз при первом запросе;
- «область действия» (scoped): экземпляр создается при каждом новом запросе и уничтожается по завершении запроса;
- «временный» (transient): новый экземпляр создается при каждом обращении.

В листинге 2.3 приведен фрагмент кода внедрения зависимостей в контейнер



### Листинг 2.3 – Регистрация зависимостей

```
builder.Services.AddSingleton(config);
builder.Services.AddSingleton(dbContext);
builder.Services.AddSingleton<IServerConnectionStore, ServerConnection-
Store>();
builder.Services.AddSingleton<IAuthRequestsStore, AuthRequestsStore>();
builder.Services.AddSingleton<ITokensStore, TokensStore>();
builder.Services.AddSingleton<IUserService, UserService>();
builder.Services.AddSingleton<IAuthService, AuthService>();
builder.Services.AddSingleton<IConnectionProtocolHelperService, Connec-
tionProtocolHelperService>();
builder.Services.AddScoped<IClaimsPrincipalHelperService, ClaimsPrinci-
palHelperService>();
builder.Services.AddSingleton<IMailServersService, MailServersService>();
builder.Services.AddScoped<ILetterService, LetterService>();
builder.Services.AddScoped<IFormatLettersSevice, FormatLettersSevice>();
builder.Services.AddScoped<IRegistrationService, RegistrationService>();
builder.Services.AddScoped<IPop3ClientService, Pop3ClientService>();
builder.Services.AddScoped<IImapClientService, ImapClientService>();
builder.Services.AddScoped<IAccountsService, AccountsService>();
```

Экземпляр конфига и контекст базы данных передаются в хранилище зависимостей напрямую, а сервисы представляются интерфейсами. При этом сервисы-хранилища и связанные с ними сервисы регистрируются циклом «singleton», а остальные сервисы – «циклом scoped»

#### 2.2.6 Двухфакторная аутентификация

Для взаимодействия с мобильным приложением генерации TOTP-паролей в ВС ДЭП используется библиотека TwoStepsAuthenticator, реализующая алгоритм генерации согласно стандарту RFC 6238. Этот стандарт соблюдается мобильными приложениями Microsoft Authenticator и Google Authenticator.

Алгоритм работы библиотеки на серверной стороне:

- генерация секретного ключа (листинг 2.4);

- получение от клиента кода, сгенерированного мобильным приложением, и генерация на сервере одноразового пароля с его проверкой (листинг 2.5).

#### Листинг 2.4 – генерация секретного ключа

```
var key = TwoStepsAuthenticator.Authenticator.GenerateKey();
```

#### Листинг 2.5 – генерация одноразового пароля и его проверка

```
var secret = user.secretAuthToken;  
var code = Request.Form["код"];  
var authenticator = new TwoStepsAuthenticator.TimeAuthenticator();  
bool isok = authenticator.CheckCode(secret, code);
```

Для предотвращения повторного использования одного кода библиотека предлагает к реализации интерфейс `IUsedCodesManager` с методами:

- `void AddCode(ulong challenge, string code, object user);`
- `bool IsCodeUsed(ulong challenge, string code, object user);`

Экземпляр интерфейса передается в конструктор класса `TimeAuthenticator`. После чего в метод этого класса `bool CheckCode(secret, code, user);` добавляется третьим параметром `user`.

## 2.3 Пользовательский интерфейс ВС ДЭП

### 2.3.1 Технологии разработки пользовательского интерфейса

Наиболее распространённой практикой разработки пользовательского веб-интерфейса являются сочетание HTML, CSS, JavaScript.

HTML (HyperText Markup Language, язык гипертекстовой разметки) – язык разметки документов для просмотра веб-страниц. Он используется для разметки веб-страниц. CSS (Cascading Style Sheets, каскадные таблицы стилей) – язык описания внешнего вида документа написанного с помощью языка разметки HTML. Он позволяет добавить документу внешнее оформление. JavaScript используется в этой связке для реализации интерактивности документов с пользователем, связи с сервером.

Однако несмотря на все возможности JavaScript, использование его в этом стеке без дополнительных средств является трудной задачей. Поэтому в этот стек технологий часто

добавляется библиотека jQuery существенно расширяющая возможность JavaScript по динамическому взаимодействию с элементами HTML. Либо же используются специализированные фреймворки для создания веб-интерфейсов Angular, Vue.js и библиотека Resct.js.

Для реализации пользовательского интерфейса ВС ДЭП выбрана библиотека React.js. React.js – это open-source JavaScript библиотека, позволяющая быстро создавать веб-интерфейсы, преимущественно одностраничного типа. В её основе лежит проектирование отдельных компонент, позволяющих переиспользовать код множество раз, тем самым реализуется принцип not repeat yourself, отсутствующий в стеке HTML + CDD + JavaScript.

К особенностям проектирование интерфейсов на React.js относятся:

- реализация HTML-разметки непосредственно в JavaScript коде. Это реализуется благодаря расширению JSX (JavaScript XML), которое позволяет писать HTML-блоки непосредственно в JavaScript функциях с последующим рендерингом. С помощью JSX реализуется определение отдельных компонент, которые потом используются в других компонентах;
- виртуальный DOM, позволяющий перегружать только измененные компоненты, а не всю страницу целиком;
- одностраничный шаблон, что позволяет загрузить страницу целиком – это позволяет экономить трафик пользователя т.к. при переключении между вкладками ему не придется подгружать код страниц каждый раз заново;
- возможность использовать написанный код не только для веб-интерфесов, но и для настольных и мобильных приложений с помощью расширения React Native;
- Расширение CSS до module.css что позволяет определять стили для отдельных компонент, избавляя от проблемы необходимости уникальных идентификаторов для HTML-элементов для из взаимосвязи с отдельным стилем.

### 2.3.2 Переопределяемые компоненты

React.js позволяет определить компоненты для повторного использования, используемые во всем проекте. К таким компонентам, например, можно отнести кнопки, формы ввода, выпадающие списки и так далее. Примеры определения компонент отображены в листингах 2.6 и 2.7. В листинге 2.8 приведен фрагмент кода с вызовом переопределяемого компонента «Форма ввода».

### Листинг 2.6 – реализация компонента «Форма ввода»

```
import React from 'react';
import styleClasses from './IrisInput.module.css';
const IrisInput = React.forwardRef((props, ref) => {
  return (
    <input ref={ref} className={styleClasses.irisInput} {...props}/>
  );
});
export default IrisInput;
```

### Листинг 2.7 – реализация компонента «Кнопка»

```
import React from 'react';
import styleClasses from './IrisButton.module.css';
const IrisButton = ({children, ...props}) => {
  return (
    <button className={styleClasses.irisButton} {...props}>
      {children}
    </button>
  );
};
export default IrisButton;
```

### Листинг 2.8 – применение компонента «Форма ввода»

```
<div className={styleClasses.central}>
  <IrisInput
    value={password}
    onChange={e => setPassword(e.target.value)}
    type="password"
    placeholder="Пароль"
    className={` ${irisInputStyles.irisInput} ${styleClasses.input}`}
  />
</div>
```

Как видно из листингов 2.6, 2.8 связывание данных в React.js является односторонним: от родительского компонента дочернему. Изменение состояния родительского компонента дочерним происходит с помощью передач в дочерний компонент callback-функций.

### 2.3.3 Форма авторизации

Форма авторизации изображена на рисунке 2.6.

