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

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 утерена мобильность архитектуры, которая может понадобится в случае смены VPS сервера на другой.

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 обратится 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 (рис. 1).

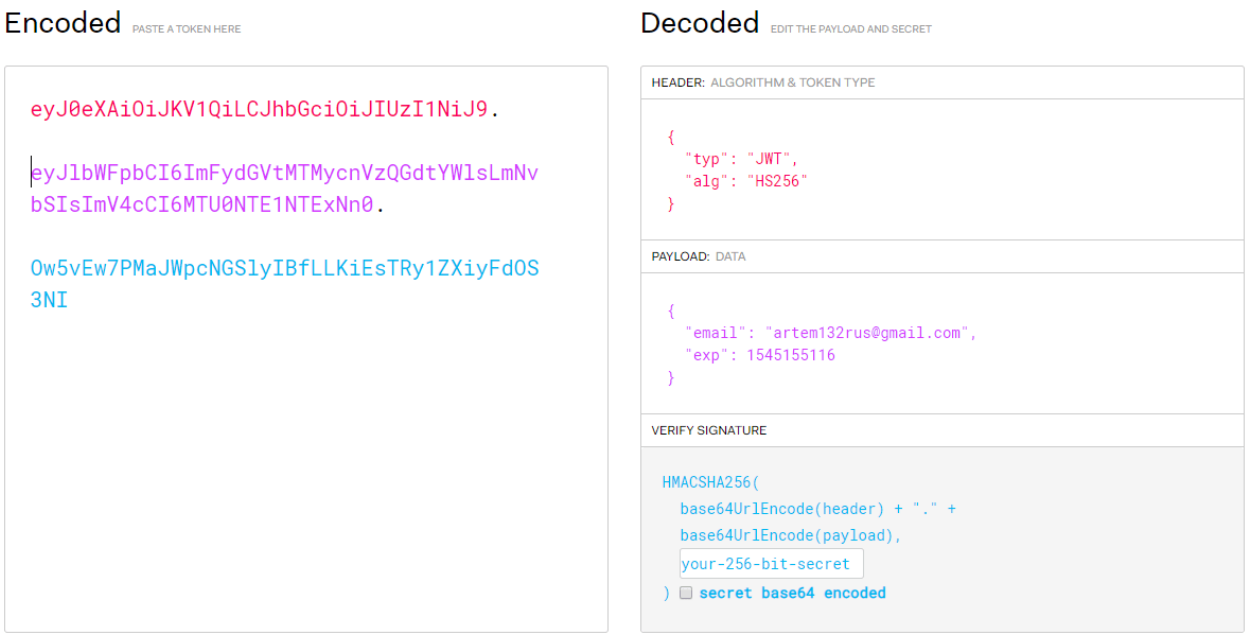

Рис. 1: Структура токена.

Первая секция (HEADER) отвечает за информацию об используемых технологиях и шифровании. 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.

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 (рис. 2). 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. 3), 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. 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 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: Пример функции 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: Пример функции 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: Пример функции 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.

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 или 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 (рис. 4), 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.

