

**ЧАСТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
«ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»**

КУРСОВАЯ РАБОТА

ПМ.11 Разработка, администрирование и защита баз данных

МДК.11.1 Технология разработки и защиты баз данных

ТЕМА: Разработка многоуровневой защиты базы данных
охранного предприятия

Выполнил:

Студент Легенький Роман Михайлович
курс II группа 2 П 11

Проверила:

преподаватель Бутова Г.А.

Работа защищена: «___» _____ 2022 г.

Оценка «_____»
(прописью)

(Ф.И.О. преподавателя)

**г. Пятигорск
2022 г.**

**ЧАСТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
«ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»**

ЗАДАНИЕ
на курсовую работу

Студенту курса II группы 2 П 11 специальности 09.02.07

«Информационные системы и программирование»

Фамилия Легенькому

Имя Роману **Отчество** Михайловичу

Тема Многоуровневая защита базы данных охранного предприятия

Содержание пояснительной записки Описание организации ЧОП;
Разработка защищенной базы данных ЧОП; Планирование ПС для доступа к
базе данных ЧОП. Описание путей реализации ПС для ЧОП; Демонстрация
результата.

Дата выдачи задания «13» января 2022 г.

Срок представления работы к защите «25» апреля 2022 г.

Руководитель работы _____

Задание принял к исполнению _____
(дата и подпись студента)

Содержание

Введение.....	5
1. Теоретическая часть.....	7
1.1 Организация ЧОП.....	7
1.2 Информационные потоки ЧОП.....	8
1.3 Процессы внутри ЧОП.....	8
1.3.1 Описание процессов.....	9
Составление контракта.....	10
Прием на работу.....	11
Регистрация и выдача оружия.....	11
Составление отчета об инциденте на объекте.....	11
Выплаты.....	11
1.4 Описание средств разработки.....	12
1.4.1 Утилиты для разработки баз данных.....	12
Plantuml v1.2021.16.....	12
SQLite v3.35.5 stable.....	12
SQLiteBrowser v3.12.2.....	13
DBVisualizer v12.1.8.....	13
1.4.2 Библиотека Qt v6.2.4.....	13
1.4.3 Системы сборки и компиляторы.....	14
CMake v3.22.3.....	14
Ninja c1.10.2.....	14
clang v13.0.1.....	14
1.4.4 Интегрированные среды разработки (IDE).....	14
QtCreator v6.0.2.....	14
VIM v8.2.....	15
GIT v2.35.1.....	15
GitHub.....	15
2.1 Разработка базы данных ЧОП.....	16
2.1.1 Разработка таблиц.....	16
2.1.2 Визуализация базы данных.....	20
2.3 Разработка архитектуры приложения.....	20
2.3.1 Информационные потоки ЧОП.....	22
2.3.2 База данных.....	22
2.3.2.1 Система безопасности.....	23
Система идентификации и авторизации.....	24
Система команд.....	25
2.4 Архитектура Клиент-Сервер.....	25
2.4.1 Протокол.....	25
Формат сообщений.....	25
2.4.2 Модель сервера.....	26
2.4.2.1 Система регистрации.....	27
2.4.3 Модель клиента.....	27

2.5 Реализация.....	28
2.5.1 Пользовательский интерфейс.....	30
2.5.2 PagesManager.....	30
2.5.3 NotifyManager.....	33
2.5.4 Журналирование.....	34
2.5.5 iiNPack.....	35
2.5.6 Сервер.....	35
2.5.6.1 Драйвер базы данных.....	36
2.5.6.1.1 Криптография.....	39
2.5.7 Менеджер подключений.....	42
2.5.7.1 Линк.....	42
2.5.7.2 Процессор подключений.....	43
2.5.7.3 Сервис регистрации.....	43
2.5.8 Клиент.....	43
2.5.8.1 Service.....	43
2.5.8.2 Менеджер групп страниц.....	44
2.6 Демонстрация результата.....	44
2.6.1 Сервер.....	44
2.6.2 Клиент.....	47
Заключение.....	53
Список источников.....	54

Введение

В настоящее время оборот информации в бизнес сфере огромен, оцифровано практически всё. Большая часть информации передаётся по средствам компьютерных сетей, в частности глобальной – Интернет.

Технология Интернет приспособлена 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 посягающий на несанкционированное получения доступа к ресурсам подключенных к Интернет. Вследствие чего начали стремительно развиваться технологии Криптографии, которые существующие еще со времён древнего Рима, и других методов борьбы со взломщиками и не только.

Взломщикам могут быть интересны любые ресурсы: журналы Бухгалтерии, системы управления предприятием и т.д.

В сфере бизнеса ЧОП злоумышленники могут быть заинтересованы в получении запланированных маршрутов инкассации, адреса жительства сотрудников и многом другом. Для пресечения перечисленных выше махинаций можно полностью отказаться от ведения своей деятельности в Интернет.

Если это не выход – тогда необходимо развертывание системы защиты, что и будет реализовано в пределах данной работы.

Решение будет выполнено в несколько шагов:

1. анализ структуры ЧОП;
2. анализ проходимых процессов в ЧОП;
3. описание средств разработки;
4. составление базы данных ЧОП;
5. описание подходов реализации защиты базы данных;
6. разработка архитектуры приложения для взаимодействия с ЧОП;

7. реализация архитектурных решений.

Отследить прогресс проекта и найти исходный код можно на github:
<https://github.com/siisgoo/siisty>.

Так же данную курсовую можно прочитать онлайн по адресу:
<https://siisgoo.github.io/siisty>.

Исходный текст курсовой расположен по адресу:
<https://github.com/siisgoo/siisty/tree/main/cursed>.

Остальные ресурсы, связанные с документацией проекта:
<https://github.com/siisgoo/siisty/tree/main/docs>.

1. Теоретическая часть

1.1 Организация ЧОП

Частное охранное предприятие (ЧОП) занимается охраной какой либо частной или государственной собственности.

ЧОП, чаще всего, состоит из следующих сущностей:

- Администрация;
- Бухгалтерия;
- Отдел кадров;
- Отдел охраны;
- Отдел инкассации;
- Отдел вооружения.

Если говорить о предоставляемых услугах более конкретно, то ЧОП для заказчика предлагает:

- Охрану объекта;
- Охрану и транспортировка ценных бумаг или металлов(инкассация)

Структура ЧОП (рис. 1.1):

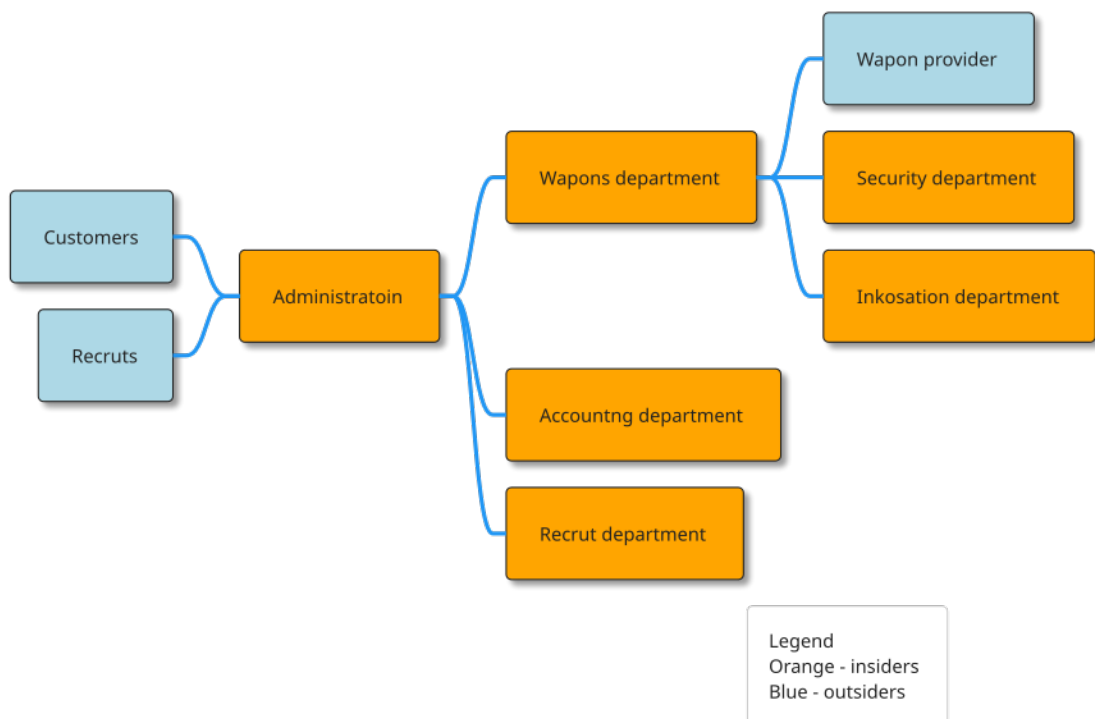


Рис. 1.1 Структура ЧОП

1.2 Информационные потоки ЧОП

Проанализировав модель ЧОП, были выявлены следующие информационные потоки (рис. 1.2):

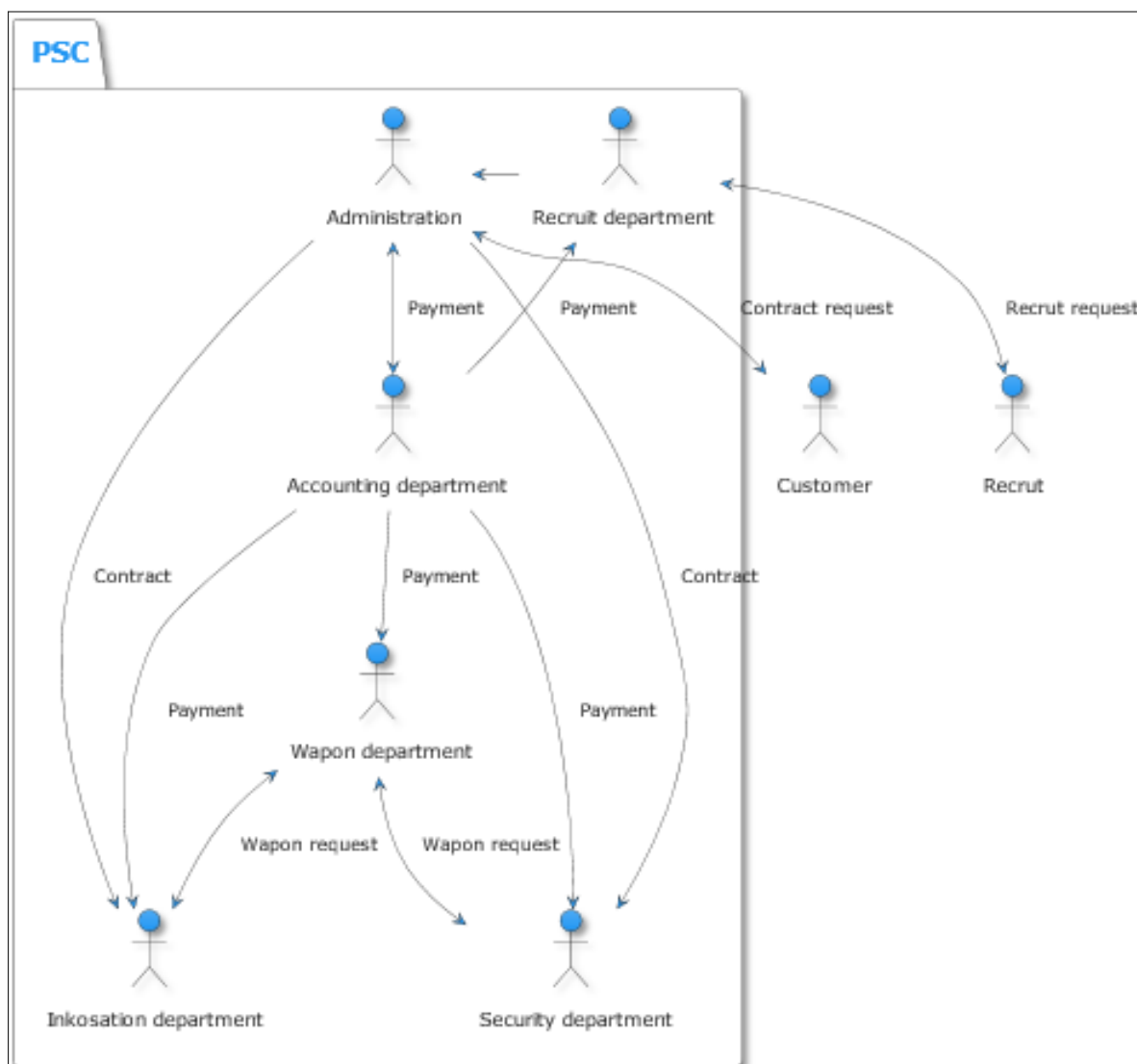


Рис. 1.2 Схема информационных потоков ЧОП

1.3 Процессы внутри ЧОП

В стандартном ЧОП можно рассматривать следующие группы процессов: Внутренние и Внешние.

