

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Математический факультет

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)»

Допущено к защите в ИАК 99.99.2019

Обучающийся _____ А.А. Уткин

Руководитель _____ преподаватель Груздев Д.В.

Воронеж 2019

Оглавление

Введение	3
1 Постановка задачи	4
2 Используемые технологии	5
3 Этапы создания сервера	7
3.1 Создание и настройка сервера	7
3.2 Настройка работы HTTPS	8
3.3 Интеграция технологии JSON Web Token (JWT)	9
3.4 Разработка собственного API	10
3.5 Реализация	12
4 Этапы создания клиента	14
4.1 Создание и настройка сервера	14
4.2 Создание интерфейса	15
4.3 Реализация	16
4.4 Принципы работы клиента	18
5 Заключение	21
6 Приложение	22
6.1 Исходный код server/app.py	22
6.2 Исходный код server/db.py	24
6.3 Исходный код client/db_api.py	28
6.4 Исходный код client/main_logic.py	30
Список литературы	47

Введение

Программы для контроля процесса разработки очень популярны в наше время. Любой разработчик, а иногда и группа разработчиков, используют различные системы для контроля выполнения задач и проектов (например Gitlab issues или Redmine). В современном мире любая серьезная разработка продукта не может полноценно выполняться без подобных инструментов. Именно поэтому, а также для получения опыта разработки клиент-серверных приложений, было принято решение разработать систему управления персоналом и проектами.

1 Постановка задачи

В процессе разработки была поставлена задача реализовать следующий функционал клиента:

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.

2 Используемые технологии

«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.

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 при взаимодействии программного обеспечения разнесённой в пространстве аппаратуры, соединённой тем или иным интерфейсом. Для установления связи между клиентом и сервером происходит по протоколу HTTPS.

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

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

Для реализации задуманных идей был разработан собственный API, с помощью которого происходит общение клиенткой части с сервером.

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

Данные между клиентом и сервером передаются POST-запросами в формате JSON.

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.

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

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

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.

В силу личного опыта и удобства, в качестве основного языка программирования был выбран Python.

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемо-

сти 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 использовался 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 PyQt5.

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

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.

MongoDB — документно-ориентированная 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.

Nginx — веб-сервер и почтовый прокси-сервер, работающий на IX-подобных операционных системах.

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

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 локальным сервером. Но, для доступности из любого места и для реального видения скорости обработки данных, был арендован VPS сервер на территории России.

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 фильтрации и таблицы маршрутизации.

В качестве ОС VPS сервера была выбрана Ubuntu 18.04 LTS с базовой настройкой доступа и безопасности. После этого необходимо установить Docker, после чего устанавливается Docker-контейнер 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 в процессе разработки имели место частые смены VPS серверов.

3.2 Настройка работы HTTPS

Для поддержки протокола передачи данных HTTPS, на сервере необходимо получить цифровой SSL сертификат.

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 удостоверяемому открытому ключу закрытого ключа и другие параметры, заверенные подписью удостоверяющего центра.

В данном случае, для шифрования трафика можно было обойтись самозаверенным сертификатом.

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 обратиться к центру сертификации Let's Encrypt.

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.

3.3 Интеграция технологии JSON Web Token (JWT)

Аутентификация пользователя происходит с помощью логина и пароля, после чего клиенту выдается токен для дальнейшего отправления данных. По истечению некоторого времени, этот токен необходимо обновить по средствам повторной аутентификации.

Токен представляет собой набор данных из трех секций в зашифрованном виде.

Encoded

PASTE A TOKEN HERE

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlbWFpbCI6ImFydGVtMTMybnVzQGdtYWlsLmNvbSI6ImV4cCI6MTU0NTE1NTEuXn0.0w5vEw7PMaJWpcNGSlyIBfLLKiEsTRy1ZXiyFdOS3NI

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "typ": "JWT",  "alg": "HS256"}
```

PAYLOAD: DATA

```
{  "email": "artem132rus@gmail.com",  "exp": 1545155116}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) ☐ secret base64 encoded
```

Рис. 1: структура токена.

Первая секция отвечает за информацию об используемых технологиях и шифровании. Во второй секции записан владелец токена (email), а также время, когда этот токен выдан. Третья секция 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.

Base64 – стандарт кодирования двоичных данных при помощи только 64 символов ASCII.

3.4 Разработка собственного API

Для взаимодействия сервера и клиента было разработано API, которое удовлетворяло всем нуждам. Созданное API позволяет передавать данные по средствам POST-запроса в формате JSON. 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, 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.eyJlbWFpbiCI"
```

Рис. 2: 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. 3: ответ сервера на запрос, представленный на рис. 2. Данные для каждого запроса были объединены в один JSON файл.

3.5 Реализация

Основная логика работы с БД и обработки запросов реализована за счет языка программирования Python. Для обращения к БД (чтение, запись) используется библиотека `rumongo`. 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.

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

Рис. 4: пример функции создания токена для клиента, прошедшего аутентификацию.

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

Рис. 5: пример функции проверки токена клиента.

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

Рис. 6: пример функции авторизации клиента.

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

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 отделённая 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, неверный запрос»);
- 404 Not Found («не найдено»).

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

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.

4 Этапы создания клиента

4.1 Создание и настройка сервера

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

`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, 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.

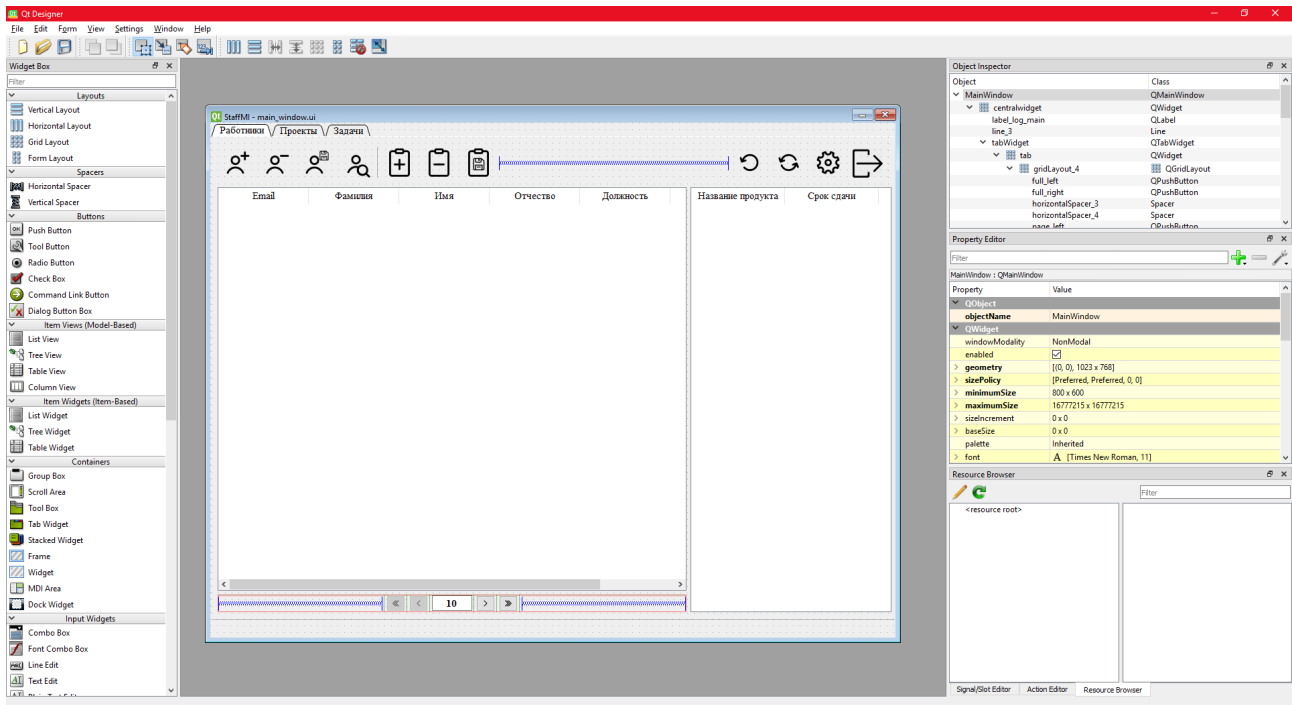


Рис. 7: редактирование главного меню клиента в Qt Designer.

