Содержание

Введение

В наше временя оборот информации в бизнес сфере огромен, отцифрованно практически всё. Большая часть информации передаётся по средствам компьютерных сетей, в частности глобальнай - Интернет.

Технология Интернет приспособленна для передачи любого вида закодированной информации, чем пользуется в преимуществе большая часть бизнес отраслей для получения своей прибыли. Но во время развития Интернет появилась такая ниша как Взломщики, люди посягающии на не санкционированное получения доступа к ресурсам подключенных к Интернет. Вследствии начали стремительно развиваться технологии Криптографии , которые существующие еще со времён древнего Рима, и других методов борьбы с Взломщиками и не только.

Взломищакам могут быть интересны любые ресурсы: журналы Бухгалтерии, системы управления предприятеим и т.д.

Подходя к теме о ведении бизнеса в сфере ЧОП многие подозрительные личности могут быть заинтересованн в получении запланированных маршрутов инкосации, адресса

жительства сотрудников… Для пресечения перечисленных выше махинаций можно полностью отказаться от ведения своей деятельности в Интернет.

Если это не выход - тогда необходимо развертывание системы защиты, чем мы и займемся в пределах данной работы. Также самым современным решением будет использование Blockchain - децентрализованный метод хранения данных, но данный вариант не будет рассматриваться, так как всё сильно усложняет и, скорее всего, по просту не приемлем.

Решение будет выполненно в несколько шагов:

1. Анализ структуры ЧОП.
2. Анализ проходимых процессов в ЧОП.
3. Описание средств разработки.
4. Составление базы данных ЧОП.
5. Разработка архитектуры приложения для взаимодействия с ЧОП.
6. Реализация архитектурных решений
7. Тестирование.

Теоритическая часть

Организация ЧОП

Охранное предприятие занимается охраной какой либо частной или государственной собственности.

ЧОП, чаще всего состоит из трех подразделений:

- Бухгалтерия
- Дирекция(она же администрация)
- Отдел кадров
- Отдел охраны
- Отдел инкосации
- Отдел вооружения

Если говорить о предоствляемых услугах более конкретно, то ЧОП для потребителя предлогает:

- Охрана объекта
- Охрана ценных бумаг или металов(инкосация)