Внутренние - описывают действия, совершаемые в пределах организации, между сотрудниками.

Внешние - действия с лицами из вне.

Лицами из вне являются сущности «Заказчик» и «Рекрут». Под рекрутом понимается желающий вступить в организацию (рис. 1.3):

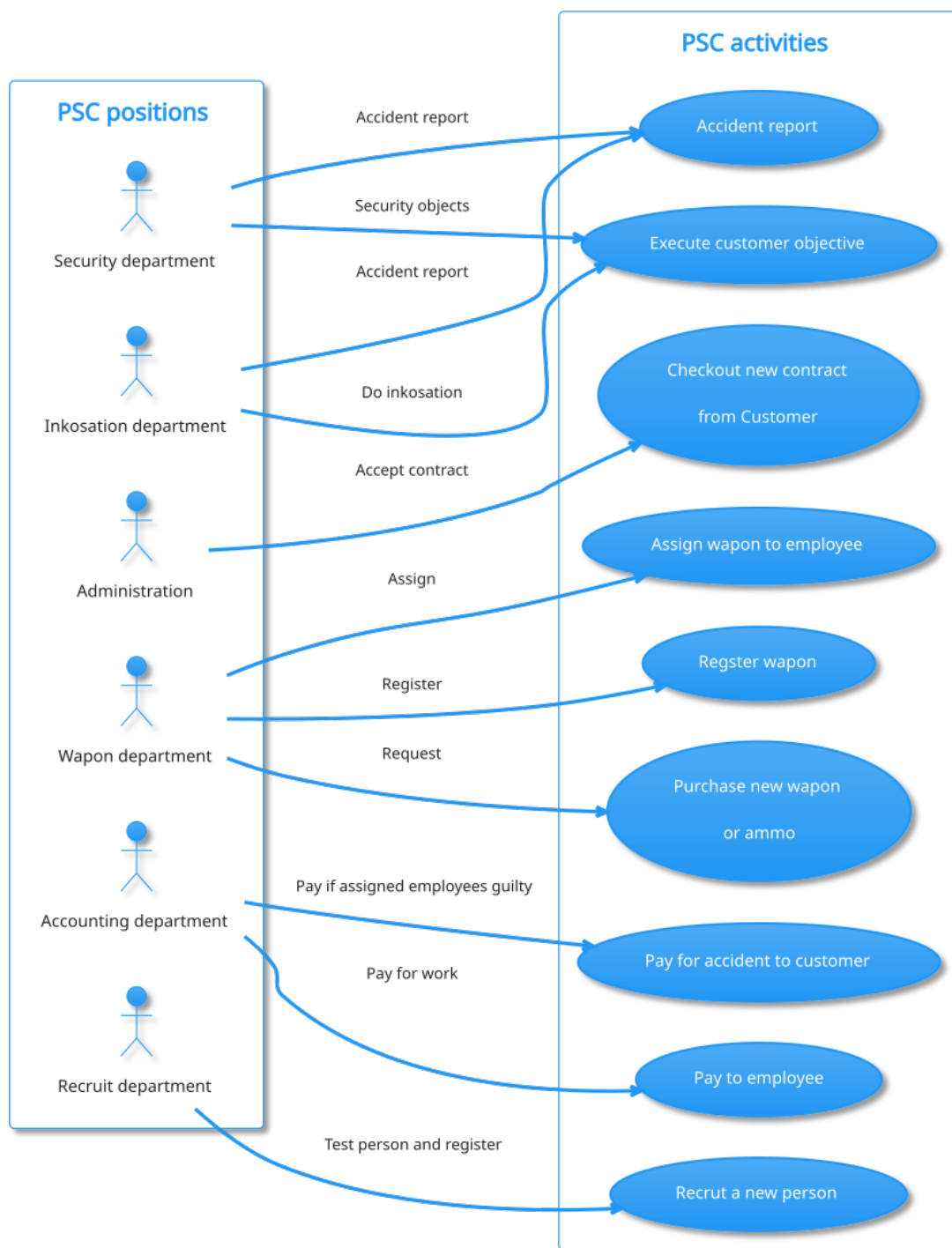


Рис. 1.3 Процессы внутри ЧОП

1.3.1 Описание процессов

В данном разделе будут более детально рассмотрены вышеперечисленные процессы.

Составление контракта

Вся прибыль ЧОП держится на составлении контрактов с заказчиками. Небольшой алгоритм по составлению и подтверждению контракта

1. Для составления контракта заказчик сначала выбирает услугу, которую он хочет получить от ЧОП.
2. Затем ЧОП предлагает ему работников, которые могли бы выполнить этот заказ или сообщают о невозможности выполнения данного контракта в данный момент.
3. После заказчик выбирает из списка сотрудников подходящих или оставляет выбор за ЧОП.
4. И наконец - оплачивает заказ.

В общем виде это выглядит следующим образом (рис. 1.4):

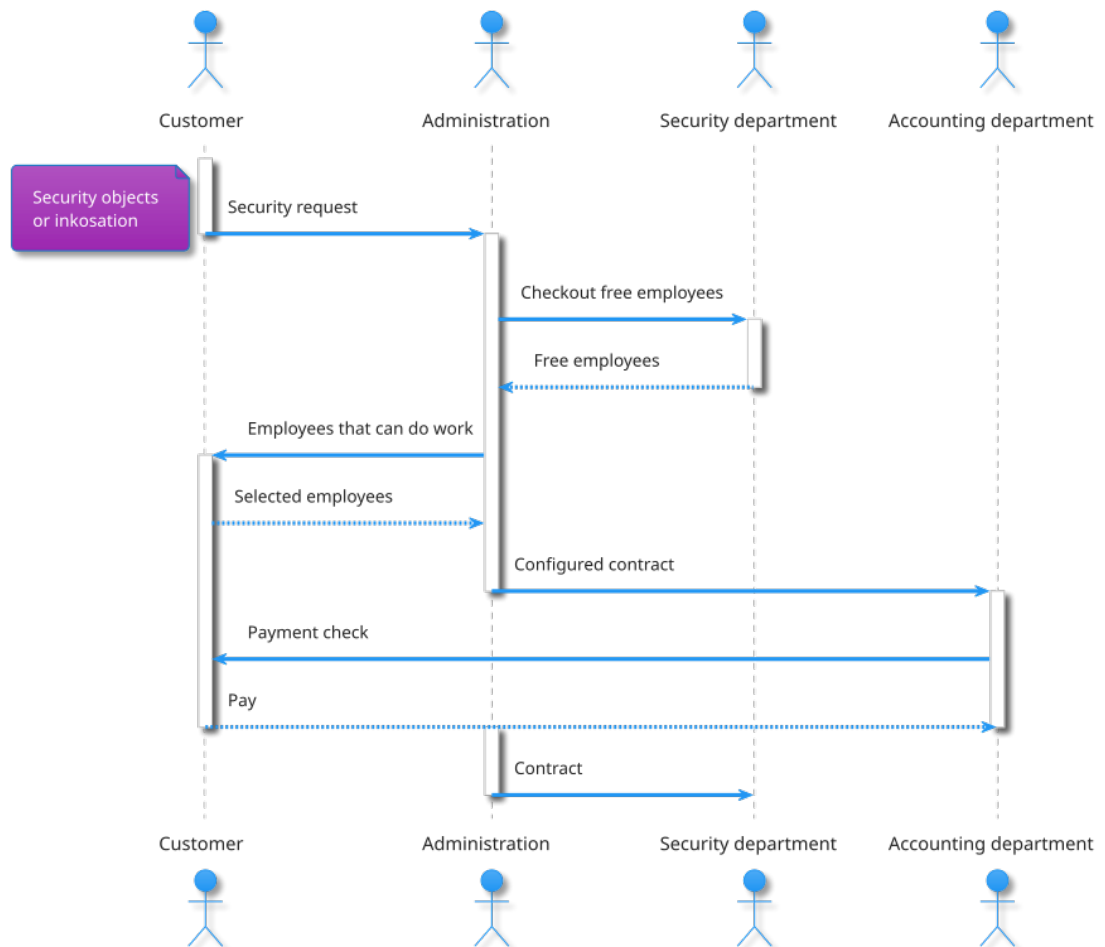


Рис. 1.4 Последовательность составления контракта

Зачастую, для обороны объектов сотрудников ЧОП используется

оружие, но для этого его нужно откуда-то брать. Закупкой будет заниматься Оружейный отдел.

Прием на работу

Прием на работу состоит из этапов подачи заявки на рассмотрение кандидатуры, одобрения департаментом рекрутинга и оповещения администрации.

Регистрация и выдача оружия

Регистрацией оружия занимается профилированный под это дело человек, эти детали будут опущены. Операция регистрации/выдачи оружия выполняется по фактору необходимости наличия оружия при службе на объекте сотрудника департаментов инкассации и охраны(проверка на умения владения оружием проводится на этапе рекрутинга), назначенным заказчиком.

Составление отчета об инциденте на объекте

Обязательным предшествующим событием перед составлением отчета является доклад(звонок/письмо/оповещение) заказчика о нарушении сохранности объекта. После администрация выезжает на место и оценивает ущерб(не имеет финансов нанять специалистов), после чего составляется полный отчет и предоставляется заказчику. В случае не 0% вины сотрудника, выполняющего свои обязанности во время инцидента из его зарплаты вычитается сумма ущерба, если сумма ущерба превышает зарплату сотрудника, сумма ущерба вычитается из текущей и последующих X зарплат так, чтобы минимальная реальная зарплата сотрудника не была ниже минимальной установленной зарплаты.

Выплаты

Выплаты работникам:

На баланс сотрудника переводится сумма, рассчитанная через некоторые показатели сотрудника и параметров объекта, на котором он выполняет службу. Выплата совершается, если сотрудник был задействован в любом объекте.

Компенсации за нанесенный ущерб объекту охраны при не нулевом проценте вины сотрудника: Сумма ущерба переводится одной транзакцией на счет клиента. Денежная сумма выплаты либо сразу взимается из текущей зарплаты виновного сотрудника/виновных сотрудников, либо берется из резервов, если сумма выплаты превышает текущую зарплату сотрудника/сотрудников.

