МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ФГБОУ ВО («ВГУ»)

Факультет математический

Клиент-серверное приложение для управления персоналом и проектами

Выпускная квалификационная работа

«Системный инженер (специалист по эксплуатации аппаратно-программных комплексов персональных ЭВМ и сетей на их основе)»

Обучающийся А.А. Уткин

Руководитель д.ф.-м.н. Д.В. Груздев

Воронеж 2019

Содержание

Введение **3**

1 Постановка задачи **4**

2 Используемые технологии **5**

3 Этапы создания сервера **9**

3.1 Создание и настройка сервера 9

3.2 Настройка работы HTTPS 10

3.3 Интеграция технологии JSON Web Token (JWT) 12

3.4 Разработка собственного API 13

3.5 Реализация серверной части 15

4 Этапы создания клиента **18**

4.1 Настройка системы для разработки клиента 18

4.2 Создание интерфейса 19

4.3 Реализация клиентской части 20

4.4 Логика работы клиента 22

5 Заключение **27**

Введение

Практически каждая команда разработчиков при создании программного продукта использует различные решения для контроля выполненных задач, такими, как, например Gitlab issues или Redmine. Без использования таких инструментов отслеживание вклада каждого человека в работу над продуктом становиться затруднительной задачей.

Исходя из важности использования таких решений, целью работы была поставлена задача разработки клиент-серверного программного обеспечения с подобным функционалом, с добавлением в него элементов управления персоналом (работниками) и проектами (задачами), используя собственный опыт разработки кроссплатформенного программного обеспечения с элементами интерфейса.

1 Постановка задачи

В процессе проектирования архитектуры приложения-клиента было решено реализовать следующий функционал:

1. Авторизация работника, смена пароля работником;
2. Добавление и увольнение работников;
3. Создание и завершение проектов;
4. Назначение проектов работнику;
5. Редактирование данных работников и проектов;
6. Отображение назначенных работнику проектов;
7. Отображения списка работников, назначенных на конкретный проект.

Серверная часть должна была выполнять запросы клиента, оперировать данными в БД, а также отвечать за аутентификацию пользователей в системе. Исходя из этого серверная часть должна была иметь следующий функционал:

1. Обработка запросов, поступающих от клиента;
2. Поддержка защищенного соединения;
3. Возможность одновременного ответа на несколько запросов;
4. Поддержка кодов состояния HTTP;
5. Контроль доступа к данным БД.

2 Используемые технологии

Разработка клиент-серверных приложений влечет за собой необходимость в использовании множества технологий. Необходим выбор протокола передачи данных, формата самих данных, способа их защиты, а также вида базы данных для их хранения. Также неотъемлемой частью работы является выбор языка программирования для реализации задуманного функционала.

Определение 1 *«Клиент — сервер» — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов.*

Определение 2 *Протокол передачи данных — набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок.*

Для поддержки защищенного соединения между клиентом и сервером было принято решение использовать протокол HTTPS.

Определение 3 *HTTPS — расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. Данные в протоколе HTTPS передаются поверх криптографических протоколов SSL или TLS.*

Определение 4 *HTTP — протокол прикладного уровня передачи данных изначально — в виде гипертекстовых документов в формате «HTML».*

Для упрощения процесса взаимодействия клиента и сервера был разработан собственный API, с помощью которого происходит составление запросов и структурирование отправляемых данных.

Определение 5 *API — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.*

Данные между клиентом и сервером было решено передавать с помощью POST-запросов в формате JSON.

Определение 6 *POST — один из многих методов запроса, поддерживаемых HTTP протоколом, используемым во Всемирной паутине. Метод запроса POST предназначен для запроса, при котором веб-сервер принимает данные, заключённые в тело сообщения, для хранения. Он часто используется для загрузки файла или представления заполненной веб-формы.*

Определение 7 *JSON — текстовый формат обмена данными, основанный на JavaScript, как и многие другие текстовые форматы, он легко читается людьми.*

Для безопасной передачи данных аутентификации на сервер используется технология JSON Web Token (JWT).

Определение 8 *JSON Web Token (JWT) — это открытый стандарт (RFC 7519) для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.*

Исходя из личного опыта разработки, в качестве основного языка программирования был выбран Python версии 3.7.

Определение 9 *Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций. Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.*

Для создания интерфейса клиента было решено использовать PyQt5 в связке с конструктором интерфейса Qt designer. Такой выбор обусловлен кроссплатформенностью этого инструмента, большой базой компонентов и возможностью создания собственных, а также подробной документацией Qt5.

Определение 10 *Qt — кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.*

Определение 11 *PyQt — набор «привязок» графического фреймворка Qt для языка программирования Python, выполненный в виде расширения Python.*

Определение 12 *Qt Designer — кроссплатформенная свободная среда для разработки графических интерфейсов (GUI) программ, использующих библиотеку Qt.*

Для хранения и обработки данных использован виртуальный сервер (VPS) со следующим окружением:

1. Сервер базы данных MongoDB;
2. Nginx для работы API;
3. Gunicorn в качестве WSGI сервера;
4. Серверная часть существует в виде Docker контейнера.

При выборе системы управления базами данных выбор пал на MongoDB. Этот выбор обусловлен открытым исходным кодом продукта, а также желанием получить опыт использования NoSQL СУБД в проекте.

Определение 13 *MongoDB — документоориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных.*

В качестве веб сервера был выбран Nginx. Для взаимодействия веб-сервера и кода на языке программирования Python был использован WSGI-сервер Gunicorn.