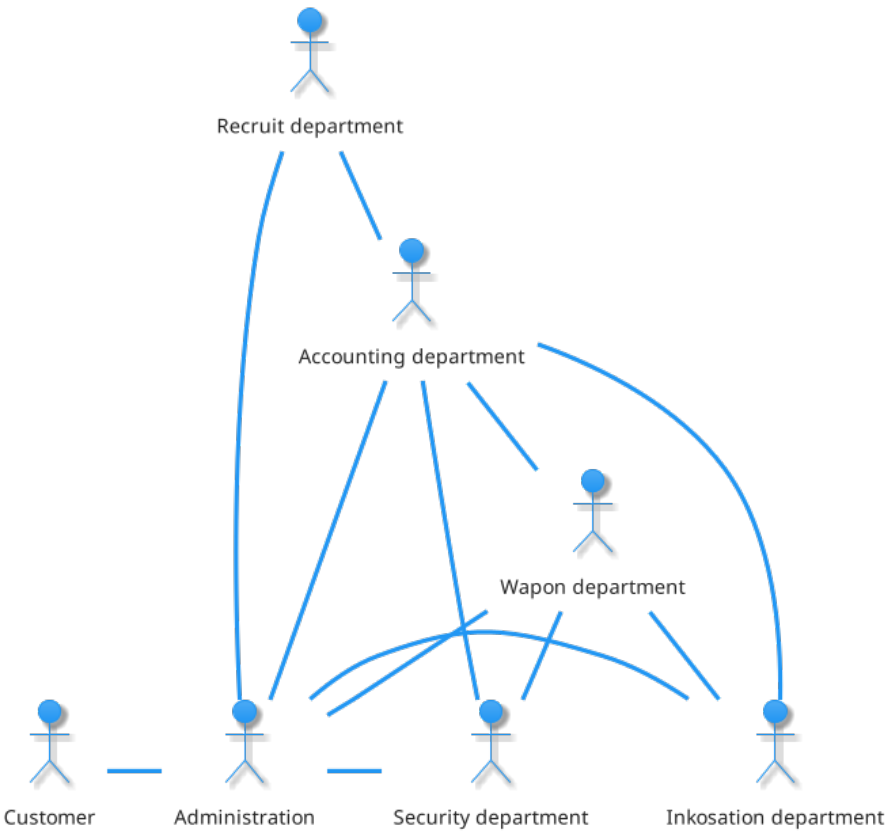


Figure 1: PSC structure

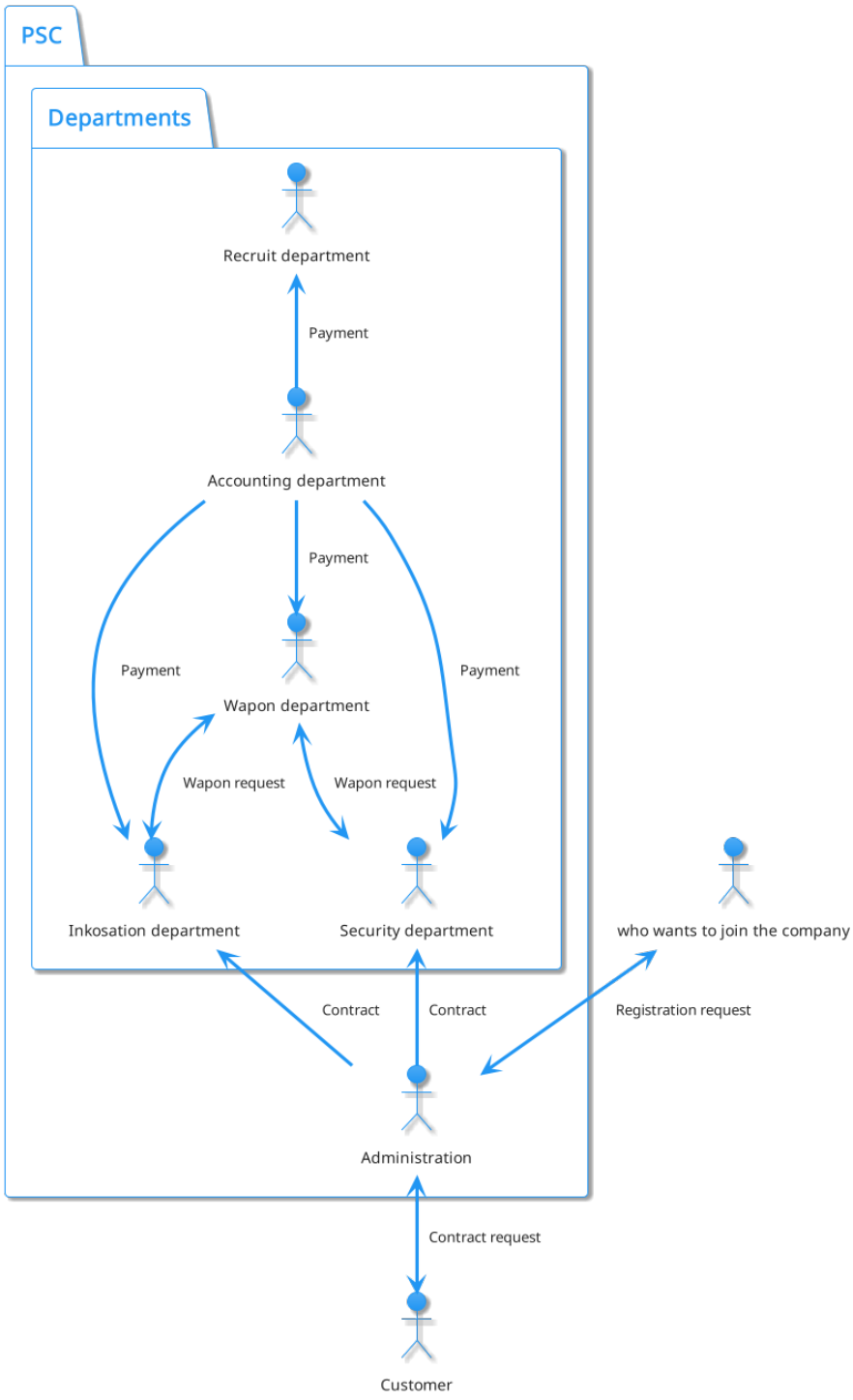


Figure 2: inside info flows

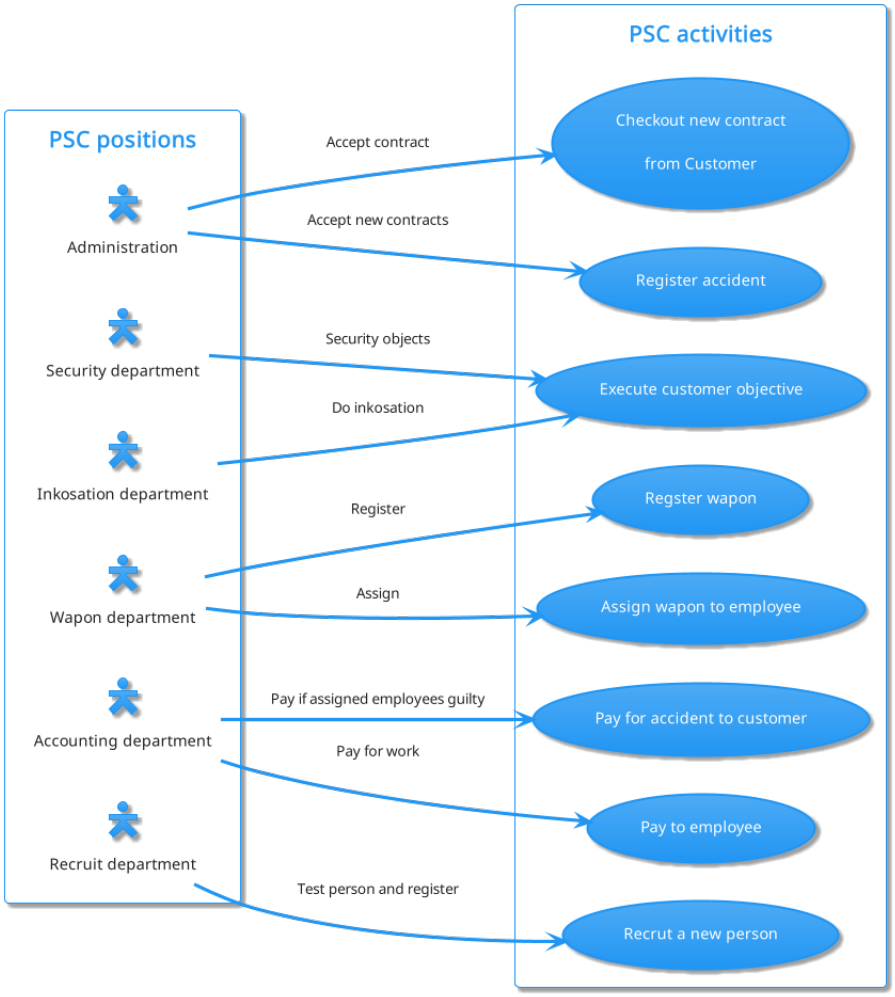


Figure 3: PSC Activities

Информационные потоки ЧОП

Процессы внутри ЧОП

Описание процессов

Составление контракта

Прием на работу

Регистрация и выдача оружия

Выплаты Выплаты работникам:

Выплаты за нанесенный ущерб объекту охраны при не нулевом проценте вины сотрудника:

Описание средстд разработки

В данном разделе будут рассмотрены средства разработки, используемые мной при создании ПС для ЧОП.

Утилиты для разработки баз данных

SQLite v3.35.5 stable

SQLite logo SQLite — компактная встраиваемая СУБД. Исходный код библиотеки передан в общественное достояние. Данная СУБД работает в безсерверной конфигурации. Если сравнивать с другими СУБД, то в равных условиях запись SQLite осуществляет медленее на 20-30% чем другие СУБД, но чтение превосходит другие на 40-50%. SQLite не имеет привелегий, только систему авторизации, но это и не нужно в моем проекте, об этом будет сказано позже. > Примичание: > Библиотека SQLite не будет использована в чистом виде, а в составе Qt v6.2.4

SQLiteBrowser v3.12.2

SQLite Browser Удобный FOSS браузер баз данных SQLite, использованный для отладки.

DBVisualizer v12.1.8

dbvis Проприетарная утилита для работы с разными СУБД, использован для генерации графа таблиц составленной базы данных ЧОП.

Библиотеки

Qt v6.2.4

Qt logo Qt - один из самых популярных и больших фреймворков c++ на рынке. Важная характеристика Qt - переносимость, т.к. я работаю на Linux.

