Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
НА ТЕМУ:
Веб-приложение для обмена данными через локальный почтовый сервис

Студент ИУ7-74Б _____ Керимов А. Ш.
(Группа) (Подпись, дата) (Фамилия И. О.)

Студент ИУ7-74Б _____ Новиков М. Р.
(Группа) (Подпись, дата) (Фамилия И. О.)

Руководитель курсовой работы _____ Рогозин Н. О.
(Подпись, дата) (Фамилия И. О.)

2020 г.

Содержание

Введение

Технология электронной почты появилась в 1965 году — сотрудники Массачусетского технологического института Ноэль Моррис и Том Ван Влек написали программу mail для операционной системы CTSS. Однако коммерческое использование электронной почты началось только в 1990-х с запуском сервиса Hotmail.

Общепринятым в мире протоколом обмена электронной почтой является протокол SMTP, который использует DNS для определения правил пересылки почты. Стандарт протокола был впервые описан в 1982 году (RFC 821), а затем дополнен в 2008 году (RFC 5321). Почтовые серверы используют SMTP для отправки и получения почтовых сообщений. Работающие на уровне пользователя клиентские почтовые приложения обычно используют SMTP только для отправки писем на почтовый сервер, а для получения сообщений применяют другие протоколы (POP, IMAP).

Курсовая работа посвящена разработке веб-приложения для обмена данными через локальный почтовый сервис и библиотек, реализующих SMTP-сервер и SMTP-клиент.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с техническим заданием на курсовую работу, необходимо разработать веб-приложения для обмена данными через локальный почтовый сервис.

Для решения поставленной задачи необходимо:

1. Провести анализ протокола SMTP.
2. Разработать алгоритмы, реализующие протокол SMTP.
3. Разработать библиотеку, реализующую SMTP-сервер.
4. Разработать библиотеку, реализующую SMTP-клиент.
5. Разработать веб-приложение для обмена данными через почтовый сервис.

1.2 Требования к разрабатываемому ПО

В соответствии с техническим заданием на курсовую работу, были установлены следующие требования к разрабатываемому программному обеспечению:

1. SMTP-сервер должен быть запущен локально.
2. Для получения сообщения сервер должен использовать разработанную библиотеку.
3. Для отправки сообщения веб-приложение должно использовать разработанную библиотеку.

4. Веб-приложение должно, помимо отправки сообщения, предусматривать авторизацию и регистрацию пользователей, просмотр входящих и исходящих сообщений, удаление сообщений.

1.3 Анализ протокола SMTP

SMTP (*англ.* **S**imple **M**ail **T**ransfer **P**rotocol — простой протокол передачи почты) — требующий соединения текстовый протокол, по которому отправитель сообщения связывается с получателем посредством выдачи командных строк и получения необходимых данных через надёжный канал, в роли которого обычно выступает TCP-соединение (по умолчанию — 25 порт). SMTP-сессия состоит из команд, посылаемых SMTP-клиентом, и соответствующих ответов SMTP-сервера. Когда сессия открыта, сервер и клиент обмениваются её параметрами. Сессия может включать ноль и более SMTP-операций (транзакций).

SMTP-операция состоит из трёх последовательностей команда/ответ (см. пример ниже). Описание последовательностей:

- `MAIL FROM` — устанавливает обратный адрес (то есть Return-Path, 5321.From, mfrom). Это адрес для возвращённых писем.
- `RCPT TO` — устанавливает получателя данного сообщения. Эта команда может быть дана несколько раз, по одной на каждого получателя. Эти адреса также являются частью оболочки.
- `DATA` — для отправки текста сообщения. Это само содержимое письма, в противоположность его оболочке. Он состоит из заголовка сообщения и тела сообщения, разделённых пустой строкой. `DATA`, по сути, является группой команд, а сервер отвечает дважды: первый раз на саму команду `DATA`, для уведомления о готовности принять текст; и второй раз после конца последовательности данных, чтобы принять или отклонить всё письмо.