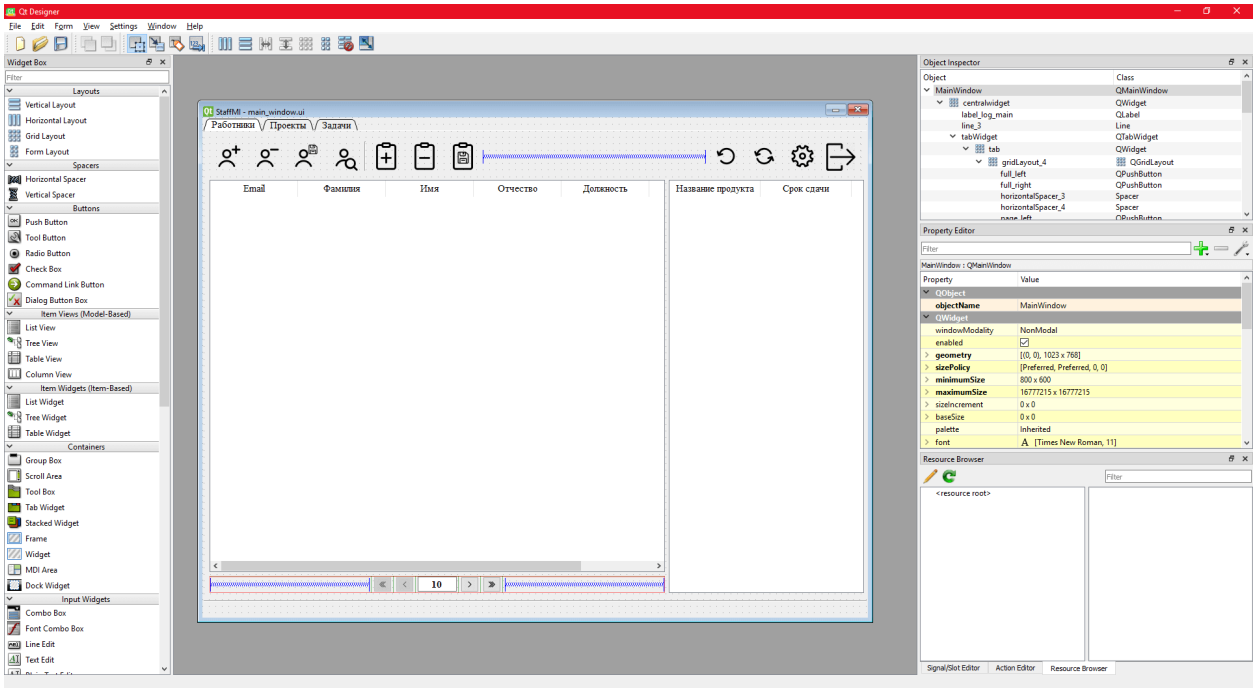

Рис. 4: Редактирование главного меню клиента в Qt Designer.

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.

Листинг 4: Функция авторизации клиента

```
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 5: Функция 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 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 (рис. 5). В нем же он может изменить пароль от своей учетной записи. После успешного прохождения этапа аутентификации, пользователь попадает на главное окно интерфейса (рис. 6), где доступны подменю работы с работниками и проектами.

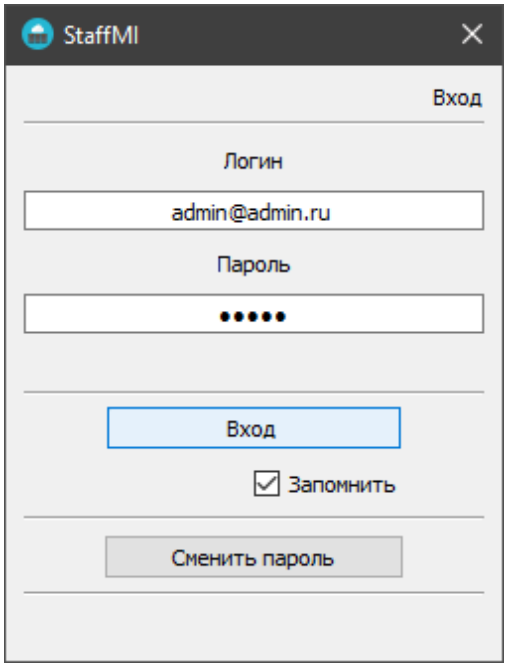

Рис. 5: Окно авторизации пользователя.