4.3 Реализация

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 производные (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.

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

Рис. 8: функция авторизации клиента.

```
def send_query(self, args):
    data = json.dumps({"requests": args, 'token': self.token})
    try:
        response = requests.post(host, data=data).json()
    except requests.exceptions.ConnectionError:
        return False
    try:
        if response['error_code'] == 403:
            self.authorization(self.user, self.pwd)
            data = json.dumps({"requests": args, 'token': self.token})
            response = requests.post(host, data=data).json()
    except (KeyError, requests.exceptions.ConnectionError) as e:
        if e == requests.exceptions.ConnectionError:
            return False
    if not (len(args) == 1):
        pass
    elif response['ok']:
        try:
            return tuple(response['content'].values())[0]['content']
        except Exception:
            pass
    return response
```


Рис. 9: функция отправки запроса на сервер и последующей обработки ответа.

```
def get_all_users(self, args):
    return self.send_query({"get_all_users": args})

def get_all_projects(self, args):
    return self.send_query({"get_all_projects": args})

def add_users(self, users):
    return self.send_query({"add_users": users})

def edit_users(self, users):
    return self.send_query({"edit_users": users})

def del_users(self, emails):
    return self.send_query({"del_users": emails})
```

Рис. 10: примеры функций-запросов.

В файле `main_logic.py` реализована вся логика работы интерфейса. Различные нажатия, события и процессы обрабатываются по средствам методов того класса, к которому они относятся. Разделение по классам необходима для реализации 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 Принципы работы клиента

После запуска приложения, пользователь увидит окно авторизации. В нем же он может изменить пароль от своей учетной записи. После успешного прохождения этапа аутентификации, пользователь попадает на главное окно интерфейса, где доступны подменю работы с работниками и проектами.

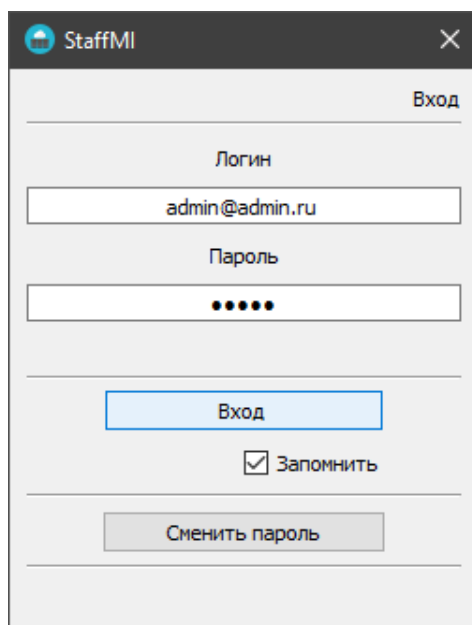


Рис. 11: окно авторизации пользователя.

Пользователь может отредактировать данные любого проекта или работника, добавить новых, 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.

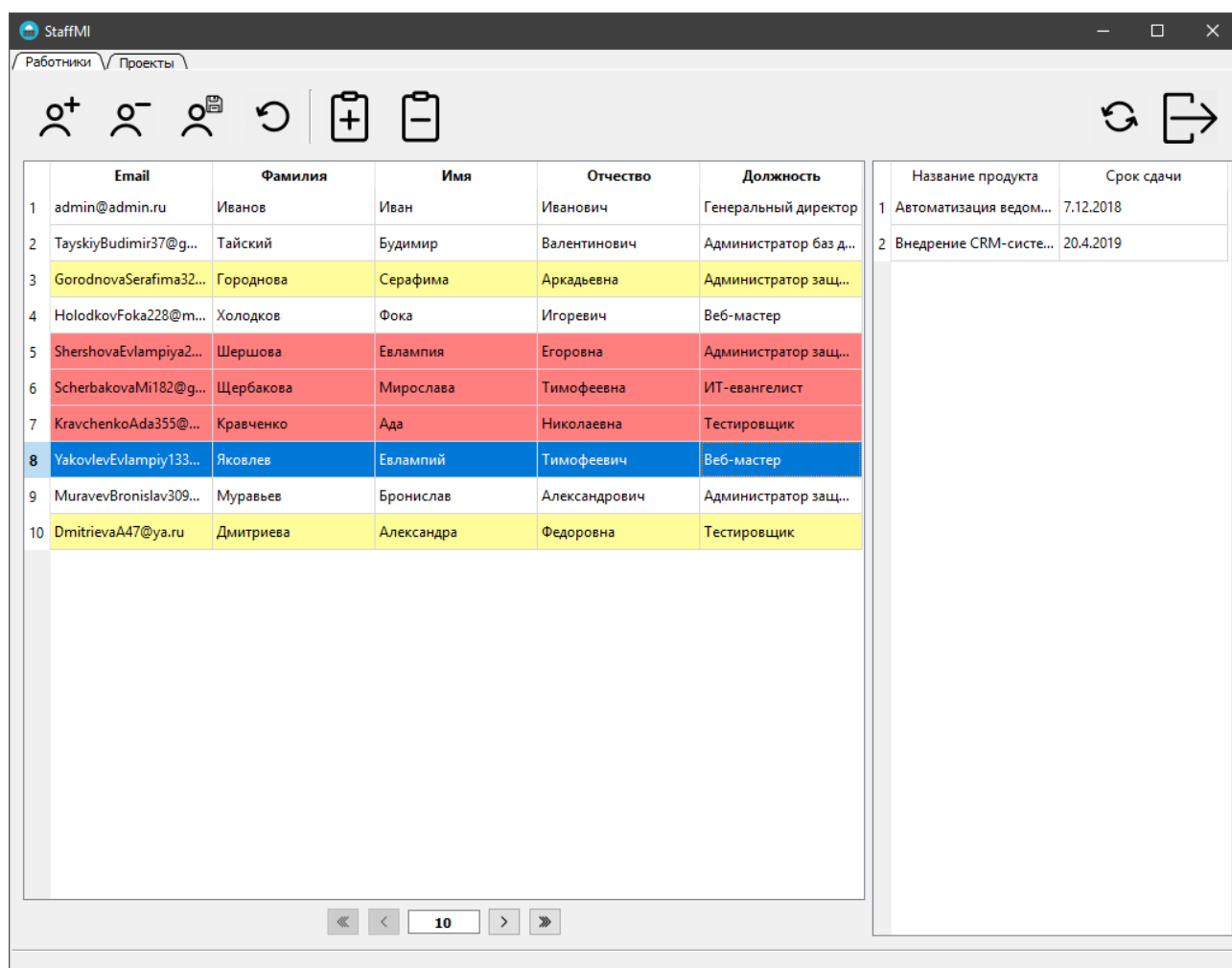


Рис. 12: главное окно интерфейса.

Добавление пользователя

Введите данные во всех полях

Email: VeselkovStanislav226@gmail

Фамилия: Веселков

Имя: Станислав

Отчество: Святославович

Должность: Разработчик ПО

Пароль: ••••••••

Подтверждение пароля: ••••••••

Добавить Отмена

Рис. 13: окно добавления нового пользователя.

Все данные, которые были изменены пользователем, будут храниться во временной памяти до тех пор, пока не будет дана команда отправки 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 функционал интерфейса урезан. 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.