Помимо промежуточных ответов для DATA-команды, каждый ответ сервера может быть положительным (код ответа 2xx) или отрицательным. Последний, в свою очередь, может быть постоянным (код 5xx) либо

временным (код 4xx). Отказ SMTP-сервера в передаче сообщения — постоянная ошибка; в этом случае клиент должен отправить возвращённое письмо. После сброса — положительного ответа, сообщение скорее всего будет отвержено. Также сервер может сообщить о том, что ожидаются дополнительные данные от клиента (код 3xx).

Согласно секции 4.5.1 RFC 5321 помимо трёх перечисленных, любой SMTP сервер должен поддерживать команды:

- `EHLO`, `HELO` — приветствие, начало сессии. Предпочтительнее использовать `EHLO`.
- `RSET`. Эта команда указывает, что текущая почтовая транзакция будет прервана.
- `NOOP`. Эта команда не влияет на параметры или ранее введенные команды. Он не определяет никаких действий, кроме отправки получателем ответа `250 OK`.
- `QUIT`. Эта команда указывает, что получатель ДОЛЖЕН отправить ответ `221 OK`, а затем закрыть канал передачи.
- `VRFY`. Эта команда просит получателя подтвердить, что аргумент идентифицирует пользователя или почтовый ящик.

1.4 Выводы

В результате анализа технического задания на курсовую работу была поставлена задача, были определены основные требования к разрабатываемому ПО.

В результате анализа протокола SMTP были определены принципы его работы, что позволяет перейти к проектированию библиотек, реализующих SMTP-сервер и SMTP-клиент.

2 Конструкторский раздел

2.1 Проектирование библиотек

2.1.1 Библиотека, реализующая SMTP-сервер

2.1.2 Библиотека, реализующая SMTP-клиент

2.2 Проектирование веб-приложения

2.2.1 База данных

Для хранения сообщений необходимо разработать базу данных.

Каждый пользователь веб-приложения описывается именем, электронным адресом и паролем.

Каждое сообщение описывается электронным адресом отправителя, темой письма, временем отправки/получения письма и содержанием письма.

Кроме того, необходима дополнительная сущность, описываемая идентификатором письма и электронным адресом получателя, так как одно письмо может быть отправлено нескольким адресатам.

На рисунке 2.1 представлена диаграмма базы данных.

2.2.2 SMTP-сервер

2.2.3 Архитектура приложения

Для проектирования приложения применим архитектурый шаблон проектирования Model-View-Controller (далее — MVC).

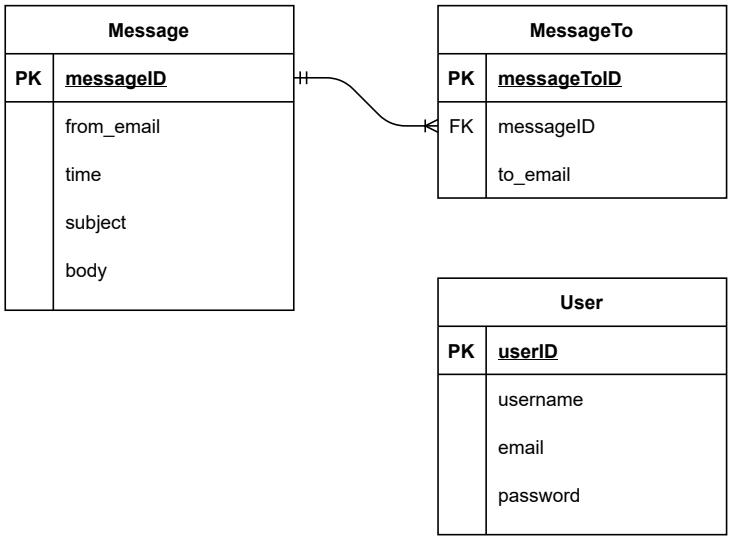

Рис. 2.1: Диаграмма базы данных

Архитектурный шаблон MVC

MVC представляет из себя схему разделения данных и бизнес-логики приложения, пользовательского интерфейса и управляющей логики на три независимых компонента: модель, представление и контроллер. Такой подход позволяет изолировать данные и управляющую логику, независимо разрабатывать, тестировать, поддерживать и модифицировать компоненты. Схема шаблона MVC представлена на рисунке 2.2.

