**GeekBrains**

**Специальность – программист Python, цифровые профессии**

**Тема дипломного проекта: Разработка информационной**

**системы контроля трудовых ресурсов проекта**

**Студент: Шаверин Никита Владимирович**

Томск - 2023 г.

СОДЕРЖАНИЕ

[1. Введение 4](#_Toc149679235)

[1.1. Актуальность проекта 4](#_Toc149679236)

[1.2. Цели и задачи проекта, план работы 5](#_Toc149679237)

[1.3. Команда проекта 7](#_Toc149679238)

[2. Теоретическая часть 9](#_Toc149679239)

[2.1. Анализ существующего рынка программного обеспечения для реализации контроля трудовых ресурсов 9](#_Toc149679240)

[2.2. Определение требований к информационной системе 16](#_Toc149679241)

[2.3. Анализ архитектурных решений и алгоритмов для реализации информационной системы 18](#_Toc149679242)

[2.4. Изучение инструментов для создания информационной системы, определение основного стека технологий 21](#_Toc149679243)

[3. Практическая часть 23](#_Toc149679244)

[3.1. Архитектура информационной системы «Совместный труд» 23](#_Toc149679245)

[3.2. Разработка структуры базы данных информационной системы 26](#_Toc149679246)

[3.3. Разработка алгоритма модуля Telegram-bot 32](#_Toc149679247)

[3.4. Механизм запросов и ответов модуля Telegram-bot 39](#_Toc149679248)

[3.5. Шаблоны отчётов 40](#_Toc149679249)

[3.5.1. Отчёт типа list 44](#_Toc149679250)

[3.5.2. Отчёт типа status 46](#_Toc149679251)

[3.5.3. Отчёт типа line 47](#_Toc149679252)

[3.5.4. Отчёты бригадира, по оборудованию 48](#_Toc149679253)

[3.6. Обмен сообщениями между пользователями системы 49](#_Toc149679254)

[3.7. Автоматическая рассылка списка задач 52](#_Toc149679255)

[4. Заключение 54](#_Toc149679256)

[4.1. Выводы, полученные во время анализа теоретической базы и практического исследования 54](#_Toc149679257)

[4.2. Оценка проекта, описание его результатов 54](#_Toc149679258)

[4.3. Общий итог работы, её практическая значимость 55](#_Toc149679259)

[4.4. Предложения по совершенствованию информационной системы «Совместный труд» 55](#_Toc149679260)

[5. Список используемой литературы 57](#_Toc149679261)

[6. Приложения 58](#_Toc149679262)

[6.1. Программный код модуля Telegram-bot, модуля управления базой данных 58](#_Toc149679263)

[6.2. Скриншоты работы веб-версии информационной системы 189](#_Toc149679264)

[6.3. Благодарственное письмо 191](#_Toc149679266)

[6.4. Рекомендательное письмо 192](#_Toc149679268)

Дипломный проект находится по адресу: <https://github.com/ShaoYoung/Diploma>

1. Введение

1.1. Актуальность проекта

4 мая 1935 года на выпуске академиков Красной Армии в Кремлёвском дворце руководитель СССР И.В. Сталин произнёс фразу «Самый ценный капитал – это люди. Кадры решают всё!» [1]. Не смотря на неоднозначное отношение современников к личности и деятельности И.В. Сталина, многое из его тогдашнего выступления было положено в основу индустриализации и стремительного развития страны.

Действительно, трудовые ресурсы представляют собой важный фактор производства, рациональное использование которого обеспечивает не только повышение уровня производства и его экономической эффективности, но и качественное развитие всей социальной системы. Рациональное использование трудовых ресурсов предприятия влияет на результаты производственно-хозяйственной деятельности, выполнение бизнес-плана и на динамику выполнения плана производства.

Эффективное использование трудовых ресурсов может быть определено в результате оценки наличных трудовых ресурсов (анализа численности работающих, обеспечение предприятия рабочей силой, состава и структуры кадров, текучести кадров и т.д.) и разработки программы по их развитию. Цель анализа трудовых ресурсов состоит в том, чтобы разработать рекомендации по повышению эффективности использования трудовых ресурсов и обосновать планы по труду. Анализ трудовых ресурсов является важным элементом механизма управления предприятием и может применяться для обоснования принятия управленческих решений, в том числе в части оплаты труда работников предприятия.

Эффективно управлять организацией, не уделяя должного внимания контролю работы её работников, невозможно. Классическая триада менеджмента отдельного процесса проекта (постановка задачи, определение срока её исполнения, контроль и оценка выполнения задачи) наряду с возрастающей номенклатурой разновидностей работ и их диверсификацией обуславливает необходимость применения автоматизированных систем управления задачами проекта. Тщательно выстроенная система контроля требует скрупулезного отношения к таким вопросам, как правильное распределение нагрузки между работниками, оперативное отслеживание скорости и качества выполнения поставленных задач, формирование понятных схем взаимодействия между подразделениями и принципов удобного документооборота. Мониторинг всех этих аспектов может отбирать много сил. Сократить трудоемкость процесса помогают современные IT-решения, которые дают руководителя четкое представление о качестве работы персонала компании, статусе выполнения задач в режиме реального времени. Подобные программные системы, используемые вместе с иными мерами воздействия, позволяют предприятиям существенно улучшать ключевые показатели эффективности и выходить на качественно новый уровень в развитии бизнеса [2, 3].

1.2. Цели и задачи проекта, план работы

Целью настоящего проекта является разработка информационной системы контроля трудовых ресурсов.

Целевая аудитория проекта - трудовые коллективы и их руководители, которые:

- Ведут планирование и распределяют ресурсы;

- Собираются для обсуждения планов, распределения задач и подведения итогов;

- Во время проведения работ (возникновении проблем с их выполнением) имеют возможность оперативного оповещения коллег о процессе (выполнения работы);

- Имеют возможность проведения разнообразного анализа выполненной, текущей и предстоящей работы по проекту.

Объективные потребности:

Необходимо сделать доступной самостоятельную работу сотрудников и их локальных руководителей:

1. Информация об общей цели и о каждой задаче всегда были доступны;

2. Регулярные уведомления о работах;

3. До начала работ каждый сотрудник должен иметь информацию о том:

- где он находится;

- что он должен сделать;

- что ему потребуется;

- кто его задерживает;

- кого он задерживает;

- что является результатом работ.

4. Отслеживание эффекта от принятых решений:

- Сохранять историю и результаты работ;

- Рассчитывать эффективность хода работ (освоенный объём);

- Рассчитывать нагрузку сотрудников;

- Рассчитывать КПД объектов (оборудования и т.п.) над которым проводились работы.

Исходя из целей проекта основные задачи проекта определяются следующими:

1. Изучить текущее состояние рынка существующего программного обеспечения, адаптированного под специфику отечественного бизнеса;

2. Рассмотреть варианты создания программного обеспечения для оперативного контроля трудовых ресурсов;

3. Разработать архитектуру информационной системы, определить алгоритм её работы;

4. Изучить основные инструменты для реализации задач, определить необходимый стек технологий;

5. Создать базу данных, реализовать с ней информационный обмен;

6. Реализовать сбор отчётов, необходимых руководителю бизнеса;

7. Разработать программу для работы информационной системы в Telegram (Telegram-bot).

План работы:

Постепенное и поступательное решение задач проекта. Активное участие в составе команды по разработке информационной системы контроля трудовых ресурсов проекта с целью доведения системы до стадии MVP (минимально жизнеспособный продукт) и последующего её модернизации.

Тестовое внедрение информационной системы планируется в Новосибирском агрокомбинате «Сады гиганта», Россия, Новосибирская область, рабочий посёлок Кольцово, 1/3, https:/sadygiganta.ru.

1.3. Команда проекта

Самый важный ресурс для достижения результата проекта - это люди, которые привлечены в проект. Разработка проекта в одиночку – устаревший бесперспективный подход в разработке программного обеспечения.

Команда проекта ― это малочисленная группа людей (обычно 3 -10 человек), которые владеют необходимыми для достижения единой цели знаниями и умениями и совместно отвечают за достижение результата [3].

Основной состав команды разработки проекта обычно такой:

* руководитель проекта
* менеджер проекта
* бэкенд-разработчик
* фронтенд-разработчик
* UI/UX дизайнер
* тестировщик

Для достижения целей и выполнения задач проекта было принято решение о формировании классической команды разработчиков, состоящей, в основном, из действующих студентов сферы IT. Поскольку вначале проекта не было чёткого представления о виде и функциональности его конечного решения с учётом отсутствия финансирования разработка велась с использованием философии гибких методологий Agile. Принимая во внимание повседневную занятость членов команды, их удалённость друг от друга, работа над проектом велась с применением фреймворка Scrum и практик Kanban [15, 16].

В работе над проектом в разное время принимали участие следующие лица:

Таблица 1. Команда проекта

|  |  |  |  |
| --- | --- | --- | --- |
| № | Фамилия, имя | Роль | Зона ответственности |
| 1 | Демченко Максим | Backend-developer | Запросы к БД, отчёты типа “line”, “brigadier”, housekeeper |
| 2 | Джурабаев Дмитрий | Backend-developer | Архитектура БД, запросы к БД |
| 3 | Ефремов Денис | Backend-developer | Сборка проекта на GitHub, анализ логов |
| 4 | Захаров Александр | Frontend-developer | Web-версия приложения |
| 5 | Орлов Дмитрий | Teamlead-BD/Analyst | Руководство командой проекта, архитектура приложения, запросы к БД, серверная часть приложения, коммуникация с заказчиком |
| 6 | Кузнецова Ксения | Frontend-developer | Web-версия приложения |
| 7 | Ножкин Владислав | Teamlead Frontend-developer | Руководство frontend-разработчиками, web-версия приложения, дизайн |
| 8 | Плотников Алексей | Product owner | Коммуникация с заказчиком |
| 9 | Закиев Марат | Backend-developer | Архитектура БД, запросы к БД |
| 10 | Шаверин Никита | Backend-developer | Telegram-бот, архитектура БД, запросы к БД, отчёты типа “list”, “status”, регламентные задания и оповещения, housekeeper |
| 11 | Бойко Никита | Project manager | Организационная работа |
| 12 | Караткевич Артур | Backend-developer | Запросы к БД, отчёты типа “status”, “brigadier” |
| 13 | Пещенкова Мария | Designer | Дизайн |
| 14 | Кашицина Анна | Project manager | Организационная работа |
| 15 | Семочкин Илья | Frontend-developer | Web-версия приложения |
| 16 | Матюшкин Алексей | Frontend-developer | Web-версия приложения |
| 17 | Хазанов Максим | Frontend-developer | Web-версия приложения |
| 18 | Купрейчик Денис | Product manager | Коммуникация с заказчиком |
| 19 | Котельников Алексей | Product manager | Коммуникация с заказчиком |

Список разработчиков информационной системы является отрытым, особенность их статуса (действующие студенты в сфере IT) обуславливает периодическую ротацию.

На момент моего включения в команду разработчиков я был способен разрабатывать алгоритмы выполнения программ, писать код на языке программирования Python, делать запросы на SQL, пользоваться GitHub, работать в операционной системе Linux, имел представление о «железе» персонального компьютера и компьютерных сетях. В процессе работы над проектом мои знания и умения существенно расширились.

2. Теоретическая часть

2.1. Анализ существующего рынка программного обеспечения для реализации контроля трудовых ресурсов

В ходе работы над проектом был изучен рынок программного обеспечения, предназначенного для организации управления проектами и задачами.

Основные продукты:

1. Bitcop

Bitcop – это система контроля выполнения задач и учета рабочего времени, которая способна анализировать все действия персонала за компьютером, что позволяет выявлять узкие места и оптимизировать бизнес-процессы. Система позволяет собирать данные о фактических затратах рабочего времени на задачи и проекты. Bitcop состоит из двух приложений. Одно, Bitcop Agency, собирает данные об активности сотрудников, мониторит их работу за компьютером, а второе, Bitcop Server, анализирует ту информацию, которая стекается на сервер со всех подключенных аккаунтов. Продукт может интегрироваться с другими цифровыми решениями (Redmine, Jira и т.д.), что позволяет собирать дополнительные данные и агрегировать их в системе [2].

Основные функции системы:

1. Формирование отчетов и наглядных графиков по затратам рабочего времени.
2. Оценка продуктивности разных временных интервалов.
3. Мониторинг открытых окон и заголовков в документах.
4. Отслеживание оффлайн-активности сотрудников.
5. Контроль нарушений и уведомление о таковых.
6. Периодическое создание скриншотов экрана ПК и их хранение в архиве.
7. Возможность установления нерабочего времени, в течение которого система не будет отслеживать действия сотрудника за компьютером.

Разработчик – российская компания «Биткоп».

1. Redmine

Redmine – открытое серверное веб-приложение для управления проектами и задачами [4].

Возможности системы:

- ведение нескольких проектов;

- система отслеживания ошибок;

- оповещение об изменениях посредством электронной почты и RSS-каналов;

- настраиваемые статусы задач;

- настраиваемые произвольные поля для задач, временных затрат, проектов и пользователей;

- учет временных затрат (часов);

- диаграммы Ганта и календарь;

- Wiki для каждого проекта;

- ведение новостей проекта, управление файлами и документами;

- форумы для каждого проекта;

- многоязыковой интерфейс, в том числе русский;

- легкая интеграция с репозиториями (SVN, CVS, Git, Mercurial, Bazaar и Darcs);

- система разделения доступа, основанная на ролях;

- поддержка множественной аутентификации LDAP;

- возможность самостоятельной регистрации новых пользователей;

- расширение функциональности системы посредством установки дополнительных плагинов;

- поддержка СУБД: MySQL, PostgreSQL, SQLite, MS SQL Server.

Разработчик Jean-Philippe Lang, Швейцария.

1. JIRA

JIRA – это продукт, предназначенный для организации управления проектами и задачами. Применение JIRA наиболее эффективно в компаниях, занимающихся разработкой каких-либо проектов или программного обеспечения, для управления деятельностью сотрудников и анализа эффективности [5].

JIRA приносит большой эффект любой компании, деятельность которой можно интерпретировать как выполнение каких-либо проектов, задач имеющих тематические и временные рамки. JIRA оперирует такими понятиями как проекты и задачи. Задачи создаются в проектах, на выполнение задач назначаются исполнители. Назначенный исполнитель приступает к выполнению задачи, комментирует ее, вводит проработанное время, присоединяет файлы к описанию задачи, просматривает файлы, относящиеся к задаче с поддержкой версионности. Все важные действия исполнителя журналируются, и о них получают сообщения руководители проекта и другие заинтересованные пользователя имеющие на это право.

JIRA позволяет контролировать ход выполнения задач и более крупно - проектов, управлять загрузкой исполнителей, выделять «узкие» места в проектах и важные задачи. Также, имеется возможность предоставить пользователям специальный доступ для ввода сообщений об ошибках и предложений по усовершенствованию.

В JIRA имеется система движения задач (workflow), позволяющая настраивать процесс движения задач в соответствии с требованиями принятыми на предприятии, управлять видом и набором полей описания задачи.

Разработчик JIRA – компания Atlassian, Австралия.

1. КСК.Service&Teamwork

КСК.Service&Teamwork – это облачное решение для организации работы и коммуникации команды как в офисе, так и в режиме удаленной работы. Позволяет контролировать, что происходит с проектом в целом и с отдельными задачами, настраивать списки задач под себя, использовать фильтры, чтобы получить больше информации о происходящем в проекте [7].

* Простые и сложные проекты в удобном интерфейсе
* Последовательное и параллельное исполнение задач, подзадачи
* Удобные инструменты для совместной работы по проектам
* Группировка задач вне проектов для исполнения и контроля
* Контроль, аналитика, отчеты

КСК.Service&Teamwork является российским продуктом, разработчик КСК Технологии.

1. A2B

А2Б - онлайн-система управления предприятием, предлагаемая к использованию на основе [SaaS](https://www.tadviser.ru/index.php/SaaS) (от англ. Software as a Service – программное обеспечение как услуга). Использовать А2Б можно с любого устройства (напр. смартфона, планшета, ноутбука), подключенного к сети интернет. Система представляет собой комплексное решение задач планирования, управления проектами, ведения продаж (CRM, контроля поручений, документооборота), организации взаимодействия между сотрудниками, учета идей [8].

А2Б – это удобные интернет-сервисы для организации таких внутренних бизнес-процессов как: долгосрочное и среднесрочное планирование, контроль поручений, управление структурой компании и персоналом, управление проектами, управление взаимоотношениями с клиентами, корпоративный портал.

Модули системы:

* CRM система - система управления взаимоотношениями с клиентами
* Проекты - система управления проектами
* Поручения - контроль исполнения поручений
* СЭД - система электронного документооборота
* Цели - стратегические цели компании
* Планы - оперативные планы подразделений и сотрудников
* Идеи - учет идей и рациональных предложений
* Расписания - ведение расписаний компании
* Главная страница - корпоративный портал компании
* Календарь - электронный ежедневник сотрудника
* Документы - хранилище электронных документов
* Персонал - телефонный справочник и управление структурой

Разработчик – ООО «А2Б», Россия.

1. Битрикс24

«Битрикс24» - это программный комплекс (сервис), предназначенный для оптимизации ведения бизнеса и контроля за производственным процессом. Битрикс24 – это набор полезных инструментов, которые помогают бизнесу работать: CRM, видеозвонки, документы онлайн, задачи и проекты, контакт-центр, конструктор сайтов и интернет-магазины. Битрикс24 помогает руководителю выстроить в своём коллективе эффективную работу [6].

Основные возможности Битрикс24:

* управление задачами и проектами;
* планирование рабочего времени;
* работа с документацией;
* автоматизация продаж;
* общение с клиентами.

Разработчик и провайдер – российская компания «Битрикс».

1. Аспро.Cloud

Аспро.Cloud – облачная платформа для совместного ведения проектов, постановки задач, управления продажами и контроля финансов. При помощи Аспро.Cloud компании автоматизируют бизнес-процессы, сокращают рутинных задач и налаживают совместную работу всех отделов. Платформа подойдёт для структурирования работы в малом бизнесе, небольших строительных компаний, рекламных и digital-агентств, юридических фирм, дизайн-студий и агентств недвижимости, производственных компаний и организаций, работающих по проектному подходу [9].

Особенности Аспро.Cloud:

* Ведение проектов любой сложности
* CRM с подробной историей по клиентам и автоматизацией
* Статистика и детализированные финансовые отчёты
* Гибкие права доступа для сотрудников
* Интеграция с популярными сервисами (телефония, банки, почта, конструкторы сайтов и т.д.)
* Выставление счетов, формирование КП и отправка актов
* 5 способов мониторинга задач: канбан-доска, диаграмма Ганта, календарь, список, GTD-планировщик
* Встроенный чат и моментальные уведомления
* Возможность создать собственную базу знаний и встроить ее на свой сайт
* Гостевой доступ для клиентов и подрядчиков
* Современное мобильное приложение

Разработчик – российская компания «Аспро».

1. Planny24

Planny24 – облачный сервис для управления процессами технической эксплуатации [13].

Основные возможности:

* Управление работами (нарядами);
* Ведение данных оборудования;
* Планирование работ;
* Отчётность и аналитика;
* Хранение технической документации;
* Облачный сервис.

Основные недостатки:

* Больше подходит для фрилансеров, среднего и малого бизнеса, индивидуальных предпринимателей;
* Только коробочное решение, нет возможности подобрать функционал под себя;
* Нет технической поддержки сотрудников;
* Авторизация с помощью e-mail-аккаунта;
* Многопользовательский доступ отсутствует;
* Отсутствует интеграция с программами.

Разработчик – ЗАО «Си Проект», Россия, г. Санкт-Петербург.

1. Мегаплан

Мегаплан - СРМ-система, созданная в первую очередь для управления проектами и обеспечения контроля над сотрудниками. Инструментарий направлен на организацию труда, оптимизацию рутинных обязанностей и автоматизацию бизнес-процессов [14].

Преимущества:

* Много функций и настроек;
* Детальный help-сервис;
* Интегрируется со многими другими программами;
* Ведение данных оборудования;
* Отчётность и аналитика;
* Управление работами.

Недостатки:

* Нужно устанавливать мобильное приложение;
* Платный облачный сервис с ежемесячной оплатой, либо дорогостоящая покупка и установка ПО на сервер компании;
* Медленная техническая поддержка.

Разработчик – российская компания «Первый бит», г. Омск.

На отечественном рынке пока находят себе применение такие иностранных приложения, как SingularityApp, TickTick, Todoist и др., однако в свете усиления санкционного давления недружественных стран с 2022 года их использование в ближайшей перспективе сопряжено с определенными рисками. К тому же некоторые приложения имеют скудный функционал в бесплатной версии, для них нужно устанавливать специальное приложение, у них высокая цена платных версий.

2.2. Определение требований к информационной системе

Анализ существующих программных продуктов обнаружил некоторые недостатки, обуславливающие дальнейшее проведение разработки ПО в данном направлении:

* необходимость установки дополнительного программного обеспечения на устройство конечного пользователя;
* сложность настойки системы для отдельного клиента;
* отсутствие возможности работы в системе при помощи существующих мессенджеров;
* сложность дальнейшей поддержки систем, разработчиком которых является иностранная компания.

Учитывая недостатки и преимущества существующих систем были сформированы требования к разрабатываемой системе.

Крайне желательно использовать существующую инфраструктуру. Без использования специальных мобильных приложений, процедур авторизации. Это должно дать лёгкое подключение пользователей в коллектив.

Информационный ресурс системы (база данных, СУБД, модули программного кода, конфигурационная информация) должен находиться на удалённом сервере. Общение с системой должно быть простым, без прохождения обучения. Максимально быстрое внедрение системы.

Проекту дадут конкурентные преимущества:

I) использование имеющихся приложений:

1. Телеграмм:

- для оперативного получения информации по задачам, долгам и планам;

- для получения уведомлений о задачах;

- общения с коллегами;

- отправки результатов работ;

2. Браузер:

- для отображения всего плана работ;

- отображения распределения нагрузки;

- карточки объекта;

II) Использование диаграмм для представления планов на различный период в прошлом и будущем:

- Упрощает инструментарий, единый интерфейс;

- Умещает весь период на одном экране;

- Позволяет переключаться и детализировать между группами оборудования, сотрудниками, работами;

- Отображает коэффициенты эффективности;

- Отображает информацию по объекту, сотруднику или работе;

III) Общение с телеграмм-ботом:

1. Естественный язык запросов, например: Мой план на завтра;

2. Использование синонимов и умолчаний:

- Мой план/Мои задачи на завтра/задачи на завтра;

3. Возможность переключиться на меню;

4. Регулярные уведомления;

5. Рассылка сообщений.

25 марта 2023 года на очередном собрании членов команды разработчиков системы было утверждено её название - «Совместный труд». Основной лозунг - ставь задачи, береги время, достигай цели.

2.3. Анализ архитектурных решений и алгоритмов для реализации информационной системы

Архитектура информационной системы – это описание продукта программирования (операционных систем, приложений, программ) с точки зрения пользователя, заказчика и специалиста по проектированию. В ней объясняется, из чего состоит система, как элементы взаимодействуют между собой [10]. Любая информационная система включает в себя три компонента:

1. Управление данными;
2. Бизнес-логика;
3. Пользовательский интерфейс.

Данные хранятся в базах данных, а управление ими осуществляется с помощью системы управления базами данных. Бизнес-логика определяет правила, по которым обрабатываются данные. Она реализуется набором процедур, написанных на различных языках программирования. Пользователь работает с интерфейсом, где логика работы информационной системы представлена в виде элементов управления – полей, кнопок, списков, таблиц и т.д. Компоненты в разных информационных системах взаимодействуют между собой различными способами [11].

Существуют следующие виды архитектур информационных систем [11, 12]:

1. Локальные информационные системы. В этом случае все компоненты системы располагаются на одном компьютере. Очевидным недостатком этой архитектуры является возможность работать только одному пользователю. Другие пользователи не имеют возможности получить доступ к данным даже для чтения
2. Файл-серверная архитектура. В данной архитектуре данные хранятся в файлах на выделенном специально для этой цели компьютере (файловом сервере). Компьютеры пользователей соединены с сервером сетью, поэтому доступ к данным могут получить несколько пользователей одновременно. Однако, кроме функций хранения данных и обеспечения доступа к ним, файловый сервер никаких функций не выполняет. Приложения, обрабатывающие данные, находятся на пользовательских компьютерах. Обработка данных на пользовательском компьютере всегда сопровождается передачей по сети большого количества информации. Недостатками данной архитектуры являются высокая загруженность сети (и, как следствие, низкая скорость работы), сложность поддержания непротиворечивости данных из-за несогласованной обработки разными пользователями.
3. Клиент-серверная архитектура. В данном виде архитектуры в состав системы управления базами данных включается новый компонент – процедурный язык программирования, с помощью которого появляется возможность создания хранимых процедур обработки данных и обработки их значительного объёма непосредственно на сервере. Данная архитектура позволяет существенно разгрузить сеть и поддерживать непротиворечивость данных за счёт их централизованной обработки. Однако языки хранимых процедур не приспособлены для полноценной реализации бизнес-логики и она реализуется на клиентском компьютере. Такой подход имеет следующие недостатки: любые изменения в бизнес-логике требуют обновления на клиентском компьютере, клиентские компьютеры должны быть достаточно производительными, система слабо защищена от взломов.
4. Трёхуровневая (многоуровневая) архитектура. В данном виде архитектуры реализация бизнес-логики перенесена с клиентских компьютеров на специальный промежуточный уровень – сервер приложений, представляющий собой комплекс программ, выполняемых на сервере. Использование сервера приложений позволяет максимально разгрузить клиентские компьютеры и сделать обработку данных ещё более централизованной, что повышает скорость и надёжность информационной системы.
5. Многоуровневая интернет-архитектура. В данном виде архитектуры к рассмотренным выше уровням добавляется web-сервер (сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными). Примером многоуровневой интернет-архитектуры можно назвать веб-приложения. Их отличительной особенностью является передача запросов и результатов их обработки через сеть Интернет и использование в качестве клиентского приложения стандартного Интернет-браузера. При этом структура информационного системы приобретает следующий вид: браузер – web-сервер – сервер приложений – сервер баз данных. Благодаря интеграции Интернет-технологии и архитектуры клиент-сервер процесс внедрения и сопровождения корпоративной информационной системы существенно упрощается при сохранении достаточно высокой эффективности и простоты совместного использования информации.

2.4. Изучение инструментов для создания информационной системы, определение основного стека технологий

Для создания информационной системы «Совместный труд» разработчиками была выбрана многоуровневая интернет-архитектура. Её реализация требует наличие следующих инструментов:

Таблица 2. Инструменты для создания многоуровневой интернет-архитектуры

|  |  |  |  |
| --- | --- | --- | --- |
| № | Уровень | Инструмент | Варианты |
| 1 | Нижний уровень | Браузер | Google Chrome, Internet Explorer, Mozilla FireFox |
|  |  | Бот | Telegram-bot |
| 2 | Средний уровень | Web-сервер | Apache, Nginx |
|  |  | Сервер приложений | Python, JavaScript |
| 3 | Верхний уровень | Система управления базой данных | PostgreSQL, MySQL |

Таблица 3. Стек основных технологий для создания информационной системы «Совместный труд»

|  |  |  |
| --- | --- | --- |
| № | Технология | Задачи |
|  | Аспро.Cloud | Управление процессом разработки программного обеспечения. |
|  | Figma | Cовместное проектирование дизайна пользовательских интерфейсов |
|  | Html, CSS, JavaScript, WebSocket | Разработка Web-приложения |
|  | Python (native и фреймворки pyTelegramBotAPI, MatPlotLib, NumPy, SciPy, cgi) | Разработка сервера приложений, телеграмм-бота |
|  | PostgreSQL | Система управления базой данных |
|  | ОС Linux (Ubuntu 22.04.2) | Сервер |
|  | Git, GitHub | Управление версиями, хранилище программного кода |
|  | PyCharm | Среда разработки программного обеспечения |
|  | DBeaver | Администрирование базы данных |
|  | Apache | Web-сервер |

3. Практическая часть

3.1. Архитектура информационной системы «Совместный труд»

В основу архитектуры информационной системы «Совместный труд» положен широко используемый паттерн проектирования MVP (Model-View-Presenter). Пользователи системы могут использовать систему при помощи интернет-браузера (веб-версия) и/или мессенджера Telegram (мобильная версия). Веб-версия системы отличается расширенным функционалом, позволяющим в полном объёме вносить информацию в базу данных о сущностях и их взаимных связях, производить необходимые изменения данных, получать широкий перечень отчётов и статистической информации. Реализация данного направления находится в зоне ответственности frontend-разработчиков. Мобильная версия предназначена для регистрации и авторизации пользователя в системе, оперативного получения наиболее важной информации в виде отчётов, расписаний и напоминаний, а также проведения некоторых видов настроек системы.

Бизнес-логика информационной системы (Presenter) выделена в отдельные программные модули, разделённые по функциональному назначению.

Слой данных информационной системы реализован с использованием СУБД PostgreSQL, программный код управления данными также выделен в отдельные программные модули.

Программный код бизнес-логики и база данных находятся на отдельном сервере под управлением операционной системы Ubuntu 22.04.2 LTS.

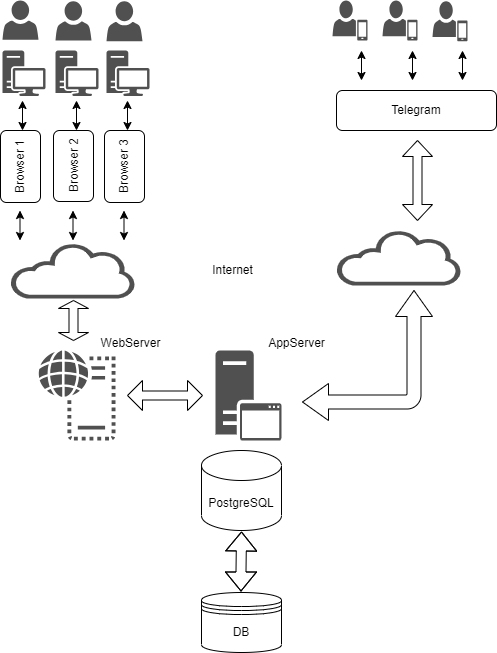


Рисунок 1. Структура информационной системы «Совместный труд»

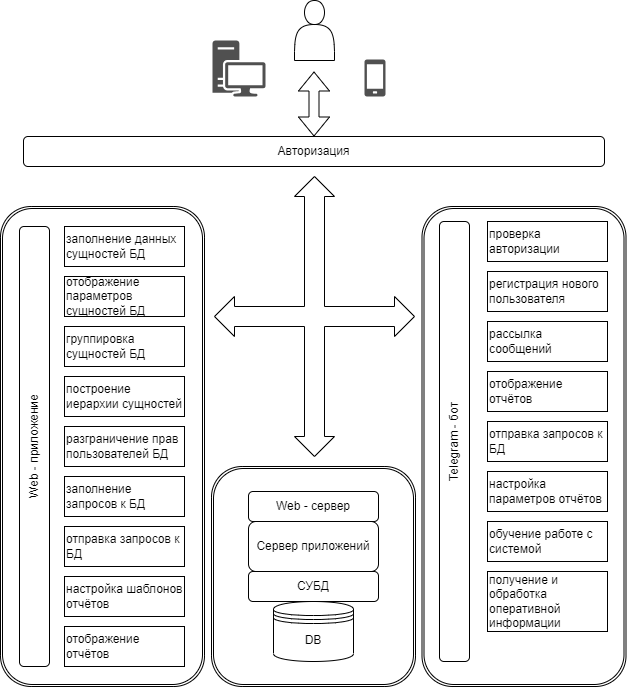


Рисунок 2. Функциональная схема информационной системы «Совместный труд»

Программный код проекта «Совместный труд» находится на GitHub по адресу <https://github.com/lesenkagit/sovtrud>

Сервер БД по адресу - <http://91.77.164.63/>

Веб-версия информационной системы: <http://sovtrud.ru/>

Telegram-bot - https://t.me/sovtrudBot

3.2. Разработка структуры базы данных информационной системы

База данных - организованная структура, предназначенная для хранения информации. Современные базы данных позволяют размещать в своих структурах не только данные, но и методы, с помощью которых происходит первичная обработка хранящейся в базе данных информации. Структура базы данных определяется составом и типами хранимых в ней записей, их организацией, связью между собой, а также особенностями предметной области и механизмами взаимодействия с сервером приложений (программным кодом приложения).

Для информационной системы «Совместный труд» была выбрана реляционная модель базы данных, использующая табличный способ хранения данных и состоящая из нескольких раздельных таблиц. В качестве СУБД была определена одна из наиболее популярных СУБД объектно-реляционная PostgreSQL, версия 15.3. Одним из определяющих факторов в пользу выбора СУБД PostgreSQL стала возможность хранения тип данных JSON в таблице SQL, поскольку данный формат позволяет существенно упростить масштабируемость информационной системы и неизменным добавлением априори неизвестных типов данных.

Сущностями информационной системы являются:

* Object – объект предметной области (проект, основное средство, предмет и т.д.);
* Relation – работа, выполняемая с объектом, её описание;
* Subject – физическое лицо, производящее работу. Тот, на кого можно назначить выполнение работы (постановку задачи).

В общем виде структура сущностей внутри БД одинакова.

Задачи, возлагаемые на СУБД:

1. Хранение информации о сущностях;
2. Хранение информации о задачах и их описании;
3. Хранение информации о правах и группах сущностей;
4. Хранение информации о сообщениях между сущностями;
5. Хранение информации об авторизации;
6. Хранение информации об отчётах;
7. Хранение информации о callback\_data Inline Keyboard телеграмм-бота. Необходимо из-за ограничения Telegram в 64 байта.

Для реализации поставленных задач в информационной системе предусмотрены следующие таблицы:

1. Entity – информация о сущностях;
2. Plan – информация о задачах;
3. Giduid – таблица соответствия групп и сущностей;
4. Authwait – таблица авторизации;
5. Message – таблица сообщений;
6. Comments – таблица комментариев;
7. Keyscallback – таблица calback\_data telegram-бота.

Рассмотрим основные из них.

