МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ФАХОВИЙ КОЛЕДЖ РАКЕТНО-КОСМІЧНОГО МАШИНОБУДУВАННЯ

ДНІПРОВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ім. О. ГОНЧАРА

Циклова комісія програмної інженерії

**КУРСОВИЙ ПРОЕКТ**

з навчальної дисципліни

"ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ"

на тему: «Програма ведення обліку матеріальних цінностей»

(вказати тему курсового проекту)

Студента IV курсу  ПЗ-17-2  групи

напряму підготовки 6.050103

Програмна інженерія

спеціальності 121 Інженерія програмного забезпечення

Кабандже М.

(прізвище та ініціали студента)

Керівник   
викладач Гапоненко Н.В.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_ Оцінка ECTS: \_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_ Сітарчук В.В.

(підпис) (прізвище та ініціали) .

\_\_\_\_\_\_\_\_\_ Ланська С.С. . (підпис) (прізвище та ініціали) .

\_\_\_\_\_\_\_\_\_ Гапоненко Н.В. .

(підпис) (прізвище та ініціали) .

м. Дніпро - 2020 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ФАХОВИЙ КОЛЕДЖ РАКЕТНО-КОСМІЧНОГО МАШИНОБУДУВАННЯ

ДНІПРОВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ім. О. ГОНЧАРА

Цикловакомісія програмної інженерії

|  |  |
| --- | --- |
|  | **ЗАТВЕРДЖУЮ**  Голова комісії ПІ  \_\_\_\_\_\_\_\_\_\_\_\_ С.С.Ланська  "\_\_\_\_"\_\_\_\_\_\_\_\_\_\_\_ 2020 р. |

**ЗАВДАННЯ**

**на виконання курсового проекту**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| з дисципліни | | | Об’єктно-орієнтоване програмування | | | | | | | |
| студенту | Кабандже Микиті | | | | | | | | | |
|  | (прізвище, ім'я та по батькові) | | | | | | | | | |
| Відділення | | Комп'ютерної та програмної інженерії | | | | | | | | |
| Спеціальність | | | | 121 Інженерія програмного забезпечення | | | | | | |
| Курс | IV | | | | | |  | Група (шифр) | | ПЗ-17-2 |
| 1 Тема проекту | | | | | «Програма ведення обліку матеріальних цінностей» | | | | | |
|  | | | | | | | | | | |
|  | | | | | | | | | | |
| 2 Початкові дані | | | | | | Перелік складів, груп товарів, інформації про | | | | |
| контрагентів (ЄДРПО, назва, адреса, телефон, контактна особа), даних про | | | | | | | | | | |
| товар (група, назва), облікових даних (замовлення, склад, товар, кількість, | | | | | | | | | | |
| дата, приймання/постачання, покупець/постачальник), алгоритм ведення | | | | | | | | | | |
| обліку за складом, за період, форма звіту накладної | | | | | | | | | | |
| Розглянуто і ухвалено на засіданні циклової комісії | | | | | | | | | програмної інженерії | |
| Протокол № 2 від 16.09.2020 р. | | | | | | | | | | |

Керівник КП \_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В.Гапоненко

(підпис) (ініціали та прізвище)

Завдання до виконання

одержав студент \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (ініціали та прізвище)

Дата видачі 16 вересня 2020 р.

Термін виконання 19 листопада 2020 р.

ЗМІСТ

[ЗМІСТ 3](#_Toc56800095)

[ВСТУП 4](#_Toc56800096)

[1 ПОСТАНОВКА ЗАДАЧІ 5](#_Toc56800097)

[2 ХАРАКТЕРИСТИКА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ПРОЕКТУ 8](#_Toc56800098)

[2.4 Опис мови стилів CSS 12](#_Toc56800099)

[2.5 Опис основних принципів ООП 13](#_Toc56800100)

[2.6 Огляд існуючих рішень 15](#_Toc56800101)

[3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ 18](#_Toc56800102)

[3.1 Опис структури бази даних 18](#_Toc56800103)

[3.2 Проектування користувацького інтерфейсу програми 23](#_Toc56800104)

[4 ІНСТРУКЦІЯ З КОРИСТУВАННЯ ПРОГРАМНОЮ СИСТЕМОЮ 30](#_Toc56800105)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 38](#_Toc56800106)

[Додаток А 39](#_Toc56800107)

[Додаток Б 54](#_Toc56800108)

ВСТУП

Облік матеріальних цінностей - одна з багатьох завдань, яка вимагає уваги, великої кількості часу і людських ресурсів.

З тих чи інших причин, найчастіше, необхідні для виконання цієї роботи ресурси недоступні. Крім того при стандартному обліку цінностей крім людського фактору виникає проблема взаємодії з великими обсягами паперової звітності, яка може бути втрачена в ході проведення оцінки.

Метою даного курсового проекту є створення програмного забезпечення для автоматизації ведення обліку матеріальних цінностей, яке дозволить, якщо не позбутися від паперової роботи і людського фактору, то значно зменшить їх вплив на кінцевий результат.

1 ПОСТАНОВКА ЗАДАЧІ

Темою курсового проєкту є «Програма ведення обліку матеріальних цінностей».

Мета проєкту: реалізувати просту, ефективну програму для ведення обліку матеріальних цінностей.

Програма повинна:

1. Вести облік товарообігу на складах, формувати накладні на постачання, відпуск та переміщення товарів.
2. Зберігати інформації про постачальників, клієнтів, користувачів програмою, складів та товарів.
3. Надати можливість розмежування прав доступу для різних користувачів.
4. Надати можливість редагування та видалення існуючої інформації, окрім інформації про накладні.
5. Формувати звітність по товарам, постачання та відпуску товарів. Фінансову звітність.

Початкові дані містять:

1. Дані про товар:

* номенклатурний номер;
* назва;
* одиниці вимірювання;
* ціна;

1. Дані про склад:

* назва;
* місто;
* адреса;
* ємкість;

1. Дані про постачальника:

* назва;
* ПІБ представника;
* номер телефону;
* реквізити;

1. Дані про клієнта:

* назва;
* ПІБ представника;
* номер телефону;

1. Дані про користувача(співробітника):

* логін;
* пароль;
* ПІБ;
* номер телефону;
* посада;

1. Дані про посаду:

* назва;

1. Дані про накладну:

* сума;
* дата створення;
* тип;
* склад постачальник;
* склад приймач;
* відповідальний за відправку;
* постачальник;
* клієнт;
* тип сплати;
* примітки;
* сторнований запис чи ні;
* відповідальний за отримання;

Мінімальні вимоги до конфігурації ПК для користування цією програмою:

* центральний процесор Pentium IV або відповідний AMD;
* оперативна пам’ять 512 МБ;
* об’єм пам’яті жорсткого диску 256мб для клієнта і 5гб для сервера;
* операційні системи Windows 7, 8, 10 або Linux.

2 ХАРАКТЕРИСТИКА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ПРОЕКТУ

2.1 Опис середовища програмування

Програма курсового проекту написана в середовищі Qt Creator Community Edition.

 Qt Creator — кросплатформенний інструментарій розробки програмного забезпечення (ПЗ) мовою програмування C++. Дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем (ОС), просто компілюючи текст програми для кожної операційної системи без зміни серцевого коду. Містить всі основні класи, які можуть бути потрібні для розробки прикладного програмного забезпечення, починаючи з елементів графічного інтерфейсу й закінчуючи класами для роботи з мережею, базами даних, OpenGL, SVG і XML. Бібліотека дозволяє керувати потоками, працювати з мережею та забезпечує кросплатформенний доступ до файлів.

Qt Creator має вбудований редактор форм, що дає змогу власноруч прописувати стиль форми можливостями мови CSS. Підтримує CMake, що дає змогу писати кросплатформенні додатки. З відомих додатків Qt використовує Google Earth, завдяки зручності використання цієї IDE .

Була обрана за можливість розробляти кросплатформенні проекти, легке перенесення проекту з однієї ОС у іншу, можливість написання мобільних додатків мовою С++, широкий вибір інструментів для розробки програмного забезпечення та зручний інтерфейс.

2.2 Опис мови програмування

Програма курсового проекту написана мовою програмування С++. Текст програми знаходиться у додатку Д.

C++, яка спочатку проектувалася, як розширення для С, а також на Java та C#, які запозичили у С синтаксис.

Мова програмування C++ найбільш поширена серед розробників програмного забезпечення. Вона є дуже зручною у розробці прикладних програм; драйверів пристроїв; розробка ОС; відео ігор. Реалізацією мови С++ займаються одночасно декілька проектів як безкоштовних, так і комерційних, а саме: GNU, Microsoft і Embarcadero (Borland).

C++ －високорівнева мова програмування загального призначення, поєднує властивості як високорівневих, так і низькорівневих мов програмування. Здатна до підтримки об'єктно-орієнтованого, узагальненого та процедурного програмування. Від свого попередника, мову програмування С, С++ відрізняється тим, що найбільшу увагу при розробці цієї мови було приділено підтримці об'єктно-орієнтованого та узагальненого програмування. Назва «мова програмування C++» походить від мови програмування C, в якому унарний оператор ++ позначає інкремент змінної.

На початку 1980-х років мову програмування С++ розробив співробітник фірми Bell Laboratories － Бьерн Страуструп. Створення мови С++ не планувалось, для власних потреб Бьерн розробив ряд удосконалень мови програмування С. Ранні версії мови С++, називались «Cі з класами» та почали з'являтися з 1980 року. Страуструп додав до мови програмування С можливість роботи з класами та об'єктами, поклавши фундамент нової, заснованої на синтаксисі С, мови програмування. Синтаксис C++ був заснований на синтаксисі C, так як Бьерн Страуструп прагнув зберегти сумісність з мовою C.

Нововведеннями С++ порівняно з С є:

* простий і практичний інтерфейс;
* підтримка об'єктно-орієнтованого програмування через класи;
* підтримка узагальненого програмування через шаблони;
* доповнення до стандартної бібліотеки;
* додаткові типи даних;
* обробка винятків;
* простори імен;
* вбудовані функції;
* перевантаження імен функцій;
* посилання і оператори управління вільно розподіленою пам'яттю.

Переваги С++:

* 1. Швидкість роботи програм на С++ майже не змінилась порівняно із С, хоча розробники отримали нові можливості і засоби.
  2. Мова С++ є кросплатформенною.
  3. Можливість роботи на низькому рівні з пам'яттю, адресами, портами.
  4. Можливість створення узагальнених алгоритмів для різних типів даних, їхня спеціалізація і обчислення на етапі компіляції з використанням шаблонів.
  5. Підтримуються різні стилі та технології програмування, включаючи традиційне директивне програмування, ООП, узагальнене програмування, мета програмування (шаблони, макроси).

Недоліки С++:

* погана підтримка модульності. Підключення інтерфейсу зовнішнього модуля через процесорну вставку заголовного файлу (#include) серйозно уповільнює компіляцію при підключенні великої кількості модулів. Для усунення цього недоліку багато компіляторів реалізують механізм компіляції заголовних файлів;
* недостача інформації про типи даних під час компіляції;
* мова C++ є складною для вивчення і для компіляції;
* деякі перетворення типів не інтуїтивні. Зокрема, операція над беззнаковим і знаковим числами видає беззнаковий результат.

2.3 Опис СУБД MySQL

MySQL — це система управління базами даних.

База даних являє собою структуровану сукупність даних. Ці дані можуть бути будь-якими — від простого списку майбутніх покупок до переліку експонатів картинної галереї або величезної кількості інформації в корпоративній мережі. Для запису, вибірки й обробки даних, що зберігаються в комп'ютерній базі даних, необхідна система управління базою даних, якою і є ПО MySQL. Оскільки комп'ютери чудово справляються з обробкою великих обсягів даних, управління базами даних відіграє центральну роль в обчисленнях. Реалізовано таке управління може бути по-різному - як у вигляді окремих утиліт, так і у вигляді коду, що входить до складу інших додатків.

MySQL — це система управління реляційними базами даних.