1.4 Описание средств разработки

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

1.4.1 Утилиты для разработки баз данных

Plantuml v1.2021.16

Средство создания UML диаграмм. Использовано для визуализации объектов и процессов. Логотип программного продукта (рис. 1.4)



Рис. 1.4 Логотип Plantuml v1.2021.16

SQLite v3.35.5 stable

SQLite — компактная встраиваемая СУБД. Логотип программного продукта (рис. 1.5)



Рис. 1.5 Логотип SQLite

Исходный код библиотеки передан в общественное достояние. Данная СУБД работает в без серверной конфигурации. Если сравнивать с другими СУБД, то в равных условиях запись SQLite осуществляет медленнее на 20-30% чем другие СУБД, но чтение превосходит другие на 40-50%. SQLite не имеет привилегий, только систему авторизации, но это и не нужно в моем проекте. Библиотека SQLite не будет использована в чистом виде, а в составе Qt v6.2.4

SQLiteBrowser v3.12.2

Удобный FOSS браузер баз данных SQLite, использованный для отладки. Логотип программного продукта (рис. 1.6)



Рис. 1.6 Логотип SQLiteBrowser v3.12.2

DBVisualizer v12.1.8



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

1.4.2 Библиотека Qt v6.2.4

Qt - один из самых популярных и больших фреймворков с++ на рынке (рис. 1.6).



Рис. 1.6 Логотип Qt v6.2.4

Важная характеристика Qt - переносимость, т.к. работа выполнялась на платформе Linux. Содержит все необходимые компоненты для создания приложения любой сложности. Имеет одну из самых мощных систем создания пользовательских интерфейсов, таких как QML+QtQuick и систему QWidget.

1.4.3 Системы сборки и компиляторы

CMake v3.22.3

Система сборки c++ (рис. 1.7):



Рис. Логатип Cmake v3.22.3

Сборка проекта и передача более низкоуровневому средству.

Ninja v1.10.2

Еще одна система сборки, только уже более низкого уровня, чем Cmake (рис. 1.8):



Рис. 1.8 Логатип Ninja v1.10.2

Функция - передача исходного кода на компиляцию.

clang v13.0.1

Компилятор семейства C (рис. 1.9):



Рис. 1.9 Логатип clang v13.0.1

1.4.4 Интегрированные среды разработки (IDE)

QtCreator v6.0.2

IDE от компании The Qt Company (рис 1.10):



Рис. 1.10 Логатим QtCreator v6.0.2

Использованный только как средство отладки и создания скелетов форм пользовательского интерфейса.

VIM v8.2

Улучшенный vi профилируемый текстовый редактор (рис. 1.11):



Рис. 1.11 Логотип Vim v8.2

В данной курсовой работе основная часть работы будет проведена в IDE, в своей основе так же прост, как и каноничный “блокнот” в Windows, только с максимальной степенью кастомизации. Главное достоинство - это управление без использования мыши и возможность настройки управляющих комбинаций максимально удобно, что сокращает время разработки.

Главной особенностью данного ПО (VIM v8.2) является возможность использования плагинов – сторонних программ, написанных на языках VimScript или Python, для расширения функционала.

Утилита профилирования и отладки программы Valgrind v3.18

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

1.4.5 Система контроля версий GIT

GIT v2.35.1

GIT - система контроля версий (рис. 1.12):



Рис. 1.12 Логатип Git

Git позволяет отслеживать и управлять версиями ПО.

GitHub

Интернет-ресурс оперирующий возможностями vsc git (рис. 1.12):



Рис. 1.13 Логотип GitHub

Цель github платформы — объединить разработчиков со всего мира, создать базу ПО с открытым исходным кодом и предоставить инструменты для корпоративной разработки ПО.

2. Практическая часть

2.1 Разработка базы данных ЧОП

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

2.1.1 Разработка таблиц

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

Листинг 1

```
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 открытом виде, а зашифрован с использованием динамической соли по алгоритму “Preferred salt algorithm”, более подробно будет рассмотрен далее.

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

Листинг 2

```
CREATE TABLE  
  "Roles"("id" INTEGER NOT NULL UNIQUE,  
    "name" TEXT NOT NULL UNIQUE,  
    "commands_id" INTEGER NOT NULL,  
    "payMultiplier" 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 roleComands - это SQL массив с ID команд, которые может выполнять пользователь с данной ролью, поэтому было принято решение об отказе от других, более тяжелых СУБД, т.к. все необходимые действия делегируются на Сервер. От СУБД требуется только хранить данные и извлекать их. Связанная таблица roleCommands:

Листинг 3

```
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:

Листинг 4

```
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 Warons - это единица зарегистрированного оружия, в полной мере описывающая необходимые характеристики для ЧОП.

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

Листинг 5

```
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.

Листинг 6

```
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:

Листинг 7

```
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 связанной таблице objectType. Таблица objectType:

Листинг 8

```
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 базовая цена за период оплаты(payPeriod) исполнителя/исполнителей контракта(assignedEmployees_id). Для привязки нескольких сотрудников, была создана еще одна таблица AssignedEmployees, являющейся массивом.

Листинг 9

```
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 как во время исполнения может произойти какой то инцидент, то была создана таблица Accidents:

Листинг 10