Определение 14 *Nginx — веб-сервер и почтовый прокси-сервер, работающий на UNIX-подобных операционных системах.*

Определение 15 *Gunicorn (Green Unicorn) — WSGI HTTP сервер, написанный на языке программирование Python, для UNIX систем.*

Определение 16 *UNIX — семейство переносимых, многозадачных и многопользовательских операционных систем, которые основаны на идеях оригинального проекта AT&T Unix, разработанного в 1970-х годах в исследовательском центре Bell Labs. Операционные системы семейства Unix характеризуются модульным дизайном, в котором каждая задача выполняется отдельной утилитой, взаимодействие осуществляется через единую файловую систему, а для работы с утилитами используется командная оболочка.*

Определение 17 *WSGI — стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером.*

Почти все компоненты, необходимые для работы сервера, были развернуты с помощью технологии Docker.

Определение 18 *Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему.*

3 Этапы создания сервера

3.1 Создание и настройка сервера

Для разработки и отладки клиент-серверной архитектуры можно было обойтись созданием и настройкой домашнего локального сервера. Но, для доступности из любого места и для реального видения скорости обработки данных, был арендован VPS сервер на территории России.

Определение 19 *VPS — услуга предоставления в аренду так называемого виртуального выделенного сервера. В плане управления операционной системой по большей части она соответствует физическому выделенному серверу. В частности: root-доступ, собственные IP-адреса, порты, правила фильтрования и таблицы маршрутизации.*

В качестве операционной системы VPS сервера была выбрана Linux Ubuntu 18.04 LTS с базовой настройкой доступа и безопасности.

Определение 20 *Ubuntu — операционная система, основанная на Debian GNU/Linux. Основным разработчиком и спонсором является компания Canonical. В настоящее время проект активно развивается и поддерживается свободным сообществом.*

После этого необходимо установить Docker, который позволит начать развертку Docker-контейнера с собранным комплектом для работы MongoDB, Gunicorn и логики архитектуры. Так же необходим Docker-контейнер с настроенным Nginx. Все эти инструменты возможно установить и использовать без использования Docker, но тогда будет утеряна мобильность архитектуры, которая может понадобится в случае смены VPS сервера на другой.

3.2 Настройка работы HTTPS

Для безопасной передачи данных между клиентом и сервером была реализована поддержка протокола передачи данных HTTPS. Для этого на сервере был получен цифровой SSL сертификат.

Определение 21 *Цифровой сертификат — выпущенный удостоверяющим центром электронный или печатный документ, подтверждающий принадлежность владельцу открытого ключа или каких-либо атрибутов. Сертификат открытого ключа удостоверяет принадлежность открытого ключа некоторому субъекту, например, пользователю. Сертификат открытого ключа содержит имя субъекта, открытый ключ, имя удостоверяющего центра, политику использования соответствующего удостоверяющему открытому ключу закрытого ключа и другие параметры, заверенные подписью удостоверяющего центра.*

В данном случае, для шифрования трафика можно было обойтись созданием самозаверенного сертификата.

Определение 22 *Самозаверенный сертификат — специальный тип сертификата, подписанный самим его субъектом. Технически данный тип ничем не отличается от сертификата, заверенного подписью удостоверяющего центра, только вместо передачи на подпись в удостоверяющий центр пользователь создаёт свою собственную сигнатуру. Создатель сертификата сам является в данном случае удостоверяющим центром.*

Но, вместо создания самозаверенного сертификата было решено обратится к центру сертификации Let's Encrypt, что позволило достичь высоких стандартов безопасности (рис. 1).

Определение 23 *Let's Encrypt — центр сертификации, предоставляющий бесплатные криптографические сертификаты X.509 для TLS-шифрования (HTTPS). Процесс выдачи сертификатов полностью автоматизирован. Проект создан для того, чтобы большая часть интернет-сайтов смогла перейти к шифрованным подключениям (HTTPS). В отличие от коммерческих центров сертификации, в данном проекте не требуется оплата, переконфигурация веб-серверов, использование электронной*

почты, обработка просроченных сертификатов, что делает процесс установки и настройки TLS-шифрования значительно более простым.

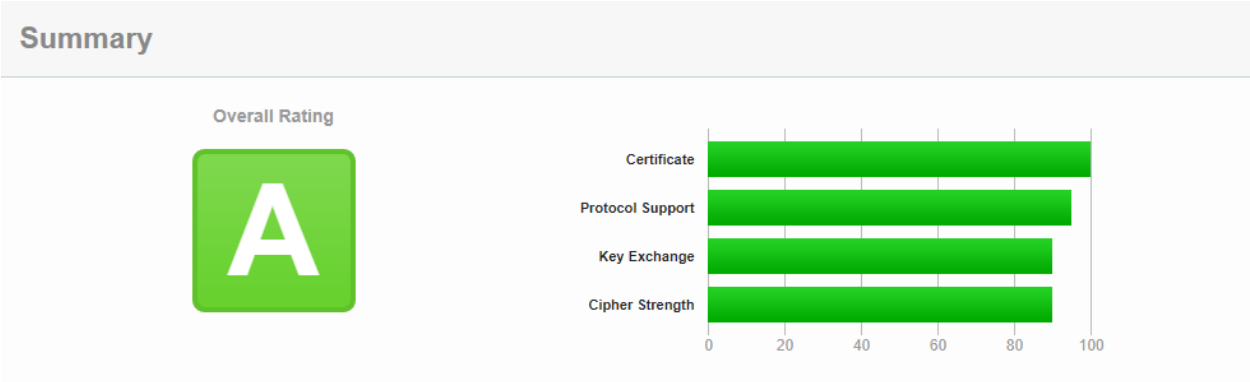

Рис. 1: Рейтинга безопасности для SSL-сертификата сервера.