В реляційній базі даних дані зберігаються не всі скопом, а в окремих таблицях, завдяки чому досягається виграш в швидкості і гнучкості. Таблиці зв'язуються між собою за допомогою відносин, завдяки чому забезпечується можливість об'єднувати при виконанні запиту дані з декількох таблиць. SQL як частина системи MySQL можна охарактеризувати як мову структурованих запитів й одну з найвикористовуваних мов для доступу до баз даних.

Програмне забезпечення MySQL — це ПЗ з відкритим кодом.

ПЗ з відкритим кодом означає, що застосовувати і модифікувати його може будь-хто. Таке ПЗ можна отримувати по Internet і використовувати безкоштовно. При цьому кожен користувач може вивчити вихідний код і змінити його відповідно до своїх потреб. Використання програмного забезпечення MySQL регламентується ліцензією GPL (GNU General Public License), в якій зазначено, що можна і чого не можна робити з цим програмним забезпеченням в різних ситуаціях. Якщо робота у рамках GPL вас не влаштовує або планується вбудовування MySQL-коду в комерційний додаток, є можливість купити комерційну ліцензовану версію у компанії MySQL AB. See section 1.6.3 ліцензії на ПЗ MySQL.

MySQL є дуже швидким, надійним і легким у використанні. MySQL має також ряд зручних можливостей, розроблених в тісному контакті з користувачами. Спочатку сервер MySQL розроблявся для управління великими базами даних з метою забезпечити більш високу швидкість роботи в порівнянні з існуючими на той момент аналогами. І ось вже протягом декількох років даний сервер успішно використовується в умовах промислової експлуатації з високими вимогами. Незважаючи на те що MySQL постійно вдосконалюється, він вже сьогодні забезпечує широкий спектр корисних функцій. Завдяки своїй доступності, швидкості і безпеки MySQL дуже добре підходить для доступу до баз даних по Internet.

Технічні можливості СУБД MySQL.

MySQL є системою клієнт-сервер, яка містить багатопоточний SQL-сервер, що забезпечує підтримку різних обчислювальних машин баз даних, а також кілька різних клієнтських програм і бібліотек, засоби адміністрування і широкий спектр програмних інтерфейсів (API). Також поставляється у вигляді багатопоточної бібліотеки, яку можна підключити до користувача додатком і отримати компактний, швидкий і легкий в управлінні продукт.

2.4 Опис мови стилів CSS

CSS — спеціальна мова, що використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних.

Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів.

Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

CSS має різні рівні та профілі. Наступний рівень CSS створюється на основі попередніх, додаючи нову функціональність або розширюючи вже наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі — сукупність правил CSS одного або більше рівнів, створені для окремих типів пристроїв або інтерфейсів. Наприклад, існують профілі CSS для принтерів, мобільних пристроїв тощо.

CSS (каскадна або блочна верстка) прийшла на заміну табличній верстці веб-сторінок. Головна перевага блочної верстки — розділення змісту сторінки (даних) та їхньої візуальної презентації.

CSS використовується авторами та відвідувачами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних переваг — можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML, XML або подібна мова розмітки) від вигляду документу (що описується в CSS).

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори тощо. CSS також дозволяє адаптувати контент до різних умов відображення (на екрані монітора, мобільного пристрою (КПК), у роздрукованому вигляді, на екрані телевізора, пристроях з підтримкою шрифту Брайля або голосових браузерах та ін.).

Один і той самий HTML або XML документ може бути відображений по-різному залежно від використаного CSS.

2.5 Опис основних принципів ООП

Об'єктно-орієнтоване програмування або ООП (object — oriented programming) — методологія програмування, заснована на представленні програми у вигляді сукупності об'єктів кожен з яких являється реалізацією визначеного типу, що використовує механізм пересилки повідомлень і класи організовані в ієрархію наслідування.

Центральний елемент ООП — абстракція. Дані за допомогою абстракції перетворяться в об'єкти, а послідовність обробки цих даних перетворюється на набір повідомлень, що передаються між цими об'єктами. Кожен з об'єктів має свою власну унікальну поведінку. До об'єктів можна звертатися як до окремих сутностей, які реагують на повідомлення, що наказують їм виконати якісь дії.

ООП характеризується наступними принципами (по Алану Кею) :

1. Усе являється об'єктом;
2. Обчислення здійснюються шляхом взаємодії (обміну даними) між об'єктами, при якому один об'єкт вимагає, щоб інший об'єкт виконав деяку дію; об'єкти взаємодіють, посилаючи і отримуючи повідомлення; повідомлення - це запит на виконання дії, доповнений набором аргументів, які можуть знадобитися при виконанні дії;
3. Кожен об'єкт має незалежну пам'ять яка складається з інших об'єктів
4. Кожен об'єкт є представником класуякий виражає загальні властивості об'єктів даного типу
5. У класі задається функціональність (поведінка об'єкту); тим самим усе об'єкти, які являються екземплярами одного класу, можуть виконувати одні і ті ж дії;
6. Класи організовані в єдину деревовидну структуру із загальним коренем, звану ієрархією спадкоємства; пам'ять і поведінка, пов'язана з екземплярами певного класу, автоматично доступні будь-якому класу, розташованому нижче в ієрархічному дереві.

Абстрагування (abstraction) — метод рішення задачі, при якому об'єкти різного роду об'єднуються загальним поняттям (концепцією), а потім згруповані суті розглядаються як елементи єдиної категорії. Абстрагування дозволяє відокремити логічний сенс фрагмента програми від проблеми його реалізації, розділивши зовнішній опис (інтерфейс) об'єкту і його внутрішню організацію (реалізацію).

Інкапсуляція (encapsulation) — техніка, при якій несуттєва з точки зору інтерфейсу об'єкту інформація ховається усередині нього.

Спадкоємство (inheritance) — властивість об'єктів, за допомогою якої екземпляри класу дістають доступ до даних і методів класів-предків без їх повторного визначення. Спадкоємство дозволяє різним типам даних спільно використовувати один і той же код, призводячи до зменшення його розміру і підвищення функціональності.

Поліморфізм (polymorphism) —- властивість, що дозволяє використовувати один і той же інтерфейс для різних дій; поліморфній змінній, наприклад, може відповідати декілька різних методів.

Поліморфізм перекроює загальний код, що реалізовує деякий інтерфейс, так, щоб задовольнити конкретним особливостям окремих типів даних. Клас (class) - безліч об'єктів, пов'язаних спільністю структури і поведінка; абстрактний опис даних і поведінки (методів) для сукупності схожих об'єктів, представники якої називаються екземплярами класу.

Об'єкт (object) — конкретна реалізація класу, що володіє характеристиками стану, поведінки і індивідуальності, синонім екземпляра.

2.6 Огляд існуючих рішень

У зв’язку з розробкою курсового проекту було прийнято рішення аналізувати існуючі програмні реалізації для співставлення перспектив проекту та для кращого розуміння того, які можливості та функціонал повинен бути втілений.

Основною програмою як аналог і приклад стала програма «СуперСклад» для ведення складського обліку. Головне вікно зображено на рисунку 2.1

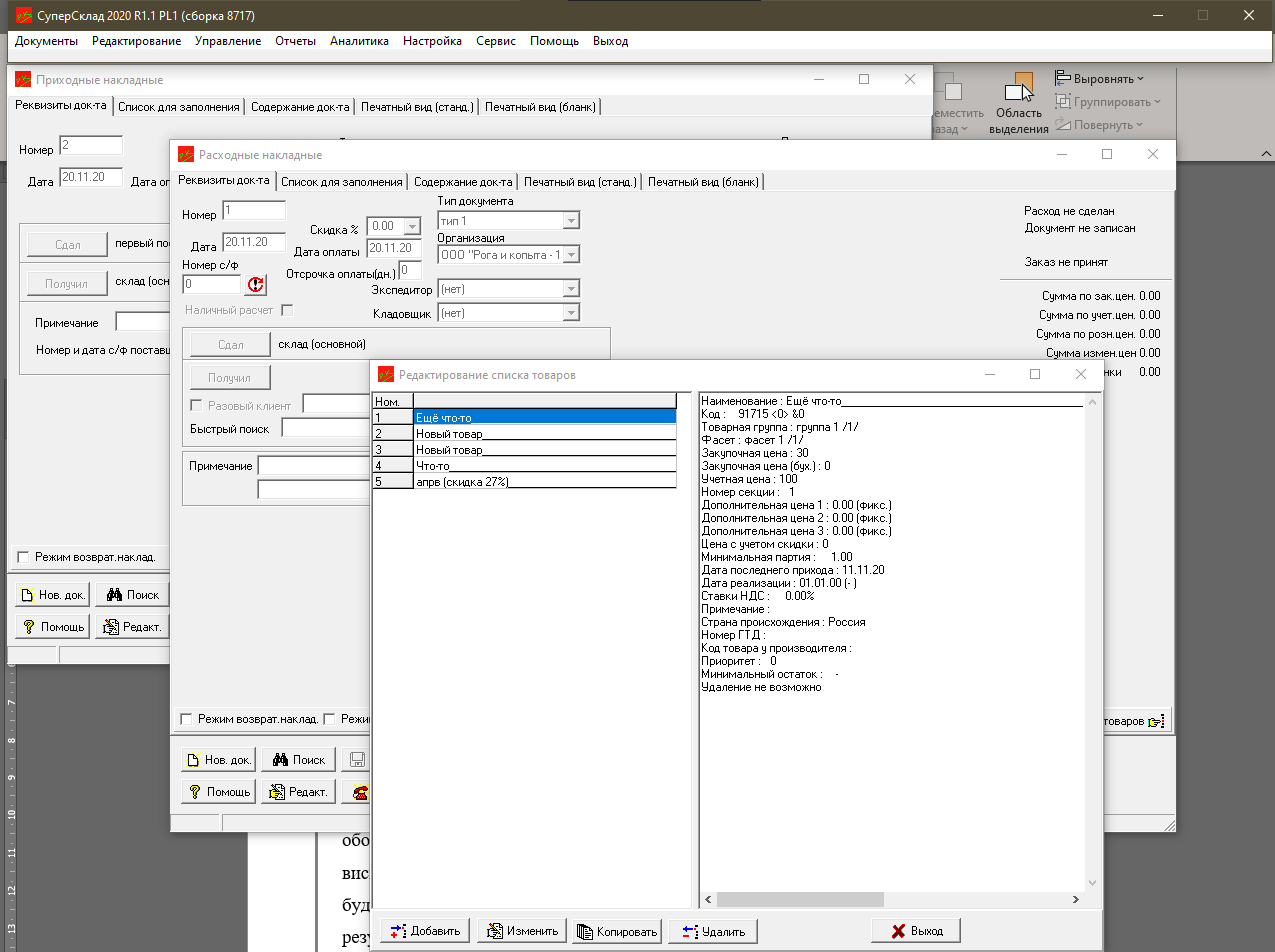


Рисунок 2.1 – Головне вікно прогами «СуперСклад»

Проста програма склад дозволяє вести повноцінний облік товарів і грошей від кіоску до великої оптової бази. Товари можна враховувати на 100 різних складах, вести взаєморозрахунки з необмеженною кількістю покупців і постачальників, виводити на друк всі необхідні первинні документи (накладна ТОРГ-12, рахунок-фактура, накладна Т-1 і інші стандартні форми). Крім того є вбудований генератор первинних документів, який дозволяє підготовленим користувачам створювати свої форми первинних документів.

Однією з основних переваг цієї програми є гнучка база даних, яка дає можливість використовувати базу даних у великій кількості галузей з необхідністю складського обліку, оскільки користувач сам обирає, яка інформація про кожний об’єкт його цікавить. Як приклад можна взяти вікно редагування інформації про товар, де кількість полей покриває потреби більшості користувачів, але їх заповнення не обов’язкове. Вікно редактора представлено на рисунку 2.2.