Системы сборки и компиляторы

CMake v3.22.3

CMake logo Система сборки c++. Сборка проекта и передача более низкоуровневому средству.

Ninja c1.10.2

Еще одна система сборки, только уже более низкого уровня, чем CMake. Передача исходного кода на компиляцию.

clang v13.0.1

Компилятор семейства C. Без коментариев.

IDE

QtCreator v6.0.2

QtCreator logo IDE от компинии The Qt Company, использованный только как средство отладки и создания скелетов форм пользовательского интерфейса.

VIM v8.2

VIM logo Моя любимая IDE, в своей основе так же прост как и каноничный “блокнот” в Windows, только с максимальной степенью кастомизации. Главное достоинство, по моему мнению, - это управление без использования мыши и возможность настройки управляющих комбинаций максимально

удобно, что сокращает время на бесполезное перемещение рук
по рабочему пространству.

Плагины

```
    Plug 'https://github.com/xolox/vim-misc'                    " auto load
    Plug 'https://github.com/xolox/vim-session'                 " session manager
    Plug 'wakatime/vim-wakatime'                                " wakatime.com
    Plug 'SirVer/ultisnips'                                     " snippets
    Plug 'https://github.com/honza/vim-snippets'                " snippents files
    Plug 'https://github.com/pangloss/vim-javascript.git'       " javascript extension
    Plug 'vim-airline/vim-airline'                              " status line
    Plug 'vim-airline/vim-airline-themes'                       " themes
    Plug 'vim-scripts/AfterColors.vim'                          " themes
    Plug 'rafi/awesome-vim-colorschemes'                        " themes
    Plug 'sonph/onehalf', { 'rtp': 'vim' }                      " theme
    Plug 'https://github.com/sjl/badwolf'                       " theme
    Plug 'https://github.com/joshdick/onedark.vim'              " theme
    Plug 'https://github.com/plasticboy/vim-markdown'           " markdown format support
    Plug 'ryanoasis/vim-devicons'                               " icons support
    Plug 'https://github.com/Yggdroot/indentLine'
    Plug 'ycm-core/YouCompleteMe'                               " code completer
    Plug 'rdnetto/YCM-Generator', { 'branch': 'stable'}
    Plug 'tpope/vim-commentary'                                 " commentary shortcuts
    Plug 'm-pilia/vim-pkgbuild'                                 " archlinux AUR PKGBUILD file
    Plug 'https://github.com/tpope/vim-surround'                " html-like tags handle utils
    Plug 'https://github.com/octol/vim-cpp-enhanced-highlight' " cpp syntax hilightinght
    Plug 'ctrlpvim/ctrlp.vim'                                   " file finder
    Plug 'preservim/nerdtree'                                   " dir tree dock
    Plug 'https://github.com/preservim/tagbar'
    Plug 'vim-scripts/bufkill.vim'                              " exit buffers without exitin
    Plug 'jreybert/vimagit'                                     " git support
    Plug 'https://github.com/ap/vim-css-color'                  " HEX-colors hilighting
    Plug 'https://github.com/matze/vim-move'                    " code moving
    Plug 'https://github.com/junegunn/vim-easy-align'           " fast text aligning
    Plug 'https://github.com/ervandew/supertab'
    Plug 'https://github.com/jiangmiao/auto-pairs'              " completing pairs
    Plug 'https://github.com/rhysd/vim-clang-format'            " auto formating
    Plug 'honza/vim-snippets'                                   " set of snippets
    Plug 'https://github.com/godlygeek/tabular'                 " tab extender
    Plug 'https://github.com/junegunn/vim-easy-align'
    Plug 'https://github.com/fadein/vim-FIGlet'  " figlet
    Plug 'https://github.com/scrooloose/syntastic'
```

Разное

Plantuml v1.2021.16

plantuml Средство создания UML диаграм. Использовано для визуализации объектов и просецссов.

BASH v5.1.16

bash Bourne Again Shell - интерпритатор, использован для автоматизации некоторых процессов. > Примечание: > Оглавления данной работы было автоматически сгенерерованно данным bash-скриптом:

```
cat "$1" > "$1".indexed

>index

i=(-1 1)
prevLen=0
while read -r line; do
    hash="$(md5sum <<< "$line" | cut -d ' ' -f 1)"
    printf "<a id=\"%s\"></a>\n%s\n" "$hash" "$line" > tmp
    sed "/$line/ {
        x
        r /home/xewii/Documents/TIT/ZXC/tmp
    }" "$1".indexed > "$1".indexed.tmp
    mv "$1".indexed.tmp "$1".indexed
    hdrLen=$(awk -F'#' '{print NF-1}' <<< "$line")
    hdrTxt=$(echo "${line//#/}")
    (( $hdrLen > 1 )) && for (( j=1; j<$hdrLen*4; j++ )); do printf ' '; done
    (( $prevLen < $hdrLen )) && i[$hdrLen]=1
    printf "%d.[%s](#%s)\n" ${i[$hdrLen]} "$hdrTxt" "$hash"
    prevLen=$hdrLen
    let i[$hdrLen]++
done <<< "$(grep --color=no -E "^#+" "$1")" > index

mv "$1".indexed tmp
printf "#       \n" > "$1".indexed
cat index >> "$1".indexed
cat tmp >> "$1".indexed

rm tmp
rm index
```

Отладка

Valgrind v3.18

Утилита профилирования и отладки программы, использовано в основном для обнаружения утечек памяти.

VCS

GIT v2.35.1

GIT - система контроля версий, сомо о себе говрит. Использовался в основном для перенесения кода между машинами и как средство дистрибъюции.

Разработы базы данных ЧОП

База данных ЧОП в пределах данной работы - головная сущность, вокрук которой будет строится весь функционал. База данных будет существовать под управлением SQLite.

Таблицы

В компанию, как известно, входит некоторое количество сотрудников, по этому, я создаю таблицу Users. Название выбрано таковым, потому что она будет содержать данные учетных записей сотрудников и клиентов ЧОП.

```
CREATE TABLE
    "Users"("id" INTEGER NOT NULL UNIQUE,
            "name" TEXT NOT NULL,
            "entryDate" TEXT NOT NULL,
            "role_id" INTEGER NOT NULL,
            "wapon_id" INTEGER,
            "email" TEXT UNIQUE,
            "login" TEXT NOT NULL UNIQUE,
            "password" BLOB NOT NULL UNIQUE,
            "salt" BLOB NOT NULL,
            "image" BLOB,
            FOREIGN KEY("wapon_id") REFERENCES "Wapons"("id")ON DELETE RESTRICT,
            FOREIGN KEY("role_id") REFERENCES "Roles"("id")ON DELETE RESTRICT,
            PRIMARY KEY("id" AUTOINCREMENT))
```

Таблица содержит данные для идентификации:

- login

- password
- salt

Пароль не храниться в открытов виде, а зашифорован с использованием динамической соли по алгоритму “Prefered salt algorithm”, более подробно будет расмотрен далее.