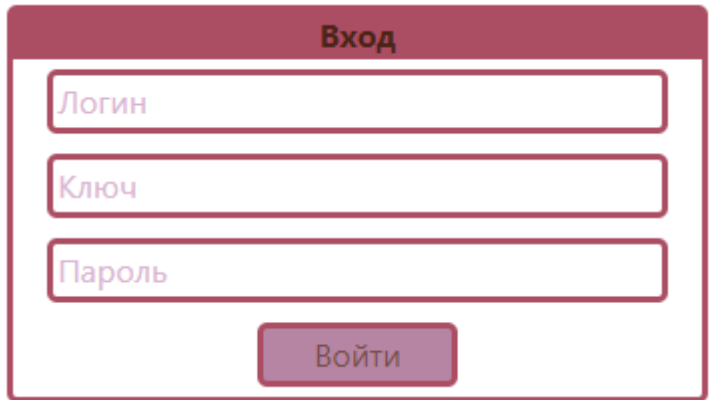


Рисунок 2.6. Форма авторизации

Форма авторизации состоит из трех полей ввода: логин, с типом текст; ключ, и пароль, которые имеют тип пароль, что не дает посторонним увидеть, что вводит пользователь, заменяя символы на символы точек.

### 2.3.4 Компоненты просмотра писем

Для просмотра писем существует таблица, в которой собраны все письма. В таблице отображены получатель и отправитель письма, его статус, начало письма, наличие вложений и время доставки получения (или отправки) письма, флажок для групповых операций с письмами. Для удобства просмотра писем таблица разбита на страницы – число писем на страницы можно изменять выпадающим списком. Пример таблицы изображен на рисунке 2.7.

Получатель	Статус	Отправитель	Письмо	Вложения	Дата и время
<input type="checkbox"/> mJalksJ@mai...	Прочитано	Sunlight	🔥Специальное предложение на часы GUESS🔥	Добавьте немного шика...	21.03.2022 17...
<input type="checkbox"/> maksim.m00...	Не прочитано	orioks@miet.ru	ОРИОКС - Задание по Лабораторной работе 4	Уважаемый пользовател...	21.03.2022 14...
<input type="checkbox"/> bof_de@inbo...	Черновик	maksim.m00...	Отчет 3. ПЗ 3. Предпринимательская деятельность. ПИН-44	Мясников М...	Отчет 3. ПЗ 3... 20.03.2022 15...
<input type="checkbox"/> bof_de@inbo...	Отправлено	maksim.m00...	Отчет 2. ПЗ 2. Рынок и рыночное регулирование экономики. ПИН-44	М...	Отчет 2. ПЗ 2... 20.03.2022 15...
<input type="checkbox"/> maksim.m00...	Прочитано	Тинькофф Ба...	Как подружить ребенка с деньгами		20.03.2022 11...

<< 1 5 >>

5  
10  
15  
25

Рисунок 2.7 – Таблица просмотра писем

Нажав на строку таблицы открывается окно для подобного просмотра письма. В нем отображается полный текст письма, а также становятся доступными к скачиванию вложения, прикрепленные к письму. Пример формы просмотра письма изображен на рисунке 2.8.

Отчет 3. ПЗ 3. Предпринимательская деятельность. ПИН-44 Мясников М.А.	
От кого: maksim.m00@mail.ru	20.03.2022 15:39
Кому: bof_de@inbox.ru	
Вложения: <a href="#">Отчет 3. ПЗ 3. Предпринимательская деятельность. ПИН-44 Мясников М.А.docx</a>	
Письмо:  -- С уважением, Максим Мясников, ПИН-44	

Рисунок 2.8 – Форма просмотра письма

### 2.3.5 Компоненты работы с почтовыми серверами

Пользователь может добавить новый почтовый сервер. Для этого необходимо ввести адрес хоста и номер порта, название под которым будет отображаться сервер и указать должен ли сервер отображаться в списках у других пользователей. Интерфейс формы добавления нового сервера изображен на рисунке 2.9.

Новый сервер

Хост

Порт

Название

☐ Публичный

Добавить

Рисунок 2.9 – Форма добавления почтового сервера

Также пользователь может редактировать созданный им сервер. В этом случае на форме доступны к редактированию поле «Название» и флаг «Публичный». Интерфейс данной формы изображен на рисунке 2.10.

Рисунок 2.10 – Форма редактирования почтового сервера

После добавления почтового сервера он будет отображаться в выпадающем списке в форме добавления новой учетной записи почтового сервера. Пример формы добавления учетной записи и список выбора сервера на этой форме изображены на рисунке 2.11.

Рисунок 2.11 – Форма добавления учетной записи, выбор хоста

На этой форме помимо сервера также указываются почтовый адрес, протокол подключения к серверу и необходимость подключаться с использованием SSL.

#### Выводы по конструкторскому разделу

В конструкторском разделе на основе разработанной в исследовательском разделе концептуальной модели сформированы критерии для выбора языка программирования и среды разработки, проведено исследования языков программирования и выбран С# как наиболее полно отвечающий критериям, а затем изучены среды программирования для этого языка с конечным выбором Visual Studio. После этого на основе концептуальной мо-

дели, входных и выходных данных, их внутреннем представлении и потребностей пользователей разработан и описан алгоритм и методы решения поставленных задач, описан процесс реализации разработанных моделей и алгоритмов. Также описан процесс разработки пользовательского интерфейса.



### 3 Испытательный раздел

#### 3.1 Средства отладки программы

Под отладкой в первую очередь понимается процесс устранения ошибок в коде. Иначе же процесс отладки можно обозначить как пошаговое выполнение кода в программной среде. При этом отлаживаемые ошибки могут быть совершенно разными: начиная от опечаток, попытки обращения к необъявленным данным и ошибок преобразования типов и заканчивая ошибками в логике программы.

Первая группа ошибок в языке C# диагностируется с помощью статического анализатора кода среды разработки – такие ошибки будут обнаружены на этапе компиляции программы. Вторая группа ошибок может быть диагностирована лишь во время выполнения программы. В таком случае для отслеживания ошибок применяется ведение журнала ошибок – лога. В дальнейшем, по записям лога определяется место возникновения ошибки и этот блок проверяется с помощью специализированного отладчика Visual Studio.

##### 3.1.1 Отладка с использованием статического анализатора кода

Среда разработки Visual Studio предоставляет интерфейс просмотра ошибок (окно «Список ошибок»). Для статического анализа применяется инструмент IntelliSense – вспомогательное средство для написания кода, предоставляющая сведения об используемом коде. Сведения могут быть одного из трех видов:

- ошибки – без их устранения проект не будет скомпилирован;
- предупреждения – сообщения о нежелательных конструкциях в коде;
- сообщения – элементы кода, которые необходимо исправить, но они не приведут к ошибке.

Для полного понимания ошибок отображается полная информация о них:

- код ошибки;
- описание ошибки;
- проект, в котором найдена ошибка;
- файл, в котором найдена ошибка;
- строка файла, содержащая ошибку;
- категория ошибки;
- источник обнаружения ошибки;

- состояния ошибки (с помощью директив препроцессора можно подавлять некоторые ошибки);
- средство обнаружения ошибки.

На рисунке 3.1 предоставлен пример окна «Список ошибок», содержащий различные по виду ошибки.