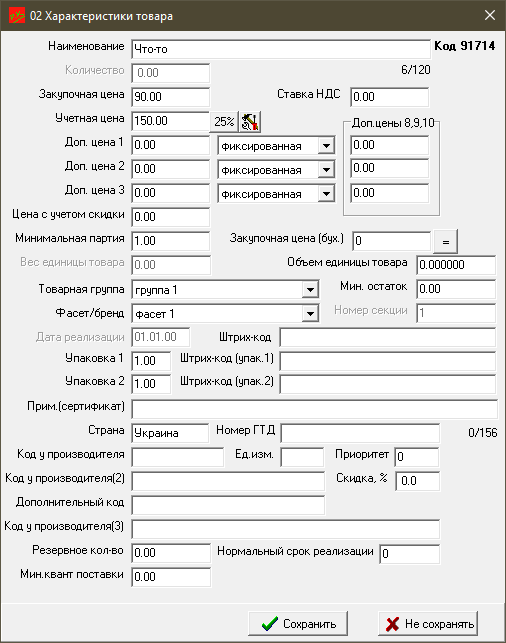


Рисунок 2.2 – Вікно редактора товару

Але ознайомившись краще з розробкою, інтерфейс виявився недостатньо інтуітивно-зрозумілим для посереднього користувача, та через намагання покрити потреби більшості користувачів — перевантаженим. Оцінивши недоліки цієї розробки, було вирішено написати програму за меншим набором функцій та простішим інтерфейсом. Оскільки, наприклад, малому та середньому бізнесу представлених функцій «СуперСклад» може виявитися забагато, то найбільш комфортним рішенням стане обрання моєї розробки.

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

## 

3.1 Опис структури бази даних

Зважаючи на необхідність зберігати великі об’єми інформації, під час проектування доцільно розділити базу даних на декілька сутностей (таблиць).

База даних складається 13-ти таблиць: clients — таблиця клієнтів, invoices — таблиця накладних, payment\_type — таблиця способу оплати, position — таблиця посад, product — таблиця товарів, product\_in\_invoice —проміжна таблиця, product\_in\_storage — проміжна таблиця, providers — таблиця постачальників, storage — таблиця складів, storno — таблиця сторнованих накладних, type\_invoice — таблиця типів накладних, user\_storage — проміжна таблиця, users — таблиця користувачів. ER-діаграма бази даних представлена на рисунку 3.1, перелік таблиць, їх поля та призначення полів наведено у таблицях 3.1 — 3.13.

Таблиця 3.1 – Таблиця clients — таблиця клієнтів

|  |  |
| --- | --- |
| Поле | Призначення |
| id | Ідентифікатор клієнта |
| PIB | ПІБ представника |
| phonenumber | Номер телефону |
| title | Назва клеєнта |

Таблиця 3.2 – Таблиця providers — таблиця постачальників

|  |  |
| --- | --- |
| Поле | Призначення |
| id | Ідентифікатор постачальника |
| title | Назва постачальника |
| PIB | ПІБ представника |
| phonenumber | Номер телефону |
| requisites | Реквизити |

Таблиця 3.3 – Таблиця invoices — таблиця накладних

|  |  |
| --- | --- |
| Поле | Призначення |
| id | Ідентифікатор накладної |
| sum | Сума вартості товару |
| date | Дата створення |
| idTypeInvoice | Ідентифікатор типу накладної |
| idStorageSender | Ідентифікатор складу відправника |
| idStorageRecipient | Ідентифікатор складу отримувача |
| idUserSender | Ідентифікатор користувача відправника |
| idProvider | Ідентифікатор постачальника |
| idClient | Ідентифікатор клієнта |
| idPayMentType | Ідентифікатор типу оплати |
| Note | Примітки |
| Storno | Показник сторнованості накладної |
| idUserRecipient | Ідентифікатор користувача отримувача |

Таблиця 3.4 – Таблиця paymenttype — таблиця типу оплати

|  |  |
| --- | --- |
| Поле | Призначення |
| id | Ідентифікатор типу |
| title | Назва типу |

Таблиця 3.5 – Таблиця position — таблиця посад

|  |  |
| --- | --- |
| Поле | Призначення |
| id | Ідентифікатор посади |
| title | Назва посади |

Таблиця 3.6 – Таблиця product — таблиця товарів

|  |  |
| --- | --- |
| Поле | Призначення |
| id | Ідентифікатор товару |
| title | Назва товару |
| measuring | Одиниця вимірювання товару |
| nomenclature | Номенклатура |
| price | Ціна |

Таблиця 3.7 – Таблиця product\_in\_invoice — проміжна таблиця товарів у накладній

|  |  |
| --- | --- |
| Поле | Призначення |
| idInvoice | Ідентифікатор накладної |
| idProduct | Ідентифікатор товару |
| price | Ціна закупівлі |
| quantity | Кількість товару |
| accounting\_price | Ціна продажу на момент поставки |

Таблиця 3.8 – Таблиця product\_in\_storage — проміжна таблиця товарів на складі

|  |  |
| --- | --- |
| Поле | Призначення |
| idProduct | Ідентифікатор товару |
| idStorage | Ідентифікатор складу |
| quantity | Кількість |

Таблиця 3.9 – Таблиця storno — таблиця сторнованих накладних

|  |  |
| --- | --- |
| Поле | Призначення |
| idInvoice | Ідентифікатор накладної |
| note | Примітки |

Таблиця 3.10 – Таблиця storage — таблиця складів

|  |  |
| --- | --- |
| Поле | Призначення |
| id | Ідентифікатор складу |
| title | Назва |
| city | Місто розташування |
| address | Адреса |
| capacity | Ємність |

Таблиця 3.11 – Таблиця type\_invoice — таблиця типів накладних

|  |  |
| --- | --- |
| Поле | Призначення |
| id | Ідентифікатор типу |
| title | Назва |

Таблиця 3.12 – Таблиця type\_invoice — проміжна таблиця доступних для користувача складів

|  |  |
| --- | --- |
| Поле | Призначення |
| idUser | Ідентифікатор користувача |
| idStorage | Ідентифікатор складу |

Таблиця 3.13 – Таблиця users — таблиця користувачів

|  |  |
| --- | --- |
| Поле | Призначення |
| id | Ідентифікатор користувача |
| login | Логін |
| password | Пароль |
| PIB | ПІБ користувача |
| phonenumber | номер телефону |
| idPosition | Ідентифікатор посади |

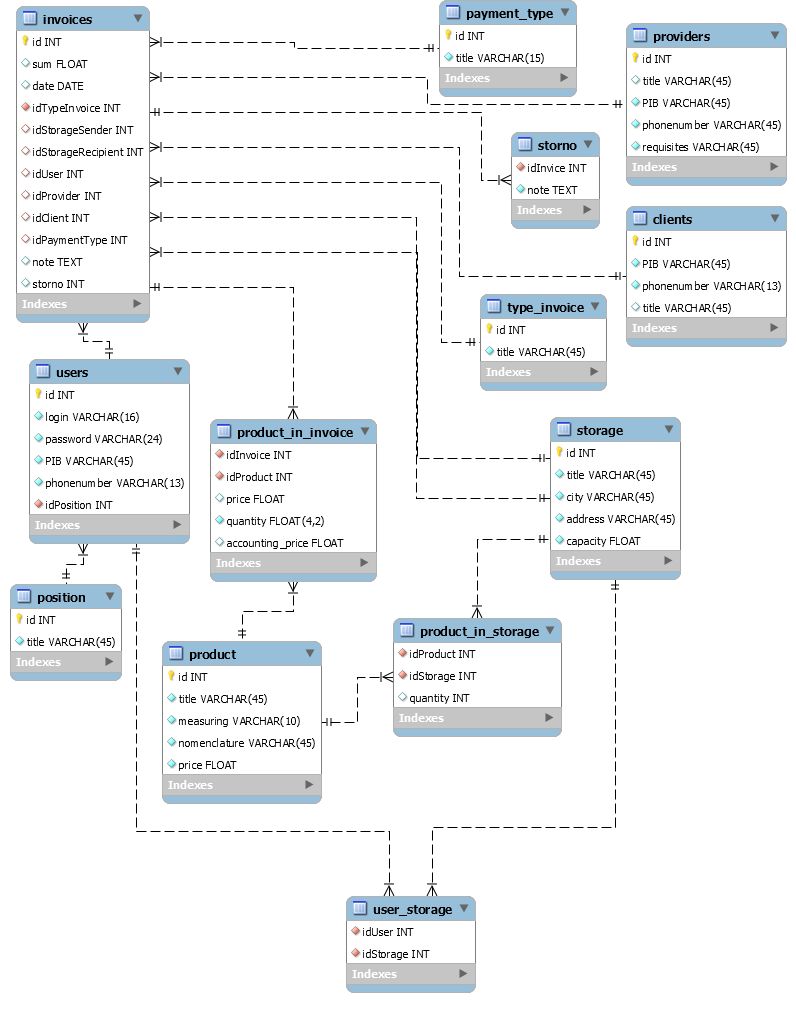


Рисунок 3.1 — ER-діаграма бази даних

3.2 Проектування користувацького інтерфейсу програми

Задачею графічного інтерфейсу програми є забезпечення комфортної взаємодії користувача з програмним додатком. Графічний інтерфейс складається з форм та елементів, які розташовуються на формах. Розробка інтерфейсу програми починається зі створення нової, стартової форми, шляхом внесення змін до властивостей автоматично згенерованої форми (Form1), а також шляхом додавання на форму нових необхідних елементів. На форму можуть бути додані різноманітні компоненти для взаємодії з текстом, числовими даними, базами даних тощо. Для перегляду і зміни значень властивостей форми і її компонентів використовується вікно Object Inspector, у якому знаходиться більшість даних про об’єкт, який є активним на даний момент (ім’я об’єкту, шрифт, ширина, висота тощо). Під час проектування дуже важливо цілеспрямовано обрати компоненти.

Інтерфейс програми складається з основної форми, яка має у собі усі необхідні елементи для взаємодії користувача із інформацією, та доступу до основних функцій, та певної кількості допоміжних форм, які реалізовують окремі функції програми, наприклад додавання або редагування інформації, та створені для більш наглядного розмежування дій та внесення простоти та ясності в користувацький інтерфейс.

Інтерфейс програми налічує такі форми:

* MainWindows — головне вікно програми в якому розміщені елементи для перегляду інформації, кнопки керування та елементи налаштування певних параметрів;
* Authorization\_Dialog — вікно для входу користувача до свого облікового запису, налічує поле для вводу логіну та паролю;
* AddInvoice\_Dialog — вікно для внесення інформації для свторюваної накладної;
* AddOrSetClient\_Dialog — вікно для внесення або редагування інформації о клієнті;
* AddOrSetProduct\_Dialog — вікно для внесення або редагування інформації о товарі;
* AddOrSetProductionInInvoice\_Dialog — вікно для внесення або редагування інформації о товарі, що додається у накладну;
* AddOrSetProvider\_Dialog — вікно для внесення або редагування інформації о постачальника;
* AddOrSetStorage\_Dialog — вікно для внесення або редагування інформації о складі;
* AddOrSetUser\_Dialog — вікно для внесення або редагування інформації о користувачеві;
* ChooseProduct\_Dialog — вікно для вибору товару зі списка;
* MoreAboutInvoice\_Dialog — вікно для перегляду детальної інформації про накладну;
* PrintPreview\_Dialog — вікно для перегляду інформації перед експортом у документ.

