УДК 004.9

ВНЕДРЕНИЕ АВТОМАТИЗАЦИИ СОЗДАНИЯ УЧЕТНЫХ ЗАПИСЕЙ ДЛЯ VOIP ТЕЛЕФОНИИ (NAUMEN SOFTPHONE)

Станислав Олегович Чиркин
ассистент
stas.chirkin@bk.ru
Владислав Олегович Чиркин
ассистент
abracadabr66@mail.ru
Даниил Михайлович Солопов
студент
danil.solopov01@mail.ru
Мичуринский государственный аграрный университет
г. Мичуринск, Россия

Аннотация. Контактные центры для осуществления своей деятельности используют ip телефонию. Из-за непопулярности на данных предприятиях наблюдается частая сменна состава операторов и создания для каждого учетной записи занимает время, для разгрузки административного состава предприятия, будут рассмотрены средства автоматизации создания учетных записей.

Ключевые слова: информационные технологии, автоматизация, создания средств автоматизации.

Для начала определим минимальные требования для создания УЗ и в каком формате они будут записываться. Минимальный набор данных для создания УЗ в naumen softphone является полное ФИО. Для выбора формата лучше ориентироваться на стандартный пакет офисного ПО, например, табличный процессор (при использовании текстового редактора мы бы столкнулись с проблемой отделения данных друг от друга.) exel или libraoffice, но точнее нам интересен формат «.xls» который может быть создан в данных табличных процессорах. Его достоинство для нас — это структура данных где каждый элементы расположен в ячейке, что упростит работу по выводу данных из файла.

Теперь нам требуется определить каким путем мы будем доставлять файлы до нашего автоматизированного ПО. Мы можем посылать «.xls» по почте или воспользоваться itsm-системы, в нашем случаи мы имеем готовый itsm-решение такое как jria service desk. Он позволит нам видеть эффективность нашего автоматизированного ПО и структурирует каждый запрос, что позволит смотреть кто сделал заявку на создание УЗ и когда.

Теперь перейдём к технической части. За основу нашего ПО был взять язык программирования python т.к. имеет нужные нам библиотеки для чтения «.xls»-файлов и пакет для обработки и анализа данных pandas.

Так как мы используем itsm-систему то мы можем добавить специальный статус, по которому наше ПО сможет самостоятельно определять запросы, которые предназначены ему, чтобы парсить запросы, мы просто проста будем просматривать лист действующих запросов и сохраняем их как json – т.к. данные внутри данной страницы содержат нужные нам данные. Скрипт сохранения данных в json предоставлен на рисунке 1, предварительно создаем json пустой страницы который мы будем использовать в нашем скрипте.

```python
def json_save(new_link_json, headers, file_json):
    try:
        profs_source = session.get(new_link_json, headers=headers).json()
        with open(file_json, 'w', encoding='utf8') as json_file:
            json.dump(profs_source, json_file, indent=4,
                      sort_keys=True, ensure_ascii=False)

        return profs_source
    except:
        logger.warning(
            f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: JSON ERROR объекты не найдены! Ссылка: {new_link_json}')
        profs_source = 'JSON_ERROR'
        return profs_source
```

Рисунок 1 – Сохранения данных в json формате.

Далия на требуется обработать данные в создана нами json-файл, тут все зависит от потребности и структуре организации. Нам требуется достать из следующие данные: номер заявки в jira, ссылку на сам ресурс, «.xls»-файл с ФИО, статус заявки и её исполнителя.

Для обработки файла нам требуется, чтобы в заявки был файл, который соответствует требованиям для этого делаем следующий скрипт (рисунок 2).

```python
elif i['mimeType'] == 'application/vnd.ms-excel':
    response = session.get(attachment_link, headers=headers)
    with open(attachment_link_name, 'wb') as file:
        file.write(response.content)
    logger.info(
        f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: {attachment_link_name} загружен!')
    new_file_name_jira, df_new = senenium_pandas_naumen(
        attachment_link_name, naumen_url,
        naumen_login, naumen_password,
        pl_ploshadka, ploshadka_dolzhnost, pl_link, auth_jira, pl_key,
        dolzhnost,
        attachment_id, logger, driver, comment)
    if new_file_name_jira == 'error_file_none_data':
        error_check_file_count += 1
        auth_jira.delete_attachment(attachment_id)
        if error_check_file_count == check_file_count:
            logger.error(
                f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: Файлы не заполнен, перевожу заявку в "Ожидании ответа заявителя"!')
            vopros_zayavitelyu = (f' - Шаблон для создания УЗ Naumen'
                                  f'\nФайл *не заполнен*. Во вложении я Вам приложил "Корретный файл Naumen" который *нужно заполнить (зеленые поля)* и приложить в заявку.'
                                  f'\nВ дальнейшем просьба использовать только этот шаблон.'
                                  f'\n----'
                                  f'\n_Ответ был предоставлен автоматизированной системой_ *Shadow Jira*')
```

Рисунок 2 – Обработка вложенного.