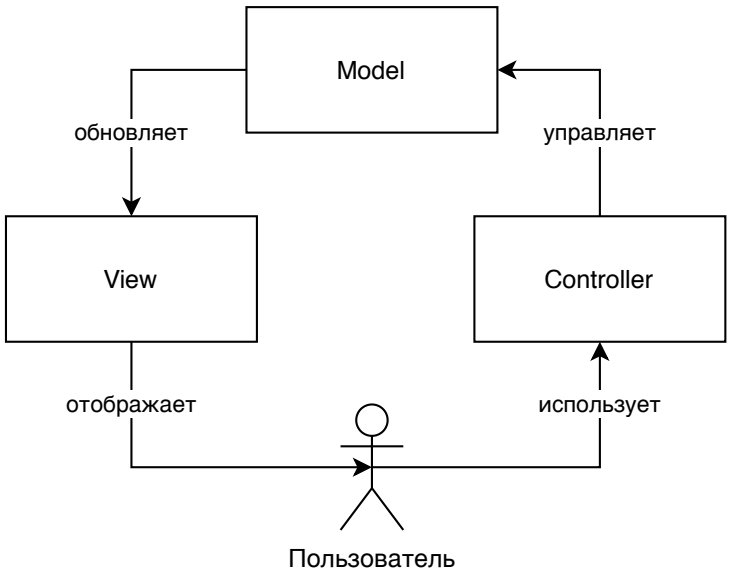

Рис. 2.2: Диаграмма шаблона MVC

Модель представляет собой данные и методы для работы с данными. В модели выполняются запросы к базе данных, бизнес-логика. Этот компонент разрабатывается таким образом, чтобы отвечать на запро-

сы контроллера, изменять свое внутреннее состояние и не зависеть от представлений.

Представление получает данные модели и отображает их пользователю. Представление не обрабатывает данные.

Контроллер является связующим компонентом — интерпретирует действия пользователя, оповещая модель об изменениях, которые необходимо внести.

UML-диаграмма компонентов приложения

На рисунке 2.3 представлена UML-диаграмма компонентов веб-приложения.