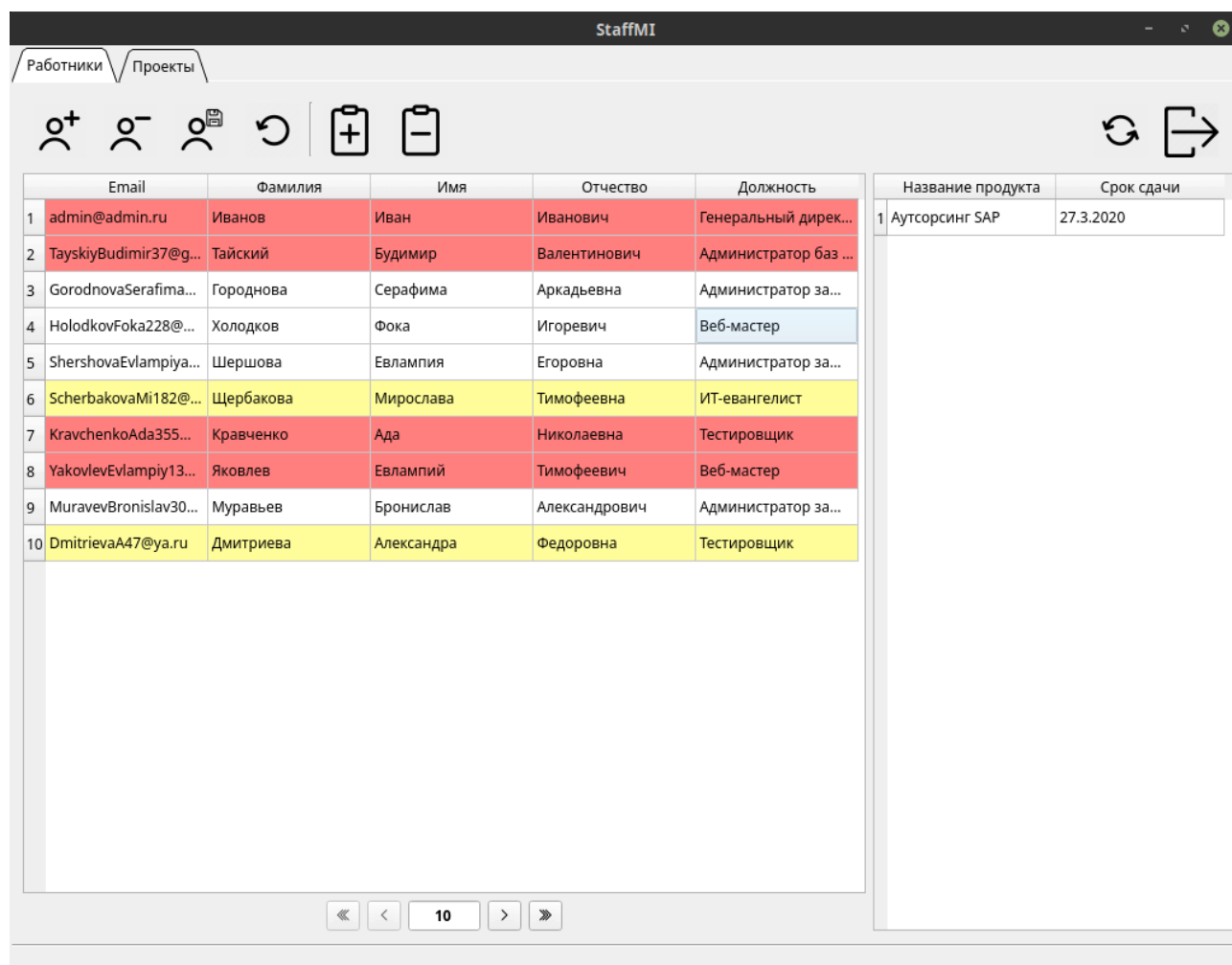


Рис. 14: главное окно интерфейса на ОС Linux Mint.

5 Заключение

Современные технологии программирования предоставляют разработчикам неограниченные возможности для реализации своих идей. В данной дипломной работе с помощью перечисленных выше технологий было разработано клиент-серверное приложение для управления персоналом и проектами, которое дало огромный толчок в понимании клиент-серверных архитектур, а также позволило получить практический опыт разработки подобных решений.

6 Приложение

6.1 Исходный код server/app.py

```
1  # -*- coding: utf-8 -*-
2
3  import functools
4  import json
5
6  from flask import Flask, Response, request
7  from jsonschema import FormatChecker, ValidationError, validate
8
9  import db
10
11  app = Flask(__name__)
12  app.secret_key = b'Xc-Z3N3G51211fgjdgfjQ=eDsUv139.Ghd4*=6~=WYT5125UN.'
13  app.config['MAX_CONTENT_LENGTH'] = 32 * 1024 * 1024
14
15  dbm = db.DBManager()
16
17  with open('schema.json', 'r') as f:
18      json_schema = json.load(f)
19
20  def response_formatter(response):
21      if isinstance(response, tuple): # TODO: Refactor this
22          ok, content, code = response
23          return {'ok': ok, 'content': content} if ok else {'ok': ok, 'content':
              content, 'error_code': code}
24  result = []
25  for r in response:
26      ok, content, code = r
27      result.append({'ok': ok, 'content': content} if ok else {'ok': ok, '
          content': content, 'error_code': code})
28  return result
29
30  def response(func):
31      @functools.wraps(func)
32      def response_wrapper(*args, **kwargs):
33          response = func(*args, **kwargs)
34          response_text = response_formatter(response)
35          status = response_text.get('error_code', 0) or 200
36          return response(status=status, mimetype='application/json', response=json.
              dumps(response_text))
37  return response_wrapper
38
39  def requests_handler(requests):
40      result = [response_formatter(getattr(dbm, key)(requests[key])) for key in
          requests]
41      return dict(zip(requests, result))
42
43  @app.route('/', methods=['GET'])
```

```

44 @response
45 def main_get():
46     return False, 'Only POST requests are allowed!', 400
47
48 @app.route('/', methods=['POST'])
49 @response
50 def main_post():
51     try:
52         r = json.loads(request.data.decode())
53     except json.decoder.JSONDecodeError:
54         return False, 'Invalid JSON!', 400
55
56     try:
57         validate(r, json_schema, format_checker=FormatChecker())
58     except ValidationError as e:
59         return False, e.message, 400
60
61     # TODO: Refactor this
62     try:
63         return True, requests_handler({'authorization': r['requests']['
64             authorization']}), 200
65     except KeyError:
66         pass
67
68     ok, text, code = dbm.check_token(r['token'])
69     if not ok:
70         return ok, text, code
71
72     return True, requests_handler(r['requests']), 200
73
74 if __name__ == "__main__":
75     app.run(host='0.0.0.0')

```

6.2 Исходный код server/db.py

```
1  # -*- coding: utf-8 -*-
2
3  import datetime
4  import os
5  from bson import ObjectId
6  from hashlib import sha256
7
8  import jwt
9  import pymongo
10
11 class DBManager:
12     def __init__(self):
13         try:
14             self.client = pymongo.MongoClient(host='db')
15         except pymongo.errors.ConnectionFailure as e:
16             print(e)
17         self.db = self.client.software
18
19         # Collections
20         self.users = self.db.users
21         self.projects = self.db.projects
22
23         self.secret = os.getenv('DB_SECRET', 'MOSTSECUREKEY')
24
25         self.add_users([
26             'email': os.getenv('ADMIN_EMAIL', 'admin@admin.ru'),
27             'pwd': os.getenv('ADMIN_PWD', '12345'),
28             'name': ['Ivanov', 'Ivan', 'Ivanovich'],
29             'position': 'Main admin'
30         ])
31
32         # Authentication
33
34     def create_token(self, email):
35         exp = datetime.datetime.utcnow() + datetime.timedelta(minutes=10)
36         token = jwt.encode({'email': email, 'exp': exp}, self.secret, algorithm='
37             HS256')
38         return token.decode()
39
40     def check_token(self, token):
41         try:
42             payload = jwt.decode(token.encode(), self.secret, algorithms=['HS256'])
43         except jwt.ExpiredSignatureError:
44             return False, 'Token expired!', 403
45         except (jwt.DecodeError, AttributeError):
46             return False, 'Invalid token!', 403
47         return True, payload['email'], 200
48
49     # Users
```



```

50 def add_users(self, users_data):
51     result = []
52     for user in users_data:
53         if self.users.find_one({'email': user['email']}):
54             result.append((False, 'User already exist!', 400))
55         else:
56             pwd_hash = sha256(user['pwd'].encode()).hexdigest()
57             user['pwd'] = pwd_hash
58             self.users.insert_one(user)
59             result.append((True, 'User has been added.', 200))
60     return result
61
62 def del_users(self, users_list):
63     result = []
64     for _id in users_list:
65         if self.users.delete_one({'_id': ObjectId(_id)}).deleted_count:
66             result.append((True, 'User has been removed.', 200))
67         else:
68             result.append((False, 'User not found!', 404))
69     return result
70
71 def edit_users(self, users_data):
72     result = []
73     for user in users_data:
74         if not self.users.find_one({'_id': ObjectId(user['_id'])}):
75             result.append([False, 'User not found!', 404])
76         else:
77             pwd_hash = sha256(user['pwd'].encode()).hexdigest()
78             user['pwd'] = pwd_hash
79             _id = user.pop('_id')
80             self.users.replace_one({'_id': ObjectId(_id)}, user)
81             result.append((True, 'User has been changed.', 200))
82     return result
83
84 def authorization(self, user_data):
85     email = user_data['email']
86     pwd = user_data['pwd']
87     user = self.users.find_one({'email': email})
88     if not user:
89         return False, 'User not found!', 404
90     if user['pwd'] == sha256(pwd.encode()).hexdigest():
91         return True, self.create_token(email), 200
92     return False, 'Wrong password!', 400
93
94 def get_all_users(self, params):
95     offset = params['offset']
96     length = params['length']
97     users = self.users.find({}, {'pwd': False})
98     users = list(users)[offset:offset + length]
99     for u in users:
100         u['_id'] = str(u['_id'])
101     return True, tuple(users), 200