Теперь нам требуется проверить содержимое файла, чтобы избежать возможных ошибок для этого требуется выполнить проверку, нам требуется проверить что сточки имеют заполненные данные с именем и фамилией, скрипт данной проверки указан на рисунке 3.

```
global get_url, error_new_login, sotrudnik_on, s, df_new_comment
cols = [0, 1, 2]  # выбираем из excel файла первые 3 столбца для Pandas
df = pd.read_excel( io: f'./{attachment_link_name}', usecols=cols)  # читаем excel файл
df["Логин"] = ""  # добавляем столбец Логин
df["Пароль"] = ""  # Добавляем столбец Пароль
if df.columns.values[0] == 'Фамилия' and df.columns.values[1] == 'Имя' and df.columns.values[2] == 'Отчество':
    check = 0
    for s in range(df.Фамилия.size):
        if df.Фамилия.size == 1:  # провека пустого файла
            error_file_none_data = 'error_file_none_data'  # файл пустой
            df_new_comment = ''
            return error_file_none_data, df_new_comment
        else:
            if s == 0:
                pass
            else:
                df.fillna(value=0,
                          inplace=True)  # заменяем все пустые клетки на 0
                if df.Фамилия[s] == 0 and df.Имя[s] == 0 and df.Отчество[s] == 0:
                    logger.warning(
                        f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: Пустая строка №{s}!')  # пропускаем все
                    # пустые строки
                elif (df.Фамилия[s] != 0 and df.Имя[s] == 0) or (df.Фамилия[s] == 0 and df.Имя[s] != 0):
                    df.Фамилия[s] = 0
                    df.Имя[s] = 0
                    logger.warning(
                        f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: Пустая строка №{s}!')
                    pass
```

Рисунок 3 – Проверка строчек имени и фамилии на соответствие.

Теперь вызываем функцию создания УЗ с помощью следующего скрипта (рисунок 4) в случае повторения логина будет вызван функция проверки УЗ.

```
                                                                                    driver)  # вызываем функцию создания УЗ
try:
    error_login = driver.find_element(By.XPATH,
                                      '//*[@id="login_errors"]/span/a')  # Если УЗ существует, то сайт даст красную строку
    error_new_login = error_login.accessible_name
    logger.info(
        f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: УЗ уже существует! Новый логин: {error_new_login}')  # пишем логи
except:
    error_new_login = 'none'
check_uz_error = False
if s == 1:
    sotrudnik_on = True  # не ставить галочку в поиске
else:
    sotrudnik_on = False  # ставить галочку в поиске
driver.get(pms_start_link)  # открываем начальную страницу в pms
time.sleep(1)
search_uz_naumen(login_create_pms_eng, check_uz_error, logger, sotrudnik_on,
                 driver)  # вызываем функцию поиска УЗ
```

Рисунок 4 – Инициализация создания УЗ и проверки.

Функция создания уз состоит из нескольких частей первая авторизация на стайте предварительно требуется создать УЗ с админ доступом что бы защитить УЗ в скрипте мы закодируем пароль с помощью base64 и получившийся результат вставим в скрипт и будем просто его декодировать, скрипт авторизации указана на рисунке 5.

```
if s == 1 and check_uz_error is True and error_new_login == '1_step':  # исключение
    password_create_pms = ''
    driver.get(naumen_url)  # переходим на главную страницу PMS
    time.sleep(1)  # спим
    email_input = driver.find_element(By.NAME, "login")  # ищем поле логин
    email_input.clear()  # очищаем
    email_input.send_keys(naumen_login)  # вводим логин
    password_input = driver.find_element(By.NAME, "password")  # ищем поле пароль
    password_input.clear()  # очищаем
    password_input.send_keys(naumen_password)  # вводим пароль
    password_input.send_keys(Keys.ENTER)  # жмем ENTER
    time.sleep(5)
```

Рисунок 5 – Авторизация в naumen.