```
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 DELETE RESTRICT,  
              PRIMARY KEY("id") )
```

Описывает происшествия, 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:

Листинг 11

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

day - это 64х битная цифра со знаком в формате UNIX time(secs since epoch).

2.1.2 Визуализация базы данных

Визуализация структуры базы данных представлена на рисунке 2.1:

2.3 Разработка архитектуры приложения

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

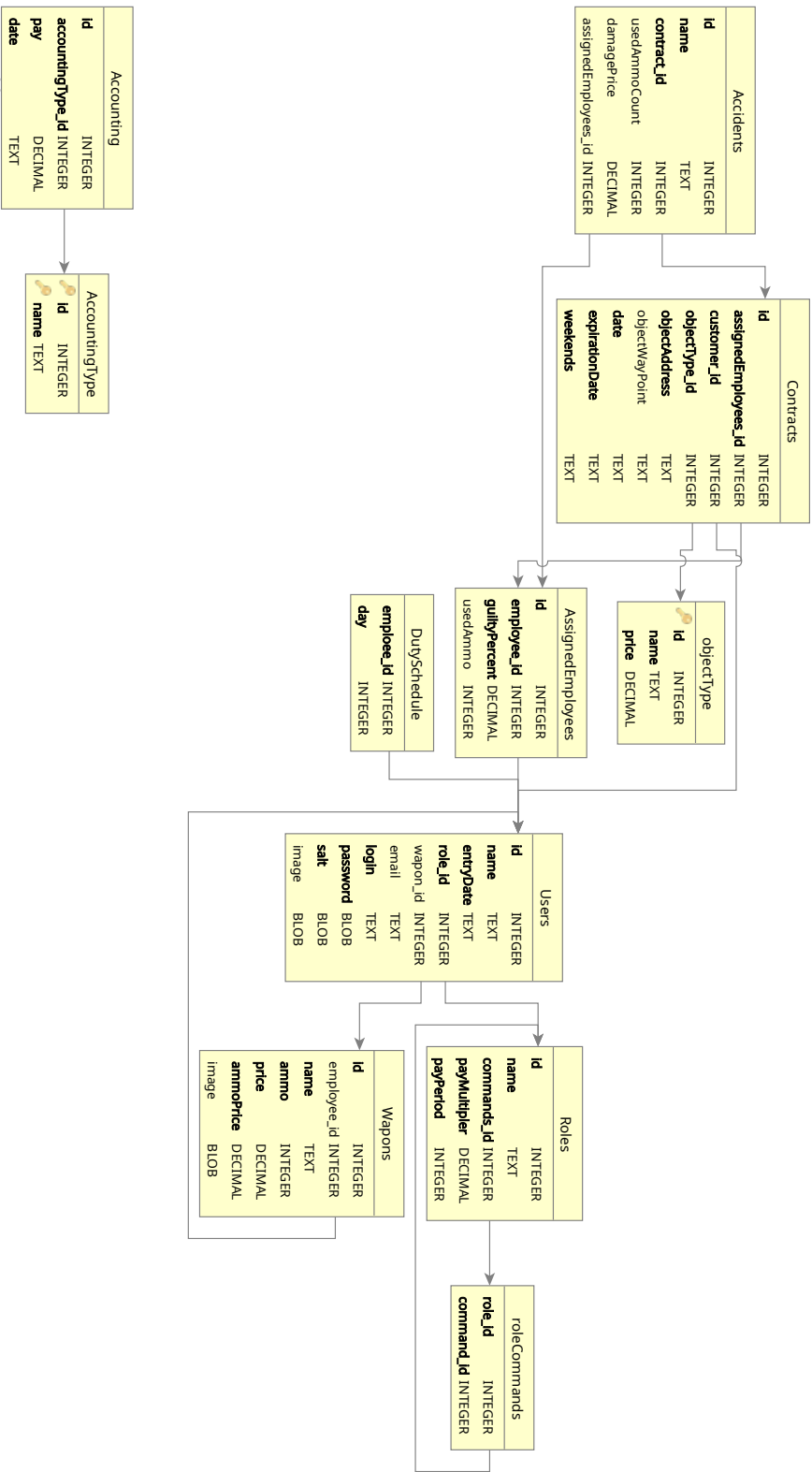


Рис. 2.1 Визуализация связей базы данных

Основная функция приложения

Основной функцией приложения 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.

2.3.1 Информационные потоки ЧОП

При вводе новой концепции оперирования с информацией, необходимо перопределить пути информационных потоков.

Модель взаимодействия с учетом сервера представлена на следующей схеме (рис. 2.2).

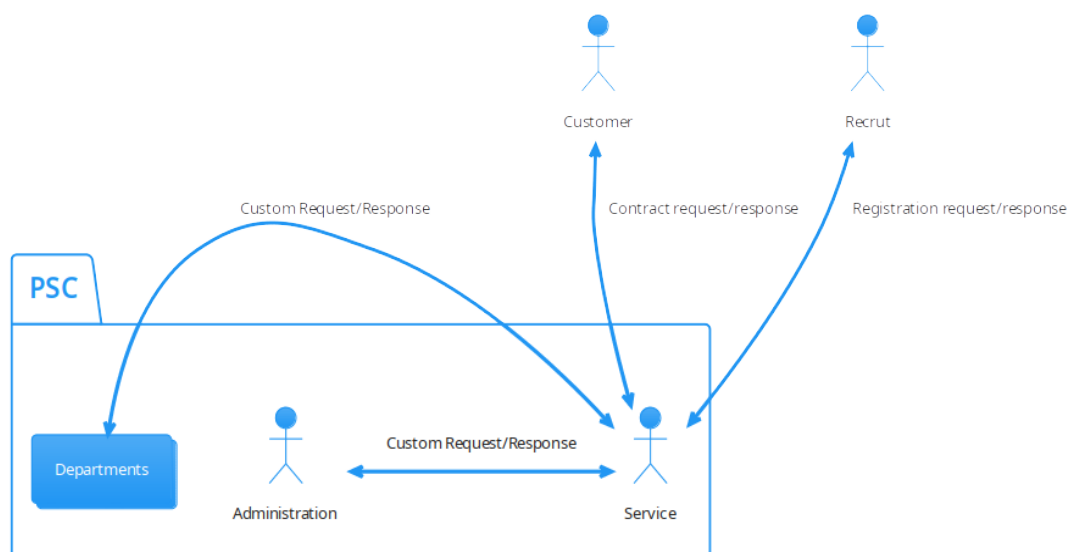


Рис. 2.2 Модель информационных потоков с учетом сервера

2.3.2 База данных

СУБД не будет управлять системой привилегий, этим будет заниматься другой код. Система основана на ролях, также, как и в обычных “умных” СУБД, к которым привязано некоторое количество возможных к исполнению команд.

База данных должна будет сама себя обслуживать и проверять целостность обязательных записей, который вшиты в исходный код (рис. 2.3).

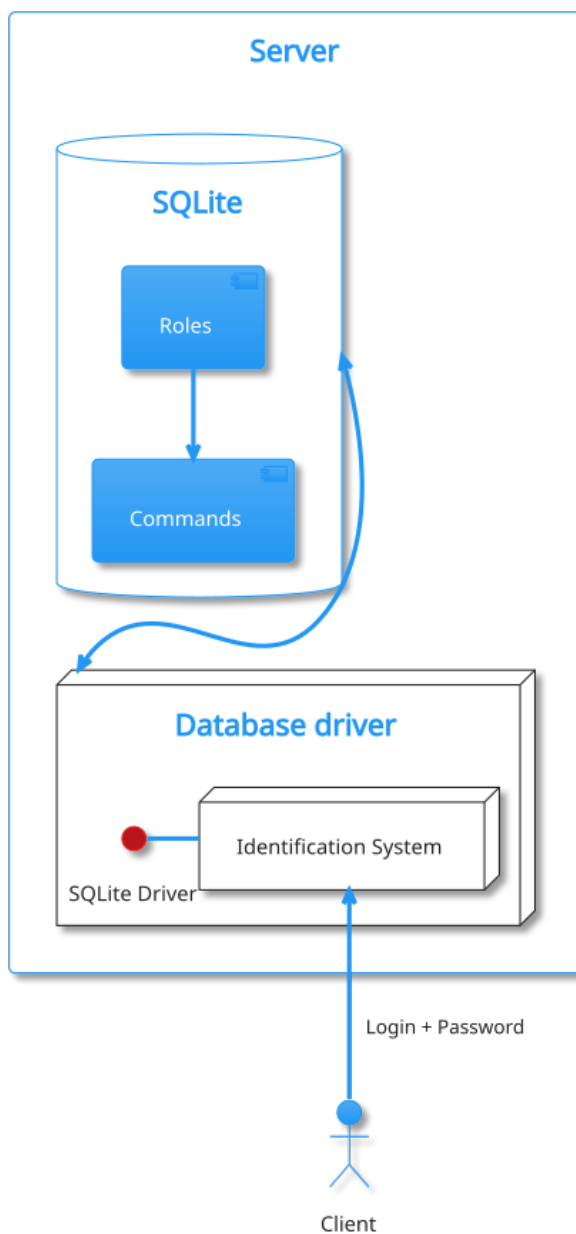


Рис. 2.4 Модель сервера

2.3.2.1 Система безопасности

Для обеспечения более высокого уровня защиты команды и роли будут вшиты в программу. При каждом старте будет проверяться 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 неизменном виде. password сохраняется в захешированном виде при использовании хэш функции sha512 с использованием 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 составленного алфавита.
5. Запись результата.

Алгоритм хеширования 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 алгоритм и выбранная хэш функция ограничивают максимальную длину пароля до 64 символов. Длина соли была выбрана 32 байтная.

Соль сохраняется вместе с паролем, чтобы обеспечить возможность идентификации.

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

Система команд

Проанализировав информационные потоки и требования ЧОП было определено и реализовано 22 команды.

2.4 Архитектура Клиент-Сервер

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

2.4.1 Протокол

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

Формат сообщений

Программа клиент и программа сервер будут вести обмен полезными данными с помощью форматированных сообщений, они были названы iiNPack.

Состоит из заголовка и полезной нагрузки.

Заголовок:

Длина заголовка - 176 бит или 22 байта. Конечно, можно было бы спокойно использовать все 192 бита(3 машинных слова), чтобы выровнять заголовок, но это не сильно повлияет на какой либо процесс.

Байты	Название	Описание
0x0 - 0x3	Size	Суммарный размер сообщения(байт)
0x4 - 0x11	ServerStamp	Время отправки сообщения сервером
0x12 -	ClientStamp	Время отправки сообщения клиентом

Байты	Название	Описание
0x19		
0x20 - 0x21	Type	Тип сообщения
0x22 - 0x23	Format	Формат назгрузки сообщения

Такблица 2.1 Заголовок сообщения IINPack

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.

2.4.2 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 пользователя, для управления организационными процессами ЧОП.

Следовательно у сервера должен быть интерфейс для создания пользователей, как и у клиента.

Схама сервера (рис. 2.5):

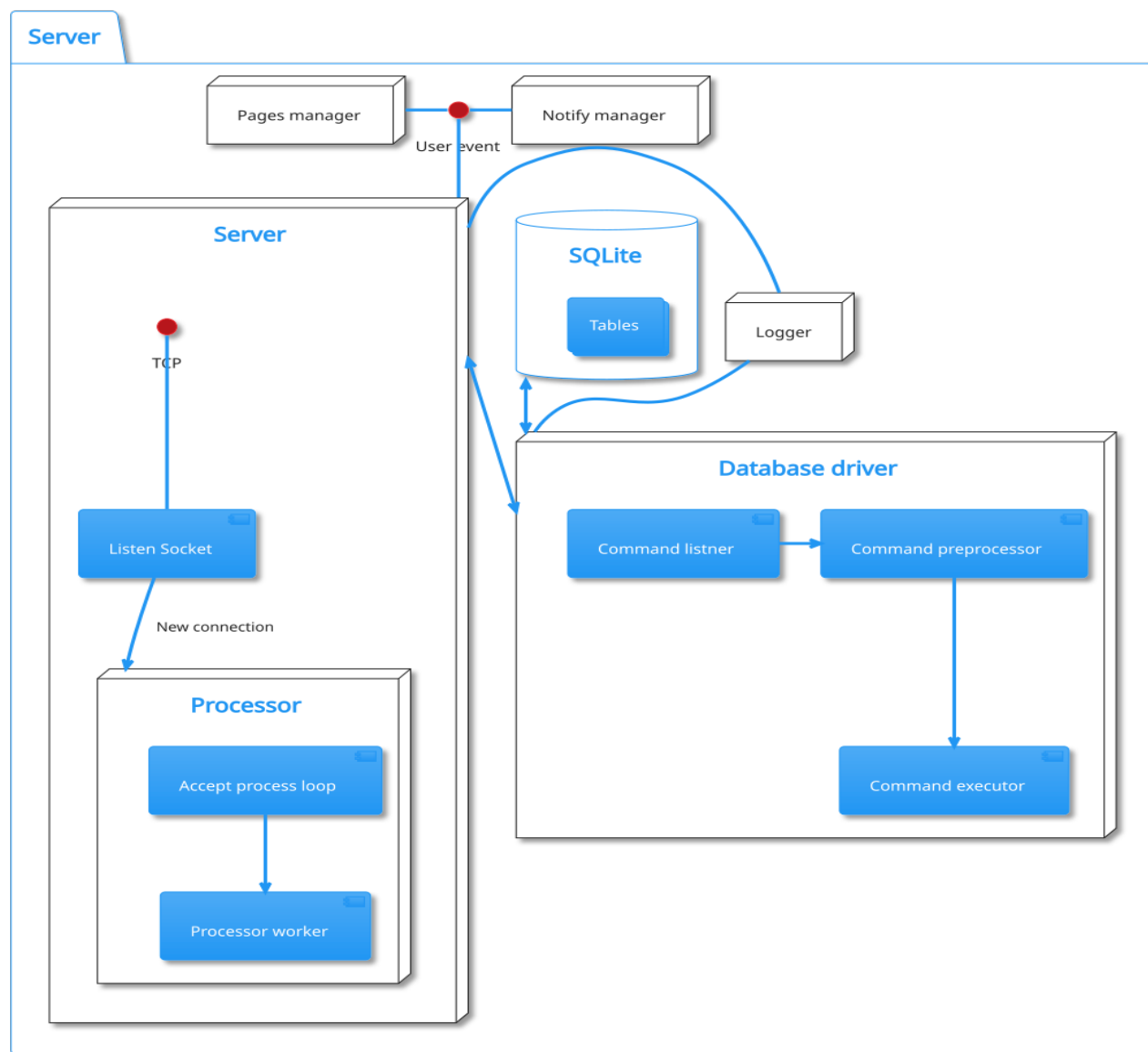


Рис. 2.5 Модель работы сервера

2.4.2.1 Система регистрации

Идентификация и авторизацией пользователей занимается база данных, а регистрацией новых пользователей типа “Заказчик” и “Рекрут” будет сервер т.к. процесс регистрации будет содержать помимо создания новой записи в таблице Users, но и подтверждение личности через электронную почту.

2.4.3 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 пользователь, который воспользовался клиентом является заказчик - клиент должен предоставить метод регистрации в систему.

Модель клиента (рис. 2.6):

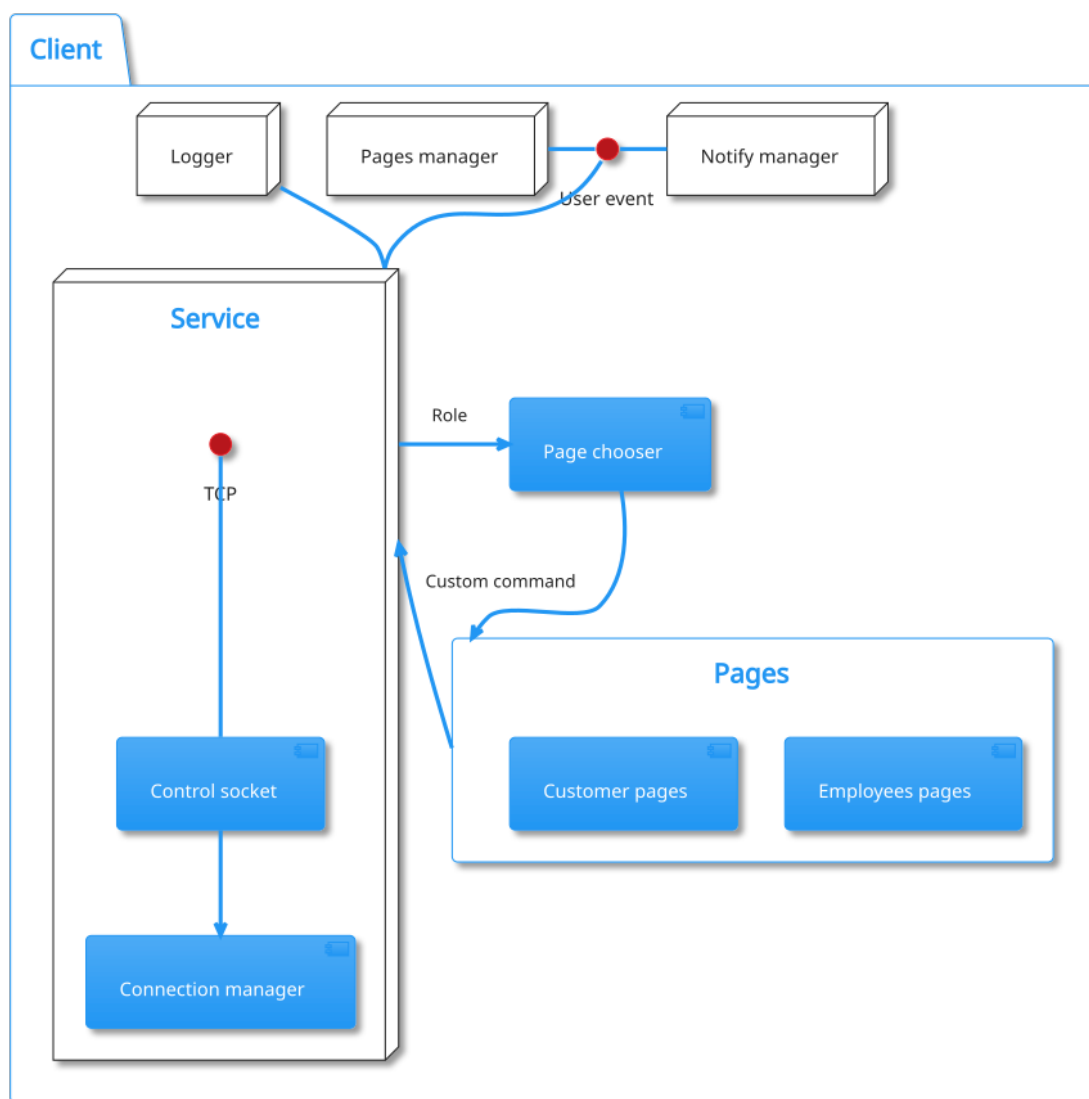


Рис. 2.6 Модель клиента

2.5 Реализация

Для реализации проекта был выбран язык C++ и фреймворк Qt6. Выбор лег в сторону C++ только потому что это мной наиболее изученный язык, на

много проще было бы всё реализовать на nodejs с express, а Qt - это лучший кроссплатформенный 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 QtDesigner.

При разработке будет применен модульный принцип программирования - отдельным модуль - динамическая библиотека или библиотека типа MODULE, загружаемая dlopen-like методом, т.к. пользовательский интерфейс для клиента тоже будет находиться в подмодулях основного приложения клиента и таким образом можно будет сократить кол-во использования памяти.

Список модулей:

- Crypto
- Database
- iiServer
- Logger
- Math
- Network
- PagesManager
- NotifyManager
- Widgets

Использование динамических библиотек обусловлено уменьшением времени компиляции и идеологией разработки на Linux. Большая часть приложения будет реализована с использованием ООП, только модуль Database будет наполовину использовать чистое функциональное c-style программирование.

Т.к. разработка ведется в пределах курсовой работы, релиз приложения не будет выходить дальше Pre-alpha и для работы на платформе Linux.

2.5.1 Пользовательский интерфейс

За отрисовку пользовательского интерфейса будет отвечать модуль Qt - QtGUI.

Пользовательский интерфейс будут составлять 5 основных сущностей:

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

2.5.2 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:

Листинг 12