```

```

102
103 def change_password(self, user_data):
104     email = user_data['email']
105     old_pwd = user_data['old_pwd']
106     new_pwd = user_data['new_pwd']
107     user = self.users.find_one({'email': email})
108     if not user:
109         return False, 'User not found!', 404
110     if user['pwd'] == sha256(old_pwd.encode()).hexdigest():
111         pwd_hash = sha256(new_pwd.encode()).hexdigest()
112         user['pwd'] = pwd_hash
113         self.users.replace_one({'_id': ObjectId(user['_id'])}, user)
114         return True, 'Password has been changed.', 200
115     return False, 'Wrong password!', 400
116
117 # Projets
118
119 def add_projects(self, projects_data):
120     result = []
121     for project in projects_data:
122         if self.projects.find_one({'name': project['name']}):
123             result.append([False, 'Project already exist!', 400])
124         else:
125             self.projects.insert_one(project)
126             result.append((True, 'Project has been added.', 200))
127     return result
128
129 def del_projects(self, projects_list):
130     result = []
131     for _id in projects_list:
132         if self.projects.delete_one({'_id': ObjectId(_id)}).deleted_count:
133             result.append((True, 'Project has been removed.', 200))
134         else:
135             result.append((False, 'Project not found!', 404))
136     return result
137
138 def edit_projects(self, projects_data):
139     result = []
140     for project in projects_data:
141         if not self.projects.find_one({'_id': ObjectId(project['_id'])}):
142             result.append([False, 'Project not found!', 404])
143         else:
144             _id = project['_id']
145             self.projects.replace_one({'_id': ObjectId(_id)}, project)
146             result.append((True, 'Project has been changed.', 200))
147     return result
148
149 def get_all_projects(self, params):
150     offset = params['offset']
151     length = params['length']
152     projects = self.projects.find({})
153     projects = list(projects)[offset:offset + length]

```

```
154     for p in projects:
155         p['_id'] = str(p['_id'])
156     return True, tuple(projects), 200
```

6.3 Исходный код client/db_api.py

```
1 import json
2 import requests
3 host = 'https://pms.kmm-vsu.ru/'
4
5
6 class API:
7     def __init__(self):
8         self.token = ''
9         self.user = ''
10        self.pwd = ''
11
12    def check_connect(self):
13        try:
14            requests.get(host)
15            return True
16        except requests.exceptions.ConnectionError:
17            return False
18
19    def authorization(self, email, pwd):
20        data = json.dumps({"requests": {"authorization": {"email": email, "pwd":
21            pwd}}, 'token': ''})
22        try:
23            response = requests.post(host, data=data).json()
24        except requests.exceptions.ConnectionError:
25            return False
26        self.token = response['content']['authorization']['content']
27        return response
28
29    def send_query(self, args):
30        data = json.dumps({"requests": args, 'token': self.token})
31        try:
32            response = requests.post(host, data=data).json()
33        except requests.exceptions.ConnectionError:
34            return False
35        try:
36            if response['error_code'] == 403:
37                self.authorization(self.user, self.pwd)
38                data = json.dumps({"requests": args, 'token': self.token})
39                response = requests.post(host, data=data).json()
40        except (KeyError, requests.exceptions.ConnectionError) as e:
41            if e == requests.exceptions.ConnectionError:
42                return False
43        if not (len(args) == 1):
44            pass
45        elif response['ok']:
46            try:
47                return tuple(response['content'].values())[0]['content']
48            except Exception:
49                pass
50        return response
```

```

50
51 def get_all_users(self, args):
52     return self.send_query({"get_all_users": args})
53
54 def get_all_projects(self, args):
55     return self.send_query({"get_all_projects": args})
56
57 def add_users(self, users):
58     return self.send_query({"add_users": users})
59
60 def edit_users(self, users):
61     return self.send_query({"edit_users": users})
62
63 def del_users(self, emails):
64     return self.send_query({"del_users": emails})
65
66 def add_projects(self, projects):
67     return self.send_query({"add_projects": projects})
68
69 def edit_projects(self, projects):
70     return self.send_query({"edit_projects": projects})
71
72 def del_projects(self, projects):
73     return self.send_query({"del_projects": projects})
74
75 def change_password(self, args):
76     return self.send_query({"change_password": args})
77
78 def get_users_count(self):
79     return self.send_query({"get_users_count": {}})
80
81 def assign_to_projects(self, args):
82     return self.send_query({"assign_to_projects": args})
83
84 def remove_from_projects(self, args):
85     return self.send_query({"remove_from_projects": args})
86
87 # def get_users_projects(self, ):
88 # return self.send_query({}, 'get_users_projects')
89
90 # def get_all_projects(self):
91 # return self.send_query({}, 'get_all_projects')

```

6.4 Исходный код client/main_logic.py

```
1  # -*- coding: utf-8 -*-
2
3  import json
4  import sys
5
6  from PyQt5 import QtWidgets
7  from PyQt5.QtCore import Qt, QTimer
8  from PyQt5.QtGui import QColor
9
10 import db_api
11 # Import Interface Files
12 from ui import add_inproject_dialog
13 from ui import (add_new_project_dialog, add_new_user_dialog, login_stack,
14                 main_window)
15 # Class responsible for the main window of working with the database
16 class miWindow(QtWidgets.QMainWindow, main_window.Ui_MainWindow):
17     def __init__(self, api):
18         super().__init__()
19         self.setupUi(self)
20
21         self.api = api
22
23         # Local save changes to workers
24         self.worker_rows_to_delete = [] # Here are the workers selected for
25                                         # deletion
26         self.worker_rows_were_changed = [] # Employees whose data has been changed
27                                         # are stored here
28         self.unpacked_worker_rows_were_changed = [] # To save data from other
29                                         # pages
30         self.new_worker_rows = [] # New employees are stored here.
31         self.old_data_workers_rows = [] # Old employee data is stored here if it
32                                         # is necessary to return it.
33
34         # Local save changes to projects
35         self.project_rows_to_delete = [] # The projects selected for deletion are
36                                         # stored here
37         self.project_rows_were_changed = [] # Projects whose data has been
38                                         # modified are stored here
39         self.unpacked_project_rows_were_changed = [] # To save data from other
40                                         # pages
41         self.new_project_rows = [] # New projects are stored here
42         self.old_data_projects_rows = [] # Old project data is stored here if you
43                                         # need to return it.
44
45         self.user_projects = {} # Dictionary "email - project"
46         self.clicked_worker_row = None # Modal selected worker
47
48         self.workers_table_page = 0 # The current page of the workers table
```

```

42     self.update_workers()
43     self.update_projects()
44
45     # buttons events for workers table
46     self.add_worker.clicked.connect(self.add_worker_click)
47     self.del_worker.clicked.connect(self.del_worker_click)
48     self.save_workers.clicked.connect(self.save_workers_click)
49     self.workers_table.doubleClicked.connect(self.changed_cell_workers_table)
50     self.workers_table.itemClicked.connect(self.show_user_projects)
51
52     # buttons events for employee projects
53     self.new_inproject.clicked.connect(self.new_inproject_click)
54     self.new_inproject.clicked.connect(self.current_projects_table.
        scrollToBottom)
55     self.del_inproject.clicked.connect(self.del_inproject_click)
56     self.save_inprojects.clicked.connect(self.save_inprojects_click)
57
58     # buttons events for projects table
59     self.add_project.clicked.connect(self.add_project_click)
60     self.del_project.clicked.connect(self.del_project_click)
61     self.save_projects.clicked.connect(self.save_projects_click)
62     self.projects_table.doubleClicked.connect(self.changed_cell_projects_table
        )
63
64     # buttons events for other functions
65     self.undo_changes_workers.clicked.connect(self.undo_changes_workers_table)
66     self.undo_changes_projects.clicked.connect(self.
        undo_changes_projects_table)
67     self.logout.clicked.connect(self.logout_click)
68     self.logout_2.clicked.connect(self.logout_click)
69     self.settings.clicked.connect(self.settings_click)
70     self.settings_2.clicked.connect(self.settings_click)
71     self.update_data.clicked.connect(self.update_workers_table_click)
72     self.update_data_2.clicked.connect(self.update_projects_table_click)
73
74     # buttons events pagination display table workers
75     self.full_left.clicked.connect(self.full_left_click)
76     self.page_left.clicked.connect(self.page_left_click)
77     self.page_right.clicked.connect(self.page_right_click)
78     self.full_right.clicked.connect(self.full_right_click)
79     self.size_page.editingFinished.connect(self.full_left_click)
80
81     # Properties for hiding columns with _id entries
82     self.workers_table.setColumnHidden(5, True)
83     self.projects_table.setColumnHidden(2, True)
84
85     self._main_timer = QTimer()
86     self._main_timer.timeout.connect(self._timer_tick)
87     self._main_timer.start(10000)
88     self.connect_status = 0
89
90     def _timer_tick(self):

