## Вступ

В якості основи архітектури було прийнято рішення обрати “клієнт-серверну” модель. На прийняття рішення вплинули такі вихідні умови:

1. Передбачено, що одне підприємство буде мати спільну базу даних для всіх робітників.
2. Ймовірно, що пристрої, на яких буде виконуватися робота, не будуть відповідати достатньо великим вимогам щодо розміру ОП та потужності процесора.

## 

## Основні класи

### Клієнт

#### Графічний інтерфейс

Кожен клас має [classname].ui файл з xml кодом для базових графічних Qt-компонент:

GUI\_classes.drawio

### 

#### UML діаграма класів клієнтської частини

Програма клієнт складається з класів **Mainwindow**, **TcpClient** та багатьох класів **віджетів** і **діалогів**. Кожен клас відіграє свою функціональну роль:

* ***MainWindow*** - відіграє роль центру керування та є головним графічним об’єктом, що буде вміщувати в собі елементи типу віджетів.
* ***TcpClient*** - відповідає за зв’язок з сервером. Є полем класу MainWindow та ініціалізується в його конструкторі. Реалізує базові публічні методи, через які інші класи будуть взаємодіяти з сервером.
* ***Віджети*** - вид класів, що будуть відігравати основну функціональну роль. Кожен віджет відведений під конкретну задачу і буде ініціалізуватися та встановлюватися у полі tabWidget.
* ***Діалоги*** - вид класів, що відповідають за модульні, “спливаючі” вікна. Зокрема, деякі серед них це вікна:
  + Авторизації,
  + Регістрації,
  + Завантаження накладних,
  + Додавання нових покупців,
  + Додавання нового товару.

UML\_Class\_diagram.drawio

### Сервер

#### UML діаграма класів серверної частини

* с

Server\_UML\_Class\_diagram.drawio

## Взаємодія клієнт-сервер

### Протокол “запит-відповідь”

За основу клієнт-серверної взаємодії було прийнято обрати найзручніший для розуміння протокол “Request-Response”. Тобто, на кожне відправлене повідомлення сервер повинен дати відповідь.

У більшості випадків від сервера буде вимагатися підтвердження отримання чи опрацювання інформації.

Основні види запитів та відповідей на них:

Requests&Responses

## Організація даних

### База даних (БД)

|  |  |
| --- | --- |
| **Назва таблиці** | **Опис** |
| Lists | Зберігає дані про всі “списки” (рахунки, накладні і т.д). |
| Records | Містить усі позиції усіх списків |
| ProductTypes | Містить список всіх можливих видів товарів. |
| Users | Зберігає дані про всіх користувачів програми. |
| UserLogs | Зберігає список діяльністостей усіх користувачів. |
| Customers | Зберігає дані про всіх покупців. |
| Cars | Кожен покупець може мати декілька авто. |
| Sellers | Зберігає дані про всіх продавців. |
| Storage | Містить список всіх присутніх на “складі” (у магазині) товарів. |

* Lists та Records відображають списки та їх записи відповідно. Принцип з’єднання “один-багато”.
* RecordTypes та Records - “один-багато”
* Users та UserLogs - “один-багато”
* Lists та Customers (для Sellers аналогічно) - “багато-один”. Один список може мати лише одного покупця (продавця).
* RecordTypes та Storage - “один-один”
* Customers та Cars - “один-багато”

Кожен список (з таблиці **Lists**), фактично, є документом, правила поведінки і види якого наведені у Бізнес-правила. Документи мають характеристику **ListType**, що фактично є enum виду, тобто приклад запису у БД: “3” - Документ виду “3” , що є “Податковою накладною”. **ListNumber** є унікальним (лише для поточного року) ідентифікатором документу, згідно Бізнес-правила про кожнорічне оновлення нумерації.

Database\_tables.drawio

## Бізнес-правила

1. Кожного року (01.01.хх 00:00) нумерація документів повинна починатися з початку. (напр. “Накладна №1”)
2. Види документів:

|  |  |
| --- | --- |
| **№** | **Вид документу** |
| 0 | Рахунок |
| 1 | Видаткова накладна |
| 2 | Товарний чек |
| 3 | Податкові накладні |
| 4 | Накладні на повернення |
| 5 | Накладна на надходження |

1. Розрізнення Sellers, як платник ПДВ і не платник.

Умова: платник ПДВ має IPN != “”

1. Розрізнення Customers, як організація і звичайний покупець.

Умова: організація має IPN != “”

1. До необхідних статистичних даних відносятся:
   1. Валовий дохід,
   2. Топ 10 покупців.

## 

## Користувацький інтерфейс

### Сервер

Для взаємодії з користувачем було вирішено обрати інтерфейс командного рядка (CLI), адже кінцевий користувач на буде яким небудь чином конфігурувати **програму-сервер** на пряму. Передбачено, що будь-які потрібні налаштування програми-сервера будуть здійснюватися за допомогою графічного інтерфейсу програми-клієнта.

Інформація, що буде виводитися **програмою-сервером** у stdout буде виключно робочою інформацією про:

* Запуск сервера,
* Адресу сервера у мережі,
* Інформація про нових підключених програм-клієнтів,
* Інформація про відключених програм-клієнтів,
* Помилки при роботі.

рис. Вигляд CLI програми-сервера

### 

### Клієнт

Функціональна специфікація, що потрібна кінцевому користувачу була визначена у “Requirements(ukr)”. Модель GUI буде базуватися на даній специфікації. Структура інтерфейсу:

GUI(ukr).drawio

Основним керуючим елементом буде QMenuBar, з якого користувач буде викликати основні модулі системи.

Кожен модуль - окремий QWidget, що запускається у QTabWidget, як вкладка.

рис. Вигляд користувацького інтерфейсу