3.3 Интеграция технологии JSON Web Token (JWT)

Аутентификация пользователя на сервере происходит с помощью логина и пароля, после чего клиенту выдается токен для дальнейшего отправления данных. По истечению некоторого времени, этот токен необходимо обновить посредством повторной аутентификации. Для структурирования и шифрования данного токена была использована технология JWT.

Токен представляет собой набор данных из трех секций в зашифрованном виде (рис. 2). Первая секция (HEADER) отвечает за информацию

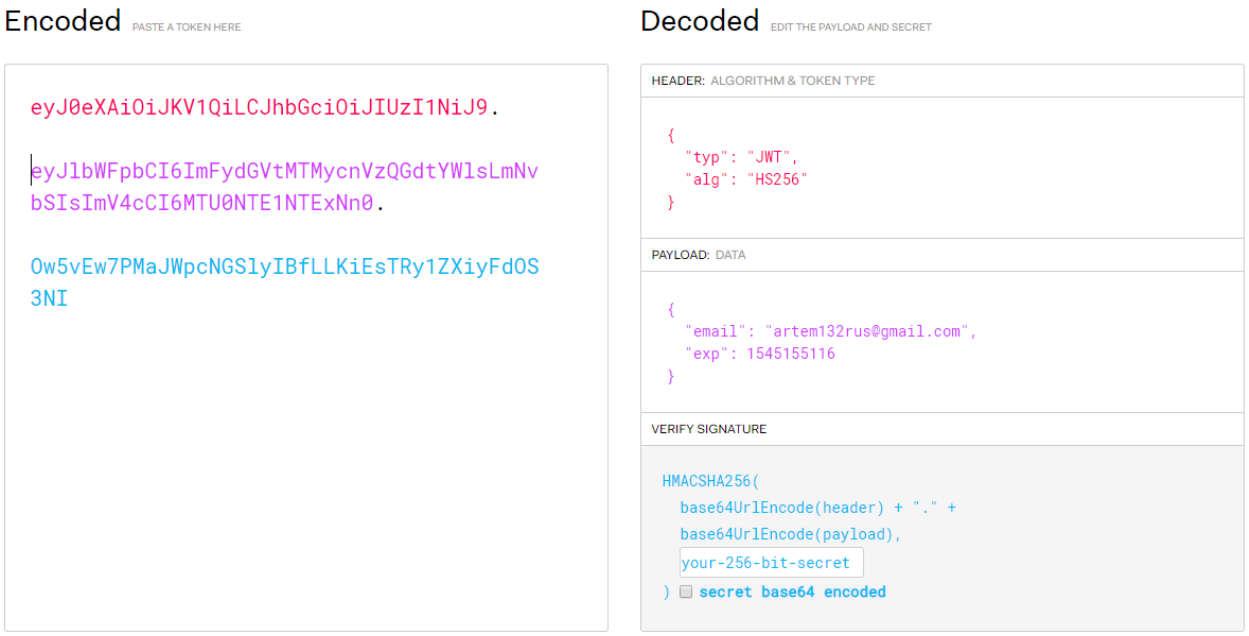

Рис. 2: Структура токена.

об используемых технологиях (JWT) и шифровании (HS256). Во второй секции (PAYLOAD) записан владелец токена (email), а также время, когда этот токен выдан. Третья секция (VERIFY SIGNATURE) содержит хеш-суммы первой и второй секции, для проверки токена на подлинность. Для безопасности, третья секция токена шифруется перед отправкой. Весь токен отправляется в кодировке Base64.

Определение 24 *Base64 — стандарт кодирования двоичных данных при помощи только 64 символов ASCII.*

3.4 Разработка собственного API

Для взаимодействия сервера и клиента было разработано собственное API, которое удовлетворяет всем нуждам проекта. Созданное API позволяет передавать данные посредством POST-запроса в формате JSON (рис. 3). Для оптимизации передачи данных, была добавлена возможность объединять множественные запросы в один POST-запрос перед отправкой. Ответ от сервера так же приходит в формате JSON (рис. 4), а если запрос был множественным, ответ на него будет содержать вложенные данные для ответа на каждый запрос.

```
{
    "requests": {
        "get_users_count": {},
        "get_all_users": {"offset": 0, "length": 1},
        "edit_users": [{
                        "_id": "5c0c1ff8086da8000a103d27",
                        "email": "admin@admin.ru",
                        "name": [
                            "Иванов",
                            "Иван",
                            "Иванович"
                        ],
                        "position": "Зам. директора"
 }]},
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlbWFpbCI"
}
```

Рис. 3: Формат запроса к API. В данном случае, выполняется запрос на получение количества пользователей, получения данных одного из них, а также последующие редактирование этого пользователя.

```
{
    "ok": true,
    "content": {
        "get_users_count": {
            "ok": true,
            "content": 27
        },
        "get_all_users": {
            "ok": true,
            "content": [{
                    "_id": "5c0c1ff8086da8000a103d27",
                    "email": "admin@admin.ru",
                    "name": [
                        "Иванов",
                        "Иван",
                        "Иванович"
                    ],
                    "position": "Зам. директора"
                }]
        },
        "edit_users": [{
                "ok": true,
                "content": "User has been changed."
            }]
    }
}
```


Рис. 4: Ответ сервера на запрос, представленный на рис. 3. Данные для каждого запроса были объединены в один JSON файл.

3.5 Реализация серверной части

Основная логика работы с БД и обработки запросов реализована за счет возможностей языка программирования Python. Для обращения к СУБД MongoDB (чтение, запись) используется библиотека pymongo. Для интеграции технологии JSON Web Token (JWT) используется библиотека jwt. Для обработки json файлов используется библиотека json.