```

```

91     self.connect_status = self.api.check_connect()
92     if self.connect_status:
93         self.label_log_main.setText("")
94         self.button_status(True)
95     else:
96         self.label_log_main.setText("Server is not available!")
97         self.button_status(False)
98
99     def button_status(self, status):
100         self.add_worker.setEnabled(status)
101         self.save_workers.setEnabled(status)
102         self.new_inproject.setEnabled(status)
103         self.del_inproject.setEnabled(status)
104         self.save_inprojects.setEnabled(status)
105         self.update_data.setEnabled(status)
106         self.add_project.setEnabled(status)
107         self.save_projects.setEnabled(status)
108         self.update_data_2.setEnabled(status)
109
110     def resizeEvent(self, event):
111         self.workers_table.setColumnWidth(0, self.width() / 6)
112         self.workers_table.setColumnWidth(1, self.width() / 12)
113         self.workers_table.setColumnWidth(2, self.width() / 12)
114         self.workers_table.setColumnWidth(3, self.width() / 12)
115         self.workers_table.setColumnWidth(4, self.width() / 4)
116         self.projects_table.setColumnWidth(0, self.width() / 1.5)
117
118     # Update employee table
119     def update_workers(self):
120         self.connect_status = self.api.check_connect()
121         if self.connect_status:
122             answer = self.api.get_all_users({"offset": self.workers_table_page, "
123                 length": int(self.size_page.text())})
124             self.user_projects.clear()
125             self.workers_table.clearContents() # Table cleaning
126             self.workers_table.setRowCount(0) #
127             self.current_projects_table.clearContents() # Clearing the users project
128             self.current_projects_table.setRowCount(0)
129             self.current_projects_table.setRowCount(0)
130             for worker in answer:
131                 self.user_projects.update({worker["email"]: worker["projects"]})
132                 row_pos = self.workers_table.rowCount()
133                 self.workers_table.insertRow(row_pos)
134                 self.workers_table.setItem(row_pos, 0, QtWidgets.QTableWidgetItem(
135                     worker["email"]))
136                 for x in range(1, 4): # Parsing of the name of the employee
137                     self.workers_table.setItem(row_pos, x, QtWidgets.QTableWidgetItem(
138                         worker["name"][x - 1]))
139                 self.workers_table.setItem(row_pos, 4, QtWidgets.QTableWidgetItem(
140                     worker["position"]))
141                 self.workers_table.setItem(row_pos, 5, QtWidgets.QTableWidgetItem(
142                     worker["_id"]))

```



```

137         # Attempt to change the row id in the table
138         index_list = [str(item + 1) for item in range(self.workers_table_page,
139             self.workers_table_page + int(self.size_page.text()))]
140         self.workers_table.setVerticalHeaderLabels(index_list)
141     else:
142         self.label_log_main.setText("Server is not available!")
143         return
144
145     # Update projects table
146     def update_projects(self):
147         self.connect_status = self.api.check_connect()
148         if self.connect_status:
149             answer = self.api.get_all_projects({"offset": self.workers_table_page, "
150                 length": 1000}) # No pages
151             self.projects_table.clearContents() # Table cleaning
152             self.projects_table.setRowCount(0)
153             for project in answer:
154                 row_pos = self.projects_table.rowCount()
155                 self.projects_table.insertRow(row_pos)
156                 self.projects_table.setItem(row_pos, 0, QtWidgets.QTableWidgetItem(
157                     project["name"]))
158                 self.projects_table.setItem(row_pos, 1, QtWidgets.QTableWidgetItem(
159                     project["deadline"]))
160                 self.projects_table.setItem(row_pos, 2, QtWidgets.QTableWidgetItem(
161                     project["_id"]))
162             else:
163                 self.label_log_main.setText("Server is not available!")
164                 return
165
166     # Unpacking data from QModelIndex into edited worker lists (solving strange
167     # model problems)
168     def unpacked_worker_changed(self):
169         for obj in self.worker_rows_were_changed:
170             self.unpacked_worker_rows_were_changed.append({
171                 "_id": obj.sibling(obj.row(), 5).data(),
172                 "email": obj.sibling(obj.row(), 0).data(),
173                 "name": [obj.sibling(obj.row(), 1).data(), obj.sibling(obj.row(), 2).
174                     data(), obj.sibling(obj.row(), 3).data()],
175                 "position": obj.sibling(obj.row(), 4).data()
176             })
177
178     # Call add user dialog
179     def add_worker_click(self):
180         self.full_right_click()
181         chose_dialog = newUserDialogWindow(self, self.api, self.workers_table,
182             self.new_worker_rows)
183         chose_dialog.exec_()
184
185     # Removal of workers
186     def del_worker_click(self):
187         selected_rows = self.workers_table.selectionModel().selectedRows()
188         for row in selected_rows:

```

```

181     table = row.model()
182     index = row.row()
183     row_id = row.sibling(row.row(), 5).data()
184     try: # If the employee is already marked as deleted, remove him from the
           lists for deletion
185         self.worker_rows_to_delete.remove(row_id)
186         for x in range(0, 5):
187             table.setData(table.index(index, x), QColor(255, 255, 255), Qt.
                           BackgroundRole)
188     except ValueError:
189         self.worker_rows_to_delete.append(row_id)
190         for x in range(0, 5):
191             table.setData(table.index(index, x), QColor(255, 127, 127), Qt.
                           BackgroundRole)
192
193 # Sending all changes to the employee table to the server
194 def save_workers_click(self):
195     self.connect_status = self.api.check_connect()
196     if self.connect_status:
197         if self.worker_rows_to_delete:
198             self.api.del_users(self.worker_rows_to_delete)
199             self.worker_rows_to_delete.clear()
200         if self.new_worker_rows:
201             self.api.add_users(self.new_worker_rows)
202             self.new_worker_rows.clear()
203         self.unpacked_worker_changed()
204         if self.unpacked_worker_rows_were_changed:
205             self.api.edit_users(self.unpacked_worker_rows_were_changed)
206             self.unpacked_worker_rows_were_changed.clear()
207             self.worker_rows_were_changed.clear()
208         self.update_workers()
209     else:
210         self.label_log_main.setText("Server is not available!")
211     return
212
213 # Employee data change
214 # [A bad signal is used, an analog is needed]
215 def changed_cell_workers_table(self):
216     row = self.workers_table.selectionModel().selectedRows()[0]
217     if not self.worker_rows_to_delete.count(row.sibling(row.row(), 5).data()):
218         table = row.model()
219         index = row.row()
220         self.old_data_workers_rows.append({
221             "_id": row.sibling(index, 5).data(), # Id is hidden in the table
222             "email": row.sibling(index, 0).data(),
223             "name": [row.sibling(index, 1).data(), row.sibling(index, 2).data(),
                     row.sibling(index, 3).data()],
224             "position": row.sibling(index, 4).data()
225         })
226         for x in range(0, 5):
227             table.setData(table.index(index, x), QColor(255, 253, 153), Qt.
                           BackgroundRole)