Как видно, таблица Users зависит от таблиц Roles и Wapons, собственно вот они:

```
CREATE TABLE
    "Roles"("id" INTEGER NOT NULL UNIQUE,
            "name" TEXT NOT NULL UNIQUE,
            "commands_id" INTEGER NOT NULL,
            "payMultipler" DECIMAL(10, 3) NOT NULL,
            "payPeriod" INTEGER NOT NULL,
            FOREIGN KEY("commands_id") REFERENCES
            "roleCommands"("role_id")ON DELETE RESTRICT,
            PRIMARY KEY("id"))
```

Роль определяет какие данные и соответственно команды можешь выполнять на сервере. Ссылается на таблицу roleCo-mands - это SQL массив с ID команд, которые может выполнять пользователь с данной ролью, по поэтому я и отказался от других, более тяжелых СУБД, т.к. все необходимые действия делегируются на Сервер, что будет рассмотрено далее, от СУБД требуется только хранить данные и извлекать их. Связаная таблица roleCommands:

```
CREATE TABLE
    "roleCommands"("role_id" INTEGER NOT NULL,
                   "command_id" INTEGER NOT NULL,
                   FOREIGN KEY("role_id") REFERENCES
                   "Roles"("id")ON DELETE RESTRICT)
```

Таблица Wapons:

```
CREATE TABLE
    "Wapons"("id" INTEGER NOT NULL UNIQUE,
             "employee_id" INTEGER UNIQUE,
             "name" TEXT NOT NULL,
             "ammo" INTEGER NOT NULL,
             "price" DECIMAL(10, 3) NOT NULL,
             "ammoPrice" DECIMAL(10, 3) NOT NULL,
             "image" BLOB,
             FOREIGN KEY("employee_id") REFERENCES
             "Users"("id")ON DELETE RESTRICT,
             PRIMARY KEY("id"))
```

Каждая запись в таблице Wapons - это еденица зарегестрированного

оружия, в полной мере описывающяя необходимые характеристики для ЧОП.

Так как организация имеет свои расходы и доходы, нам нужно сохранять эти данные. Таблица Accounting:

```
CREATE TABLE
    "Accounting" ("id" INTEGER NOT NULL UNIQUE,
                  "accountingType_id" INTEGER NOT NULL,
                  "pay" DECIMAL(10, 3) NOT NULL,
                  "date" TEXT NOT NULL,
                  FOREIGN KEY("accountingType_id")
                  REFERENCES "AccountingType"("id") ON DELETE RESTRICT,
                  PRIMARY KEY("id") )
```

Зависит от таблицы AccountingType, описывающей какого рода транзакция была совершена.

```
CREATE TABLE
    "AccountingType" ("id" INTEGER NOT NULL UNIQUE,
                      "name" TEXT NOT NULL UNIQUE,
                      PRIMARY KEY("id","name") )
```

ЧОП получает доход от контрактов, по этому была составлена таблица Contracts:

```
CREATE TABLE
    "Contracts"("id" INTEGER NOT NULL UNIQUE,
                "assignedEmployees_id" INTEGER NOT NULL,
                "customer_id" INTEGER NOT NULL,
                "objectType_id" INTEGER NOT NULL,
                "objectAddress" TEXT NOT NULL,
                "objectWayPoint" TEXT, "date" TEXT NOT NULL,
                "expirationDate" TEXT NOT NULL,
                "weekends" TEXT NOT NULL,
                FOREIGN KEY("customer_id") REFERENCES
                "Users"("id")ON DELETE RESTRICT,
                PRIMARY KEY("id"),
                FOREIGN KEY("assignedEmployees_id") REFERENCES
                "AssignedEmployees"("employee_id") ON DELETE RESTRICT,
                FOREIGN KEY("objectType_id") REFERENCES
                "objectType"("id")ON DELETE RESTRICT)
```

Запись в таблице Contracts это сделка вида, описанного в связаной таблице objetType. Таблица objetType:

```
CREATE TABLE
    "objectType" ( "id" INTEGER NOT NULL UNIQUE,
                   "name" TEXT NOT NULL UNIQUE,
```

```
                    "price" DECIMAL(10, 3) NOT NULL,
                    PRIMARY KEY("id") )
```

Запись в данной таблице описывает объект контракта, где указывается базавая цена за период оплаты(payPeriod) исполнителя/исполнителей контракта(assignedEmployees_id). Для привязки нескольких сотрудников, была создана еще одна таблица AssignedEmployees, являющейся массивом.

```
CREATE TABLE
    "AssignedEmployees" ( "id" INTEGER NOT NULL,
                          "employee_id" INTEGER NOT NULL,
                          "guiltyPercent" DECIMAL(10, 3) NOT NULL,
                          "usedAmmo" INTEGER,
                          FOREIGN KEY("employee_id") REFERENCES
                          "Users"("id") ON DELETE RESTRICT,
                          PRIMARY KEY("id") )
```

Для создания контракта в данной таблице нужно только 2 поля - id, employee_id. Так так во время исполнения может произойти какой то ицедент, то была создана таблица Accidents:

```
CREATE TABLE
    "Accidents" ("id" INTEGER NOT NULL UNIQUE,
                 "name" TEXT NOT NULL,
                 "contract_id"
                 INTEGER NOT NULL,
                 "usedAmmoCount" INTEGER,
                 "damagePrice" DECIMAL(10, 3),
                 "assignedEmployees_id" INTEGER,
                 FOREIGN KEY("contract_id") REFERENCES "Contracts"("id") ON DELETE RESTRICT,
                 FOREIGN KEY("assignedEmployees_id") REFERENCES "AssignedEmployees"("id") ON
                 PRIMARY KEY("id") )
```

Описывает проишествия, произошедшие во время исполнения контракта. Если у нас есть контракты, описывающие некую деятельность с учетом выходных и рамок начала и окончания службы, то можно было бы ускорить вычисление рабочего времени по дням с помощью препроцессинга данных из записи Contracts. Таблицей, в которую сохраняются транслированные данные является - DutySchedule:

```
CREATE TABLE
    "DutySchedule" ( "emploee_id" INTEGER NOT NULL,
                     "day" INTEGER NOT NULL
                     FOREIGN KEY("emploee_id") REFERENCES "Users"("id") ON DELETE RESTRICT )
```

day - это 64х битная цыфра со знаком в формате UNIX

time(secs since epoch).