Каждое логическое действие представляет собой обособленную функцию, которая будет вызвана при необходимости.

Листинг 1: Функция сервера для создания токена для клиента, прошедшего аутентификацию.

```
1 def create_token(self, email):
2     exp = datetime.datetime.utcnow() + datetime.timedelta(minutes=10)
3     token = jwt.encode({'email': email, 'exp': exp}, self.secret, algorithm=
          'HS256')
4     return token.decode()
```

Листинг 2: Функция сервера для проверки токена клиента.

```
1 def check_token(self, token):
2     try:
3         payload = jwt.decode(token.encode(), self.secret, algorithms=['HS256'
              ])
4     except jwt.ExpiredSignatureError:
5         return False, 'Token expired!', 403
6     except (jwt.DecodeError, AttributeError):
7         return False, 'Invalid token!', 403
8     return True, payload['email'], 200
```

Листинг 3: Функция сервера для авторизации клиента.

```
1 def authorization(self, user_data):
2     email = user_data['email']
3     pwd = user_data['pwd']
4     user = self.users.find_one({'email': email})
5     if not user:
6         return False, 'User not found!', 404
7     if user['pwd'] == sha256(pwd.encode()).hexdigest():
8         return True, self.create_token(email), 200
9     return False, 'Wrong password!', 400
```

При взаимодействии с сервером используются коды состояния HTTP для сообщения о статусе различных операций.

Определение 25 *Код состояния HTTP — часть первой строки ответа сервера при запросах по протоколу HTTP (HTTPS). Он представляет собой целое число из трёх десятичных цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделенная пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа.*

Примерами таких кодов могут быть:

- 200 OK («хорошо»);
- 400 Bad Request («плохой или неверный запрос»);
- 404 Not Found («не найдено»).

Листинг 4: Функция сервера для связывания проекта с работником. В ней присутствует множество условий проверки, каждый из которых даст некий код состояния HTTP.

```
def assign_to_projects(self, data):
    result = []
    for x in data:
        user = self.users.find_one({'email': x['email']})
        if not user:
            result.append((False, 'User not found!', 404))
            continue
        project = self.projects.find_one({'name': x['project']})
        if not project:
            result.append((False, 'Project not found!', 404))
            continue
        connection = self.connections.find_one({'user': user['_id'], '
            project': project['_id']})
        if connection:
            result.append((False, 'Project already assigned!', 400))
            continue
        self.connections.insert_one({'user': user['_id'], 'project': project
            ['_id']})
        result.append((True, 'Project has been assigned!', 200))
    return result
```

Для проверки правильной работы всех модулей сервера были использованы unit-тесты, исходные коды которых содержатся в файле tests.py.

Определение 26 *Юнит-тестирование — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.*

Листинг 5: Функция сервера для тестирования удаления пользователя из базы данных.

```
1  def test_del_users(self):
2      users_list = [{'email': 'test0@test.ru'}, {'email': 'test1@test.ru'}]
3      result = self.dbm.del_users(users_list)
4      self.assertEqual(len(result), 2, f'Too many response! ({len(result)})')
5      for r in result:
6          _, content, code = r
7          self.assertEqual(code, 200, f'Test users has not been removed! ({
               content})')
8      result = self.dbm.del_users(users_list)
9      for r in result:
10         _, content, code = r
11         self.assertEqual(code, 404, f'Removed non-existent user! ({content})'
               )
```

4 Этапы создания клиента

4.1 Настройка системы для разработки клиента

Разработка клиентской части происходила в ОС Windows 10, где и решено было установить необходимые инструменты.

Для установки языка программирования Python с официального сайта был взят установочный файл и запущен с правами администратора. Дальнейшая настройка не требовалась.

Установка PyQt5 возможна с помощью менеджера пакетов pip, который идет в комплекте с языком программирования Python. После этого настройка не требуется. Такие вспомогательные инструменты, как Qt Designer будут установлены автоматически.

Определение 27 *pip — система управления пакетами, которая используется для установки и управления программными пакетами, написанными на языке программирования Python.*

4.2 Создание интерфейса

Для создания макета интерфейса клиента использовался инструмент Qt Designer (рис. 5), позволяющий сразу увидеть результаты работы, включив превью-режим. Qt Designer создает UI-файлы, которые возможно конвертировать в необходимый формат. В данном случае, конвертирование происходило в формат языка программирования Python.