Далее открываем ссылку для создания УЗ, ссылки требуется предварительно добавить в файл где будут указаны ссылки на создания УЗ под каждую площадку (это особенность создания УЗ в naumen), скрипт создания указан на рисунке 6.

```
link_naumen_ploshadka = dict_ploshadka_in_link[ploshadka_dolzhnost]  # ищем нужную ссылку с файла config_naumen
driver.get(link_naumen_ploshadka)  # переходим по этой ссылке
time.sleep(2)  # спим
fio_imenit_padezh = ''
if df.Отчество[s] != 0:  # Если отчество есть
    fio_imenit_padezh = df.Фамилия[s] + ' ' + df.Имя[s] + ' ' + df.Отчество[s]
else:  # если отчества нет
    fio_imenit_padezh = df.Фамилия[s] + ' ' + df.Имя[s]
fio_imenit_padezh = fio_imenit_padezh.split()  # разбиваем по пробелам
driver.find_element(By.NAME, 'lastName').send_keys(fio_imenit_padezh[0])  # вставляем в нужное поле фамилию
driver.find_element(By.NAME, 'firstName').send_keys(fio_imenit_padezh[1])  # вставляем в нужное поле имя
if df.Отчество[s] != 0:  # если отчество есть
    driver.find_element(By.NAME, 'middleName').send_keys(fio_imenit_padezh[2])  # то вставляем его в нужное поле
if error_new_login == 'none' or error_new_login == '1_step':  # исключение
    if df.Отчество[s] != 0:  # если отчество есть
        login_create_pms_rus = fio_imenit_padezh[1][0] + '.' + fio_imenit_padezh[2][0] + '.' + fio_imenit_padezh[
            0]  # запонимаем для создания логина
    else:
        login_create_pms_rus = fio_imenit_padezh[1][0] + '.' + fio_imenit_padezh[
            0]  # запоминаем для создания логина
```

Рисунок 6 – Создание УЗ в naumen.

И последняя часть скрипта создания логин и пароля для новых УЗ, а также добавления ей должности, которая указана в заявке (рисунок 7).

```
driver.find_element(By.LINK_TEXT, dolzhnost).click()  # кликаем на нужную должность (она же роль)
if dolzhnost == 'Супервизор':  # Supervisors_remoteView - это профиль для роли СВ или админ состава
    try:
        driver.find_element(By.CLASS_NAME, 'selectBox-arrow').click()  # нажимаем на выпадающий список профилей
        time.sleep(1)
        driver.find_element(By.LINK_TEXT, 'Supervisors_remoteView').click()  # нажимаем на нужный
    except:
        pass
time.sleep(1)  # спим
login_create_pms_eng = transliterate(login_create_pms_rus)  # вызываем функцию генерации логина
driver.find_element(By.ID, 'login').send_keys(login_create_pms_eng)  # записываем логиин в нужное поле
password_create_pms = random.randint( a: 1000,  b: 9999)  # генерируем пароль
driver.find_element(By.ID, 'password').clear()  # очищаем поле пароля
driver.find_element(By.ID, 'password_confirm').clear()  # очищаем поле подтверждения пароля
driver.find_element(By.ID, 'password').send_keys(password_create_pms)  # вставляем пароль в поле Пароль
driver.find_element(By.ID, 'password_confirm').send_keys(
    password_create_pms)  # ставляем пароль в поле Подтвержение пароля
time.sleep(1)  # спим
driver.find_element(By.ID, 'redirect').click()  # сохраняем уз
df.loc[s, 'Логин'] = login_create_pms_eng  # записываем в память программы логин
df.loc[s, 'Пароль'] = int(password_create_pms)  # записываем в память программы пароль
logger.info(f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: УЗ создана: {login_create_pms_eng}')  # пишем логи
new_file_name_jira = f'./attachment/Готовый{pl_key}.xlsx'  # запоминаем название файла
time.sleep(2)  # спим
return login_create_pms_eng, password_create_pms, fio_imenit_padezh
```