```
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:

Листинг 13

```

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 QStackedWidget - 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 PagePathFrame, наследуемый от QFrame, является второй основной сущность пейджера. Основным методом класса является метод:

Листинг 14

```

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:

Листинг 15

```

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.

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

Третьей сущностью является навигационная панель - NavWidget. Содержит связанные с данной страницей ссылки в виде кнопок. Главный метод добавления навигационного меню:

Листинг 16

```

int NavWidget::addNav(const QVector<QString> & pages, bool createBack){
    NavFrame *fnav = new NavFrame(pages, createBack, this);
    this->addWidget(fnnav);
    connect(fnnav, 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:

Листинг 17

```
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);
}
```

2.5.3 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 использую фабрики NotifyItemFactory.

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

Листинг 18

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

Обращение к созданному уведомлению происходит по выделенному uid, который возвращает createNotifyItem.

Все существующие уведомления хранятся в именованном массиве:

Листинг 19

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

Но лучше было бы хранить их в связанном списке, так бы сохранялся их порядок появления и упрощалась перегруппировка с возможностью более быстрой обработки анимации движения появления и скрытия, пока что единственная доступная анимация - изменение прозрачности.

Существующие уведомления перегруппировываются при заверении, изменении размера окна. При изменении максимальной ширины уведомления

Для управления поп-апами из вне используется система QProperty в мета объекте.

Класс является *thread-safe*.

2.5.4 Журналирование

Логер реализован по простой модели работы в отдельном потоке. Использует систмему уровней уведомления, для фильтрации вывода. Для вывода в лог используется система событий Qt. Для фильтрации доступны уровни:

Листинг 20

```
enum LogLevel {  
    Trace = 0,  
    Debug,  
    Info,  
    Warning,  
    Error,  
    Fatal,  
};
```

2.5.5 iiNPack

Класс реализующий протокол описанный выше, включает методы упаковки данных для упрощения процесса передачи данных.

Главные определения в классе:

Листинг 21

```
enum PacketType : quint8{
    AUTHORIZATION_REQUEST,
    REQUEST,
    RESPONSE,
    ERROR_MESSAGE,
};
enum PacketLoadType : quint8{
    JSON = 0,
    XML,
    RAW,
};
enum ResponseError : quint8{
    ACCESS_DENIED = 0,
    NETWORK_ERROR,
    REQUEST_ERROR,
    UNSUPPORTED_FORMAT,
    UNSUPPORTED_TYPE,
    PARSE_ERROR,
};
struct Header{
    /* 0x0 - 0x3 */ quint32 Size;          /* Overall packet load size in bytes */
    /* 0x4 - 0x11 */ qint64 ServerStamp; /* Send time on server; using QDateTime
    SecsSinceEpoch */
    /* 0x12 - 0x19 */ qint64 ClientStamp; /* Send time on client; using QDateTime
    SecsSinceEpoch */
    /* 0x20 - 0x21 */ quint8 PacketType; /* Type of packet; enum class PacketType
    */
    /* 0x22 - 0x23 */ quint8 PacketLoadType; /* Load format, see enum class
    PacketLoadType */
};
```

2.5.6 Сервер

В данном подразделе будут описаны пути реализации приложения-сервера.

2.5.6.1 Драйвер базы данных

По сути, драйвер базы данных - это декоратор QSqlDatabase класса предназначенный для работы в отдельном потоке. По мимо доступа к базе данных, класс реализует систему идентификации и аутентификации.

Т.к. драйвер будет работать в отдельном потоке, чтобы не тормозить другие потоки, из которых вызываются методы драйвера, он имеет в себе очередь команд на выполнение:

Листинг 22

```
struct DatabaseCmd {  
    int executorRole;  
    QJsonObject data;  
    DriverAssistant * waiter;  
};  
DatabaseCmd cmd = _cmdQueue.dequeue();
```

Т.к. передача команды в очередь на исполнение драйверу реализована с помощью сигналов, данный объект DatabaseCmd необходимо ввести в систему мета компиляции:

Листинг 23

```
Q_DECLARE_METATYPE(Database::DatabaseCmd)
```

Обработка происходит в такой незамысловатой петле:

Листинг 24

```
void Driver::worker(){  
    QMutexLocker lock(&_queueMtx);  
    if (_cmdQueue.length()) {  
        DatabaseCmd cmd = _cmdQueue.dequeue();  
        this->executeCommand((RoleId)cmd.executorRole, cmd.data, cmd.waiter);  
    }  
    if (_running) {  
        QTimer::singleShot((_cmdQueue.length() ? 100, 0), this, SLOT(worker()));  
    }  
}
```

И наконец, функция которая реализует механизм аутентификации:

Листинг 25

```
void Driver::executeCommand(Database::RoleId role, QJsonObject obj,  
DriverAssistant* waiter) {
```

```

if (!waiter) {
    throw QString("Driver::" + QString(__func__) + ": Null waiter passed!");
}
if (role != ROLE_AUTO) {
    if (role > ROLES_COUNT || role < (RoleId)0) {
        waiter->Failed(CmdError(AccessDenied, "Invalid Role ID passed"));
        return;
    }
}
int command_n;
if (auto val = obj["command"]; val.isDouble()) {
    command_n = val.toInt();
} else {
    waiter->Failed(CmdError(InvalidCommand, "No command passed"));
    return;
}
if (command_n > COMMANDS_COUNT || command_n < 0) {
    waiter->Failed(CmdError(InvalidCommand, "Command not exists"));
    return;
}
if (role != ROLE_AUTO) {
    // check permission for execute command
    if (!_roles[role].commands.contains(command_n)) {
        waiter->Failed(CmdError(AccessDenied, "You not have access to execute
this command"));
        return;
    }
}
if (auto val = obj["arg"]; val.isObject()) {
    QJsonObject target = val.toObject();
    auto cmd = _commands[command_n];
    CmdError rc = cmd.executor(target);
    if (rc.Ok()) {
        waiter->Success(target);
    } else {
        waiter->Failed(rc);
    }
} else {
    waiter->Failed(CmdError(InvalidParam, "No parameters passed"));}}

```

DriverAssistant - это отдельный класс, который уведомляет объект, который ожидает данные от драйвера, содержит два метода и два сигнала, описывающих успешное или не успешное завершение выполнение команды.

Роли и команды хранятся статично в объекте драйвера в объектах:

Листинг 26

```
struct role_set {  
    int id; // equal to index  
    const char * name;  
    QVector<CommandId> commands;  
};  
struct command_set {  
    int id; // equal to index  
    const char * name;  
    command_exec_t executor;  
    bool sendback;  
};
```

Команды хранятся в массиве, роли - именованном массиве. И заполняются в конструкторе драйвера.

Команды(executor) напрямую получают данные переданные клиент-программами, и возвращают значение в том же переданном аргументе, практически без обработки, таким образом на них ложиться задача валидации переданных аргументов и само исполнение. Пример команды:

Листинг 27

```
/* login: string  
 * password: string */  
CmdError exec_identify(QJsonObject& obj){  
    QSqlQuery q;  
    QString login;  
    QString password;  
    login = obj.take("login").toString();  
    password = obj.take("password").toString();  
    if (!login.length() || !password.length()) {  
        return CmdError(InvalidParam, "Passed empty parameters");  
    }  
    q.prepare("SELECT id, name, role_id, password, salt FROM Users "  
              "WHERE login = :login");  
    q.bindValue(":login", login);  
    if (!q.exec()) {  
        return CmdError(SQLError, q.lastQuery() + " " + q.lastError().text());  
    }  
    if (!q.next()) {  
        return CmdError(AccessDenied, "No user registreted with login: " + login);  
    }  
}
```

```

    }
    QByteArray salt = q.record().value("salt").toByteArray();
    QByteArray real_passwordHash = q.record().value("password").toByteArray();
    QByteArray passed_passwordHash = encryptPassword(password.toLatin1(),
salt);
    if (real_passwordHash != passed_passwordHash) {
        return CmdError(AccessDenied, "Invalid password");
    }
    obj["role_id"] = q.record().value("role").toInt();
    obj["name"] = q.record().value("name").toString();
    obj["id"] = q.record().value("id").toString();
    return CmdError();
}

```

2.5.6.1.1 Криптография

При регистрации пользователя, как было сказано ранее, используется функция хеширования пароля, для большей безопасности учетных записей. Можно было бы использовать просто функцию хеширования, но был выбран более сложный и надежный путь.

Кроме того, что при хешировании используется динамическая соль, так она еще и “умная”, написана так, чтобы максимально увеличивать энтропию пароля, но и “умная” вставка соли в строку пароля. Алгоритм был найден мной в журнале IAENG International Journal of Computer Science 2016 года.

Суть алгоритма в том, чтобы приводить пароль к максимальной энтропии с использованием динамической умной соли и использования особого метода вставки соли в пароль по одному из 4 или 5 правил на выбор.

Умная вставка, по мнению автора, должна свести радужные таблицы к эффективному минимуму, что при тестах он и продемонстрировал.