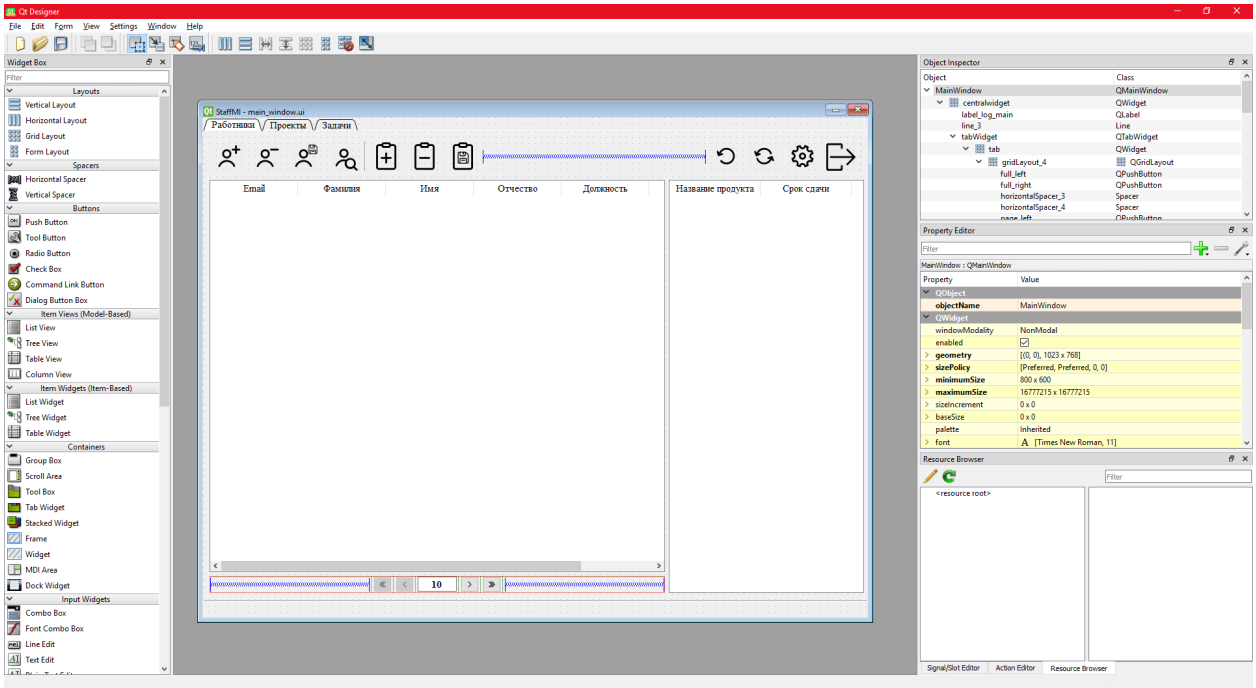

Рис. 5: Редактирование главного меню клиента в Qt Designer.

Для просмотра результатов работы с интерфейсом в данном инструменте присутствует превью-режим. Для проверки всех интерактивных элементов интерфейса конвертированный UI-файл был подключен к программе с минимальным echo-функционалом.

4.3 Реализация клиентской части

Основная логика работы и обработки событий в интерфейсе клиента реализована за счет языка программирования Python. Для обращения серверу используется собственное API. Для обработки json файлов используется библиотека json. Для работы с Qt5 используется библиотека PyQt5 и ее компоненты (QtWidgets, QtCore, QtGui и т.д.). Для работы с post-запросами используется библиотека requests.

В файле db_api.py реализована работа собственного API. Он представляет список функций-запросов к серверу, которые вызываются по мере необходимости.

Листинг 6: Функция клиента для авторизации пользователя на сервере.

```
1  def authorization(self, email, pwd):
2      data = json.dumps({"requests": {"authorization": {"email": email, "pwd":
           pwd}}, 'token': ''})
3      try:
4          response = requests.post(host, data=data).json()
5      except requests.exceptions.ConnectionError:
6          return False
7      self.token = response['content']['authorization']['content']
8      return response
```

Листинг 7: Функция клиента для отправки запроса на сервер и последующей обработки ответа.

```
1  def send_query(self, args):
2      data = json.dumps({"requests": args, 'token': self.token})
3      try:
4          response = requests.post(host, data=data).json()
5      except requests.exceptions.ConnectionError:
6          return False
7      try:
8          if response['error_code'] == 403:
9              self.authorization(self.user, self.pwd)
10             data = json.dumps({"requests": args, 'token': self.token})
11             response = requests.post(host, data=data).json()
12     except (KeyError, requests.exceptions.ConnectionError) as e:
13         if e == requests.exceptions.ConnectionError:
14             return False
15     if not (len(args) == 1):
```

```
16          pass
17      elif response['ok']:
18          try:
19              return tuple(response['content'].values())[0]['content']
20          except Exception:
21              pass
22      return response
```

В файле main_logic.py реализована вся логика работы интерфейса. Различные нажатия, события и процессы обрабатывается с помощью методов того класса, к которому они относятся. Разделение по классам была необходима для реализации принципа многооконного интерфейса. Каждый такой класс имеет свои методы для обработки различных действий и событий. В данный момент таких классов 5, а именно:

- miWindow — класс, отвечающий за главное окно интерфейса.
- loginStackWindow — класс, отвечающий за окно авторизации и смены пароля пользователя.
- inprojectDialogWindow — класс, отвечающий за диалог добавления работника в проект.
- newProjectDialogWindow — класс, отвечающий за диалог создания нового проекта.
- newUserDialogWindow — класс, отвечающий за диалог добавления нового пользователя.

Главенствующим классом считается miWindow, он же и самый объемный. Несмотря на это, первым делом, пользователь увидит окно авторизации, за которое отвечает класс loginStackWindow, в котором и будет создан объект главного класса.

4.4 Логика работы клиента

После запуска приложения, пользователь увидит окно авторизации (рис. 6). В нем же он может изменить пароль от своей учетной записи. После успешного прохождения этапа аутентификации, пользователь попадает на главное окно интерфейса (рис. 8), где доступны подменю работы с работниками и проектами. Если же аутентификации не была успешной, будет выдана соответствующая ошибка (рис. 7), также как и в случае, если с сервером не была установлена связь.