Пользователь может отредактировать данные любого проекта или работника, добавить новых (рис. 7), 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 изменений на сервер. Сделанные изменения можно отменить, если они еще не были отправлены на сервер. Для предотвращения переизбытка

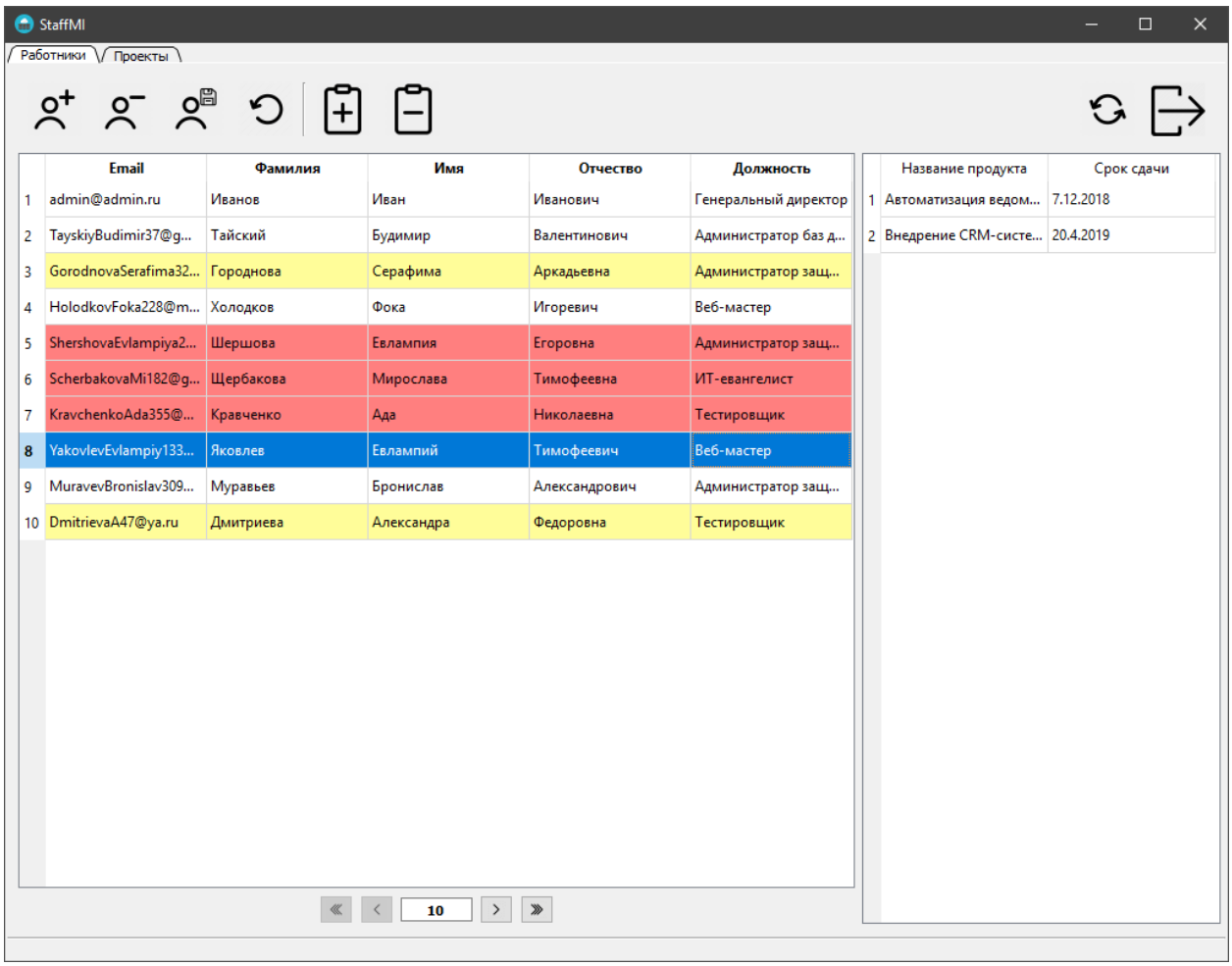

Рис. 6: Главное окно интерфейса.

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 отличатся от интерфейса на ОС Windows 10 (рис. 8).

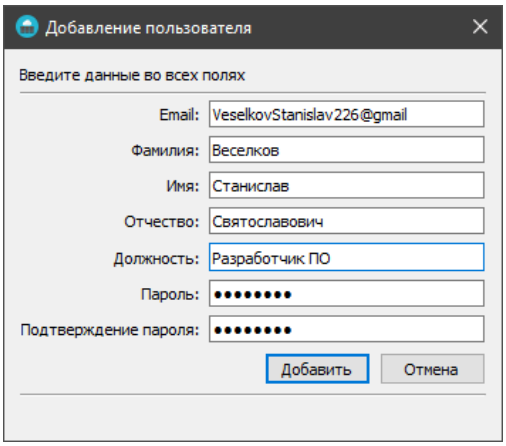

Рис. 7: Окно добавления нового пользователя.