Список ошибок									
<div> <div>Все решение</div> <div> <div>6 Ошибки</div> <div>3 Предупреждения</div> <div>3 Сообщения</div> <div>Сборка и IntelliSense</div> </div> </div>									
Код	Описание	Проект	Файл	Строка	Категория	Источник	Состояние под	Средство	
CA1018	Укажите AttributeUsage в DbGetterDataAttribute.	Iris	DbGetterDataAttrib...	3	Design	IntelliSense	Активные	Microsoft.CodeAn...	
CS1002	Требуется ";".	UnitTests	CheckDbMethods.cs	3	Compiler	IntelliSense	Активные	Компилятор	
CS1002	Требуется ";".	UnitTests	CheckDbMethods.cs	13	Compiler	IntelliSense	Активные	Компилятор	
CS1003	Синтаксическая ошибка, требуется ",".	UnitTests	FormatLettersServic...	49	Compiler	IntelliSense	Активные	Компилятор	
CS1573	Параметр "userId" не имеет совпадающего тега param в комментарии XML для "ServerConnectionHelper.EnsureUserHaveConnections (Dictionary<int, List<ServerConnection>>, int)" (в отличие от остальных параметров)	Iris	ServerConnectionH...	17	Compiler	IntelliSense	Активные	Компилятор	
CS1591	Отсутствует комментарий XML для публично видимого типа или члена "FormatLettersService.FormatLetters (IEnumerable<LetterContract>, ResponseFormat)"	Iris	FormatLettersSevic...	41	Compiler	IntelliSense	Активные	Компилятор	
CS1591	Отсутствует комментарий XML для публично видимого типа или члена "DbGetterDataAttribute"	Iris	DbGetterDataAttrib...	3	Compiler	IntelliSense	Активные	Компилятор	
CS0006	Не удалось найти файл метаданных "D:\Study MIET\7 семестр\VCRIris\bin\Debug\net6.0\ref\Iris.dll".	UnitTests	CSC	1	Build	Сборка	Активные	Компилятор	
CS0246	Не удалось найти тип или имя пространства имен "RequiredAttribute" (возможно, отсутствует директива using или ссылка на сборку).	Iris	User.cs	16	Compiler	IntelliSense	Активные	Компилятор	
CS0246	Не удалось найти тип или имя пространства имен "Required" (возможно, отсутствует директива using или ссылка на сборку).	Iris	User.cs	16	Compiler	IntelliSense	Активные	Компилятор	
IDE1006	Нарушение правила именования: Эти слова должны начинаться с прописных символов: isAdmin	Iris	User.cs	27	Style	IntelliSense	Активные	Компилятор	
IDE0051	Закрытый элемент "AuthExceptionHandler" не используется.	Iris	AuthExceptionHandler.cs	11	CodeQuality	IntelliSense	Активные	Компилятор	

Рисунок 3.1 – Окно «Список ошибок»

Помимо, окна «Список ошибок», ошибки. источником обнаружения которых является IntelliSense. отображаются в коде программы в виде волнистого нижнего подчеркивания (красного цвета – ошибки, зеленого – предупреждения (рисунок 3.2); сообщения отображаются подчеркиванием в виде трех точек (рисунок 3.3)). При наведении курсора на проблемный фрагмент возникает всплывающее окно, описывающее проблему и предлагающее способы решения.

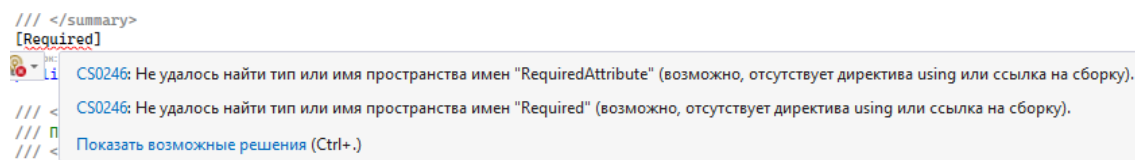


Рисунок 3.2 – Отображение ошибок и предупреждений в коде

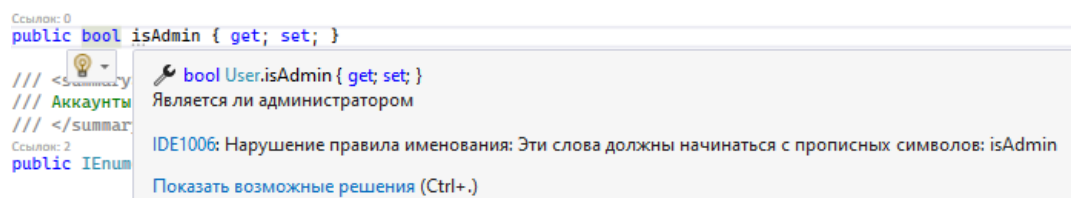


Рисунок 3.3 – Отображение сообщений в коде

Кроме отображения ошибок статический анализатор кода Visual Studio предоставляет инструмент очистки кода. Этот инструмент позволяет привести код к удобочитаемому виду, а также соблюдение соглашения code-style (стандарт оформления кода) для языка. Полный набор возможных исправлений доступных инструменту очистки кода представлен на рисунке 3.4.

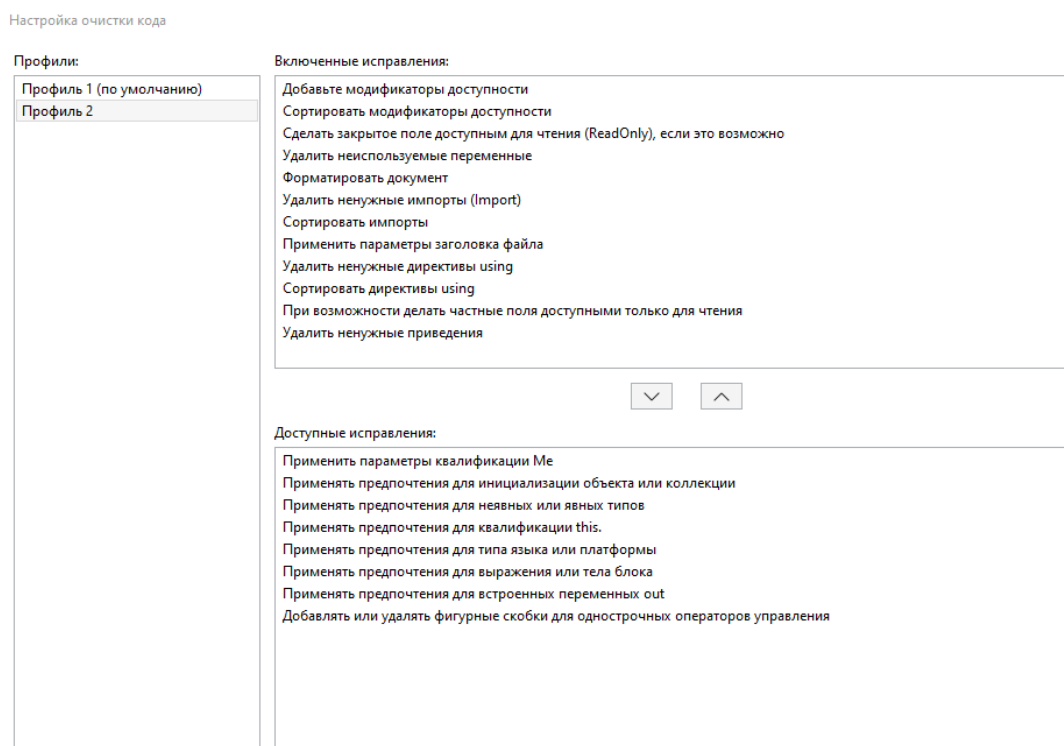


Рисунок 3.4 – Настройка очистки кода

Как видно из рисунка, можно создать несколько профилей очистки кода включающие в себя разные исправления.