Таблица 4. Entity. Хранение информации о сущностях.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя столбца | Тип данных | Описание |
| 1 | id | serial4 | Entity id |
| 2 | parentid | int4 | id родительской сущности |
| 3 | uid | int4 | id создателя |
| 4 | gid | int4 | id группы сущности |
| 5 | type | varchar(32) | тип сущности (object, group, subject, relation, project) |
| 6 | name | varchar(128) | имя сущности |
| 7 | parameters | jsonb | параметры сущности, хранятся в NoSQL бинарном представлении данных формата json |
| 8 | rights | varchar(10) | права сущности |
| 9 | is\_active | bit(1) | состояние сущности |

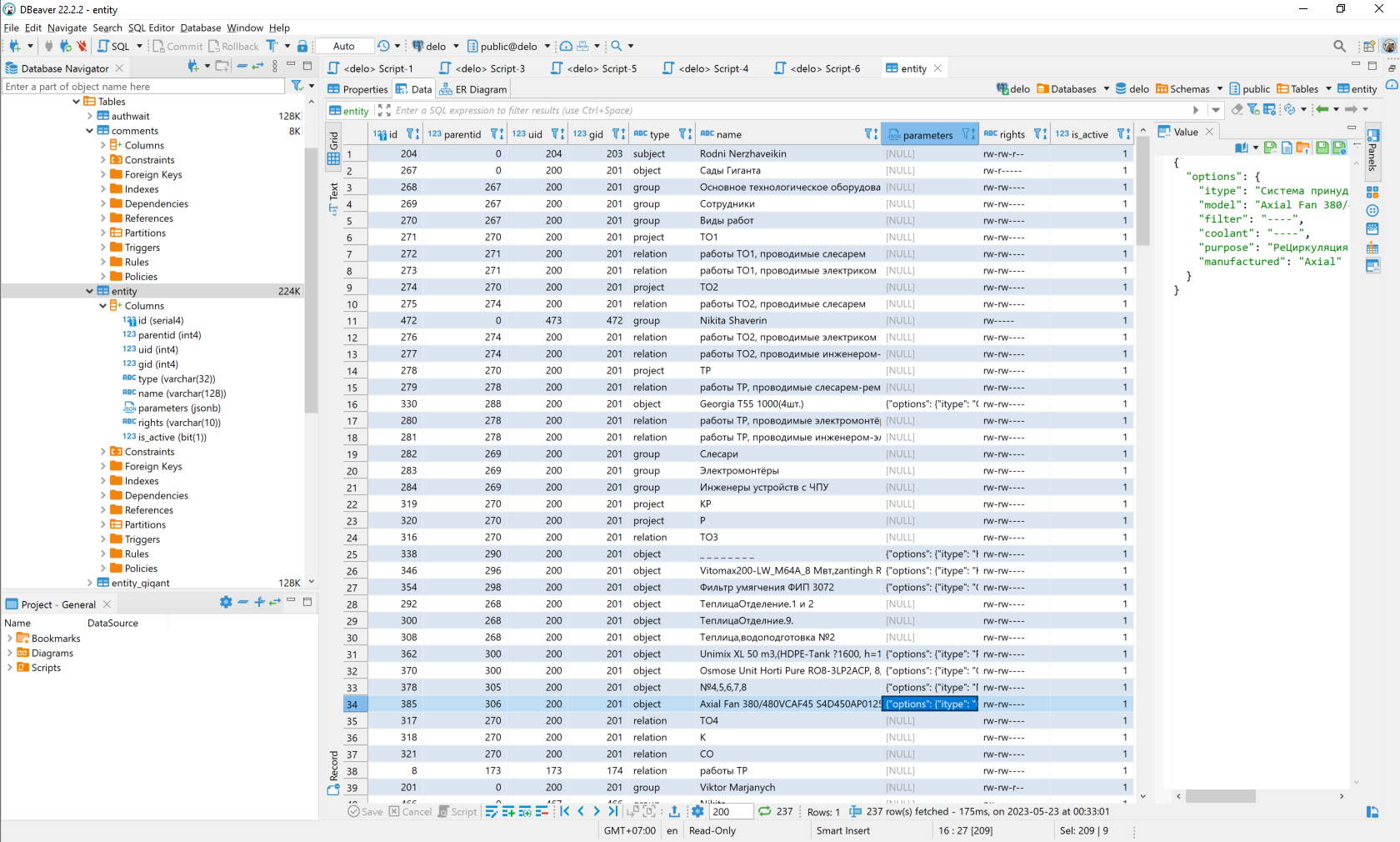


Рисунок 3. Скриншот фрагмента таблицы Entity DBeaver

Таблица 5. Plan. Хранение информации о задачах.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя столбца | Тип данных | Описание |
| 1 | id | serial4 | Plan id |
| 2 | subjectid | int4 | id субъекта, на кого поставлена задача |
| 3 | relationid | int4 | id работы, которая включена в задачу |
| 4 | objected | int4 | id объекта, который включён в задачу |
| 5 | plandate | date | плановая дата выполнения задачи |
| 6 | factdate | date | фактическая дата выполнения задачи |
| 7 | anoncedate | date | дата постановки задачи |
| 8 | prev | int4 | id предыдущей задачи |
| 9 | next | int4 | id следующей задачи |
| 10 | projectid | int4 | id проекта |
| 11 | projectmark | varchar | поле комментария |

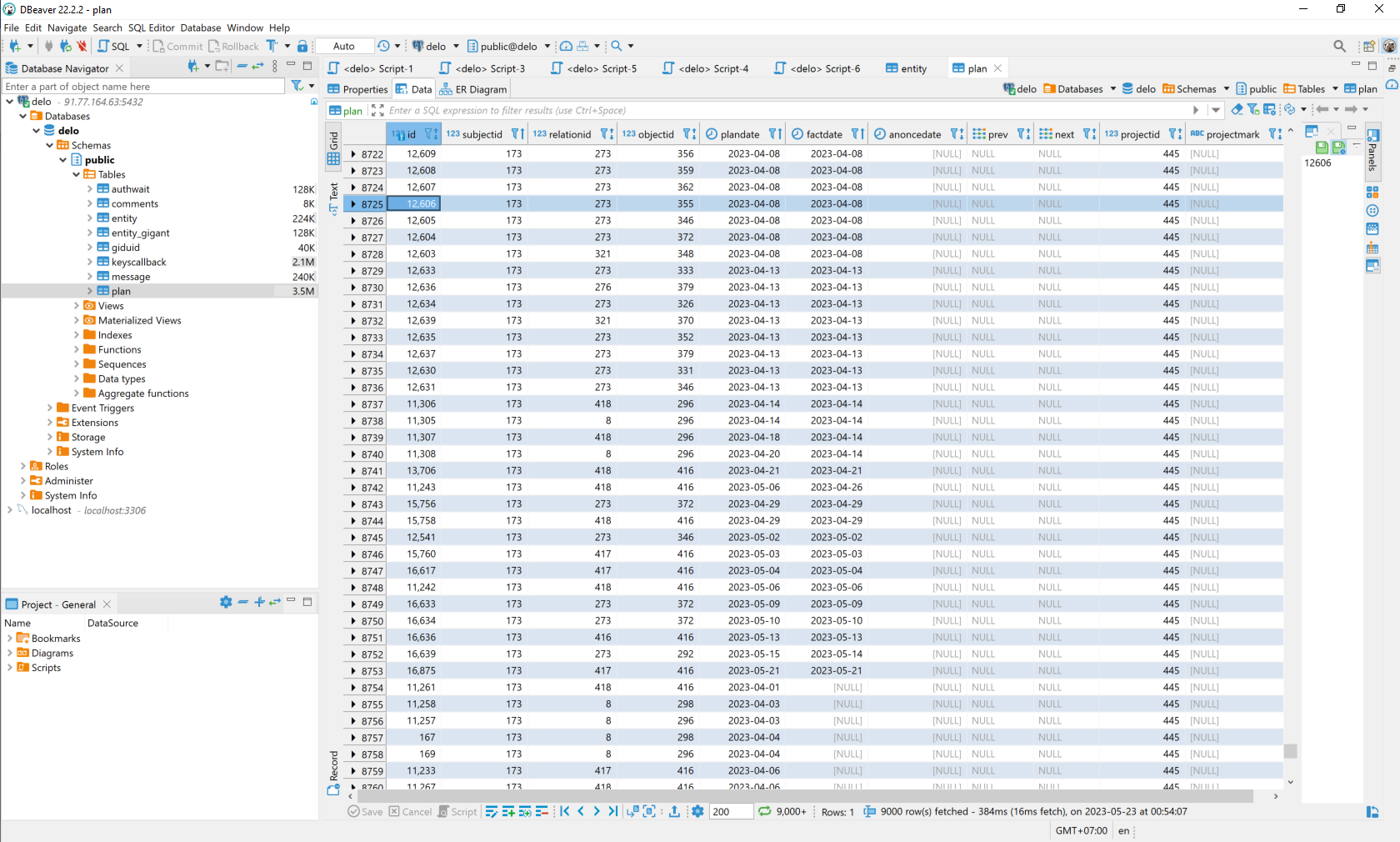


Рисунок 4. Скриншот фрагмента таблицы Plan DBeaver

Таблица 6. GidUid. Группы и пользователи.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя столбца | Тип данных | Описание |
| 1 | gid | int4 | Group\_id |
| 2 | entity\_id | int4 | Entity id |

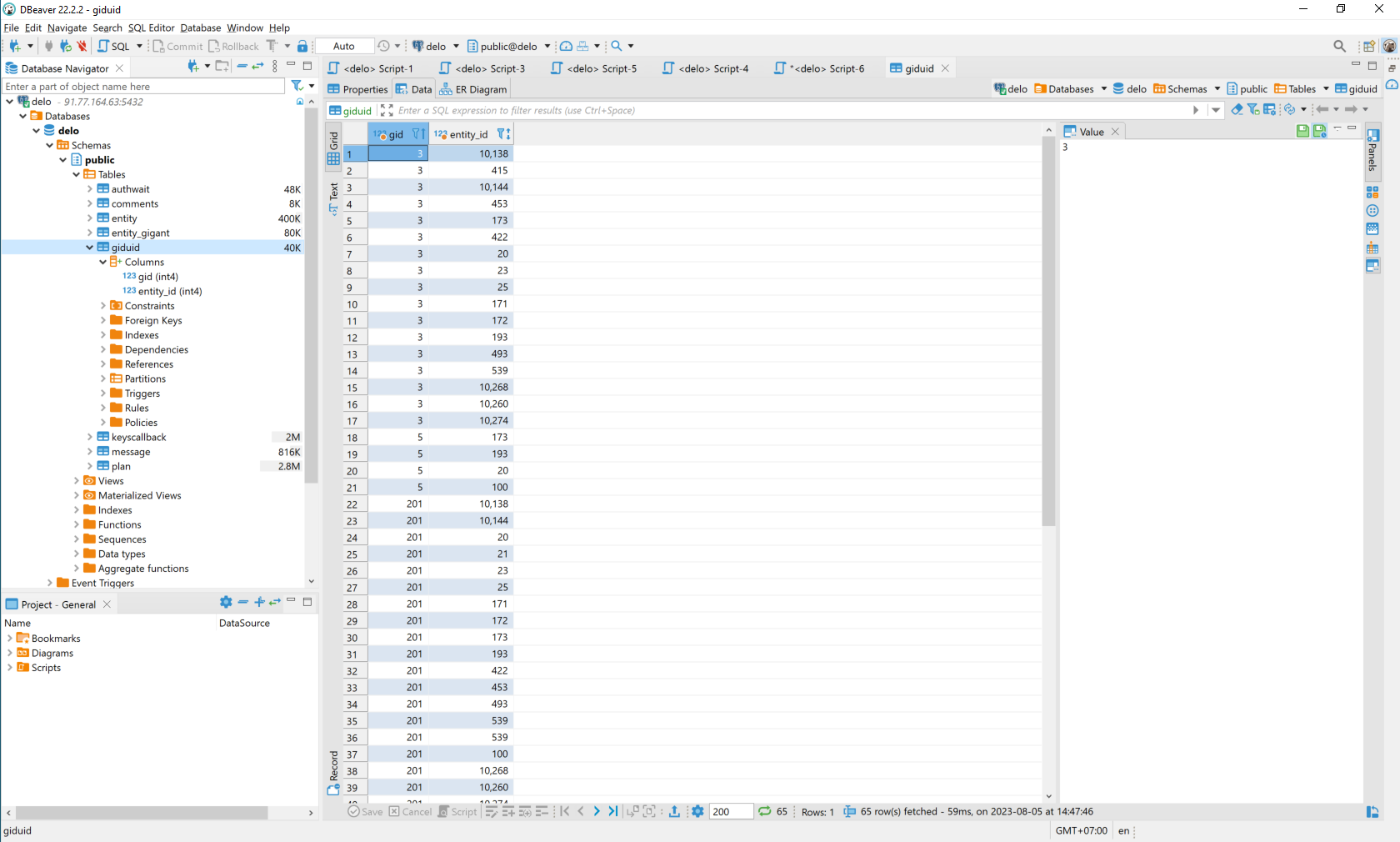


Рисунок 5. Скриншот фрагмента таблицы GidUid DBeaver

Таблица 7. Message. Сообщения пользователей.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя столбца | Тип данных | Описание |
| 1 | uid | int4 | id субъекта, кому предназначено сообщение |
| 2 | plandate | timestamp | время записи сообщения в базу данных |
| 3 | senddate | timestamp | время отправки сообщения телеграмм-ботом получателю |
| 4 | text | varchar | текст сообщения |
| 5 | parse\_mode | varchar | формат сообщения |
| 6 | mid | serial4 | идентификационный номер сообщения |

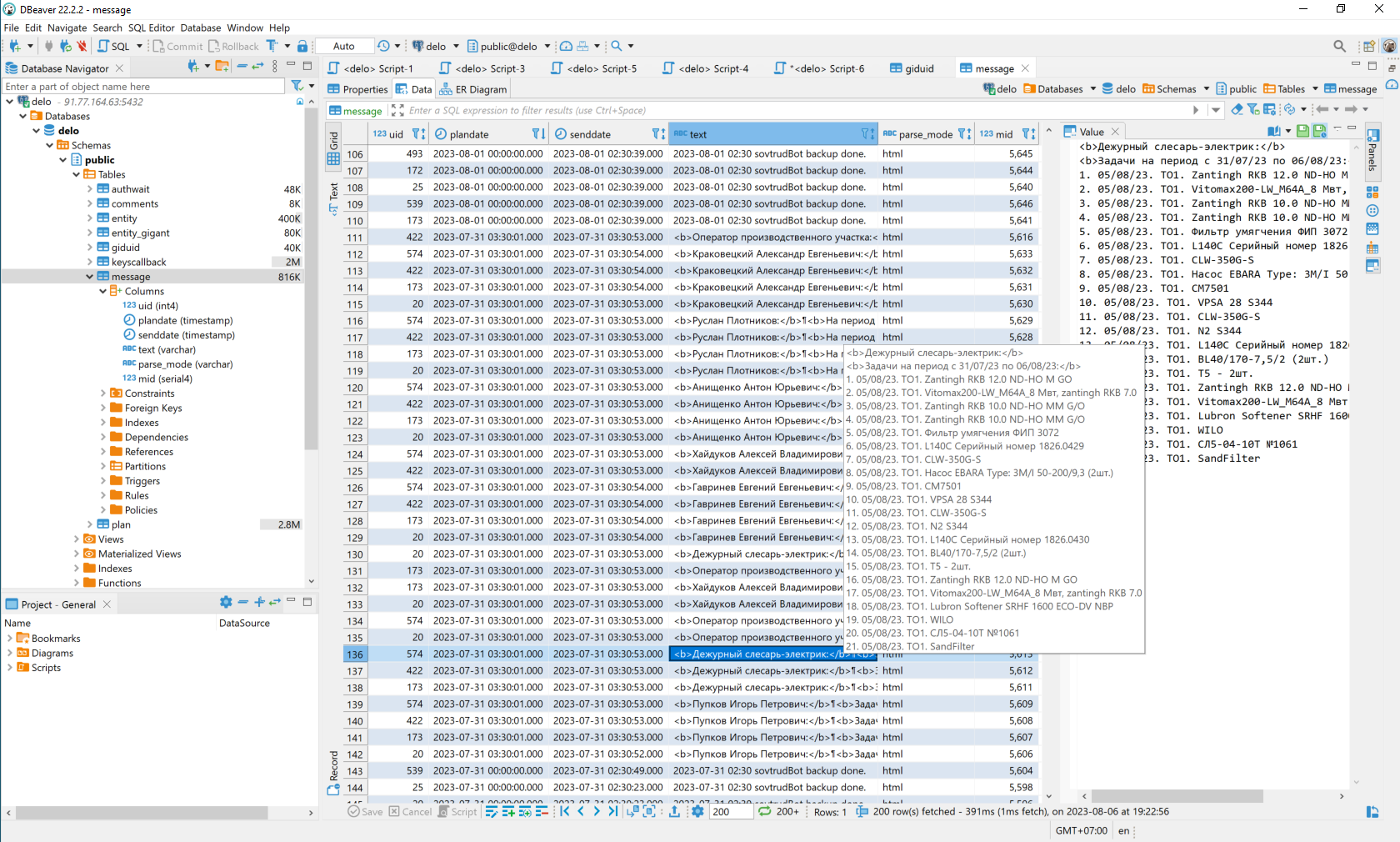


Рисунок 6. Скриншот фрагмента таблицы Message DBeaver

Таблица 8. Keyscallback. Таблица calback\_data telegram-бота.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя столбца | Тип данных | Описание |
| 1 | chatid | int8 | telegram chatid |
| 2 | hash | varchar | hash callback |
| 3 | callback | varchar | callback на button Inline Keyboard telegram-бота |
| 4 | timeout | timestamp | время жизни callback |

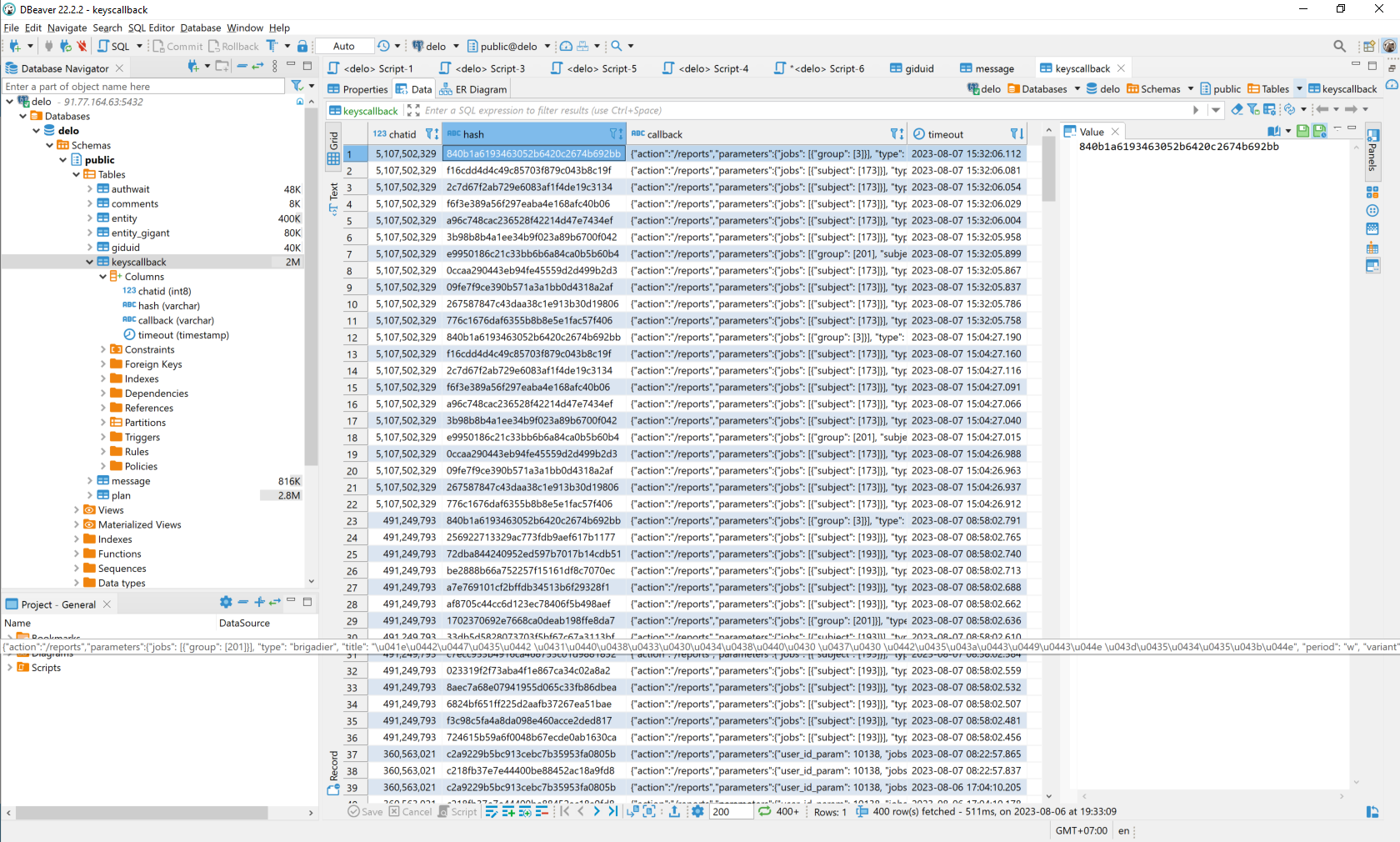


Рисунок 7. Скриншот фрагмента таблицы Keyscallback

На уровне SQL реализованы следующие функции:

Таблица 9. Функции, реализованные на уровне SQL

|  |  |  |
| --- | --- | --- |
| № | Функция | Назначение |
| 1 | get\_all\_node\_children | получение потомков сущности |
| 2 | get\_group\_entities | получение сущностей, входящих в группу |
| 3 | get\_user\_groups | получение групп, в которые входит пользователь |
| 4 | isrights | получение прав сущности |
| 5 | newuser | создание нового пользователя |

3.3. Разработка алгоритма модуля Telegram-bot

Модуль Telegram-bot разработан с использованием библиотеки pyTelegramBotAPI [17, 18]. Алгоритм его работы приведён на рисунке 8.

В начале работы происходит загрузка конфигурационной информации из файла (параметры подключения к базе данных, token телеграмм-бота, параметры логирования и т.д.), демоном запускается телеграмм-бот, который с заданной периодичностью проверяет наличие новых сообщений. В случае прихода новых сообщений стартует модуль worker, в который передаётся полученное сообщение. Модуль worker запускается в отдельном потоке для каждого сообщения. Если новых сообщений с момента прошедшего опроса состояния не поступило, запускается модуль housekeeper, который делает рассылку всех неотправленных сообщений из таблицы message согласно указанным в них пользовательским telegram\_id. Также модуль housekeeper производит чистку безнадёжных сообщений в таблице message.

Модуль worker посредством модуля tokens парсит входящее сообщение и формирует управляющую структуру Tokens. В случае получения какого-либо файла во входящем сообщении, модуль worker разбирает подпись к нему, определяет его тип, именует и сохраняет в соответствующей директории на сервере. В последующем полученные файлы могут быть отправлены адресату или проанализированы руководителем проекта при наличии соответствующих прав доступа.

После разбора сообщения и формирования управляющей структуры Tokens модуль worker передаёт её в модуль route, который проверяет наличие пользователя с указанным в Tokens telegram\_id в базе данных. В случае отсутствия такого пользователя происходит его регистрация (при установке соответствующего маркера) в базе данных и создание необходимых шаблонов, подписок, параметров telegram и т.д.

Далее из управляющей структуры извлекается параметр action (действие, выбранное пользователем), согласно которому запускается соответствующий обработчик (start, date, message, settings, help, time\_zone, reports). Управляющая структура Tokens передаётся в обработчик в качестве аргумента . По завершению своей работы каждый обработчик модифицирует Tokens согласно полученному результату (формирует текст сообщения, клавиатуры InlineKeyboard и/или ReplyKeyboard, сохраняет графические файлы и ссылки на них и т.д.) и возвращает Tokens, согласно которому модуль tokens формирует ответное сообщение и отправляет его в соответствующему пользователю.

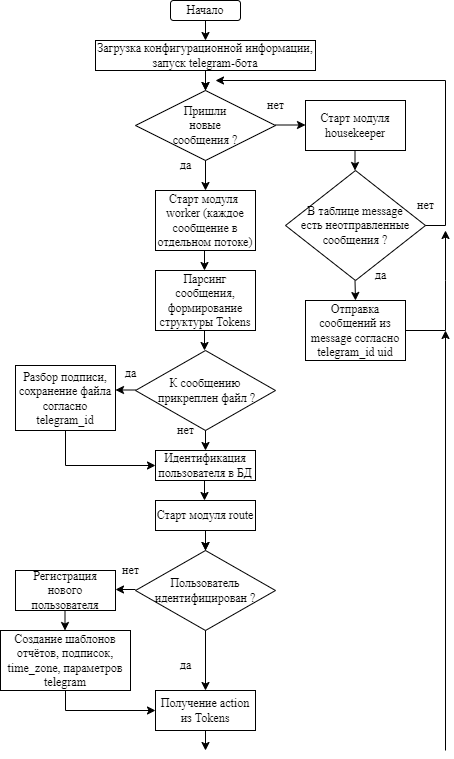


Рисунок 8. Алгоритм работы модуля Telegram-бот

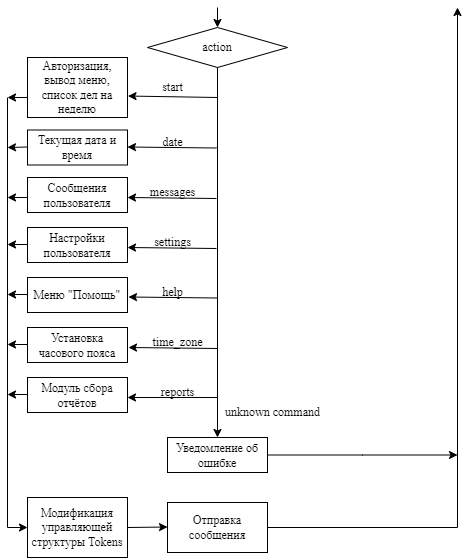


Рисунок 8. Алгоритм работы модуля Telegram-бот (продолжение)

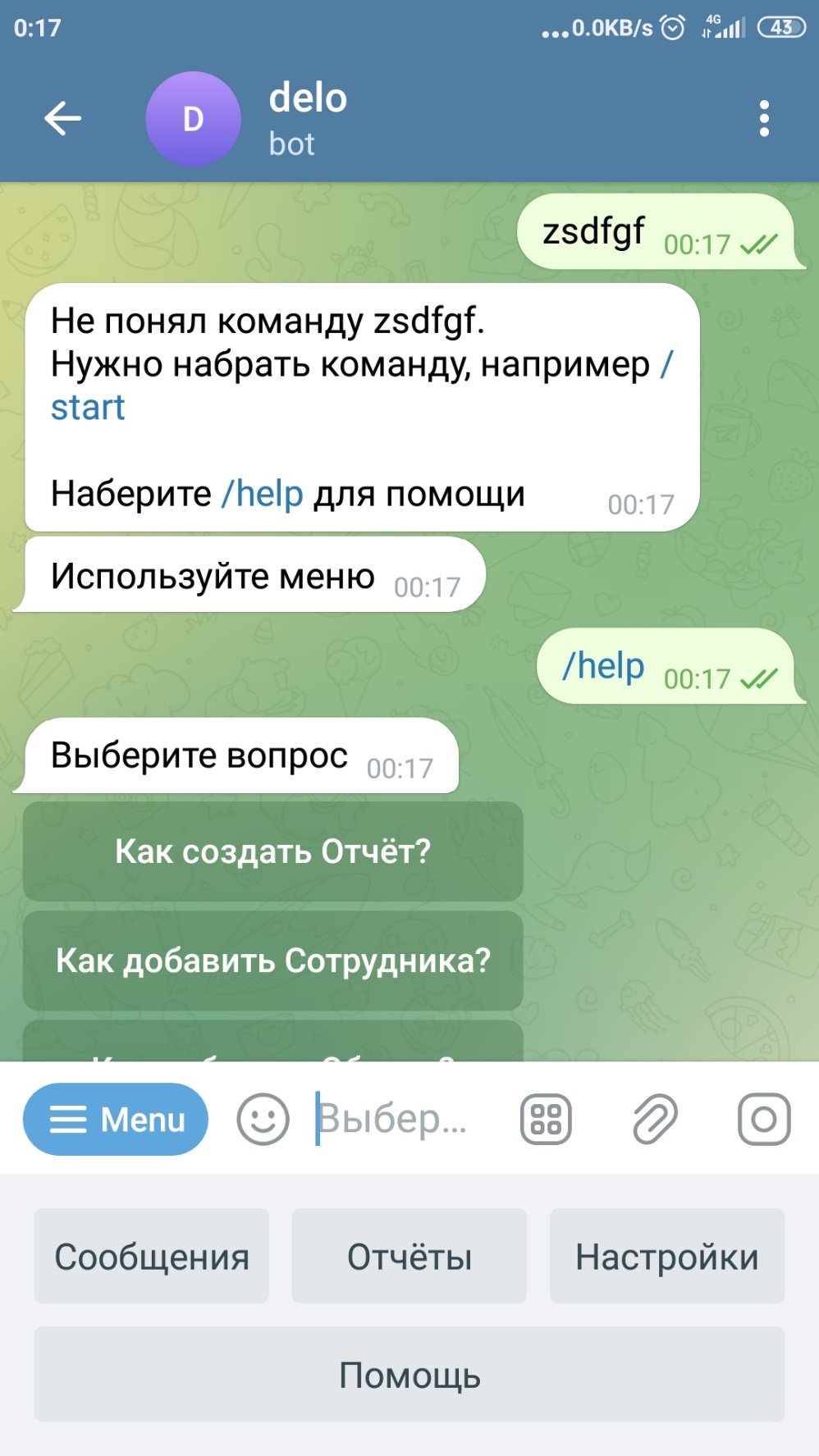


Рисунок 9. Меню Telegram-бот

Наиболее информативным для пользователя системы является модуль сбора отчётов, который запускается нажатием соответствующей кнопки ReplyKeyboard или ввода текстовой команды пользователем.

При первом вызове модуля сбора отчётов происходит формирование перечня отчётов для данного пользователя согласно шаблонам, записанным в базе данных. Формируется InlineKeyboard, в которой одна кнопка соответствует одному отчёту. InlineKeyboard размещается в управляющей структуре Tokens, согласно которой формируется ответное сообщение пользователю в виде набора кнопок.

При нажатии пользователем на одну из кнопок InlineKeyboard происходит считывание типа отчёта (list, status, line, brigadier) и вызов соответствующего модуля-обработчика. Аргументом обработчика передаётся управляющая структура Tokens.

Обработчик определяет период отчёта, отправляет запросы в базу данных, получает необходимую информацию, формирует ответ пользователю (InlineKeyboard со списком задач, сводную информацию в текстовом виде, график), создаёт преамбулу (составитель отчёта, период и т.д.), модифицирует и возвращает управляющую структуру Tokens.

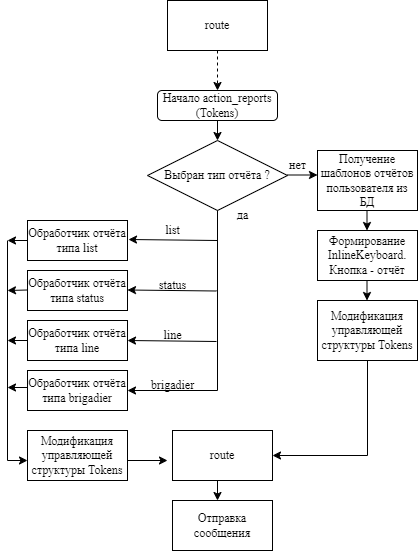


Рисунок 10. Алгоритм работы модуля сбора отчётов

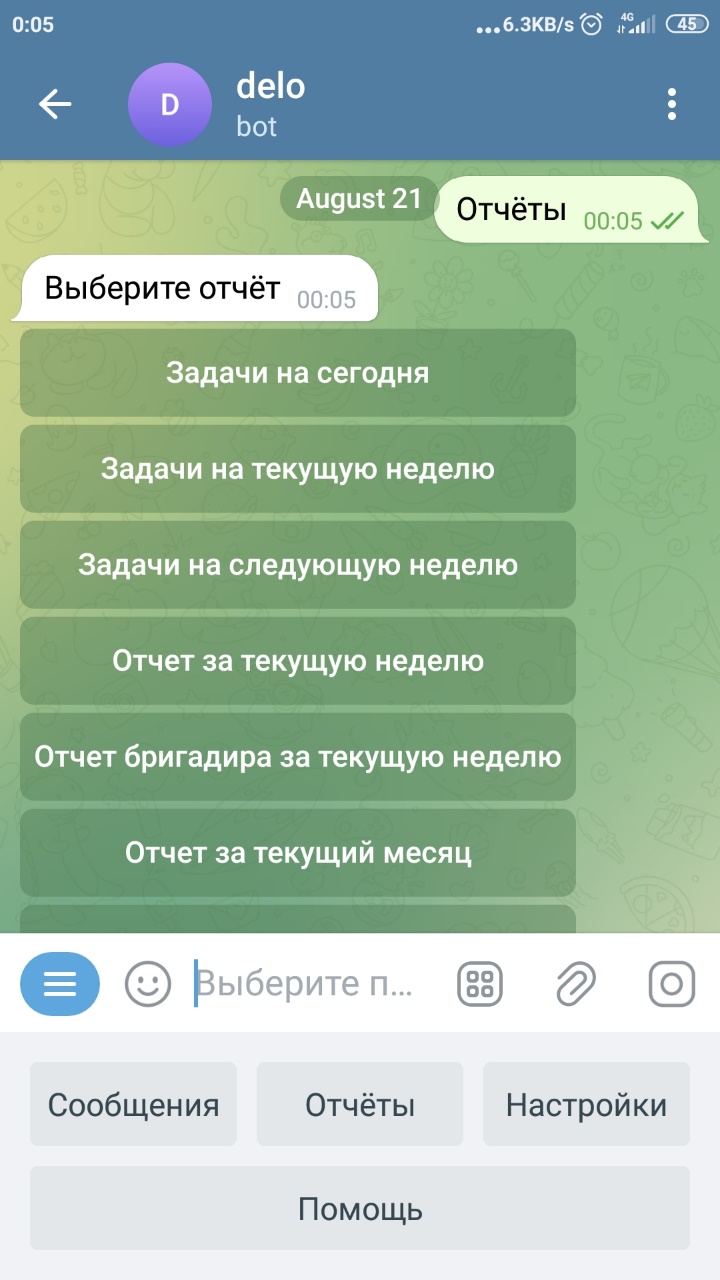


Рисунок 11. Меню «Отчёты»

3.4. Механизм запросов и ответов модуля Telegram-bot

Информационный обмен между модулями системы организован с использованием управляющей структуры Tokens. Каждый модуль системы получает Tokens в виде аргумента, модифицирует её в соответствии с результатом своей работы и возвращает обратно в вызывающий модуль. В конечном итоге модуль tokens на основе данных управляющей структуры Tokens формирует сообщение пользователю (текст, кнопки, файлы, изображения и т.д.), которое впоследствии отправляется в чат Telegram.

Управляющая структура Tokens представляет собой словарь Python:

Tokens = dict()  
Tokens['chat\_id'] – id chat telegram,Tokens['user'] = dict() – данные пользователя:

Tokens['user']['id'] – telegram user id,  
Tokens['user']['first\_name'] – имя пользователя,  
Tokens['user']['last\_name'] – фамилия пользователя,  
Tokens['user']['username'] – никнейм пользователя;

Tokens['request'] = dict() – запрос пользователя:

Tokens['request']['type'] - тип запроса (сообщение или callback\_query),  
Tokens['request']['id'] – message\_id,  
Tokens['request']['date'] – дата сообщения,  
Tokens['request']['text'] – текст сообщения,  
Tokens['request']['data'] – данные сообщения (в случае нажатия пользователем клавиши InlineKeyboard)

Tokens['answer'] = dict() – ответ на запрос пользователя:

Tokens['answer']['text'] – текст ответа,

Tokens['answer']['parse\_mode'] – режим вывода ответа (text / html),

Tokens['answer']['bottomKeys'] – меню ReplyKeyboard,

Tokens['answer']['img'] – ссылки на графические файлы для вывода,

Tokens['answer']['keys'] – список кнопок InlineKeyboard,

Tokens['answer']['currentkeyrow'] – ряд кнопок InlineKeyboard,

Tokens['answer']['keyboard'] – клавиатура (Inline / Reply).