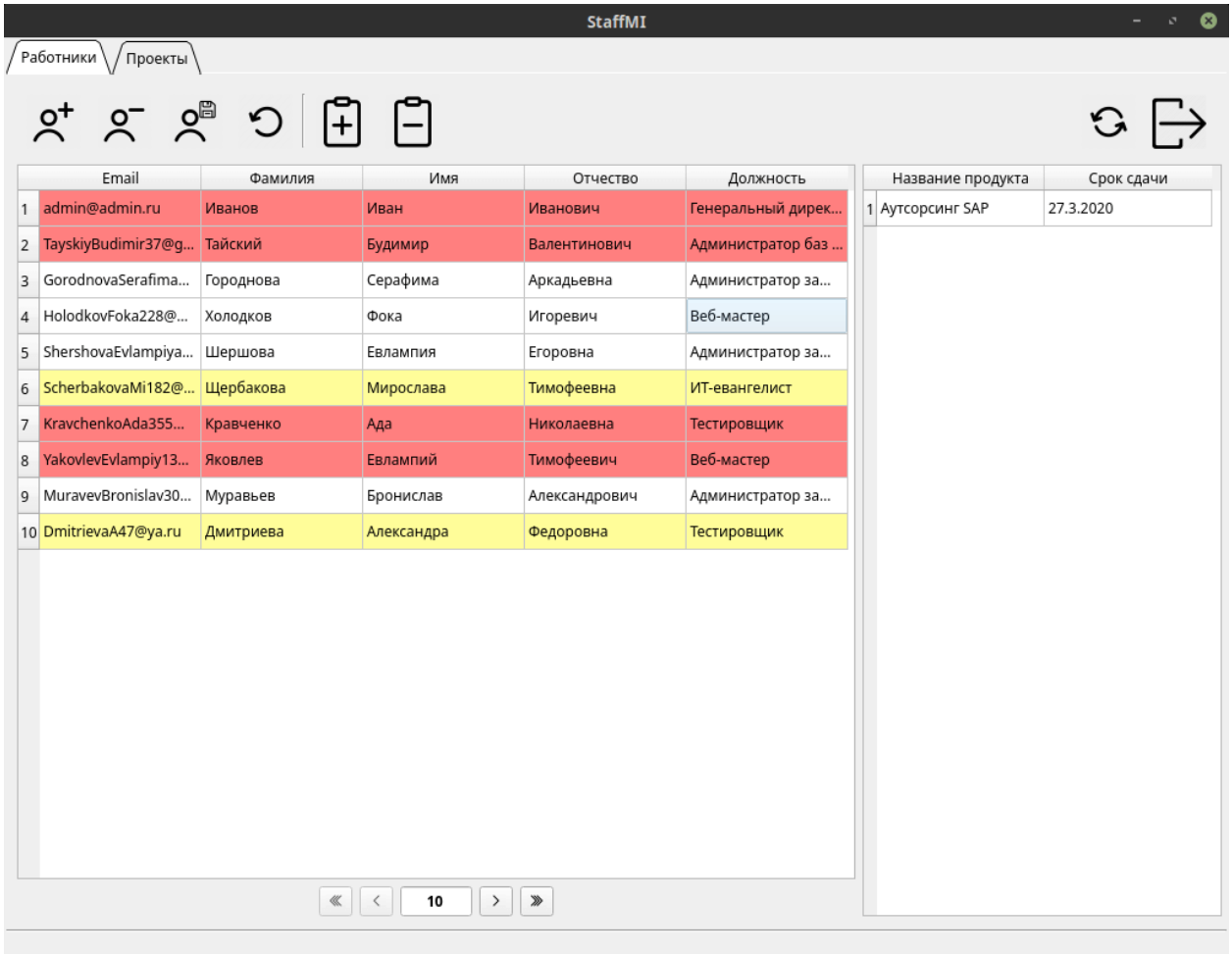

Рис. 8: Главное окно интерфейса на ОС Linux Mint.