### 3.1.2 Отладка с использованием логгера и встроенного отладчика

В ВС ДЭП логирование ошибок реализовано с помощью использования библиотеки Serilog. Логгер настроен так, что отображает информацию о типе сообщения ошибки

(ошибка, предупреждение, информационное сообщение), номер потока возникновения ошибки, дата и время возникновения, сообщение. Так, если во время работы ВС ДЭП возникает какая-либо ошибка, можно обратиться к текстовому файлу, куда сохраняются записи лога (новый файл создается раз в сутки, либо при достижении лимита по занимаемой памяти). Зная примерное время возникновения ошибки необходимо найти запись об этой ошибке – в сообщении об ошибке будет расписан стек вызова, по которому определяется место возникновения ошибки в коде. Для дальнейшей отладки ошибке используется встроенный отладчик Visual Studio.

Встроенный отладчик Visual Studio предоставляет инструмент «Точка останова», прерывающий выполнение программы для передачи управления отладчику. При выполнении программы до точки останова вызывает отладчик, для дальнейшей отладки кода.

Первая возможность отладчика – это пошаговое выполнение кода. Навигация по коду представляется тремя функциями: «шаг с заходом», «шаг с обходом» и «шаг с выходом» (рисунок 3.4).

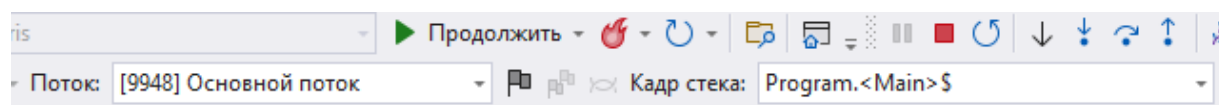


Рисунок 3.4 – Кнопки навигации по коду на панели инструментов Visual Studio («шаг с заходом» – стрелка вниз, «шаг с обходом» – полукруглая стрелка и «шаг с выходом» – стрелка вверх)

«Шаг с заходом» используется для остановки выполнения на каждом операторе, т.е. происходит пошаговое выполнение операторов кода. При вызове вложенных функций отладчик переходит до самого нижнего уровня вложенности.

Если же на определенном этапе отладки вложенной функции ясно, что она корректна и оставшиеся операторы необходимо пропустить, используется «шаг с выходом». В таком случае код вложенной функции будет выполнен, а затем выполнение остановится на первом операторе после вызова этой функции.

А если заход в функцию еще не произошел, но известно об ее корректности – необходимо использовать «шаг с обходом». Как и для «шага с выходом», код функции будет выполнен, а остановка произойдет на следующем после ее вызова операторе.

Вторая возможность отладчика – это просмотр стека вызова (рисунок 3.5) и просмотр значений переменных (рисунок 3.6).

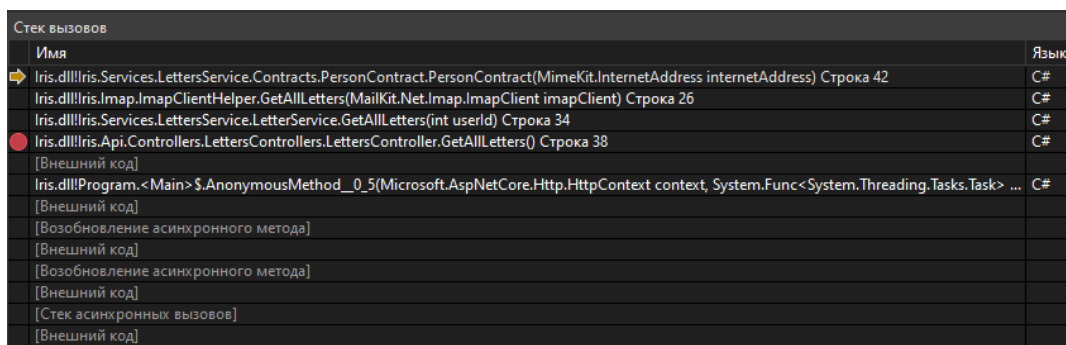


Рисунок 3.5 – Окно стека вызовов. Красная точка – место прерывания выполнения точкой останова, желтая стрелка – текущее место выполнения программы

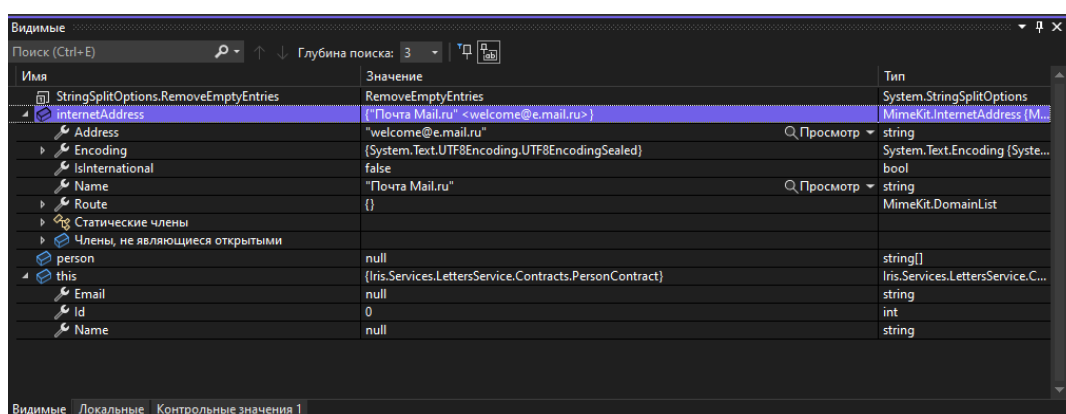


Рисунок 3.6. – Окно просмотра значений видимых переменных

Помимо просмотра видимых значений, во вкладке «Контрольные значения» можно зафиксировать переменные, за которыми необходимо наблюдать постоянно, либо же преобразовывать имеющиеся данные.

### 3.2 Средства тестирования программы

Тестирование программного обеспечения – это процесс поиска ошибок в исследуемом ПО, с целью проверки реального поведения программы с ожидаемыми от нее результатами.

По уровню тестирования выделяются четыре уровня тестирования:

- модульное – тестирование отдельных программных модулей (классов и методов);
- интеграционное – тестирование взаимодействия модулей друг с другом;
- системное – тестирование всей программы, готовой к релизу на различных платформах;

- приемочное – тестирование, проводимое заказчиком при приеме готового продукта.

По знанию внутренней реализации продукта тестирование делится на:

- тестирование черного ящика – известны только интерфейсы входных и выходных данных; однако таким подходом трудно проверить все граничные условия и специфичные ошибки;
- тестирование белого ящика – применяется в случае, если известна реализация тестируемого продукта; позволяет полностью покрыть все различные варианты поведения программы;

При тестировании ВС ДЭП применялись модульное тестирование белого ящика и интеграционное тестирование черного ящика.

### 3.2.1 Модульное тестирование

Модульные тесты – это уровень тестирования, на котором тестируется функциональность отдельных модулей – классов и их методов. Но помимо непосредственно проверки функциональности кода модульные тесты имеют также дополнительный смысл: грамотно написанные модульные тесты могут служить документацией к коду, предоставляя информацию о том, что должен делать модуль, какие может принимать входные данные и что может давать на выходе. Однако не любой код может быть эффективно покрыт модульными тестами – тестирование модулей-классов направленно на объектно-ориентированный код.

Выделяются следующие характеристики модульных тестов:

- быстрота выполнения отдельных тестов – большое количество быстровыполнимых тестов;
- изолированность – не зависят от внешних источников (таких как база данных или внешние сервисы);
- повторяемость – результаты выполнения тестов не должны изменяться при повторном запуске, если не был изменен тестируемый код;
- самопроверяемость – тест сам понимает пройден ли он успешно.