3.5. Шаблоны отчётов

Для сбора и вывода пользователям информации из базы данных о задачах в системе реализовано хранение шаблонов отчётов. Каждая новая запись таблицы entity с типом subject (момент регистрации пользователя в системе) получает в поле parameters json объект, содержащий шаблоны отчётов. Каждый шаблон содержит информацию о работах (subject и group), типе отчёта, его названии, периоде сбора и варианте вывода.

"reports": [

{

"jobs": [

{

"subject": [

173

]

}

],

"type": "list",

"title": "Задачи на сегодня",

"period": "d",

"variant": "plan"

},

{

"jobs": [

{

"subject": [

173

]

}

],

"type": "list",

"title": "Задачи на текущую неделю",

"period": "w",

"variant": "plan"

},

{

"jobs": [

{

"subject": [

173

]

}

],

"type": "list",

"title": "Задачи на следующую неделю",

"period": "+w1",

"variant": "plan"

},

{

"jobs": [

{

"subject": [

173

]

}

],

"type": "status",

"title": "Отчет за текущую неделю",

"period": "w",

"variant": "all"

},

{

"jobs": [

{

"group": [

201

],

"subject": [

173

]

}

],

"type": "brigadier",

"title": "Отчет бригадира за текущую неделю",

"period": "w",

"variant": "all"

},

{

"jobs": [

{

"subject": [

173

]

}

],

"type": "status",

"title": "Отчет за текущий месяц",

"period": "m",

"variant": "all"

},

{

"jobs": [

{

"subject": [

173

]

}

],

"type": "line",

"title": "Отчет-график за текущий месяц",

"period": "m",

"variant": "all"

},

{

"jobs": [

{

"subject": [

173

]

}

],

"type": "status",

"title": "Отчет за прошлую неделю",

"period": "-w1",

"variant": "all"

},

{

"jobs": [

{

"subject": [

173

]

}

],

"type": "status",

"title": "Отчет за прошлый месяц",

"period": "-m1",

"variant": "all"

},

{

"jobs": [

{

"subject": [

173

]

}

],

"type": "line",

"title": "Отчет-график за прошлый месяц",

"period": "-m1",

"variant": "all"

},

{

"jobs": [

{

"group": [

3

]

}

],

"type": "bubble",

"title": "График бригадира за текущий месяц",

"period": "m",

"variant": "fact"

},

{

"jobs": [

{

"group": [

201

],

"subject": [

173

]

}

],

"type": "equipment",

"title": "Отчет по оборудованию",

"period": "m",

"variant": "all"

},

{

"jobs": [

{

"group": [

3

],

"subject": [

173

]

}

],

"type": "equipment\_bubble",

"title": "График по оборудованию",

"period": "m",

"variant": "fact"

}

],

На основании значения ключа “type” обработчик action\_reports определяет тип отчёта, получает из поля parameters таблицы entity базы данных параметры запрашиваемого отчёта и вызывает модуль-обработчик соответствующего отчёта.

Всего по состоянию на конец октября 2023 года в системе предусмотрено четыре типа отчётов:

1. Отчёт типа list;
2. Отчёт типа status;
3. Отчёт типа line;
4. Отчёты бригадира, по оборудованию.

3.5.1. Отчёт типа list

Отчёт типа list служит для сбора и отображения информации о назначенных и невыполненных на момент запроса задачах пользователя. Отчёт отображается в виде запрограммированных кнопок InlineKeyboard. Период сбора задач определяется параметром шаблона period (день, неделя, месяц и т.д.). Работники (группы работников), попадающие в выборку отчёта, определяются параметром отчёта jobs.

При нажатии на кнопку какой-либо работы выводится подробная информация и возможность отметить её выполнение, сообщить о проблеме и назначить другого работника (при наличии соответствующих прав у текущего пользователя).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Рисунок 12. Варианты вывода отчёта типа list

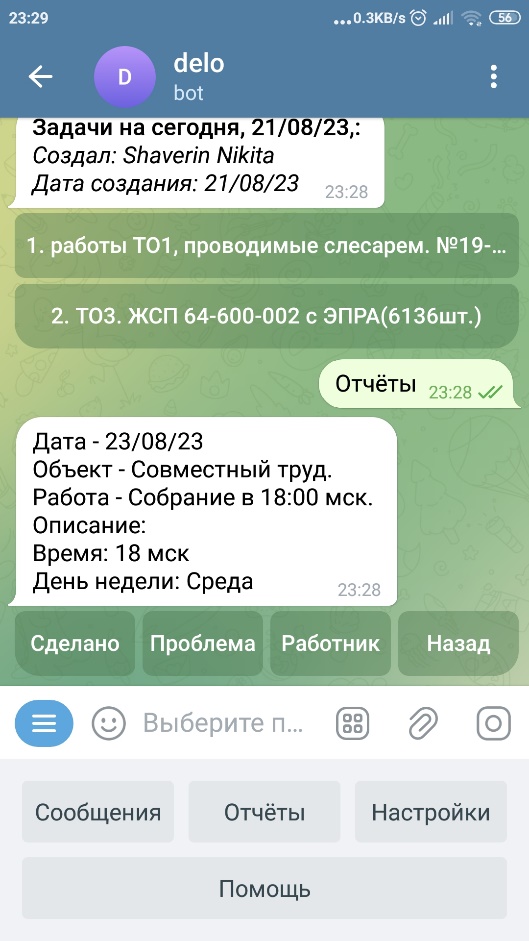


Рисунок 13. Информация о назначенной работе

3.5.2. Отчёт типа status

Отчёт типа status предназначен для сбора и отображения статистической информации о задачах и их статусах на определённый период времени. Отчёт отображается в текстовом режиме в виде таблицы. Параметры сбора отчёта определяются его шаблоном.

|  |  |
| --- | --- |
|  |  |

Рисунок 14. Варианты вывода отчёта типа status

3.5.3. Отчёт типа line

Отчёт типа line предназначен для сбора и отображения информации о назначенных задачах и их выполнении за определённый период времени. Отчёт отображается в виде графика. Задачи к исполнению отображаются синим цветом (в узловых точках плановые даты выполнения и количество задач), фактическое выполнение поставленных задач отображается зелёным цветом (в узловых точках фактические даты выполнения и количество выполненных задач). Параметры сбора отчёта определяются его шаблоном.

|  |  |
| --- | --- |
|  |  |

Рисунок 15. Варианты вывода отчёта типа line

3.5.4. Отчёты бригадира, по оборудованию

Отчёт бригадира предназначен для сбора и отображения информации о задачах всех работников, объединённых в общую группу (бригаду). Отчёт выводится для пользователей, входящих в список бригадиров группы. Отчёт бригадира представлен в двух вариантах: текстовом (разновидность отчёта типа status) и в виде пузырьковой диаграммы (тип bubble).

Отчёт по оборудованию предназначен для сбора и отображения информации об оборудовании с просроченными датами технического обслуживания. Представлен в двух вариантах: текстовом (разновидность отчёта типа status) и в виде пузырьковой диаграммы (тип bubble).

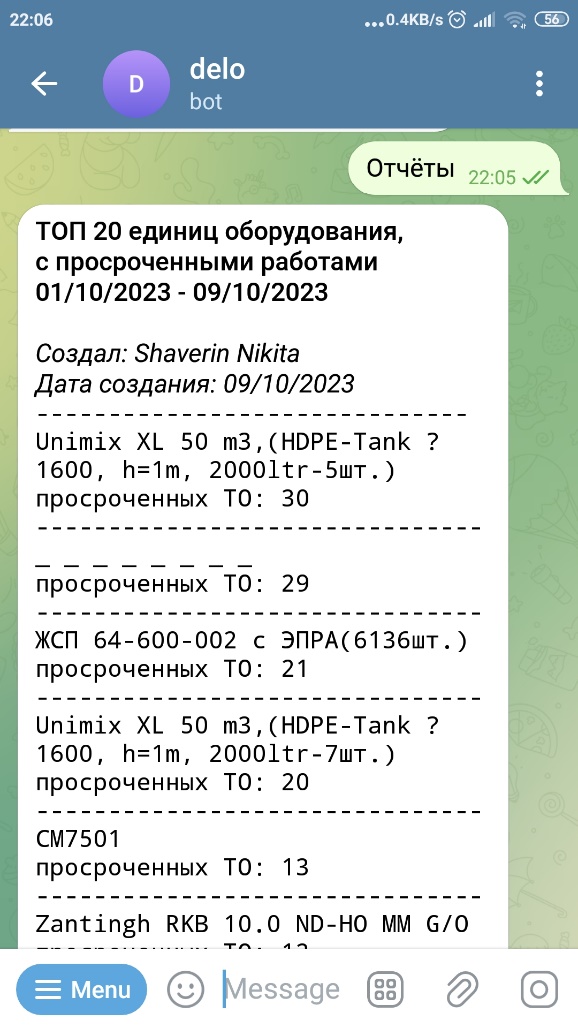


Рисунок 16. Вариант вывода отчёта по оборудованию

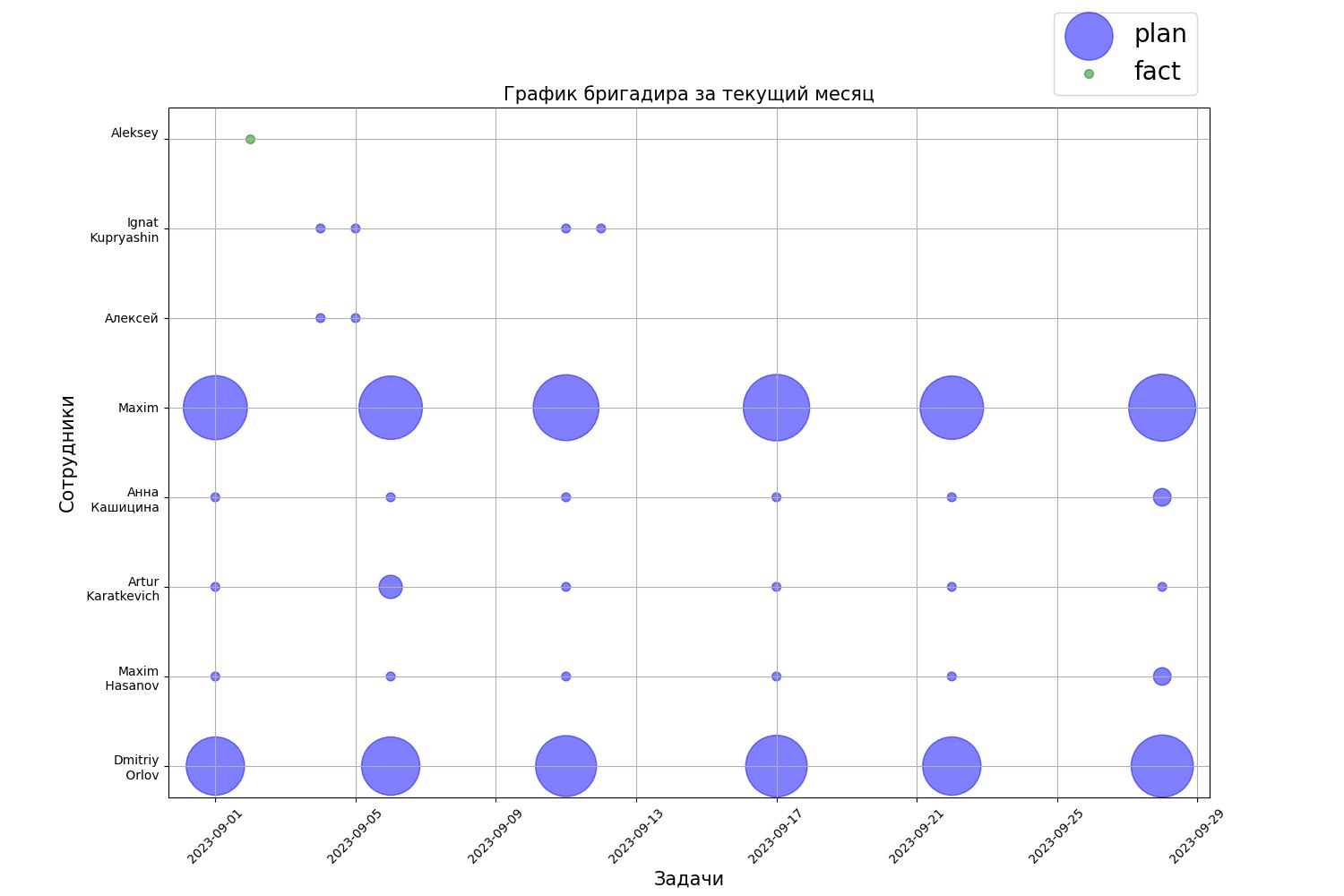


Рисунок 17. Вариант вывода отчёта бригадира

3.6. Обмен сообщениями между пользователями системы

Кроме стандартного обмена текстовыми сообщениями между пользователями Telegram, в telegram-боте реализована возможность отправки и сохранения файлов (photo, voice, audio, document). При получении от пользователя какого-либо файла, telegram-бот определяет его тип, именует в соответствии с telegram\_id пользователя, текущей датой и временем и сохраняет на сервере. При наличии соответствующей подписи к файлу последний может быть переслан указанному в подписи пользователю.

|  |  |
| --- | --- |
|  |  |

Рисунок 18. Пересылка файлов

Также в telegram-боте (в режиме отчёта типа list) реализована возможность рассылки уведомлений выполнении поставленной задачи и о проблемах при выполнении поставленной задачи. Данные уведомления рассылаются только пользователям, входящим в соответствующий список рассылки сообщений (списки problem\_subscribers и brig\_subscribers таблицы entity / parameters / options у entity типа group).

|  |  |  |
| --- | --- | --- |
|  |  |  |

Рисунок 19. Уведомления пользователям о проблеме, выполнении задачи

При наличии необходимых прав пользователь имеет возможность назначить иного исполнителя (из членов своей группы) для своей задачи. Данный функционал также реализован в режиме отчёта типа list. При этом в таблице базы данных plan по соответствующему id задачи меняется поле subjectid. После внесений изменений в базу данных новому исполнителю приходит уведомление о назначении задачи и плановом сроке её выполнения.

|  |  |
| --- | --- |
|  |  |

Рисунок 20. Переназначение задачи другому пользователю

3.7. Автоматическая рассылка списка задач

В модуле week\_tasks в системе реализована возможность автоматической рассылки каждому пользователю списка задач на предстоящую неделю (период может изменяться). Список дел по всем подчинённым сотрудникам, входящим в группу, рассылается руководителю (бригадиру) и иным subject, включённым в список brigadiers. Данный функционал реализован с помощью планировщика задач cron.

|  |  |
| --- | --- |
|  |  |

Рисунок 21. Автоматическая рассылка списка задач

4. Заключение

4.1. Выводы, полученные во время анализа теоретической базы и практического исследования

Современный эффективный менеджмент невозможен без применения автоматизированных систем управления задачами, дающих руководителю четкое представление о качестве работы персонала компании, статусе выполнения задач в режиме реального времени. В связи с усилившимся санкционным давлением ряда недружественных России стран рынок доступного программного обеспечения заметно оскудел, что подогревает интерес бизнеса к данному направлению.

Разработка современного ПО должна использовать уже имеющиеся проверенные паттерны проектирования. Использование многоуровневой интернет-архитектуры существенно упрощает процесс внедрения, сопровождения и развития информационной системы.

Использование в разработке бэкенд и телеграмм-бота языка программирования Python (и его фреймворков) позволило упростить и ускорить процесс разработки ПО.

4.2. Оценка проекта, описание его результатов

Совместными усилиями команды разработчиков была разработана MVP (minimal viable product) версия продукта, состоящая из двух представлений (веб-версия и telegram-бот), модуля бизнес-логики и базы данных (с расширенным функционалом). В системе реализован следующий функционал:

* Регистрация/авторизация пользователей;
* Заполнение базы данных сущностями (объект, работа, субъект);
* Карточка сущности, построение иерархии сущностей;
* Разграничение прав сущностей и их группировка;
* Создание шаблонов отчётов;
* Настройка и отображение различных вариантов отчётов типа list, status, line, brigadier;
* Рассылка уведомлений и напоминаний о назначении, проблеме и выполнении задач;
* Автоматическая рассылка уведомления о назначенных задачах группам работников;
* Обмен сообщениями и файлами.

4.3. Общий итог работы, её практическая значимость

Осенью 2023 года MVP-версия информационной системы «Совместный труд» передана на апробацию в Новосибирский агрокомбинат «Сады гиганта» (Россия, Новосибирская область, рабочий посёлок Кольцово, 1/3, https:/sadygiganta.ru). На балансе агрокомбината числится несколько сотен единиц промышленного оборудования и производственных сооружений, требующих постоянного периодического регламентного технического обслуживания. Команда проекта планирует автоматизировать процесс контроля проведения регламентных работ, проводить периодический сбор статистической информации о проведённых работах и загрузке обслуживающего персонала для минимизации затрат по обслуживанию и рисков несвоевременного проведения регламентных работ.

4.4. Предложения по совершенствованию информационной системы «Совместный труд»

Вывести новый продукт на рынок России, привлечь к тестированию 2-3 компании. Закрепиться на рынке, найти заказчика, готового платить за развитие продукта.

В следующих релизах увеличить использование объектно-ориентированного подхода программирования (значения в Tokens, обработка запросов от пользователя и др.)

Использовать открытую крипто-платформу Telegram Mini Apps для повышения функциональности и информативности telegram-бота.

Для упрощения работы и расширения функционала использовать последнюю версию фреймворка pyTelegramBotAPI 4.14.0. с последующим переходом на асинхронный фреймворк aiogram.

Для автоматизации ввода данных в информационную систему из баз данных потенциальных заказчиков реализовать информационный обмен с системой 1С.

5. Список используемой литературы

1. <https://кц-прогресс.рф/history.html>
2. <https://bitcop.ru/blog/kontrol-vypolnenija-zadach-sotrudnikami>
3. <https://sisu.ut.ee/basic_of_project_work/>
4. <https://www.redmine.org/>
5. <http://jira.ru/>
6. <https://www.bitrix24.ru/>
7. <https://subscribe.kck.ru/#/st>
8. <https://a2b.su/>
9. <https://aspro.cloud/>
10. <https://gb.ru/blog/arhitektuta-sistemy/>
11. Рогозов Ю.И., Свиридов А.С., Кучеров С.А. Архитектура информационных систем: учебное пособие. – Ростов-на-Дону: Изд-во ЮФУ, 2014. – 117 c.
12. <https://spravochnick.ru/bazy_dannyh/bazy_dannyh_vvedenie/arhitektura_informacionnoy_sistemy/>
13. <https://planny24.ru/>
14. <https://megaplan.ru/>
15. <https://www.atlassian.com/ru/agile/scrum>
16. <https://www.atlassian.com/ru/agile/kanban>
17. <https://pypi.org/project/pyTelegramBotAPI/>
18. <https://core.telegram.org/bots/api>
19. <https://core.telegram.org/bots/webapps>

6. Приложения

6.1. Программный код модуля Telegram-bot, модуля управления базой данных

Код программных модулей информационной системы «Совместный труд» находится по адресу: <https://github.com/lesenkagit/sovtrud>.

Далее приводятся коды программных модулей, необходимых для работы telegram-бота и управления базой данных.

**telegramBot.py** (Д. Орлов, Н. Шаверин)

#!/usr/bin/python3  
  
import os  
import sys  
import signal  
import re  
import telebot  
import threading  
sys.path.append("../core")  
import core\_log as log  
import core\_config as cf  
import housekeeper  
import worker  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 #  
 # Signal to Ignore  
 # if parent is not INIT  
 #  
  
 # COMMENT  
 if os.getppid() != 1:  
 signal.signal( signal.SIGTTOU, signal.SIG\_IGN )  
 signal.signal( signal.SIGTTIN, signal.SIG\_IGN )  
 signal.signal( signal.SIGTSTP, signal.SIG\_IGN )  
  
 config = cf.configuration()  
  
  
 # COMMENT  
 os.chdir( config['common']['homedir'] + config['bot']['dir'])  
 os.umask(0x17)  
  
  
 #  
 #  
 # Demon  
 # fork new process  
 #  
  
  
 # COMMENT  
 try:  
 pid = os.fork()  
 if pid != 0:  
 #  
 # # close parent process  
 sys.exit(0)  
 except Exception as err:  
 log.log( "Can't fork() process: " + str(err), severity="error", facility="bot" )  
 sys.exit(1)  
  
  
 #  
 #  
 # Child process there  
 #  
 #  
  
 # COMMENT  
 pid = os.getpid()  
  
 # get lead group  
  
 # COMMENT  
 os.setsid()  
  
 log.log( "Daemon started", severity="info", facility="bot" )  
  
  
 #  
 # close all filedescriptor  
 #  
  
 # COMMENT  
 os.closerange( 0,1024 )  
  
  
 #  
 #  
 # Tokens struct  
 # Управляющая структура  
 #  
 Tokens = dict()  
  
  
 #-------- Подключаем бота ---------  
 #  
 #  
 bot = None  
  
 try:  
 bot = telebot.TeleBot( config['bot']['token'] )  
 log.log( "Connected to telegram chat.", severity="info", facility="bot" )  
  
 except Exception as err:  
 log.log( "Bot can't connected. It's runnig now may be. Error: "+str(err), severity="error", facility="bot" )  
 sys.exit(1)  
  
  
 #  
 # номер следующего обновления с сообщениями  
 # нужно указывать в функции get\_update при чтении обновлений с серевера  
 #  
 offset = 0  
  
 #  
 # Вечный цикл  
 # Infinity pooling  
 #  
 #  
 while True:  
  
  
 #  
 # Check client message  
 #  
 try:  
  
 messages = bot.get\_updates( offset=offset, timeout=int( config['bot']['update\_timeout']) )  
  
 except Exception as err:  
 log.log( "get update error: "+str(err), severity="error", facility="bot update" )  
  
  
 #  
 # Два пути:  
 # 1) Если новые сообщения пришли - запускаем Workers  
 # 2) Если новых сообщений нет - запускаем Домашнюю работу  
 #  
 #  
 if len(messages) > 0:  
  
  
  
 # Путь 1  
 #  
 # Заускаем Workers для каждого сообщения  
 #  
 #  
 for message in messages:  
  
 #  
 # New Thread  
 #  
 try:  
 newworker = threading.Thread( target=worker.start, args=(bot, message,) )  
 ## add in worker list  
 ##Worker\_threads.append( newworker )  
  
 # update offset  
 if offset <= message.update\_id:  
 offset = message.update\_id + 1  
  
 except Exception as err:  
 log.log( "Can't create Worker: "+str(err), severity="error", facility="bot" )  
  
  
 #  
 # Путь 2  
 # запускаем Домашних работников  
 #  
 #  
 else:  
  
 housekeeper.start( bot )

**worker.py** (Д. Орлов, М. Закиев, Д. Джурабаев, Н. Шаверин)

from telebot import types  
  
import json  
import re  
import datetime  
  
import sys  
sys.path.append('../core')  
import core\_log as log  
import core\_profile as p  
  
import tokens  
import route  
  
  
#  
# Start - пришло сообщение - обрабатываем и выдаём ответ  
# одно сообщение обрабатывает один процесс (worker). У worker одна входящя функция - Start  
#  
# на входе bot - id бота. Надо знать к какому боту мы привязаны, кому отправлять сообщения  
# message - сообщение от телеграм, со всеми параметрами (как документации)  
#  
# на выходе - функция ни чего не возвращает, но ответ готовит в token['answer']  
#  
def start( bot, message ):  
  
# log.log( str(Tokens['chat\_id']) + "worker.start", severity="debug", facility="worker" )  
  
 #  
 # 1 Парсим входящее сообщение  
 # 2 выполняем действия  
 # 3 Отправляем ответ  
 #  
 #  
  
 #  
 #  
 # 1 Парсим входящее сообщение  
 #  
 #  
 try:  
  
 #  
 # 1 Парсим входящее сообщение  
 #  
 Tokens = tokens.parse\_message( message )  
  
  
 # =============================================================================================================  
 # если пользователь отправил photo, voice, audio, document, то получаем, именуем и раскладываем по директориям согласно типу файла  
 try:  
 # print(f'message - {message}')  
 if message.message:  
 # photo  
 if message.message.photo:  
 path\_photo = '../received\_files/photo/'  
 path\_photo += str(Tokens['chat\_id'])  
 path\_photo += datetime.datetime.now().strftime('\_%Y\_%m\_%d\_%H\_%M\_%S')  
  
 fileID = message.message.photo[-1].file\_id  
 file\_info = bot.get\_file(fileID)  
  
 file\_ext = file\_info.file\_path[-4:]  
 path\_photo += file\_ext  
  
 downloaded\_file = bot.download\_file(file\_info.file\_path)  
 with open(path\_photo, 'wb') as new\_file:  
 new\_file.write(downloaded\_file)  
 Tokens['request']['text'] = 'file\_delivered'  
 # voice  
 elif message.message.voice:  
 path\_voice = '../received\_files/voice/'  
 path\_voice += str(Tokens['chat\_id'])  
 path\_voice += datetime.datetime.now().strftime('\_%Y\_%m\_%d\_%H\_%M\_%S')  
  
 fileID = message.message.voice.file\_id  
 file\_info = bot.get\_file(fileID)  
  
 file\_ext = file\_info.file\_path[-4:]  
 path\_voice += file\_ext  
  
 downloaded\_file = bot.download\_file(file\_info.file\_path)  
 with open(path\_voice, 'wb') as new\_file:  
 new\_file.write(downloaded\_file)  
 Tokens['request']['text'] = 'file\_delivered'  
 # audio  
 elif message.message.audio:  
 path\_audio = '../received\_files/audio/'  
 path\_audio += str(Tokens['chat\_id'])  
 path\_audio += datetime.datetime.now().strftime('\_%Y\_%m\_%d\_%H\_%M\_%S')  
  
 fileID = message.message.audio.file\_id  
 file\_info = bot.get\_file(fileID)  
  
 file\_ext = file\_info.file\_path[-4:]  
 path\_audio += file\_ext  
 # print(f'path\_audio - {path\_audio}')  
  
 downloaded\_file = bot.download\_file(file\_info.file\_path)  
 with open(path\_audio, 'wb') as new\_file:  
 new\_file.write(downloaded\_file)  
 Tokens['request']['text'] = 'file\_delivered'  
 # document  
 elif message.message.document:  
 path\_document = '../received\_files/document/'  
 path\_document += str(Tokens['chat\_id'])  
 path\_document += datetime.datetime.now().strftime('\_%Y\_%m\_%d\_%H\_%M\_%S')  
  
 file\_ext = message.message.document.file\_name[-4:]  
 path\_document += file\_ext  
  
 fileID = message.message.document.file\_id  
 file\_info = bot.get\_file(fileID)  
 downloaded\_file = bot.download\_file(file\_info.file\_path)  
 with open(path\_document, 'wb') as new\_file:  
 new\_file.write(downloaded\_file)  
 Tokens['request']['text'] = 'file\_delivered'  
  
 # # *TODO разобрать подпись к файлу. Пока можно сделать пересылку по полученному chat\_id или через посик в тексте caption* # # Если есть подпись к файлу  
 # if message.message.caption:  
 # try:  
 # print(f'Подпись к файлу: {message.message.caption = }')  
 # input\_caption = message.message.caption.lower()  
 # if 'brig' in input\_caption or 'бриг' in input\_caption:  
 # # *TODO можно сделать выборку получателей из БД. Отдельной функцией. Пригодится для каждого типа файлов.* # # пока стоит мой telegram\_id  
 # chat\_id\_to\_send = 5107502329  
 # caption = f'Файл от пользователя {Tokens["chat\_id"]}'  
 # # img = dict()  
 # # img.update({'path': '../img/work\_in\_progress.gif'})  
 # # img.update({'caption': 'Ведутся работы по настройке сервиса'})  
 # # Tokens = tokens.update\_answer(Tokens, add\_img=img)  
 # print(f'{chat\_id\_to\_send = }')  
 # # print(f'{path\_photo = }')  
 # # print(f'{caption = }')  
 # # *TODO сделать пересылку всех типов файлов (фото, аудио, голос и документ)* # with open(path\_photo, 'rb') as file\_to\_send:  
 # # res\_send\_photo = bot.send\_photo(chat\_id=int(Tokens['chat\_id']), photo=img, caption=caption, protect\_content=False)  
 # res\_send\_photo = bot.send\_photo(chat\_id=chat\_id\_to\_send, photo=file\_to\_send, caption=caption, protect\_content=False)  
 # # res\_send\_document = bot.send\_document(chat\_id=chat\_id, document=file\_to\_send, caption=caption, protect\_content=False)  
 # except Exception as e:  
 # print(f'Ошибка {e = }')  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Save user file: " + str(err), severity="error", facility="worker")  
 # =============================================================================================================  
  
 # Если пришёл ответ с кнопок (callback) - удаляем InlineKeyboard вместе с заголовком  
 if 'request' in Tokens and 'type' in Tokens['request'] and Tokens['request']['type'] == 'callback\_query':  
 tokens.del\_inline\_keyboard(bot, Tokens)  
  
 #  
 # Find user\_id by telegramId  
 #  
 try:  
 user\_id = p.get\_id\_by\_telegramid( Tokens['user']['id'] )  
  
 if user\_id:  
 Tokens.update( {"user\_id": user\_id } )  
 except Exception as err:  
 log.log( str(Tokens['chat\_id']) + " Can't define user: "+str(err), severity="error", facility="worker" )  
  
 #  
 # Посмотрим, что получилось  
 text = json.dumps( Tokens, indent=4 )  
 log.log( str(Tokens['chat\_id']) + "\tTokens: "+text, severity="debug", facility="worker" )  
  
  
 except Exception as err:  
 log.log( str(Tokens['chat\_id']) + " Parsing message: "+str(err), severity="error", facility="worker" )  
  
 #  
 #  
 # 2 Выполняем действия  
 # Отправляем router для поиска нужной функции  
 #  
 try:  
 #  
 # Отправляем запрос на отработку  
 #  
 Tokens = route.route(Tokens )  
  
 except Exception as err:  
 log.log( str(Tokens['chat\_id']) + " Prepare answer: "+str(err), severity="error", facility="worker" )  
  
  
  
  
 #  
 # 3 Отправляем ответ  
 #  
 #  
 #  
 try:  
  
 tokens.send\_answer( bot, Tokens )  
  
 except Exception as err:  
 log.log( str(Tokens['chat\_id']) + " Send answer: "+str(err), severity="error", facility="worker" )

**housekeeper.py** (Д. Орлов, М. Демченко, Н. Шаверин)

import sys  
  
import telebot  
sys.path.append('../core')  
import core\_log as log  
import core\_message as msg  
from time import sleep  
  
  
def start(bot):  
 *"""Housekeeper выбирает из таблицы все неотправленные сообщения и отправляет, заполняя поле sended """*# log.log( "I am ready.", severity="info", facility="housekeeper" )  
 try:  
 mid = list()  
 # получить из БД все сообщения where plandate is not null and senddate is null  
 messages = msg.message\_to\_sent()  
  
 if len(messages) != 0:  
 # проходим по каждому сообщению  
 for mes in messages:  
  
 # новое решение  
 # если есть telegram\_id, отправляем сообщение  
 if mes[0]:  
 try:  
 # отправляем сообщение  
  
 # если длина текстового сообщения > 4096 символов (ограничение телеграмма), то разбиваем на несколько сообщений по 4096 символов каждое  
 if len(mes[2]) > 4096:  
 for x in range(0, len(mes[2]), 4096):  
 bot.send\_message(chat\_id=mes[0], text=mes[2][x:x + 4096], parse\_mode='html')  
 else:  
 bot.send\_message(chat\_id=mes[0], text=mes[2], parse\_mode='html')  
  
 # если сообщение отправилось, добавляем его для пометки как "отправлено"  
 sleep(1)  
 mid.append(mes[1])  
 except Exception as err:  
 log.log("Housekeeper can`t send message, because " + str(err), severity="error", facility="housekeeper")  
 # если бот не смог отправить сообщение из-за chat not found (чат не найден) или bot was blocked by the user (бот заблокирован пользователем), то удаляем это сообщение из БД  
 if 'error code: 400' in str(err).lower() or 'error code: 403' in str(err).lower():  
 # if str(err) == 'A request to the Telegram API was unsuccessful. Error code: 400. Description: Bad Request: chat not found' or str(err) == 'A request to the Telegram API was unsuccessful. Error code: 403. Description: Forbidden: bot was blocked by the user':  
 msg.message\_delete([mes[1]])  
 # иначе удаляем из БД сообщение без telegram\_id  
 else:  
 msg.message\_delete([mes[1]])  
 # После отправки сообщений помечаем сообщения как отправленные (те, что попали в список).  
 # Неудачные отправки будут продублированы в следующем сеансе работы housekeeper  
 try:  
 msg.message\_sent(mid)  
 except Exception as err:  
 log.log("Housekeeper can`t mark message as sent, because " + str(err), severity="error", facility="housekeeper")  
  
 return True  
 else:  
 return False  
 except Exception as err:  
 log.log("Housekeeper not working, because " + str(err), severity="error", facility="housekeeper")

**tokens.py** (Д. Орлов, М. Демченко, Н. Шаверин)

#  
# Tokens - структура содержащая поля для обработки одного запроса от пользователя  
#  
# мы будем передавать её из функции в функцию - необходимо привести поля к единой маленькой структуре   
#  
#  
# структура не содержит истории или состояния пользователя  
# она содержит только текущий запрос и поля для составления ответа  
#  
# Структура должна приводиться к единому виду, например  
# Сообщение с обычным текстом:  
# {  
# "chat\_id": 414366402,  
# "user": {  
# "id": 414366402,  
# "first\_name": "Dmitriy",  
# "last\_name": "Orlov",  
# "username": "gisplan"  
# },  
# "request": {  
# "type": "text",  
# "id": 2278,  
# "date": 1674030934,  
# "text": "hello"  
# },  
# "answer": {}  
# }  
#  
# Или сообщение + нажатая кнопка  
# кнопка ограничена по длине, поэтому содержит только хэш (data). По хэшу вытаскиваются параметры, которые бот хотел передать (сам себе). Параметры готовятся и сохраняются в специальной таблице, то следующего запроса.  
# {  
# "chat\_id": 414366402,  
# "user": {  
# "id": 414366402,  
# "first\_name": "Dmitriy",  
# "last\_name": "Orlov",  
# "username": "gisplan",  
# "is\_user\_registered": True  
  
  
# },  
# "request": {  
# "type": "callback\_query",  
# "id": 2280,  
# "date": 1674030934,  
# "data": "ttyew67728",  
# "text": "Keys",  
# "keys": [  
# {  
# "callback\_data": "ttyew67728",  
# "keytext": "hello"  
# }  
# ]  
# },  
# "answer": {}  
# }  
#  
#  
# Итого поля в Tokens  
#  
# chat\_id - кому отправлять ответ  
# user - информация о пользователе  
# request - выжимка из message с параметрами запроса  
# answer - сюда мы готовим ответ  
#  
  
  
from telebot import types  
  
import json  
import re  
  
import sys  
  
sys.path.append('../core')  
import core\_log as log  
import keyscallback as keycb  
import os  
  