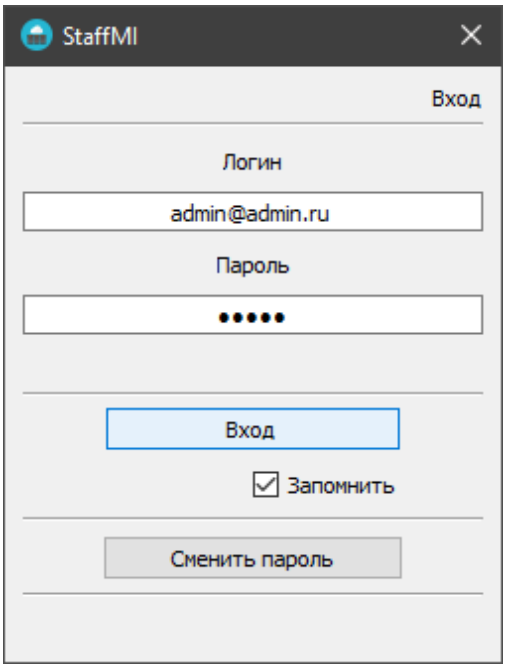

Рис. 6: Окно авторизации пользователя.

В главном окне пользователь может отредактировать данные любого проекта или работника, добавить новых (рис. 9) (рис. 10), связать выбранных работников с проектом, а также удалять любые проекты и любых работников. Все изменения данных будут отображены специальными цветами:

- Зеленый — добавленная запись;
- Желтый — отредактированная запись;
- Красный — удаленная запись.

Все данные, которые были изменены пользователем, будут хранится во временной памяти до тех пор, пока не будет дана команда отправки

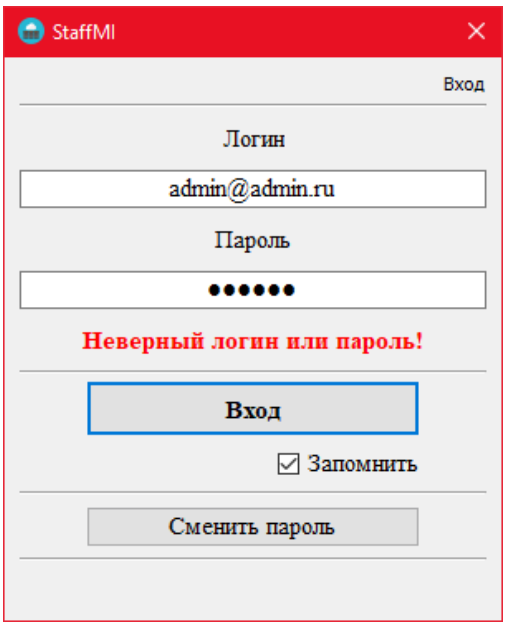

Рис. 7: Ошибка авторизации пользователя.

изменений на сервер. Сделанные изменения можно отменить, если они еще не были отправлены на сервер. Для предотвращения переизбытка используемой оперативной памяти используется постраничное отображение информации, размер этих страниц можно настроить.

В режиме реального времени происходит проверка соединения с сервером. Если произойдет разрыв соединения, пользователь будет предупрежден, а часть функционала интерфейса станет недоступной для взаимодействия.

Для общения с сервером, клиенту необходимо передавать ему вместе с запросами актуальный токен. Если был отправлен истекший токен, сервер отправит в ответе информацию об этом. В этом случае, обновление истекшего токена происходит во время работы программы так, что бы пользователь не замечал этого — повторный ввод логина и пароля требоваться не будет.

За счет кроссплатформенности PyQt5, интерфейс клиента на других ОС не будет отличаться от интерфейса на ОС Windows 10 (рис. 11), за исключением элементов интерфейса самой системы (например, стиль верхней панели окна).