Поскольку модульное тестирование применяется для тестирования отдельных модулей возникает потребность в реализации «заглушек» имитирующих поведение других модулей, с которыми связан тестируемый модуль. Выделяют два типа реализации таких «заглушек»: моки и стабы.

Стаб – это класс-заглушка, который имитирует возвращаемые значения при вызове функции и не хранит в себе состояние. Мок – в отличие от стаба является полноценным объектом, который сохраняет состояние.

Для оптимизации выполнения тестов распространен паттерн модульных тестов AAA – arrange-act-assert. Предполагается, что каждый тест должен пройти стадии:

- arrange – инициализация данных необходимых для проведения теста;
- act – выполнение тестируемого метода;
- assert – проверка результатов;
- очистка памяти.

Применение этого паттерна позволяет сделать код тестов более читаемым за счет декомпозиции кода, что может существенно облегчить отладку для проваленных тестов.

### 3.2.2 Реализация модульного тестирования

Для реализации модульных тестов в ВС ДЭП используется фреймворк NUnit. Рассмотрим реализацию тестов с помощью этого фреймворка на примере тестирования класса MailServersService (листинг 3.1).

#### Листинг 3.1 – тесты класса MailServersService

```
using Iris.Api.Controllers.ConnectionsControllers;
using Iris.Database;
using Iris.Exceptions;
using Iris.Services.MailServersService;
using NUnit.Framework;
using System.Linq;
using UnitTests.Database;

namespace UnitTests.Tests.Services
{
    [TestFixture]
    public class MailServersServiceTests
    {
        private const string ExistAccountName = "AccName";
        private const string Host = "iris";
        private const int Port = 1;
```

```

private readonly DatabaseContext _dbContext = TestDatabase.Instance;
private int _userId;

private IMailServersService _mailServersService;

[SetUp]
public void SetUp()
{
    var user = _dbContext.Users.Add(new User
    {
        Name = "user"
    });
    _dbContext.SaveChanges();
    _userId = user.Entity.Id;

    var server = _dbContext.MailServers.Add(new MailServer
    {
        Host = Host,
        Port = Port,
    });
    _dbContext.SaveChanges();

    _dbContext.Accounts.Add(new Account
    {
        Name = ExsistAccountName,
        UserId = _userId,
        Password = "",
        MailServerId = server.Entity.Id
    });
    _dbContext.SaveChanges();

    _mailServersService = new MailServersService(_dbContext);
}

[TearDown]
public void TearDown()
{
    _dbContext.Accounts.RemoveRange(_dbContext.Accounts);
    _dbContext.MailServers.RemoveRange(_dbContext.MailServers);
}

```



```

        _dbContext.Users.RemoveRange(_dbContext.Users);

        _dbContext.SaveChanges();
    }

    [Test]
    public void GetMailServerAccounts_UserId_Accounts()
    {
        var servers = _mailServersService.GetMailServerAccounts(_userId);

        Assert.IsTrue(servers.Count() == 1);
    }

    [Test]
    public void GetAvailableMailServers_UserId_Accounts()
    {
        var servers = _mailServersService.GetAvailableMailServers(_userId);

        Assert.IsTrue(servers.Count() == 1);
    }

    [TestCase(Host + "1", Port + 1)]
    [TestCase(Host + "1", Port)]
    [TestCase(Host, Port + 1)]
    public void NewMailServer_NotExist_AddServer(string host, int port)
    {
        _mailServersService.NewMailServer(new MailServerContract
        {
            Host = host,
            Port = port
        });

        Assert.IsTrue(_dbContext.MailServers.Count() == 2);
    }

    [Test]
    public void NewMailServer_Exist_Exception()
    {

```

```

        Assert.Throws<ServerAlreadyExistException>(() => _mailServersService.NewMailServer(new MailServerContract
        {
            Host = Host,
            Port = Port
        }));
    }
}

```

Рассмотрим основные элементы класса тестов.

[TestFixture] – атрибут, обозначающий класс, содержащий модульные тесты.

[SetUp] – атрибут, обозначающий метод, выполняющийся перед каждым тестом.

[TearDown] – атрибут, обозначающий метод, выполняющийся после каждого теста.

[Test], [TestCase] – атрибуты, обозначающие методы тесты. Первый атрибут используется, если тест не имеет входных данных; второй – для передачи в тест различных входных данных.

Также рассмотрим фрагмент класса AuthServiceTests (листинг 3.2).

### Листинг 3.2 – Реализация стаба

```

private Mock<IUserService> _userService;
private IAuthService _authService;

[SetUp]
public void SetUp()
{
    _userService = new Mock<IUserService>();
    _userService.Setup(_ => _.GetUserByLogin(Username)).Returns(new
User { Name = Username, Password = Password, Token = Token });

    _config = new Config
    {
        AuthConfig = new AuthConfig
        {
            JwtSecurityKey = Encoding.ASCII.GetBytes("8u5j4WXfR74kDGE38k32zIBrLuDELjSTGzTx97OWwVY01-0uaayMdBlBWfZ55Fy8"),

```

```

        JwtLifetime = 24 * 3600
    }
};

_authService = new AuthService(_userService.Object, _config);
}

```

Стабы реализуются с помощью библиотеки Moq, реализацией класса Mock. Для реализации возвращаемых значений методами стаба в зависимости от входящих аргументов используется цепочка методов Mock.Setup() – ISetup.Returns().

Самопроверяемость тестов реализуется с помощью статического класса Assert, предоставляющего доступ к методам проверки результата, таким как:

- сравнение ожидаемого результат с полученным;
- проверка полученного результата, на 0, true, false null;
- проверка выбрасывания исключения тестируемым методом;
- методы принудительного успешного и неуспешного выполнения тестов.

Для поддержания документирующей функции модульных тестов в ВС ДЭП принято правило наименования и разделения тестов:

- один тест-класс соответствует одному тестируемому классу;
- правило именования тест-класса: [название тестируемого класса] + Tests;
- правило наименования методов-тестов: [название тестируемого метода]\_[характеристика входных данных]\_[ожидаемый результат].

### 3.2.3 Результаты модульного тестирования

Для модульного-тестирования ВС ДЭП методом белого ящика было разработано 98 тестов, со следующими сценариями:

- тестирование классов контроллеров:
  - взаимодействие с учетными записями;
  - работа с авторизацией;
  - получение писем;
  - взаимодействие с почтовыми серверами;
  - работа с регистрацией;
  - работа с обновлением статусов писем;

- тестирование конфигурационных классов;
- тестирование сервисов-зависимостей:
  - работа с учетными записями;
  - авторизация;
  - взаимодействие с классом ClaimsPrincipal;
  - взаимодействие м протоколами подключения;
  - форматирование писем;
  - тестирование работы Imap;
  - работа с почтовыми серверами;
  - взаимодействие с классом PersonContract;
  - тестирование работы Pop3;
  - тестирование работы регистрации;
  - тестирование работы с классом User;
- тестирование зависимостей-хранилищ:
  - тестирование хранилища запросов авторизации;
  - тестирование хранилища подключений;
  - тестирование хранилища токенов выданных пользователям.

Для анализа тестового покрытия кода ВС ДЭП воспользуемся расширение для Sisual Studio – JetBrains Dot Cover. Результаты анализа представлены на рисунке 3.7.