Форми зображені на рисунках 3.1 — 3.12

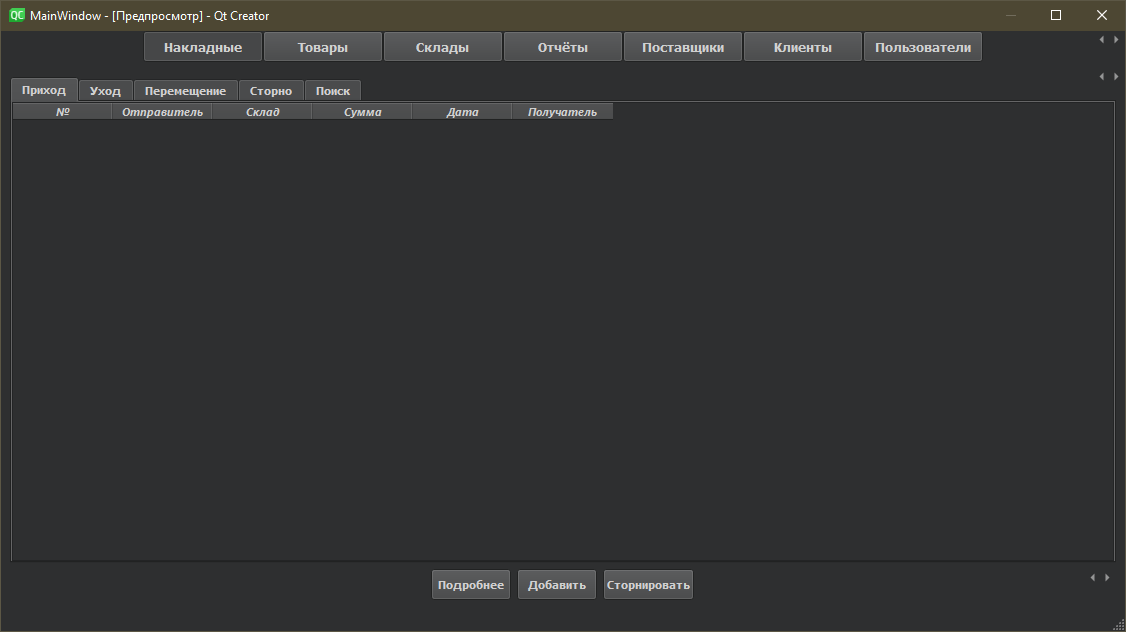


Рисунок 3.1 – Головне вікно прогами MainWindow

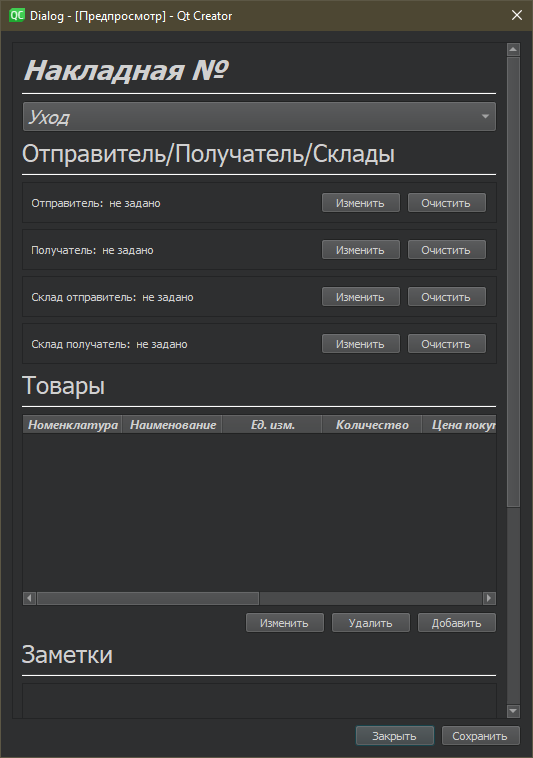


Рисунок 3.2 – Вікно AddInvoice\_Dialog

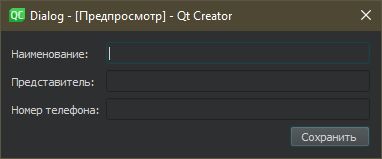


Рисунок 3.3 – Вікно AddOrSetClient\_Dialog

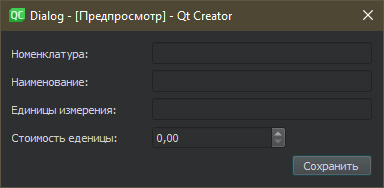


Рисунок 3.4 – Вікно AddOrSetProduct\_Dialog

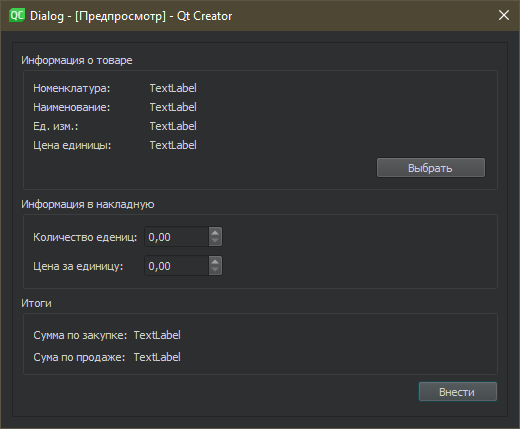


Рисунок 3.5 – Вікно AddOrSetProductionInInvoice\_Dialog

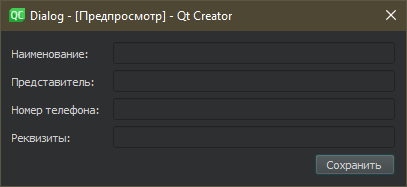


Рисунок 3.6 – Вікно AddOrSetProvider\_Dialog

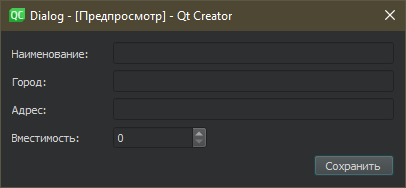


Рисунок 3.7 – Вікно AddOrSetStorage\_Dialog

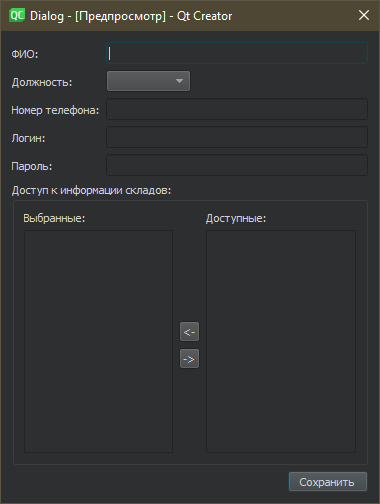


Рисунок 3.8 – Вікно AddOrSetUser\_Dialog

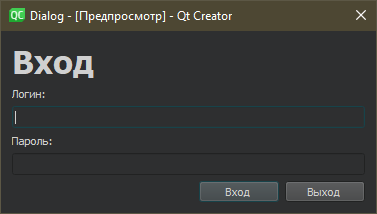


Рисунок 3.9 – Вікно Authorization\_Dialog

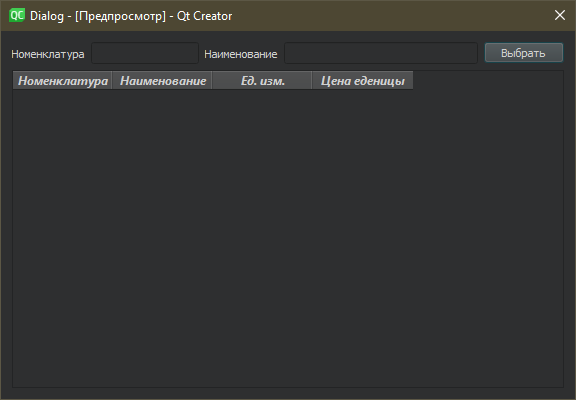


Рисунок 3.10 – Вікно ChooseProduct\_Dialog

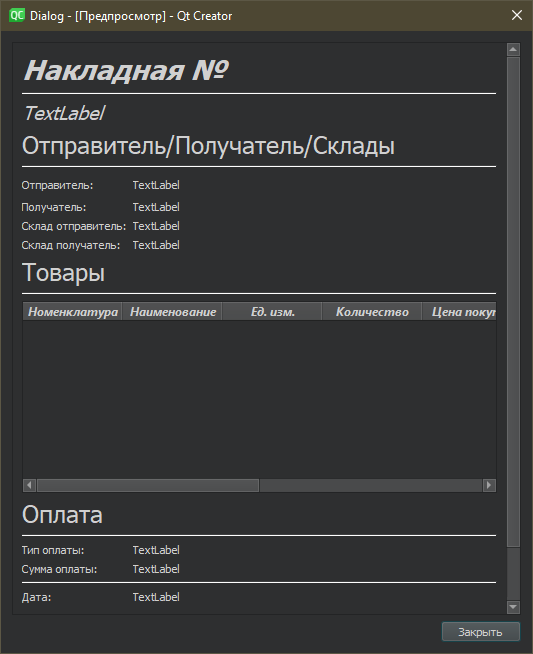


Рисунок 3.11 – Вікно MoreAboutInvoice\_Dialog

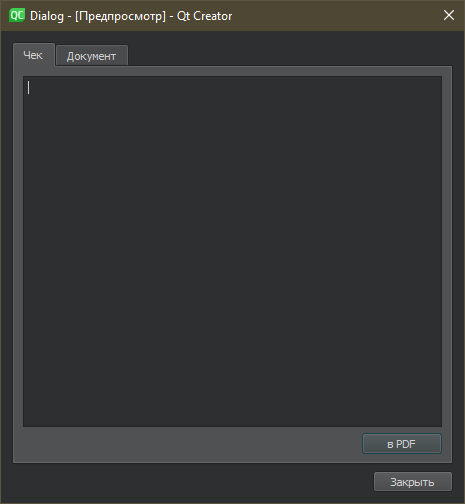


Рисунок 3.12 – Вікно PrintPreview\_Dialog

ВИСНОВОК

Під час роботи над курсовим проектом «Програма ведення обліку матеріальних цінностей» засобами середовища Qt Creator було створено програму, яка дозволяє зручно вести облік товару користувачам, для яких великий функціонал програм аналогів не потрібен, а лише заважає у повсякденному користуванні.

Програма виконує усі поставлені перед нею задачі, зберігає інформацію по кожній накладній, дозволяє вести облік користувачів, клієнтів, постачальників, товарів, надає можливість вести облік товарів на декількох складах. Дозволяє розмежувати доступ до інформації як по посаді користувача, так і по складам. Також формує необхідну звітність по постачанню і відпуску, по товарам, складам і фінансову звітність.

Розроблена програмна оболонка має інтуїтивно-зрозумілий графічний інтерфейс та приємне кольорове оформлення, а користувач потребує лише базових знань роботи з комп’ютером.

Програма може бути корисною та актуальною у малому та середньому бізнесі, де необхідно вести облік матеріальних цінностей. Завдяки зрозумілому інтерфейсу і базовому функціоналу, забезпечить швидке і комфортне користування программою.

4 ІНСТРУКЦІЯ З КОРИСТУВАННЯ ПРОГРАМНОЮ СИСТЕМОЮ

Робота з програмою починається з вікна авторизації. Воно має поля для вводу логіну та паролю. При відсутності з’єднання з базою даних, вікно повідомить про неможливість підключення, якщо було введено невірний логін та пароль, буде показано відповідне повідомлення. Вікно авторизації зображено на рисунку 4.1

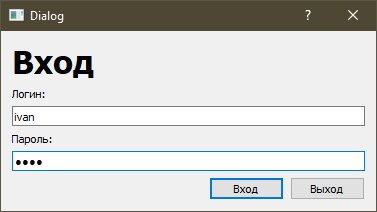


Рисунок 4.1 – Вікно авторизації

Після успішної авторизації користувачу стане доступне головне вікно з доступними для нього функціями. Програма має можливість розмежовувати доступні функції в залежності від посади користувача. Програма має три посади для розмежовування доступу: «Директор», «Бухгалтер» та «Головний складу». Також розмежування доступу є і по складах, для цього при створенні нового користувача потрібно обрати посаду и вибрати зі списку ті склади, до яких буде мати доступ користувач. Головне вікно програми зображено на рисунку 4.2.