```

```

228         self.worker_rows_were_changed.append(row)
229
230     # Add selected employee to project
231     # [Awful implementation, you also need to get away from instant sending to
        the server]
232     def new_inproject_click(self):
233         rows = self.workers_table.selectionModel().selectedRows()
234         if rows:
235             self.connect_status = self.api.check_connect()
236             if self.connect_status:
237                 answer = self.api.get_all_projects({"offset": 0, "length": 1000}) # :/
238                 chose_dialog = inprojectDialogWindow(answer)
239                 chose_dialog.exec_()
240                 answer_user = chose_dialog.answer
241                 if answer_user:
242                     data = []
243                     for item in rows:
244                         index = item.row()
245                         email = item.sibling(index, 0).data()
246                         data.append({
247                             "email": email,
248                             "project": answer_user[0].text()
249                         })
250                         self.user_projects[email].append({"name": answer_user[0].text(), "
                            deadline": answer_user[1].text()})
251                     self.api.assign_to_projects(data)
252                     row_pos = self.current_projects_table.rowCount()
253                     self.current_projects_table.insertRow(row_pos)
254                     self.current_projects_table.setItem(row_pos, 0, QtWidgets.
                        QTableWidgetItem(answer_user[0]))
255                     self.current_projects_table.setItem(row_pos, 1, QtWidgets.
                        QTableWidgetItem(answer_user[1]))
256             else:
257                 self.label_log_main.setText("Server is not available!")
258             return
259
260     # Remove selected worker from project
261     # [Need to get away from instant sending to the server]
262     def del_inproject_click(self):
263         selected_rows = self.current_projects_table.selectionModel().selectedRows
            ()
264         request = []
265         for row in selected_rows:
266             name = row.sibling(row.row(), 0).data()
267             request.append({
268                 "email": self.clicked_worker_row,
269                 "project": name
270             })
271         for item in self.user_projects[self.clicked_worker_row]:
272             # Search in the "email - project" dictionary of project matching for
                the selected employee
273             # [We need the best solution to find matches]

```

```

274         if item["name"] == name and item["deadline"] == row.sibling(row.row(),
275             1).data():
276             self.user_projects[self.clicked_worker_row].remove(item)
277         list_index_rows = sorted([i.row() for i in selected_rows]) # Creating a
278             # separated list of indices of selected lines
279         while len(list_index_rows): # Deleting rows from an employee's project
280             # table
281             self.current_projects_table.removeRow(list_index_rows[-1])
282             list_index_rows.pop()
283         self.api.remove_from_projects(request)
284
285     def save_inprojects_click(self):
286         print("save_inprojects_click")
287
288     # Call the project creation dialog
289     def add_project_click(self):
290         chose_dialog = newProjectDialogWindow(self.api, self.projects_table, self.
291             new_project_rows)
292         chose_dialog.exec_()
293
294     # Project deletion
295     def del_project_click(self):
296         selected_rows = self.projects_table.selectionModel().selectedRows()
297         for row in selected_rows:
298             table = row.model()
299             index = row.row()
300             row_id = row.sibling(row.row(), 2).data()
301             try: # If the employee is already marked as deleted, remove him from the
302                 # lists for deletion
303                 self.project_rows_to_delete.remove(row_id)
304                 for x in range(0, 5):
305                     table.setData(table.index(index, x), QColor(255, 255, 255), Qt.
306                         BackgroundRole)
307             except ValueError:
308                 self.project_rows_to_delete.append(row_id)
309                 for x in range(0, 5):
310                     table.setData(table.index(index, x), QColor(255, 127, 127), Qt.
311                         BackgroundRole)
312
313     # Saving changes to the project table
314     def save_projects_click(self):
315         self.connect_status = self.api.check_connect()
316         if self.connect_status:
317             if self.project_rows_to_delete:
318                 self.api.del_projects(self.project_rows_to_delete)
319                 self.project_rows_to_delete.clear()
320             if self.new_project_rows:
321                 self.api.add_projects(self.new_project_rows)
322                 self.new_project_rows.clear()
323             for obj in self.project_rows_were_changed:
324                 # [Will need to be moved to the function if the pages appear]
325                 self.unpacked_project_rows_were_changed.append({

```

```

319         "_id": obj.sibling(obj.row(), 2).data(),
320         "name": obj.sibling(obj.row(), 0).data(),
321         "deadline": obj.sibling(obj.row(), 1).data()
322     })
323     if self.unpacked_project_rows_were_changed:
324         self.api.edit_projects(self.unpacked_project_rows_were_changed)
325         self.unpacked_project_rows_were_changed.clear()
326         self.project_rows_were_changed.clear()
327     self.update_projects()
328 else:
329     self.label_log_main.setText("Server is not available!")
330     return
331
332 # Change project data
333 # [A bad signal is used, an analog is needed]
334 def changed_cell_projects_table(self):
335     row = self.projects_table.selectionModel().selectedRows()[0]
336     if not self.project_rows_to_delete.count(row.sibling(row.row(), 2).data()):
337
338         table = row.model()
339         index = row.row()
340         self.old_data_projects_rows.append({
341             "_id": row.sibling(index, 2).data(),
342             "name": row.sibling(index, 0).data(),
343             "deadline": row.sibling(index, 1).data()
344         })
345         for x in range(0, 5):
346             table.setData(table.index(index, x), QColor(255, 253, 153), Qt.
347                 BackgroundRole)
348         self.project_rows_were_changed.append(row)
349
350 # Undo changes for employee table
351 def undo_changes_workers_table(self):
352     self.worker_rows_to_delete.clear()
353     self.unpacked_worker_rows_were_changed.clear()
354     self.workers_table.selectAll() # The selection of all elements, followed
355     by painting in white
356     for item in self.worker_rows_were_changed:
357         row_id = item.sibling(item.row(), 5).data()
358         index = item.row()
359         for old_item in self.old_data_workers_rows: # Returning old data if
360             editing
361             if old_item["_id"] == row_id:
362                 self.workers_table.setItem(index, 0, QtWidgets.QTableWidgetItem(
363                     old_item["email"]))
364                 for x in range(1, 4):
365                     self.workers_table.setItem(index, x, QtWidgets.QTableWidgetItem(
366                         old_item["name"][x - 1]))
367                 self.workers_table.setItem(index, 4, QtWidgets.QTableWidgetItem(
368                     old_item["position"]))
369             break
370     self.worker_rows_were_changed.clear()

```

```

364     self.old_data_workers_rows.clear()
365     rows = self.workers_table.selectedItems()
366     for x in rows:
367         x.setBackground(QColor(255, 255, 255))
368     for item in [item["email"] for item in self.new_worker_rows]: # Delete new
        users if added
369         self.workers_table.removeRow(self.workers_table.findItems(item, Qt.
            MatchContains)[0].row())
370     self.new_worker_rows.clear()
371
372     # Discarding changes to the project table
373     def undo_changes_projects_table(self):
374         self.project_rows_to_delete.clear()
375         self.unpacked_project_rows_were_changed.clear()
376         self.projects_table.selectAll() # The selection of all elements, followed
            by painting in white
377         for item in self.project_rows_were_changed:
378             row_id = item.sibling(item.row(), 2).data()
379             index = item.row()
380             for old_item in self.old_data_projects_rows: # Returning old data if
                editing
381                 if old_item["_id"] == row_id:
382                     self.projects_table.setItem(index, 0, QTableWidgetItem(
                        old_item["name"]))
383                     self.projects_table.setItem(index, 1, QTableWidgetItem(
                        old_item["deadline"]))
384                 break
385         self.project_rows_were_changed.clear()
386         self.old_data_projects_rows.clear()
387         rows = self.projects_table.selectedItems()
388         for x in rows:
389             x.setBackground(QColor(255, 255, 255))
390         for item in [item["name"] for item in self.new_project_rows]: # Delete new
            users if added
391             self.projects_table.removeRow(self.projects_table.findItems(item, Qt.
                MatchContains)[0].row())
392         self.new_project_rows.clear()
393
394     # User logout
395     # [Somewhere here memory leaks...]
396     def logout_click(self):
397         self._main_timer.stop()
398         self.workers_table.clear()
399         self.projects_table.clear()
400         self.current_projects_table.clear()
401         self.last_window._login_timer.start(10000)
402         self.destroy()
403         self.last_window.show()
404
405     # Displaying employee projects after selecting them
406     # [A bad signal is used, an analog is needed]
407     def show_user_projects(self):