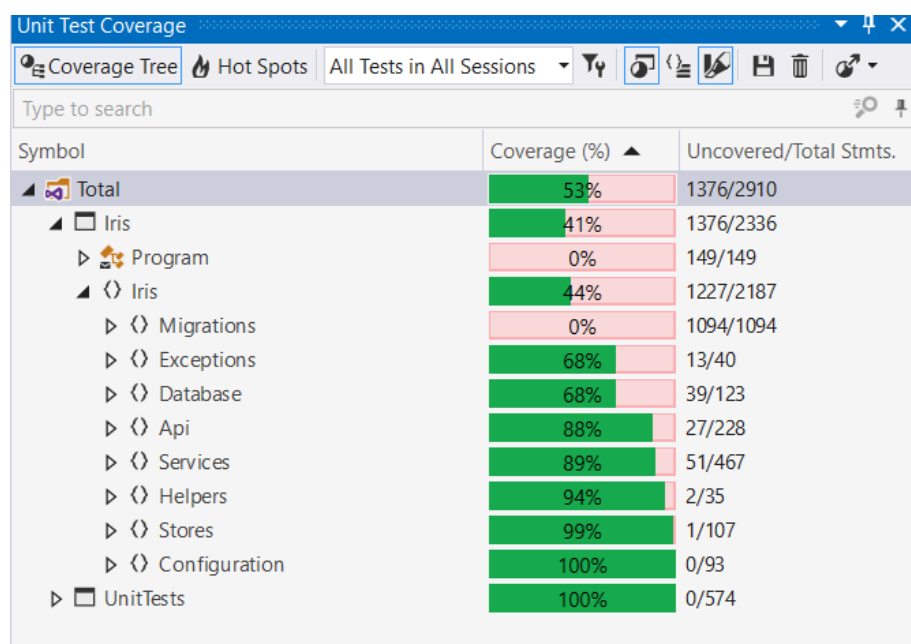


Рисунок 3.7 – Окно «Unit Test Coverage»

Результаты выполнения тестов представлены на рисунке 3.8.

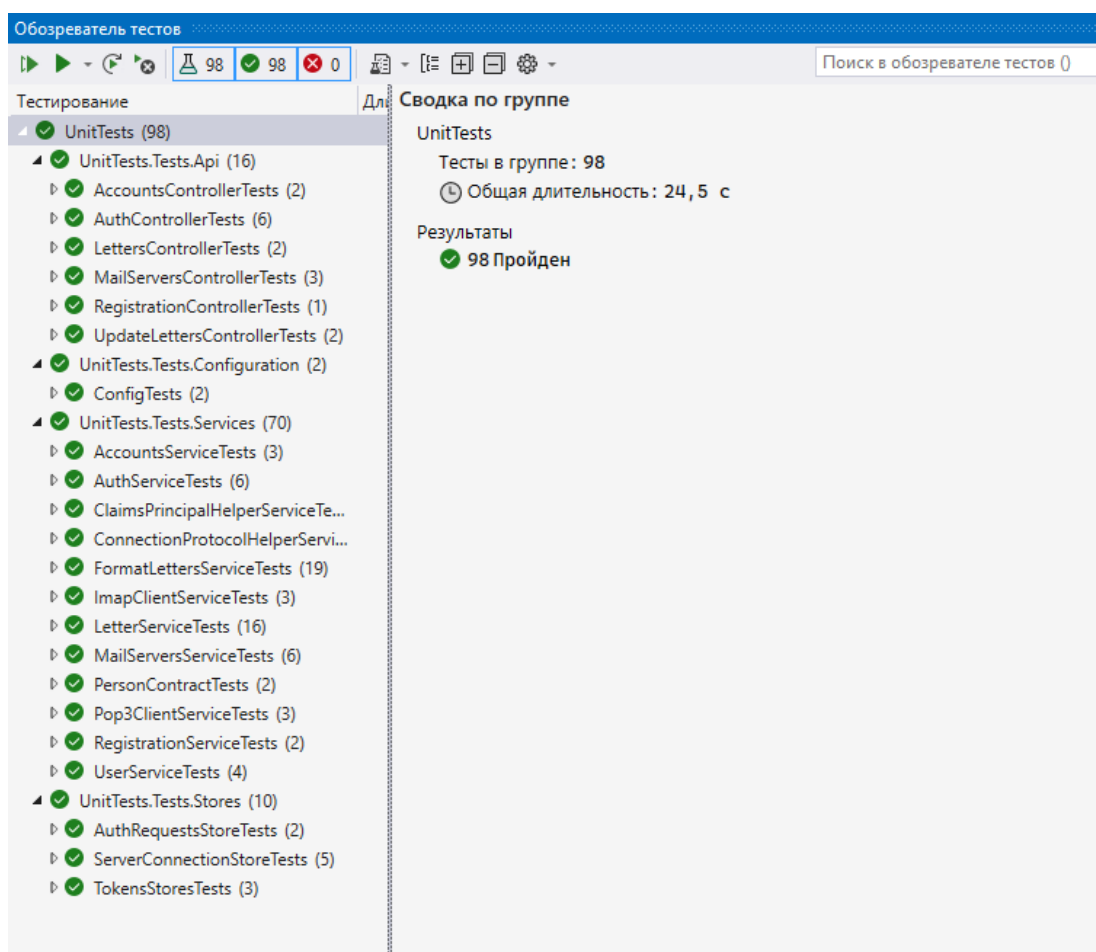


Рисунок 3.8. – Окно «Обозреватель тестов»

Все тесты успешно выполнены, общая длительность выполнения тестов составила 24.5 секунд.

Фактическое покрытие кода ВС ДЭП составило 53%, при 43% непосредственно для выполняемого решения «Iris». Однако, в силу особенности класса Program и набора классов Migrations, отвечающих за запуск веб-службы и применения миграций к локальной базе данных, они не тестируются. Поэтому, опуская строки этих классов, можно говорить о покрытии кода в решении «Iris» в 88%.

### 3.2.4 Интеграционное тестирование

Интеграционное тестирование применяется для тестирования взаимодействия между модулями программы и реализуется методом черного ящика.

В ВС ДЭП интеграционное тестирование заключается в тестирование методов контроллеров, т.е. тестирование API-методы веб-службы.

Для тестирования API-методов использована программа Postman. Postman – это API-платформа для создания и использования API. Он позволяет создавать, сохранять и выполнять http-запросы к API веб-службы, предоставляет результат выполнения запроса, позволяет его сохранить. Интерфейс программы Postman представлен на рисунке 3.9.