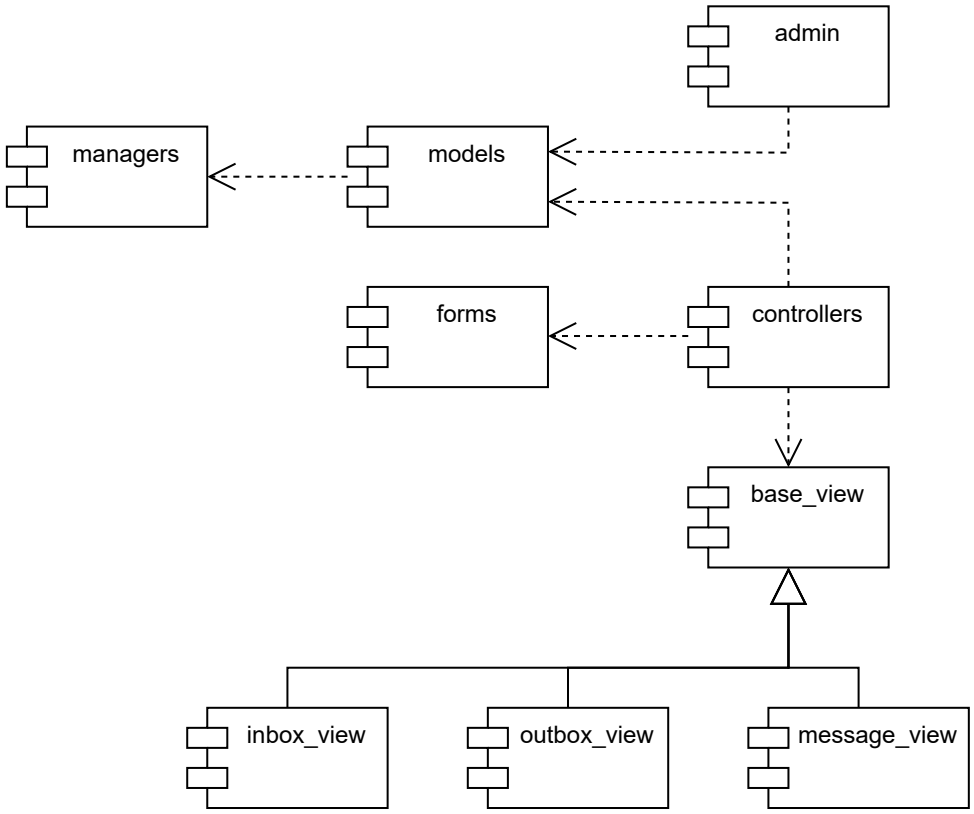

Рис. 2.3: UML-диаграмма компонентов приложения

2.3 Выводы

В конструкторском разделе были спроектированы библиотеки, реализующие протокол SMTP. Были спроектированы база данных и ар-

хитектура приложения, построена UML-диаграмма компонентов веб-приложения.

3 Технологический раздел

3.1 Выбор языка программирования и среды разработки

В качестве языка программирования был выбран язык Python.

В качестве фреймворка для разработки веб-приложения был выбран фреймворк Django. Django является одним из наиболее популярных фреймворков для разработки веб-приложений на Python. В качестве СУБД был выбран SQLite, который используется в Django по умолчанию.

В качестве среды разработки была выбрана IDE PyCharm. PyCharm содержит редактор кода, отладчик, средства для статического анализа кода, средства для сборки проекта, тесную интеграцию с фреймворком Django.

3.2 Реализация библиотек

3.2.1 Библиотека, реализующая SMTP-сервер

3.2.2 Библиотека, реализующая SMTP-клиент

3.3 Реализация веб-приложения

3.3.1 Реализация SMTP-сервера

3.3.2 Реализация приложения

Реализация шаблона проектирования

В основе Django лежит шаблон проектирования MVC, который во фреймворке называется Model-View-Template, где Model — модель, являющаяся фактически ORM-сущностью, View — контроллер, Template — представление. Бизнес-логику в Django принято выделять в отдельный компонент.

Компоненты приложения

Выделим три компонента: компонент доступа к данным, компонент графического интерфейса пользователя, компонент, связывающий данные и графический интерфейс пользователя.

Компонент доступа к данным представляет из себя классы `django.db.models.Models` (далее — модели), данные в которых соответствуют атрибутам таблиц в базе данных. Модели содержат методы для обработки данных на уровне строки, например метод `get_to_emails()`, возвращающий список адресатов сообщения. На листинге 3.1 представлена модель `Message`, соответствующая таблице Message в базе данных.

Листинг 3.1: Модель `Message`

```
class Message(models.Model):
    from_email = models.CharField(max_length=254)
    time = models.DateTimeField()
    subject = models.TextField()
```

```
    body = models.TextField()

    objects = MessageManager()

    def get_to_emails(self):
        to_emails = []
        for recipient in self.recipient.all():
            to_emails.append(recipient.to_email)
        return to_emails
```

Для работы с данными на уровне таблицы используются классы, называемые менеджерами. Эти классы наследуются от `django.db.models.Managers` и содержат методы для доступа к данным. На листинге 3.2 приведен пример менеджера.

Листинг 3.2: Менеджер `MessageManager`

```
class MessageManager(models.Manager):
    def get_inbox(self, email):
        return self.filter(recipient__to_email=email).order_by('-time')

    def get_outbox(self, email):
        return self.filter(from_email=email).order_by('-time')
```

Классы-контроллеры в Django наследуются от класса `django.views.generic.DefaultView`. При этом существуют классы для выполнения типичных задач представления данных: для отображения списка объектов — `ListView`, для отображения информации о конкретном объекте — `DetailView` и др. На листинге 3.3 представлен класс `MessageDetailView`.

Листинг 3.3: Контроллер `MessageDetailView`

```
@method_decorator(login_required, name='dispatch')
class MessageDetailView(DetailView):
    model = Message
    template_name = 'webmail/message.html'
    context_object_name = 'message'

    def get_object(self, *args, **kwargs):
        entity = super().get_object(*args, **kwargs)
        if self.request.user.email not in entity.get_to_emails() +
           [entity.from_email]:
            raise Http404
        return entity
```

Компонент интерфейса представляет из себя набор HTML-страниц,

использующих шаблонизатор Django, который позволяет использовать шаблоны для генерации конечных страниц. Шаблонизатор позволяет, в частности, переиспользовать код и ускоряет верстку веб-приложения. Данные передаются из контроллера в качестве параметров на страницу.