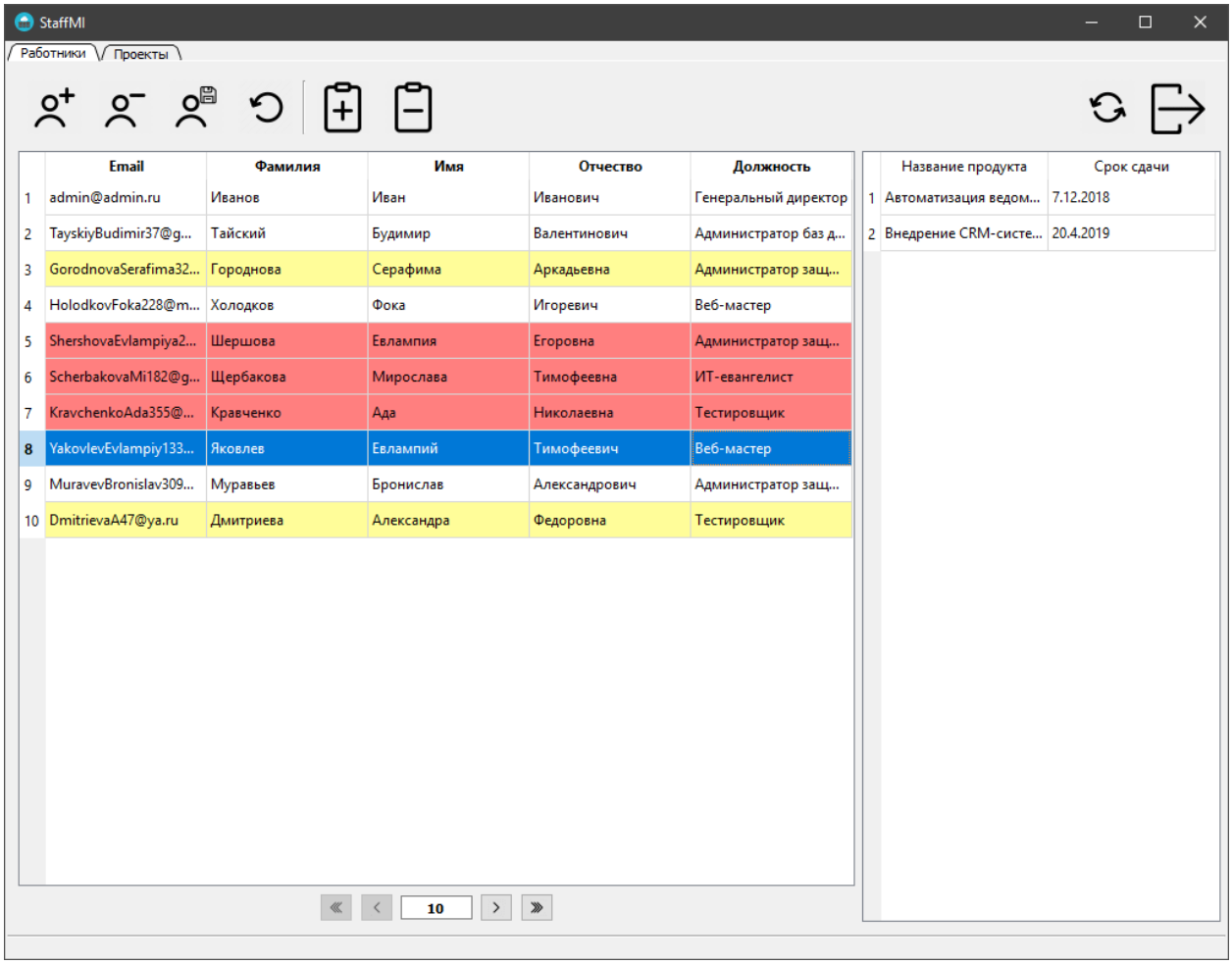

Рис. 8: Главное окно интерфейса.

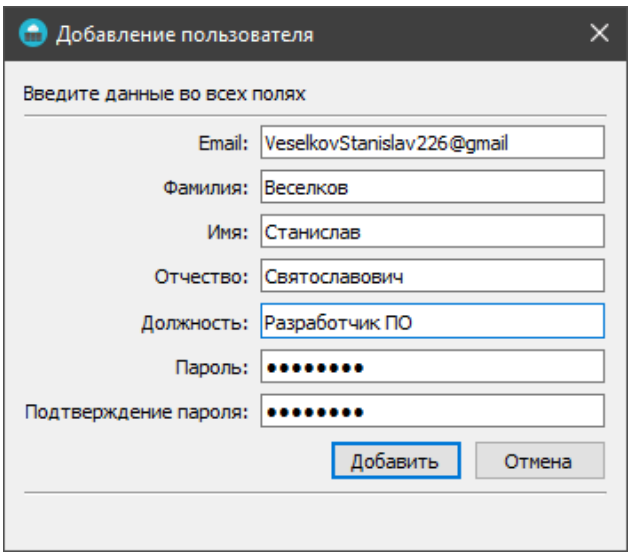

Рис. 9: Диалог добавления нового пользователя.

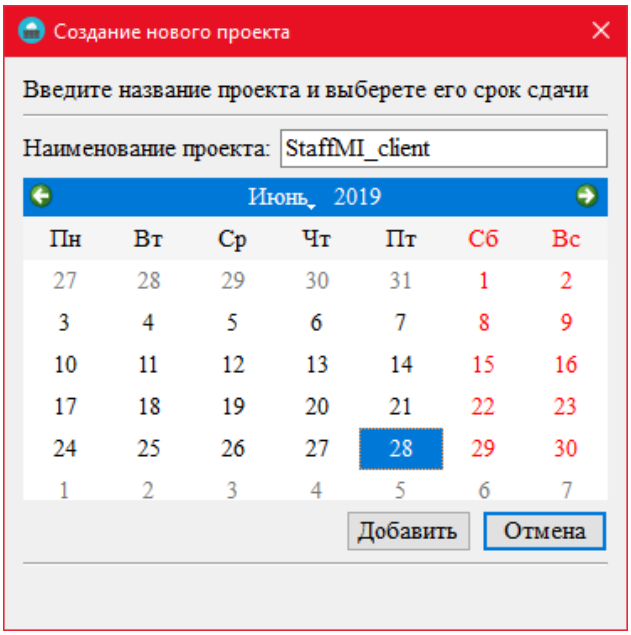

Рис. 10: Диалог добавления нового проекта.