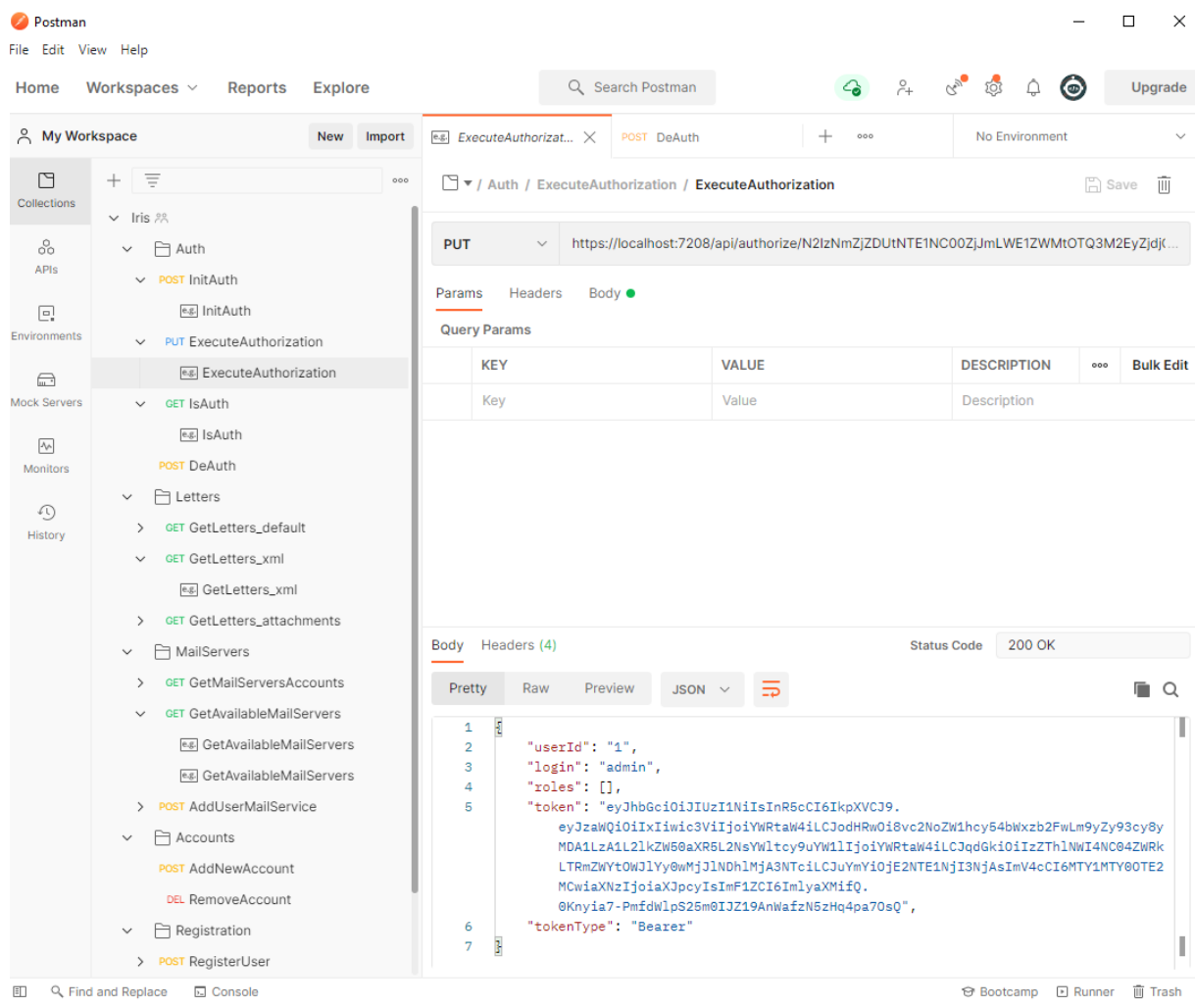


Рисунок 3.9 – Результат запроса ExecuteAuthorization

После создания и сохранения всех запросов их можно экспортировать в JSON-файл, служащий дополнительной документацией к проекту (рисунок 3.10).

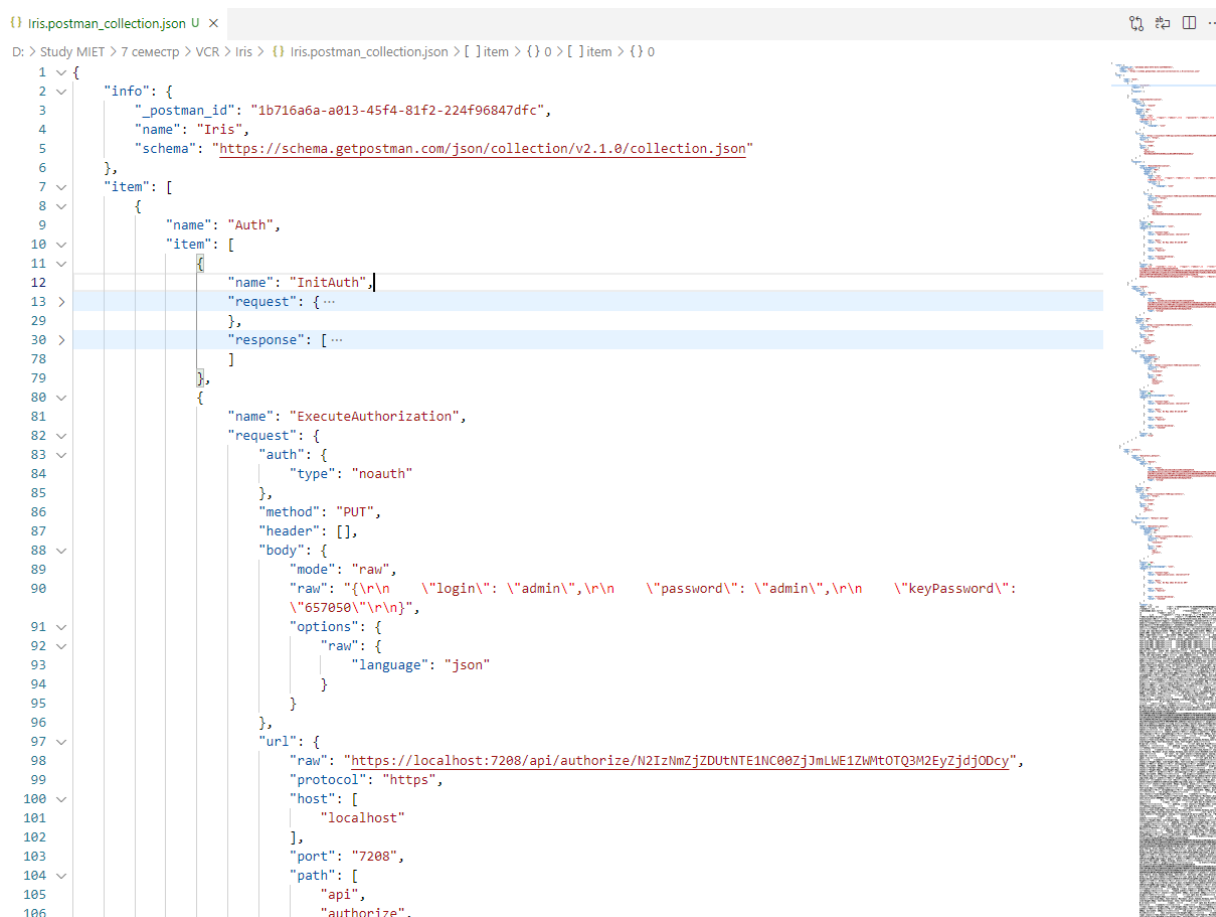


Рисунок 3.10 – JSON-файл экспортированных интеграционных тестов

При экспорте сохраняется вся информация о названиях запросов, их описании, данных запроса, данных полученного ответа. При необходимости повторного проведения интеграционных тестов импортируется обратно в Postman.

Для ВС ДЭП были разработаны и проведены тесты для всех API-методов, а именно:

- контроллер работы с учетными записями;
  - добавить новую учетную запись;
  - удалить учетную запись;
- контроллер авторизации;
  - инициализировать авторизацию;
  - выполнить авторизацию;
  - де-авторизация;
  - проверка авторизации;
- контроллер работы с почтовыми серверами;
  - получить список аккаунтов к почтовым серверам пользователя;

- получить список доступных почтовых серверов;
  - добавить новый почтовый сервер;
- контроллер получения писем;
  - получить письма по запросу;
  - получить письма по запросу;
- контроллер изменения писем;
  - установить флаг;
  - удалить письмо;
- контроллер регистрации пользователей;
  - зарегистрировать пользователя.

Все тесты были успешно выполнены и сохранены в JSON-файл.