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']['
            authorization']}), 200
64    except KeyError:
65      pass
66
67    ok, text, code = dbm.check_token(r['token'])
68    if not ok:
69      return ok, text, code
70
71    return True, requests_handler(r['requests']), 200
72
73  if __name__ == "__main__":
```

```
74    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=
         'HS256')
37     return token.decode()
38
39   def check_token(self, token):
```

```
40      try:
41        payload = jwt.decode(token.encode(), self.secret, algorithms=['HS256'
            ])
42      except jwt.ExpiredSignatureError:
43        return False, 'Token expired!', 403
44      except (jwt.DecodeError, AttributeError):
45        return False, 'Invalid token!', 403
46      return True, payload['email'], 200
47
48    # Users
49
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 = user.pop('_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":
           pwd}}, 'token': ''})
21     try:
22       response = requests.post(host, data=data).json()
23     except requests.exceptions.ConnectionError:
24       return False
25     self.token = response['content']['authorization']['content']
26     return response
27
28   def send_query(self, args):
29     data = json.dumps({"requests": args, 'token': self.token})
30     try:
31       response = requests.post(host, data=data).json()
32     except requests.exceptions.ConnectionError:
33       return False
34     try:
35       if response['error_code'] == 403:
36         self.authorization(self.user, self.pwd)
37         data = json.dumps({"requests": args, 'token': self.token})
38         response = requests.post(host, data=data).json()
39     except (KeyError, requests.exceptions.ConnectionError) as e:
```

```
40        if e == requests.exceptions.ConnectionError:
41          return False
42      if not (len(args) == 1):
43        pass
44      elif response['ok']:
45        try:
46          return tuple(response['content'].values())[0]['content']
47        except Exception:
48          pass
49      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,
       main_window)
14
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
         deletion
25     self.worker_rows_were_changed = [] # Employees whose data has been
         changed are stored here
26     self.unpacked_worker_rows_were_changed = [] # To save data from other
         pages
27     self.new_worker_rows = [] # New employees are stored here.
28     self.old_data_workers_rows = [] # Old employee data is stored here if it
          is necessary to return it.
29
30     # Local save changes to projects
31     self.project_rows_to_delete = [] # The projects selected for deletion
         are stored here
32     self.project_rows_were_changed = [] # Projects whose data has been
         modified are stored here
```

```
33      self.unpacked_project_rows_were_changed = [] # To save data from other
            pages
34      self.new_project_rows = [] # New projects are stored here
35      self.old_data_projects_rows = [] # Old project data is stored here if
            you need to return it.
36
37      self.user_projects = {} # Dictionary "email - project"
38      self.clicked_worker_row = None # Modal selected worker
39
40      self.workers_table_page = 0 # The current page of the workers table
41
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, "
            length": int(self.size_page.text())})
123       self.user_projects.clear()
124       self.workers_table.clearContents() # Table cleaning
125       self.workers_table.setRowCount(0) #
126       self.current_projects_table.clearContents() # Clearing the users
            project table
127       self.current_projects_table.setRowCount(0)
128       for worker in answer:
129         self.user_projects.update({worker["email"]: worker["projects"]})
130         row_pos = self.workers_table.rowCount()
131         self.workers_table.insertRow(row_pos)
132         self.workers_table.setItem(row_pos, 0, QtWidgets.QTableWidgetItem(
              worker["email"]))
133         for x in range(1, 4): # Parsing of the name of the employee
134           self.workers_table.setItem(row_pos, x, QtWidgets.QTableWidgetItem(
                worker["name"][x - 1]))
135         self.workers_table.setItem(row_pos, 4, QtWidgets.QTableWidgetItem(
              worker["position"]))
136         self.workers_table.setItem(row_pos, 5, QtWidgets.QTableWidgetItem(
              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,
            self.workers_table_page + int(self.size_page.text()))]
139       self.workers_table.setVerticalHeaderLabels(index_list)
140     else:
141       self.label_log_main.setText("Server is not available!")
```

```
142       return
143
144   # Update projects table
145   def update_projects(self):
146     self.connect_status = self.api.check_connect()
147     if self.connect_status:
148       answer = self.api.get_all_projects({"offset": self.workers_table_page,
              "length": 1000}) # No pages
149       self.projects_table.clearContents() # Table cleaning
150       self.projects_table.setRowCount(0)
151       for project in answer:
152         row_pos = self.projects_table.rowCount()
153         self.projects_table.insertRow(row_pos)
154         self.projects_table.setItem(row_pos, 0, QtWidgets.QTableWidgetItem(
              project["name"]))
155         self.projects_table.setItem(row_pos, 1, QtWidgets.QTableWidgetItem(
              project["deadline"]))
156         self.projects_table.setItem(row_pos, 2, QtWidgets.QTableWidgetItem(
              project["_id"]))
157     else:
158       self.label_log_main.setText("Server is not available!")