Визуализация базы данных

Datbase visualization

Разработка архитектуры приложения

Приложение для взаимодействия с моделью ЧОП, построенной ранее, должно предоставлять функционал для реализации всех описанных процессов взаимодействия с моделью ЧОП.

Основа

Основной функцией приложения, как понятно из темы курсовой работы, будет обеспечение безопасности данных. Исходя из этого в голову приходит идея организовать клиент-серверную архитектуру приложения, но это не главная причина почему выбрана такая архитектура. Основная причина - необходимость принимать заказы от клиентов, предоставить им функционал для удобного взаимодействия с персоналом ЧОП, но по большей части он будет взаимодействовать с клавиатурой. И для персонала ЧОП тоже будет намного удобнее и быстрее использовать унифицированные методы итерации с базой данных и самой организацией.

Информационные потоки ЧОП

Информационные потоки ЧОП

Обращение к базе данных

Как было сказано ранее, СУБД не будет управлять системой привелегий, этим будет заниматься другой код. Система, которую я разработал основана на ролях, так же как и в обычных “умных” СУБД, к которым привязано некоторое количество возможных к исполнению команд.

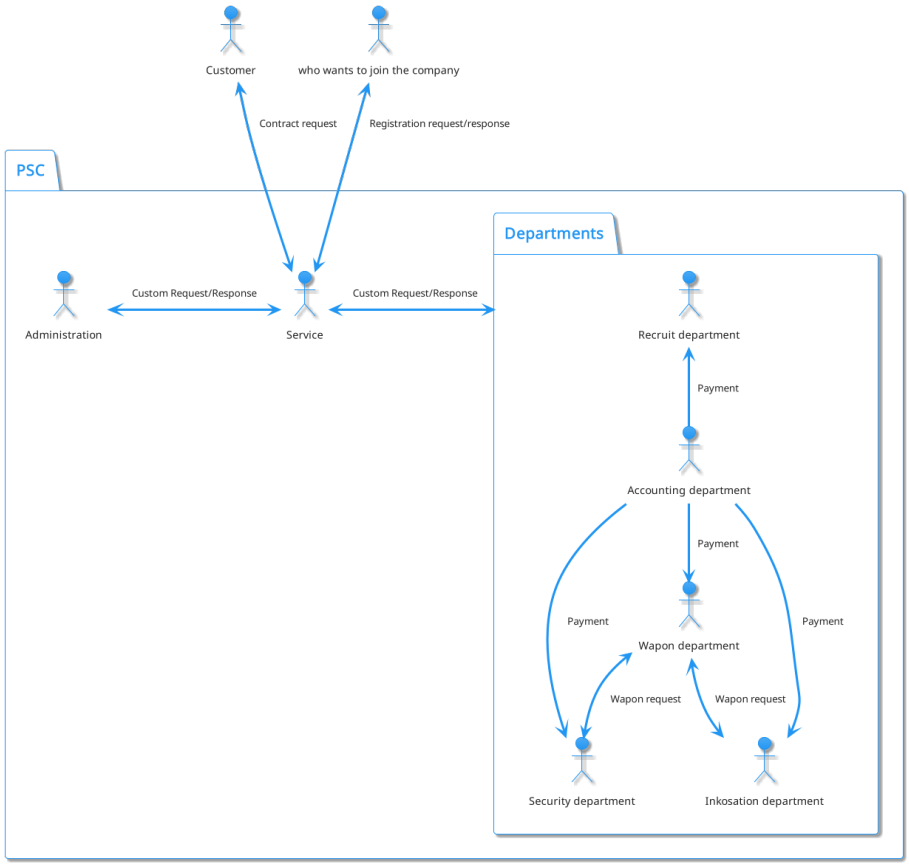
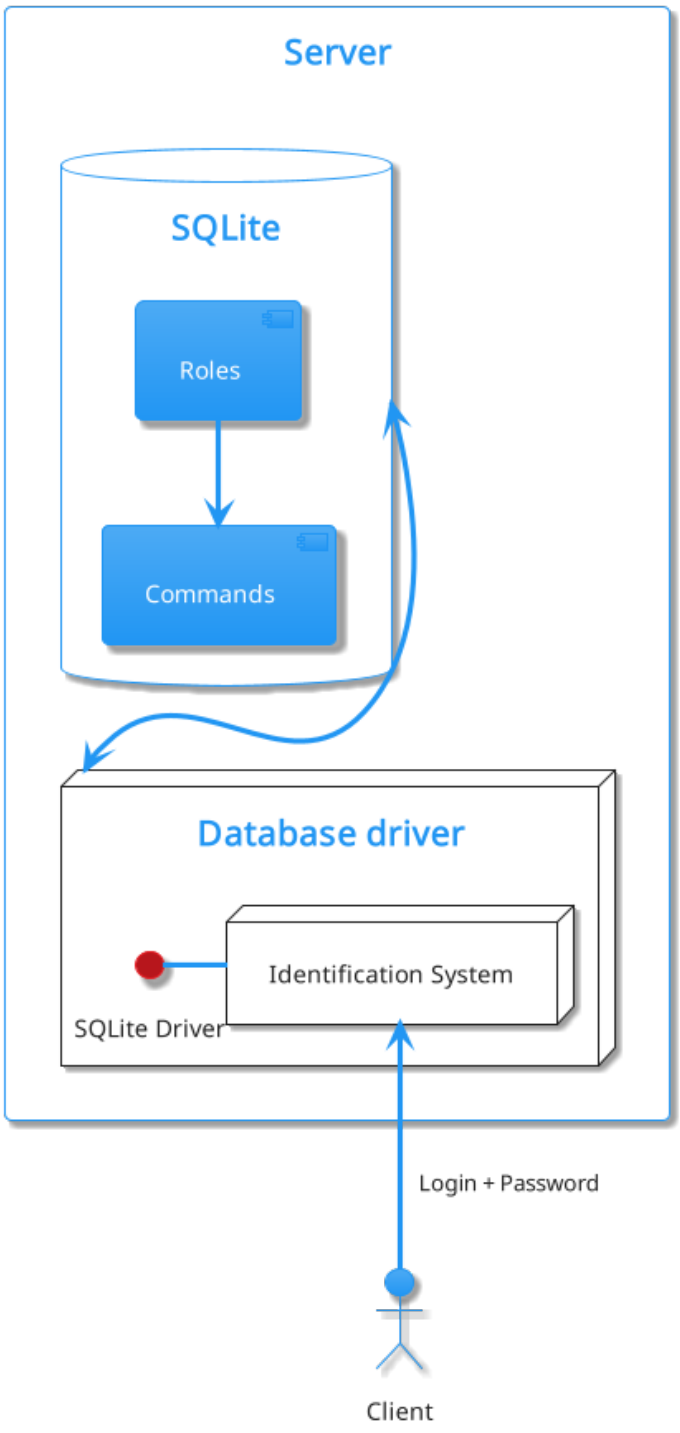