3.3.3 Интерфейс

Интерфейс приложения представляет из себя страницу, в шапке которой расположено название приложение, навигационное меню, позволяющее перейти к спискам входящих (Inbox) и исходящих сообщений (Outbox), информацию об авторизованном пользователе.

На странице Inbox расположена таблица, содержащая входящие письма. Каждое письмо представлено адресантом, темой письма и содержимым письма, которое при переполнении первой строки скрывается. Для каждого письма предусмотрена кнопка удаления. Структура страницы Outbox аналогична. При нажатии на письмо происходит переход на страницу Message, на которой отображена более подробная информация о письме: адресат и список адресантов, время получения письма, тема письма, полное содержимое письма. На каждой странице приложения справа от заголовка страницы расположена кнопка Compose, которая открывает модальное окно, позволяющее написать письмо.

На рисунках 3.1–3.4 представлен интерфейс веб-приложения.

3.4 Тестирование и отладка

Для программы использовался отладчик и статический анализатор кода, встроенные в среду разработки PyCharm.

Библиотеки, SMTP-сервер и веб-приложение были протестированы в полном объеме, все обнаруженные ошибки были исправлены.

3.5 Выводы

В результате разработки было реализовано программное обеспечение в полном соответствии с техническим заданием и предъявляемыми

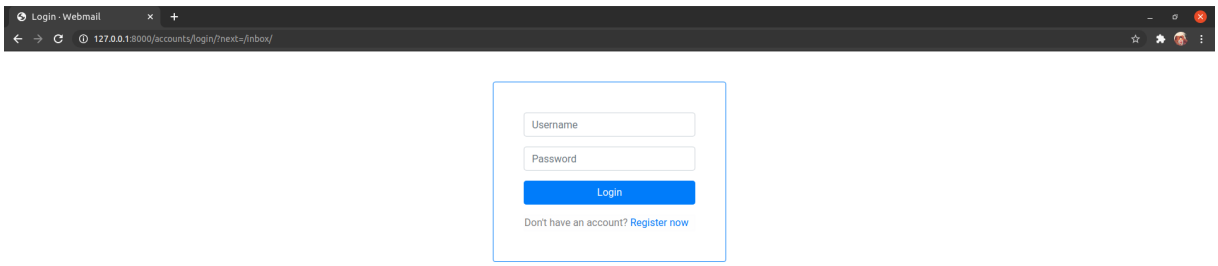

Рис. 3.1: Авторизация в веб-приложении

требованиями.