  
#  
#  
# Parse - Получить Tokens из message.   
#  
#  
#  
#  
def parse\_message(message):  
 #  
 # Приведём входные параметры от пользователя к единой структуре  
 #  
 Tokens = dict()  
 Tokens['chat\_id'] = None  
 Tokens['user'] = dict()  
 Tokens['request'] = dict()  
 Tokens['answer'] = dict()  
  
  
 try:  
  
 #  
 #  
 # Если пришло нажатие с кнопки  
 #  
 #  
  
  
 if message.callback\_query:  
 try:  
 #  
 # разберём общую часть - пользователь  
 #  
 Tokens['user']['id'] = message.callback\_query.from\_user.id  
 Tokens['user']['first\_name'] = message.callback\_query.from\_user.first\_name  
 Tokens['user']['last\_name'] = message.callback\_query.from\_user.last\_name  
 Tokens['user']['username'] = message.callback\_query.from\_user.username  
  
 # None меняем на ''  
 if not Tokens['user']['first\_name']:  
 Tokens['user']['first\_name'] = ''  
 if not Tokens['user']['last\_name']:  
 Tokens['user']['last\_name'] = ''  
  
 # print(f"{Tokens['user']['last\_name']} {Tokens['user']['first\_name']}")  
  
  
 #  
 # Чат  
 #  
 Tokens['chat\_id'] = message.callback\_query.message.chat.id  
  
 #  
 # Тип сообщения  
 #  
 Tokens['request']['type'] = 'callback\_query'  
 Tokens['request']['id'] = message.callback\_query.message.id  
 Tokens['request']['date'] = message.callback\_query.message.date  
 Tokens['request']['text'] = message.callback\_query.message.text  
 Tokens['request']['data'] = message.callback\_query.data  
 callback\_txt = keycb.decode(Tokens, Tokens['request']['data'])  
 if callback\_txt:  
 Tokens['request']['data'] = callback\_txt  
  
 except Exception as err:  
 log.log("\tParse callback\_query " + str(err), severity="error", facility="tokens")  
  
 #  
 # Если пришёл текст  
 # start <hash> - приходит как текст  
 #  
 else:  
 #  
 # разберём общую часть - пользователь  
 #  
 Tokens['user']['id'] = message.message.from\_user.id  
 # # для теста регистрации нового пользователя  
 # Tokens['user']['id']=777777  
 Tokens['user']['first\_name'] = message.message.from\_user.first\_name  
 Tokens['user']['last\_name'] = message.message.from\_user.last\_name  
 Tokens['user']['username'] = message.message.from\_user.username  
  
 # None меняем на ''  
 if not Tokens['user']['first\_name']:  
 Tokens['user']['first\_name'] = ''  
 if not Tokens['user']['last\_name']:  
 Tokens['user']['last\_name'] = ''  
  
  
 #  
 # Чат  
 #  
 Tokens['chat\_id'] = message.message.chat.id  
  
 #  
 # Тип сообщения  
 #  
 Tokens['request']['type'] = 'text'  
 Tokens['request']['id'] = message.message.id  
 Tokens['request']['date'] = message.message.date  
 Tokens['request']['text'] = message.message.text  
  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + "Parsing message: " + str(err), severity="error", facility="tokens")  
  
 return Tokens  
  
  
# Ограничения:  
# Формат файла - png (решение принято на собрании 18.03.2023 года  
# Максимальный размер файла - 10 Мб (API Telegram)  
# Суммарная ширина и высота изображения - не более 10000  
# Соотношение ширины и высоты изображения - не более 20  
# Описание изображения - строка длиной не более 1024 символа  
# ВНИМАНИЕ, в будущем будет реализовано удаление изображения из директории /sovtrud/img после его вывода в ТГ-бот  
def update\_answer(Tokens, update\_text=None, update\_parse\_mode=None, add\_menu=None, add\_key\_text=None,  
 add\_key\_callback=None, add\_key\_row=False, clear\_all=False, add\_img=None):  
 #  
 # Clear All  
 # удаляем весь answer и создаём пустой  
 #  
 if clear\_all == True:  
 del (Tokens['answer'])  
 Tokens['answer'] = dict()  
  
 #  
 # update text  
 # пока ограничим 2048 символов  
 if update\_text != None:  
 tmp = ''  
 try:  
 tmp = Tokens['answer']['text'] + update\_text  
 except Exception:  
 tmp = update\_text  
 Tokens['answer'].update({'text': tmp[0:2048]})  
  
 #  
 # update parse\_mode  
 #  
 if update\_parse\_mode != None:  
 Tokens['answer'].update({'parse\_mode': update\_parse\_mode})  
  
 #  
 # Add keyboard  
 #  
 if add\_menu != None:  
 Tokens['answer'].update({'bottomKeys': add\_menu}) # строки кнопок  
  
 #  
 # Add img  
 #  
 if add\_img != None:  
 path = add\_img['path']  
 # print(f'Добавил путь к img {path}')  
 Tokens['answer'].update({'img': add\_img}) # добавляем img  
  
 #  
 # Add Inline keyboard  
 #  
 if add\_key\_text != None:  
  
 try:  
 #  
 # Если кнопки ещё не созданы создадим новые  
 #  
 if 'keys' not in Tokens['answer']:  
 Tokens['answer'].update({'keys': list()}) # строка для кнопок  
 Tokens['answer']['keys'].append(list()) # массив кнопок в строке  
 Tokens['answer'].update({  
 'currentkeyrow': 0}) # количество строк с кнопками (кнопки всегда будем добавлять в последнюю строку)  
  
 #  
 # Если требуют создать новую строку  
 # но, если в текущей строке нет кнопок, то новую строку не добавляем  
 #  
 if add\_key\_row == True:  
 # если в строке есть кнопки  
 if len(Tokens['answer']['keys'][Tokens['answer']['currentkeyrow']]):  
 Tokens['answer']['currentkeyrow'] = Tokens['answer']['currentkeyrow'] + 1  
 Tokens['answer']['keys'].append(list())  
  
 #  
 # Если есть callbak к кнопке добавляем  
 #  
 if add\_key\_callback != None:  
 #  
 # Запишем callback в БД, а вернувшийся hash поставим на callback кнопки. по возвращении расшифруем  
 #  
 callback\_hash = keycb.encode(Tokens, add\_key\_callback)  
 # callback\_hash = ''  
 if callback\_hash:  
 # если хэш вернулся, значит значение кнопки сохранено в бд  
 Tokens['answer']['keys'][Tokens['answer']['currentkeyrow']].append(  
 types.InlineKeyboardButton(add\_key\_text, callback\_data=callback\_hash))  
 else:  
 # если хэш пришёл пустой, то в БД ни чего не записалось. Оставим наш callback на кнопке  
 Tokens['answer']['keys'][Tokens['answer']['currentkeyrow']].append(  
 types.InlineKeyboardButton(add\_key\_text, callback\_data=add\_key\_callback))  
  
  
 #  
 # Добавляем кнопку с текстом без каллбека  
 #  
 else:  
 Tokens['answer']['keys'][Tokens['answer']['currentkeyrow']].append(  
 types.InlineKeyboardButton(add\_key\_text))  
  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Update answer. Add key " + str(add\_key\_text) + " callback " + str(  
 add\_key\_callback) + ": " + str(err), severity="error", facility="tokens")  
  
 #  
 # add Key to Keyboard  
 #  
 try:  
  
 Tokens['answer'].update({'keyboard': types.InlineKeyboardMarkup(Tokens['answer']['keys'])})  
  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Update answer. Prepare keyboard " + str(  
 Tokens['answer']['keys']) + ": " + str(err), severity="error", facility="tokens")  
  
  
 return Tokens  
  
  
#  
#  
#  
#  
#  
#  
  
#  
# Send Answer - отправка подготовленного сообщения  
#  
#  
#  
def send\_answer(bot, Tokens):  
 try:  
 # если не указан тип текста - html | markdown удалим тип текста  
 if 'parse\_mode' in Tokens['answer']:  
 if Tokens['answer']['parse\_mode'] != 'html' and Tokens['answer']['parse\_mode'] != 'markdown' and \  
 Tokens['answer']['parse\_mode'] != 'markdownv2':  
 Tokens['answer']['parse\_mode'] = None  
 else:  
 Tokens['answer']['parse\_mode'] = None  
  
 #  
 # Разный набор параметров отправки c кнопками и без  
 #  
 if 'keyboard' in Tokens['answer']:  
 try:  
 bot.send\_message(int(Tokens['chat\_id']), Tokens['answer']['text'],  
 parse\_mode=Tokens['answer']['parse\_mode'], reply\_markup=Tokens['answer']['keyboard'])  
 except Exception as err:  
 # если клавиатура получилась слишком большая  
 text\_message = f'<b>Не могу отобразить все ваши дела!\nИх слишком много!</b>'  
 bot.send\_message(int(Tokens['chat\_id']), text\_message, parse\_mode='html')  
 log.log(str(Tokens['chat\_id']) + " Send answer: " + str(err), severity="error", facility="tokens")  
  
  
 #  
 # Только текст  
 #  
  
  
 elif 'text' in Tokens['answer']:  
 # Tokens['answer']['text'] = 'x' \* 4097  
 # если длина текстового сообщения > 4096 символов (ограничение телеграмма), то разбиваем на несколько сообщений по 4096 символов каждое  
 if len(Tokens['answer']['text']) > 4096:  
 for x in range(0, len(Tokens['answer']['text']), 4096):  
 bot.send\_message(int(Tokens['chat\_id']), Tokens['answer']['text'][x:x + 4096],  
 parse\_mode=Tokens['answer']['parse\_mode'])  
 else:  
 bot.send\_message(int(Tokens['chat\_id']), Tokens['answer']['text'], parse\_mode=Tokens['answer']['parse\_mode'])  
  
 #  
 # Если нужны нижние кнопки  
 #  
 if 'bottomKeys' in Tokens['answer']:  
 if Tokens['answer']['bottomKeys'] == 'tgbot\_menu':  
 get\_tgbot\_menu(bot, Tokens)  
 elif Tokens['answer']['bottomKeys'] == 'registration\_menu':  
 get\_registration(bot, Tokens)  
  
 #  
 # Если есть img  
 #  
 if 'img' in Tokens['answer']:  
 path = Tokens['answer']['img']['path']  
 caption = Tokens['answer']['img']['caption']  
 with open(path, 'rb') as img:  
 res\_send\_photo = bot.send\_photo(chat\_id=int(Tokens['chat\_id']), photo=img, caption=caption, protect\_content=False)  
 # после вывода изображения его необходимо удалить с сервера  
 # пока закомментировано  
 # if res\_send\_photo:  
 # # print(f'Удаляю файл {path}')  
 # os.remove(path)  
  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Send answer: " + str(err), severity="error", facility="tokens")  
  
  
# Меню телеграм-бота. Формирует ReplyKeyboard (3+1 кнопки внизу, в два ряда)  
def get\_tgbot\_menu(bot, Tokens):  
 # Вместо надписи на кнопках можно вывести эмодзи или расположить кнопки в два ряда  
 btn\_message = types.KeyboardButton(text='Сообщения')  
 btn\_reports = types.KeyboardButton(text='Отчёты')  
 btn\_settings = types.KeyboardButton(text='Настройки')  
 btn\_help = types.KeyboardButton(text='Помощь')  
 tgbot\_menu = types.ReplyKeyboardMarkup(resize\_keyboard=True, row\_width=3, one\_time\_keyboard=False,  
 input\_field\_placeholder='Выберите пункт меню')  
 # Добавляем ряд кнопок  
 tgbot\_menu.row(btn\_message, btn\_reports, btn\_settings)  
 # Добавляем ряд кнопок  
 tgbot\_menu.row(btn\_help)  
 bot.send\_message(chat\_id=int(Tokens['chat\_id']), text='Используйте меню', reply\_markup=tgbot\_menu)  
 return ('get\_tgbot\_menu\_OK')  
  
  
# Регистрация. Формирует ReplyKeyboard (кнопка "Зарегистрировать" внизу)  
def get\_registration(bot, Tokens):  
 btn\_registration = types.KeyboardButton(text='Зарегистрировать')  
 tgbot\_menu = types.ReplyKeyboardMarkup(resize\_keyboard=True, row\_width=1, one\_time\_keyboard=False,  
 input\_field\_placeholder='Для регистрации нажмите кнопку ниже')  
 # Добавляем ряд кнопок  
 tgbot\_menu.row(btn\_registration)  
 bot.send\_message(chat\_id=int(Tokens['chat\_id']), text='Используйте меню', reply\_markup=tgbot\_menu)  
 return ('get\_registration\_OK')  
  
  
# Удаление InlineKeyboard и сообщения  
def del\_inline\_keyboard(bot, Tokens):  
 # После нажатия на кнопку # InlineKeyboard, клавиатуру можно полностью удалить.  
 bot.edit\_message\_reply\_markup(Tokens['chat\_id'], Tokens['request']['id'])  
 # Можно удалить полностью сообщение вместе с кнопками  
 bot.delete\_message(Tokens['chat\_id'], Tokens['request']['id'])

**route.py** (Д. Орлов, Н. Шаверин)

import json  
import re  
import sys  
  
sys.path.append('../core')  
sys.path.append('../img')  
import core\_log as log  
import core\_Entity as ce  
import core\_profile as cp  
import core\_pg as pg  
import tokens  
  
sys.path.append('actions')  
import action\_start  
import action\_date  
import action\_help  
import action\_reports  
  
sys.path.append('reports')  
import report\_list  
  
  
#  
#  
# Route - Выделяет из запроса action и выполняет, соответствующую функцию  
#  
# 1 Выделяем action и parameters из Tokens['request'] (из запроса) - для всех запросов одинаково  
# 2 Выполняем функцию, соответствующую action  
#   
  
  
def route(Tokens):  
 #  
 #  
 # 1 Выделим Action и выполним соответствующую функцию  
 #  
 #  
 action = get\_action\_from\_request(Tokens)  
  
 #  
 #  
 # Добавим пользователя, если его нет в БД  
 #  
 #  
  
 if 'user\_id' not in Tokens:  
 try:  
 user\_id = register\_new\_user(Tokens)  
 if user\_id:  
 log.log(str(Tokens['chat\_id']) + " registered new user: " + str(Tokens['user']['id']), severity="info",  
 facility="route")  
 Tokens.update({'user\_id': user\_id})  
 else:  
 log.log(str(Tokens['chat\_id']) + " Can't register new user: " + str(Tokens['user']['id']),  
 severity="warning", facility="route")  
  
 # После успешной регистрации  
 Tokens = tokens.update\_answer(Tokens, update\_text="Добро пожаловать!", add\_menu='tgbot\_menu')  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Can't register new user: " + str(err), severity="error",  
 facility="route")  
 # если пользователь есть  
 else:  
 #  
 # Проверяем action  
 # действие, которое пришло в запросе  
 #  
 try:  
  
 # START  
 if re.match(r'/start', action, flags=re.IGNORECASE):  
 #  
 # Start Menu  
 #  
 Tokens = tokens.update\_answer(Tokens, add\_menu='tgbot\_menu')  
 # Плюс report\_list с задачами на сегодня  
 Tokens = action\_start.action(Tokens)  
  
 # DATE  
 elif re.match(r'/date', action, flags=re.IGNORECASE):  
 Tokens = action\_date.action(Tokens)  
  
 # СООБЩЕНИЯ  
 elif re.match(r'.\*messages.\*|.\*сообщения.\*|\U0001F4E9', action, flags=re.IGNORECASE):  
 img = dict()  
 img.update({'path': '../img/work\_in\_progress.gif'})  
 img.update({'caption': 'Ведутся работы по настройке сервиса'})  
 Tokens = tokens.update\_answer(Tokens, add\_img=img)  
  
 # НАСТРОЙКИ  
 elif re.match(r'.\*settings.\*|.\*настройки.\*|\U00002699', action, flags=re.IGNORECASE):  
 img = dict()  
 img.update({'path': '../img/work\_in\_progress.gif'})  
 img.update({'caption': 'Ведутся работы по настройке сервиса'})  
 Tokens = tokens.update\_answer(Tokens, add\_img=img)  
  
 # ОТЧЁТЫ  
 elif re.match(r'.\*reports.\*|.\*отчёты.\*|\U0001F4C4', action, flags=re.IGNORECASE):  
 Tokens = action\_reports.action(Tokens)  
  
 # ПОМОЩЬ  
 elif re.match(r'.\*help.\*|.\*помощь.\*|\U00002753', action, flags=re.IGNORECASE):  
 # print('route.py - вызываю action\_help.action')  
 Tokens = action\_help.action(Tokens)  
  
 elif re.match(r'/time\_zone|.\*time\_zone.\*|.\*timezone.\*|.\*time.\*|.\*время.\*|.\*зона.\*|.\*пояс.\*', action, flags=re.IGNORECASE):  
 try:  
 time\_zone = int(Tokens['request']['text'].split()[1])  
 if time\_zone not in range(-12, 13):  
 time\_zone = 0  
 except Exception as err:  
 time\_zone = 0  
 log.log(str(Tokens['chat\_id']) + " Can't find time\_zone " + action + ": " + str(err), severity="error", facility="route")  
  
 # получить time\_zone из entities.parameters можно core\_profile.get\_parameters( Tokens['user\_id'], 'time\_zone' )  
 if ce.setParametersValue(Tokens['user\_id'], Tokens['user\_id'], 'time\_zone', time\_zone):  
 Tokens = tokens.update\_answer(Tokens, update\_text=f'Ваш часовой пояс изменён.\nТеперь он составляет {time\_zone} от времени сервера (МСК)')  
 else:  
 Tokens = tokens.update\_answer(Tokens, update\_text=f'Ведутся работы по настойке системы.\nФункция {action} скоро будет доступна. time\_zone = {time\_zone}')  
  
 # отправлен файл  
 elif action == 'file\_delivered':  
 Tokens = tokens.update\_answer(Tokens,update\_text="Файл доставлен.")  
  
 # если action не распознан  
 else:  
 Tokens = tokens.update\_answer(Tokens,  
 update\_text="Не понял команду " + action + ".\nНужно набрать команду, например /start\n\nНаберите /help для помощи\n")  
 # Выводим меню бота, если пользователь ввёл какую-то чепуху  
 #  
 # Start Menu  
 #  
 Tokens = tokens.update\_answer(Tokens, add\_menu='tgbot\_menu')  
  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Can't route to action " + action + ": " + str(err), severity="error",  
 facility="route")  
  
 return Tokens  
  
  
# Регистрация нового пользователя  
def register\_new\_user(Tokens):  
 # собираем имя  
  
 # тест, если у пользователя не заполнена фамилия или имя  
 first\_name = Tokens['user']['first\_name'] if ( Tokens['user']['first\_name'] and Tokens['user']['first\_name'] != None) else ''  
 last\_name = Tokens['user']['last\_name'] if ( Tokens['user']['last\_name'] and Tokens['user']['last\_name'] != None ) else ''  
 full\_name = first\_name + ' ' + last\_name  
  
  
 #собираем parameters\_telegram  
 parameters\_telegram = "{\"telegram\":{\"id\":" + str((Tokens['user']['id'])) + ",\"firstname\":\"" + first\_name + "\",\"lastname\":\"" + last\_name + "\",\"username\":\"\"}, "  
 # отчёты по умолчанию  
  
 # временная зона относительно Мск  
 time\_zone = "\"time\_zone\": 0}"  
  
  
 # параметры для регистрации  
 # parameters = parameters\_telegram + time\_zone + default\_reports  
 parameters = parameters\_telegram + time\_zone  
  
 user\_id\_db = cp.newuser(full\_name, parameters)  
  
 # записать user\_id и user\_gid в каждый отчёт пользователя и сохранить его в БД  
  
 user\_entity = ce.get(user\_id\_db, user\_id\_db)  
 user\_id = user\_entity[0]['id']  
  
# шаблоны отчётов пользователя при регистрации  
 report\_list = list()  
 report\_list.append("[")  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"list\", \"title\": \"Задачи на сегодня\", \"period\": \"d\", \"variant\": \"plan\"}}")  
 report\_list.append(",")  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"list\", \"title\": \"Задачи на текущую неделю\", \"period\": \"w\", \"variant\": \"plan\"}}")  
 report\_list.append(",")  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"list\", \"title\": \"Задачи на следующую неделю\", \"period\": \"+w1\", \"variant\": \"plan\"}}")  
 report\_list.append(",")  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"status\", \"title\": \"Отчет за текущую неделю\", \"period\": \"w\", \"variant\": \"all\"}}")  
 report\_list.append(",")  
 # отчёт бригадира  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"brigadier\", \"title\": \"Отчет бригадира за текущую неделю\", \"period\": \"w\", \"variant\": \"all\"}}")  
 report\_list.append(",")  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"status\", \"title\": \"Отчет за текущий месяц\", \"period\": \"m\", \"variant\": \"all\"}}")  
 report\_list.append(",")  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"line\", \"title\": \"Отчет-график за текущий месяц\", \"period\": \"m\", \"variant\": \"all\"}}")  
 report\_list.append(",")  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"status\", \"title\": \"Отчет за прошлую неделю\", \"period\": \"-w1\", \"variant\": \"all\"}}")  
 report\_list.append(",")  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"status\", \"title\": \"Отчет за прошлый месяц\", \"period\": \"-m1\", \"variant\": \"all\"}}")  
 report\_list.append(",")  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"line\", \"title\": \"Отчет-график за прошлый месяц\", \"period\": \"-m1\", \"variant\": \"all\"}}")  
 report\_list.append(",")  
 # График бригадира за текущий месяц  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"bubble\", \"title\": \"График бригадира за текущий месяц\", \"period\": \"m\", \"variant\": \"fact\"}}")  
 report\_list.append(",")  
 # Отчет по оборудованию  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"equipment\", \"title\": \"Отчет по оборудованию\", \"period\": \"m\", \"variant\": \"all\"}}")  
 report\_list.append(",")  
 # График по оборудованию  
 report\_list.append(f"{{\"jobs\": [{{\"subject\": [{user\_id}]}}], \"type\": \"equipment\_bubble\", \"title\": \"График по оборудованию\", \"period\": \"m\", \"variant\": \"fact\"}}")  
 # конец списка отчётов  
 report\_list.append("]")  
  
  
 default\_reports = "".join(report\_list)  
  
  
 # Записываем список отчётов  
 ce.setParametersValue(user\_id, user\_id, 'reports', default\_reports)  
  
  
 return user\_id\_db  
  
  
#  
# Из Tokens['request'] выбирает Action  
#  
# 1 Если callback\_data установлен, то нажали кнопку. Скорее всего data содержит json = {"action":"...",...}  
# 2 Если это текст, то возмём первое слово. С развитием будем по тексту понимать, что просят (Может быть ничего)  
#  
#  
def get\_action\_from\_request(Tokens):  
 #   
 action = None  
  
 try:  
 #  
 # Если callback указан (т.е. нажата кнопка), то action и параметры берём оттуда  
 #  
 if Tokens['request']['type'] == 'callback\_query':  
  
 #  
 # переводим параметры в json  
 #  
 try:  
 parameters = json.loads(Tokens['request']['data'])  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Can't load action from callback " + str(err), severity="error",  
 facility="route")  
  
 #  
 # выделяем среди параметров action  
 #  
 action = parameters['action']  
  
 # от пользователя пришёл файл  
 elif Tokens['request']['text'] == 'file\_delivered':  
 action = 'file\_delivered'  
  
 # от пользователя пришёл файл, но возникла ошибка его сохранения  
 elif Tokens['request']['text'] == None:  
 action = 'file\_sent'  
 #  
 # Иначе пришёл текст  
 #  
 else:  
  
 #  
 # делим строку на параметры  
 #  
 parameters = Tokens['request']['text'].split()  
  
 #  
 # первый параметр - action  
 #  
 action = parameters[0]  
  
  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " get action from request: " + str(err), severity="error", facility="route")  
  
 return action

**action\_reports.py** (Н. Шаверин)

import json  
import sys  
  
sys.path.append('../core')  
sys.path.append('../')  
sys.path.append('reports')  
  
import core\_log as log  
import core\_pg as pg  
import core\_profile as cp  
import tokens  
import report\_status  
import report\_line  
import report\_list  
import report\_brigadier  
import report\_bubble  
import report\_equipment  
import report\_equipment\_bubble  
import Test\_report\_job  
import core\_Entity as ent  
  
  
#  
#  
#  
def info():  
 return "Отчёты"  
  
  
#  
# action\_reports  
# телеграм получил /reports  
# т.е. нужно вернуть список отчётов конкретного пользователя  
#  
def action(Tokens):  
 #  
 # проверяем есть ли Параметры отчёта в callback\_json  
 #  
 parameters = None  
 #  
 try:  
 # переводим параметры в json  
 if 'data' in Tokens['request']:  
 data = json.loads( Tokens['request']['data'] )  
 if 'parameters' in data:  
 parameters = data['parameters']  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Can't found reports type "+str( err ), severity="error", facility="action\_reports")  
  
  
  
 if parameters and 'type' in parameters:  
 log.log(str(Tokens['chat\_id']) + " Report parameters "+json.dumps( parameters ), severity="debug", facility="action\_reports")  
  
  
 #  
 # по parameters.type вызываем соответствующий отчёт  
 #  
 try:  
 #  
 # Status  
 #  
 if parameters['type'] == 'status':  
  
 Tokens = report\_status.make\_report(Tokens, parameters)  
  
 #  
 # Brigadier  
 #  
 elif parameters['type'] == 'brigadier':  
 Tokens = report\_brigadier.make\_report(Tokens, parameters)  
  
 #  
 # Bubble  
 #  
 elif parameters['type'] == 'bubble':  
 Tokens = report\_bubble.make\_report(Tokens, parameters)  
  
  
 #  
 # Equipment  
 #  
 elif parameters['type'] == 'equipment':  
 Tokens = report\_equipment.make\_report(Tokens, parameters)  
  
  
 #  
 # Equipment\_bubble  
 #  
 elif parameters['type'] == 'equipment\_bubble':  
 Tokens = report\_equipment\_bubble.make\_report(Tokens, parameters)  
  
  
 #  
 # Line  
 #  
 elif parameters['type'] == 'line':  
 Tokens = report\_line.make\_report(Tokens, parameters)  
  
 #  
 # List  
 #  
 elif parameters['type'] == 'list':  
 Tokens = report\_list.make\_report(Tokens, parameters)  
  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Action. Can't select report type: " + str(err), severity="error", facility="action\_reports")  
  
  
 # user нажал кнопку 'Отчёты' на ReplyKeyboard  
 # покажем ему все его отчёты  
 else:  
 # если в БД есть отчёты пользователя, выводим кнопки с этими отчётами  
 # Получение списка отчётов из БД по user\_id\_db  
 reports = cp.get\_parameters( Tokens['user\_id'], 'reports' )  
  
  
  
#try:  
 if len(reports):  
 Tokens = tokens.update\_answer(Tokens, update\_text='Выберите отчёт', clear\_all=True)  
 for report in reports:  
 #  
 # добавляем кнопки с отчётами  
 try:  
 # проверка ключей jobs на пустоту  
 # если все ключи пустые, то кнопка не выводится  
 for job in report['jobs']:  
 for parameter in job:  
 # если хотя бы один ключ job не пустой, то выводим кнопки  
 if job[parameter]:  
 key\_text = report['title']  
 key\_callback = '{"action":"/reports","parameters":' + json.dumps( report ) + '}'  
  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text=key\_text, add\_key\_callback=key\_callback, add\_key\_row=True)  
  
 break  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Can't add report key: "+str(err), severity="error", facility="action\_reports")  
  
 else:  
 Tokens = tokens.update\_answer(Tokens, update\_text='У вас нет отчётов')  
  
 return Tokens

**action\_help.py** (Н. Бойко, Н. Шаверин)

import json  
import sys  
sys.path.append('../core')  
sys.path.append('../')  
import core\_log as log  
  
import tokens  
  
#  
#  
#  
def info():  
 return "Помощь"  
  
  
#  
# action\_help  
# телеграм получил /help  
# т.е. нужно вернуть помощь  
#  
def action( Tokens ):  
  
 #  
 # Если вернулся callback, значит мы уже в меню помощи. Нужный пункт должен быть в option  
 #  
 # Если в request нет data, то логируем выводим общее меню Help с заголовком "Выберите вопрос"  
 callback\_json = None  
 try:  
 callback\_json = json.loads( Tokens['request']['data'] )  
  
 except Exception as err:  
 # print('Пришел общий action help')  
 log.log(str(Tokens['chat\_id']) + " Action Help: " + str(err), severity="error", facility="action\_help")  
  
  
 #  
 # проверяем есть ли callback\_json option  
 #  
 if callback\_json and 'option' in callback\_json:  
  
 #  
 # Как создать отчёт?  
 #  
 if callback\_json['option'] == 'addreport':  
 answer\_text = 'Нажмите кнопку <b>“Настройки”</b>, затем выберите пункт <b>“Отчеты”</b>. У Вас появится список доступных отчетов и кнопка <b>“Создать новый отчет”</b>, нажимая ее Вы попадаете в меню создания отчета. Задав необходимые параметры, новый отчет появится в списке отчетов, который можно найти, нажав кнопку <b>“Отчеты”</b> в главном меню.'  
 Tokens = tokens.update\_answer(Tokens, update\_text=f'Отчёт добавляется так:\n{answer\_text}', update\_parse\_mode='html')  
 #  
 # Как добавить сотрудника?  
 #  
 elif callback\_json['option'] == 'addemployee':  
 answer\_text = 'Нажмите кнопку <b>“Настройки”</b>, затем выберите пункт <b>“Сотрудники”</b>. У Вас появится список добавленных сотрудников и кнопка <b>“Добавить нового сотрудника”</b>, нажимая ее Вы попадаете в меню добавления. Вписав ФИО и должность, новый сотрудник появится в списке.'  
 Tokens = tokens.update\_answer(Tokens, update\_text=f'Сотрудник добавляется так:\n{answer\_text}', update\_parse\_mode='html')  
 #  
 # Как добавить объект?  
 #  
 elif callback\_json['option'] == 'addobject':  
 answer\_text = 'Нажмите кнопку <b>“Настройки”</b>, затем выберите пункт <b>“Объекты”</b>. У Вас появится список доступных объектов и кнопка <b>“Добавить новый объект”</b>, нажимая ее Вы попадаете в меню создания объекта. Задав наименование, новый объект появится в списке'  
 Tokens = tokens.update\_answer(Tokens, update\_text=f'Объект добавляется так:\n{answer\_text}', update\_parse\_mode='html')  
 #  
 # Как добавить задачу?  
 #  
 elif callback\_json['option'] == 'addjob':  
 answer\_text = 'Нажмите кнопку <b>“Настройки”</b>, затем выберите пункт <b>“Задачи”</b>. У Вас появится список доступных задач и кнопка <b>“Создать новую задачу”</b>, нажимая ее Вы попадаете в меню создания задачи. Задав наименование, новая задача появится в списке.'  
 Tokens = tokens.update\_answer(Tokens, update\_text=f'Задача добавляется так:\n{answer\_text}', update\_parse\_mode='html')  
 #  
 # Как добавить группу?  
 #  
 elif callback\_json['option'] == 'addgroup':  
 answer\_text = 'Нажмите кнопку <b>“Настройки”</b>, затем выберите пункт <b>“Группы”</b>. У Вас появится список доступных групп и кнопка <b>“Создать новую группу”</b>, нажимая ее Вы попадаете в меню создания группы. Задав необходимые параметры, новая группа появится в списке.'  
 Tokens = tokens.update\_answer(Tokens, update\_text=f'Группа добавляется так:\n{answer\_text}', update\_parse\_mode='html')  
 #  
 # Как добавить проект?  
 #  
 elif callback\_json['option'] == 'addproject':  
 answer\_text = 'Нажмите кнопку <b>“Настройки”</b>, затем выберите пункт <b>“Проекты”</b>. У Вас появится список доступных проектов и кнопка <b>“Создать новый проект”</b>, нажимая ее Вы попадаете в меню создания проекта. Задав наименование, новый проект появится в списке.'  
 Tokens = tokens.update\_answer(Tokens, update\_text=f'Проект добавляется так:\n{answer\_text}', update\_parse\_mode='html')  
 #  
 # Как добавить напоминание?  
 #  
 elif callback\_json['option'] == 'addalarm':  
 answer\_text = 'Нажмите кнопку <b>“Настройки”</b>, затем выберите пункт <b>“Напоминания”</b>. У Вас появится список актуальных напоминаний и кнопка <b>“Создать новое напоминание”</b>, нажимая ее Вы попадаете в меню создания напоминания. Задав необходимые параметры, новое напоминание появится в списке.'  
 Tokens = tokens.update\_answer(Tokens, update\_text=f'Напоминание добавляется так:\n{answer\_text}', update\_parse\_mode='html')  
  
  
 #  
 # если нет option, то пришёл общий action=/help  
 #  
 else:  
 try:  
 #print('action\_help. пришёл общий action=/help')  
 # Очищаем всё что было в сообщении - создаём новое сообщение  
 Tokens = tokens.update\_answer(Tokens, update\_text='Выберите вопрос', clear\_all = True)  
   
 # добавляем кнопки  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text='Как создать Отчёт?', add\_key\_callback='{"action":"/help","option":"addreport"}', add\_key\_row=True)  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text='Как добавить Сотрудника?', add\_key\_callback='{"action":"/help","option":"addemployee"}',add\_key\_row=True)  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text='Как добавить Объект?', add\_key\_callback='{"action":"/help","option":"addobject"}', add\_key\_row=True)  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text='Как добавить Задачу?', add\_key\_callback='{"action":"/help","option":"addjob"}', add\_key\_row=True)  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text='Как добавить Группу?', add\_key\_callback='{"action":"/help","option":"addgroup"}', add\_key\_row=True)  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text='Как добавить Проект?', add\_key\_callback='{"action":"/help","option":"addproject"}', add\_key\_row=True)  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text='Как добавить Напоминание?', add\_key\_callback='{"action":"/help","option":"addalarm"}', add\_key\_row=True)  
   
 log.log(str(Tokens['chat\_id']) + " Action Help", severity="debug", facility="action\_help")  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Action Help: " + str(err), severity="error", facility="action\_help")  
  
  
 return Tokens

**action\_date.py** (Н. Шаверин)

from telebot import types  
  
import json  
import re  
from datetime import datetime  
  
  
import sys  
sys.path.append('../core')  
sys.path.append('../')  
import core\_log as log  
  
import tokens  
  
#  
#  
def info():  
 return "Дата"  
  
  
#  
# accept\_date  
# телеграм получил /date  
# т.е. нужно вернуть дату..  
#  
def action( Tokens ):  
  
 try:  
 dt = datetime.now()  
 strf = dt.strftime("%Y-%m-%d %H:%M:%S")  
 Tokens = tokens.update\_answer(Tokens, update\_text=strf + '\n')  
 log.log(str(Tokens['chat\_id']) + " Action Date", severity="debug", facility="action\_date")  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Action Date: " + str(err), severity="error", facility="action\_date")  
  
 return Tokens

**report\_list.py** (Н. Шаверин)

# Отчёт типа list  
  
# HTML style  
# To use this mode, pass HTML in the parse\_mode field when using sendMessage. The following tags are currently supported:  
#  
# <b>bold</b>, <strong>bold</strong>  
# <i>italic</i>, <em>italic</em>  
# <a href="URL">inline URL</a>  
# <code>inline fixed-width code</code>  
# <pre>pre-formatted fixed-width code block</pre>  
# Please note:  
#  
# Only the tags mentioned above are currently supported.  
# Tags must not be nested.  
# All <, > and & symbols that are not a part of a tag or an HTML entity must be replaced with the corresponding HTML entities (< with &lt;, > with &gt; and & with &amp;).  
# All numerical HTML entities are supported.  
# The API currently supports only the following named HTML entities: &lt;, &gt;, &amp; and &quot;.  
import sys  
  
sys.path.append('../core')  
sys.path.append('../../core')  
sys.path.append('../')  
  
import json  
  
import core\_Job as cj  
import core\_message as cm  
import core\_Entity as e  
import utils  
import tokens  
import core\_log as log  
import core\_profile as cp  
import datetime  
  