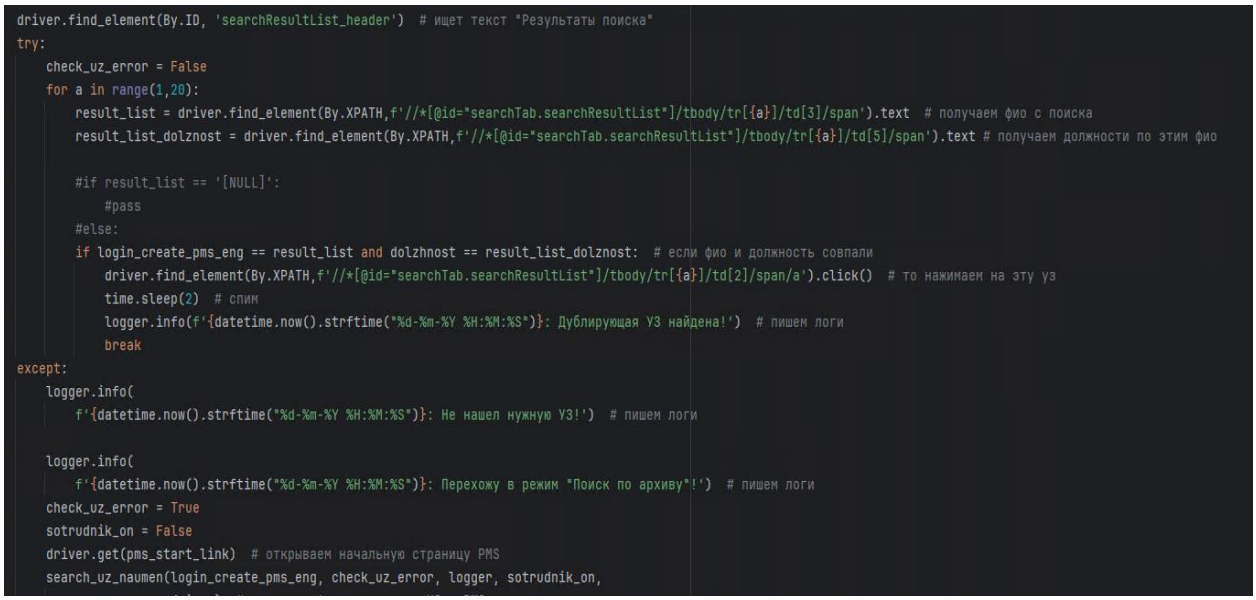
Рисунок 7 – Создания логина и пароля.

В случае если форма выдаст ошибку что такой логин уже существует, то требуется провернуть скрипт проверки. Для этого запускаем поиск УЗ на веб-интерфейсе naumen (рисунок 8).

```
def search_uz_naumen(login_create_pms_eng, check_uz_error, logger, sotrudnik_on, driver):  # ищем уз в PMS
    logger.info(f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: Ищу УЗ: {login_create_pms_eng} в PMS')  # пишем логи
    time.sleep(3)  # спим
    for attempt in range(3):
        try:
            driver.find_element(By.ID, 'SystemSearch').click()  # нажимаем на поиск
            driver.find_element(By.ID, 'searchString').clear()  # очищаем поле с логином
            driver.find_element(By.ID, 'searchString').send_keys(login_create_pms_eng)  # вставляем нужный логин для поиска
            if sotrudnik_on:  # если нужно выбрать графу Сотрудник, то получаем True
                driver.find_element(By.ID, 'searchTypeSelect_box').click()  # нажимаем на выпадающий список
                time.sleep(1)  # спим
                driver.find_element(By.LINK_TEXT, 'Сотрудник').click()  # находим слово Сотрудник и кликаем
            if check_uz_error:  # Если True, то ставим галочку для поиска в архиве
                driver.find_element(By.ID, "searchInArchive").click()  # ставим галочку
            time.sleep(1)
            button = driver.find_element(By.ID, "doSearch")  # ищем кнопку найти
            button.click()  # нажимаем на нее
            break
        except Exception as e:
            print(f"ошибка {e}")
    else:
        print('не удалось начать поиск')
```

Рисунок 8 – Инициализация поиска.

Далее начинаем парсить логины в списке поиска для этого будем искать УЗ с совпадающим ФИО, должностью и площадкой, в случае если УЗ не найден будет произведен поиск по заблокированным УЗ. Если и там совпадающих УЗ не будет, то к логину просто добавится цифра. Скрипт просмотра поиска указан на рисунке 9.