Чтобы определить слабые стороны пароля необходимо просто перебрать символы пароля и распределить их по трем группам:

- Буквы
- Цифры
- Специальные знаки

и исходя из процентного содержания в исходной строке выбираем алфавит для генерации соли:

```

static void countChars(int& spec, int& dig, int& alph, const char * str) {
    spec = dig = alph = 0;
    for (int i = 0; i < strlen(str); i++) {
        char ch = str[i];
        if (isalpha(ch)) { alph++;
        } else if (isdigit(ch)) { dig++;
        } else if (isgraph(ch) || isspace(ch)) { spec++;
        }
    }
}

quint8 passWeaknesses(const QByteArray& data){
    int len = data.length();
    int spec, nums, alph;
    countChars(spec, nums, alph, data.data());
    double spec_c = static_cast<double>(spec)/len,
           nums_c = static_cast<double>(nums)/len,
           alph_c = static_cast<double>(alph)/len;
    // weakness determinission algo:
    // 1. find max coef char type - mostch
    // 2. if mostch count grater then 50% of password mean that weak in both other
char types;
    // 3. if diff of other char types grater then 10% mean that weak only in one min
char type;
    // 4. if diff less or eq to 10% - weak in both;
    auto determWeakness = [](double f, double s, double t, quint8 wf, quint8 ws,
quint8 wt) -> quint8 {
        if (f > s && f > t) {
            if (f > 0.5) {
                return ws | wt;
            } else {
                if (std::abs(s - t) > 0.1) {
                    return (s > t ? wt : ws);
                } else {
                    return ws | wt;
                }
            }
        }
    };
    return 0;
};

quint8 w1 = determWeakness(spec_c, nums_c, alph_c, Special, Digit, Alpha);
quint8 w2 = determWeakness(nums_c, spec_c, alph_c, Digit, Special, Alpha);
quint8 w3 = determWeakness(alph_c, spec_c, nums_c, Alpha, Special, Digit);

```



```

return std::max(w1, std::max(w2, w3)); // only one gr then 0
}

```

Метод создания динамической соли:

Листинг 29

```

static quint32 strongRand(quint32 min,
                          quint32 max = std::numeric_limits<quint32>::max()){
return QRandomGenerator::securelySeeded().generate() % (max+1 - min) + min;
}
static char randCharFrom(const QLatin1String& d, quint32 rand32) { return
d[rand32 % d.size()].toLatin1(); }
enum charWeakness : quint8 {
    Alpha = 0x1,
    Digit = 0x2,
    Special = 0x4,
};
static QMap<quint8, std::function<char(quint32)>> saltCharGen({
    { Alpha,      [](quint32 rand32) { return randCharFrom(alphabet,
rand32); },
    { Digit,      [](quint32 rand32) { return randCharFrom(digits, rand32); },
    { Special,    [](quint32 rand32) { return randCharFrom(specials, rand32); },
    { Alpha | Digit, [](quint32 rand32) { return (strongRand(0, 1) ?
randCharFrom(alphabet, rand32) : randCharFrom(digits, rand32)); },
    { Alpha | Special, [](quint32 rand32) { return (strongRand(0, 1) ?
randCharFrom(alphabet, rand32) : randCharFrom(specials, rand32)); },
    { Special | Digit, [](quint32 rand32) { return (strongRand(0, 1) ?
randCharFrom(specials, rand32) : randCharFrom(digits, rand32)); },
});
QByteArray saltGen(quint8 w){
    QByteArray salt(salt_length, '\0');
    QVector<quint32> rand;
    rand.resize(salt_length);
    QRandomGenerator::securelySeeded().fillRange(rand.data(), rand.size());
    auto genf = saltCharGen[w];
    for (int i = 0; i < salt_length; i++) {
        salt[i] = genf(rand[i]);
    }
    return salt;
}

```

И заключающим аккордом является сама функция генерации хэша пароля:

```

static void setBit(unsigned long& num, unsigned long bit) { num |= (1 << bit); }
static int  getBit(unsigned long num, unsigned long bit) { return (num & (1 <<
bit )) >> bit; }
QByteArray encryptPassword(const QByteArray& pass, const QByteArray& salt,
QCryptographicHash::Algorithm hashAlgo){
    QByteArray toCrypt;
    QByteArray hash = QCryptographicHash::hash(pass, hashAlgo);
    int prev = -1;
    int cur;
    int inserted = 0;
    for (int i = 0; i < pass.length(); i++) {
        toCrypt.push_back(pass[i]);
        if (inserted < salt.length()) {
            cur = getBit(pass[i], sizeof(pass[i])) ^
                getBit(hash[i], sizeof(hash[i]));
            if (cur == 1) {
                toCrypt.push_back(salt[inserted++]);
            } else if (prev == 0) { // two consecutive zeros
                toCrypt.push_back(salt[inserted++]);
                toCrypt.push_back(salt[inserted++]);
                i++; //skip 2
            }
            prev = cur;
        }
    }
    for (int i = inserted; i < salt_length; i++) {
        toCrypt.push_back(salt[i]);
    }
    return QCryptographicHash::hash(toCrypt, hashAlgo);
}

```

2.5.7 Менеджер подключений

Менеджер подключений- это класс содержащий стандартные методы создания серверного слушающего и клиентских сокетов и обработки “сообщений” от клиентских сокетов.

2.5.7.1 Линк

Линк - объект содержащий клиентский сокет, созданный сервером и содержащий информацию о подключенном клиенте `siisty`, методы по

парсингу типизированных “сообщений” `iiNPack` и отсылке “сообщений”.
Генерирует сигналы о изменении состояния сокета и о приеме сообщения.

2.5.7.2 Процессор подключений

Обработка всех TCP линков обслуживается в многопоточном режиме с использованием “диспетчера уведомлений”. На каждое подключение создается линк и помещается в связанный список линков.

При получении сигнала типа “сообщение” от линка создается воркер в пуле потоков, в котором и происходит обработка сообщения, откуда с помощью событийной модели данные передаются в другие модули и это является бутылочным горлышком данной модели, когда из потока данных всех линков все сливается в один, для обеспечения SOLID стиля кода.

2.5.7.3 Сервис регистрации

Т.к. для подачи заявки на вступление в ЧОП и составление контракта клиентом необходимо иметь обратную связь с инициатором события, приложению сервера необходимо иметь какой то сервис, обеспечивающий регистрацию, в нашем случае регистрация будет с подтверждением по e-mail.

Для регистрации нужно послать определенное сообщение на сервер и после отправить еще одно с кодом подтверждения из письма на указанный e-mail. После этого пользователь будет зарегистрирована в системе как Альфа-рекрут или Клиент и с ним можно будет связываться с помощью внутренних сервисов `siisty-server`.

2.5.8 Клиент

В данном подразделе будут описаны пути реализации приложения-клиента.

2.5.8.1 Service

Это объект на подобии Линк, описанного выше, организует интерфейс для доступа к сервисам ЧОП. Инициализирует протоколы подключения к серверу: авторизация и регистрация.

2.5.8.2 Менеджер групп страниц

Менеджер групп страниц - это простой ассоциативный массив, ключем является роль, а данными является inline-функция развертывающая необходимый UI.

Объявление менеджера групп страниц:

Листинг 31

```
using pagemanSetuper = std::function<void(Controller *, PagesManager *, Service *)>;
inline QMap<int, pagemanSetuper> pagerPresets{
    { MAIN_PAGE_ID,      mainPage },
    { Database::ROLE_Admin,      admin },
    { Database::ROLE_Security,    security },
    { Database::ROLE_Recruter,    recruter },
    { Database::ROLE_Inkosor,     incosor },
    { Database::ROLE_WaponManager, waponManager },
    { Database::ROLE_Customer,    customer },
};
```

2.6 Демонстрация результата

В данном разделе будут рассмотрены функции реализованного ПС.

2.6.1 Сервер

Стартовая страница (рис. 2.7):

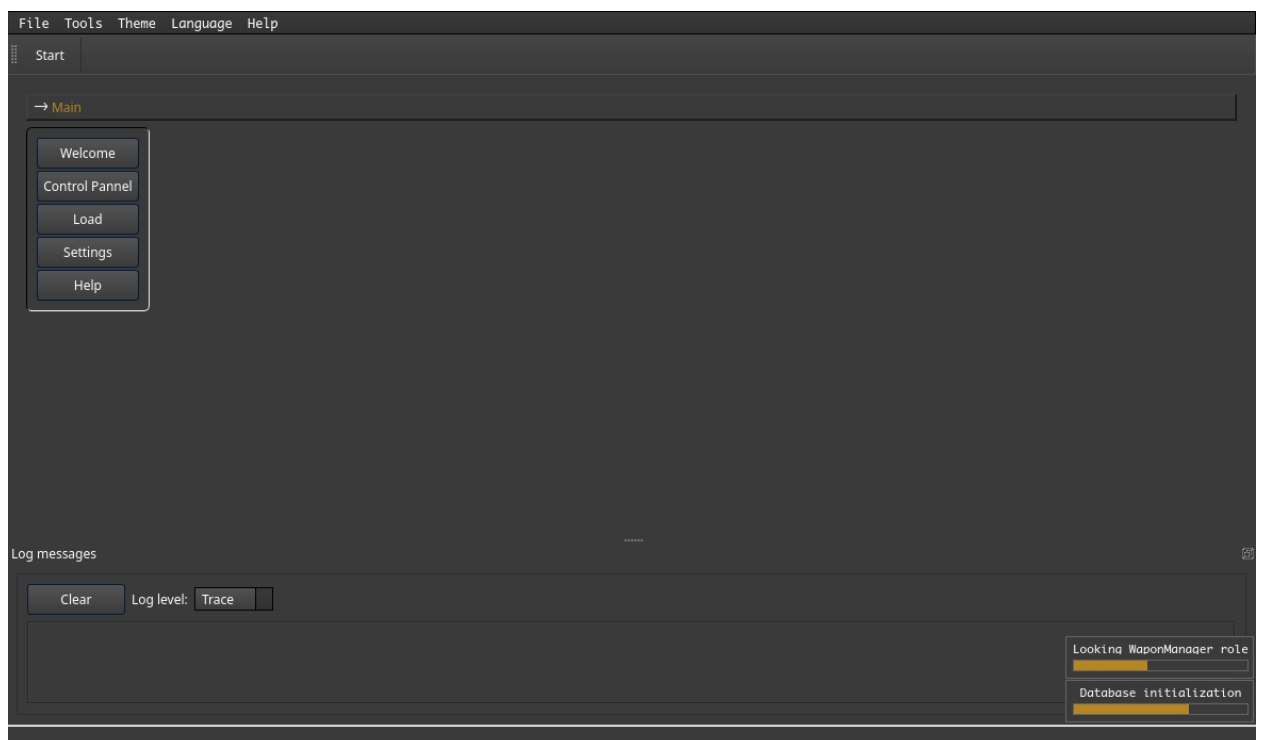


Рис 2.7

Страница регистрации на стороне сервера(рис. 2.8):

File Tools Theme Language Help

Stop

→ Main → Control Pannel → Users → Register user

Register user
Users list
Back

Employee registration

Name: Villa Visp Danger
Login: siisgoo
Password: ••••••••••
Entry date: 5/10/22
Wapon: Update
Role: Admin
Email: siisgoo@mail.ru
Image: tars/kyatava.jpg Open
Save Discard

Рис. 2.8

Страница Welcome(рис. 2.9):

File Tools Theme Language Help

Stop

→ Main → Welcome

Welcome
Control Pannel
Load
Settings
Help

Welcome to sIIIsTy Server manager

sIIIsTy server (iiServer) its a part of sIIIsTy packet.

That program provides a server thats control database, now using *SQLite v3 stable*. This modification serve the Private Security Company (PSC).

Main purpose is a serve client connections and give clients access to database consider the client Role.

iiServer also take care about securty connections support by use optional SSL/TLS.

Right to access the database is static and built-in program but mirrored into database. Right system based on Roles system thats determines commands thats able to executed in server per each Role. To change its, download the [source code](#) and make changes consider the *README.md*.