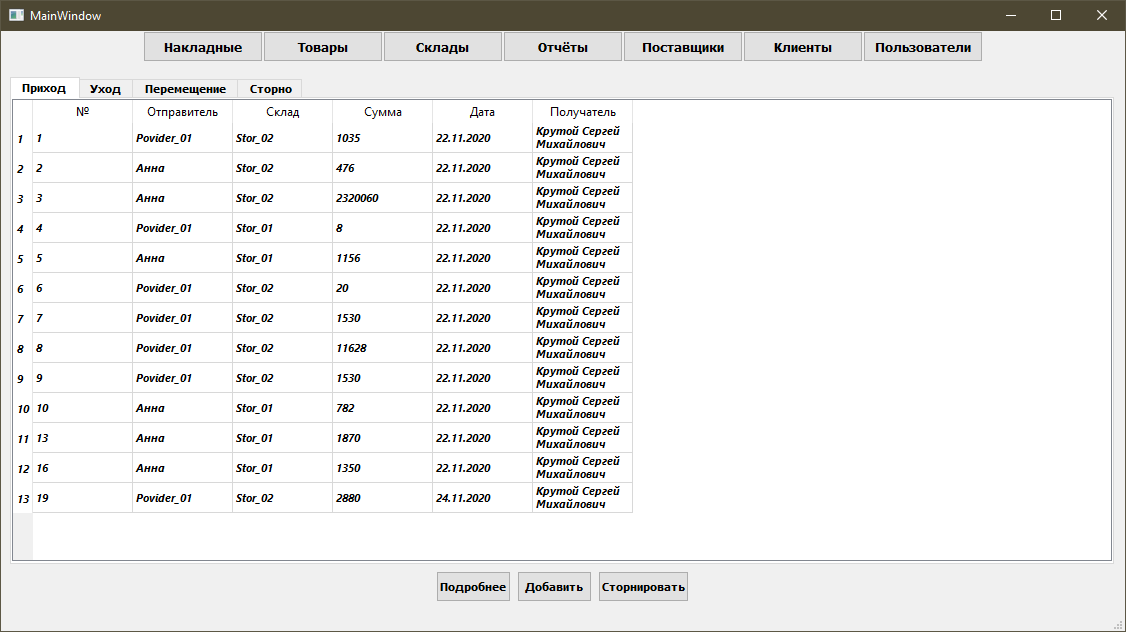


Рисунок 4.2 – Головне вікно програми

Програма може працювати з накладними, які розмежовані окремо на накладні по постачанню, продажу та переміщенню, окремо виділені сторновані накладні. Сторнована накладна – це накладна, в якій була зроблена помилка, але оскільки видаляти і редагувати накладні не дозволяється, вона помічається як сторнована і в фінансових розрахунках не приймає участі. Для того щоб перейти до роботи з накладними треба натиснути кнопку «Накладні». Вікно накладних зображено на рисунку 4.3.

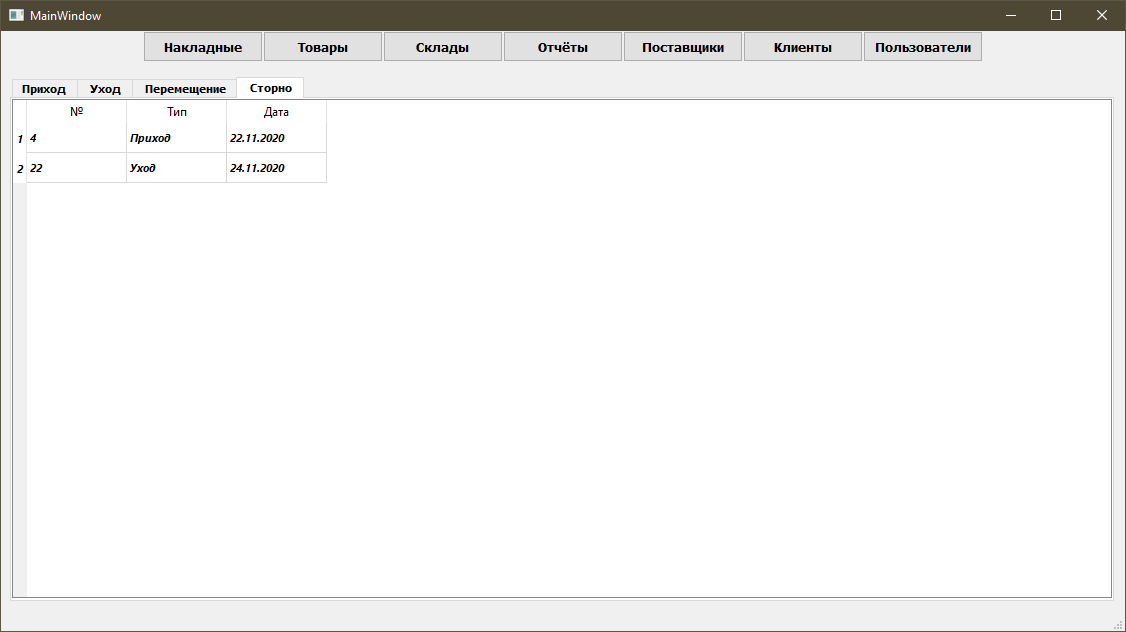


Рисунок 4.3 – Вікно перегляду накладних

Накладні можна додавати, переглядати детальну інформацію по кожній та сторнувати. Для виконання усіх цих дій треба обрати необхідний запис у таблиці и натиснути відповідну кнопку. На рисунку 4.4 зображено вікно додавання накладної.

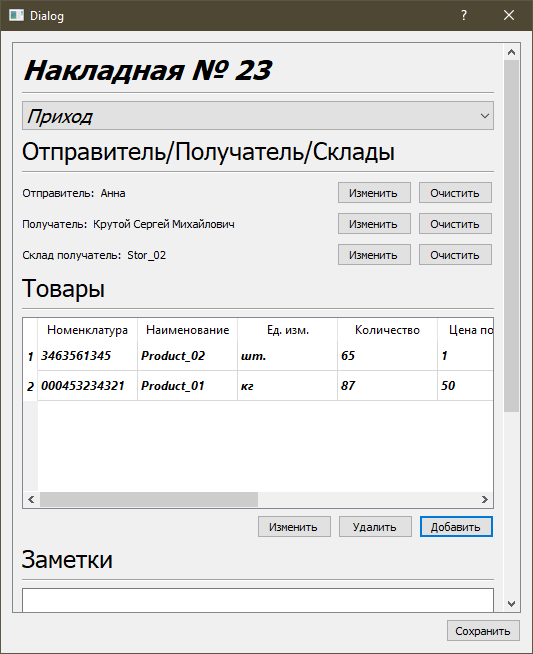


Рисунок 4.4 – Вікно додавання накладної

Для успішного додання накладної, треба заповнити усі поля у вікні, це тип накладної, постачальники та отримувачі, список товарів, метод оплати, також для кожної накладної доступні примітки. Приклад заповнення вікна накладної зображено на рисунку 4.5

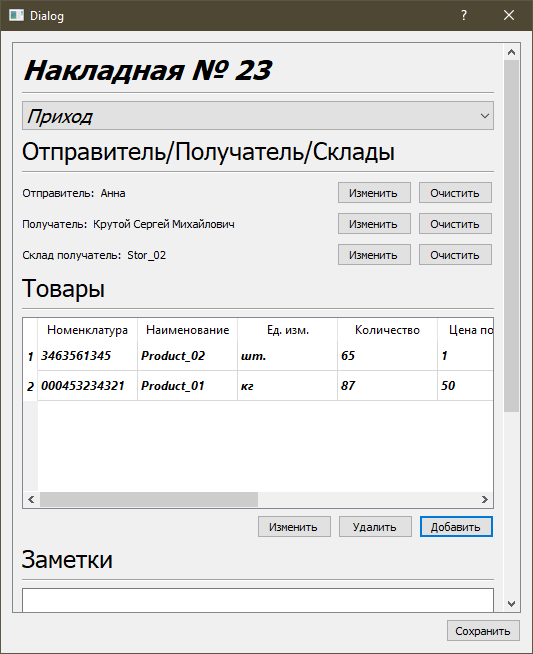


Рисунок 4.5 – Приклад заповнення

Програма дозволяє додавати та редагувати товари, видалення доступне лише для тих товарів, які ще не встигли потрапити до накладної. Таке обмеження введено для уникнення помилок у базі даних та при формуванні звітності. Вікно додавання та редагування товару зображено на рисунку 4.6.

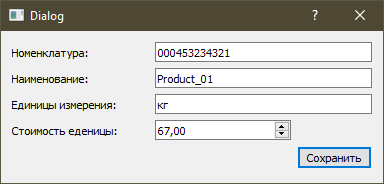


Рисунок 4.6 – Вікно додавання та редагування товару

Програма дозволяє додавати та редагувати склади, видалення доступне лише для тих складів, які ще не встигли потрапити до накладної. Таке обмеження введено для уникнення помилок у базі даних та при формуванні звітності. Вікно додавання та редагування складу зображено на рисунку 4.7

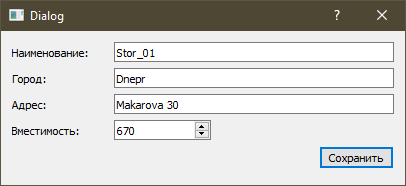


Рисунок 4.7 – Вікно додавання та редагування складу

Програма дозволяє додавати та редагувати постачальників, видалення доступне лише для тих постачальників, які ще не встигли потрапити до накладної. Таке обмеження введено для уникнення помилок у базі даних та при формуванні звітності. Вікно додавання та редагування постачальників зображено на рисунку 4.8

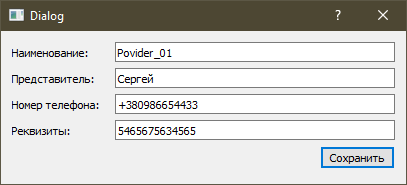


Рисунок 4.8 – Вікно додавання та редагування постачальників

Програма дозволяє додавати та редагувати клієнтів, видалення доступне лише для тих клієнтів, які ще не встигли потрапити до накладної. Таке обмеження введено для уникнення помилок у базі даних та при формуванні звітності. Вікно додавання та редагування клієнтів зображено на рисунку 4.9

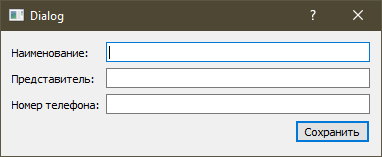


Рисунок 4.9 – Вікно додавання клієнта

Програма дозволяє додавати та редагувати користувачів, видалення доступне лише для тих користувачів, які ще не встигли потрапити до накладної. Таке обмеження введено для уникнення помилок у базі даних та при формуванні звітності. Вікно додавання та редагування користувачів зображено на рисунку 4.10

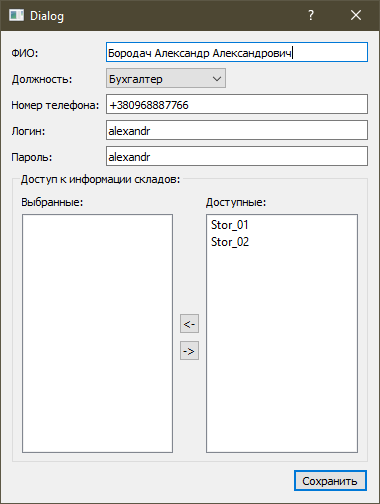


Рисунок 4.10 – Вікно додавання та редагування користувачів

Програма дозволяє формувати звітність. Доступні звіти по постачанню товарів, по продажу товарів, звіт найпопулярніших товарів та звіт прибутку. Звіти спочатку формуються у вікні програми, але за бажанням їх можна експортувати у PDF-формат для відправки або друку. На вибір є два варіанти для представлення звіту, у вигляді чеку або у вигляді документу, обидва варіанти добре експортуються и виглядають на листі. При експорті можна обрати дерикторію для зберігання, за замовчуванням це папка програми. Вікно формування звітності зображено на рисунку 4.11. Вікно перегляду звіту перед експортом зображено на рисунку 4.12.