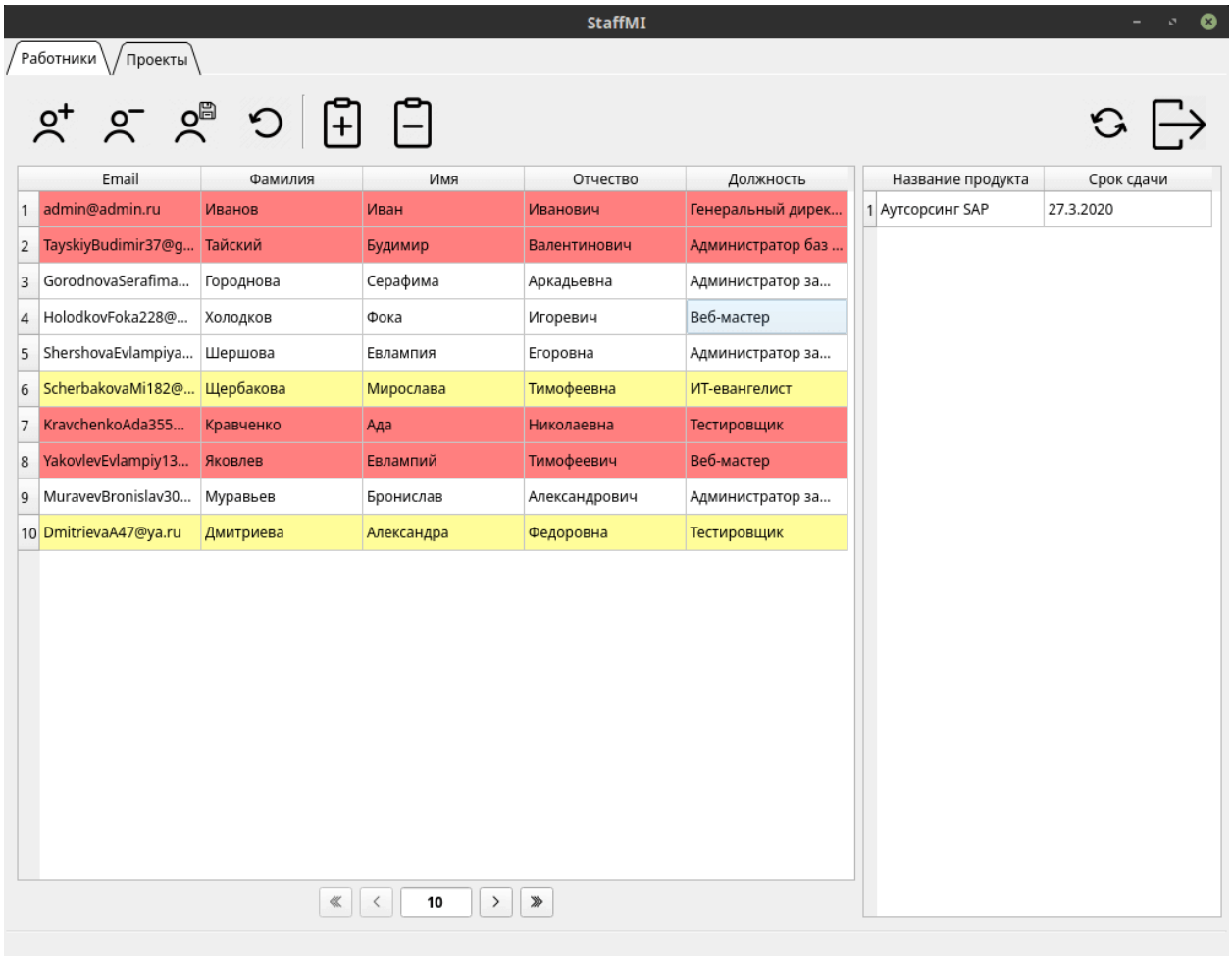

Рис. 11: Главное окно интерфейса в ОС Linux Mint.

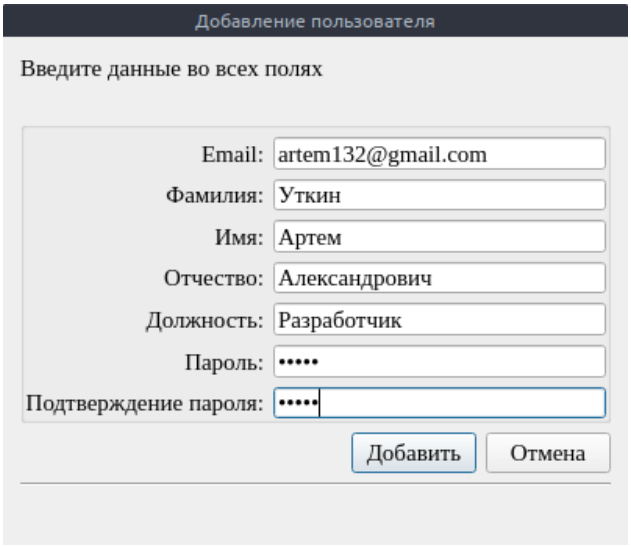

Рис. 12: Диалог добавления нового пользователя в ОС Linux Mint.

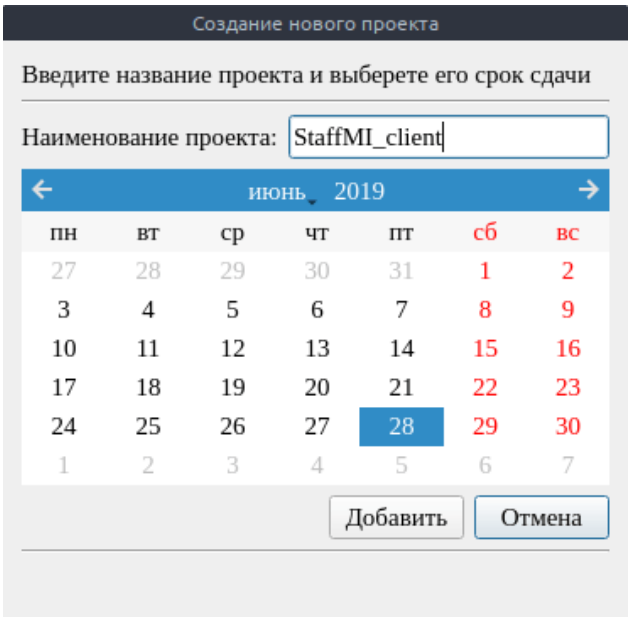

Рис. 13: Диалог добавления нового проекта в ОС Linux Mint.

5 Заключение

Современные технологии программирования предоставляют разработчикам неограниченные возможности для реализации своих идей. В данной дипломной работе, с помощью перечисленных выше технологий и собственного опыта разработки кроссплатформенного программного обеспечения с элементами интерфейса было, разработано клиент-серверное приложение для управления персоналом (работниками) и проектами (задачами) Создание данного ПО дало огромный толчок в понимании клиент-серверных архитектур, опыт работы с PyQt5, а также позволило получить практический опыт разработки подобных решений.