```

```

408     self.current_projects_table.clearContents()
409     self.current_projects_table.setRowCount(0)
410     row = self.workers_table.selectionModel().selectedRows()[0]
411     self.clicked_worker_row = row.sibling(row.row(), 0).data()
412     try:
413         for project in self.user_projects[self.clicked_worker_row]:
414             row_pos = self.current_projects_table.rowCount()
415             self.current_projects_table.insertRow(row_pos)
416             self.current_projects_table.setItem(row_pos, 0, QtWidgets.
                QTableWidgetItem(project["name"]))
417             self.current_projects_table.setItem(row_pos, 1, QtWidgets.
                QTableWidgetItem(project["deadline"]))
418     except KeyError:
419         pass
420
421     def settings_click(self):
422         print("settings_click")
423
424     # Updating data for the table of workers through the server
425     def update_workers_table_click(self):
426         self.update_workers()
427
428     # Data update for project table via server
429     def update_projects_table_click(self):
430         self.update_projects()
431
432     # Page back one
433     def page_left_click(self):
434         self.workers_table_page -= int(self.size_page.text())
435         self.full_right.setEnabled(True)
436         self.page_right.setEnabled(True)
437         if not self.workers_table_page:
438             self.full_left.setEnabled(False)
439             self.page_left.setEnabled(False)
440         self.unpacked_worker_changed()
441         self.worker_rows_were_changed.clear()
442         self.update_workers()
443
444     # One page ahead
445     def page_right_click(self):
446         self.workers_table_page += int(self.size_page.text())
447         if self.api.get_users_count() - self.workers_table_page <= int(self.
            size_page.text()):
448             self.full_right.setEnabled(False)
449             self.page_right.setEnabled(False)
450         self.full_left.setEnabled(True)
451         self.page_left.setEnabled(True)
452         self.unpacked_worker_changed()
453         self.worker_rows_were_changed.clear()
454         self.update_workers()
455
456     # Page change to first

```



```

457 def full_left_click(self):
458     self.workers_table_page = 0
459     self.full_left.setEnabled(False)
460     self.page_left.setEnabled(False)
461     self.full_right.setEnabled(True)
462     self.page_right.setEnabled(True)
463     self.unpacked_worker_changed()
464     self.worker_rows_were_changed.clear()
465     self.update_workers()
466
467 # Page change to last
468 def full_right_click(self):
469     size = self.api.get_users_count()
470     self.workers_table_page = size - size % int(self.size_page.text())
471     self.full_right.setEnabled(False)
472     self.page_right.setEnabled(False)
473     self.full_left.setEnabled(True)
474     self.page_left.setEnabled(True)
475     self.unpacked_worker_changed()
476     self.worker_rows_were_changed.clear()
477     self.update_workers()
478
479 # [X]
480 def closeEvent(self, event):
481     event.accept()
482     quit()
483
484 # Class responsible for the stack window
485 class loginStackWindow(QtWidgets.QDialog, login_stack.Ui_login_dialog):
486     def __init__(self):
487         super().__init__()
488         self.setupUi(self)
489         self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint)
490         self.api = db_api.API()
491
492 # Finding or creating a new file of saved logged entries
493 # [You need to encrypt this file]
494 try:
495     with open("memory.json") as f:
496         self.data = json.load(f)
497 except IOError:
498     self.data = {"user_info": {"login": "", "pwd": ""}, "flag": False}
499     with open("memory.json") as f:
500         self.data = json.load(f)
501
502 # page_login(0) buttons events and data logic
503 self.check_save_loginpwd.setChecked(self.data["flag"])
504 self.input_login.setText(self.data["user_info"]["login"])
505 self.input_pwd.setText(self.data["user_info"]["pwd"])
506
507 # buttons events
508 self.login_button.clicked.connect(self.login_button_click)

```



```

509     self.newpwd_button.clicked.connect(self.newpwd_button_click)
510
511     # page_replace_pwd(1) buttons events
512     self.save_newpwd_button.clicked.connect(self.save_newpwd_button_click)
513     self.back_login_button.clicked.connect(self.back_login_button_click)
514
515     self._login_timer = QTimer()
516     self._login_timer.timeout.connect(self._timer_tick)
517     self._login_timer.start(10000)
518     self.connect_status = 0
519
520     def _timer_tick(self):
521         self.connect_status = self.api.check_connect()
522         if self.connect_status:
523             self.label_log_login.setText("")
524             self.button_status(True)
525         else:
526             self.label_log_login.setText("Server is not available!")
527             self.button_status(False)
528
529     def button_status(self, status):
530         self.login_button.setEnabled(status)
531         self.save_newpwd_button.setEnabled(status)
532
533     # page_login(0) login button
534     def login_button_click(self):
535         self.api.user = self.input_login.text()
536         self.api.pwd = self.input_pwd.text()
537         flag = self.check_save_loginpwd.isChecked()
538         self.connect_status = self.api.check_connect()
539         if self.connect_status:
540             answer = self.api.authorization(self.api.user, self.api.pwd)
541             if not answer["content"]["authorization"]["ok"]:
542                 self.error_loginpwd.setText("Wrong login or password!")
543             elif flag:
544                 with open("memory.json", "w") as f:
545                     f.write(json.dumps({"user_info": {"login": self.api.user, "pwd": self.
546                                     api.pwd}, "flag": flag}))
547                 self._login_timer.stop()
548                 self.miWindow = miWindow(self.api)
549                 self.miWindow.last_window = self
550                 self.destroy()
551                 self.miWindow.show()
552             else:
553                 with open("memory.json", "w") as f:
554                     f.write(json.dumps({"user_info": {"login": "", "pwd": ""}, "flag":
555                                     False}))
556                 self._login_timer.stop()
557                 self.input_login.setText("")
558                 self.input_pwd.setText("")
559                 self.miWindow = miWindow(self.api)
560                 self.miWindow.last_window = self

```

```

559         self.destroy()
560         self.miWindow.show()
561     else:
562         self.label_log_login.setText("Server is not available!")
563         return
564
565     # page_login(0) go to the user login change window
566     def newpwd_button_click(self):
567         self.login_stack.setCurrentIndex(1) # page_replace_login
568
569     # page_replace_pwd(2) back button
570     def back_login_button_click(self):
571         self.login_stack.setCurrentIndex(0) # page_login
572
573     # page_replace_pwd(2) button user pwd changes
574     def save_newpwd_button_click(self):
575         login = self.input_login_reppwd.text()
576         old_pwd = self.input_oldpwd.text()
577         new_pwd = self.input_newpwd.text()
578         self.connect_status = self.api.check_connect()
579         if self.connect_status:
580             if old_pwd != new_pwd:
581                 answer = self.api.change_password({"email": login, "old_pwd": old_pwd,
582                                                    "new_pwd": new_pwd})
583                 if answer == "Password has been changed.":
584                     self.error_reppwd.setStyleSheet("color: rgb(75, 225, 0);; font-weight
585                                                       : bold;")
586                     self.error_reppwd.setText("Password successfully changed")
587                 else:
588                     self.error_reppwd.setStyleSheet("color: rgb(255, 0, 0);; font-weight:
589                                                       bold;")
590                     self.error_reppwd.setText("Wrong login or password!")
591             else:
592                 self.error_reppwd.setStyleSheet("color: rgb(255, 0, 0);; font-weight:
593                                                   bold;")
594                 self.error_reppwd.setText("New password is the same as current!")
595         else:
596             self.label_log_reppwd.setText("Server is not available!")
597
598     # [X]
599     def closeEvent(self, event):
600         event.accept()
601         quit()
602
603     class inprojectDialogWindow(QDialog, add_inproject_dialog.
604                                Ui_add_inproject_dialog):
605         def __init__(self, list_projects):
606             super().__init__()
607             self.setupUi(self)
608
609             self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint)