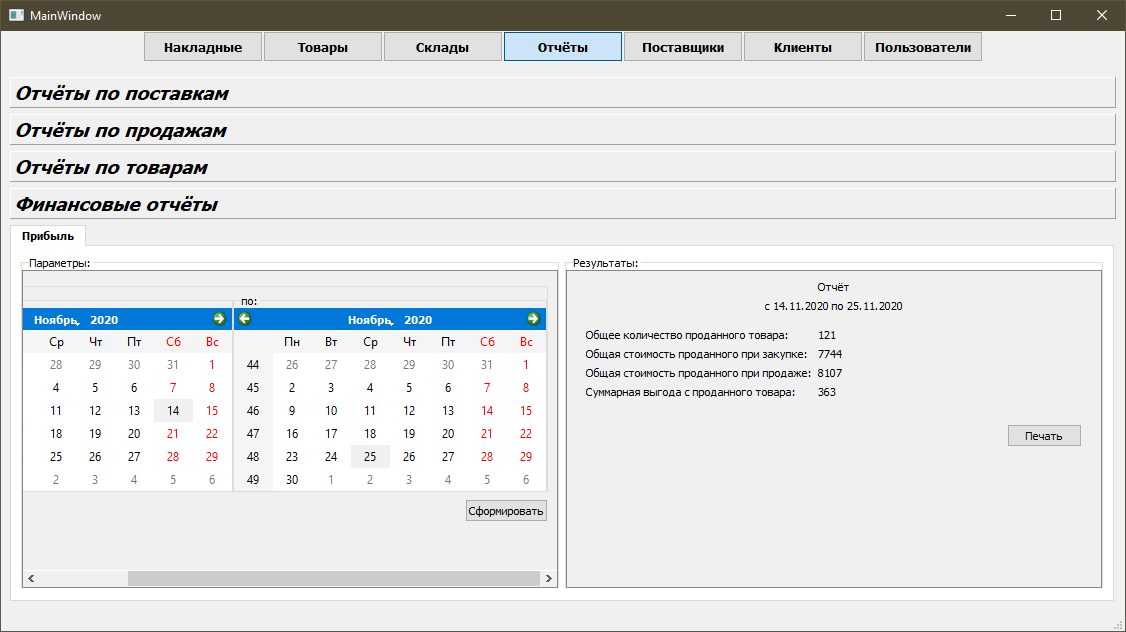


Рисунок 4.11 – Вікно формування звітності

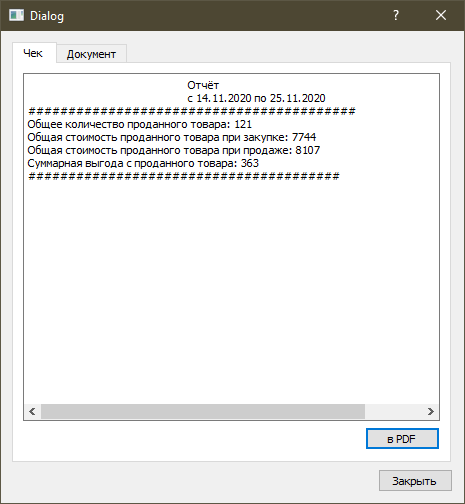


Рисунок 4.12 – Вікно перегляду звіту перед експортом

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Відео та лекції по виченню Qt Creatorehttps://resources.qt.io/resources-by-content-type-videos-demos-tutorials.
2. Електронний научно-практический журнал «Современная техника и технологии» [Електронний ресурс] // technology.snauka.ru. Режим доступу: <http://technology.snauka.ru/2016/12/11465>
3. М. Саммерфілд «Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на С++», Символ-Плюс, 2011
4. Ж. Бланшет, М. Саммерфілд «QT 4: программирование GUI на С++», КУДИЦ-Пресс, 2008
5. М. Шлеє «Qt 5.3. Профессиональное программирование на C++», БХВ-Петербург, 2015
6. Офіційна документація — doc.qt.io

Додаток А

Лістинг програми

Лістинг А.1 – Клас роботи з базою даних

#ifndef DATABASEMODULE\_H

#define DATABASEMODULE\_H

#include "objectsdatabase.h"

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QVariant>

#include <QDebug>

*struct* **resultQuery**

{

bool flag;

QString message;

};

*class* **DatabaseModule**

{

QSqlDatabase db;

user connectedUser;

*public*:

**DatabaseModule**();

bool **setDatabaseConnection**(QSqlDatabase db\_connection, user user\_connection);

int **getPrivileges**();

resultQuery **addInvoice**(invoice invoice);

resultQuery **addStorage**(storage storage);

resultQuery **addProduct**(product product);

resultQuery **addUser**(user user);

resultQuery **addProvider**(provider provider);

resultQuery **addClient**(client client);

invoice **getInvoiceById**(int id);

user **getUserById**(int id);

client **getClientById**(int id);

storage **getStorageById**(int id);

provider **getProviderById**(int id);

typeInvoice **getTypeInvoiceById**(int id);

paymenttype **getPaymentTypeById**(int id);

storno **getStornoByIdInvoice**(int idInvoice);

position **getPositionById**(int id);

product **getProductById**(int id);

QList <product> **getListProductByIdStorage**(int idStorage);

QList <product> **getListProductByIdInvoice**(int idInvoice);

QList <product> **getListProducts**();

QList <storage> **getListStorages**();

QList <position> **getListPositions**();

QList <client> **getListClients**();

QList <provider> **getListProviders**();

QList <invoice> **getListInvoiceByType**(int idType);

QList <invoice> **getListInvoiceByType**(int idType, int idStorage);

QList <invoice> **getListInvoiceByTypeAvialableConnectedUser**(int idType);

QList <storno> **getStornedInvoices**();

QList <user> **getListUsers**();

};

#endif *//* *DATABASEMODULE\_H*

#include "databasemodule.h"

DatabaseModule::**DatabaseModule**()

{

}

bool DatabaseModule::**setDatabaseConnection**(QSqlDatabase db\_connection, user user\_connection)

{

db = db\_connection;

connectedUser = user\_connection;

*return* db.open();

}

int DatabaseModule::**getPrivileges**()

{

storage tmp\_storage;

db.open();

QSqlQuery query;

query.prepare("SELECT id FROM users WHERE login = :login");

query.bindValue(":login", QVariant(connectedUser.login).toString());

query.exec();

*if*(query.size() == 0)

*return* 0;

*else*

{

query.next();

connectedUser = getUserById(query.value(0).toInt());

}

*return* connectedUser.position.id;

}

resultQuery DatabaseModule::**addUser**(user user)

{

resultQuery result;

QSqlQuery query;

query.prepare("SELECT id FROM users WHERE login = :login");

query.bindValue(":login", QVariant(user.login).toString());

query.exec();

*if*(query.size())

{

result.message = "Пользователь с таким логином уже существует.";

result.flag = *false*;

*return* result;

}

*else*

{

query.prepare("INSERT INTO users (login, password, PIB, phonenumber, idPosition) VALUES(:login, :password, :PIB, :phonenumber, :idPosition)");

query.bindValue(":login", QVariant(user.login).toString());

query.bindValue(":password", QVariant(user.password).toString());

query.bindValue(":PIB", QVariant(user.pib).toString());

query.bindValue(":phonenumber", QVariant(user.phonenumber).toString());

query.bindValue(":idPosition", user.position.id);

query.exec();

*if*(user.listStorages.size() != 0)

{

query.prepare("SELECT id FROM users WHERE login = :login");

query.bindValue(":login", QVariant(user.login).toString());

query.exec();

query.next();

user.id = query.value(0).toInt();

query.prepare("INSERT user\_storage VALUES(:idUser, :idStorage)");

*for*(int i = 0; i < user.listStorages.size(); i++)

{

query.bindValue(":idUser", user.id);

query.bindValue(":idStorage", user.listStorages.at(i).id);

query.exec();

}

}

result.flag = *true*;

result.message = "Пользователь добавлен.";

*return* result;

}

}

resultQuery DatabaseModule::**addClient**(client client)

{

resultQuery result;

QSqlQuery query;

query.prepare("SELECT id FROM clients WHERE title = :title");

query.bindValue(":title", QVariant(client.title).toString());

query.exec();

*if*(query.size())

{

result.message = "Клиент с таким наименованием уже существует.";

result.flag = *false*;

*return* result;

}

*else*

{

query.prepare("INSERT INTO clients (PIB, phonenumber, title) VALUES(:PIB, :phonenumber, :title)");

query.bindValue(":PIB", QVariant(client.pib).toString());

query.bindValue(":phonenumber", QVariant(client.phonenumber).toString());

query.bindValue(":title", QVariant(client.title).toString());

query.exec();

result.flag = *true*;

result.message = "Клиент добавлен.";

*return* result;

}

}

resultQuery DatabaseModule::**addProvider**(provider provider)

{

resultQuery result;

QSqlQuery query;

query.prepare("SELECT id FROM providers WHERE title = :title");

query.bindValue(":title", QVariant(provider.title).toString());

query.exec();

*if*(query.size())

{

result.message = "Поставщик с таким наименованием уже существует.";

result.flag = *false*;

*return* result;

}

*else*

{

query.prepare("INSERT INTO providers (title, PIB, phonenumber, requisites) VALUES(:title, :PIB, :phonenumber, :requisites)");

query.bindValue(":title", QVariant(provider.title).toString());

query.bindValue(":PIB", QVariant(provider.pib).toString());

query.bindValue(":phonenumber", QVariant(provider.phonenumber).toString());

query.bindValue(":requisites", QVariant(provider.requisites).toString());

query.exec();

result.flag = *true*;

result.message = "Поставщик добавлен.";

*return* result;

}

}

resultQuery DatabaseModule::**addStorage**(storage storage)

{

resultQuery result;

QSqlQuery query;

query.prepare("SELECT id FROM storage WHERE title = :title");

query.bindValue(":title", QVariant(storage.title).toString());

query.exec();

*if*(query.size())

{

result.message = "Склад с таким наименованием уже существует.";

result.flag = *false*;

*return* result;

}

*else*

{

query.prepare("INSERT INTO storage (title, city, address, capacity) VALUES(:title, :city, :address, :capacity)");

query.bindValue(":title", QVariant(storage.title).toString());

query.bindValue(":city", QVariant(storage.city).toString());

query.bindValue(":address", QVariant(storage.address).toString());

query.bindValue(":capacity", storage.capacity);

query.exec();

result.flag = *true*;

result.message = "Склад добавлен.";

*return* result;

}

}

resultQuery DatabaseModule::**addProduct**(product product)

{

resultQuery result;

QSqlQuery query;

query.prepare("SELECT id FROM product WHERE title = :title OR nomenclature = :nomenclature");

query.bindValue(":title", product.title);

query.bindValue(":nomenclature", product.nomenclature);

query.exec();

*if*(query.size())

{

result.message = "Такой товар уже существует.";

result.flag = *false*;

*return* result;

}

*else*

{

query.prepare("INSERT INTO product (title, measuring, nomenclature, price) VALUES(:title, :measuring, :nomenclature, :price)");

query.bindValue(":title", product.title);

query.bindValue(":measuring", product.measuring);

query.bindValue(":nomenclature", product.nomenclature);

query.bindValue(":price", product.price);

query.exec();

result.flag = *true*;

result.message = "Товар добавлен.";

*return* result;

}

}

resultQuery DatabaseModule::**addInvoice**(invoice invoice)

{

resultQuery result;

QSqlQuery query;

*switch* (invoice.typeInvoice.id)

{

*case* 0:

{

query.prepare("INSERT INTO invoices (sum, date, idTypeInvoice, idStorageRecipient, idProvider, note, storno, idUserRecipient) VALUES(:sum, :date, :idTypeInvoice, :idStorageRecipient, :idProvider, :note, 0, :idUserRecipient)");

query.bindValue(":sum", invoice.sum);

query.bindValue(":date", invoice.data);

query.bindValue(":idTypeInvoice", invoice.typeInvoice.id);

query.bindValue(":idStorageRecipient", invoice.storageRecipient.id);

query.bindValue(":idProvider", invoice.provider.id);

query.bindValue(":note", QVariant(invoice.note).toString());

query.bindValue(":idUserRecipient", invoice.userRecipient.id);

query.exec();

query.prepare("INSERT INTO product\_in\_invoice VALUES(:idInvoice, :idProduct, :price, :quantity, :accounting\_price)");