159       return
160
161   # Unpacking data from QModelIndex into edited worker lists (solving
        strange model problems)
162   def unpacked_worker_changed(self):
163     for obj in self.worker_rows_were_changed:
164       self.unpacked_worker_rows_were_changed.append({
165         "_id": obj.sibling(obj.row(), 5).data(),
166         "email": obj.sibling(obj.row(), 0).data(),
167         "name": [obj.sibling(obj.row(), 1).data(), obj.sibling(obj.row(), 2).
              data(), obj.sibling(obj.row(), 3).data()],
168         "position": obj.sibling(obj.row(), 4).data()
169       })
170
171   # Call add user dialog
172   def add_worker_click(self):
173     self.full_right_click()
174     chose_dialog = newUserDialogWindow(self, self.api, self.workers_table,
          self.new_worker_rows)
175     chose_dialog.exec_()
```

```
176
177   # Removal of workers
178   def del_worker_click(self):
179     selected_rows = self.workers_table.selectionModel().selectedRows()
180     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()
              , 1).data():
275           self.user_projects[self.clicked_worker_row].remove(item)
276     list_index_rows = sorted([i.row() for i in selected_rows]) # Creating a
          separated list of indices of selected lines
277     while len(list_index_rows): # Deleting rows from an employee"s project
          table
278       self.current_projects_table.removeRow(list_index_rows[-1])
279       list_index_rows.pop()
280     self.api.remove_from_projects(request)
281
282   def save_inprojects_click(self):
```

```
283     print("save_inprojects_click")
284
285   # Call the project creation dialog
286   def add_project_click(self):
287     chose_dialog = newProjectDialogWindow(self.api, self.projects_table,
          self.new_project_rows)
288     chose_dialog.exec_()
289
290   # Project deletion
291   def del_project_click(self):
292     selected_rows = self.projects_table.selectionModel().selectedRows()
293     for row in selected_rows:
294       table = row.model()
295       index = row.row()
296       row_id = row.sibling(row.row(), 2).data()
297       try: # If the employee is already marked as deleted, remove him from
            the lists for deletion
298         self.project_rows_to_delete.remove(row_id)
299         for x in range(0, 5):
300           table.setData(table.index(index, x), QColor(255, 255, 255), Qt.
              BackgroundRole)
301       except ValueError:
302         self.project_rows_to_delete.append(row_id)
303         for x in range(0, 5):
304           table.setData(table.index(index, x), QColor(255, 127, 127), Qt.
              BackgroundRole)
305
306   # Saving changes to the project table
307   def save_projects_click(self):
308     self.connect_status = self.api.check_connect()
309     if self.connect_status:
310       if self.project_rows_to_delete:
311         self.api.del_projects(self.project_rows_to_delete)
312         self.project_rows_to_delete.clear()
313       if self.new_project_rows:
314         self.api.add_projects(self.new_project_rows)
315         self.new_project_rows.clear()
316       for obj in self.project_rows_were_changed:
317         # [Will need to be moved to the function if the pages appear]
318         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       table = row.model()
338       index = row.row()
339       self.old_data_projects_rows.append({
340         "_id": row.sibling(index, 2).data(),
341         "name": row.sibling(index, 0).data(),
342         "deadline": row.sibling(index, 1).data()
343       })
344       for x in range(0, 5):
345         table.setData(table.index(index, x), QColor(255, 253, 153), Qt.
              BackgroundRole)
346       self.project_rows_were_changed.append(row)
347
348   # Undo changes for employee table
349   def undo_changes_workers_table(self):
350     self.worker_rows_to_delete.clear()
351     self.unpacked_worker_rows_were_changed.clear()
352     self.workers_table.selectAll() # The selection of all elements, followed
          by painting in white
353     for item in self.worker_rows_were_changed:
354       row_id = item.sibling(item.row(), 5).data()
355       index = item.row()
356       for old_item in self.old_data_workers_rows: # Returning old data if
          editing
```

```
357         if old_item["_id"] == row_id:
358           self.workers_table.setItem(index, 0, QtWidgets.QTableWidgetItem(
                old_item["email"]))
359           for x in range(1, 4):
360             self.workers_table.setItem(index, x, QtWidgets.QTableWidgetItem(
                  old_item["name"][x - 1]))
361           self.workers_table.setItem(index, 4, QtWidgets.QTableWidgetItem(
                old_item["position"]))
362           break
363     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, QtWidgets.QTableWidgetItem(
                old_item["name"]))
383           self.projects_table.setItem(index, 1, QtWidgets.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.api.pwd}, "flag": flag}))
```

```
546         self._login_timer.stop()
547         self.miWindow = miWindow(self.api)
548         self.miWindow.last_window = self
549         self.destroy()
550         self.miWindow.show()
551       else:
552         with open("memory.json", "w") as f:
553           f.write(json.dumps({"user_info": {"login": "", "pwd": ""}, "flag":
                False}))
554         self._login_timer.stop()
555         self.input_login.setText("")
556         self.input_pwd.setText("")
557         self.miWindow = miWindow(self.api)
558         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, "new_pwd": new_pwd})
582         if answer == "Password has been changed.":
583           self.error_reppwd.setStyleSheet("color: rgb(75, 225, 0);; font-
              weight: bold;")
```

```
584             self.error_reppwd.setText("Password successfully changed")
585           else:
586             self.error_reppwd.setStyleSheet("color: rgb(255, 0, 0);; font-
                  weight: bold;")
587             self.error_reppwd.setText("Wrong login or password!")