#  
#  
def info():  
 return "Отчёт. List."  
  
# Приводит дату к виду дд/мм/гг  
def get\_formatted\_date(date):  
 date\_list = str(date).split('-')  
 date\_list[0] = date\_list[0][2] + date\_list[0][3]  
 date\_list.reverse()  
 date\_format = '/'.join(date\_list)  
 return date\_format  
  
# Проверка telegram\_id у подписчиков. Если его нет или 0, то из списка рассылки исключается.  
def check\_telegram\_id(subscribers: list)->list:  
 for subscriber in subscribers:  
 # если у subject\_id заполнен parameters.telegram.id  
 sub\_telegram = cp.get\_parameters\_value(subscriber, 'telegram')  
 if not (sub\_telegram and sub\_telegram['id']):  
 subscribers.remove(subscriber)  
 return subscribers  
  
# Подготовка отчёта типа "List"  
# На входе Tokens, parameters  
# На выходе Tokens, в котором Inline\_Keyboard с отчётами (каждая кнопка - отчёт) и период  
  
def make\_report(Tokens, parameters = None):  
 # Список (дела на period)  
 # jobs = list()  
 # Заголовок  
  
 #  
 # user\_id ( Telegramid -> id(entity) )  
 #  
  
 # сборка отчёта переделана не по user\_id, а по subject  
  
 # если нет parameters (команда /start), то parameters -> задачи на сегодня для 'subject': [Tokens['user\_id']]  
 if not parameters:  
 parameters = {'jobs': [{'subject': [Tokens['user\_id']]}],  
 'type': 'list', 'title': 'Задачи на текущую неделю', 'period': 'w', 'variant': 'plan'}  
  
 # если установлен уровень клавиатуры (пользователь уже нажал на кнопку с отчётом)  
 if 'keyboard\_level' in parameters:  
 if parameters['keyboard\_level'] == 2:  
 Tokens = keyboard\_level\_2(Tokens)  
 elif parameters['keyboard\_level'] == 3:  
 Tokens = keyboard\_level\_3(Tokens)  
 elif parameters['keyboard\_level'] == 4:  
 Tokens = keyboard\_level\_4(Tokens)  
 else:  
 # Период из parameters, user\_id\_param - parameters['jobs'][0]['subject'][0]  
 period = parameters['period']  
 if isinstance(parameters['jobs'][0]['subject'], list):  
 user\_id\_param = parameters['jobs'][0]['subject'][0]  
 elif isinstance(parameters['jobs'][0]['subject'], int):  
 user\_id\_param = parameters['jobs'][0]['subject']  
 else:  
 user\_id\_param = Tokens['user\_id']  
  
 # получаем time\_zone пользователя  
 time\_zone = cp.get\_parameters\_value(Tokens['user\_id'], 'time\_zone')  
 d1, d2 = utils.period2date(period, time\_zone)  
 # если даты равны или одной из дат нет, то работаем по "сегодня"  
 if (d1 == d2) or (d1 is None) or (d2 is None):  
 period\_text = f'сегодня, {get\_formatted\_date(d1)},'  
 today = True  
 else:  
 period\_text = f'период с {get\_formatted\_date(d1)} по {get\_formatted\_date(d2)}'  
 today = False  
  
 jobpattern = dict()  
  
 jobpattern.update({"jobs": [{"subject": user\_id\_param}]})  
 jobpattern.update({"period": [str(d1), str(d2)]})  
 jobpattern.update({"variant": "plan"})  
  
 # в таблице plan по user\_id\_param (subjectid) ищем дела id дел на сегодня. id\_jobs - список.  
 id\_jobs = cj.search(jobpattern)  
  
 # Получаем имя и фамилию создателя отчёта  
 my\_last\_name = str(Tokens['user']['last\_name']) if (  
 Tokens['user']['last\_name'] and Tokens['user']['last\_name'] != None) else ''  
 my\_name = str(Tokens['user']['first\_name']) if (  
 Tokens['user']['first\_name'] and Tokens['user']['first\_name'] != None) else ''  
 creator = '<i>Создал: ' + my\_last\_name + ' ' + my\_name + '</i>\n'  
  
 # корректируем дату согласно time\_zone  
 dt = datetime.datetime.today() + datetime.timedelta(hours=time\_zone)  
 dt = get\_formatted\_date(dt.strftime("%Y-%m-%d"))  
  
 creator\_and\_date = creator + f'<i>Дата создания: {dt}</i>'  
  
 # Проверить, если список пустой или None, то выдать, что дел на сегодня нет  
 if id\_jobs is None:  
 jobs\_for\_period = f'<b>На {period\_text} дел нет.</b>\n{creator\_and\_date}'  
 else:  
 if len(id\_jobs) == 0:  
 jobs\_for\_period = f'<b>На {period\_text} дел нет.</b>\n{creator\_and\_date}'  
 else:  
 jobs\_for\_period = f'<b>Задачи на {period\_text}:</b>\n{creator\_and\_date}'  
  
 #  
 # Get parameters. Получаем список полей по id из plan  
 #  
 try:  
 jobs = cj.get(id\_jobs)  
  
 # сортируем plan\_jobs по plan\_jobs['plandate']  
 jobs.sort(key=lambda dictionary: dictionary['plandate'])  
 counter = 0  
  
 # список id relations  
 relations\_id = [job['relationid'] for job in jobs]  
 # список id objects  
 objects\_id = [job['objectid'] for job in jobs]  
  
# список id subjects  
 subjects\_id = [job['subjectid'] for job in jobs]  
  
 # все id сущностей в одном списке  
 entities\_id = relations\_id + objects\_id + subjects\_id  
 # избавляемся от дублей  
 entities\_id = list(set(entities\_id))  
  
 entities = e.get(user\_id\_param, entities\_id)  
  
 # находим и дописываем к каждой job 'relation' и 'object'. При наличии пары из entities сохраняем в plan\_jobs[]. Без пары игнорируются.  
 plan\_jobs = []  
 for job in jobs:  
 append\_job = False  
 for entity in entities:  
 if job['relationid'] == entity['id']:  
 job.update({'relation': entity})  
 append\_job = True  
 if job['objectid'] == entity['id']:  
 job.update({'object': entity})  
 append\_job = True  
 if append\_job:  
 plan\_jobs.append(job)  
  
 #  
 # get object  
 #  
 # проходим циклом по каждому plan\_job и делаем ему кнопку  
 for plan\_job in plan\_jobs:  
 relation = None  
 obj = None  
  
 # текст на InlineButton  
 key\_text = ''  
 # callback на кнопку  
 rel\_obj = dict()  
  
 # сохраняем в callback user\_id\_param  
 rel\_obj.update({'user\_id\_param': user\_id\_param})  
 # сохраняем jobs в callback jobs. будут нужны при нажатии на кнопку "Назад"  
 rel\_obj.update({'jobs': parameters['jobs']})  
  
 # id из plan  
 rel\_obj.update({'planid': plan\_job['id']})  
  
 # пока нет памяти предыдущего диалога  
 # запишем period для возможности возврата по кнопке "Назад"  
 rel\_obj.update({'period': period})  
  
 rel\_obj.update({'plandate': plan\_job['plandate']})  
  
 #  
 # get relation  
 #  
 # записываем relation  
 if 'relationid' in plan\_job:  
 # работа  
 relation = plan\_job['relation']  
 rel\_obj.update({'relation': relation})  
  
  
 # записываем object  
 if 'objectid' in plan\_job:  
 # объект (с которым надо сделать работу)  
 obj = plan\_job['object']  
 rel\_obj.update({'obj': obj})  
  
 # записываем entities\_id  
 rel\_obj.update({'entities\_id': entities\_id})  
  
 counter += 1  
  
 # Номер на кнопку  
 if today:  
 key\_text += str(counter) + '. '  
 else:  
 key\_text += str(counter) + '. ' + get\_formatted\_date(plan\_job['plandate']) + '. '  
  
 if relation:  
  
 # # ТЕКСТ НА КНОПКУ (работа)  
 key\_text += relation['name'] + '.'  
  
 else:  
 pass  
  
 if obj:  
  
 # # ТЕКСТ НА КНОПКУ (объект)  
 key\_text += ' ' + obj['name']  
  
 else:  
 pass  
  
 # тип отчёта на key\_callback  
 rel\_obj.update({'type': 'list'})  
 # следующий уровень клавиатуры - 2  
 rel\_obj.update({'keyboard\_level': 2})  
  
  
  
 # делаем кнопку  
 key\_callback = '{"action":"/reports","parameters":' + json.dumps(rel\_obj) + '}'  
  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text=key\_text, add\_key\_callback=key\_callback,  
 add\_key\_row=True)  
  
 # максимальное количество кнопок  
 if counter == 50:  
 text\_message = f'<b>Вывожу только первые {counter} задач!\nБольше не могу, т.к. их слишком много.\n\n</b>'  
 Tokens = tokens.update\_answer(Tokens, update\_text=text\_message, update\_parse\_mode='html')  
 break  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Find jobs: " + str(err), severity="error",  
 facility="report\_list\_make\_report")  
  
 Tokens = tokens.update\_answer(Tokens, update\_text=jobs\_for\_period, update\_parse\_mode='html')  
  
 return Tokens  
  
  
# пользователь нажал кнопку с делом. выводим подробное описание дела и четыре кнопки: сделано, проблема, работник, назад. кнопки в один ряд.  
def keyboard\_level\_2(Tokens):  
  
 data = json.loads(Tokens['request']['data'])  
 parameters = data['parameters']  
  
 job\_for\_period = ''  
  
 try:  
 job\_for\_period += 'Дата - ' + get\_formatted\_date(parameters['plandate']) + '\n'  
 # дополнительная информация по объекту  
 obj\_info = ''  
 job\_for\_period += 'Объект - ' + parameters['obj']['name'] + '.' + obj\_info + '\n'  
 # дополнительная информация по работе  
 rel\_info = '\n'  
  
 # описание работы выводится более расширенно через parameters.description  
 description = 'Описание:\n'  
  
 if 'description' in parameters['relation'] and parameters['relation']['description'] and len(parameters['relation']['description']):  
 if isinstance(parameters['relation']['description'], dict):  
 for key, value in parameters['relation']['description'].items():  
 description += key + ': ' + value + '\n'  
 else:  
 description += str(parameters['relation']['description'])  
 else:  
 description += 'Отсутствует'  
  
 job\_for\_period += 'Работа - ' + parameters['relation']['name'] + '.' + rel\_info + description  
  
 # формируем и выводим название дела  
 Tokens = tokens.update\_answer(Tokens, update\_text=job\_for\_period, update\_parse\_mode='html')  
  
# тип отчёта на key\_callback  
 parameters.update({'type': 'list'})  
 # следующий уровень клавиатуры - 3  
 parameters.update({'keyboard\_level': 3})  
  
 # делаем кнопку "Сделано"  
 key\_text = 'Сделано'  
 # добавляем ключ в callback  
  
 parameters.update({'job\_modify': 'made'})  
 key\_callback = '{"action":"/reports","parameters":' + json.dumps(parameters) + '}'  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text=key\_text, add\_key\_callback=key\_callback,  
 add\_key\_row=False)  
  
 # делаем кнопку "Проблема"  
 key\_text = 'Проблема'  
 # добавляем ключ в callback  
 parameters.update({'job\_modify': 'problem'})  
 key\_callback = '{"action":"/reports","parameters":' + json.dumps(parameters) + '}'  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text=key\_text, add\_key\_callback=key\_callback,  
 add\_key\_row=False)  
  
 # делаем кнопку "Назначить работника"  
 key\_text = 'Работник'  
 # добавляем ключ в callback  
 parameters.update({'job\_modify': 'worker'})  
 key\_callback = '{"action":"/reports","parameters":' + json.dumps(parameters) + '}'  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text=key\_text, add\_key\_callback=key\_callback,  
 add\_key\_row=False)  
  
 # делаем кнопку "Назад"  
 key\_text = 'Назад'  
 # добавляем ключ в callback  
 parameters.update({'job\_modify': 'back'})  
 key\_callback = '{"action":"/reports","parameters":' + json.dumps(parameters) + '}'  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text=key\_text, add\_key\_callback=key\_callback,  
 add\_key\_row=False)  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Can't made button " + str(err), severity="error", facility="report\_list.keyboard\_level\_2")  
  
 return Tokens  
  
  
# если нажата одна из кнопок дел (сделано, проблема, работник, назад)  
# по callback определяем кнопку и запускаем обработчик  
def keyboard\_level\_3(Tokens):  
  
 data = json.loads(Tokens['request']['data'])  
  
 # то, что находится в callback кнопки  
 parameters = data['parameters']  
  
 # Получаем id пользователя из parameters, его firstname и lastname из БД по user\_id\_param  
 user\_id\_param = parameters['user\_id\_param']  
 user\_telegram = cp.get\_parameters\_value(user\_id\_param, 'telegram')  
 first\_name = user\_telegram['firstname']  
 last\_name = user\_telegram['lastname']  
  
 # в parameters, добавлен ключ user\_id из subject  
  
 try:  
 # список подписчиков  
 subscribers\_list = list()  
 for entity\_id in parameters['entities\_id']:  
 entity\_subscribers = cp.get\_parameters\_value(entity\_id, 'subscribers')  
 # если подписчики есть  
 if entity\_subscribers:  
 subscribers\_list = subscribers\_list + entity\_subscribers  
 # убираем дубли  
 subscribers\_list = list(set(subscribers\_list))  
 except Exception as err:  
 subscribers\_list = []  
 log.log(str(Tokens['chat\_id']) + " Can't get subscribers " + str(err), severity="error", facility="report\_list.keyboard\_level\_3")  
  
 # бригадиры (кому будут рассылаться уведомления о выполнении работы)  
 brig\_subscribers = []  
 # помощники (кому будут рассылаться уведомления о проблеме)  
 problem\_subscribers = []  
 # названия групп  
 names\_of\_groups = ''  
 try:  
 # группы, в которые входит user  
 user\_groups = cp.get\_groups(user\_id\_param)  
  
 # готовим названия групп, в которые входит user  
 groups\_entities = e.get(user\_id\_param, user\_groups)  
 for ent in groups\_entities:  
 if ent['type'] == 'group':  
 # добавляем имя группы  
 names\_of\_groups += ent['name']  
 names\_of\_groups += ', '  
 # если у группы есть 'brig\_subscribers'  
 if ent['options'] and 'brig\_subscribers' in ent['options'] and ent['options']['brig\_subscribers']:  
 # расширяем список бригадиров  
 brig\_subscribers.extend(ent['options']['brig\_subscribers'])  
  
 # если у группы есть 'problem\_subscribers'  
 if ent['options'] and 'problem\_subscribers' in ent['options'] and ent['options']['problem\_subscribers']:  
 # расширяем список помощников  
 problem\_subscribers.extend(ent['options']['problem\_subscribers'])  
  
 names\_of\_groups = names\_of\_groups[:-2]  
 # удаляем дубли и проверяем telegram  
 brig\_subscribers = check\_telegram\_id(list(set(brig\_subscribers)))  
 problem\_subscribers = check\_telegram\_id(list(set(problem\_subscribers)))  
  
 except Exception as err:  
  
 log.log(str(Tokens['chat\_id']) + " Can't get user\_groups " + str(err), severity="error", facility="report\_list.keyboard\_level\_3")  
  
  
 try:  
  
 # работа  
 relation = parameters['relation']['name']  
 # объект, на котором выполнена работа  
 object = parameters['obj']['name']  
 # плановая дата  
 plandate = parameters['plandate']  
 # приводим к общему виду  
 plandate = get\_formatted\_date(plandate)  
  
 # проверяем, какая кнопка была нажата  
  
 # сделано  
 if parameters['job\_modify'] == 'made':  
  
 # COMMIT - установить факт выполнения работы (запись factdate в plan)  
 if cj.commit(parameters['planid']):  
 text = f'<b>Выполнено</b>\n'  
 text = text + f"кто: {first\_name} {last\_name} [{names\_of\_groups}]\n"  
 text = text + f"что: {relation}\n"  
 text = text + f"объект: {object}\n"  
 text = text + f"план: {plandate}"  
  
 # # Версия с рассылкой сообщений бригадирам групп ==========================================================  
 # список рассылки = список бригадиров групп, в которые входит пользователь  
 subscribers\_list = brig\_subscribers  
  
 if len(subscribers\_list) == 0:  
 message\_text = 'Сообщения не отправлены, список рассылки пуст.'  
 else:  
 message\_text = text + '\n\n' + cm.create\_message(text, subscribers\_list, [])  
 # # ========================================================================================================  
  
 # Версия без рассылки сообщений при выполнении работы ======================================================  
 # message\_text = text  
 # ==========================================================================================================  
 else:  
 message\_text = 'Ошибка базы данных. Пожалуйста, повторите операцию.'  
  
 Tokens = tokens.update\_answer(Tokens, update\_text=message\_text, update\_parse\_mode='html')  
  
 # проблема  
 elif parameters['job\_modify'] == 'problem':  
 text = f'<b>Проблема</b>\n'  
 text = text + f"кто: {first\_name} {last\_name} [{names\_of\_groups}]\n"  
 text = text + f"что: {relation}\n"  
 text = text + f"объект: {object}\n"  
 text = text + f"план: {plandate}"  
  
 # # Версия с рассылкой сообщений помощникам ==========================================================  
 # список рассылки = список помощников групп, в которые входит пользователь  
 subscribers\_list = problem\_subscribers  
  
 # если список рассылки пуст  
 if len(subscribers\_list) == 0:  
 message\_text = 'Сообщения не отправлены, список рассылки пуст.'  
 else:  
 message\_text = text + '\n\n' + cm.create\_message(text, subscribers\_list, [])  
  
 Tokens = tokens.update\_answer(Tokens, update\_text=message\_text, update\_parse\_mode='html')  
  
 # назначить работника на дело  
 elif parameters['job\_modify'] == 'worker':  
  
 # id групп пользователя  
 id\_user\_groups = cp.get\_groups\_entities(user\_groups)  
  
 # по id ищем только subject  
 id\_subjects = e.search(user\_id\_param, {  
 'id': id\_user\_groups,  
 'type': 'subject'  
 })  
  
 # список subject  
 subjects = e.get(user\_id\_param, id\_subjects)  
  
 plan\_info = dict()  
 # тип отчёта  
 plan\_info.update({'type': 'list'})  
 # следующий уровень клавиатуры - 4  
 plan\_info.update({'keyboard\_level': 4})  
 # user\_id\_param  
 plan\_info.update({'user\_id\_param': user\_id\_param})  
 # planid  
 plan\_info.update({'planid': parameters['planid']})  
 # plandate  
 plan\_info.update({'plandate': plandate})  
  
 for subject in subjects:  
 # имя работника выводим на кнопку  
 key\_text = subject['name']  
 # subjectid  
 plan\_info.update({'subjectid': subject['id']})  
  
 key\_callback = '{"action":"/reports","parameters":' + json.dumps(plan\_info) + '}'  
  
 # print(f'Кнопка {key\_text} - {key\_callback}')  
  
 Tokens = tokens.update\_answer(Tokens, add\_key\_text=key\_text, add\_key\_callback=key\_callback,  
 add\_key\_row=True)  
  
 # краткое описание выводим в виде текста  
 text = f'<b>Назначить задачу:</b>\n'  
 text = text + f"плановая дата: {plandate}\n"  
 text = text + f"объект: {object}\n"  
 text = text + f"работа: {relation}\n"  
 message\_text = text  
  
 Tokens = tokens.update\_answer(Tokens, update\_text=message\_text, update\_parse\_mode='html')  
  
 # назад  
 elif parameters['job\_modify'] == 'back':  
 # переход назад (заново выводим список дел)  
 # для перехода на второй уровень keyboard удаляем 'keyboard\_level' из parameters  
 del parameters['keyboard\_level']  
 Tokens = make\_report(Tokens, parameters)  
  
 else:  
 # Просто текст  
 text = 'Просто текст'  
 Tokens = tokens.update\_answer(Tokens, update\_text=text, update\_parse\_mode='html')  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Can't read callback " + str(err), severity="error",  
 facility="report\_list\_keyboard\_level\_3")  
  
 return Tokens  
  
  
# назначение работника для работы  
# по callback определяем работника и меняем subjectid в plan  
# отправляем сообщение-уведомление  
def keyboard\_level\_4(Tokens):  
  
 # то, что находится в callback кнопки  
 data = json.loads(Tokens['request']['data'])  
  
 parameters = data['parameters']  
  
 try:  
 job = {  
 'id': parameters['planid'],  
 'subject': parameters['subjectid']  
 }  
 cj.update(job)  
  
 text = f"Вам назначена новая задача со сроком исполнения <b>{parameters['plandate']}</b>"  
 # отправляем сообщение-уведомление работнику  
 count\_message = cm.create\_message(text, [parameters['subjectid']], [])  
 message\_text = f'Работник назначен. {count\_message}.'  
  
 except Exception as err:  
 message\_text = 'Работник НЕ назначен. ОШИБКА.'  
 log.log(str(Tokens['chat\_id']) + " Can't appoint worker " + str(err), severity="error",  
 facility="report\_list\_keyboard\_level\_4")  
  
 Tokens = tokens.update\_answer(Tokens, update\_text=message\_text, update\_parse\_mode='html')  
  
  
 return Tokens

**report\_status.py** (А. Караткевич)

import json  
import sys  
  
sys.path.append('../core')  
sys.path.append('../')  
sys.path.append('../reports')  
  
import core.core\_log as log  
import core.utils as util  
from telegram import tokens  
from datetime import datetime  
import jobppaterns  
  
  
def make\_report(Tokens: dict, parameters: json) -> dict:  
 try:  
 if 'period' in parameters:  
  
 # в parameters добавляем даты начала и конца периода  
 parameters = util.add\_date\_in\_report(parameters)  
  
 # формируем правильный jobpattern за исключением 'variant'  
 jobpattern = jobppaterns.form\_jobpattern(parameters)  
 # добавляем в Tokens заголовок  
  
 Tokens = report\_header(Tokens, parameters)  
  
 count\_jobs = jobppaterns.total\_jobs(jobpattern)  
  
 detailed\_jobs = jobppaterns.detailed\_fact\_jobs(jobpattern)  
 log.log("detailed\_jobs =" + str(detailed\_jobs), severity="test", facility="report\_status")  
  
 message = message\_total(count\_jobs, detailed\_jobs)  
  
 #message = 'Тестовое сообщение'  
  
 # добавляем сообщение для вывода в телеграмм  
 Tokens = tokens.update\_answer(Tokens, update\_parse\_mode='html', update\_text='<pre>' + message + '</pre>')  
   
 else:  
 Tokens = tokens.update\_answer(Tokens, update\_text='Задан некорректный период для отчета')  
  
 log.log(str(Tokens['chat\_id']) + " Report\_title: " + parameters['title'] + " is OK ", severity="debug",  
 facility="report\_status")  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Report\_status.make\_report: " + str(err), severity="error",  
 facility="report\_status")  
  
 return Tokens  
  
  
def report\_header(Tokens: dict, report: dict) -> dict:  
 *"""  
 функция формирует заголовок для отчета* ***:param*** *Tokens:* ***:param*** *report:* ***:return****:  
 """* # Шапка отчёта  
 # преобразуем дату в читаемый формат  
 eu\_d1 = util.change\_date\_format(report['d1'])  
 eu\_d2 = util.change\_date\_format(report['d2'])  
 current\_date = str(datetime.today().date())  
 format\_current\_date = util.change\_date\_format(current\_date)  
 # Получаем имя и фамилию владельца  
 my\_last\_name = str(Tokens["user"]['last\_name'])  
 my\_name = str(Tokens["user"]['first\_name'])  
  
 Tokens = tokens.update\_answer(Tokens, clear\_all=True, update\_parse\_mode='html')  
 Tokens = tokens.update\_answer(Tokens, update\_text="<b>" + report['title'] + "</b>")  
  
 #message = 'Тестовое сообщение'  
  
  
 # период  
 if 'd1' and 'd2' in report:  
 Tokens = tokens.update\_answer(Tokens, update\_text='<b> ' + eu\_d1 + " - " + eu\_d2 + '</b>\n\n')  
 Tokens = tokens.update\_answer(Tokens, update\_text='<i>Создал: ' + my\_last\_name + ' ' + my\_name + '</i>\n')  
 Tokens = tokens.update\_answer(Tokens, update\_text='<i>Дата создания: ' + format\_current\_date + '</i>')  
  
 Tokens = tokens.update\_answer(Tokens, update\_text='\n')  
  
 return Tokens  
  
  
def message\_total(count\_jobs, detailed\_jobs) -> str:  
 message = ' \n'  
 message = message + f'{"Осталось по плану" :18}: {str(count\_jobs["plan"])} \n\n'  
 message = message + f'{"Выполнено" :18}: {str(count\_jobs["fact"])} \n\n'  
 message = message + f'{" - вне плана":18}: {str(detailed\_jobs["notplan"])} \n'  
 message = message + f'{" - вовремя":18}: {str(detailed\_jobs["on\_time"])} \n'  
 message = message + f'{" - с опозданием":18}: {str(detailed\_jobs["with\_delay"])}\n\n'  
 message = message + f"{'Переходящие долги':18}: {str(count\_jobs['debt'])}\n\n"  
 message = message + f"{'Некорректые задачи':18}: {str(count\_jobs['err'])} \n"  
 return message

**report\_line.py** (А. Караткевич, М. Демченко)

import json  
import sys  
from collections import Counter  
import datetime  
  
sys.path.append('../core')  
sys.path.append('../')  
sys.path.append('../reports')  
  
import matplotlib.pyplot as plt  
import numpy as np  
import core\_log as log  
import utils as util  
from telegram import tokens  
import core\_Job as job  
import core\_config as cf  
from telegram.reports import report\_status as rs  
  
  
#  
#  
#  
def info():  
 return "Отчёт. Line."  
  
  
def make\_report(Tokens: dict, parameters: dict) -> dict:  
 *"""  
 Функция создания отчета с графиком* ***:param*** *Tokens: Tokens* ***:param*** *parameters: параметры отчета* ***:return****:  
 """* try:  
 # проверяем есть ли в параметрах отчета заданный период  
 if 'period' in parameters:  
 # формируем путь для изображения  
 path\_to\_img = path\_to\_image(Tokens, parameters)  
  
 # в parameters добавляем даты начала и конца периода  
 parameters = util.add\_date\_in\_report(parameters)  
  
 # формируем текст для шапки графика  
 user\_id = Tokens["user\_id"]  
 Tokens = rs.report\_header(Tokens, parameters)  
  
 # формируем pattern для плановых работ и фактических  
 jobpattern\_plan = get\_jobpattern\_plan(parameters, user\_id)  
 jobpattern\_fact = get\_jobpattern\_fact(parameters, user\_id)  
  
 # Поиск запланированных и выполненых работ  
 task\_list\_plan: list = job.search(jobpattern\_plan)  
 task\_list\_fact: list = job.search(jobpattern\_fact)  
  
 # Сформированные списки  
 result\_plan: list = job.get(task\_list\_plan)  
 result\_fact: list = job.get(task\_list\_fact)  
  
 create\_graphic(result\_plan, result\_fact, path\_to\_img, parameters)  
  
 # формирование пути для картинки с графиком  
 img = dict()  
 img.update({'path': path\_to\_img})  
 img.update({'caption': ''})  
  
 Tokens = tokens.update\_answer(Tokens, update\_parse\_mode='html', add\_img=img)  
  
 log.log(str(Tokens['chat\_id']) + " Report\_title: " + parameters['title'] + " is OK ", severity="debug",  
 facility="report\_line")  
  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Report\_line.make\_report: " + str(err), severity="error",  
 facility="report\_line")  
  
 return Tokens  
  
  
def get\_jobpattern\_plan(report: json, user\_id) -> dict:  
 *"""  
 Функция формирует паттерн запроса в БД* ***:param*** *user\_id: user\_id* ***:param*** *report: отчет* ***:return****: jobpattern запрос  
 """* jobpattern\_plan = dict()  
  
 jobpattern\_plan.update({"jobs": [{"subject": user\_id}]})  
 jobpattern\_plan.update({"period": [report["d1"], report["d2"]]})  
 jobpattern\_plan.update({"variant": "plan"})  
  
 return jobpattern\_plan  
  
  
def get\_jobpattern\_fact(report: json, user\_id) -> dict:  
 jobpattern\_fact: dict = dict()  
  
 jobpattern\_fact.update({"jobs": [{"subject": user\_id}]})  
 jobpattern\_fact.update({"period": [report["d1"], report["d2"]]})  
 jobpattern\_fact.update({"variant": "fact"})  
  
 return jobpattern\_fact  
  
  
def create\_graphic(result\_plan, result\_fact, path\_to\_img, parameters):  
 *"""  
 Функция создающая график сохраняя его в папке /img* ***:param*** *result\_plan: массив с плановыми работами* ***:param*** *result\_fact: массив с фактическими работами* ***:param*** *path\_to\_img: путь до папки* ***:param*** *parameters:* ***:return****:  
 """* x\_plan = np.array(get\_date\_plan(result\_plan, result\_fact))  
 y\_plan = np.array(get\_job\_plan(result\_plan, result\_fact))  
 x\_fact = np.array(get\_date\_fact(result\_fact))  
 y\_fact = np.array(get\_job\_fact(result\_fact))  
  
 fig, ax = plt.subplots()  
  
 ax.plot(x\_plan, y\_plan, 'b--o', label='План')  
 ax.plot(x\_fact, y\_fact, 'g-o', label='Факт')  
  
 ax.legend()  
  
 if parameters['period'] == '-m1':  
 date = datetime.date.today() - datetime.timedelta(days=31)  
 elif parameters['period'] == '-m2':  
 date = datetime.date.today() - datetime.timedelta(days=62)  
 else:  
 date = datetime.date.today()  
  
 plt.xlabel(date.strftime('%B'))  
  
 x\_axis = np.hstack([x\_plan, x\_fact])  
 x\_axis = np.sort(x\_axis)  
 plt.xticks(x\_axis)  
  
 plt.title(str(parameters['title']))  
 plt.ylabel("Выполнено работ")  
  
 plt.grid()  
  
 return plt.savefig(path\_to\_img)  
  
  
def get\_date\_plan(result\_plan: list, result\_fact: list) -> list:  
 *"""Функция для получения массива с датами запланированных работ, принимает массив работ"""* plan\_array: list = list()  
 plan\_get: list = list()  
  
 # Иттерируемся по массиву с работами и из каждой работы забираем дату и срезом записываем число в новый массив  
 for job in result\_plan:  
 plandate = job.get("plandate")  
 if plandate != 'None':  
 plan\_get.append(int(plandate[8:10]))  
  
 for job in result\_fact:  
 plandate = job.get("plandate")  
 if plandate != 'None':  
 plan\_get.append(int(plandate[8:10]))  
  
 # создаем множество с неповторяющимися элементами(датами)  
 plan\_set = set(plan\_get)  
  
 # записываем эти даты в новый массив  
 for elem in plan\_set:  
 plan\_array.append(elem)  
  
 # сортируем массив  
 plan\_array.sort()  
  
 return plan\_array  
  
  
def get\_job\_plan(result\_plan: list, result\_fact: list):  
 *"""Функция принимает массив с датами запланированных работ и возвращает массив с количеством самих работ"""* plan\_get: list = list()  
 plan\_jobs: list = list()  
 count\_plan: list = list()  
 count: int = 0  
  
 # Иттерируемся по массиву с работами и из каждой работы заюираем дату и срезом записываем число в новй массив  
 for job in result\_plan:  
 plandate = job.get("plandate")  
 if plandate != 'None':  
 plan\_get.append(int(plandate[8:10]))  
  
 for job in result\_fact:  
 plandate = job.get("plandate")  
 if plandate != 'None':  
 plan\_get.append(int(plandate[8:10]))  
  
 # сортируем массив  
 plan\_get.sort()  
  
 # считаем сколько раз встречается дата и записываем в новую переменную  
 plan\_counter = Counter(plan\_get)  
  
 # сортируем по возрастанию количество повторящихся дат  
 plan\_sorted = sorted(plan\_counter.items())  
  
 # добавляем количество работ в новый массив  
 for elem in plan\_sorted:  
 plan\_jobs.append(elem[1])  
  
 # считаем сколько всего работ выполнено к следующей дате  
 for elem in plan\_jobs:  
 count += int(elem)  
 count\_plan.append(count)  
  
 return count\_plan  
  
  
def get\_date\_fact(result\_fact: list) -> list:  
  
 # массивы для хранения выполненых работ  
 fact\_array = list()  
 fact\_get = list()  
  
 # итерируемся по всем работам смотрим только те у которых дата выполнения не равна None и добавляем их в массив в  
 # виде одного числа  
 for job in result\_fact:  
 factdate = job.get("factdate")  
 if factdate != 'None':  
 factdate = job.get("factdate")  
 fact\_get.append(int(factdate[8:10]))  
  
 # Создаем множество не повторяющихся дат  
 fact\_set = set(fact\_get)  
  
 # Итерируемся по ним и записывам в массив  
 for elem in fact\_set:  
 fact\_array.append(elem)  
  
 # сортируем этот массив  
 fact\_array.sort()  
  
 return fact\_array  
  
  
def get\_job\_fact(result\_fact: list) -> list:  
 # Массивы для хранения выполненых работ и их количества  
 fact\_get = list()  
 fact\_jobs = list()  
 fact\_plan = list()  
 count: int = 0  
  
 # собираем все работы в один массив в виде чисел  
 for job in result\_fact:  
 factdate = job.get("factdate")  
 if factdate != 'None':  
 factdate = job.get("factdate")  
 fact\_get.append(int(factdate[8:10]))  
  