Open setup page

Log messages

Clear Log level: Trace

Start serving on: 0.0.0.0:2142

Рис. 2.9

Страница управления сервером(рис. 2.10):

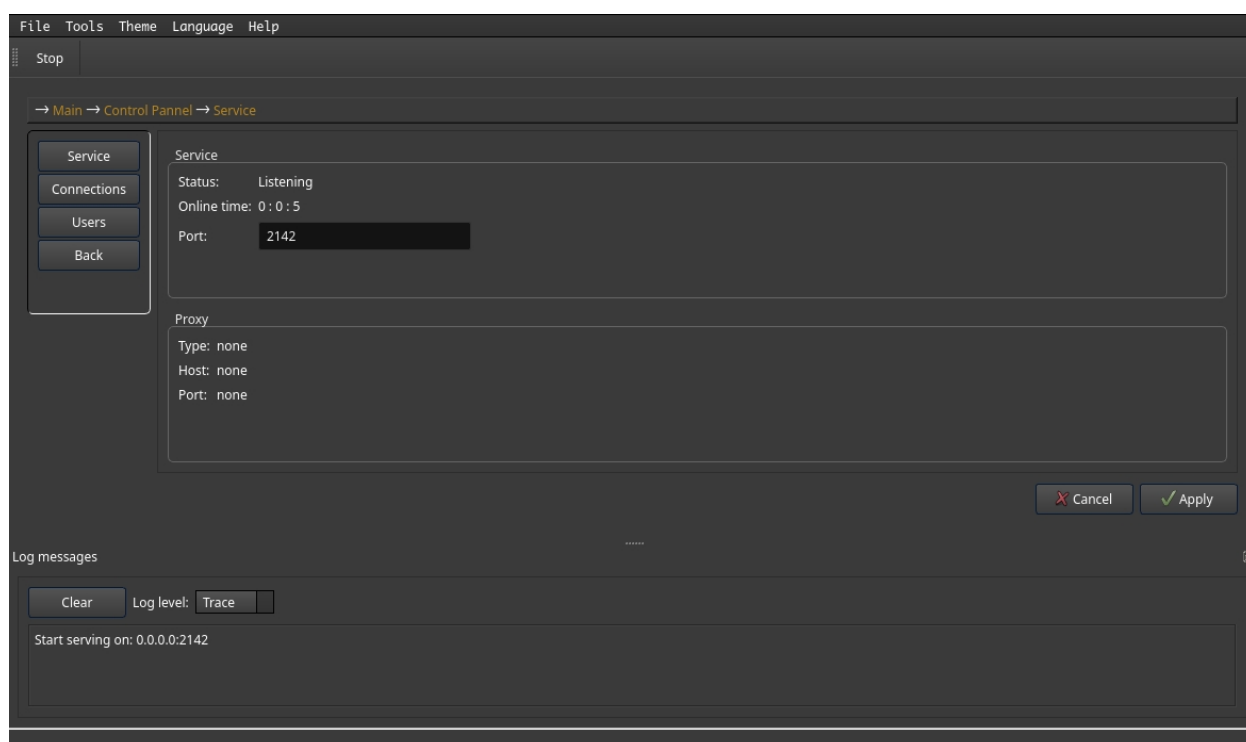


Рис. 2.10

Страница просмотра активных соединений с сервером(рис. 2.11):

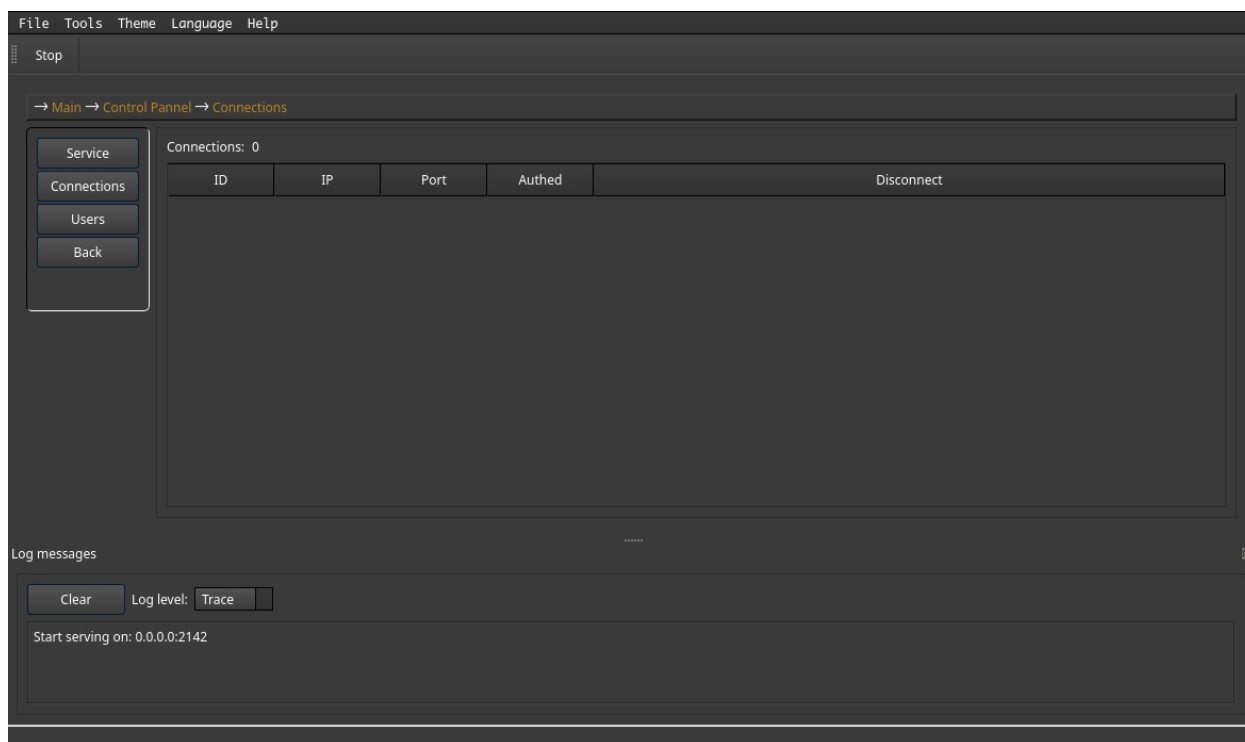


Рис. 2.11

2.6.2 Клиент

Начально окно (рис. 2.12):

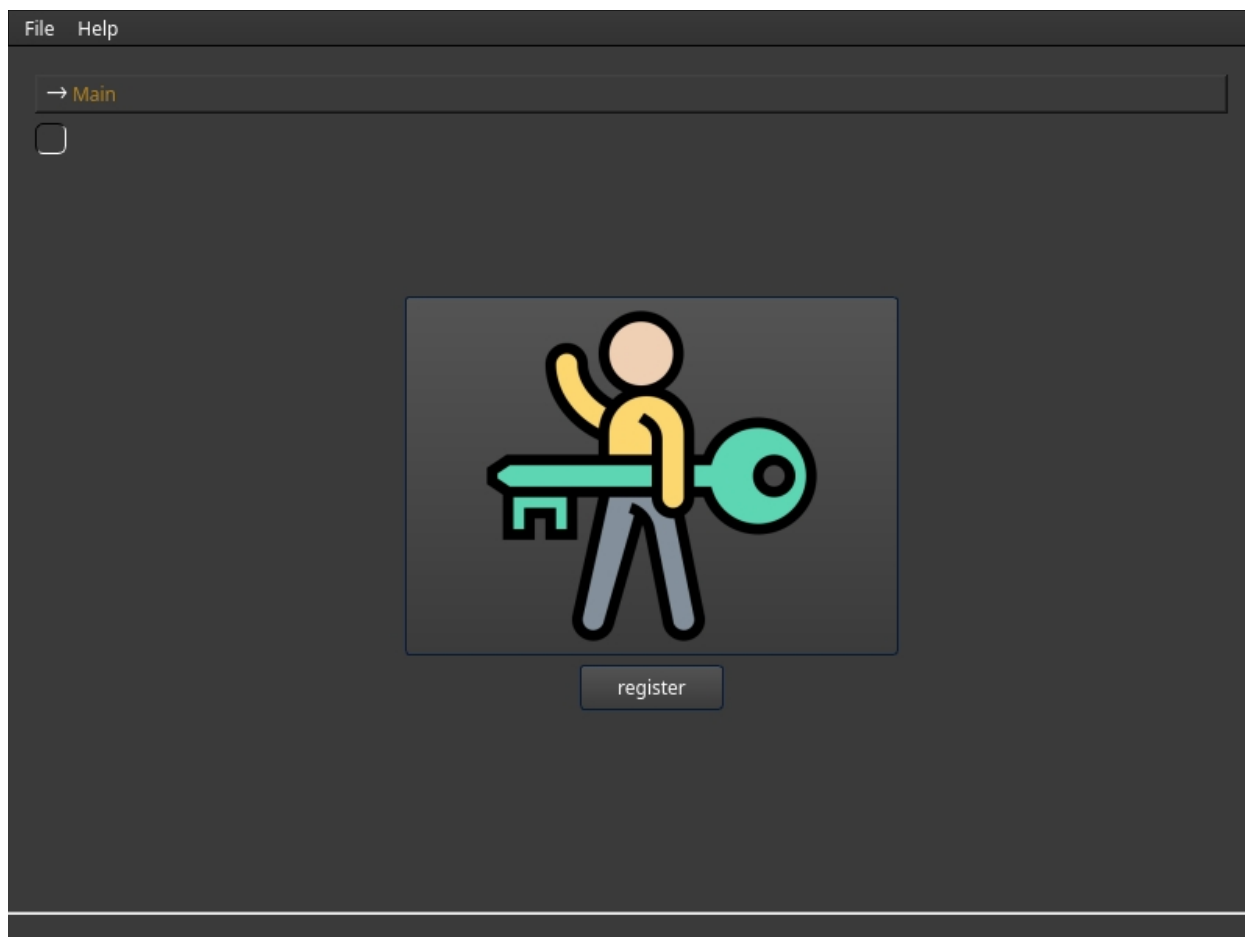


Рис. 2.12

Главная страница роли Admin , для администрации ЧОП(рис. 2.13):

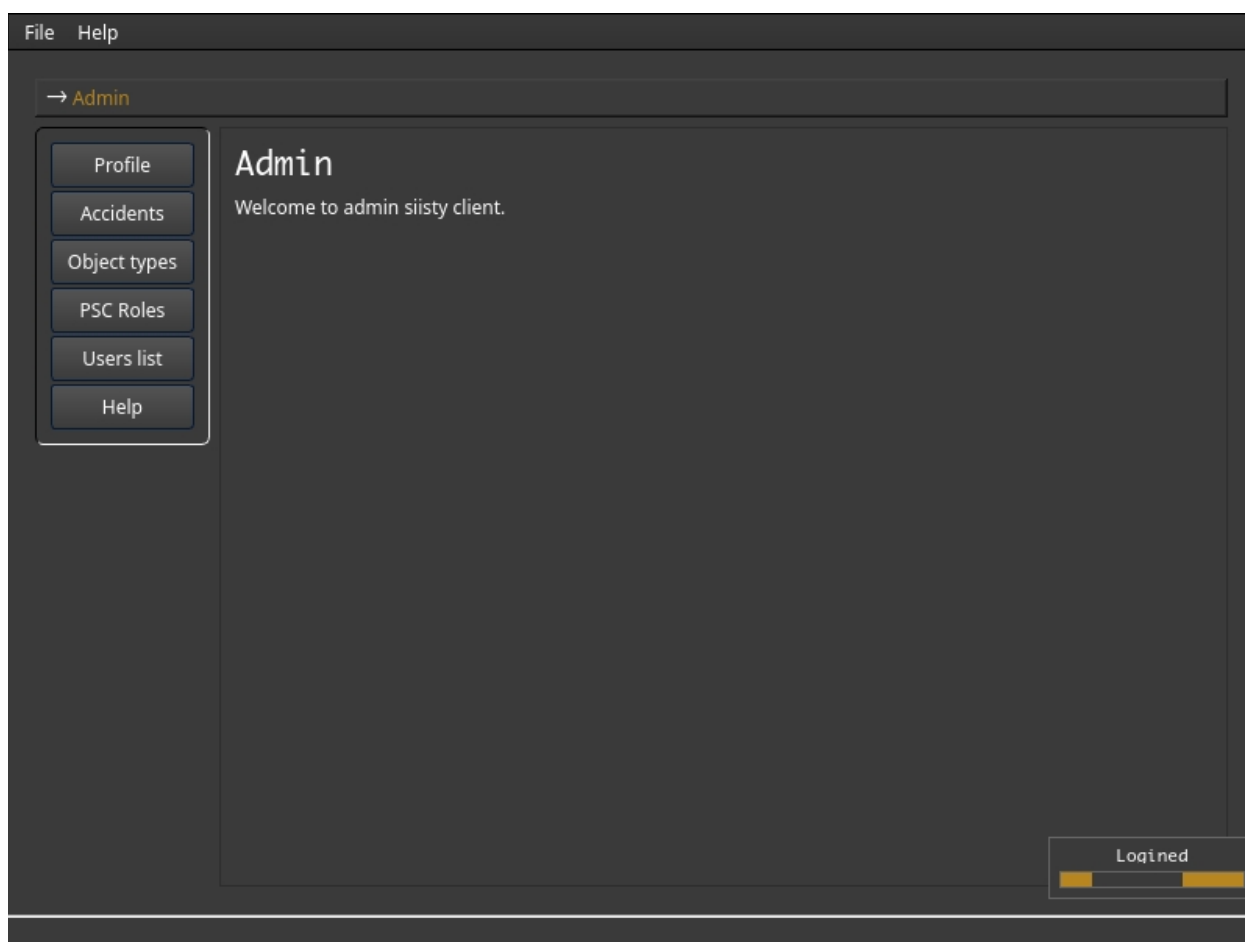


Рис. 2.13

Модальное окно входа в систему(рис. 2.14):