```

```

606     self.answer = False
607
608     # buttons events
609     self.add_button.clicked.connect(self.add_button_click)
610     self.cancel_button.clicked.connect(self.cancel_button_click)
611
612     for project in list_projects:
613         row_pos = self.table_projects.rowCount()
614         self.table_projects.insertRow(row_pos)
615         self.table_projects.setItem(row_pos, 0, QtWidgets.QTableWidgetItem(
616             project["name"]))
617         self.table_projects.setItem(row_pos, 1, QtWidgets.QTableWidgetItem(
618             project["deadline"]))
619
620     # Confirmation of choice
621     def add_button_click(self):
622         self.answer = self.table_projects.selectedItems()
623         self.close()
624
625     # Cancel selection
626     def cancel_button_click(self):
627         self.close()
628
629     # Enable confirmation button if item is selected
630     # [Perhaps you can do better]
631     def on_table_projects_itemClicked(self, item):
632         self.add_button.setEnabled(True)
633
634     # The class responsible for adding a new project window
635     class newProjectDialogWindow(QtWidgets.QDialog, add_new_project_dialog.
636         Ui_add_new_project_dialog):
637         def __init__(self, api, table, list_new_projects):
638             super().__init__()
639             self.setupUi(self)
640             self.api = api
641             self.table = table
642             self.list = list_new_projects
643             self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint)
644
645             # buttons events
646             self.add_button.clicked.connect(self.add_button_click)
647             self.cancel_button.clicked.connect(self.cancel_button_click)
648
649             self._new_project_timer = QTimer()
650             self._new_project_timer.timeout.connect(self._timer_tick)
651             self._new_project_timer.start(10000)
652             self.connect_status = 0
653
654             def _timer_tick(self):
655                 self.connect_status = self.api.check_connect()
656                 if self.connect_status:
657                     self.label_error.setText("")

```

```

655         self.button_status(True)
656     else:
657         self.label_error.setText("Server is not available!")
658         self.button_status(False)
659
660     def button_status(self, status):
661         self.add_button.setEnabled(status)
662
663     # Add confirmation
664     def add_button_click(self):
665         self.connect_status = self.api.check_connect()
666         if self.connect_status:
667             name = self.line_project_name.text()
668             deadline = self.calendarWidget.selectedDate()
669             if not name or deadline.isNull():
670                 self.label_error.setText("Not all data is filled!")
671             elif len(name) > 65:
672                 self.label_error.setText("Project name is too big!")
673             else:
674                 date_deadline = str(deadline.day()) + "." + str(deadline.month()) + "."
675                     + str(deadline.year())
676                 data = [{"name": name, "deadline": date_deadline}]
677                 answer = self.api.add_projects(data)
678                 if answer["content"]["add_projects"][0]["ok"]:
679                     row_pos = self.table.rowCount()
680                     last_row = self.api.get_all_projects({"offset": -1, "length": row_pos
681                         + 42})[0]
682                     self.api.del_projects([last_row["_id"]])
683                     self.table.insertRow(row_pos)
684                     self.table.setItem(row_pos, 0, QtWidgets.QTableWidgetItem(name))
685                     self.table.setItem(row_pos, 1, QtWidgets.QTableWidgetItem(
686                         date_deadline))
687                     self.list.extend(data)
688                     self.table.selectRow(row_pos)
689                     row = self.table.selectedItems()
690                     for x in row:
691                         x.setBackground(QColor(122, 255, 206))
692                     self.table.scrollToBottom()
693                     self._new_project_timer.stop()
694                     self.close()
695                 else:
696                     self.label_error.setText("A project with this name already exists!")
697             else:
698                 self.label_error.setText("Server is not available!")
699         return
700
701     # Cancel add
702     def cancel_button_click(self):
703         self._new_project_timer.stop()
704         self.close()
705
706     # The class responsible for adding a new user window

```

```

704 class newUserDialogWindow(QtWidgets.QDialog, add_new_user_dialog.
    Ui_add_new_user_dialog):
705     def __init__(self, main_class, api, table, list_new_workers):
706         super().__init__()
707         self.setupUi(self)
708         self.api = api
709         self.table = table
710         self.list = list_new_workers
711         # [I need help :( )]
712         self.main_class = main_class
713         self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint)
714
715         # buttons events
716         self.add_button.clicked.connect(self.add_button_click)
717         self.cancel_button.clicked.connect(self.cancel_button_click)
718
719         self._new_user_timer = QTimer()
720         self._new_user_timer.timeout.connect(self._timer_tick)
721         self._new_user_timer.start(10000)
722         self.connect_status = 0
723
724     def _timer_tick(self):
725         self.connect_status = self.api.check_connect()
726         if self.connect_status:
727             self.label_error.setText("")
728             self.button_status(True)
729         else:
730             self.label_error.setText("Dve myasnykh katlety gril, spetsialnyy sous
                syr")
731             self.button_status(False)
732
733     def button_status(self, status):
734         self.add_button.setEnabled(status)
735
736     # Add confirmation
737     def add_button_click(self):
738         self.connect_status = self.api.check_connect()
739         if self.connect_status:
740             email = self.lineEdit_email.text()
741             surname = self.lineEdit_surname.text()
742             name = self.lineEdit_name.text()
743             patron = self.lineEdit_patron.text()
744             pos = self.lineEdit_pos.text()
745             pwd = self.lineEdit_pwd.text()
746             confpwd = self.lineEdit_confpwd.text()
747             if not (email and name and surname and patron and pos and pwd and
                confpwd):
748                 self.label_error.setText("Not all fields are filled!")
749             elif not (pwd == confpwd):
750                 self.label_error.setText("Passwords do not match!")
751             else:

```

```

752     data = [{"email": email, "pwd": pwd, "name": [surname, name, patron], "
              position": pos}]
753     answer = self.api.add_users(data)
754     if not answer["ok"]:
755         self.label_error.setText("Invalid Email View!")
756     elif answer["content"]["add_users"][0]["ok"]:
757         row_pos = self.table.rowCount()
758         last_row = self.api.get_all_users({"offset": self.api.get_users_count
              ()-1, "length": self.api.get_users_count()})[0]
759         self.api.del_users([last_row["_id"]])
760         self.table.insertRow(row_pos)
761         # [I need help :( )]
762         index_list = [str(item+1) for item in range(self.main_class.
              workers_table_page, self.main_class.workers_table_page + int(self.
              main_class.size_page.text()))]
763         self.main_class.workers_table.setVerticalHeaderLabels(index_list)
764         self.table.setItem(row_pos, 0, QtWidgets.QTableWidgetItem(email))
765         self.table.setItem(row_pos, 1, QtWidgets.QTableWidgetItem(surname))
766         self.table.setItem(row_pos, 2, QtWidgets.QTableWidgetItem(name))
767         self.table.setItem(row_pos, 3, QtWidgets.QTableWidgetItem(patron))
768         self.table.setItem(row_pos, 4, QtWidgets.QTableWidgetItem(pos))
769         self.list.extend(data)
770         self.table.selectRow(row_pos)
771         row = self.table.selectedItems()
772         for x in row:
773             x.setBackground(QColor(122, 255, 206))
774         self.table.scrollToBottom()
775         self._new_user_timer.stop()
776         self.close()
777     else:
778         self.label_error.setText("A user with this Email already exists!")
779     else:
780         self.label_error.setText("Server is not available!")
781     return
782
783     # Cancel add
784     def cancel_button_click(self):
785         self._new_user_timer.stop()
786         self.close()
787
788     def main():
789         app = QtWidgets.QApplication(sys.argv)
790         login_window = loginStackWindow()
791         login_window.show()
792         app.exec_()
793
794     if __name__ == "__main__":
795         main()

```

Список литературы

- [1] Документация Qt5 [Электронный ресурс]: для версии 5.12. URL: <https://doc.qt.io/qt-5/> (дата обращения: 01.11.2018).
- [2] Документация языка программирования Python [Электронный ресурс]: для версии 3.6. URL: <https://docs.python.org/3/> (дата обращения: 10.10.2018).
- [3] Шлее Макс. Qt 5.10. Профессиональное программирование на C++. — Санкт-Петербург, 2018. — 1072 с.
- [4] Марк Саммерфилд. Программирование на Python 3. Подробное руководство. — Символ-Плюс, 2009. — 608 с.