Figure 4: PSC with server info flows


Server
SQLite
Roles
Commands
Database driver
Identification System
SQLite Driver
Login + Password
Client

Система безопасности

Для обеспечения более высокого уровня защиты команды и роли будут вшиты в программу. При каждом старте будет проверятся валидность первоначальных данных и имеющихся на данный момент в базе данных. Также, как мера безопасности к ПС будут предложены хеш суммы исполняемых файлов.

Система идентификации и авторизации При регистрации нового пользователя создается новая запись в таблице Users, все поля, кроме поля password сохраняются в неизменнов виде. password сохраняется в захешированном виде при использовании хэш функции sha512 с использование динамической соли.

Алгоритм создания соли:

1. Подсчет количества символов разных типов в пароле.
2. Выявление слабых сторон пароля.
3. Формирование алфавита для создания соли.
4. С помощью “strog random” функции выборка символов из составленого алфавита.
5. Запись результата.

Алгоритм хэширования пароля:

1. Генерация соли.
2. Взятие первичного хеша с пароля.
3. Итерационно выбрать по байтно методом XOR младших битов первичного хеша и переданного пароля места вставки соли.
4. Хеширование получившейся строки.
5. Запись результата.

Данный алгорит и выбраная хэш функция ограничивают максимальную длину пароля до 64 символов. Длина соли была вабрана 32 байтная. Соль сохраняется вместе с паролем, чтобы обеспечить возможность идентификации.

Для идентификации проводится сравнивание значение пароля из базы данных с переданным в функцию шифрования с солью данной записи пользователя пароля.

Система команд Проанализировав информационные потоки и требования ЧОП я составил 22 команды, в исходном коде объявление команд выглядит вот так:

```
// id, name, executor
#define COMMANDS_MAP(XX) \
```

```
XX( 0,  MAKE_CONTRACT,         exec_make_contract      ) \
XX( 1,  MAKE_DUTY_SCHEDULE,    exec_make_duty_schedule ) \
XX( 2,  REGISTER_ACCIDENT,     exec_register_accident  ) \
XX( 3,  REGISTER_EMPLOYEE,     exec_register_employee  ) \
XX( 4,  REGISTER_CUSTOMER,     exec_register_customer  ) \
XX( 5,  REGISTER_OBJECT_TYPE, exec_register_object_type) \
XX( 6,  REGISTER_WAPON,        exec_register_wapon     ) \
XX( 7,  ASSIGN_WAPON,          exec_assign_wapon       ) \
XX( 8,  PAY_AMMO,              exec_pay_ammo           ) \
XX( 9,  PAY_EMPLOYEE,          exec_pay_employee       ) \
XX( 10, PAY_ACCIDENT,          exec_pay_accident       ) \
XX( 11, EDIT_OBJECT_TYPE,      exec_edit_object_type   ) \
XX( 12, UPDATE_ROLE,           exec_update_role        ) \
XX( 13, GET_USER_INFO,         exec_get_user_info      ) \
XX( 14, GET_ACCIDENT_DETAILS, exec_get_accident_details) \
XX( 15, GET_ACCOUNTING_ENTRY, exec_get_accounting_entry) \
XX( 16, GET_OBJECT_DETAILS,   exec_get_object_detalils) \
XX( 17, GET_ROLE_DETAILS,      exec_get_role_details   ) \
XX( 18, GET_WAPON_DETAILS,     exec_get_wapon_details  ) \
XX( 19, GET_DUTY_SCHEDULE,     exec_get_duty_schedule  ) \
XX( 20, CREATE_TABLE,          exec_create_table       ) \
XX( 21, IDENTIFY,              exec_identify           ) \
```

Архитектура Клиент-Сервер

Архитектура Клиен-Сервер была выбрана не только для обеспечения безопасности базы данных, но и для реализвации удаленного унифицированного доступа к услугам и возможностям ЧОП. Так для сотрудников упрощается способ комуникации с начальством, подчиненными. А заказчики смогут после регистрации в системе могут удаленно составить контракт по охране.

Протокол

Общение клиентов с сервером будет осуществяться по протоколу TCP опционально с использованием SSL/TLS(опциональность введена для более удобного тестирования).

Формат сообщений Поверх формата TCP пакета я ввел для ПС дополнительные поля.

Состоит из заголовка и нагрузки.

Заголовок Длина заголовка - 176 бит или 22 байта. Кончно, можно было бы спокойно использовать все 192 бита(3

машинных слова), чтобы выравнять заголовок, но это не сильно повлияет на какой либо процесс. Поля:

(занимаемые байты в сообщении) Имя: Описание

- (0x0 - 0x3) Size: суммарный размер сообщения в байтах
- (0x4 - 0x11) ServerStamp: Время отправки сообщения сервером
- (0x12 - 0x19) ClientStamp: Время отправки сообщения клиентом
- (0x20 - 0x21) PacketType: Тип сообщения
- (0x22 - 0x23) PacketLoadType: Тип формата нагрузки сообщения

Таким образом, поле Size позволяет считывать последовательность байт в единое сообщение из нескольких пакетов TCP. Поля xxxStamp используются как некий идентификатор сообщения. Тип пакета и формат нагрузки определяют собственно тип нагрузки и сообщения в целом.

Модель сервера

Основные задачи сервера:

- обслуживании базы данных
- реализации механизма идентификации и авторизации, также регистрации
- принятие входящих запросов на подключение
- работа в режиме Запрос => Ответ
- Мониторин протекающих процессов

Также, развертывание сервера будет означать создание первого Amin пользователя, для управления организационными процессами ЧОП. Следовательно у сервера должен быть интерфейс для создания пользователей, как и у клиента.