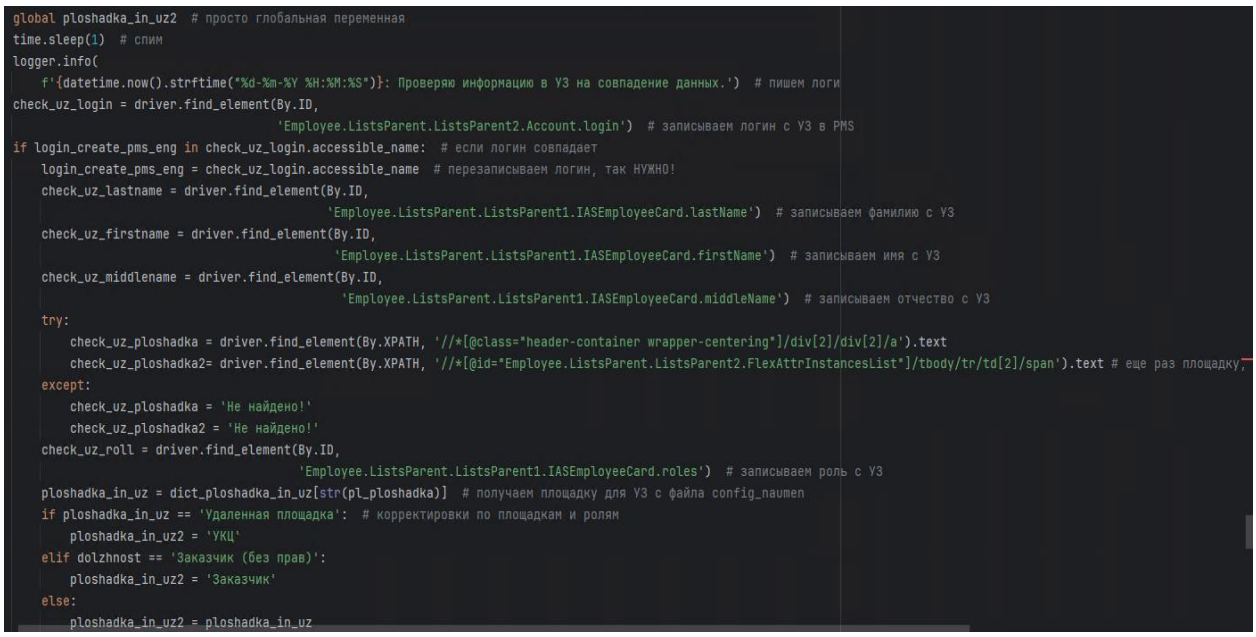
```
driver.find_element(By.ID, 'searchResultList_header')  # ищет текст "Результаты поиска"
try:
    check_uz_error = False
    for a in range(1,20):
        result_list = driver.find_element(By.XPATH,f'//*[@id="searchTab.searchResultList"]/tbody/tr[{a}]/td[3]/span').text  # получаем фио с поиска
        result_list_dolznost = driver.find_element(By.XPATH,f'//*[@id="searchTab.searchResultList"]/tbody/tr[{a}]/td[5]/span').text # получаем должности по этим фио

        #if result_list == '[NULL]':
            #pass
        #else:
        if login_create_pms_eng == result_list and dolzhnost == result_list_dolznost:  # если фио и должность совпали
            driver.find_element(By.XPATH,f'//*[@id="searchTab.searchResultList"]/tbody/tr[{a}]/td[2]/span/a').click()  # то нажимаем на эту уз
            time.sleep(2)  # спим
            logger.info(f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: Дублирующая УЗ найдена!')  # пишем логи
            break
except:
    logger.info(
        f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: Не нашел нужную УЗ!')  # пишем логи

    logger.info(
        f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: Перехожу в режим "Поиск по архиву"!')  # пишем логи
    check_uz_error = True
    sotrudnik_on = False
    driver.get(pms_start_link)  # открываем начальную страницу PMS
    search_uz_naumen(login_create_pms_eng, check_uz_error, logger, sotrudnik_on,
```

Рисунок 9 – Проверка УЗ в поиске.