 # Сортируем этот массив  
 fact\_get.sort()  
  
 # считаем выполненые работы за каждое число  
 fact\_counter = Counter(fact\_get)  
  
 # еще раз их сортируем  
 fact\_sorted = sorted(fact\_counter.items())  
  
 # Добавляем все подсчитанные работы в один массив  
 for elem in fact\_sorted:  
 fact\_jobs.append(elem[1])  
  
 # считаем их суммы и добалвяем в массив  
 for elem in fact\_jobs:  
 count += elem  
 fact\_plan.append(count)  
  
 return fact\_plan  
  
  
def path\_to\_image(Tokens: dict, parameters) -> str:  
 *"""  
 Функция принимает Токенс для извлечения id и название файла и возвращает путь к файлу* ***:param*** *Tokens: Tokens* ***:param*** *parameters: название файла* ***:return****:  
 """* config = cf.configuration()  
 user\_id = Tokens['user']['id']  
 path = config['common']['homedir'] + '/img/' + str(user\_id) + "\_" + parameters['period'] + '\_line' + '.png'  
  
 return path

**report\_bubble.py** (А. Караткевич, М. Демченко)

import sys  
import pandas as pd  
  
sys.path.append('../core')  
sys.path.append('../')  
sys.path.append('../reports')  
  
import core.core\_log as log  
import core.utils as util  
from telegram import tokens  
from telegram.reports import report\_status as rs  
import analisys\_pandas as ap  
import matplotlib.pyplot as plt  
import core.core\_config as cf  
  
  
def info():  
 return "Пузырьковый отчет"  
  
  
def make\_report(Tokens: dict, parameters: dict) -> dict:  
 try:  
  
 if 'period' in parameters:  
 # добавляем в отчет даты начала и конца периода  
 parameters = util.add\_date\_in\_report(parameters)  
  
 # Изменяем шапку отчета  
 Tokens = rs.report\_header(Tokens, parameters)  
  
 # формируем путь для графика  
 path\_to\_img = path\_to\_image(Tokens, parameters)  
  
 # пузырьковый отчет - используем bubble\_report:  
 bubble\_report(parameters, path\_to\_img)  
  
 # словарь с путем графика  
 img = dict()  
 img.update({'path': path\_to\_img})  
 img.update({'caption': ''})  
  
 Tokens = tokens.update\_answer(Tokens, update\_parse\_mode='html', add\_img=img)  
  
 else:  
 Tokens = tokens.update\_answer(Tokens, update\_text='Задан некорректный период для отчета')  
  
 log.log(str(Tokens['chat\_id']) + " Report\_title: " + parameters['title'] + " is OK ", severity="debug",  
 facility="report\_bubble")  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Report\_bubble.make\_report: " + str(err), severity="error",  
 facility="report\_bubble")  
  
 return Tokens  
  
  
def bubble\_df(parameters: dict, variant: str):  
 df = ap.total\_dataframe(parameters, variant)  
  
 if df.empty:  
 df = pd.DataFrame(columns=['subjectid', 'plandate', 'factdate'])  
 else:  
 df = df.groupby(['subjectid', 'plandate', 'factdate']).size().reset\_index()  
  
 df.columns = ['subjectid', 'plandate', 'factdate', 'task']  
 name = variant + 'date'  
 df[name] = pd.to\_datetime(df[name])  
  
 df = df.sort\_values(by=name, ascending=True)  
 name\_list = df.subjectid.tolist()  
 name = ap.get\_name(name\_list)  
  
 df["name"] = name  
  
 df.task = df.task \* 50  
  
 return df  
  
  
def bubble\_report(parameters, path\_to\_img):  
 df\_plan = bubble\_df(parameters, 'plan')  
 df\_fact = bubble\_df(parameters, 'fact')  
 print(df\_plan)  
 plt.figure(figsize=(15, 10))  
 plt.rc('axes', unicode\_minus=False)  
 if not df\_plan.empty:  
  
 plt.scatter('plandate', 'name', s='task', alpha=0.5, data=df\_plan, color='blue')  
  
 if not df\_fact.empty:  
  
 plt.scatter('factdate', 'name', s='task', alpha=0.5, data=df\_fact, color='green')  
  
 plt.title(str(parameters['title']), fontdict={'fontsize': 15})  
 plt.ylabel("Сотрудники", size=15)  
 plt.xlabel("Задачи", size=15)  
 plt.xticks(rotation=45)  
 plt.grid()  
 plt.legend(['plan', 'fact'], prop={'size': 20}, bbox\_to\_anchor=(1,1), loc='lower right')  
  
 plt.savefig(path\_to\_img)  
  
  
def path\_to\_image(Tokens: dict, parameters) -> str:  
 *"""  
 Функция принимает Токенс для извлечения id и название файла и возвращает путь к файлу* ***:param*** *Tokens: Tokens* ***:param*** *name: название файла* ***:return****:  
 """* config = cf.configuration()  
 user\_id = Tokens['user']['id']  
 path = config['common']['homedir'] + '/img/' + str(user\_id) + "\_" + parameters['period'] + '-bubble' + '.png'  
  
 return path

**report\_brigadier.py** (А. Караткевич, М. Демченко)

import json  
import sys  
  
sys.path.append('../core')  
sys.path.append('../')  
sys.path.append('../reports')  
  
import core.core\_log as log  
import core.utils as util  
from telegram import tokens  
import core.core\_Entity as e  
import core.core\_Job as job  
import core.core\_profile as cp  
from telegram.reports import report\_status as rs  
import core.core\_Entity as ent  
import analisys\_pandas  
  
  
def info():  
 return "Отчёт. Бригадир."  
  
  
def make\_report(Tokens: dict, parameters: dict) -> dict:  
 try:  
  
 if 'period' in parameters:  
 # добавляем в отчет даты начала и конца периода  
 parameters = util.add\_date\_in\_report(parameters)  
 all\_jobs = analisys\_pandas.count\_jobs\_subject(parameters)  
  
 Tokens = rs.report\_header(Tokens, parameters)  
  
 message = message\_for\_lead(all\_jobs)  
  
 # добавляем сообщение для вывода в телеграмм  
 Tokens = tokens.update\_answer(Tokens, update\_parse\_mode='html', update\_text='<pre>' + message + '</pre>')  
  
 else:  
 Tokens = tokens.update\_answer(Tokens, update\_text='Задан некорректный период для отчета')  
  
 log.log(str(Tokens['chat\_id']) + " Report\_title: " + parameters['title'] + " is OK ", severity="debug",  
 facility="report\_status")  
 except Exception as err:  
 log.log(str(Tokens['chat\_id']) + " Report\_brigadier.make\_report: " + str(err), severity="error",  
 facility="report\_status")  
  
 return Tokens  
  
  
def message\_for\_lead(all\_jobs: dict) -> str:  
 message = " \n" + f'{"План":4} : {"Факт":4} : {"Долг":4} \n'  
 message = message + "--------------------\n"  
 for k, v in all\_jobs.items():  
 name = e.get(k, k)  
 if len(name) > 0:  
 message = message + f'{name[0]["name"]:20} \n '  
 message = message + f'{str(v[1]):4}: {str(v[2]):3}: {str(v[3]):4} \n'  
 message = message + "---------------------\n"  
 return message

**week\_tasks.py** (Н. Шаверин)

#!/usr/bin/python3  
import sys  
  
sys.path.append('../core')  
sys.path.append('/opt/venv/sovtrud/core')  
  
import core\_Job as cj  
import core\_message as cm  
import core\_Entity as e  
import core\_log as log  
import core\_profile as cp  
import utils  
  
  
def get\_formatted\_date(date):  
 *'''  
 Приводит дату к виду дд/мм/гг* ***:param*** *date:* ***:return****: строка в формате дд/мм/гг  
 '''* date\_list = str(date).split('-')  
 date\_list[0] = date\_list[0][2] + date\_list[0][3]  
 date\_list.reverse()  
 date\_format = '/'.join(date\_list)  
 return date\_format  
  
  
def make\_list(list\_groups):  
 *'''  
 В текстовом виде формирует задачи на определённый период для пользователей, являющихся членами групп list\_groups  
 Пользователи без telegram в БД игнорируются, даже при наличии задач  
 Записывает задачи в message для последующей рассылки housekeeper* ***:param*** *list\_groups: список групп* ***:return****: -  
 '''* # # работники и дела (для бригадира)  
 # subject\_jobs = dict()  
  
 # список бригадиров  
 # brigadiers = [173]  
 brigadiers = [20, 173, 422, 574]  
  
 # список членов группы  
 subject\_list = cp.get\_groups\_entities(list\_groups)  
 for subject\_id in subject\_list:  
 # если у subject\_id заполнен parameters.telegram.id  
 sub\_telegram = cp.get\_parameters\_value(subject\_id, 'telegram')  
 # print(subject\_id, sub\_telegram)  
  
 # if sub\_telegram and sub\_telegram['id'] and (subject\_id == 173 or subject\_id == 20):  
 # if sub\_telegram and sub\_telegram['id'] and (subject\_id == 574):  
 if sub\_telegram and sub\_telegram['id']:  
 # Период, на который будет собираться список дел для subject\_id  
 period = 'w'  
 variant = 'plan'  
 # получаем time\_zone subject\_id  
 time\_zone = cp.get\_parameters\_value(subject\_id, 'time\_zone')  
  
 # начальная и конечная даты  
 d1, d2 = utils.period2date(period, time\_zone)  
  
 # если даты равны или одной из дат нет, то работаем по "сегодня"  
 if (d1 == d2) or (d1 is None) or (d2 is None):  
 period\_text = f'сегодня, {get\_formatted\_date(d1)},'  
 today = True  
 else:  
 period\_text = f'период с {get\_formatted\_date(d1)} по {get\_formatted\_date(d2)}'  
 today = False  
  
 # паттерн для запроса  
 jobpattern = dict()  
 jobpattern.update({"jobs": [{"subject": subject\_id}]})  
 jobpattern.update({"period": [str(d1), str(d2)]})  
 jobpattern.update({"variant": variant})  
  
 # в таблице plan по subject\_id ищем дела id дел на сегодня. id\_jobs - список.  
 id\_jobs = cj.search(jobpattern)  
  
 # Проверить, если список пустой или None, то выдать, что дел на сегодня нет  
 if id\_jobs is None or len(id\_jobs) == 0:  
 jobs\_for\_period = f'<b>На {period\_text} дел нет.</b>\n'  
 else:  
 jobs\_for\_period = f'<b>Задачи на {period\_text}:</b>\n'  
 try:  
 # Get parameters. Получаем список полей по id\_jobs из plan  
 jobs = cj.get(id\_jobs)  
 # сортируем plan\_jobs по plan\_jobs['plandate']  
 jobs.sort(key=lambda dictionary: dictionary['plandate'])  
  
 # список id relations  
 relations\_id = [job['relationid'] for job in jobs]  
  
 # список id objects  
 objects\_id = [job['objectid'] for job in jobs]  
  
 # все id сущностей в одном списке (без дублей)  
 entities\_id = list(set(relations\_id + objects\_id))  
  
 # все сущности  
 entities = e.get(subject\_id, entities\_id)  
  
 # находим и дописываем к каждой job 'relation' и 'object'. При наличии пары из entities сохраняем в plan\_jobs[]. Без пары игнорируются.  
 plan\_jobs = []  
 for job in jobs:  
 append\_job = False  
 for entity in entities:  
 if job['relationid'] == entity['id']:  
 job.update({'relation': entity})  
 append\_job = True  
 if job['objectid'] == entity['id']:  
 job.update({'object': entity})  
 append\_job = True  
 if append\_job:  
 plan\_jobs.append(job)  
  
 # проходим циклом по каждому plan\_job и делаем текст  
 for counter, plan\_job in enumerate(plan\_jobs, 1):  
  
 # текст с делом  
 if today:  
 jobs\_for\_period += str(counter) + '. '  
 else:  
 jobs\_for\_period += str(counter) + '. ' + get\_formatted\_date(plan\_job['plandate']) + '. '  
  
 jobs\_for\_period += plan\_job['relation']['name'] + '. ' + plan\_job['object']['name'] + '\n'  
  
  
 except Exception as err:  
 log.log('=== Crone ===. ' + 'make\_jobs\_for\_period: ' + str(err), severity='error',  
 facility='week\_tasks\_make\_list')  
 try:  
 # отправляем jobs сотруднику  
 # message\_text = cm.create\_message(jobs\_for\_period, [subject\_id], [])  
  
 # отправляем бригадиру сообщение по jobs сотрудника  
 brig\_message = f"<b>{e.get(subject\_id, subject\_id)[0]['name']}:</b>\n{jobs\_for\_period}"  
 message\_text = cm.create\_message(brig\_message, brigadiers, [])  
  
 log.log('=== Cron ===. ' + 'make\_jobs\_for\_period for subject\_id: ' + str(subject\_id),  
 severity='debug',  
 facility='week\_tasks\_make\_list')  
 # print(message\_text)  
 except Exception as err:  
 log.log('=== Crone ===. ' + 'make\_jobs\_for\_period: ' + str(err), severity='error',  
 facility='week\_tasks\_make\_list')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 3 - группа "Совместный труд"  
 #make\_list([3])  
 # 10004 - группа "Садов гиганта"  
 log.log('=== cron === Send jobs for period ', severity='info', facility='week\_task')  
 make\_list([10004])

**utils.py** (А. Караткевич, М. Демченко, Н. Шаверин)

import datetime  
import calendar  
import core\_profile  
  
  
def verification\_date(period: str) -> str:  
 """  
 функция прверяет и при необходимости исправляет ввходной параметр period  
 **:param** period: приходит из report\_status  
 **:return**: либо исправленный параметр period , либо none если не удалось обработать  
 """  
 plus\_minus = ['+', '-']  
 format\_date = ['d', 'w', 'm', 'y']  
 # разбиваем данные и удаляем пробелы  
 period = period.split()  
 # объединяем вновь в строку  
 period = "".join(period)  
  
 # если только один символ и это один из трех в format\_date, то возвращаем его  
 if len(period) == 1 and period in format\_date:  
 return period  
 # если 2 символа и первый из них + или - а также второй в format\_date, то возвращаем его  
 elif len(period) == 2 and period[0] in plus\_minus and period[1] in format\_date:  
 return period + "1"  
 # если больше 2х символов и и первый из них + или -, а также второй в format\_date, а оставшиеся цифры, то возвращаем его  
 elif len(period) > 2 and period[0] in plus\_minus and period[1] in format\_date and period[2:].isdecimal():  
 return period  
 # если входной формат -1w, преобразуем в -w1  
 elif len(period) > 2 and period[0] in plus\_minus and period[-1] in format\_date and period[1:-1].isdecimal():  
 return period[0] + period[-1] + period[1:-1]  
 # в противном случае возвраща  
 else:  
 return None  
  
  
def period2date(period: str, time\_zone: int = 0) -> datetime:  
 """  
 функция преобразования формата даты от телеграмбота в date формат  
  
 **:param** period: str -d2 -d1 d +d1 +d2, -w2, +w1, -m3, +m2  
  
 **:return**: start\_date и stop\_date в формате yy-mm-dd  
 """  
 # результат  
 ret\_start\_date = ''  
 ret\_stop\_date = ''  
  
 # текущая дата  
 current\_date = datetime.datetime.today()  
  
 # =================================================================  
 # ДОБАВЛЕНО 07.05.2023  
 # в аргументах функции тоже есть изменения  
 # текущая дата с учётом часового пояса пользователя  
 current\_date = current\_date + datetime.timedelta(hours = time\_zone)  
 # =================================================================  
  
 # текущий год  
 current\_year = current\_date.year  
  
 # текущий месяц  
 current\_month = current\_date.month  
  
 # текущее число  
 current\_day = current\_date.day  
  
 # номер дня в неделе  
 day\_of\_week = calendar.weekday(current\_year, current\_month, current\_day)  
  
 # разбираем формат даты  
 # в plus\_minus знак +или-  
 # в symbol знак d, w, m  
 # в quantity количество  
  
 if len(period) > 1:  
 plus\_minus = period[0]  
 symbol = period[1:2]  
 quantity = int(period[2:])  
 else:  
 symbol = period  
 plus\_minus = "!"  
  
 if symbol == "d":  
 if plus\_minus == "+":  
 stop\_date = current\_date + datetime.timedelta(days=quantity)  
 start\_date = stop\_date  
 elif plus\_minus == "-":  
 start\_date = current\_date - datetime.timedelta(days=quantity)  
 stop\_date = start\_date  
 else:  
 start\_date = current\_date  
 stop\_date = current\_date  
  
 elif symbol == "w":  
 if plus\_minus == "+":  
 first\_day\_week = 7 \* quantity - day\_of\_week  
 start\_day = current\_date + datetime.timedelta(days=first\_day\_week)  
 start\_date = start\_day  
 stop\_date = start\_day + datetime.timedelta(days=6)  
  
 elif plus\_minus == "-":  
 first\_day\_week = 7 \* quantity + day\_of\_week  
 start\_day = current\_date - datetime.timedelta(days=first\_day\_week)  
 start\_date = start\_day  
 stop\_date = start\_day + datetime.timedelta(days=6)  
  
 else:  
 start\_date = current\_date - datetime.timedelta(day\_of\_week)  
 stop\_date = start\_date + datetime.timedelta(days=6)  
  
 elif symbol == "m":  
 if plus\_minus == "+":  
 future\_date = current\_date + datetime.timedelta(days=quantity \* 30)  
 future\_date = datetime.datetime(future\_date.year, future\_date.month, 1)  
 start\_date = future\_date  
 day\_in\_month = calendar.monthrange(future\_date.year, future\_date.month)[1]  
 stop\_date = datetime.datetime(future\_date.year, future\_date.month, day\_in\_month)  
  
 elif plus\_minus == "-":  
 future\_date = current\_date - datetime.timedelta(days=quantity \* 30)  
 future\_date = datetime.datetime(future\_date.year, future\_date.month, 1)  
 start\_date = future\_date  
 day\_in\_month = calendar.monthrange(future\_date.year, future\_date.month)[1]  
 stop\_date = datetime.datetime(future\_date.year, future\_date.month, day\_in\_month)  
 else:  
 start\_date = datetime.datetime(current\_year, current\_month, 1)  
 day\_in\_month = calendar.monthrange(current\_year, current\_month)[1]  
 stop\_date = datetime.datetime(current\_year, current\_month, day\_in\_month)  
  
 elif symbol == "y":  
 if plus\_minus == "+":  
 current\_year = current\_year + quantity  
 if plus\_minus == "-":  
 current\_year = current\_year - quantity  
 start\_date = datetime.datetime(current\_year, 1, 1)  
 stop\_date = datetime.datetime(current\_year, 12, 31)  
  
 try:  
 ret\_start\_date = start\_date.date()  
 except Exception:  
 pass  
 try:  
 ret\_stop\_date = stop\_date.date()  
 except Exception:  
 pass  
  
 return ret\_start\_date, ret\_stop\_date  
  
  
def add\_date\_in\_report(report: dict, time\_zone: int = 0) -> dict:  
 """  
 функция в случае валидных данных добавляет в report даты начала и конца периода  
 **:param** report:  
 **:return**: report  
 """  
 validate\_period = verification\_date(report['period'])  
 if validate\_period:  
 d1, d2 = period2date(validate\_period, time\_zone)  
 report["d1"] = str(d1)  
 report["d2"] = str(d2)  
 return report  
  
  
def change\_date\_format (sql\_date: str)->str:  
 """функция преобразует входной формат даты '2023-01-01' в читаемый '01-01-2023'  
 **:param** sql\_date: str 2023-01-01  
 **:return**: str 01-01-2023  
 """  
 d\_list = sql\_date.split('-')  
 eu\_d = d\_list[2]+'/'+d\_list[1]+'/'+d\_list[0]  
 return eu\_d

**core\_Entity.py** (Д. Орлов, М. Закиев, Д. Джурабаев, М. Демченко, А. Караткевич, Н. Шаверин)

#  
# Запись, чтение и поиск сущностей в таблице Entity  
#  
#  
# search( pattern ) - поиск сущностей по шаблону.  
# get ( idlist ) - выдача параметров/описания задач  
# update( entitydict ) - создание/обновление задач  
# delete( idlist ) - удаление задач (пометка на удаление)  
#  
#  
  
import json  
import core\_config as conf  
import core\_log as log  
import core\_pg as pg  
import re  
  
  
#  
# 1 SEARCH  
#  
# на входе:  
# uid - от имени кого искать (id из entity).  
# pattern - json объект со структурой сущности, поля которого являются шаблоном поиска. Если поле указано, то необходимо составить по нему условие и вставить в запрос  
# на выходе список, найденных задач  
#  
def search(uid, pattern):  
 #  
 # складываем ответ  
 #  
 result = list()  
  
 #  
 # Query  
 #  
 query = " select array\_agg(id) from entity where "  
  
 #  
 # ready to LIMIT and OFFSET  
 #  
 offset = None  
 limit = None  
  
 # make query  
 where = " isrights( " + str(uid) + ", id, 'r' )='t' and is\_active='1' "  
 for k in pattern:  
 log.log("pattern " + str(k) +" => " + str(pattern[k]), severity="debug", facility="core\_entity")  
  
 #  
 # parent id = All nodes who in parent and all parent branch  
 #  
 if k == 'parentid':  
 if len(where) > 0:  
 where += ' AND '  
 if isinstance(pattern[k], int):  
 where += " array[" + str(pattern[k]) + "]&&path\_parent\_id(id)='t' "  
 elif isinstance(pattern[k], list):  
 where += " array[" + str(pattern[k]) + "]&&path\_parent\_id(id)='t' "  
  
 #  
 # group = All nodes who included in this groups  
 #  
 if k == 'group':  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += " array[" + str(pattern[k]) + "]&&get\_user\_groups(id)='t' "  
 elif isinstance(pattern[k], list):  
 where += " array[" + str(pattern[k]) + "]&&get\_user\_groups(id)='t' "  
  
 #  
 # name  
 #  
 if k == 'name':  
 if len(where) > 0:  
 where += ' AND '  
  
 where += str(k) + " ilike '%" + str(pattern[k]) + "%' "  
  
 #  
 # type  
 #  
 if k == 'type':  
 if len(where) > 0:  
 where += ' AND '  
  
 # single value  
 if isinstance(pattern[k], str):  
 where += str(k) + " = '" + str(pattern[k]) + "'"  
 elif isinstance(pattern[k], list):  
 where += str(k) + " IN ('" + str("','".join(pattern[k])) + "') "  
  
 #  
 # uid  
 #  
 if k == 'uid':  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += str(k) + ' = ' + str(pattern[k])  
 elif isinstance(pattern[k], list):  
 where += str(k) + ' = ANY( ARRAY' + str(pattern[k]) + ') '  
  
 #  
 # gid - any group from gid or Groups. PL function allgroups(id) -  
 #  
 if k == 'gid':  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += str(pattern[k]) + ' = ' + str(pattern[k])  
 elif isinstance(pattern[k], list):  
 where += str(k) + ' = ANY( ARRAY' + str(pattern[k]) + ') '  
  
 #  
 # id - any id from entity. Добавлено для поиска id от другого id (для отправки сообщений)  
 #  
 if k == 'id':  
 # Убираем права. Потом надо будет разобраться и вернуть  
 where = re.sub(r'isrights.+and', '', where)  
  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += str(pattern[k]) + ' = ' + str(pattern[k])  
 # list  
 elif isinstance(pattern[k], list):  
 where += str(k) + ' = ANY( ARRAY' + str(pattern[k]) + ') '  
  
 #  
 # parameters  
 #  
 if k == "telegramid":  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += "(parameters->'telegram'->'id')::int = " + str(pattern[k])  
 elif isinstance(pattern[k], list):  
 where += "(parameters->'telegram'->'id')::int = ANY( ARRAY [" + str(pattern[k]) + ']) '  
  
 #  
 # LIMIT  
 #  
 if k == "limit":  
 # single value  
 if isinstance(pattern[k], list):  
 limit = str(pattern[k][0])  
 else:  
 limit = str(pattern[k])  
  
  
 #  
 # OFFSET  
 #  
 if k == "offset":  
 # single value  
 if isinstance(pattern[k], list):  
 offset = str(pattern[k][0])  
 else:  
 offset = str(pattern[k])  
 #  
 # ADD WHERE  
 #  
 query += where  
  
 #  
 # ADD OFFSET LIMIT  
 #  
 if limit != None or offset != None:  
 query += ' ORDER BY id '  
  
 if limit != None:  
 query += " LIMIT " + limit  
  
 if offset != None:  
 query += " OFFSET " + offset  
  
 log.log("query: " + query, severity="debug", facility="core\_entity")  
  
 #  
 # connect database  
 #  
 #  
 #  
 #  
  
 #  
 # Читаем конфиг  
 #  
 # config = conf.configuration()  
  
 rows = pg.execute(query)  
  
 #  
 # ответ  
 #  
 if rows:  
 result = rows[0][0]  
  
 return result  
  
  
#  
# GET - получить параметры сущностей  
#  
# на входе -  
# id - от имени кого надо искать  
# список id - список, который вернул search  
# на выходе - массив объектов  
#  
#  
def get(uid, idlist):  
 #  
 # result  
 #  
 result = list()  
  
 if idlist:  
  
 #  
 # prepare query  
 #  
 query = "select e2.id, e2.parentid, path\_parent\_name(e2.id) as path\_parent\_name, path\_parent\_id(e2.id) as path\_parent\_id, e2.uid, e2.gid, e2.rights, e2.type, e2.name, e2.parameters->'options' as options, get\_user\_groups(id), e2.parameters->'description' as description from entity e2 where "  
  
 #  
 # проверка прав  
 #  
 where = " isrights( " + str(uid) + ", id, 'r' )='t' and is\_active='1' AND "  
  
 if isinstance(idlist, list):  
 where += ' id in ('  
 for i in range(len(idlist)):  
 if i > 0:  
 where += ','  
 where += str(idlist[i])  
 where += ') '  
 else:  
 where += ' id = ' + str(idlist)  
 #  
 #  
 #  
 query += where  
  
 log.log("query:" + query, severity="debug", facility="core\_entity")  
  
 rows = pg.execute(query)  
 for row in rows:  
  
 if len(row):  
 one = dict()  
 try:  
 one.update({"id": int(row[0])})  
 except:  
 pass  
 try:  
 one.update({"parentid": int(row[1])})  
 except:  
 pass  
 try:  
 one.update({"path\_parent\_name": str(row[2])})  
 except:  
 pass  
 try:  
 one.update({"path\_parent\_id": row[3]})  
 except:  
 pass  
 try:  
 one.update({"uid": int(row[4])})  
 except:  
 pass  
 try:  
 one.update({"gid": int(row[5])})  
 except:  
 pass  
 try:  
 one.update({"rights": str(row[6])})  
 except:  
 pass  
 try:  
 one.update({"type": str(row[7])})  
 except:  
 pass  
 try:  
 one.update({"name": str(row[8])})  
 except:  
 pass  
 try:  
 one.update({"options": row[9]})  
 except:  
 pass  
 try:  
 one.update({"groups": row[10]})  
 except:  
 pass  
 try:  
 one.update({"description": row[11]})  
 except:  
 pass  
  
 result.append(one)  
  
 #  
 # return  
 #  
 return result  
  
  
  
#  
# insert/update  
#  
# на входе - id, objjson  
# id - от чьего имени спрашивать права на запись  
# objjson - массив объектов для сохранения  
#  
#  
def update(uid, objects):  
 #  
 # gid - группа для создания новых объектов.  
 # группа берётся у пользователя создающего объект.  
 # Запросим группу при первом обращении к Insert  
 #  
 gid = 0  
  
 #  
 # if object - one element. Prepare array with one element  
 #  
 if isinstance(objects, dict):  
 arr = list()  
  
 arr.append(objects)  
 objects = arr  
  
 #  
 # теперь мы всегда имеем массив элементов для update|insert  
 #  
 #  
 if isinstance(objects, list):  
 #  
 # update  
 #  
 for obj in objects:  
  
 #  
 # Insert OR update  
 #  
 # check id: if id exists - update  
 #  
 #  
 if 'id' in obj and obj['id']:  
 #  
 # update  
 #  
 query = ""  
 set = ''  
  
 try:  
 if int(obj['parentid']) > 0:  
 if len(set)>0:  
 set += ', '  
  
 set += 'parentid = ' + str(obj['parentid'])  
 except Exception:  
 pass  
  
 try:  
 if int(obj['uid']) > 0:  
 if len(set)>0:  
 set += ', '  
  
 set += 'uid = ' + str(obj['uid'])  
 except Exception:  
 pass  
  
 try:  
 if int(obj['gid']) > 0:  
 if len(set)>0:  
 set += ', '  
  
 set += 'gid = ' + str(obj['gid'])  
 except Exception:  
 pass  
  
 try:  
 if obj['rights'] != "None":  
 if len(set)>0:  
 set += ', '  
  
 set += "rights = '" + str(obj['rights']) +"'"  
 except Exception:  
 pass  
  
 try:  
 if obj['type'] != "None":  
 if len(set)>0:  
 set += ', '  
  
 set += "type = '" + str(obj['type']) + "'"  
 except Exception:  
 pass  
  
 try:  
 if obj['name'] != "None":  
 if len(set)>0:  
 set += ', '  
  
 set += "name = '" + str(obj['name']) + "'"  
 except Exception:  
 pass  
  
 try:  
 if obj['parameters'] != "None":  
 if len(set)>0:  
 set += ', '  
  
 set += 'parameters = ' + str(obj['parameters'])  
 except Exception:  
 pass  
  
 #  
 # where  
 #  
 query += "Update entity SET " + set + " where isrights( " + str(  
 uid) + ", id, 'w' )='t' and is\_active='1' and id = " + str(obj['id']) +" returning id"  
  
  
 #  
 #  
 # INSERT  
 #  
 #  
 else:  
  
 #  
 # Перед созданием объекта нужно взять группу пользователя, создающего объект - это будет группа для нового объекта  
 #  
 if gid == 0:  
  
 query = "select gid from entity where id=" + str(uid)  
 val = pg.execute(query)  
 try:  
 gid = int(val)  
 except:  
 gid = 0  
  
 #  
 # query  
 #  
 query = ""  
  
 fields = ""  
 values = ""  
  
 try:  
 if int(obj['parentid']) > 0:  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'parentid'  
 values += str(obj['parentid'])  
 except Exception:  
 pass  
  
 #  
 # 'uid' in obj:  
 #  
 if len(fields):  
 fields += ', '  
 values += ', '  
 fields += ' uid'  
 values += str(uid)  
  
 #  
 # if 'gid' in obj:  
 #  
 if len(fields):  
 fields += ', '  
 values += ', '  
 fields += ' gid'  
 values += str(gid)  
  
 #  
 # 'rights' in obj:  
 #  
 if len(fields):  
 fields += ', '  
 values += ', '  
 fields += 'rights'  
 values += "'rw-r-----'"  
  
 try:  
 if obj['type'] != "None":  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'type'  
 values += "'" + str(obj['type']) + "'"  
 except Exception:  
 pass  
  
 try:  
 if obj['name'] != "None":  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'name'  
 values += "'" + str(obj['name']) + "'"  
 except Exception:  
 pass  
  
 try:  
 if obj['parameters'] != "None":  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'parameters'  
 values += "'" + str(obj['parameters']) + "'"  
 except Exception:  
 pass  
  
 #  
 # is\_active = 1  
 #  
 fields += ', is\_active'  
 values += ", '1'"  
  
 query += "Insert into entity (" + fields + ") values( " + values + ") returning id"  
  
 #  
 #  
 #  
 #  
 log.log( "query: "+query, severity="debug", facility="core\_Entity" )  
 ret=pg.execute(query)  
  
 #  
 #  
 # return  
 #  
 if ret != None:  
 if isinstance( ret, list ) and len(ret)>0:  
 tmplist = list()  
 for r in ret:  
 tmplist.append( r[0] )  
 ret = tmplist  
 return ret  
   
   
   
   
   
   
  
  
# DELETE - удалить сущности (пометить как удалённые)  
#  
# на входе -  
# id - от имени кого надо удалить  
# список id - список, который вернул search  
#  
# 1 Работаем с каждым в списке по отдельности  
# 2. Проверяем права на объект  
# 3. Если прав хватает, то перепривязываем своих "детей" к своему "родителю"  
# 4. Помечаем, как удалённый  
#  
#  
def delete(uid, idlist):  
 #  
 # result  
 #  
 result = list()  
  
 #  
 # 1 Работаем с каждым в списке по отдельности  
 #  
 for l in idlist:  
  
 #  
 # 2. Проверяем права на объект  
 #  
 query = "select 1 from entity where isrights( " + str(uid) + ", id, 'w' ) and is\_active='1' and id=" + str(l)  
 f = pg.execute(query)  
 if len(f) > 0:  
 #  
 # Если прав хватает, то перепривязываем своих "детей" к своему "родителю"  
 #  
 query = "select parentid from entity where id=" + str(l)  
 p = pg.execute(query)  
 if len(p) > 0:  
 p = p[0][0]  
  
 query = "update entity set parentid=" + str(  
 p) + " where id in (select id from entity where parentid=" + str(l) + ")"  
 pg.execute(query)  
  
 #  
 # prepare query  
 #  
 query = " update entity set is\_active='0' where "  
  
 #  
 # права  
 #  
 where = " isrights( " + str(uid) + ", id, 'w' ) and is\_active='1' AND "  
  
 if isinstance(idlist, int):  
 where += ' id = ' + str(idlist)  
 elif isinstance(idlist, list):  
 where += ' id in ('  
 for i in range(len(idlist)):  
 if i > 0:  
 where += ','  
 where += str(idlist[i])  
 where += ') '  
 #  
 #  
 #  
 query += where  
  
 rows = pg.execute(query)  
  
  
#  
# GET - получить массив из параметров сущностей  
# сейчас массивами являются:  
# у пользователя  
# search\_ref - сохранённые запросы  
# reports - параметры отчётов  
#  
#  
#  
# на входе -  
# uid - от имени кого надо искать  
# objid - у какого объекта искать параметры  
# arrayname - имя массива  
# на выходе - массив  
#  
#  
def getParametersArray(uid, objid, arrayname):  
 #  
 # result  
 #  
 result = list()  
  
 log.log("getParametersArray ( " + str(uid) + ", " + str(objid) + ", " + arrayname + " )", severity="debug",  
 facility="coreEntity")  
  
 #  
 # Проверка входных параметров  
 #  
 if uid > 0 and objid > 0 and arrayname > '':  
  
 #  
 # prepare query  
 #  
 query = "select jsonb\_array\_elements( e2.parameters->'" + arrayname + "') from entity e2 where "  
  
 #  
 # проверка прав  
 #  
 where = " isrights( " + str(uid) + ", id, 'r' )='t' and is\_active='1' AND "  
  
 if isinstance(objid, int):  
 where += ' id = ' + str(objid)  
 elif isinstance(objid, list):  
 where += ' id in ('  
 for i in range(len(objid)):  
 if i > 0:  
 where += ','  
 where += str(objid[i])  
 where += ') '  
 #  
 #  
 #  
 query += where  
  
 # print( 'Query> ', query )  
  
 log.log("query: " + str(query), severity="debug", facility="coreEntity")  
  
 rows = pg.execute(query)  
 for row in rows:  
  
 if len(row):  
 try:  
 result.append(row[0])  
 except Exception:  
 pass  
  
 #  
 # return  
 #  
 return result  
  
  
#  
# ADD - добавить элемент/массив элементов в массив из параметров сущностей  
# сейчас массивами являются:  
# у пользователя  
# search\_ref - сохранённые запросы  
# reports - параметры отчётов  
#  
#  
#  
# на входе -  
# uid - от имени кого надо искать  
# objid - у какого объекта искать параметры  
# arrayname - имя массива  
# element - элемент для добавления. Может быть массивом  
# на выходе - массив  
#  
#  
def addParametersArray(uid, objid, arrayname, element):  
 #  
 # result  
 #  
 result = True  
  
 log.log("addParametersArray ( " + str(uid) + ", " + str(objid) + ", " + arrayname + ", " + str(element) + ")",  
 severity="debug", facility="coreEntity")  
  