Модель клиента

Клиент - это программа, предоставляющая некоторый функционал, в который в любом случае входит пользовательский интерфейс, после успешного входа в систему, функционал будет разниться в зависимости от роли вошедшего в систему.

Все что требуется от клиента - это показывать пользовательский интерфейс, предоставить некоторый метод для входа в систему сервера и отправлять команды серверу, и ждать ответа. В частности, когда пользватель, который воспользовался

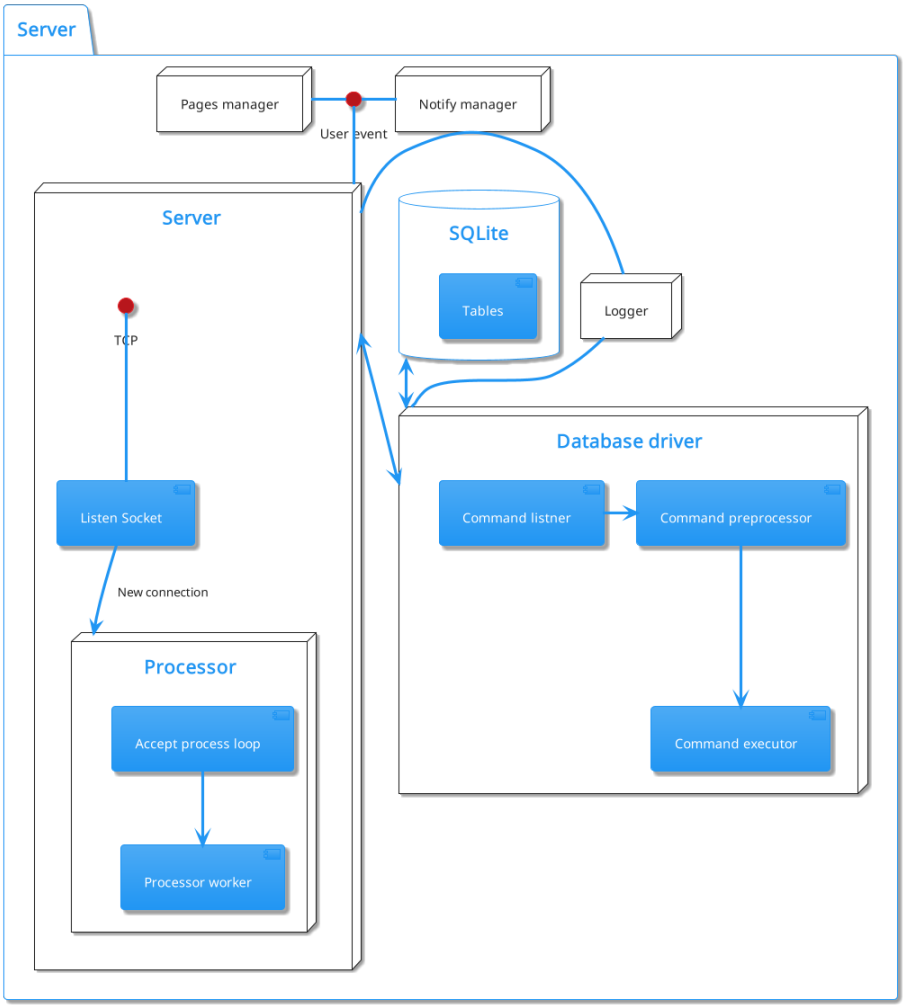


Figure 5: Server abstract model

клиентом является заказчик - клиент должен предоставить метод регистрации в систему.

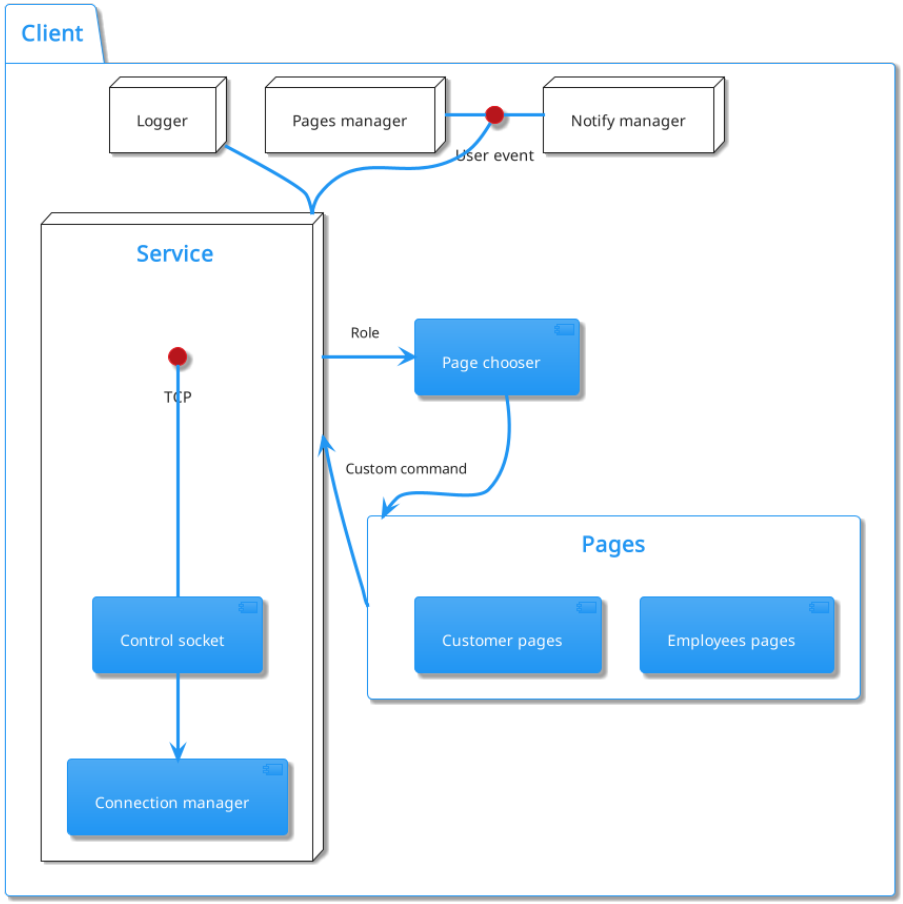


Figure 6: Client abstract mode

Реализация

Для реализации проекта был выбран язык C++ и фреймвокр Qt6. Выбор лег в сторону C++ только потому что это мной наиболее изученный язык, на много проще было бо всё реализовать на nodejs с express, а Qt - это лучшый кросплатформенный фреймворк лично для меня, он упростил разработку процентов на 60% и сократил зависимости проекта на 100%(используются только библиотеки из пакета Qt). Сборкой проекта будет

заниматься CMake с делегацией компиляции Ninja, C++ компилятор - clang. Для отладки буду использовать GDB в составе QtCreator и Valgrind. Создания шаблонов форм так же будет осуществляться в QtCreator в модуле QtDisigner.