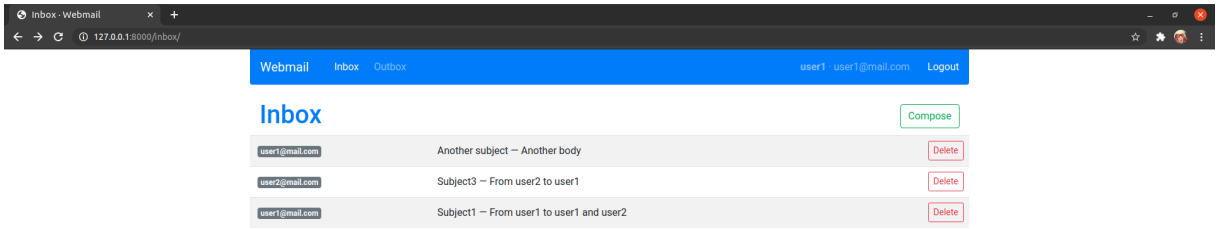

Рис. 3.2: Вкладка входящих сообщений

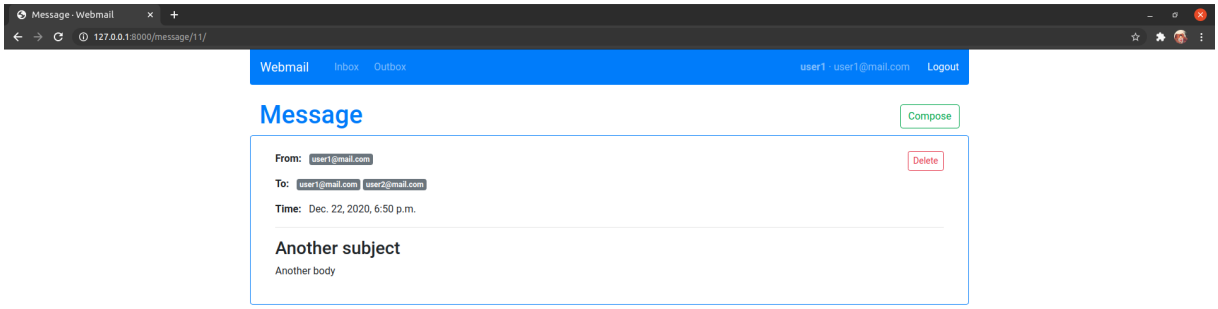

Рис. 3.3: Просмотр сообщения

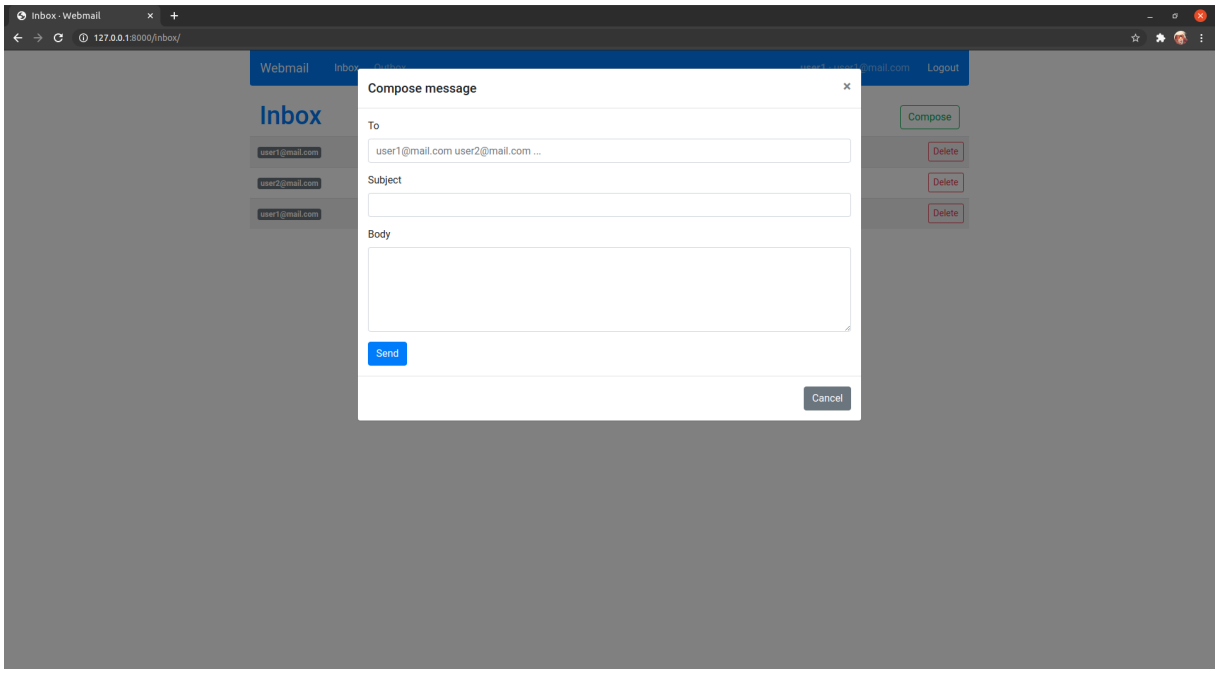

Рис. 3.4: Создание сообщения

Заключение

В результате выполнения курсовой работы были разработаны веб-приложение для обмена данными через локальный почтовый сервис и библиотеки, реализующие протокол SMTP.

В процессе выполнения курсовой работы были выполнены все поставленные задачи, а именно: проанализирован протокол SMTP; спроектированы и разработаны библиотеки, реализующие протокол SMTP, SMTP-сервер; разработано веб-приложение для обмена данными через почтовый сервис. Программное обеспечение было протестировано в полном объеме.