 #  
 # Проверка входных параметров  
 #  
 if uid > 0 and objid > 0 and arrayname > '':  
  
 #  
 # prepare query  
 #  
 query = """update entity set parameters = jsonb\_set( parameters, '{""" + arrayname + """}', parameters->'""" + arrayname + """'||'""" + str(  
 element) + """'::jsonb) """  
  
 query += ' where '  
  
 #  
 # проверка прав  
 #  
 where = " isrights( " + str(uid) + ", id, 'w' )='t' and is\_active='1' AND "  
  
 if isinstance(objid, int):  
 where += ' id = ' + str(objid)  
 elif isinstance(objid, list):  
 where += ' id in ('  
 for i in range(len(objid)):  
 if i > 0:  
 where += ','  
 where += str(objid[i])  
 where += ') '  
 #  
 #  
 #  
 query += where  
  
 log.log("query: " + str(query), severity="debug", facility="coreEntity")  
  
 try:  
 rows = pg.execute(query)  
 except Exception as err:  
 result = False  
  
 #  
 # return  
 #  
 return result  
  
  
#  
# GET - получить value из параметров сущностей по ключу  
#  
#  
# на входе -  
# uid - от имени кого надо искать  
# objid - у какого объекта искать параметры  
# key - ключ  
# на выходе - value  
#  
#  
def getParametersValue(uid, objid, key):  
  
 value = None  
  
 #  
 # Проверка входных параметров  
 #  
 if uid > 0 and objid > 0 and key > '':  
  
 #  
 # prepare query  
 #  
 query = "select e2.parameters->'" + key + "' from entity e2 where "  
  
 #  
 # проверка прав  
 #  
 where = " isrights( " + str(uid) + ", id, 'r' )='t' and is\_active='1' AND "  
  
 if isinstance(objid, int):  
 where += ' id = ' + str(objid)  
 elif isinstance(objid, list):  
 where += ' id in ('  
 for i in range(len(objid)):  
 if i > 0:  
 where += ','  
 where += str(objid[i])  
 where += ') '  
 #  
 #  
 #  
 query += where  
  
 log.log("query: " + str(query), severity="debug", facility="coreEntity")  
  
 try:  
 value = pg.execute(query)[0][0]  
 log.log("getParametersValue ( " + str(uid) + ", " + str(objid) + ", " + key + " )", severity="debug",  
 facility="coreEntity")  
 except Exception as err:  
 value = None  
  
 #  
 # return  
 #  
 return value  
  
  
#  
# SET - установить значение по ключу в parameters  
#  
# на входе -  
# uid - от имени кого надо менять  
# objid - у какого объекта менять параметры. может быть списком  
# key - имя ключа  
# value - новое значение  
# на выходе - boolean  
#  
#  
def setParametersValue(uid, objid, key, value):  
 #  
 # result  
 #  
 result = True  
  
 log.log("setParametersValue ( " + str(uid) + ", " + str(objid) + ", " + key + ", " + str(value) + ")",  
 severity="debug", facility="coreEntity")  
  
 #  
 # Проверка входных параметров  
 #  
 if uid > 0 and objid > 0 and key > '':  
  
 #  
 # prepare query  
 #  
 query = "update entity set parameters = jsonb\_set( parameters, '{ "  
 query += key  
 query += " }', '"  
 query += str(value).replace("""'""",'"')  
 query += "'::jsonb, true)"  
  
 query += ' where'  
 #  
 # проверка прав  
 #  
 where = " isrights( " + str(uid) + ", id, 'w' )='t' and is\_active='1' AND "  
  
 if isinstance(objid, int):  
 where += ' id = ' + str(objid)  
 elif isinstance(objid, list):  
 where += ' id in ('  
 for i in range(len(objid)):  
 if i > 0:  
 where += ','  
 where += str(objid[i])  
 where += ') '  
 #  
 #  
 #  
 query += where  
  
  
 log.log("query: " + str(query), severity="debug", facility="coreEntity")  
  
 try:  
 rows = pg.execute(query)  
 except Exception as err:  
 result = False  
  
 #  
 # return  
 #  
 return result  
  
  
  
#  
# REMOVE - удалить элемент/массив элементов из массива из параметров сущностей  
# сейчас массивами являются:  
# у пользователя  
# search\_ref - сохранённые запросы  
# reports - параметры отчётов  
#  
#  
#  
# на входе -  
# uid - от имени кого надо искать  
# objid - у какого объекта искать параметры  
# arrayname - имя массива  
# element - элемент для удаления. Может быть массивом  
# элемент будет задан ключ:значение  
# или [ ключ: значение, ключ: значение,...]  
# на выходе - массив  
#  
#  
def removeParametersArray(uid, objid, arrayname, element):  
 #  
 # result  
 #  
 result = True  
  
 log.log("removeParametersArray ( " + str(uid) + ", " + str(objid) + ", " + arrayname + ", " + str(element) + ")",  
 severity="debug", facility="coreEntity")  
  
 #  
 # Проверка входных параметров  
 #  
 if uid > 0 and objid > 0 and arrayname > '':  
  
 #  
 # Проверяем входные параметры.  
 # На вход должны поступать структуры ключ:значение, например: {"title":"Весь отчёт"}  
 # Если на входе одиночный ответ, то приведём его к массиву - универсиализируем дальнейшую обработку  
 #  
 inputelements = list()  
 if isinstance(element, list):  
 inputelements = element  
 else:  
 inputelements.append(element)  
  
 #  
 # Удаляем каждый пришедший элемент поотдельности (в запросе удаление идёт по индексу в массиве)  
 #  
 for elm in inputelements:  
  
 #  
 # prepare query  
 #  
 query = """update entity set parameters = jsonb\_set ( parameters, '{""" + arrayname + """}', (parameters->'""" + arrayname + """')::jsonb - (  
 select arr.position::int   
 from entity, jsonb\_array\_elements(parameters->'""" + arrayname + """') with ordinality arr(title, position)   
 where """  
  
 #  
 # Для удаления элемента нужен его индекс в массиве  
 # индекс ищем в зависимости от типа переданных параметров  
 #  
 # dict - это как задумывалось  
  
 if isinstance(elm, dict):  
  
 #  
 # для всех элементов справочника  
 # для вставки AND если элемент не единственный  
 #  
 firstelm = True  
 for k in elm.keys():  
 if not firstelm:  
 query += " and "  
  
 query += """title->>'""" + str(k) + """' = '""" + str(elm[k]) + "'"  
  
 #  
 # добавляем для какого id мы ищем параметры  
 #  
 where = """ and """  
 if isinstance(objid, int):  
 where += ' id = ' + str(objid)  
 elif isinstance(objid, list):  
 where += ' id in ('  
 for i in range(len(objid)):  
 if i > 0:  
 where += ','  
 where += str(objid[i])  
 where += ') '  
  
 query += where + """ )  
 )   
 """  
  
 #  
 # проверка прав  
 #  
 where = " isrights( " + str(uid) + ", id, 'w' )='t' and is\_active='1' AND "  
  
 if isinstance(objid, int):  
 where += ' id = ' + str(objid)  
 elif isinstance(objid, list):  
 where += ' id in ('  
 for i in range(len(objid)):  
 if i > 0:  
 where += ','  
 where += str(objid[i])  
 where += ') '  
 #  
 #  
 #  
 query += " where " + where  
  
 log.log("query: " + str(query), severity="debug", facility="coreEntity")  
  
 try:  
 rows = pg.execute(query)  
 except Exception as err:  
 result = False  
  
 #  
 # return  
 #  
 return result  
  
  
#  
# 1 SEARCH and GET  
# работает так же как SEARCH, но возвращает список расшифрованных entity  
#  
def searchget(uid, pattern):  
 #  
 # складываем ответ  
 #  
 result = list()  
  
 #  
 # Query  
 #  
 query = " select e2.id, e2.parentid, path\_parent\_name(e2.id) as path\_parent\_name, path\_parent\_id(e2.id) as path\_parent\_id, e2.uid, e2.gid, e2.rights, e2.type, e2.name, e2.parameters->'options' as options, get\_user\_groups(id), e2.parameters->'description' as description from entity e2 where "  
  
 #  
 # ready to LIMIT and OFFSET  
 #  
 offset = None  
 limit = None  
  
 # make query  
 where = " isrights( " + str(uid) + ", id, 'r' )='t' and is\_active='1' "  
 for k in pattern:  
 log.log("pattern " + str(k) +" => " + str(pattern[k]), severity="debug", facility="core\_entity")  
  
 #  
 # parent id = All nodes who in parent and all parent branch  
 #  
 if k == 'parentid':  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += " array[" + str(pattern[k]) + "]&&path\_parent\_id(id)='t' "  
 elif isinstance(pattern[k], list):  
 where += " array[" + str(pattern[k]) + "]&&path\_parent\_id(id)='t' "  
  
 #  
 # group = All nodes who included in this groups  
 #  
 if k == 'group':  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += " array[" + str(pattern[k]) + "]&&get\_user\_groups(id)='t' "  
 elif isinstance(pattern[k], list):  
 where += " array[" + str(pattern[k]) + "]&&get\_user\_groups(id)='t' "  
  
 #  
 # name  
 #  
 if k == 'name':  
 if len(where) > 0:  
 where += ' AND '  
  
 where += str(k) + " ilike '%" + str(pattern[k]) + "%' "  
  
 #  
 # type  
 #  
 if k == 'type':  
 if len(where) > 0:  
 where += ' AND '  
  
 # single value  
 if isinstance(pattern[k], str):  
 where += str(k) + " = '" + str(pattern[k]) + "'"  
 elif isinstance(pattern[k], list):  
 where += str(k) + " IN ('" + str("','".join(pattern[k])) + "') "  
  
 #  
 # uid  
 #  
 if k == 'uid':  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += str(k) + ' = ' + str(pattern[k])  
 elif isinstance(pattern[k], list):  
 where += str(k) + ' = ANY( ARRAY' + str(pattern[k]) + ') '  
  
 #  
 # gid - any group from gid or Groups. PL function allgroups(id) -  
 #  
 if k == 'gid':  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += str(pattern[k]) + ' = ' + str(pattern[k])  
 elif isinstance(pattern[k], list):  
 where += str(k) + ' = ANY( ARRAY' + str(pattern[k]) + ') '  
  
 #  
 # id - any id from entity. Добавлено для поиска id от другого id (для отправки сообщений)  
 #  
 if k == 'id':  
 # Убираем права. Потом надо будет разобраться и вернуть  
 where = re.sub(r'isrights.+and', '', where)  
  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += str(pattern[k]) + ' = ' + str(pattern[k])  
 # list  
 elif isinstance(pattern[k], list):  
 where += str(k) + ' = ANY( ARRAY' + str(pattern[k]) + ') '  
  
 #  
 # parameters  
 #  
 if k == "telegramid":  
 if len(where) > 0:  
 where += ' AND '  
 # single value  
 if isinstance(pattern[k], int):  
 where += "(parameters->'telegram'->'id')::int = " + str(pattern[k])  
 elif isinstance(pattern[k], list):  
 where += "(parameters->'telegram'->'id')::int = ANY( ARRAY [" + str(pattern[k]) + ']) '  
  
 #  
 # LIMIT  
 #  
 if k == "limit":  
 # single value  
 if isinstance(pattern[k], list):  
 limit = str(pattern[k][0])  
 else:  
 limit = str(pattern[k])  
  
  
 #  
 # OFFSET  
 #  
 if k == "offset":  
 # single value  
 if isinstance(pattern[k], list):  
 offset = str(pattern[k][0])  
 else:  
 offset = str(pattern[k])  
 #  
 # ADD WHERE  
 #  
 query += where  
  
 #  
 # ADD OFFSET LIMIT  
 #  
 if limit != None or offset != None:  
 query += ' ORDER BY id '  
  
 if limit != None:  
 query += " LIMIT " + limit  
  
 if offset != None:  
 query += " OFFSET " + offset  
  
  
 log.log("query: " + query, severity="debug", facility="core\_entity")  
  
 #  
 # connect database  
 #  
 #  
 #  
 #  
  
 #  
 # Читаем конфиг  
 #  
 # config = conf.configuration()  
  
 rows = pg.execute(query)  
 for row in rows:  
  
 if len(row):  
 one = dict()  
 try:  
 one.update({"id": int(row[0])})  
 except:  
 pass  
 try:  
 one.update({"parentid": int(row[1])})  
 except:  
 pass  
 try:  
 one.update({"path\_parent\_name": str(row[2])})  
 except:  
 pass  
 try:  
 one.update({"path\_parent\_id": row[3]})  
 except:  
 pass  
 try:  
 one.update({"uid": int(row[4])})  
 except:  
 pass  
 try:  
 one.update({"gid": int(row[5])})  
 except:  
 pass  
 try:  
 one.update({"rights": str(row[6])})  
 except:  
 pass  
 try:  
 one.update({"type": str(row[7])})  
 except:  
 pass  
 try:  
 one.update({"name": str(row[8])})  
 except:  
 pass  
 try:  
 one.update({"options": row[9]})  
 except:  
 pass  
 try:  
 one.update({"groups": row[10]})  
 except:  
 pass  
 try:  
 one.update({"dscription": row[11]})  
 except:  
 pass  
  
 result.append(one)  
  
  
 return result

**core\_log.py** (Д. Орлов, М. Закиев, Д. Джурабаев, М. Демченко, А. Караткевич, Н. Шаверин)

#  
# логирование сообщений   
#  
##########################  
from os.path import join, dirname  
import os  
from datetime import datetime  
import time  
  
import sys  
import core\_config as cf  
  
#  
# Формат записей в логе  
#  
# DataTime Severity Facility text  
# где  
# Datatime - текущее время Функция логирования возьмёт сама на момент поступления сообщения  
# Severity - серьёзность события. Чем выше, тем серьёзней. Основной момент. До Error - продолжать работать можно, выше приложение продолжить работу не может.  
# emergency/critical  
# error  
# warning - возможно, что-то не указано, но система взяла значения по умолчанию и продолжает работать  
# notice -  
# info - по умолчанию  
# debug  
#  
# facility - система/модуль/функция, которая отправляет сообщение  
#  
def log( text, severity='info', facility="none" ):  
  
  
 #  
 # читаем конфигурацию  
 #  
 config = cf.configuration()  
  
 #  
 # лог файл  
 #  
 log\_file = config['common']['homedir'] + config['common']['logfile']  
  
 #  
 # Файл блокировки (lock)  
 # в одно время только один кто-то может писать в файл, иначе он испортится.  
 # для этого процесс выставляет Флаг (Lock), пишет в файл, удаляет Флаг (unlock)  
 # Остальные ждут пока предыдущий снимет блокировку  
 #  
 lock\_file = config['common']['homedir'] + config['common']['lockfile']  
  
 #  
 # Чтобы не зависнуть в ожидании блокировки навечно сделаем счётчик максимального количества попыток  
 # если он пройдёт - просто выйдем из функции без записи в лог  
 #  
 max\_try = 7  
  
 #  
 # ждём удаления предыдущей блокировки  
 #  
 while os.path.exists( lock\_file ) and max\_try > 0:  
 time.sleep(1)  
 max\_try -= 1  
  
 #  
 # Если блокировка снята, но не максимальное кол-во попыток исчерпано  
 #  
 if max\_try > 0:  
  
  
 try:  
 #  
 # Мы выставляем Lock  
 #  
  
 # lock = open( lock\_file, "x" )  
 # Переписал работу с lock\_file  
 with open(lock\_file, 'x') as lock\_fhndl:  
 # формируем запись  
 lock\_fhndl.write('BLOCK')  
  
 #  
 # берём текущую дату  
 #  
 dt = datetime.now()  
 today = dt.strftime("%Y-%m-%d %H:%M:%S")  
  
 #  
 # PID  
 #  
 pid = os.getpid()  
  
 #  
 # открываем log файл для записи  
 #  
 with open(log\_file, 'a') as log\_fhndl:  
 # log\_fhndl = open( log\_file, "a" )  
 # формируем запись  
 s = f'{today}\t{str(severity):10} '  
 if len(str(facility)) < 10:  
 s += f'{str(facility):10} '  
 else:  
 s += f'{str(facility)} '  
 s += f'[{str(pid)}]\t{str(text)}\n'  
 log\_fhndl.write(s)  
 # log\_fhndl.close ()  
  
  
 except Exception as err:  
 # print(f'Core\_log. Ошибка записи {log\_file}. {err}')  
 # даже если ошибка, ни чего не делаем  
 pass  
  
 #  
 # unlock  
 #  
 try:  
 os.remove(lock\_file)  
 # print(f'Core\_log. {lock\_file} удалён')  
 except Exception as err:  
 # print(f'Core\_log. Не получилось удалить {lock\_file}. Ошибка {err}')  
 pass

**core\_message.py** (Д. Орлов, М. Закиев, Д. Джурабаев, М. Демченко, А. Караткевич, Н. Шаверин)

from datetime import datetime  
import json  
  
import core\_pg as pg  
import core\_Entity as ent  
import core\_log as log  
import core\_profile as cp  
  
  
def create\_message(text, uid=[], gid=[], plan=f'{datetime.now():%Y-%m-%d %H:%M:%S}', parse\_mode='html'):  
 *"""Функция для формирования сообщений в БД"""* count\_message = 0  
 try:  
 if len(uid) != 0:  
 if len(gid) != 0:  
 # получим список id членов всех gid  
 groups\_members\_id = cp.get\_groups\_entities(gid)  
 # print(f'groups\_members\_id - {groups\_members\_id}')  
  
 # по id ищем только subject  
 rows = ent.search(uid[0], {  
 'id': groups\_members\_id,  
 'type': 'subject'  
 })  
  
 # добавляем полученные из групп id в общий список id  
 if rows:  
 for g in rows:  
 if g not in uid:  
 uid.append(g)  
  
 # Сообщения отправляются всем id одним запросом  
 query = "INSERT INTO message(uid, plandate, text, parse\_mode) VALUES \n"  
 for i in uid:  
 query += "("  
 query += str(i)  
 query += ", '"  
 query += plan  
 query += "', '"  
 query += text  
 query += "', '"  
 query += str(parse\_mode)  
 query += "')"  
 query += ","  
 query += '\n'  
 count\_message += 1  
 # уберём последнюю запятую  
 query = query[:-2]  
 pg.execute(query)  
  
 else:  
 return None  
  
 except Exception as err:  
 log.log(" Can't create message " + str(err), severity="error", facility="core\_message")  
  
 return f"Сообщений сформировано: {count\_message}"  
  
  
def message\_sent(mid=[]):  
 *""" Функция изменения статуса отправки сообщения на отправленное"""* for m in mid:  
 query = "UPDATE message SET senddate = '"  
 query += f"{datetime.now():%Y-%m-%d %H:%M:%S}'"  
 query += " WHERE mid = "  
 query += str(m)  
 pg.execute(query)  
  
  
def message\_search(uid=[], gid=[], offset=None, not\_sent=False):  
 *"""Функция поиска определенных сообщений"""* mid\_list = []  
 query = "SELECT \* FROM message WHERE "  
 if len(uid) >= 1:  
 query += "uid IN "  
 query += str(tuple(uid))  
 if len(gid) >= 1:  
 ids = ent.search(23, {'gid': gid})  
 for i in ids:  
 uid.append(i)  
 query += "uid IN "  
 query += str(tuple(uid))  
  
 if len(uid) == 1:  
 query += "uid = "  
 query += str(uid[0])  
  
 if offset is not None:  
 query += " AND mid >= "  
 query += str(offset)  
  
 if not\_sent is True:  
 query += " AND senddate is NULL"  
  
 elif not\_sent is True:  
 query += "senddate is NULL"  
  
 elif offset is not None:  
 query += " mid >= "  
 query += str(offset)  
 rows = pg.execute(query)  
 for row in rows:  
 mid\_list.append(row[5])  
  
 return tuple(mid\_list)  
  
  
def message\_delete(mid=[]):  
 *"""Удаление сообщений"""* if len(mid) >= 1:  
 for m in mid:  
 query = "DELETE FROM message WHERE mid = "  
 query += str(m)  
 pg.execute(query)  
 else:  
 pass  
  
  
def message\_get(mid=[]):  
 *"""Функция возвращает список сообщений"""* one\_message = dict()  
 message\_list = list()  
  
 if mid:  
  
 query\_select = "SELECT \* FROM message WHERE mid IN "  
 query\_select += str(tuple(mid))  
 query\_update = "UPDATE message SET senddate = '"  
 query\_update += f"{datetime.now():%Y-%m-%d %H:%M:%S}'"  
 query\_update += " WHERE mid IN "  
 query\_update += str(tuple(mid))  
 pg.execute(query\_update)  
 messages = pg.execute(query\_select)  
 for message in messages:  
 if message:  
 one\_message.update({'mid': int(message[5])})  
 one\_message.update({'uid': int(message[0])})  
 one\_message.update({'plandate': str(message[1])})  
 one\_message.update({'senddate': str(message[2])})  
 one\_message.update({'parse\_mode': str(message[4])})  
 one\_message.update({'text': str(message[3])})  
 message\_list.append(one\_message)  
  
 return message\_list  
  
  
def message\_to\_sent():  
 query = "select (select parameters->'telegram'->'id' from entity where id = m.uid) as tlgid, mid, text, parse\_mode " \  
 "from message m where plandate is not null and senddate is null;"  
 messages = pg.execute(query)  
  
 return messages  
  
  
def search( user\_id, srch='my', gid=None, period=None, offset=None ):  
 #  
 # result array  
 #  
 mid = list()  
  
  
 #  
 # query  
 #  
 query = "select array( select mid from message"  
 where =''  
  
 if srch == 'my':  
 where += " uid = "+str(user\_id)  
  
 elif gid == 'group':  
 #  
 # select gid  
 #  
 if gid:  
 where += " and uid in (select id from entity where "+str(gid)+" = ANY( allgroups(id) )"  
  
 if period:  
 #  
 # add d1 - d2  
 # or add d1  
 if isinstance( period, list ):  
 if len( period ) >1:  
 where += " and plandate between '"+str(period[0])+"' and '"+str(period[1])+"'"  
 else:  
 where += " and plandate >= '"+str(period[0])+"'"  
 else:  
 where += " and plandate = '"+str(period)+"'"  
  
 if offset:  
 #  
 # add min mid  
 #  
 where += " and mid >= "+str(offset)  
  
 #  
 # add Where to query  
 #  
 query += " where " + where + ")"  
  
 log.log( "query: "+query, severity='debug', facility='core\_message' )  
  
 try:  
 mid = pg.execute( query )[0][0]  
 except Exception as err:  
 log.log( "Can't search message: "+str(err), severity='error', facility='core\_message' )  
  
 return mid  
  
#  
#  
# GET ( [mid1, mid2,...] ) - return message text  
#  
#  
def get( user\_id, mid\_idx ):  
  
 #  
 # return data  
 #  
 data = list()  
  
  
 #  
 # query  
 #  
 query = "Select mid, uid, plandate, senddate, text, parse\_mode from message "  
 where = " uid = "+str( user\_id )+" and "  
  
  
  
 #  
 # if mid\_idx is Array - Query is .... mid = ANY (..)  
 # if mid\_idx is int - Query is .... mid = mid\_idx  
 #  
 if isinstance( mid\_idx, list ):  
 where += " mid = ANY( array"+str(mid\_idx)+ ")"  
 elif isinstance( mid\_idx, int ) or isinstance( mid\_idx, str ):  
 where += " mid = "+ str( mid\_idx )  
  
 #  
 # add Where to query  
 #  
 query += " where " + where  
 log.log( "query: "+query, severity='debug', facility='core\_message' )  
  
 try:  
 msg = pg.execute( query )  
  
 for m in msg:  
 one = dict()  
 one.update( {"mid" : m[0]} )  
 one.update( {"uid" : m[1]} )  
 one.update( {"plan": m[2].strftime("%Y-%m-%d")} )  
 one.update( {"fact": m[3].strftime("%Y-%m-%d")} )  
 one.update( {"msg" : m[4]} )  
 one.update( {"parse\_mode" : m[5]} )  
  
 data.append( one )  
  
  
 except Exception as err:  
 log.log( "Can't get message: "+str(err), severity='error', facility='core\_message' )  
  
  
 return data

**core\_profile.py** (Д. Орлов, М. Закиев, Д. Джурабаев, М. Демченко, А. Караткевич, Н. Шаверин)

#  
# Функции проверки профиля   
#  
# set\_hash - вернуть hash для идентификации польльзователя. Сохранить хэш в БД  
# check\_hash - проверить, есть ли кто-то откликнулся с нашим хэшем (кто-то выполнил accept\_hash)  
# accept\_hash - принять hash  
# delete\_hash - удалить   
# get\_userid\_by\_telegramid - проверяет есть ли такой телеграм ID в нашей БД  
# newuser - создать нового пользователя  
# get\_report - вернуть отчёты пользователя  
# get\_telegram - вернуть параметры Телеграм  
# get\_groups - вернуть список всех групп пользователя  
#  
#  
import time  
import random  
  
  
import core\_config as cf  
import core\_log as log  
import core\_pg as pg  
import core\_Entity as entity  
  
#  
# set\_hash   
#  
# 1 генерирует хэш  
# 2 записывает хэш в БД  
#   
def set\_hash(socket, newhash=None):  
 #  
 # Вычисляем время таймаута, после которого прекращаем авторизацию  
 #  
 dt = int(time.time()) + 60  
  
 #  
 # 1 если хэш не задан, то генерируем его  
 #  
 if newhash == None:  
 #  
 # 1 Генерируем шестизначное число  
 newhash = str(random.randint(111111, 999999))  
 # log.log( socket + " new hash: " +str(newhash), severity="debug", facility="core\_profile" )  
  
 #  
 # Записываем в БД  
 #  
 try:  
 query = "INSERT INTO authwait (hash, socket, timeout) values('" + str(newhash) + "','" + str(socket) + "',to\_timestamp(" + str(dt) + ") )"  
 # log.log( socket+ " set hash query: " +query, severity="debug", facility="core\_profile" )  
  
 pg.execute(query)  
 except Exception as err:  
 log.log( str(socket) + " set hash: " + str(err), severity="error", facility="core\_profile")  
  
 return newhash  
  
  
#  
# chek\_hash  
#  
# 1. запрашиваем из таблицы telegramid для нашего хэш  
#  
#  
def check\_hash(socket, newhash):  
 #  
 # вернём telegramid  
 #  
 tlg\_id = False  
  
  
  
 #  
 # готовим запрос проверки ответа от телеграмма  
 #  
 query = """SELECT  
 (select id from entity e where e.type='subject' and e.parameters->'telegram'->>'id' = a.telegramid) as user\_id,  
 socket,  
 telegramid  
 FROM authwait a where a.hash = '""" + str(newhash) + """' AND socket='""" + str(socket) + """' AND a.telegramid is not NULL limit 1"""  
 #  
 # выполняем запрос  
 #  
 try:  
 rows = pg.execute(query)  
 if len(rows):  
 tlg\_id = rows[0][0]  
  
 except Exception as err:  
 log.log( str(socket) + " check hash: " + str(err), severity="error", facility="core\_profile")  
  
 return tlg\_id  
  
  
  
  
  
#  
# accept\_hash  
#  
#  
#  
def accept\_hash(tlgid, newhash):  
 query = """UPDATE authwait set telegramid = """ + str(tlgid) + " where now() < timeout and hash = '" + str(newhash) + "'"  
 try:  
 pg.execute(query)  
  
 except Exception as err:  
 log.log(socket + " Accept hash: " + str(err), severity="error", facility="core\_profile")  
  
  
  
  
  
#  
# delete\_hash  
#  
#  
#  
def delete\_hash(socket, newhash):  
 query = "DELETE from authwait where hash='" + str(newhash) + "'"  
 try:  
 pg.execute(query)  
  
 except Exception as err:  
 log.log(socket + " Delete hash: " + str(err), severity="error", facility="core\_profile")  
  
  
#  
# get profile by telegramid  
#  
#  
#  
def get\_id\_by\_telegramid(tlgid):  
 user\_id = None  
  
 rows = list()  
  
 try:  
 if tlgid:  
 query = "select id from entity where parameters->'telegram'->>'id' = '" + str(tlgid) + "'"  
 rows = pg.execute(query)  
 except Exception as err:  
 log.log( " Can't get id by telgramID "+str(tlgid)+": " + str(err), severity="error", facility="core\_profile")  
  
 #  
 # если вернулась одна строка (это отлично) - вернём одно число  
 if len(rows) == 1:  
 user\_id = rows[0][0]  
  
 #  
 # Если больше одной строки - значит, что-то таких абонентов несколько - вернём массив  
 elif len(rows) > 1:  
 user\_id = list()  
 for row in rows:  
 user\_id.append(row[0])  
  
 return user\_id  
  
  
  
  
#  
#  
# get Any Parameters. paramname - list  
#  
#  
def get\_parameters( user\_id, paramname ):  
  
 param = None  
  
 try:  
 param = entity.getParametersArray( user\_id, user\_id, paramname )  
 except Exception as err:  
 log.log(" Can't get parameter "+str(paramname)+" for user "+str(user\_id)+": " + str(err), severity="error", facility="core\_profile")  
  
 return param  
  
  
#  
#  
# get Any Parameters. key - key  
#  
#  
def get\_parameters\_value( user\_id, key ):  
  
 param = None  
  
 try:  
 param = entity.getParametersValue( user\_id, user\_id, key )  
 except Exception as err:  
 log.log(" Can't get parameter "+str(key)+" for user "+str(user\_id)+": " + str(err), severity="error", facility="core\_profile")  
  
 return param  
  
  
#  
#  
# get Reports - return  
#  
#  
def get\_reports( user\_id ):  
  
 param = get\_parameters( user\_id, 'reports' )  
  
 return param  
  
  
#  
#  
# get telegram - return  
#  
#  
def get\_telegram( user\_id ):  
  
 param = get\_parameters( user\_id, 'telegram' )  
  
 return param  
  
  
#  
#  
# get Options - return  
#  
#  
def get\_options( user\_id ):  
  
 param = get\_parameters( user\_id, 'options' )  
  
 return param  
  
  
  
  
#  
#  
# New User - Create new Group + User  
# return user\_id  
#  
def newuser( username, parameters ):  
  
 user\_id = None  
  
 try:  
 query = f"Select newuser( '{username}','{parameters}' )"  
 pg\_data = pg.execute( query )  
 user\_id = pg\_data[0][0]  
  
 except Exception as err:  
 log.log("newuser(). Can't create new user "+username+" with parameters "+str(parameters)+": "+str(err), severity="error", facility="core\_profile")  
  
 return user\_id  
  
  
  
#  
#  
# get user groups  
#  
#  
def get\_groups( user\_id ):  
  
 data = None  
  
 try:  
 query = "select get\_user\_groups("+str( user\_id ) +")"  
 data = pg.execute( query )[0][0]  
 except Exception as err:  
 log.log(" Can't get user's group: " + str(err), severity="error", facility="core\_profile")  
  
 return data  
  
  
#  
#  
# get groups entities  
# На входе список групп, на выходе список id сущностей, входящих в эти группы  
#  
#  
def get\_groups\_entities(groups: list) -> list:  
 data = None  
  
 try:  
 str\_groups = '['  
 for group in groups:  
 str\_groups += str(group) + ', '  
 str\_groups = str\_groups[:-2] + ']'  
 query = "select get\_group\_entities ( ARRAY " + str\_groups + " )"  
 data = pg.execute(query)[0][0]  
 except Exception as err:  
 log.log(" Can't get group`s entities: " + str(err), severity="error", facility="core\_profile")  
  
 return data  
  
#  
#  
# get user name  
#  
#  
def get\_username( user\_id ):  
  
 data = None  
  
 try:  
 data = entity.get( user\_id, user\_id )  
 except Exception as err:  
 log.log(" Can't get user's name: " + str(err), severity="error", facility="core\_profile")  
  
 return data

**core\_job.py** (Д. Орлов, М. Закиев, Д. Джурабаев, М. Демченко, А. Караткевич, Н. Шаверин)

#  
# Запись, чтение и поиск job в таблице Plan  
#  
#  
# search( jobpattern ) - поиск задач по шаблону  
# get ( idlist ) - выдача параметров/описания задач  
# update( jobsdict ) - создание/обновление задач  
# delete( idlist ) - удаление задач (пометка на удаление)  
#  
#  
  
from datetime import datetime  
import json  
import core\_config as conf  
import core\_log as log  
import core\_pg as pg  
  
  
  
#  
# 1 SEARCH  
#  
# на входе json объект со структурой задачи, поля которого являются шаблоном поиска. Если поле указано, то необходимо составить по нему условие и вставить в запрос  
# на выходе список, найденных задач  
#  
def search( jobpattern ):  
 #  
 # складываем ответ  
 #  
 answer = list()  
   
 #log.log( "job search pattern: "+ json.dumps( jobpattern, indent=4 ), severity="debug", facility="core\_job" )  
  
 #  
 # составляем запрос  
 #  
 query = "select array\_agg(id) from plan "  
  
 #  
 # Where =   
 #  
 # period and type  
 # AND   
 # Jobs  
 #  
 where = ""  
   
 #  
 # Jobs  
 #  
 Jobs = ""  
   
   
 #  
 # Jobs  
 #  
 #  
 #  
 if 'jobs' in jobpattern:  
   
 for onejob in jobpattern['jobs']:  
 if len( Jobs ) > 0:  
 Jobs += " OR "  
   
 jobwhere=""  
 if 'subject' in onejob:  
 if isinstance( onejob['subject'], list ):  
 jobwhere += " subjectid in (" + ",".join( str(i) for i in onejob['subject'] ) + ") "  
 else:  
 jobwhere += " subjectid in (" + str(onejob['subject']) + ") "  
 if 'object' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
 if isinstance( onejob['object'], list ):  
 jobwhere += " objectid in (" + ",".join( str(i) for i in onejob['object'] ) + ") "  
 else:  
 jobwhere += " objectid in (" + str(onejob['object']) + ") "  
 if 'relation' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
 if isinstance( onejob['relation'], list ):  
 jobwhere += " relationid in (" +",".join( str(i) for i in onejob['relation'] ) + ") "  
 else:  
 jobwhere += " relationid in (" + str(onejob['relation']) + ") "  
 if 'project' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
 if isinstance( onejob['project'], list ):  
 jobwhere += " projectid in (" +",".join( str(i) for i in onejob['project'] ) + ") "  
 else:  
 jobwhere += " projectid in (" + str(onejob['project']) + ") "  
 if 'parent' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
 grplist = str(onejob['parent']).lstrip("[").rstrip("]")  
 jobwhere += " ( (array[" + grplist + "] && path\_parent\_id( subjectid )) or (array[" + grplist + "] && path\_parent\_id( objectid )) or (array[" + grplist + "] && path\_parent\_id( relationid )) )"  
 if 'group' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
   
 grplist = str(onejob['group']).lstrip("[").rstrip("]")  
 jobwhere += " ( (array[" + grplist + "] && get\_user\_groups( subjectid )) or (array[" + grplist + "] && get\_user\_groups( objectid )) or (array[" + grplist + "] && get\_user\_groups( relationid )) )"  
   
 Jobs +="("+jobwhere+ ")"  
   