Пользовательский интерфейс

За отрисовку GUI будет отвечать модуль Qt - QtGUI.

Пользовательский интерфейс будут составлять 5 основных сущностей:

- Бар состояния
- Меню бар
- Тул бар
- Пейджер(PagesManager)
- Менеджер уведомлений(NotifyManager)

PagesManager

Данный класс отвечает за агрегацию “страниц” GUI и их переключение. > Страница - это отдельный виджет(QWidget) или объект-наследник. Сам класс PagesManager - это тоже виджет, наследуемый от QFrame(он тоже наследник QWidget).

Содержит в себе именованный массив страниц:

```
struct Page {
    QWidget * widget = nullptr;
    int navId = -1;
    QVector<QString> edges = {};
};

QMap<QString, Page> _pages;
```

Как видно из данного участка кода, одна страница может быть связана с другими по имени(можно было бы связывать напрямую с другим объектом Page, но выбранный мной подход более прост в реализации). Страница добавляется в общий массив методом:

```
void
PagesManager::addPage(const QString& name, QWidget* wp,
                      const QVector<QString>& edges)
{
    if (!wp) {
        throw QString("empty widgt passed");
    }
```

```
    wp->setObjectName(name);
    _pages[name] = {wp, -1, edges};
    _view->addWidget(wp);
}
```

view - это QStacketWidget - место размещения страниц и является основной сущностью пейджера. Также, класс содержит перегруженный метод добавления корневой страницы. Она необходима для построения путей к страницы.

За построение пути отвечает агрегируемый класс PagePath-Frame, наследуемый от QFrame, является второй основной сущность пейджера. Основным методом класса является методод:

```
void
PagePathFrame::changePath(const QVector<QString>& path)
{
    reset();
    for (auto node : path) {
        QWidget * lbl = new QLabel(_delemiter, this);
        lbl->setFont(QFont("Jet Brains Mono", 9));
        lbl->setSizePolicy(QSizePolicy::Maximum, QSizePolicy::Maximum);
        _layout->addWidget(lbl);
        ClickableLabel * clbl = new ClickableLabel(node, this);
        clbl->setSizePolicy(QSizePolicy::Maximum, QSizePolicy::Maximum);
        clbl->setCursor(QCursor(Qt::CursorShape::PointingHandCursor));
        clbl->setStyleSheet("color: #b78620");
        connect(clbl, &ClickableLabel::clicked, [this, node] { Q_EMIT clicked(node);  });
        _layout->addWidget(clbl);
    }
    adjustSize();
}
```

Путь к страницы вычисляется в методе пейджера:

```
QVector<QString>
PagesManager::pagePath(const QString& page)
{
    QVector<QString> path { page  };
    QString search = page;
    bool done = false;
    int tries = 0;

    while (!done && tries < _pages.size() + 1) {
        for (auto i : _pages) {
            if (search == _root) {
```

```
                done = true;
                break;
            }
            for (auto node : i.edges) {
                if (node == search) {
                    search = i.widget->objectName();
                    path.push_front(search);
                    // exit outer?
                    break;
                }
            }
        }
        tries++;
    }
    return path;
}
```

Не самый эффективный метод, можно было бы хранить сразу все возможные пути в массиве. Метод основан на поиск в глубину в графе, если я не ошибаюсь.

Как видно, каждый элемент фрейма, при нажатии генерирует сигнал о нажатии, для отправки PagesManager. При получении пейджер меняет страницу.

Третьей сущностью явялется навигационная панель - NavWidget. Содержит связаные с данной страницей ссылки в виде кнопок. Гланый метод добавления навигационного меню:

```
int
NavWidget::addNav(const QVector<QString>& pages, bool createBack)
{
    NavFrame * fnav = new NavFrame(pages, createBack, this);
    this->addWidget(fnav);
    connect(fnav, SIGNAL(clicked(QString)), this, SIGNAL(clicked(QString)));
    adjustSize();
    return this->count()-1;
}
```

Так же при нажатии меняет страницу.

Для того, чтобы связать страницы и навигационное меню используется рекурсивный метод:

```
void
PagesManager::bindPages(const QString& parent, const QVector<QString>& childs)
{
    int nid = _nav->addNav(childs, parent != _root);
    _pages[parent].navId = nid;
```

```
    for (auto child : childs) {
        if (_pages[child].edges.length() > 0) {
            bindPages(child, _pages[child].edges);
        } else {
            _pages[child].navId = nid;
        }
    }
}

void
PagesManager::finalize()
{
    if (_root == QString()) {
        throw "Cannot finalize PagesManager without root page";
    }
    bindPages(_root, _pages[_root].edges);
    changePage(_root);
}
```

NotifyManager

Сущность выполняющая динамическое позиционирование всплывающих уведомлений разного типа. В своей основе - это лейаут поверх всего рабочего пространства приложения и процессор, обрабатывающий запросы на создание новых уведомлений и управления существующими, уведомление - это разновидность класса NotifyItem. Главная причина почему этот класс существует - возможность создавать thread-safe уведомления из любого потока. Т. к. любой виджет вне потока существования виджета-родителя не будет напрямую связан с его петлей событий. Как известно, объект начинает существовать там, где он был создан с помощью оператора new. Поэтому для передачи на обработку NotifyManager'у используюся фабрики NotifyItemFactory.

Таким образом, мне получилось создать простой интерфейс для создания всплывающих уведомлей:

```
void setItemPropery(int uid, const QByteArray& name, const QVariant& value);
void createNotifyItem(NotifyItemFactory*, int& uid);
```

Обращение к созданому уведомлению происходит по выделеному uid, который возвращает createNotifyItem.

Все существующие уведомления храняться в именованом массиве:

```
QMap<int, NotifyItem*> _items;
```

Логгер

iiNPack

Сервер

Драйвер базы данных

Криптография

Мнеденжер подключений

Менеджер сессий

ClientLink

Процессор подключения

Клиент

Service

Менеджер групп страниц

Заключение

Список литиратуры

1. https://wiki.qt.io - документация Qt
2. https://www.sqlite.org/ - документация SQLite