588         else:
589           self.error_reppwd.setStyleSheet("color: rgb(255, 0, 0);; font-weight:
                bold;")
590           self.error_reppwd.setText("New password is the same as current!")
591       else:
592         self.label_log_reppwd.setText("Server is not available!")
593
594     # [X]
595     def closeEvent(self, event):
596       event.accept()
597       quit()
598
599
600 class inprojectDialogWindow(QtWidgets.QDialog, add_inproject_dialog.
      Ui_add_inproject_dialog):
601     def __init__(self, list_projects):
602       super().__init__()
603       self.setupUi(self)
604
605       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(
            project["name"]))
616         self.table_projects.setItem(row_pos, 1, QtWidgets.QTableWidgetItem(
            project["deadline"]))
617
618     # Confirmation of choice
```

```
619   def add_button_click(self):
620     self.answer = self.table_projects.selectedItems()
621     self.close()
622
623   # Cancel selection
624   def cancel_button_click(self):
625     self.close()
626
627   # Enable confirmation button if item is selected
628   # [Perhaps you can do better]
629   def on_table_projects_itemClicked(self, item):
630     self.add_button.setEnabled(True)
631
632 # The class responsible for adding a new project window
633 class newProjectDialogWindow(QtWidgets.QDialog, add_new_project_dialog.
      Ui_add_new_project_dialog):
634   def __init__(self, api, table, list_new_projects):
635     super().__init__()
636     self.setupUi(self)
637     self.api = api
638     self.table = table
639     self.list = list_new_projects
640     self.setWindowFlags(self.windowFlags() & ~Qt.WindowContextHelpButtonHint
          )
641
642     # buttons events
643     self.add_button.clicked.connect(self.add_button_click)
644     self.cancel_button.clicked.connect(self.cancel_button_click)
645
646     self._new_project_timer = QTimer()
647     self._new_project_timer.timeout.connect(self._timer_tick)
648     self._new_project_timer.start(10000)
649     self.connect_status = 0
650
651   def _timer_tick(self):
652     self.connect_status = self.api.check_connect()
653     if self.connect_status:
654       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()) +
              "." + str(deadline.year())
675         data = [{"name": name, "deadline": date_deadline}]
676         answer = self.api.add_projects(data)
677         if answer["content"]["add_projects"][0]["ok"]:
678           row_pos = self.table.rowCount()
679           last_row = self.api.get_all_projects({"offset": -1, "length":
                row_pos+42})[0]
680           self.api.del_projects([last_row["_id"]])
681           self.table.insertRow(row_pos)
682           self.table.setItem(row_pos, 0, QtWidgets.QTableWidgetItem(name))
683           self.table.setItem(row_pos, 1, QtWidgets.QTableWidgetItem(
                date_deadline))
684           self.list.extend(data)
685           self.table.selectRow(row_pos)
686           row = self.table.selectedItems()
687           for x in row:
688             x.setBackground(QColor(122, 255, 206))
689           self.table.scrollToBottom()
690           self._new_project_timer.stop()
691           self.close()
692         else:
693           self.label_error.setText("A project with this name already exists!"
                )
694     else:
```

```
695         self.label_error.setText("Server is not available!")
696         return
697 
698     # Cancel add
699     def cancel_button_click(self):
700       self._new_project_timer.stop()
701       self.close()
702 
703 # 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, spetsiaalnyy 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 програмирования 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.