#### Выводы по испытательному разделу

В испытательном разделе были проведены отладка и тестирование разработанной ВС ДЭП. Для этого были рассмотрены средства отладки программ и произведено выполнение выбранных методов, описаны тест-кейсы для модульного и интеграционного тестирования методами черного и белого ящиков.

Отладка производилась с использованием статического анализатора кода, с использованием логгера и встроенного отладчика.

Тестирование состояло из модульного тестирования, проведенного методом белого ящика, и интеграционного тестирования, проведенного методом черного ящика.



## ЗАКЛЮЧЕНИЕ

Результатом выпускной квалификационной работы является рабочая версия веб-службы для доступа к электронной почте на основе двухфакторной аутентификации. ВС ДЭП позволяет использовать предоставляемый функционал взаимодействия с почтовыми серверами.

Повышение скорости внедрения этого функционала в другие программы положительно сказывается на процессе разработки. Поэтому ВС ДЭП является актуальным и практически значимым для компаний, связанных с разработкой, затрагивающей взаимодействие с электронной почтой.

В рамках выпускной квалификационной работы решены следующие задачи:

- исследована предметная область и актуализирована изучаемая проблема;
- произведен обзор существующих аналогов и проведен их сравнительный анализ;
- описана концептуальная модель предметной области
- описаны входные и выходные данные, форматы их внутреннего представления в программе;
- проведен сравнительный анализ, выбор языка программирования и среды разработки;
- разработан и описан алгоритм и методы решения поставленных задач;
- выполнена программная реализация ВС ДЭП, разработан пользовательского интерфейса;
- произведены отладка и тестирование ВС ДЭП;
- разработано руководство программиста ВС ДЭП.

## СПИСОК ЛИТЕРАТУРЫ

1. Доронина А.А., Касимов Р.А., Федотова Е.Л. Методические указания по подготовке выпускной квалификационной работы по направлению подготовки бакалавров 09.03.04 «Программная инженерия» / под ред. Л.Г. Гагариной. М.: МИЭТ, 2021. 28 с.
2. Гост 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения
3. ГОСТ Р 7.0.5-2008. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая ссылка. Общие требования и правила составления.
4. ГОСТ 7.32-2017. Отчет о научно-исследовательской работе. Структура и правила оформления.
5. Илюшечкин В.М., Илюшечкина Л.В. Учебно-методические разработки для самостоятельной работы студентов, изучающих дисциплину «Базы данных». - М.: МИЭТ, 2007. - 88 с.: ил.
6. А. А. Афанасьев, Л. Т. Веденьев, А. А. Воронцов. Аутентификация. Теория и практика обеспечения безопасного доступа к информационным ресурсам. Учебное пособие для вузов. 2012. – 550 с.
7. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. П75 Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб: Питер, 2001. — 368 с.: ил. (Серия «Библиотека программиста»)
8. Буч, Гради, Максимчук, Роберт А., Энгл, Майкл У, Янг, Бобби Дж.,Коналлен, Джим, Хьюстон, Келли А. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд.: Пер. с англ. -М.: 000 "И.Д. Вильяме", 2008. -720 с.:ил. —Парал. тит. англ.
9. Язык программирования C# 7 и платформы .NET и .NET Core, 8-е изд.: Пер. с англ. — СПб. : ООО “Диалектика”, 2018 — 1328 с. : ил. — Парал. тит. англ.
10. Simple Object Access Protocol (SOAP) 1.1 [Электронный ресурс]. URL: <https://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
11. NCSC-TG-017 [Электронный ресурс]. URL: [https://csirt.org/color\\_%20books/NCSC-TG-017.pdf](https://csirt.org/color_%20books/NCSC-TG-017.pdf)
12. RFC 6238 - TOTP: Time-Based One-Time Password Algorithm [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc6238>

13. RFC 4226 - HOTP: An HMAC-Based One-Time Password Algorithm [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc6238>
14. RFC 3174 - US Secure Hash Algorithm 1 (SHA1) [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc6238>
15. IMAP, SMTP и POP3 Mail.ru – Помощь Mail.ru Почта [Электронный ресурс]. URL: <https://help.mail.ru/mail/mailer/popsmtpt>
16. Пароли для внешних приложений – Помощь Mail.ru Почта [Электронный ресурс]. URL: <https://help.mail.ru/mail/security/protection/external>
17. Другие программы – Почта. Справка [Электронный ресурс]. URL: <https://yandex.ru/support/mail/mail-clients/others.html>
18. Пароли приложений – Яндекс ID. Справка [Электронный ресурс]. URL: <https://yandex.ru/support/id/authorization/app-passwords.html>
19. Как настроить доступ к Gmail в сторонних почтовых клиентах – Справка – Gmail [Электронный ресурс]. URL: <https://support.google.com/mail/answer/7126229?hl=ru>
20. Как использовать почтовый POP-клиент для работы с письмами Gmail клиентах – Справка – Gmail [Электронный ресурс]. URL: <https://support.google.com/mail/answer/7104828>
21. Настройка POP, IMAP и SMTP для Outlook.com [Электронный ресурс]. URL: <https://support.microsoft.com/ru-ru/office/настройка-pop-imap-и-smtp-для-outlook-com-d088b986-291d-42b8-9564-9c414e2aa040>
22. Центр разработчиков Microsoft Graph [Электронный ресурс]. URL: <https://developer.microsoft.com/ru-ru/graph>
23. Gmail API | Google Developers [Электронный ресурс]. URL: <https://developers.google.com/gmail/api>
24. API – документация biz.mail.ru 1.0 [Электронный ресурс]. URL: <https://biz.mail.ru/developer/api.html>
25. Serilog — simple .NET logging with fully-structured events [Электронный ресурс]. URL: <https://serilog.net/>
26. API Documentation & Design Tools for Teams | Swagger [Электронный ресурс]. URL: <https://swagger.io/>

27. Java Documentation - Get Started [Электронный ресурс]. URL: <https://docs.oracle.com/en/java/>
28. Документация по C#. Начало работы, руководства, справочные материалы. | Microsoft Docs [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>
29. Welcome to Python.org [Электронный ресурс]. URL: <https://www.python.org/>
30. Современный учебник JavaScript [Электронный ресурс]. URL: <https://learn.javascript.ru/>
31. PHP: Hypertext Preprocessor [Электронный ресурс]. URL: <https://www.php.net/>
32. Язык программирования Ruby [Электронный ресурс]. URL: <https://www.ruby-lang.org/ru/>
33. Visual Studio: IDE и редактор кода для разработчиков и групп, работающих с программным обеспечением [Электронный ресурс]. URL: <https://visualstudio.microsoft.com/ru/>
34. Rider: кросс-платформенная IDE для .NET [Электронный ресурс]. URL: <https://www.jetbrains.com/ru-ru/rider/>
35. MonoDevelop | MonoDevelop [Электронный ресурс]. URL: <https://www.monodevelop.com/>
36. SharpDevelop [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/SharpDevelop>
37. eclipse/aCute: Eclipse aCute - C# edition in Eclipse IDE [Электронный ресурс]. URL: <https://github.com/eclipse/aCute>
38. SQLite Home Page [Электронный ресурс]. URL: <https://www.sqlite.org/index.html>
39. Отладка в Visual Studio (Windows) [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/visualstudio/debugger/?view=vs-2022>
40. Рекомендации по написанию модульных тестов - .NET [Электронный ресурс]. URL: <https://docs.microsoft.com/ru-ru/dotnet/core/testing/unit-testing-best-practices>
41. Тестирование [Электронный ресурс]. URL: <https://pvs-studio.com/ru/blog/terms/0093/>