В случае если повторяющееся УЗ найдена, то начинаем процесс корректировки данных УЗ, для этого воспользуемся скриптом ниже (рисунок 10).



```
global ploshadka_in_uz2  # просто глобальная переменная
time.sleep(1)  # спим
logger.info(
    f'{datetime.now().strftime("%d-%m-%Y %H:%M:%S")}: Проверяю информацию в УЗ на совпадение данных.')  # пишем логи
check_uz_login = driver.find_element(By.ID,
                                     'Employee.ListsParent.ListsParent2.Account.login')  # записываем логин с УЗ в PMS
if login_create_pms_eng in check_uz_login.accessible_name:  # если логин совпадает
    login_create_pms_eng = check_uz_login.accessible_name  # перезаписываем логин, так НУЖНО!
    check_uz_lastname = driver.find_element(By.ID,
                                            'Employee.ListsParent.ListsParent1.IASEmployeeCard.lastName')  # записываем фамилию с УЗ
    check_uz_firstname = driver.find_element(By.ID,
                                             'Employee.ListsParent.ListsParent1.IASEmployeeCard.firstName')  # записываем имя с УЗ
    check_uz_middlename = driver.find_element(By.ID,
                                              'Employee.ListsParent.ListsParent1.IASEmployeeCard.middleName')  # записываем отчество с УЗ
    try:
        check_uz_ploshadka = driver.find_element(By.XPATH, '//*[@class="header-container wrapper-centering"]/div[2]/div[2]/a').text
        check_uz_ploshadka2= driver.find_element(By.XPATH, '//*[@id="Employee.ListsParent.ListsParent2.FlexAttrInstancesList"]/tbody/tr/td[2]/span').text # еще раз площадку,
    except:
        check_uz_ploshadka = 'Не найдено!'
        check_uz_ploshadka2 = 'Не найдено!'
    check_uz_roll = driver.find_element(By.ID,
                                        'Employee.ListsParent.ListsParent1.IASEmployeeCard.roles')  # записываем роль с УЗ
    ploshadka_in_uz = dict_ploshadka_in_uz[str(pl_ploshadka)]  # получаем площадку для УЗ с файла config_naumen
    if ploshadka_in_uz == 'Удаленная площадка':  # корректировки по площадкам и ролям
        ploshadka_in_uz2 = 'УКЦ'
    elif dolzhnost == 'Заказчик (без прав)':
        ploshadka_in_uz2 = 'Заказчик'
    else:
        ploshadka_in_uz2 = ploshadka_in_uz
```

Рисунок 10 – Корректировка данных УЗ.

После практического внедрения и месяца отработки мы можем увидеть показатели эффективности. За месяц наш скрипт отработал 140 заявок на создания УЗ отчет показан на рисунок 11.

Дата	БОТ Shadow Jira
01.10.2025	8
02.10.2025	9
03.10.2025	2
04.10.2025	0
05.10.2025	0
06.10.2025	17
07.10.2025	19
08.10.2025	6
09.10.2025	4
10.10.2025	1
11.10.2025	0
12.10.2025	0
13.10.2025	10
14.10.2025	7
15.10.2025	8
16.10.2025	4
17.10.2025	2
18.10.2025	0
19.10.2025	0
20.10.2025	7
21.10.2025	11
22.10.2025	2
23.10.2025	4
24.10.2025	3
25.10.2025	2
26.10.2025	0
27.10.2025	5
28.10.2025	7
29.10.2025	2
30.10.2025	0
31.10.2025	0
	140

Рисунок 11 – Выгруженный отчет из jira service desk.

Список литературы:

1. Антао Т. Сверхбыстрый Python: руководство / перевод с английского А. Ю. Гинько. Москва: ДМК Пресс, 2023. 370 с. ISBN 978-5-93700-226-6.

2. Сергеева, О. А. Программирование на Python: учебно-методическое пособие / Кемерово: КемГУ, 2024. 157 с. ISBN 978-5-8353-3123-9.

3. Давыдова Н. А., Боровская Е. В. Программирование: учебное пособие / 5-е изд. (эл.). Москва: Лаборатория знаний. 2025. 241 с. ISBN 978-5-93208-831-9.

4. pandas 2.3.3 documentation // Официальный сайт – URL: https://pandas.pydata.org/docs/

5. JIRA Service Desk Documentation // Atlassian Support – URL: https://confluence.atlassian.com/servicedesk011/jira-service-desk-documentation-445188602.html

UDC 004.9

IMPLEMENTATION OF AUTOMATED ACCOUNT CREATION FOR VOIP TELEPHONY (NAUMEN SOFTPHONE)

Stanislav Ol. Chirkin
assistant
stas.chirkin@bk.ru
Vladislav Ol. Chirkin
assistant
abracadabr66@mail.ru
Daniil M. Solopov
student
danil.solopov01@mail.ru
Michurinsk State Agrarian University
Michurinsk, Russia

Annotation. The contact centers use IP telephony to carry out their activities. Due to the unpopularity of these enterprises, there is a frequent shift in the composition of operators and the creation of an account for each takes time, to unload the administrative staff of the enterprise, automation tools for creating accounts will be considered.

Keywords: information technology, automation, creation of automation tools.

Статья поступила в редакцию 24.10.2025; одобрена после рецензирования 20.12.2025; принята к публикации 29.12.2025.
The article was submitted 24.10.2025; approved after reviewing 20.12.2025; accepted for publication 29.12.2025.