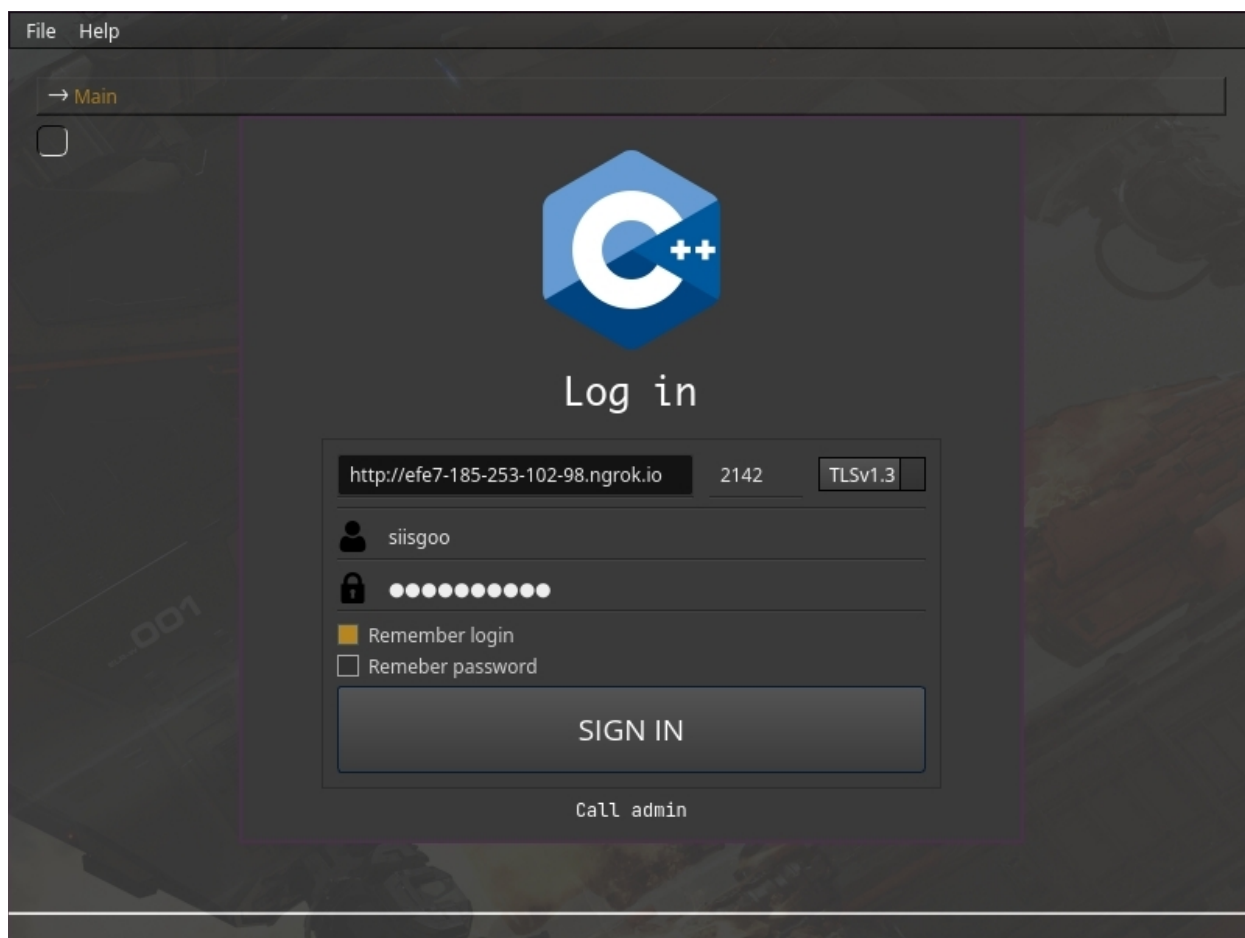
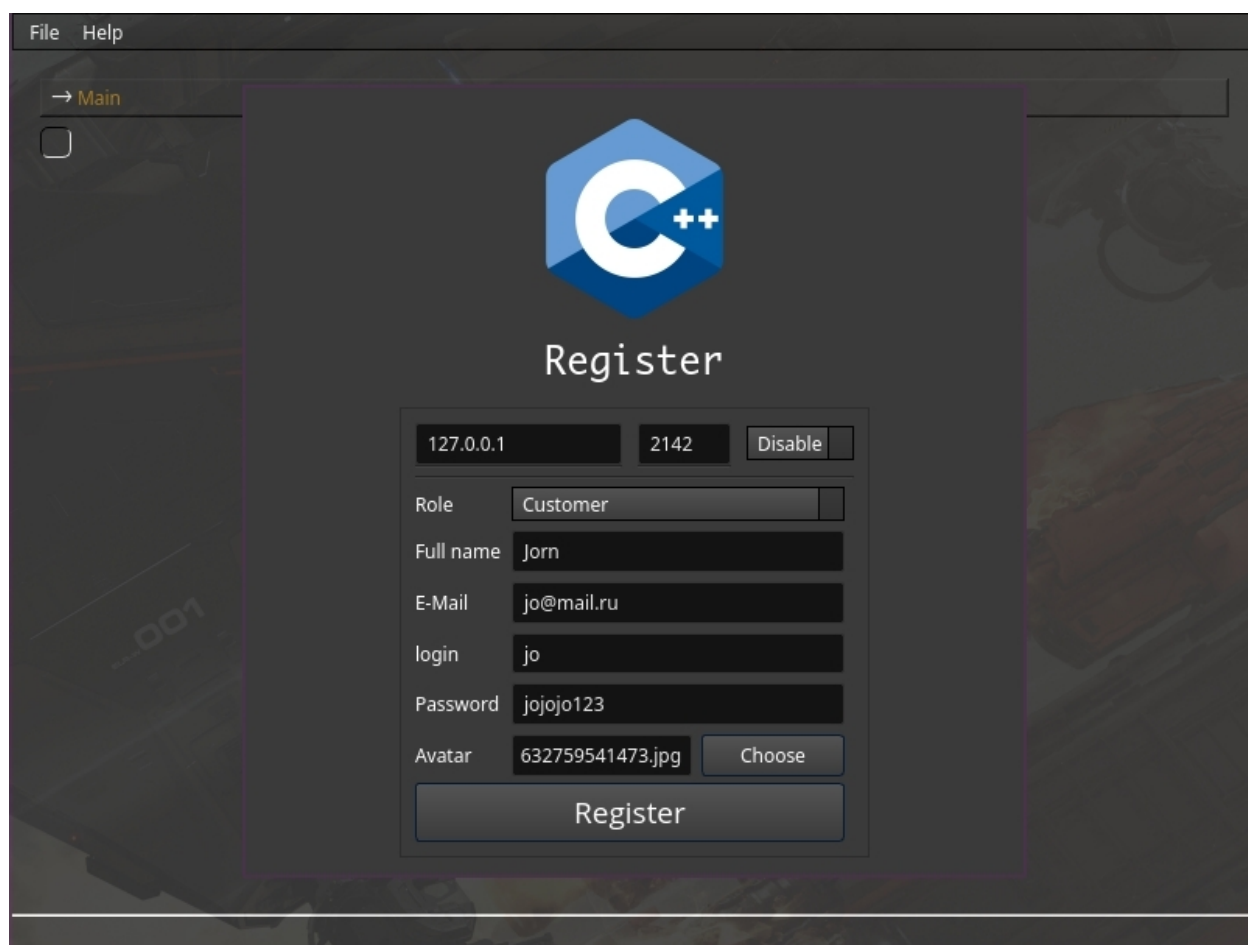


Рис. 2.14

Модальное окно регистрации (рис. 2.15):



The image shows a screenshot of a C++ IDE with a modal registration window open. The window has a dark theme and a blue C++ logo at the top. The registration form includes the following fields and controls:

- IP address: 127.0.0.1
- Port: 2142
- Disable button
- Role: Customer (dropdown menu)
- Full name: Jorn
- E-Mail: jo@mail.ru
- login: jo
- Password: jojojo123
- Avatar: 632759541473.jpg
- Choose button
- Register button

Рис. 2.15

Страница создания и просмотра контрактов клиентом(рис. 2.16):

File Help

→ Customer → Contracts

Profile
Accidents
Contracts
Help

Active contracts

	start date	expiration d	object	assigned employee
0	5/13/2022	10/1/2022	House	3
1	10/1/2022	12/1/2023	Office	8

Object type
Address
Way point
Start date 1/1/00 11:59 PM
Expiration 5/13/22 9:29 AM
Weekends -
Employee -

Add employee

Price: 0

Create new

Update

Рис. 2.16

Страница просмотра личных расходов и доходов(рис. 2.17):

File Help

→ Customer → Profile → Accounting

Accounting

Back

	Pay amount	Date	Objective
0	13400	4/10/2022	Contract
1	108000	4/20/2022	Contract
2	40241	4/21/2022	Accident

income 40241 outcome 121400

Рис. 2.17

Заключение

Таким образом, на базе ЧОП была разработана база данных, защищенная методами авторизации и разграничения доступа по ролям, также ограничили дистрибьюцию самого источника БД(файла), соответствующим должностям в ЧОП и клиент-серверное ПО для доступа к базе данных, точнее к, тщательно проанализированным, услугам и процессам ЧОП, его услугам и процессам. Клиентское ПО включает в себя функционал как для сотрудников, так и для внешних объектов, таких как рекруты и заказчики услуг ЧОП. Серверное ПО предлагает сервисы авторизованного доступа к услугам ЧОП. От утечек источника БД была защищена “соленым” хешированием данных для идентификации пользователей. Все клиент-серверные операции возможны как в формате подключения TCP с SSL/TLS, так и в незащищенном TCP в зависимости от конфигурации сервера. Проект написан на C++ с использованием фреймворка Qt6 и предназначен для использования в среде GNU/Linux, распространение ведется на <https://github.com/siisgoo/siisty> под лицензией MIT.

Список источников

1. Unix Network Programming: The Sockets Networking Api - Addison-Wesley Professional; Subsequent edition (November 1, 2022) - by W. Richard Stevens (Author), Bill Fenner (Author), Andrew M. Rudoff (Author)
2. Forge Your Future with Open Source: Build Your Skills. Build Your Network. Build the Future of Technology - VM (Vicky) Brasseur
3. The Art of Computer Programming, Volumes 1-4 – Donald E. Knuth
4. <https://wiki.qt.io> — документация Qt
5. <https://www.sqlite.org> – документация sqlite3
6. http://www.iaeng.org/IJCS/issues_v43/issue_1/IJCS_43_1_04.pdf