 #  
 # Period & Variant  
 #   
 #  
 #  
 #  
 if 'variant' not in jobpattern:  
 jobpattern.update( {'variant':'all'} )  
 if 'period' not in jobpattern:  
 dt = datetime.today()  
 jobpattern.update( {'period':dt.strftime('%Y-%m-%d')} )  
 elif isinstance( jobpattern['period'], list ):  
 try:  
 if jobpattern['period'][0] == None:  
 jobpattern['period'][0] = '2020-01-01'  
 except:  
 jobpattern['period'].append( '2020-01-01' )  
  
 try:  
 if jobpattern['period'][1] == None:  
 jobpattern['period'][1] = '2030-01-01'  
 except:  
 jobpattern['period'].append( '2030-01-01' )  
  
   
 #  
 # ALL  
 #  
 if jobpattern['variant'] == 'all':  
 if len(where) > 0:  
 where += ' AND '  
   
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " ( (plandate is NOT NULL) or (factdate is NOT NULL) ) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate befor factdate is null or after  
 where += " ( "  
 where += " (plandate >= '"+jobpattern['period'][0]+"' and plandate <= '"+jobpattern['period'][1]+"') "  
 where += " or (factdate >= '"+jobpattern['period'][0]+"' and factdate <= '"+jobpattern['period'][1]+"') "  
 where += " ) "  
  
 #  
 # PLAN  
 #  
 elif jobpattern['variant'] == 'plan':  
 if len(where) > 0:  
 where += ' AND '  
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " (plandate >= '"+jobpattern['period']+"') and ( factdate is null ) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate in period and factdate is null or after  
 where += " (factdate is null) and "  
 where += " (plandate >= '"+jobpattern['period'][0]+"' and plandate <= '"+jobpattern['period'][1]+"') "  
   
   
 #  
 # FACT  
 #  
 elif jobpattern['variant'] == 'fact':  
 if len(where) > 0:  
 where += ' AND '  
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " ( factdate is not null ) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate in period and factdate is null or after  
 where += " (factdate >= '"+jobpattern['period'][0]+"' and factdate <= '"+jobpattern['period'][1]+"') "  
   
   
   
 #  
 # DEBT  
 #  
 elif jobpattern['variant'] == 'debt':  
 if len(where) > 0:  
 where += ' AND '  
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " ((plandate < '"+jobpattern['period']+"') and ( factdate is null )) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate in period and factdate is null or after  
 where += " (factdate is null) and "  
 where += " (plandate >= '"+jobpattern['period'][0]+"' and plandate <= '"+jobpattern['period'][1]+"') "  
   
   
   
 #  
 # ERR  
 #  
 elif jobpattern['variant'] == 'err':  
 if len(where) > 0:  
 where += ' AND '  
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " (plandate is null) and "  
 where += " ( factdate is null or factdate > '"+jobpattern['period']+"' ) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate in period and factdate is null or after  
 where += " (plandate is null) and "  
 where += " ( factdate is null or factdate > '"+jobpattern['period'][1]+"' ) "  
   
  
 #  
 # finish query  
 #  
 if len(where):  
 query += " WHERE (" + where + ") "  
 if len( Jobs ) > 0:  
 query += " AND (" + Jobs +")"  
 elif len( Jobs ) > 0:  
 query += " Where (" + Jobs +")"  
   
   
 log.log( "query: "+ query, severity="debug", facility="core\_job" )  
   
   
 #  
 # execute query  
 #  
 rows = pg.execute( query )  
   
 #  
 # ответ  
 #  
 if rows:  
 answer = rows[0][0]  
  
 return answer  
  
  
#  
# 2 GET  
#  
# на входе список мз id задач, на выходе список из id и параметрами задач, либо пустой список, если по заданным параметрам задача не найдена  
#  
  
def get( idlist ):  
 #  
 # сюда сложим результат  
 # или он вернётся пустым   
 #  
 answer=list()  
  
  
 #  
 # читаем конфиг. Ищем переменные для подключения к БД  
 #  
 #cfg = conf.configuration()  
 #host = cfg['database']['host']  
 #user = cfg['database']['user']  
 #pass = cfg['database']['pass']  
 #dbname = cfg['database']['dbname']  
  
  
 jobs\_list = []  
 rows = []  
  
 #  
 # если список пришёл не пустой  
 #  
 if idlist:  
  
 #  
 # составляем запрос  
 #  
 query = "select id, subjectid, relationid, objectid, plandate, factdate, anoncedate, projectid from plan "  
 where = ""  
 #  
 # where  
 #  
 if isinstance( idlist, int ):  
 where += " id = " + str(idlist)  
   
 elif isinstance( idlist, list ):  
 where += " id in ("  
 for i in range(len(idlist)):  
 if i>0:  
 where += ','  
 where += str( idlist[i] )  
 where += ')'  
  
  
 #finish query  
 query += " WHERE " + where  
 log.log( "query: "+ query, severity="debug", facility="core\_job" )  
  
 #  
 # execute query  
 #  
 rows = pg.execute( query )  
  
 for row in rows:  
 if row:  
 #print( "ROW: " + str(row) )  
 onerow = dict()  
 try:   
 onerow.update( { 'id' : int( str(row[0]) ) })  
 except:  
 onerow.update( { 'id' : 0 })  
 try:   
 onerow.update( { 'subjectid' : int( str(row[1]) ) })  
 except:  
 onerow.update( { 'subjectid' : 0 })  
 try:   
 onerow.update( { 'relationid': int( str(row[2]) ) })  
 except:  
 onerow.update( { 'relationid': 0})  
 try:   
 onerow.update( { 'objectid' : int( str(row[3]) ) })  
 except:  
 onerow.update( { 'objectid' : 0})  
 try:   
 onerow.update( { 'plandate' : str(row[4]) })  
 except:  
 onerow.update( { 'plandate' : ''})  
 try:   
 onerow.update( { 'factdate' : str(row[5]) })  
 except:  
 onerow.update( { 'factdate' : ''})  
 try:   
 onerow.update( { 'anoncedate': str(row[6]) })  
 except:  
 onerow.update( { 'anoncedate': ''})  
 try:   
 onerow.update( { 'projectid' : int( str(row[7]) ) })  
 except:  
 onerow.update( { 'projectid' : 0})  
   
 #  
 # add to answer  
 #  
 answer.append( onerow )  
  
 #log.log( "Get can't parse row: "+str(err), severity="error", facility="core\_job" )  
  
  
 return answer  
  
  
#  
#  
# 3 UPDATE( job ) - создание/обновление задач  
#  
# Если id задачи задано - update  
# иначе insert  
#  
def update( jobs ):  
  
 #  
 # return None - default  
 #  
 ret=None  
  
 #  
 # if jobs - one element. Prepare array with one element  
 #  
 if isinstance( jobs, dict ):  
  
 arr = list()  
  
 arr.append( jobs )  
 jobs = arr  
  
  
 #  
 # теперь мы всегда имеем массив элементов для update|insert  
 # id | subjectid | relationid | objectid | plandate | factdate | anoncedate | prev | next | projectid | projectmark  
 #----+-----------+------------+----------+------------+------------+------------+------+------+-----------+-------------  
 # 1 | 12 | 9 | 18 | 2022-09-01 | 2022-09-01 | | | | |  
 # 2 | 12 | 9 | 16 | 2022-09-02 | 2022-09-02 | | | | |  
  
 #  
 if isinstance( jobs, list ):  
 #  
 # update  
 #  
 for obj in jobs:  
   
 #log.log( "Update jobs:"+json.dumps( obj ), severity='debug', facility='core\_job' )  
  
 #  
 # Insert OR update  
 #  
 # check id: if id exists - update  
 #  
 #  
 if 'id' in obj and obj['id']:  
  
 #  
 # UPDATE  
 #  
 query = "Update plan SET "  
 set = ''  
   
 try:  
 if int( obj['subject'] ) > 0:  
 if len(set):  
 set += ', '  
  
 set += 'subjectid = ' + str( obj['subject'] )  
 except Exception:  
 pass  
  
  
 try:  
 if int( obj['relation'] ) > 0:  
 if len(set):  
 set += ', '  
  
 set += 'relationid = ' + str( obj['relation'] )  
 except Exception:  
 pass  
  
  
 try:  
 if int( obj['object'] ) > 0:  
 if len(set):  
 set += ', '  
  
 set += 'objectid = ' + str( obj['object'] )  
 except Exception:  
 pass  
  
  
 try:  
 if obj['plandate'] != "None":  
 if len(set):  
 set += ', '  
  
 set += 'plandate = ' + "'" + str( obj['plandate'] ) +"'"  
 except Exception:  
 pass  
  
  
 try:  
 if obj[ 'factdate'] != "None":  
 if len(set):  
 set += ', '  
  
 set += 'factdate = ' + "'" + str( obj['factdate'] ) +"'"  
 except Exception:  
 pass  
  
  
 try:  
 if obj['anoncedate'] != "None":  
 if len(set):  
 set += ', '  
  
 set += 'anoncedate = ' + "'" + str( obj['anoncedate'] ) +"'"  
 except Exception:  
 pass  
  
  
  
 try:  
 if int( obj['prev'] ) > '0':  
 if len(set):  
 set += ', '  
  
 set += 'prev = ' + str( obj['prev'] )  
 except Exception:  
 pass  
  
  
 try:  
 if int( obj['next']) > 0:  
 if len(set):  
 set += ', '  
  
 set += 'next = ' + str( obj['next'] )  
 except Exception:  
 pass  
  
  
 try:  
 if int( obj['project'] ) > 0:  
 if len(set):  
 set += ', '  
  
 set += 'projectid = ' + str( obj['project'] )  
 except Exception:  
 pass  
  
  
 try:  
 if obj[ 'projectmark' ] != 'None':  
 if len(set):  
 set += ', '  
  
 set += 'projectmark = ' + "'" + str( obj['projectmark'] ) +"'"  
 except Exception:  
 pass  
  
  
  
  
 #  
 # where  
 #  
 if isinstance( obj['id'], list ):  
 tmpstr = ','.join( str(n) for n in obj['id'] )  
 query += set + " where id in (" + tmpstr + ") returning id"  
   
 else:  
 query += set + " where id = " + str( obj['id'] ) + " returning id"  
  
  
 #  
 #  
 # INSERT  
 #  
 #  
 else:  
  
 #  
 # query  
 #  
 query = ""  
  
 fields=""  
 values=""  
  
 try:  
 if int( obj['subject'] ) > 0:  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'subjectid'  
 values += str( obj['subject'] )  
 except Exception:  
 pass  
  
  
  
 try:  
 if int( obj['relation'] ) > 0:  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'relationid'  
 values += str( obj['relation'] )  
 except Exception:  
 pass  
  
  
 try:  
 if int( obj['object'] ) > 0:  
 if len(fields):  
 fields += ', '  
 values += ', '  
   
 fields += 'objectid'  
 values += str( obj['object'] )  
 except Exception:  
 pass  
  
  
 try:  
 if obj['plandate'] != "None":  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'plandate'  
 values += "'"+str( obj['plandate'] )+"'"  
 except Exception:  
 pass  
  
  
 try:  
 if obj['factdate'] != "None":  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'factdate'  
 values += "'"+str( obj['factdate'] )+"'"  
 except Exception:  
 pass  
  
  
 try:  
 if obj['anoncedate'] != "None":  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'anoncedate'  
 values += "'"+str( obj['anoncedate'] )+"'"  
 except Exception:  
 pass  
  
  
 try:  
 if int(obj['prev']) > 0 :  
 if len(fields):  
 fields += ', '  
 values += ', '  
   
 fields += 'prev'  
 values += str( obj['prev'] )  
 except Exception:  
 pass  
  
  
  
 try:  
 if int( obj['next'] ) > 0:  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'next'  
 values += str( obj['next'] )  
 except Exception:  
 pass  
  
  
 try:  
 if int( obj['project'] ) > 0:  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'projectid'  
 values += str( obj['project'] )  
 except Exception:  
 pass  
  
  
 try:  
 if obj['projectmark'] != "None":  
 if len(fields):  
 fields += ', '  
 values += ', '  
  
 fields += 'projectmark'  
 values += "'"+str( obj['projectmark'] )+"'"  
 except Exception:  
 pass  
  
  
 #  
 # Make query  
 #  
 query = "Insert into plan (" + fields + ") values( " + values + ") returning id"  
  
  
 #  
 # query output  
 #  
 #print( "Query> ", query )  
  
 #  
 # execute  
 #  
 if query:  
 ret = pg.execute( query )   
  
 #  
 # return  
 #  
 if ret != None:  
 if isinstance( ret, list ) and len(ret)>0:  
 tmplist = list()  
 for r in ret:  
 tmplist.append( r[0] )  
 ret = tmplist  
 return ret  
  
  
  
  
  
  
#  
# DELETE - удалить работы   
#  
# на входе - список id - список, который вернул search  
#.  
# 1 Удалим всех в списке  
#  
#  
def delete( idlist ):  
 #  
 # inlist  
 #  
 inlist = ''  
  
  
 if isinstance( idlist, int ):  
 #  
 # если значение одно  
 #  
 inlist = str(idlist)  
   
 elif isinstance( idlist, list ):  
  
 #  
 # составим список, через запятую  
 #  
 for l in idlist:  
 if len( inlist ) > 0:  
 inlist += ','  
 inlist += str( l )  
  
 #  
 # prepare delete query  
 #  
 query = "Delete from plan where id in (" + inlist + ")"  
 #print( "Query> " + query )  
  
  
 #  
 # execute  
 #  
 pg.execute( query )   
  
  
  
#  
# COMMIT - установить факт выполнения работы  
#  
# Устанавливает factdate всем задачам, переданным как массив id  
# Если factdate не задан, то ставит текущую дату  
#  
def commit( idlist, newfactdate=None ):  
 #  
 # вернём Ок  
 #  
 answer=False  
  
  
 #  
 # Если newfactdate не задан, берём текущую дату  
 #  
 if newfactdate == None or newfactdate == '':  
 dt = datetime.today()  
 newfactdate = dt.strftime("%Y-%m-%d")  
   
 #  
 # собираем запрос  
 #  
 query = ''  
 try:  
   
 query = "Update plan set factdate='" + str(newfactdate) + "'"  
 where = ""  
 if isinstance( idlist, int ):  
 where += " id = " + str( idlist )  
  
 elif isinstance( idlist, list ):  
 inlist = list()  
 for l in idlist:  
 if len( inlist ) > 0:  
 inlist += ','  
 inlist += str( l )  
 where += " id in (" + str( l ) +") "   
   
 query += " WHERE " + where  
  
 except Exception as err:  
 log.log( "Can't prepare query for Commit factdate: " + str(err), severity="error", facility="core\_job" )  
   
 try:  
 pg.execute( query )  
 answer = True  
 except Exception as err:  
 log.log( "Can't Commit factdate: " + str(err), severity="error", facility="core\_job" )  
   
  
 #  
 # answer  
 #  
 return answer  
  
  
  
#  
# 1 SEARCH and GET  
#  
# тоже самое что и get но выдаёт список с расшифровкой entity  
#  
#  
def searchget( jobpattern ):  
 #  
 # складываем ответ  
 #  
 answer = list()  
   
 #  
 # offset and limit  
 #  
 offset = None  
 limit = None  
   
   
 #log.log( "job searchget pattern: "+ json.dumps( jobpattern, indent=4 ), severity="debug", facility="core\_job" )  
  
 #  
 # составляем запрос  
 #  
 query = """select id,   
 (select e2.id ||','|| e2.type ||','|| e2.name from entity e2 where e2.id = j.subjectid),  
 (select e3.id ||','|| e3.type ||','|| e3.name from entity e3 where e3.id = j.relationid),  
 (select e4.id ||','|| e4.type ||','|| e4.name from entity e4 where e4.id = j.objectid),  
 plandate, factdate, anoncedate,   
 (select e5.id ||','|| e5.type ||','|| e5.name from entity e5 where e5.id = j.projectid)  
 from plan j """  
  
 #  
 # Where =   
 #  
 # period and type  
 # AND   
 # Jobs  
 #  
 where = ""  
   
 #  
 # Jobs  
 #  
 Jobs = ""  
   
   
 #  
 # Jobs  
 #  
 #  
 #  
 if 'jobs' in jobpattern:  
   
 for onejob in jobpattern['jobs']:  
 #print( "job: ", json.dumps( onejob, indent=4 ) )  
 if len( Jobs ) > 0:  
 Jobs += " OR "  
   
 jobwhere=""  
 if 'subject' in onejob:  
 if isinstance( onejob['subject'], list ):  
 jobwhere += " subjectid in (" + ",".join( str(i) for i in onejob['subject'] ) + ") "  
 else:  
 jobwhere += " subjectid in (" + str(onejob['subject']) + ") "  
 if 'object' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
 if isinstance( onejob['object'], list ):  
 jobwhere += " objectid in (" + ",".join( str(i) for i in onejob['object'] ) + ") "  
 else:  
 jobwhere += " objectid in (" + str(onejob['object']) + ") "  
 if 'relation' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
 if isinstance( onejob['relation'], list ):  
 jobwhere += " relationid in (" +",".join( str(i) for i in onejob['relation'] ) + ") "  
 else:  
 jobwhere += " relationid in (" + str(onejob['relation']) + ") "  
 if 'project' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
 if isinstance( onejob['project'], list ):  
 jobwhere += " projectid in (" +",".join( str(i) for i in onejob['project'] ) + ") "  
 else:  
 jobwhere += " projectid in (" + str(onejob['project']) + ") "  
 if 'parent' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
 jobwhere += " ( (array[" + str(onejob['parent']) + "] && path\_parent\_id( subjectid )='t') or (array[" + str(onejob['parent']) + "] && path\_parent\_id( objectid )='t') or (array[" + str(onejob['parent']) + "] && path\_parent\_id( relationid )='t') )"  
 if 'group' in onejob:  
 if len( jobwhere ) > 0:  
 jobwhere += " and "  
 jobwhere += " ( (array[" + str(onejob['group']) + "] && get\_user\_groups( subjectid )='t') or (array[" + str(onejob['group']) + "] && get\_user\_groups( objectid )='t') or (array[" + str(onejob['group']) + "] && get\_user\_groups( relationid )='t') )"  
   
 Jobs +="("+jobwhere+ ")"  
   
 #  
 # Period & Variant  
 #   
 #  
 #  
 #  
 if 'variant' not in jobpattern:  
 jobpattern.update( {'variant':'all'} )  
 if 'period' not in jobpattern:  
 dt = datetime.today()  
 jobpattern.update( {'period':dt.strftime('%Y-%m-%d')} )  
 elif isinstance( jobpattern['period'], list ):  
 try:  
 if jobpattern['period'][0] == None:  
 jobpattern['period'][0] = '2020-01-01'  
 except:  
 jobpattern['period'].append( '2020-01-01' )  
  
 try:  
 if jobpattern['period'][1] == None:  
 jobpattern['period'][1] = '2030-01-01'  
 except:  
 jobpattern['period'].append( '2030-01-01' )  
   
   
 #  
 # ALL  
 #  
 if jobpattern['variant'] == 'all':  
 if len(where) > 0:  
 where += ' AND '  
   
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " ( (plandate is NOT NULL) or (factdate is NOT NULL) ) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate befor factdate is null or after  
 where += " ( "  
 where += " (plandate >= '"+jobpattern['period'][0]+"' and plandate <= '"+jobpattern['period'][1]+"') "  
 where += " or (factdate >= '"+jobpattern['period'][0]+"' and factdate <= '"+jobpattern['period'][1]+"') "  
 where += " ) "  
  
 #  
 # PLAN  
 #  
 elif jobpattern['variant'] == 'plan':  
 if len(where) > 0:  
 where += ' AND '  
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " (plandate >= '"+jobpattern['period']+"') and ( factdate is null ) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate in period and factdate is null or after  
 where += " (factdate is null) and "  
 where += " (plandate >= '"+jobpattern['period'][0]+"' and plandate <= '"+jobpattern['period'][1]+"') "  
   
   
 #  
 # FACT  
 #  
 elif jobpattern['variant'] == 'fact':  
 if len(where) > 0:  
 where += ' AND '  
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " ( factdate is not null ) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate in period and factdate is null or after  
 where += " (factdate >= '"+jobpattern['period'][0]+"' and factdate <= '"+jobpattern['period'][1]+"') "  
   
   
   
 #  
 # DEBT  
 #  
 elif jobpattern['variant'] == 'debt':  
 if len(where) > 0:  
 where += ' AND '  
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " ((plandate < '"+jobpattern['period']+"') and ( factdate is null )) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate in period and factdate is null or after  
 where += " (factdate is null) and "  
 where += " (plandate >= '"+jobpattern['period'][0]+"' and plandate <= '"+jobpattern['period'][1]+"') "  
   
   
   
 #  
 # ERR  
 #  
 elif jobpattern['variant'] == 'err':  
 if len(where) > 0:  
 where += ' AND '  
   
 # Если дата задана одним числом то только за эту дату  
 #   
 if isinstance( jobpattern['period'], str ):  
 where += " (plandate is null) and "  
 where += " ( factdate is null or factdate > '"+jobpattern['period']+"' ) "  
   
 # Если дата список  
 #  
 elif isinstance( jobpattern['period'], list ):  
 # plandate in period and factdate is null or after  
 where += " (plandate is null) and "  
 where += " ( factdate is null or factdate > '"+jobpattern['period'][1]+"' ) "  
   
  
  
  
  
 #  
 # OFFSET and LIMIT  
 #  
 if 'offset' in jobpattern:  
 offset = jobpattern['offset']  
  
 if 'limit' in jobpattern:  
 limit = jobpattern['limit']  
   
   
  
  
  
 #  
 # finish query  
 #  
 if len(where):  
 query += " WHERE (" + where + ") "  
 if len( Jobs ) > 0:  
 query += " AND (" + Jobs +")"  
 elif len( Jobs ) > 0:  
 query += " Where (" + Jobs +")"  
   
  
  
  
 #  
 # ADD OFFSET and LIMIT  
 #  
 if offset != None or limit != None:  
 query += ' order by id '  
   
 if offset != None:  
 query += " OFFSET " + str(offset)  
  
 if limit != None:  
 query += " LIMIT " + str(limit)  
  
   
 log.log( "query: "+ str(query), severity="debug", facility="core\_job" )  
   
   
   
 #  
 # execute query  
 #  
 rows = pg.execute( query )  
   
 #  
 # ответ  
 #  
 for r in rows:  
 #log.log( "type r: "+ str(type(r)) + " len " + str(len(r)), severity="debug", facility="core\_job" )  
  
 try:  
 once = dict()  
   
 once.update( {"id" :r[0]} )  
 f = r[1].split(',')  
 if len(f)>0:  
 once.update( {"subjectid" :f[0]} )  
 once.update( {"subjecttype" :f[1]} )  
 once.update( {"subjectname" :f[2]} )  
 f = r[2].split(',')  
 if len(f)>0:  
 once.update( {"relationid" :f[0]} )  
 once.update( {"relationtype":f[1]} )  
 once.update( {"relationname":f[2]} )  
 f = r[3].split(',')  
 if len(f)>0:  
 once.update( {"objectid" :f[0]} )  
 once.update( {"objecttype" :f[1]} )  
 once.update( {"objectname" :f[2]} )  
 once.update( {"plandate" :str(r[4])} )  
 once.update( {"factdate" :str(r[5])} )  
 once.update( {"announcedate":str(r[6])} )  
 f = r[7].split(',')  
 if len(f)>0:  
 once.update( {"projectid" :f[0]} )  
 once.update( {"projecttype" :f[1]} )  
 once.update( {"projectname" :f[2]} )  
   
 answer.append( once )  
   
 except Exception as err:  
 log.log( "searchget add once: "+ str(err), severity="debug", facility="core\_job" )  
   
  
  
  
 return answer

6.2. Скриншоты работы веб-версии информационной системы

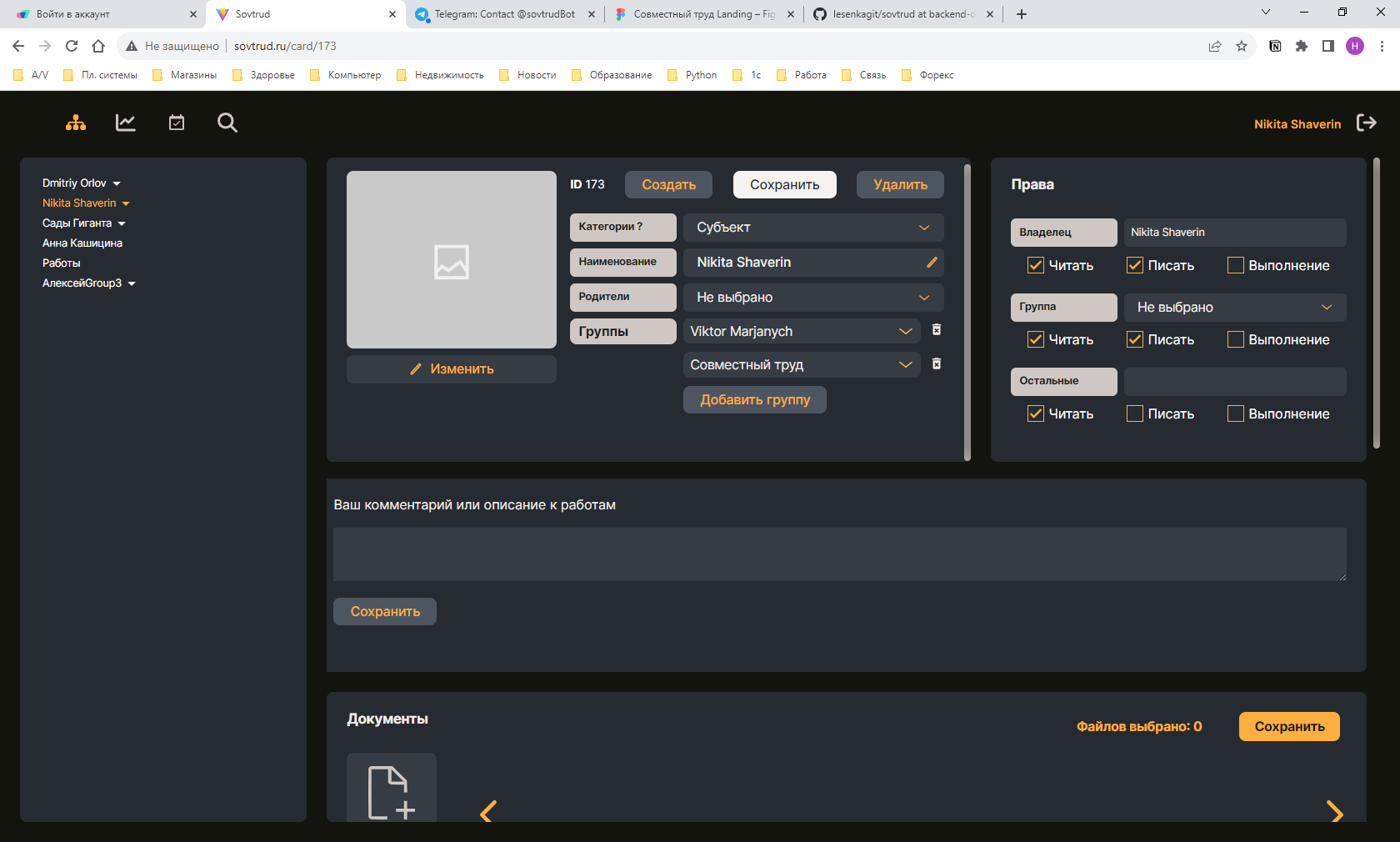
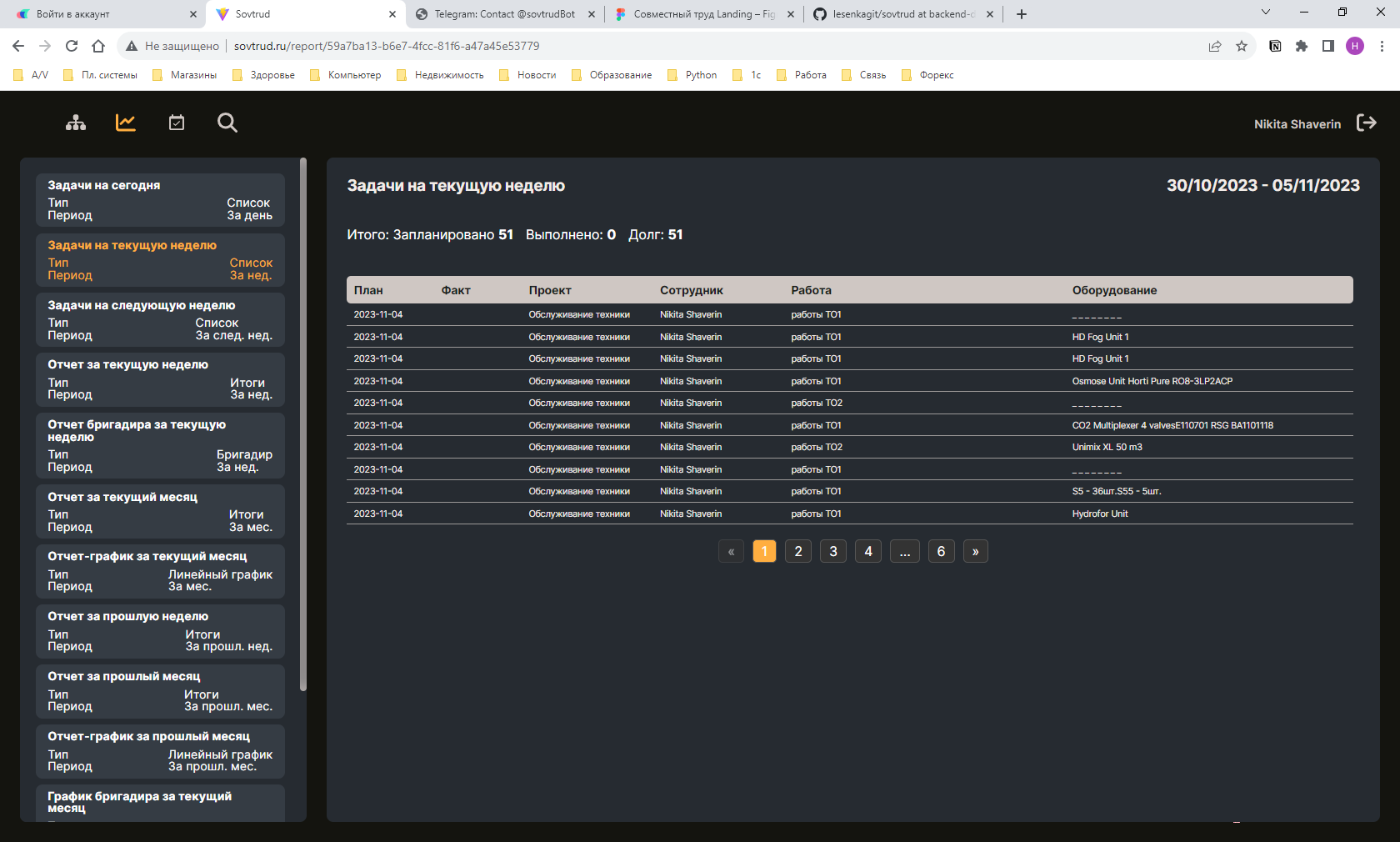


Рисунок П1. Веб-версия ИС. Страница пользователя.

Рисунок П2. Веб-версия ИС. Отчёт типа list.

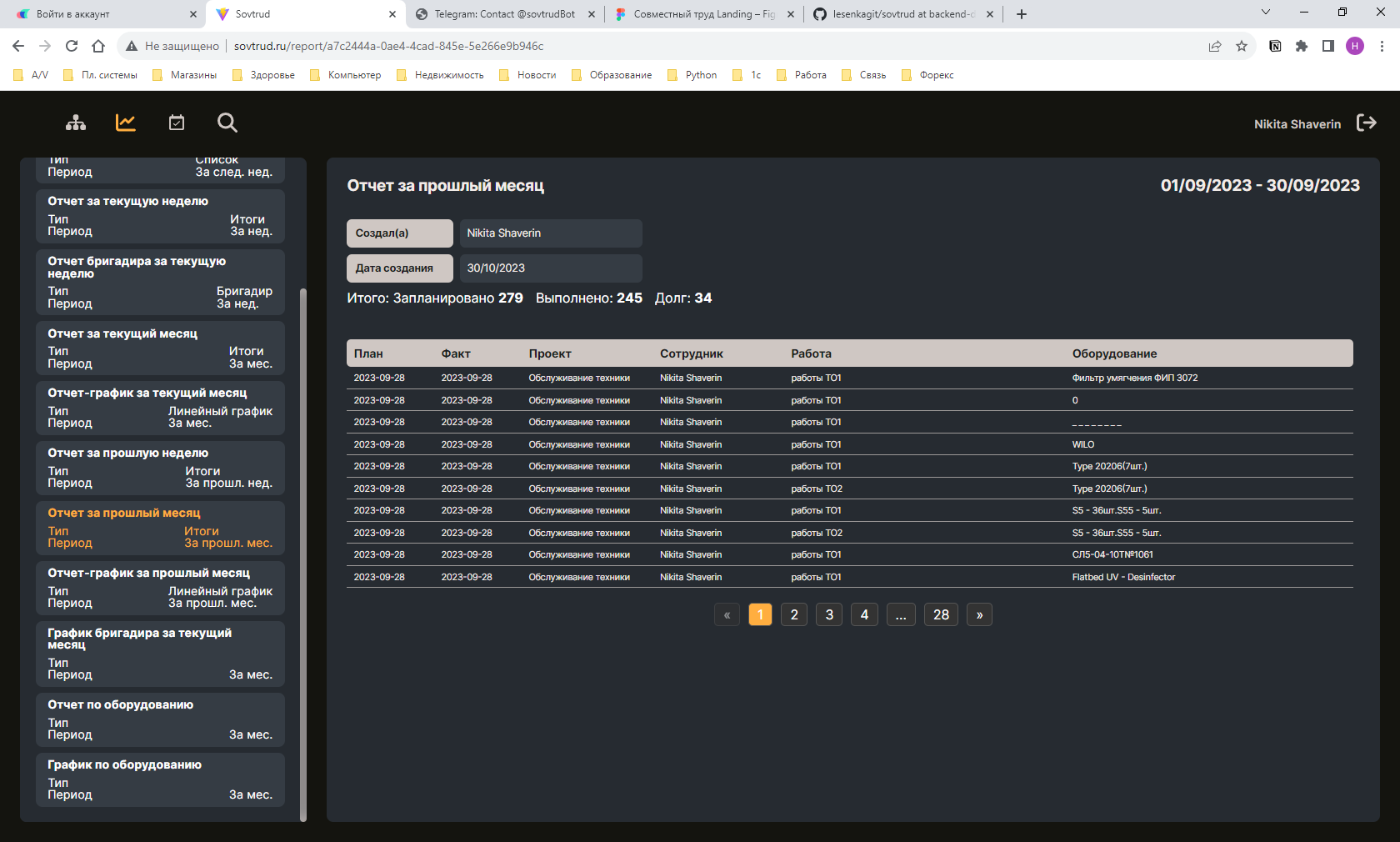


Рисунок П3. Веб-версия ИС. Отчёт типа status.



Рисунок П4. Веб-версия ИС. Отчёт типа line.

6.3. Благодарственное письмо



6.4. Рекомендательное письмо