*for*(int i = 0; i < invoice.listProducts.size(); i++)

{

query.bindValue(":idInvoice", invoice.id);

query.bindValue(":idProduct", invoice.listProducts.at(i).id);

query.bindValue(":price", invoice.listProducts.at(i).priceSender);

query.bindValue(":quantity", invoice.listProducts.at(i).quantity);

query.bindValue(":accounting\_price", invoice.listProducts.at(i).price);

query.exec();

}

*break*;

}

*case* 1:

{

query.prepare("INSERT INTO invoices (sum, date, idTypeInvoice, idStorageSender, idUserSender, idClient, note, storno, idPaymentType) VALUES(:sum, :date, :idTypeInvoice, :idStorageSender, :idClient, :note, 0, :idPaymentType)");

query.bindValue(":sum", invoice.sum);

query.bindValue(":date", invoice.data);

query.bindValue(":idTypeInvoice", invoice.typeInvoice.id);

query.bindValue(":idStorageSender", invoice.storageSender.id);

query.bindValue(":idClient", invoice.client.id);

query.bindValue(":note", QVariant(invoice.note).toString());

query.bindValue(":idUserSender", invoice.userSender.id);

query.bindValue(":idPaymentType", invoice.paymentType.id);

query.exec();

query.prepare("INSERT INTO product\_in\_invoice VALUES(:idInvoice, :idProduct, :price, :quantity)");

*for*(int i = 0; i < invoice.listProducts.size(); i++)

{

query.bindValue(":idInvoice", invoice.id);

query.bindValue(":idProduct", invoice.listProducts.at(i).id);

query.bindValue(":price", invoice.listProducts.at(i).priceSender);

query.bindValue(":quantity", invoice.listProducts.at(i).quantity);

query.exec();

}

*break*;

}

*case* 2:

{

query.prepare("INSERT INTO invoices (date, idTypeInvoice, idStorageSender, idUserSender, idStorageRecipient, idUserRecipient, note, storno) VALUES(:date, :idTypeInvoice, :idStorageSender, :idUserSender, :idStorageRecipient, :idUserRecipient, :note, 0)");

query.bindValue(":date", invoice.data);

query.bindValue(":idTypeInvoice", invoice.typeInvoice.id);

query.bindValue(":idUserSender", invoice.userSender.id);

query.bindValue(":idStorageSender", invoice.storageSender.id);

query.bindValue(":idUserRecipient", invoice.userRecipient.id);

query.bindValue(":idStorageRecipient", invoice.storageRecipient.id);

query.bindValue(":note", QVariant(invoice.note).toString());

query.exec();

query.prepare("INSERT INTO product\_in\_invoice VALUES(:idInvoice, :idProduct, :quantity)");

*for*(int i = 0; i < invoice.listProducts.size(); i++)

{

query.bindValue(":idInvoice", invoice.id);

query.bindValue(":idProduct", invoice.listProducts.at(i).id);

query.bindValue(":quantity", invoice.listProducts.at(i).quantity);

query.exec();

}

*break*;

}

}

result.flag = *true*;

result.message = "Накладная добавлена.";

*return* result;

}

typeInvoice DatabaseModule::**getTypeInvoiceById**(int id)

{

typeInvoice type;

QSqlQuery query;

query.prepare("SELECT \* FROM type\_invoice WHERE id = :id");

query.bindValue(":id", id);

query.exec();

query.next();

type.id = query.value(0).toInt();

type.title = query.value(1).toString();

*return* type;

}

position DatabaseModule::**getPositionById**(int id)

{

position pos;

QSqlQuery query;

query.prepare("SELECT \* FROM position WHERE id = :id");

query.bindValue(":id", id);

query.exec();

query.next();

pos.id = query.value(0).toInt();

pos.title = query.value(1).toString();

*return* pos;

}

user DatabaseModule::**getUserById**(int id)

{

user us;

QSqlQuery query;

query.prepare("SELECT \* FROM users WHERE id = :id");

query.bindValue(":id", id);

query.exec();

query.next();

us.id = query.value(0).toInt();

us.login = query.value(1).toString();

us.password = query.value(2).toString();

us.pib = query.value(3).toString();

us.phonenumber = query.value(4).toString();

us.position = getPositionById(query.value(5).toInt());

QSqlQuery qu2;

qu2.prepare("SELECT idStorage FROM user\_storage WHERE idUser = :idUser");

qu2.bindValue(":idUser", us.id);

qu2.exec();

*while*(qu2.next())

us.listStorages.push\_back(getStorageById(qu2.value(0).toInt()));

*return* us;

}

provider DatabaseModule::**getProviderById**(int id)

{

provider pr;

QSqlQuery qu;

qu.prepare("SELECT \* FROM users WHERE id = :id");

qu.bindValue(":id", id);

qu.exec();

qu.next();

pr.id = qu.value(0).toInt();

pr.title = qu.value(1).toString();

pr.pib = qu.value(2).toString();

pr.phonenumber = qu.value(3).toString();

pr.requisites = qu.value(4).toString();

*return* pr;

}

QList <product> DatabaseModule::**getListProductByIdStorage**(int idStorage)

{

QList <product> listProducts;

product pr;

QSqlQuery qu;

qu.prepare("SELECT \* FROM product WHERE id = (SELECT idProduct FROM product\_in\_storage WHERE idStorage = :idStorage)");

qu.bindValue(":idStorage", idStorage);

qu.exec();

QSqlQuery qu2;

qu2.prepare("SELECT quantity FROM product\_in\_storage WHERE idProduct = :idProduct AND idStorage = :idStorage");

qu2.bindValue(":idStorage", idStorage);

*while*(qu.next())

{

pr.id = qu.value(0).toInt();

pr.title = qu.value(1).toString();

pr.measuring = qu.value(2).toString();

pr.nomenclature = qu.value(3).toString();

pr.price = qu.value(4).toFloat();

qu2.bindValue(":idProduct", pr.id);

qu2.exec();

qu2.next();

pr.quantity = qu2.value(0).toInt();

listProducts.push\_back(pr);

}

*return* listProducts;

}

QList <product> DatabaseModule::**getListProductByIdInvoice**(int idInvoice)

{

QList <product> listProducts;

product pr;

QSqlQuery qu;

qu.prepare("SELECT \* FROM product WHERE id = (SELECT idProduct FROM product\_in\_invoice WHERE idInvoice = :idInvoice)");

qu.bindValue(":idInvoice", idInvoice);

qu.exec();

QSqlQuery qu2;

qu2.prepare("SELECT price, quantity, accounting\_price FROM product\_in\_invoice WHERE idProduct = :idProduct AND idInvoice = :idInvoice");

qu2.bindValue(":idInvoice", idInvoice);

*while*(qu.next())

{

pr.id = qu.value(0).toInt();

pr.title = qu.value(1).toString();

pr.measuring = qu.value(2).toString();

pr.nomenclature = qu.value(3).toString();

qu2.bindValue(":idProduct", pr.id);

qu2.exec();

qu2.next();

pr.priceSender = qu2.value(0).toFloat();

pr.quantity = qu2.value(1).toInt();

pr.price = qu2.value(2).toFloat();

listProducts.push\_back(pr);

}

*return* listProducts;

}

storage DatabaseModule::**getStorageById**(int id)

{

storage st;

QSqlQuery qu;

qu.prepare("SELECT \* FROM storage WHERE id = :id");

qu.bindValue(":id", id);

qu.exec();

qu.next();

st.id = qu.value(0).toInt();

st.title = qu.value(1).toString();

st.city = qu.value(2).toString();

st.address = qu.value(3).toString();

st.capacity = qu.value(4).toInt();

st.listProducts = getListProductByIdStorage(st.id);

*return* st;

}

client DatabaseModule::**getClientById**(int id)

{

client cl;

QSqlQuery qu;

qu.prepare("SELECT \* FROM clients WHERE id = :id");

qu.bindValue(":id", id);

qu.exec();

qu.next();

cl.id = qu.value(0).toInt();

cl.pib = qu.value(1).toString();

cl.phonenumber = qu.value(2).toString();

cl.title = qu.value(3).toString();

*return* cl;

}

paymenttype DatabaseModule::**getPaymentTypeById**(int id)

{

paymenttype pt;

QSqlQuery qu;

qu.prepare("SELECT \* FROM payment\_type WHERE id = :id");

qu.bindValue(":id", id);

qu.exec();

qu.next();

pt.id = qu.value(0).toInt();

pt.title = qu.value(1).toString();

*return* pt;

}

QList <invoice> DatabaseModule::**getListInvoiceByType**(int idType)

{

QList <invoice> listInvoices;

invoice tmp\_invoice;

QSqlQuery query;

query.prepare("SELECT \* FROM invoices WHERE idTypeInvoice = :idTypeInvoice");

query.bindValue(":idTypeInvoice", idType);

query.exec();

*while*(query.next())

{

tmp\_invoice.id = query.value(0).toInt();

tmp\_invoice.sum = query.value(1).toFloat();

tmp\_invoice.data = query.value(2).toDate();

tmp\_invoice.typeInvoice = getTypeInvoiceById(query.value(3).toInt());

tmp\_invoice.listProducts = getListProductByIdInvoice(tmp\_invoice.id);

*switch* (tmp\_invoice.typeInvoice.id)

{

*case* 1:

{

tmp\_invoice.storageRecipient = getStorageById(query.value(5).toInt());

tmp\_invoice.provider = getProviderById(query.value(7).toInt());

tmp\_invoice.note = query.value(10).toString();

tmp\_invoice.storno = query.value(11).toInt();

tmp\_invoice.userRecipient = getUserById(query.value(12).toInt());

*break*;

}

*case* 2:

{

tmp\_invoice.storageSender = getStorageById(query.value(4).toInt());

tmp\_invoice.userSender = getUserById(query.value(6).toInt());

tmp\_invoice.client = getClientById(query.value(8).toInt());

tmp\_invoice.paymentType = getPaymentTypeById(query.value(9).toInt());

tmp\_invoice.note = query.value(10).toString();

tmp\_invoice.storno = query.value(11).toInt();

*break*;

}

*case* 3:

{

tmp\_invoice.storageSender = getStorageById(query.value(4).toInt());

tmp\_invoice.storageRecipient = getStorageById(query.value(5).toInt());

tmp\_invoice.userSender = getUserById(query.value(6).toInt());

tmp\_invoice.userRecipient = getUserById(query.value(12).toInt());

tmp\_invoice.note = query.value(10).toString();

tmp\_invoice.storno = query.value(11).toInt();

*break*;

}

}

listInvoices.push\_back(tmp\_invoice);

}

*return* listInvoices;

}

QList <invoice> DatabaseModule::**getListInvoiceByType**(int idType, int idStorage)

{

QList <invoice> listInvoices;

invoice tmp\_invoice;

QSqlQuery query;

query.prepare("SELECT \* FROM invoices WHERE idTypeInvoice = :idTypeInvoice AND (idStorageSender = :idStorage OR idStorageRecipient = :idStorage)");

query.bindValue(":idTypeInvoice", idType);

query.bindValue(":idStorage", idStorage);

query.exec();

*while*(query.next())

{

tmp\_invoice.id = query.value(0).toInt();

tmp\_invoice.sum = query.value(1).toFloat();

tmp\_invoice.data = query.value(2).toDate();

tmp\_invoice.typeInvoice = getTypeInvoiceById(query.value(3).toInt());

tmp\_invoice.listProducts = getListProductByIdInvoice(tmp\_invoice.id);

*switch* (tmp\_invoice.typeInvoice.id)

{

*case* 1:

{

tmp\_invoice.storageRecipient = getStorageById(query.value(5).toInt());

tmp\_invoice.provider = getProviderById(query.value(7).toInt());

tmp\_invoice.note = query.value(10).toString();

tmp\_invoice.storno = query.value(11).toInt();

tmp\_invoice.userRecipient = getUserById(query.value(12).toInt());

*break*;

}

*case* 2:

{

tmp\_invoice.storageSender = getStorageById(query.value(4).toInt());

tmp\_invoice.userSender = getUserById(query.value(6).toInt());

tmp\_invoice.client = getClientById(query.value(8).toInt());

tmp\_invoice.paymentType = getPaymentTypeById(query.value(9).toInt());

tmp\_invoice.note = query.value(10).toString();

tmp\_invoice.storno = query.value(11).toInt();

*break*;

}

*case* 3:

{

tmp\_invoice.storageSender = getStorageById(query.value(4).toInt());

tmp\_invoice.storageRecipient = getStorageById(query.value(5).toInt());

tmp\_invoice.userSender = getUserById(query.value(6).toInt());

tmp\_invoice.userRecipient = getUserById(query.value(12).toInt());

tmp\_invoice.note = query.value(10).toString();

tmp\_invoice.storno = query.value(11).toInt();

*break*;

}

}

listInvoices.push\_back(tmp\_invoice);

}

*return* listInvoices;

}

QList <invoice> DatabaseModule::**getListInvoiceByTypeAvialableConnectedUser**(int idType)

{

QList <invoice> listInvoices;

*for*(int i = 0; i < connectedUser.listStorages.size(); i++)

listInvoices += getListInvoiceByType(idType, connectedUser.listStorages.at(i).id);

*return* listInvoices;

}

QList <product> DatabaseModule::**getListProducts**()

{

QList <product> listProducts;

product pr;

QSqlQuery qu;

qu.prepare("SELECT \* FROM product");

qu.exec();

*while*(qu.next())

{

pr.id = qu.value(0).toInt();

pr.title = qu.value(1).toString();

pr.measuring = qu.value(2).toString();

pr.nomenclature = qu.value(3).toString();

pr.price = qu.value(4).toFloat();

listProducts.push\_back(pr);

}

*return* listProducts;

}

QList <storage> DatabaseModule::**getListStorages**()

{

QList <storage> listStorages;

storage st;

db.open();

QSqlQuery qu;

QSqlQuery qu2;

*for*(int i = 0; i < connectedUser.listStorages.size(); i++)

{

qu.prepare("SELECT \* FROM storage WHERE id = :idStorage");

qu.bindValue(":idStorage", connectedUser.listStorages.at(i).id);

*//qu.prepare("SELECT* *\** *FROM* *storage");*

qu.exec();

qu2.prepare("SELECT sum(quantity) FROM product\_in\_storage WHERE idStorage = :idStorage");

*while*(qu.next())

{

st.id = qu.value(0).toInt();

st.title = qu.value(1).toString();

st.city = qu.value(2).toString();

st.address = qu.value(3).toString();

st.capacity = qu.value(4).toInt();

qu2.bindValue(":idStorage", st.id);

qu2.exec();

qu2.next();

st.workload = qu2.value(0).toInt();

st.listProducts = getListProductByIdStorage(st.id);

listStorages.push\_back(st);

}

}

*return* listStorages;

}

QList <position> DatabaseModule::**getListPositions**()

{

QList <position> listPositions;

position pos;

QSqlQuery qu;

qu.prepare("SELECT \* FROM position");

qu.exec();

*while*(qu.next())

{

pos.id = qu.value(0).toInt();

pos.title = qu.value(1).toString();

listPositions.push\_back(pos);

}

*return* listPositions;

}

QList <user> DatabaseModule::**getListUsers**()

{

QList <user> listUsers;

user us;

QSqlQuery qu;

qu.prepare("SELECT \* FROM users");

qu.exec();

*while*(qu.next())

{

us.id = qu.value(0).toInt();

us.login = qu.value(1).toString();

us.password = qu.value(2).toString();

us.pib = qu.value(3).toString();

us.phonenumber = qu.value(4).toString();

us.position = getPositionById(qu.value(5).toInt());

QSqlQuery qu2;

qu2.prepare("SELECT idStorage FROM user\_storage WHERE idUser = :idUser");

qu2.bindValue(":idUser", us.id);

qu2.exec();

*while*(qu2.next())

us.listStorages.push\_back(getStorageById(qu2.value(0).toInt()));

listUsers.push\_back(us);

}

*return* listUsers;

}

QList <client> DatabaseModule::**getListClients**()

{

QList <client> listClients;

client cl;

QSqlQuery qu;

qu.prepare("SELECT \* FROM clients");

qu.exec();

*while*(qu.next())

{

cl.id = qu.value(0).toInt();

cl.pib = qu.value(1).toString();

cl.phonenumber = qu.value(2).toString();

cl.title = qu.value(3).toString();

listClients.push\_back(cl);

}

*return* listClients;

}

QList <provider> DatabaseModule::**getListProviders**()

{

QList <provider> listProviders;

provider pr;

QSqlQuery qu;

qu.prepare("SELECT \* FROM providers");

qu.exec();

*while*(qu.next())

{

pr.id = qu.value(0).toInt();

pr.title = qu.value(1).toString();

pr.pib = qu.value(2).toString();

pr.phonenumber = qu.value(3).toString();

pr.requisites = qu.value(4).toString();

listProviders.push\_back(pr);

}

*return* listProviders;

}

Додаток Б

Лістинг бази даних

Лістинг Б.1 – База даних проекту

CREATE DATABASE IF NOT EXISTS `baza` /\*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci \*/ /\*!80016 DEFAULT ENCRYPTION='N' \*/;

USE `baza`;

-- MySQL dump 10.13 Distrib 8.0.22, for Win64 (x86\_64)

--

-- Host: 127.0.0.1 Database: baza

-- ------------------------------------------------------

-- Server version 8.0.22

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!50503 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `clients`

--

DROP TABLE IF EXISTS `clients`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `clients` (

`id` int NOT NULL AUTO\_INCREMENT,

`PIB` varchar(45) NOT NULL,

`phonenumber` varchar(13) NOT NULL,

`title` varchar(45) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `invoices`

--

DROP TABLE IF EXISTS `invoices`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `invoices` (

`id` int NOT NULL AUTO\_INCREMENT,

`sum` float DEFAULT NULL,

`date` date DEFAULT NULL,

`idTypeInvoice` int NOT NULL,

`idStorageSender` int DEFAULT NULL,

`idStorageRecipient` int DEFAULT NULL,

`idUserSender` int DEFAULT NULL,

`idProvider` int DEFAULT NULL,

`idClient` int DEFAULT NULL,

`idPaymentType` int DEFAULT NULL,

`note` text,

`storno` int DEFAULT NULL,

`idUserRecipient` int DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `idTypeInvoice` (`idTypeInvoice`),

KEY `idStorageSender` (`idStorageSender`),

KEY `idStorageRecipient` (`idStorageRecipient`),

KEY `idProvider` (`idProvider`),

KEY `idClient` (`idClient`),

KEY `idPaymentType` (`idPaymentType`),

KEY `idUserRecipient\_idx` (`idUserRecipient`),

KEY `idUserSender` (`idUserSender`),

CONSTRAINT `invoices\_ibfk\_1` FOREIGN KEY (`idTypeInvoice`) REFERENCES `type\_invoice` (`id`),

CONSTRAINT `invoices\_ibfk\_2` FOREIGN KEY (`idStorageSender`) REFERENCES `storage` (`id`),

CONSTRAINT `invoices\_ibfk\_3` FOREIGN KEY (`idStorageRecipient`) REFERENCES `storage` (`id`),

CONSTRAINT `invoices\_ibfk\_5` FOREIGN KEY (`idProvider`) REFERENCES `providers` (`id`),

CONSTRAINT `invoices\_ibfk\_6` FOREIGN KEY (`idClient`) REFERENCES `clients` (`id`),

CONSTRAINT `invoices\_ibfk\_7` FOREIGN KEY (`idPaymentType`) REFERENCES `payment\_type` (`id`),

CONSTRAINT `invoices\_ibfk\_8` FOREIGN KEY (`idUserSender`) REFERENCES `users` (`id`),

CONSTRAINT `invoices\_ibfk\_9` FOREIGN KEY (`idUserRecipient`) REFERENCES `users` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `payment\_type`

--

DROP TABLE IF EXISTS `payment\_type`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `payment\_type` (

`id` int NOT NULL AUTO\_INCREMENT,

`title` varchar(15) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `position`

--

DROP TABLE IF EXISTS `position`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `position` (

`id` int NOT NULL AUTO\_INCREMENT,

`title` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `product`

--

DROP TABLE IF EXISTS `product`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `product` (

`id` int NOT NULL AUTO\_INCREMENT,

`title` varchar(45) NOT NULL,

`measuring` varchar(10) NOT NULL,

`nomenclature` varchar(45) NOT NULL,

`price` float NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `product\_in\_invoice`

--

DROP TABLE IF EXISTS `product\_in\_invoice`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `product\_in\_invoice` (

`idInvoice` int NOT NULL,

`idProduct` int NOT NULL,

`price` float DEFAULT NULL,

`quantity` int NOT NULL,

`accounting\_price` float DEFAULT NULL,

KEY `idInvoice` (`idInvoice`),

KEY `idProduct` (`idProduct`),

CONSTRAINT `product\_in\_invoice\_ibfk\_1` FOREIGN KEY (`idInvoice`) REFERENCES `invoices` (`id`),

CONSTRAINT `product\_in\_invoice\_ibfk\_2` FOREIGN KEY (`idProduct`) REFERENCES `product` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `product\_in\_storage`

--

DROP TABLE IF EXISTS `product\_in\_storage`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `product\_in\_storage` (

`idProduct` int NOT NULL,

`idStorage` int NOT NULL,

`quantity` int DEFAULT NULL,

KEY `idProduct` (`idProduct`),

KEY `idStorage` (`idStorage`),

CONSTRAINT `product\_in\_storage\_ibfk\_1` FOREIGN KEY (`idProduct`) REFERENCES `product` (`id`),

CONSTRAINT `product\_in\_storage\_ibfk\_2` FOREIGN KEY (`idStorage`) REFERENCES `storage` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `providers`

--

DROP TABLE IF EXISTS `providers`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `providers` (

`id` int NOT NULL AUTO\_INCREMENT,

`title` varchar(45) DEFAULT NULL,

`PIB` varchar(45) NOT NULL,

`phonenumber` varchar(45) NOT NULL,

`requisites` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `storage`

--

DROP TABLE IF EXISTS `storage`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `storage` (

`id` int NOT NULL AUTO\_INCREMENT,

`title` varchar(45) NOT NULL,

`city` varchar(45) NOT NULL,

`address` varchar(45) NOT NULL,

`capacity` int NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `storno`

--

DROP TABLE IF EXISTS `storno`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `storno` (

`idInvice` int NOT NULL,

`note` text NOT NULL,

KEY `idInvice` (`idInvice`),

CONSTRAINT `storno\_ibfk\_1` FOREIGN KEY (`idInvice`) REFERENCES `invoices` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `type\_invoice`

--

DROP TABLE IF EXISTS `type\_invoice`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `type\_invoice` (

`id` int NOT NULL AUTO\_INCREMENT,

`title` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `user\_storage`

--

DROP TABLE IF EXISTS `user\_storage`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `user\_storage` (

`idUser` int NOT NULL,

`idStorage` int NOT NULL,

KEY `idUser` (`idUser`),

KEY `idStorage` (`idStorage`),

CONSTRAINT `user\_storage\_ibfk\_1` FOREIGN KEY (`idUser`) REFERENCES `users` (`id`),

CONSTRAINT `user\_storage\_ibfk\_2` FOREIGN KEY (`idStorage`) REFERENCES `storage` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `users`

--

DROP TABLE IF EXISTS `users`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `users` (

`id` int NOT NULL AUTO\_INCREMENT,

`login` varchar(16) NOT NULL,

`password` varchar(24) NOT NULL,

`PIB` varchar(45) NOT NULL,

`phonenumber` varchar(13) NOT NULL,

`idPosition` int NOT NULL,

PRIMARY KEY (`id`),

KEY `idPosition` (`idPosition`),

CONSTRAINT `users\_ibfk\_1` FOREIGN KEY (`idPosition`) REFERENCES `position` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

-- Dump completed on 2020-11-20 21:11:52